11. Sampling Methods

Sampling Methods

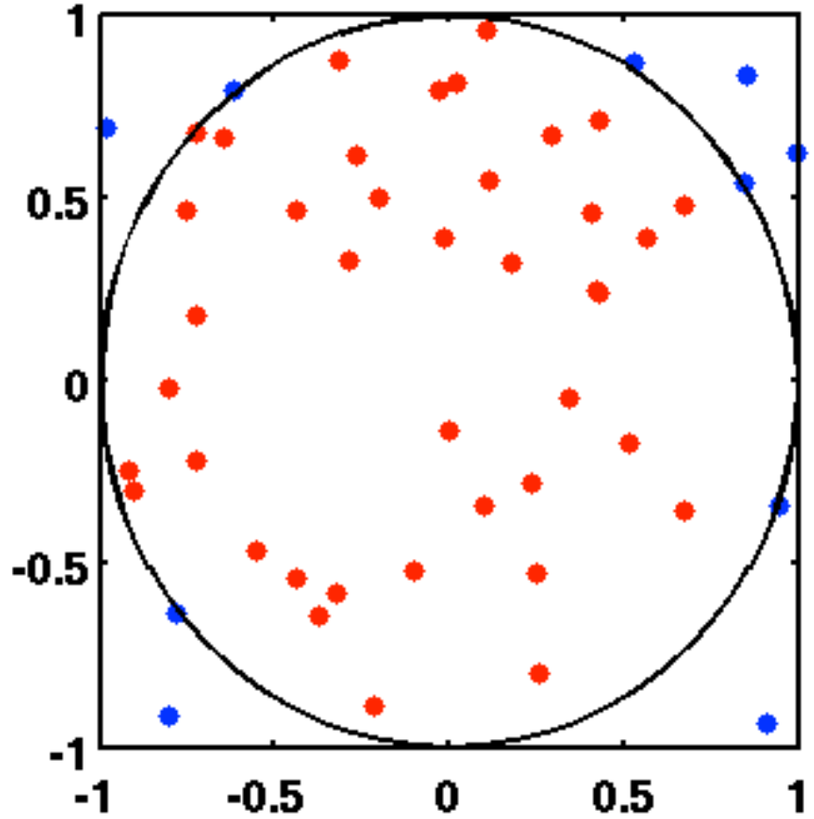
Sampling Methods are widely used in Computer Science

- as an **approximation** of a deterministic algorithm
- to represent **uncertainty** without a parametric model
- to obtain higher computational **efficiency** with a small approximation error

Sampling Methods are also often called **Monte Carlo Methods**

Example: Monte-Carlo Integration

- Sample in the bounding box
- Compute fraction of inliers
- Multiply fraction with box size



Non-Parametric Representation

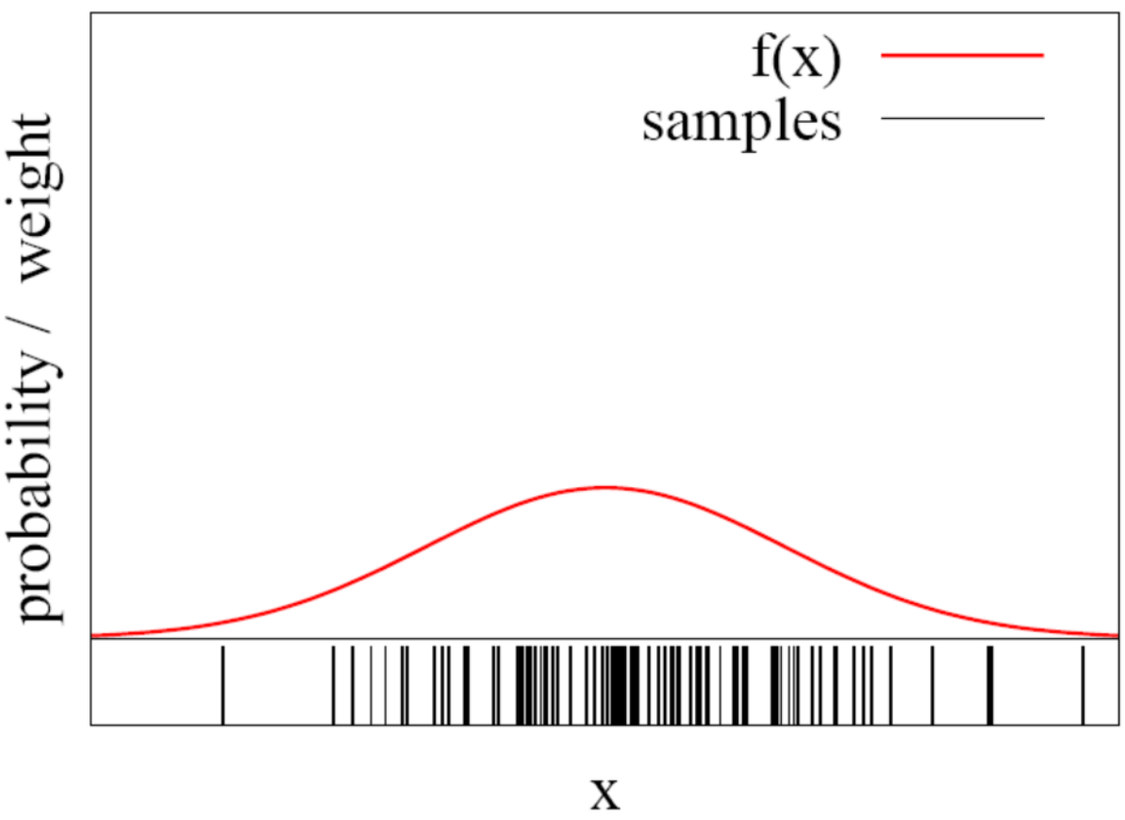
Probability distributions (e.g. a robot‘s belief) can be represeted:

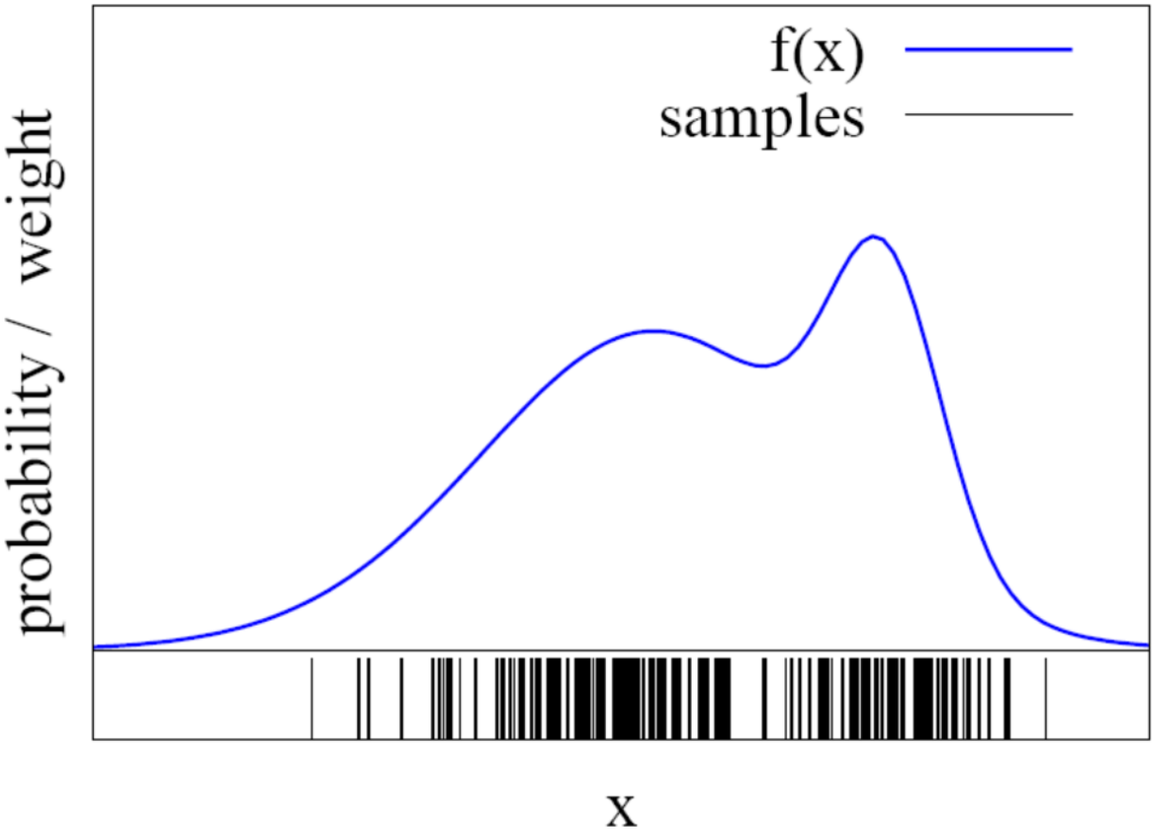
- **Parametrically**: e.g. using mean and covariance of a Gaussian
- **Non-parametrically:** using a set of *hypotheses* (samples) drawn from the distribution

Advantage of non-parametric representation:

- No restriction on the *type* of distribution (e.g. can be multi-modal, non- Gaussian, etc.)

Non-Parametric Representation





The more samples are in an interval, the higher the probability of that interval

But:

How to draw samples from a function/distribution?

