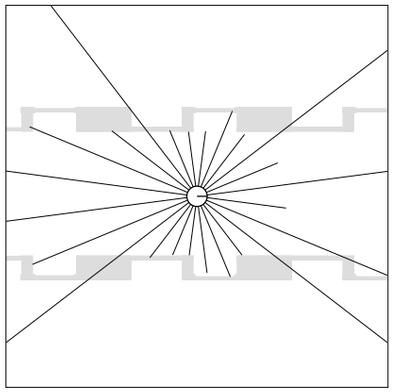
CS4278/CS5478 Intelligent Robots: Algorithms and Systems

Lin Shao

NUS

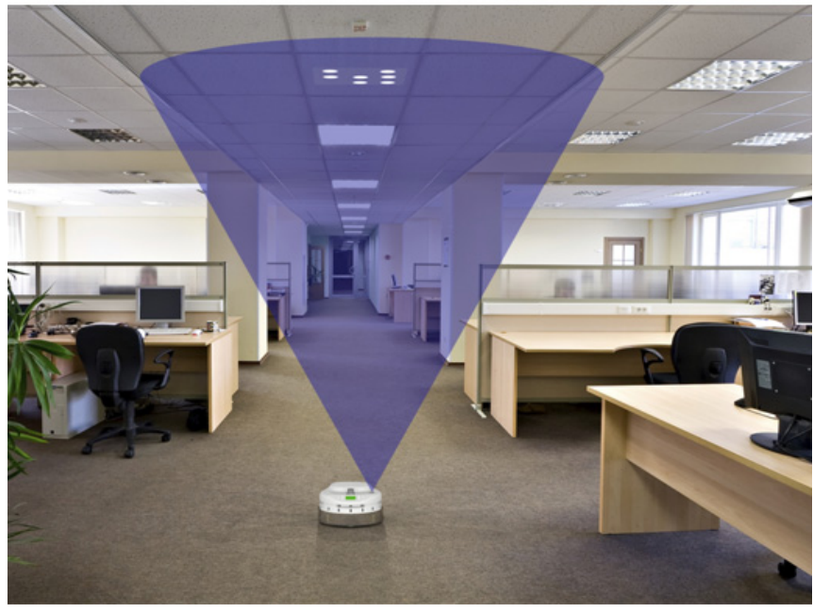
Localization. This question of "**where**" seems surprisingly challenging. Where is the robot? We want to determine the robot pose from the sensor data.

Input

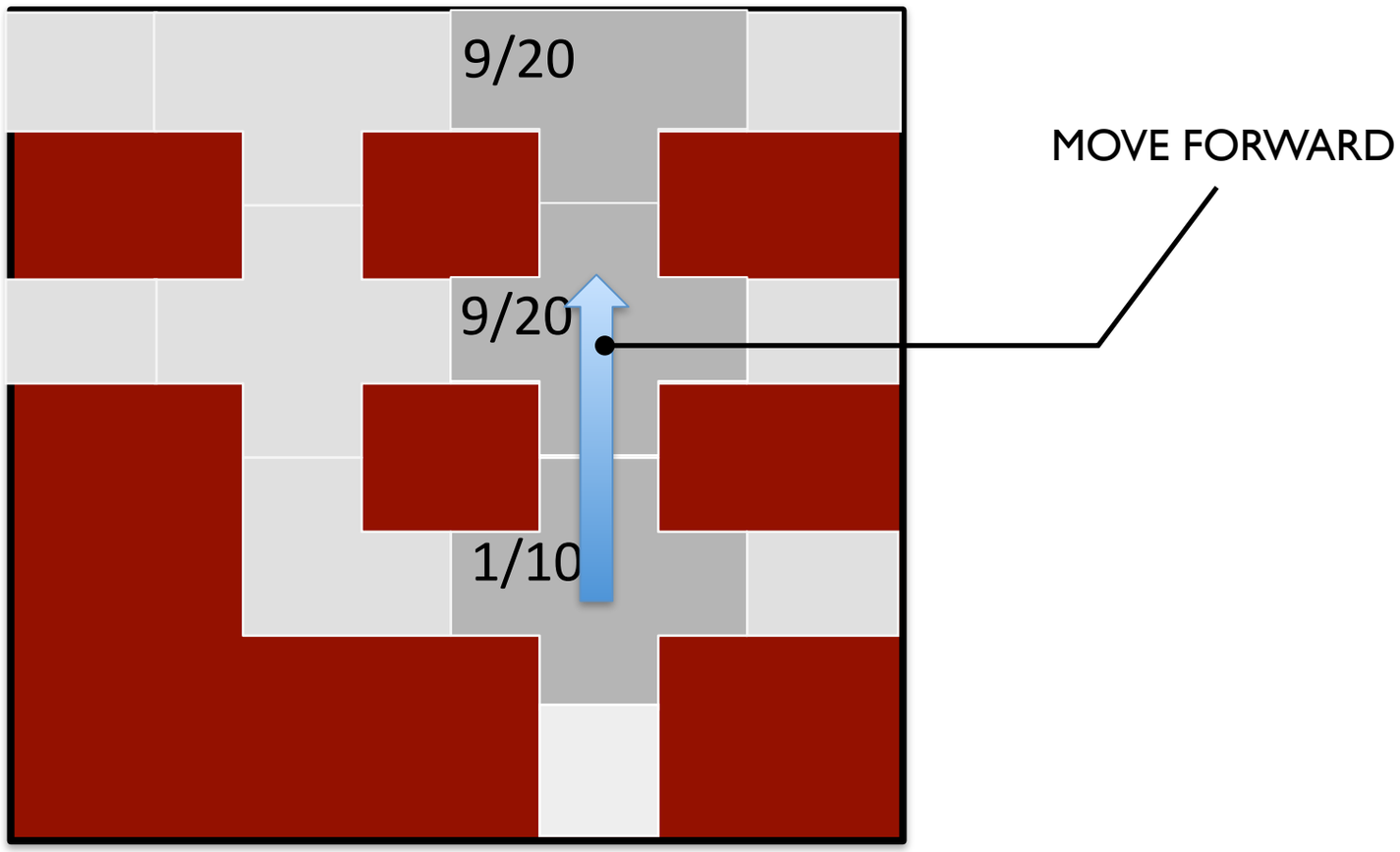


Output

$$(x, y, \theta)$$