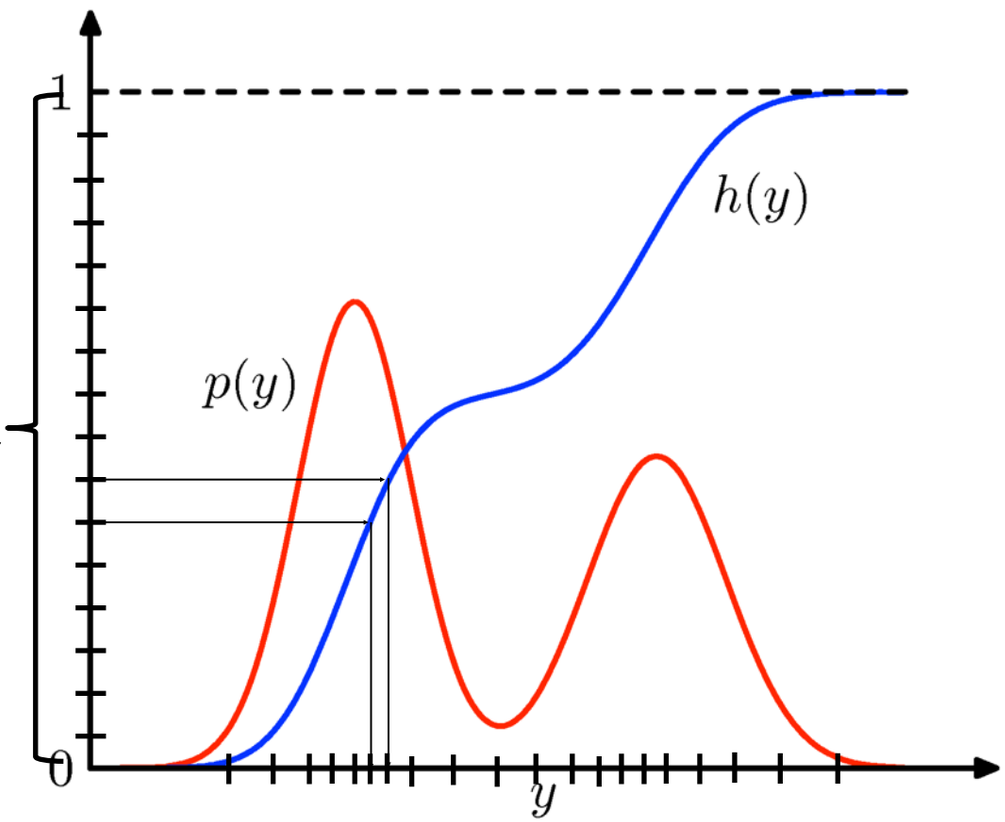
Sampling from a Distribution

There are several approaches:

- Probability transformation
  - Uses inverse of the c.d.f h
- Rejection Sampling
- Importance Sampling
- MCMC

$$h(y) = \int_{-\infty}^{y} p(\hat{y}) d\hat{y}$$

“Cumulative distribution Function”



Probability transformation:

- Sample uniformly in [0,1]
- Transform using h^{-1}

But:

- Requires calculation of h and its inverse

Rejection Sampling

1. Simplification:

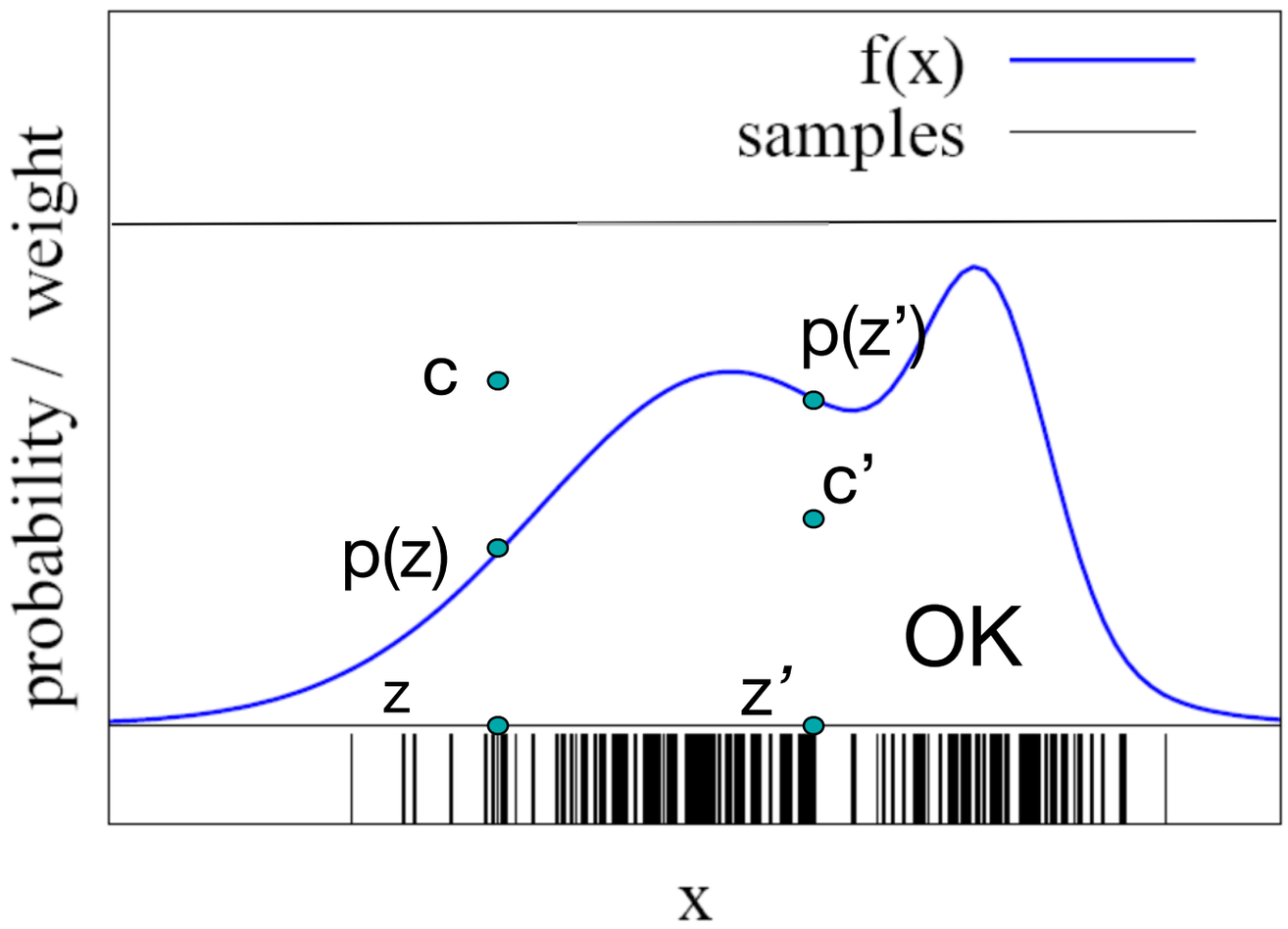
- Assume $p(z) < 1$ for all z
- Sample z uniformly
- Sample c from $[0, 1]$

- **If** $f(z) > c$:

  keep the sample

  otherwise:

  reject the sample

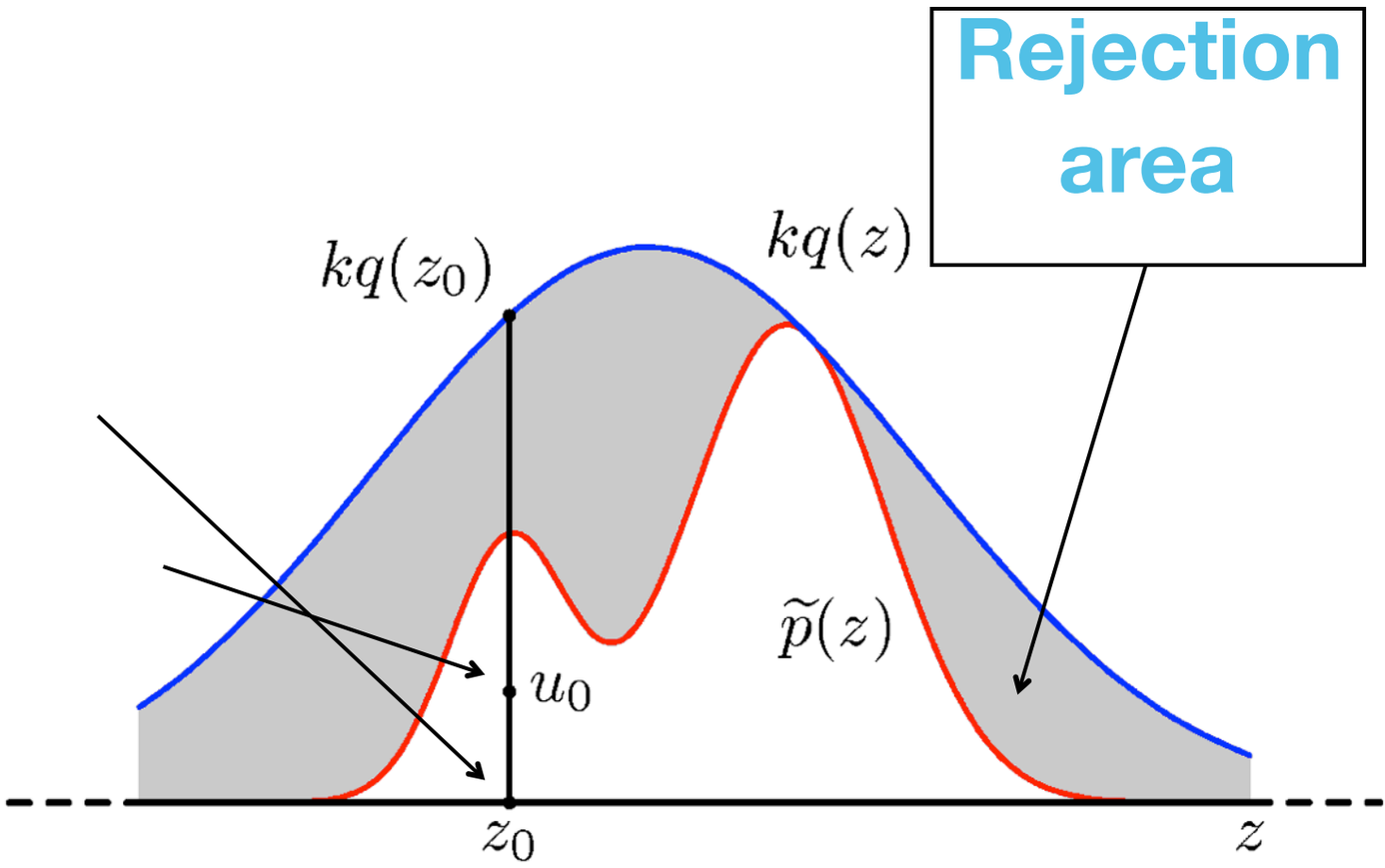


Rejection Sampling

2. General case:

Assume we can evaluate $p(z) = \frac{1}{Z_p}\tilde{p}(z)$ (unnormalized)

- Find **proposal distribution** q
  - Easy to sample from q
- Find k with $kq(z) \geq \tilde{p}(z)$
- Sample from q
- Sample uniformly from [0,kq(z_0)]
- Reject if $u_0 > \tilde{p}(z_0)$



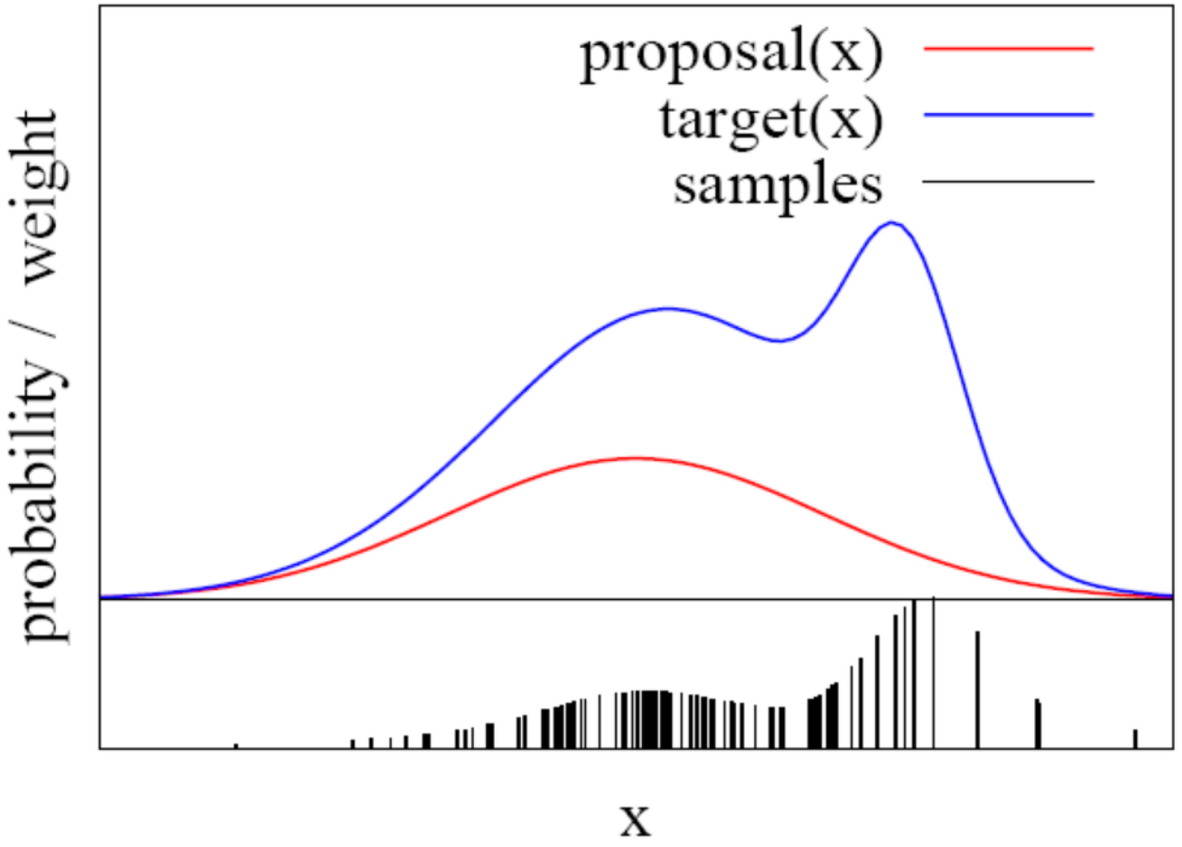
But: Rejection sampling is inefficient.

Importance Sampling

- **Idea:** assign an **importance weight** w to each sample
- With the importance weights, we can account for the "differences between p and q "

$$w(x) = p(x)/q(x)$$

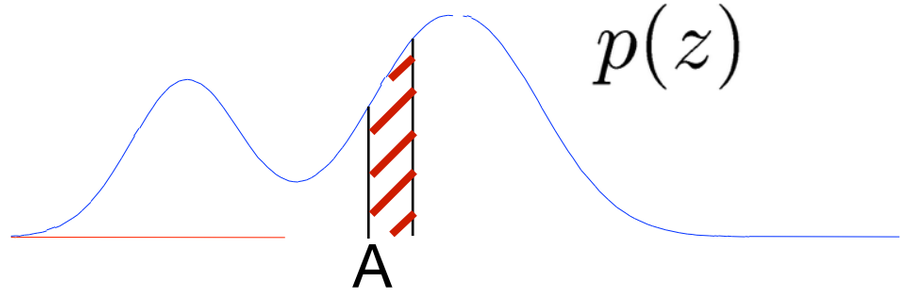
- p is called **target**
- q is called **proposal** (as before)



Importance Sampling

- **Explanation:** The prob. of falling in an interval A is the **area** under p
- This is equal to the expectation of the **indicator function** $I(x \in A)$

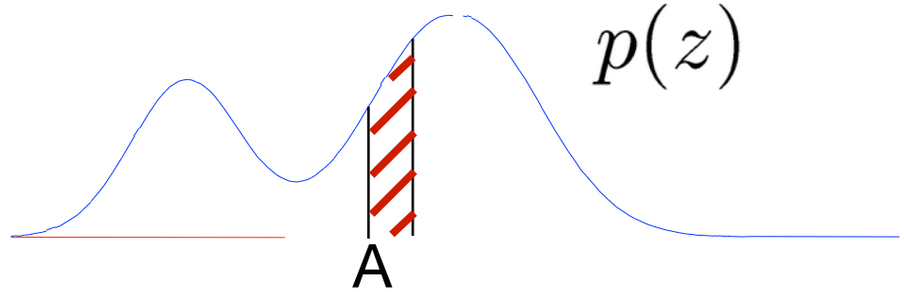
$$E_p[I(z \in A)] = \int p(z) I(z \in A) dz$$



Importance Sampling

- **Explanation:** The prob. of falling in an interval A is the **area** under p
- This is equal to the expectation of the **indicator function** $I(x \in A)$

$$E_p[I(z \in A)] = \int p(z) I(z \in A) dz$$



$$= \int \frac{p(z)}{q(z)} q(z) I(z \in A) dz = E_q[w(z) I(z \in A)]$$

Requirement: $p(x) > 0 \Rightarrow q(x) > 0$

Approximation with samples drawn from q:

$$E_q[w(z) I(z \in A)] \approx \frac{1}{L} \sum_{l=1}^{L} w(z_l) I(z_l \in A)$$

The Particle Filter

- **Non-parametric** implementation of Bayes filter
- Represents the belief (posterior) $\mathrm{Bel}(x_t)$ by a set of **random state samples.**
- This representation is **approximate.**
- Can represent distributions that are **not Gaussian**.
- Can model **non-linear** transformations.

Basic principle:

- Set of state hypotheses (“particles”)
- Survival-of-the-fittest

The Bayes Filter Algorithm (Rep.)

$$\mathrm{Bel}(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \mathrm{Bel}(x_{t-1}) dx_{t-1}$$

Algorithm Bayes_filter $(\mathrm{Bel}(x), d)$

1. if d is a sensor measurement z then
2. $\quad \eta = 0$
3. $\quad$ for all x do
4. $\quad\quad \mathrm{Bel}'(x) \leftarrow p(z \mid x)\mathrm{Bel}(x)$
5. $\quad\quad \eta \leftarrow \eta + \mathrm{Bel}'(x)$
6. $\quad$ for all x do $\mathrm{Bel}'(x) \leftarrow \eta^{-1}\mathrm{Bel}'(x)$
7. else if d is an action u then
8. $\quad$ for all x do $\mathrm{Bel}'(x) \leftarrow \int p(x \mid u, x')\mathrm{Bel}(x')dx'$
9. return $\mathrm{Bel}'(x)$

Mathematical Description

Set of weighted samples:

$$\mathcal{X}_t := \{\langle x_t^{[1]}, w_t^{[1]} \rangle, \langle x_t^{[2]}, w_t^{[2]} \rangle, \ldots, \langle x_t^{[M]}, w_t^{[M]} \rangle\}$$

State hypotheses

Importance weights

The samples represent the probability distribution:

$$p(x) = \sum_{i=1}^{M} w_t^{[i]} \cdot \delta_{x_t^{[i]}}(x)$$

Point mass distribution (“Dirac”)

The Particle Filter Algorithm

Algorithm *Particle_filter*$(\mathcal{X}_{t-1}, u_t, z_t)$:

1. $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2. **for** $m = 1$ **to** M **do**
3. sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
4. $w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$
5. $\bar{\mathcal{X}}_t \leftarrow \bar{\mathcal{X}}_t \cup \langle x_t^{[m]}, w_t^{[m]} \rangle$
6. **for** $m = 1$ **to** M **do**
   draw i with prob. $\propto w_t^{[i]}$
   $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup \langle x_t^{[i]}, 1/M \rangle$
7. **return** $\mathcal{X}_t$

Sample from proposal

Compute sample weights

Resampling

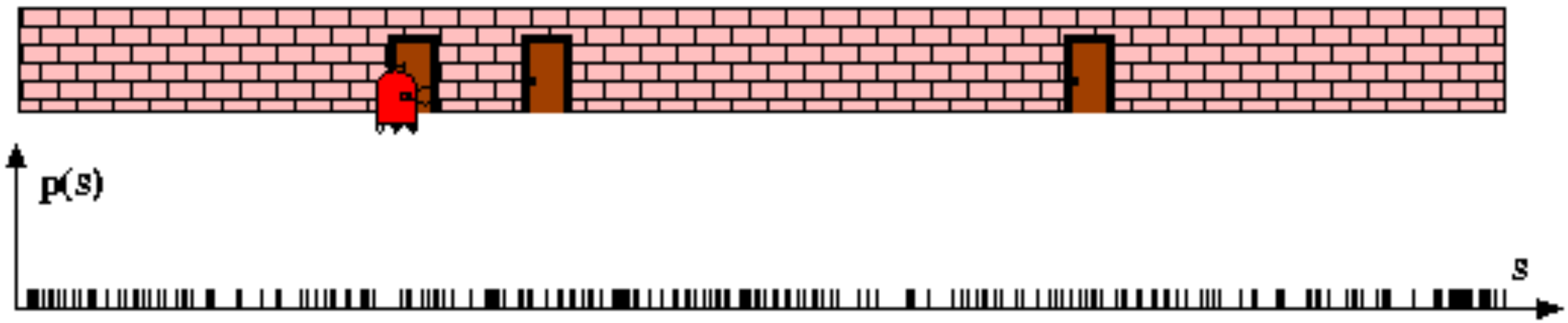
Localization with Particle Filters

- Each particle is a potential **pose** of the robot
- Proposal distribution is the motion model of the robot (**prediction step**)
- The observation model is used to compute the importance weight (**correction step**)

Randomized algorithms are usually called Monte Carlo algorithms, therefore we call this:

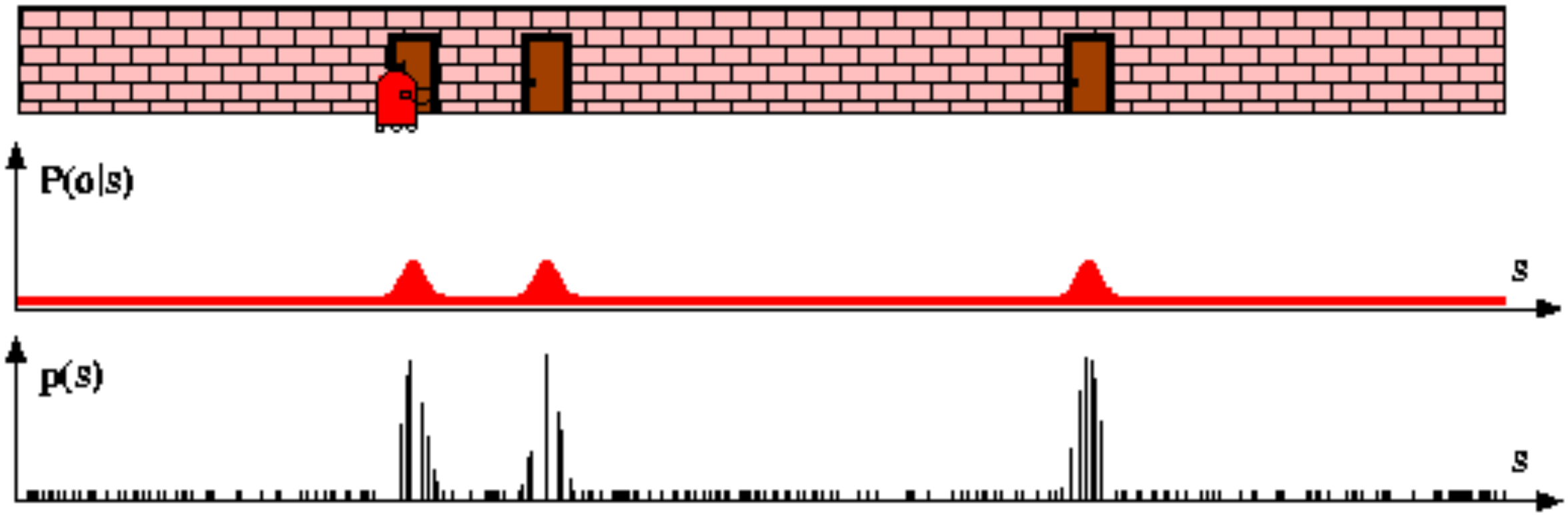
Monte-Carlo Localization

A Simple Example



- The initial belief is a uniform distribution (global localization).
- This is represented by an (approximately) uniform sampling of initial particles.

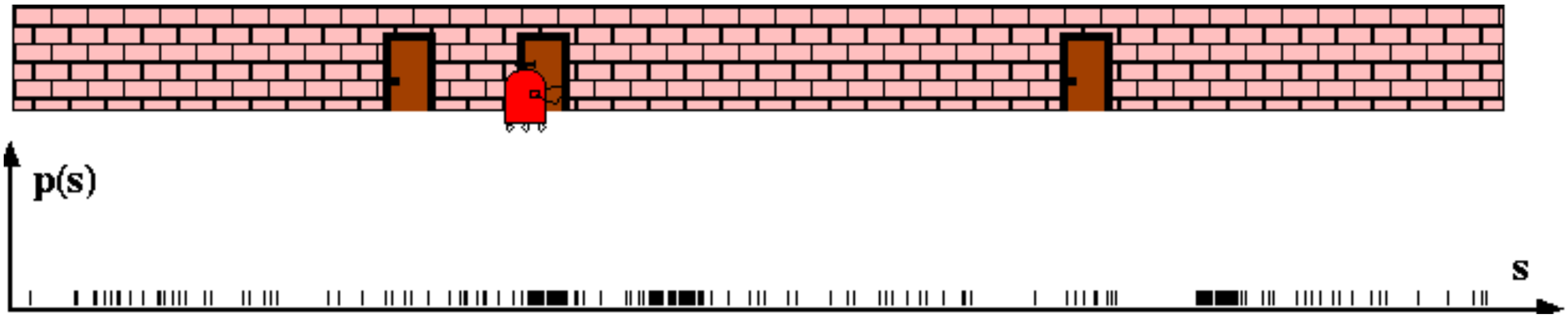
Sensor Information



The sensor model $p(z_t \mid x_t^{[m]})$ is used to compute the new importance weights:

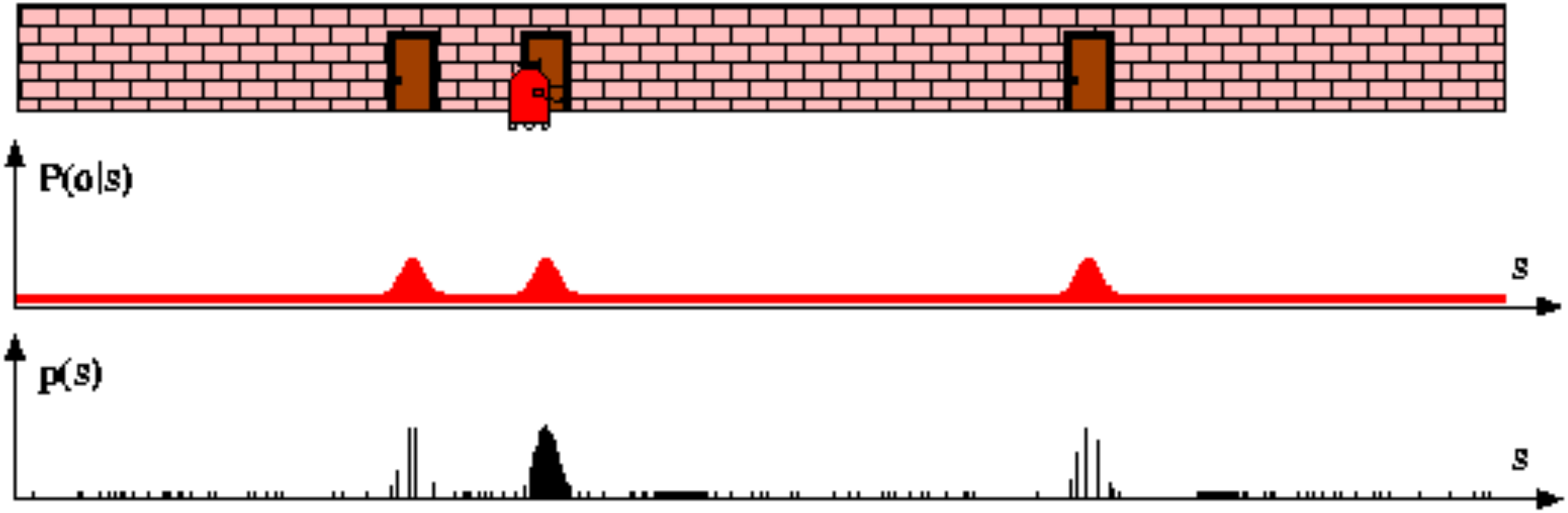
$$w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$$

Robot Motion



After resampling and applying the motion model $p(x_t \mid u_t, x_{t-1}^{[m]})$ the particles are distributed more densely at three locations.

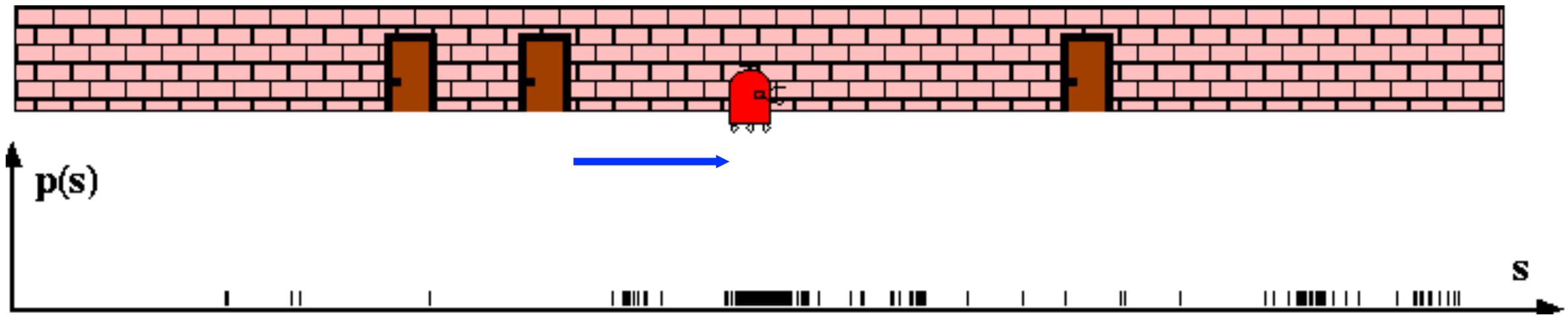
Sensor Information



Again, we set the new importance weights equal to the sensor model.

$$w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$$

Robot Motion



Resampling and application of the motion model:

One location of dense particles is left.

The robot is localized.

A Closer Look at the Algorithm…

Algorithm *Particle_filter* $(\mathcal{X}_t, u_t, z_t)$:

1. $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2. **for** $m = 1$ **to** M **do**
3. sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
4. $w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$
5. $\bar{\mathcal{X}}_t \leftarrow \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6. **for** $m = 1$ **to** M **do**
   draw i with prob. $\propto w_t^{[i]}$
   $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup x_t^{[i]}$
7. **return** $\mathcal{X}_t$

Sample from proposal

Compute sample weights

Resampling

Sampling from Proposal

This can be done in the following ways:

- Adding the motion vector to each particle directly (this assumes perfect motion)

$$\text{sample } x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$$

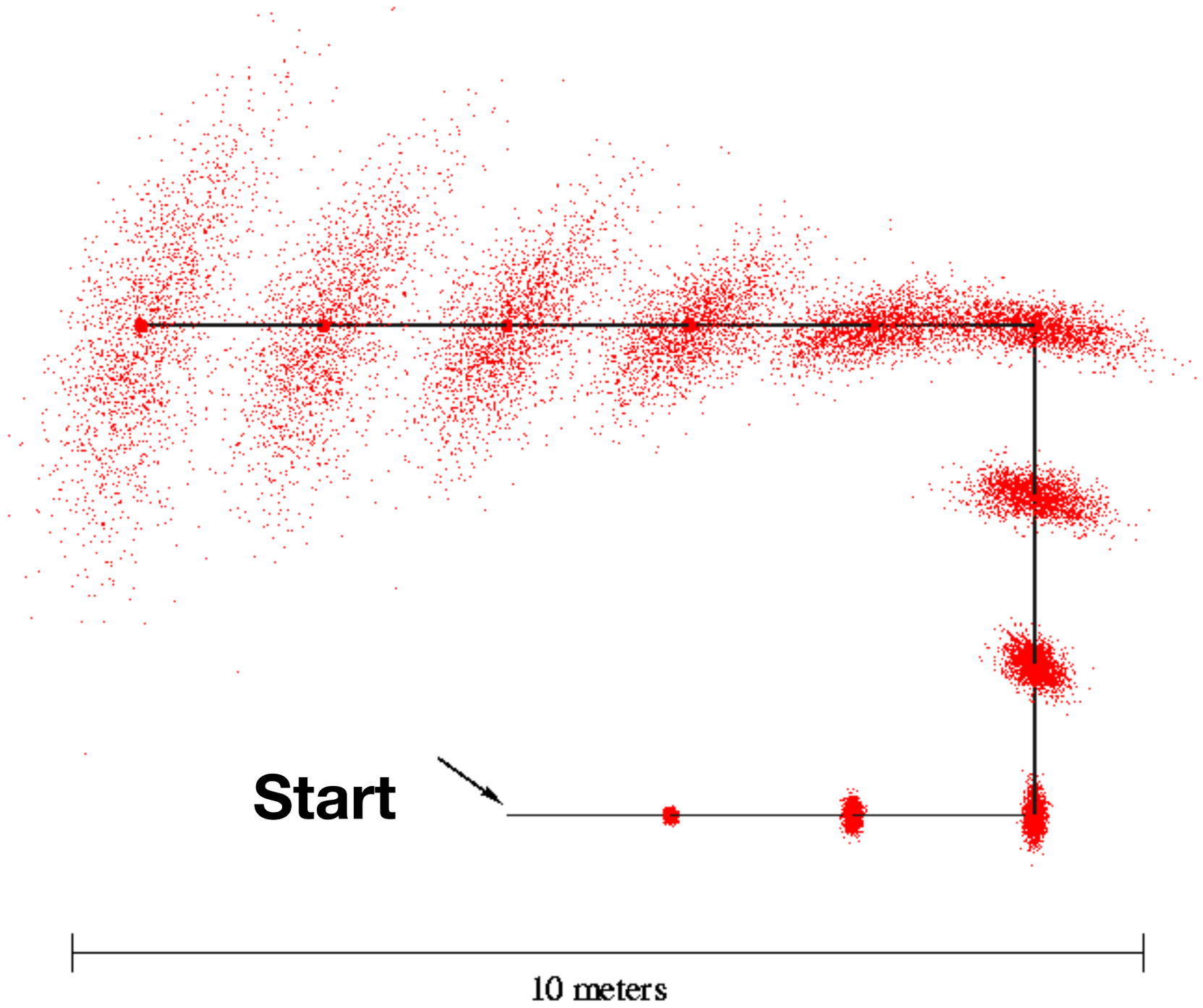
- Sampling from the motion model ,
  e.g. for a 2D motion with translation velocity v and rotation velocity w we have: $p(x_t \mid u_t, x_{t-1}^{[m]})$

$$\mathbf{u}_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix} \qquad \mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}$$

Position

Orientation

Motion Model Sampling (Example)



Computation of Importance Weights

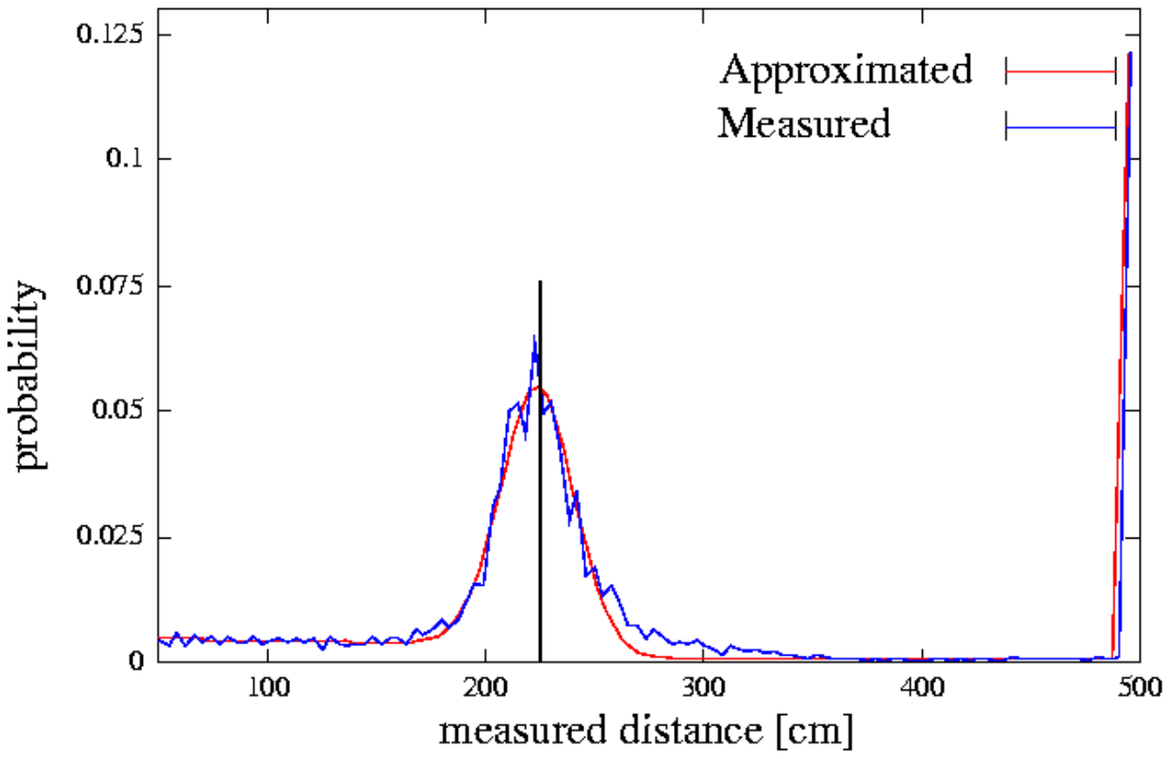
Computation of the sample weights:

- Proposal distribution: $g(x_t^{[m]}) = p(x_t^{[m]} \mid u_t, x_{t-1}^{[m]}) \mathrm{Bel}(x_{t-1}^{[m]})$
  (we sample from that using the motion model)
- Target distribution (new belief): $f(x_t^{[m]}) = \mathrm{Bel}(x_t^{[m]})$
  (we can not directly sample from that → importance sampling)
  $w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$
- Computation of importance weights:

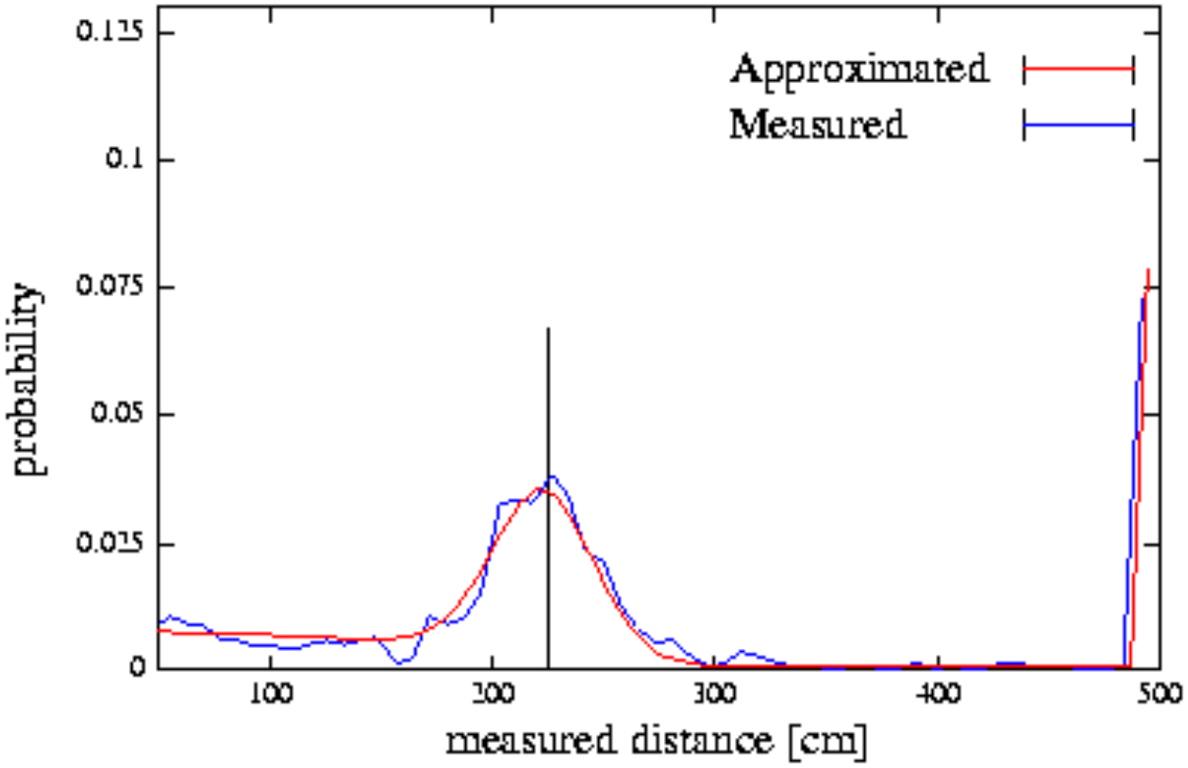
$$w_t^{[m]} = \frac{f(x_t^{[m]})}{g(x_t^{[m]})} \propto \frac{p(z_t \mid x_t^{[m]}) p(x_t^{[m]} \mid u_t, x_{t-1}^{[m]}) \mathrm{Bel}(x_{t-1}^{[m]})}{p(x_t^{[m]} \mid u_t, x_{t-1}^{[m]}) \mathrm{Bel}(x_{t-1}^{[m]})} = p(z_t \mid x_t^{[m]})$$

Proximity Sensor Models

- How can we obtain the sensor model $p(z_t \mid x_t^{[m]})$?
- Sensor Calibration:



Laser sensor



Sonar sensor

Resampling

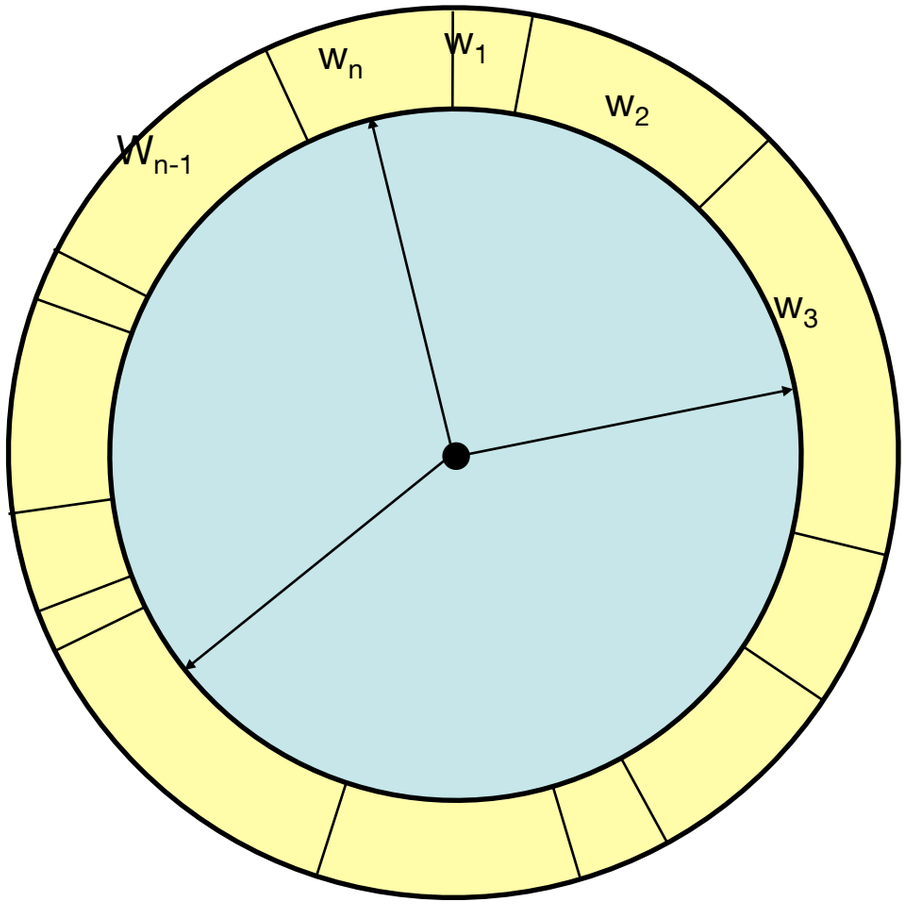
- Given: Set $\bar{\mathcal{X}}_t$ of weighted samples.
- Wanted : Random sample, where the probability of drawing x_i is equal to w_i.
- Typically done M times with replacement to generate new sample set $\mathcal{X}_t$

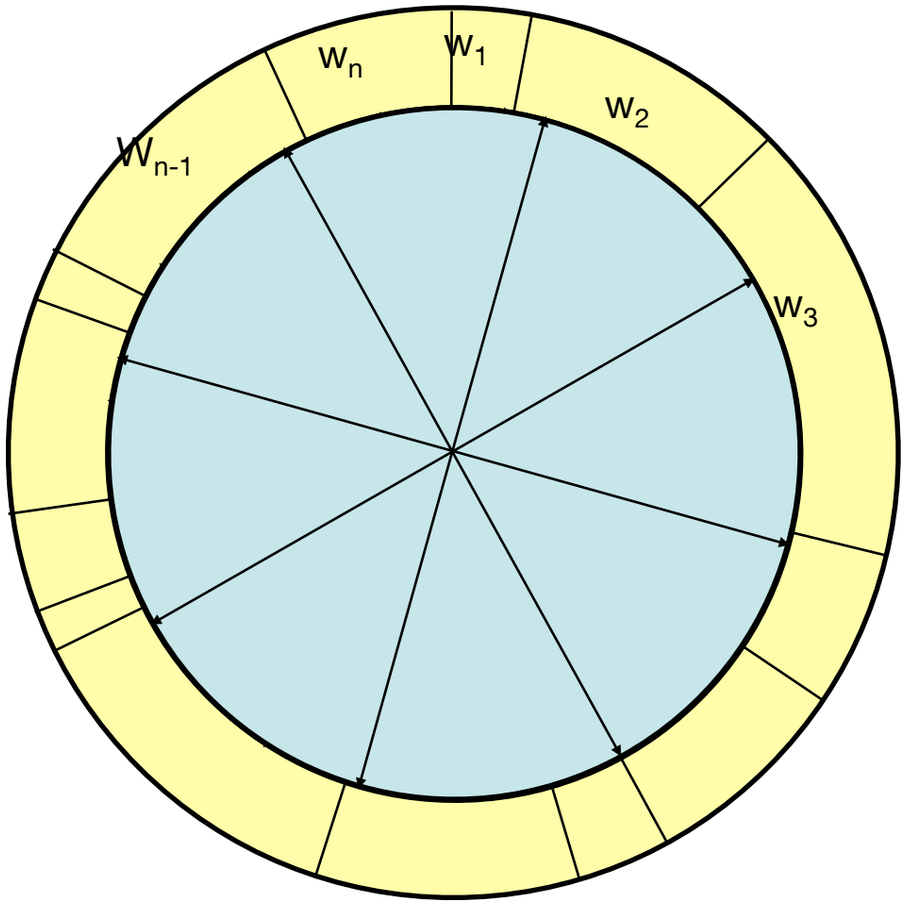
for $m = 1$ **to** M **do**

draw i with prob. $\propto w_t^{[i]}$

$\mathcal{X}_t \leftarrow \mathcal{X}_t \cup x_t^{[i]}$

Resampling

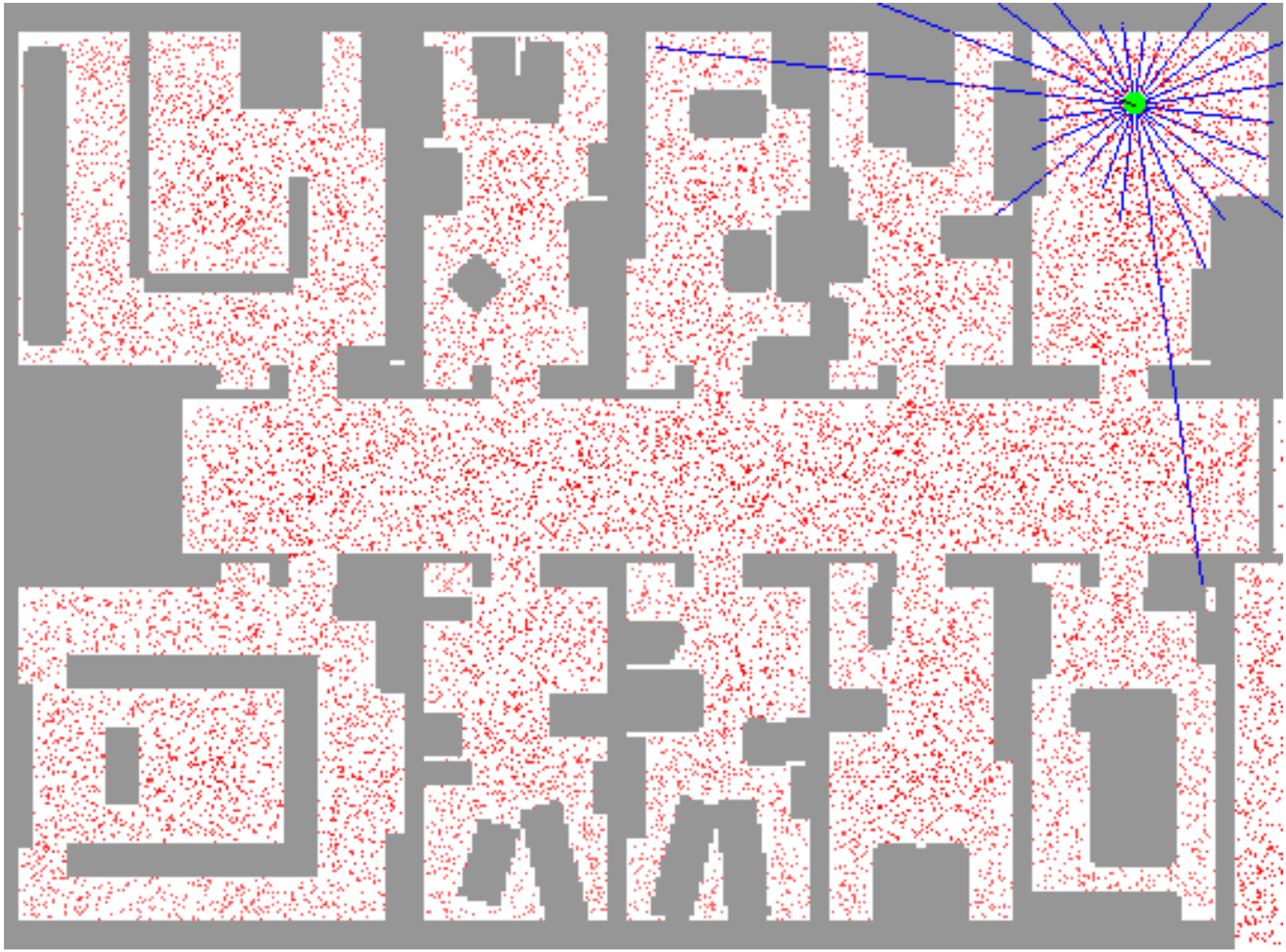




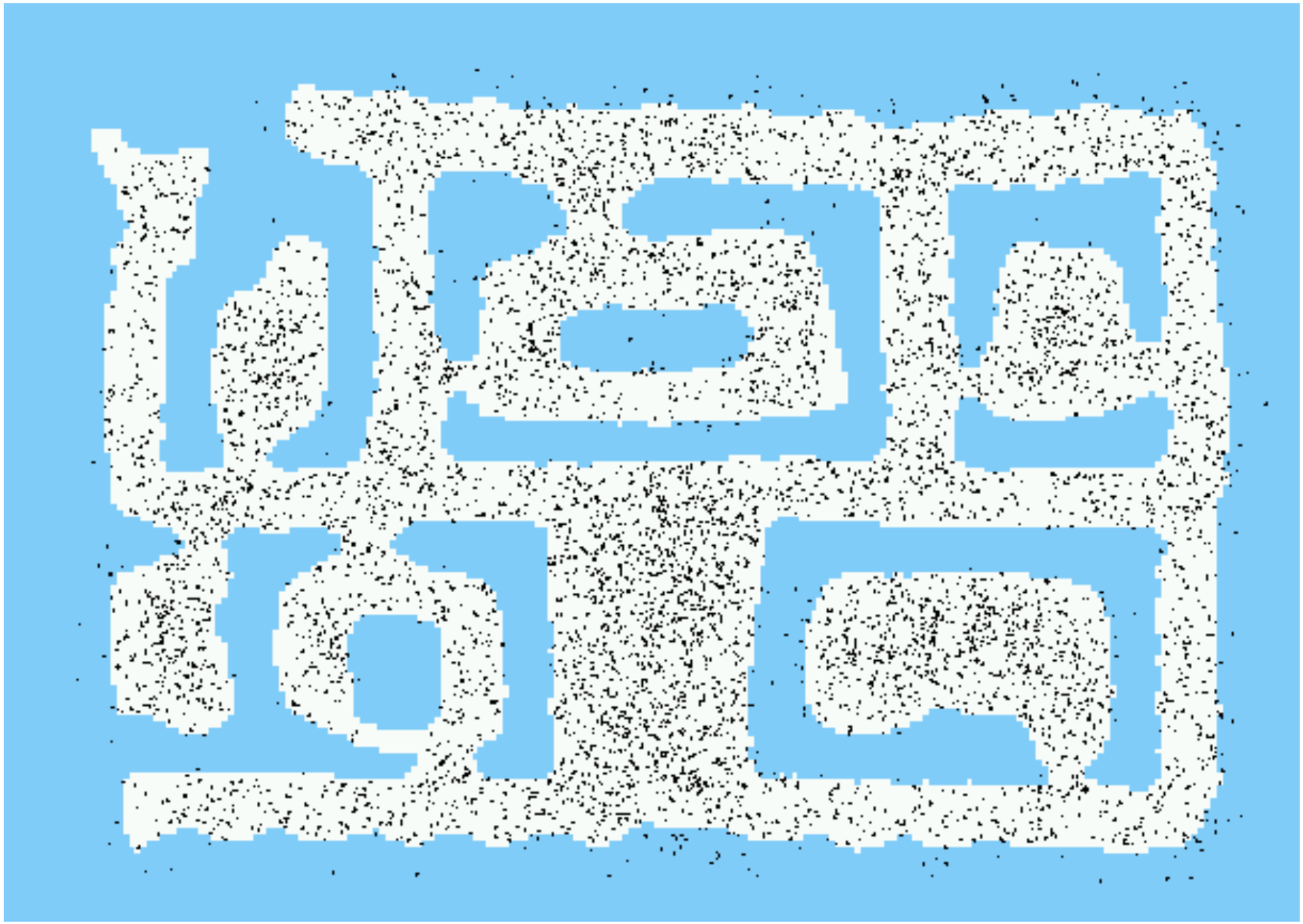
- Standard n-times sampling results in high variance
- This requires more particles
- O(nlog n) complexity

- Instead: low variance sampling only samples once
- Linear time complexity
- Easy to implement

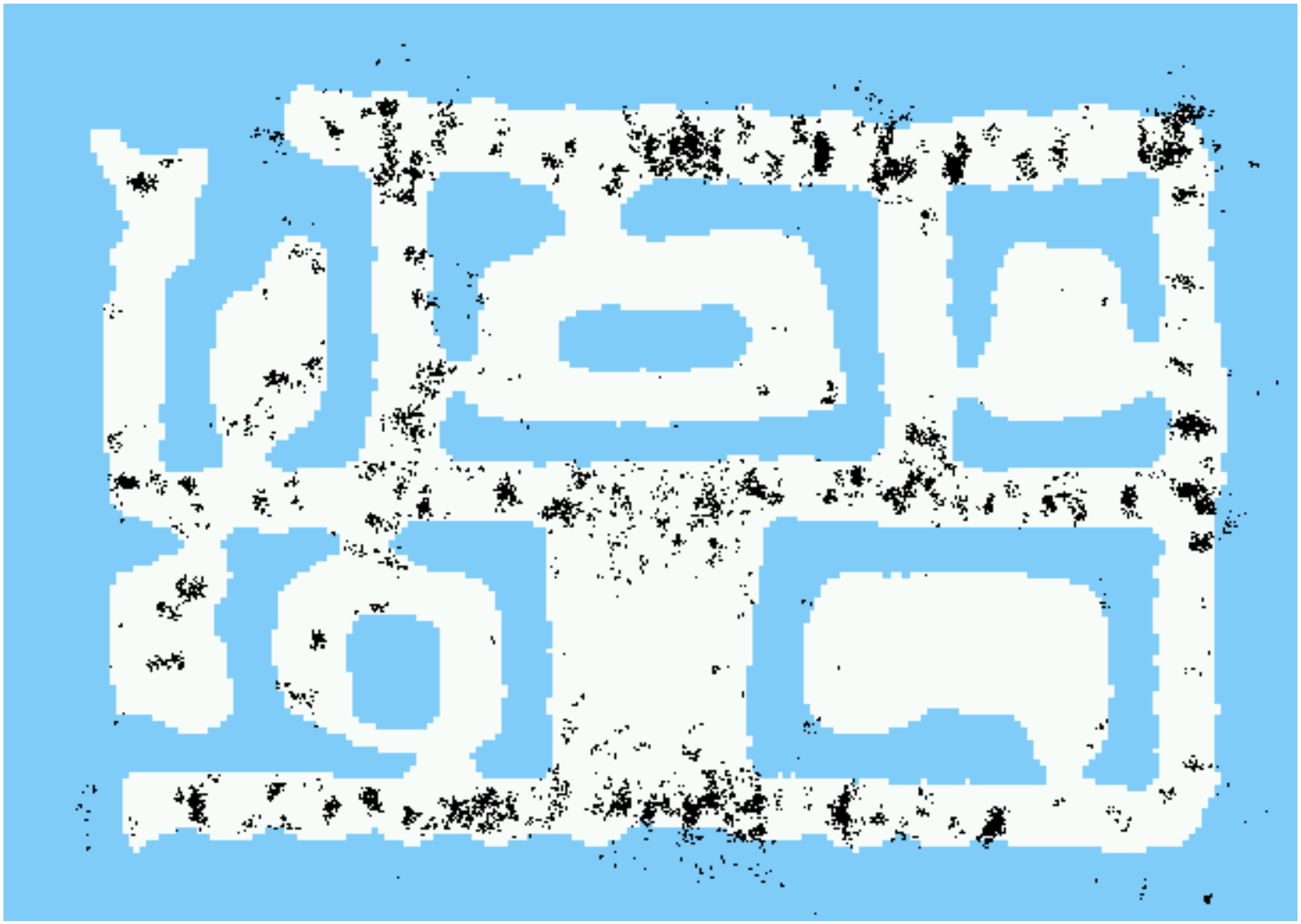
Sample-based Localization (sonar)



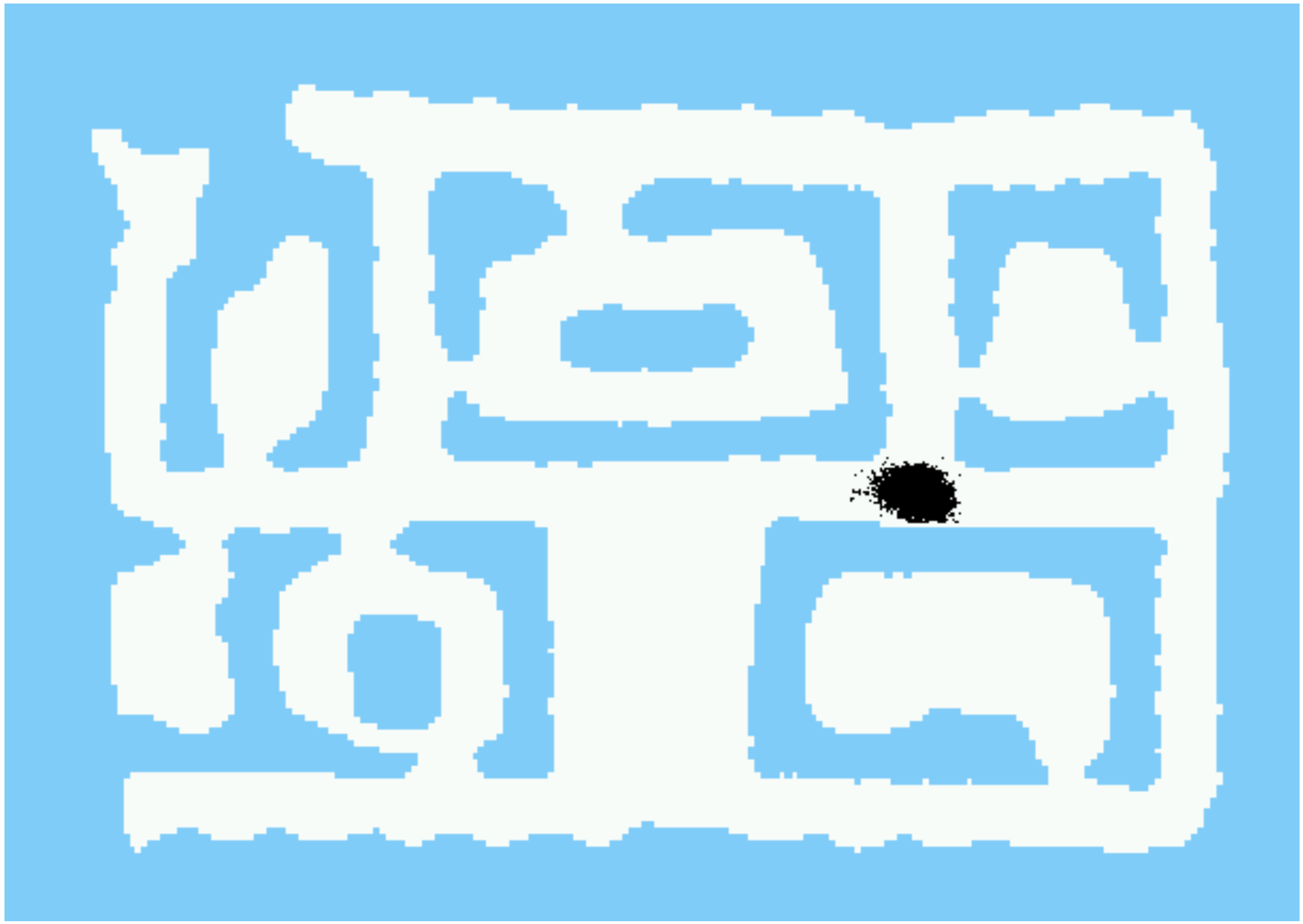
Initial Distribution



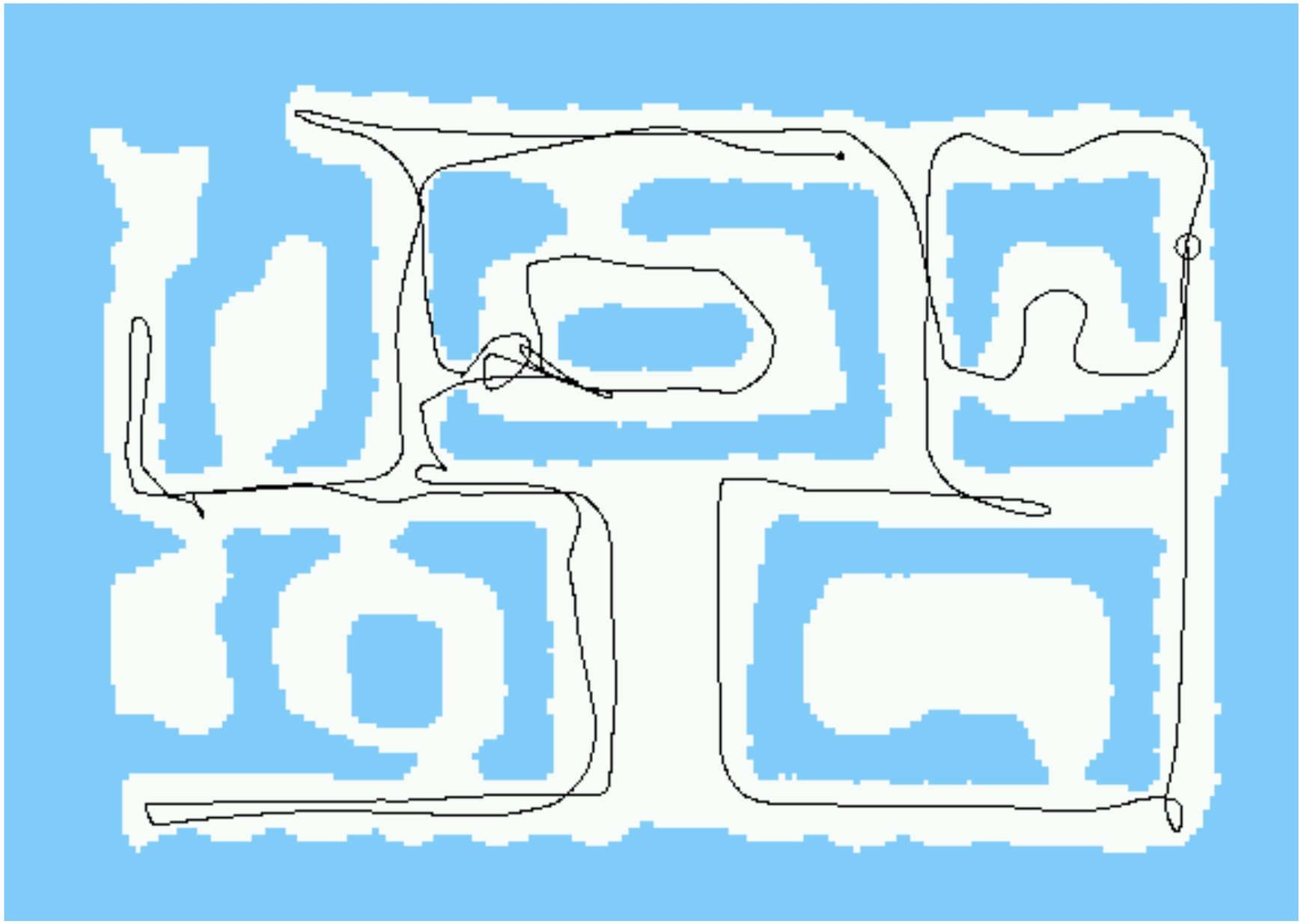
After Ten Ultrasound Scans



After 65 Ultrasound Scans



Estimated Path



Kidnapped Robot Problem

The approach described so far is able to

- track the pose of a mobile robot and to
- globally localize the robot.

- How can we deal with localization errors (i.e., the kidnapped robot problem)?

Idea: Introduce uniform samples at every resampling step

- This adds new hypotheses

Summary

- There are mainly 4 different types of sampling methods: Transformation method, rejections sampling, importance sampling and MCMC
- Transformation only rarely applicable
- Rejection sampling is often very inefficient
- Importance sampling is used in the particle filter which can be used for robot localization
- An efficient implementation of the resampling step is the low variance sampling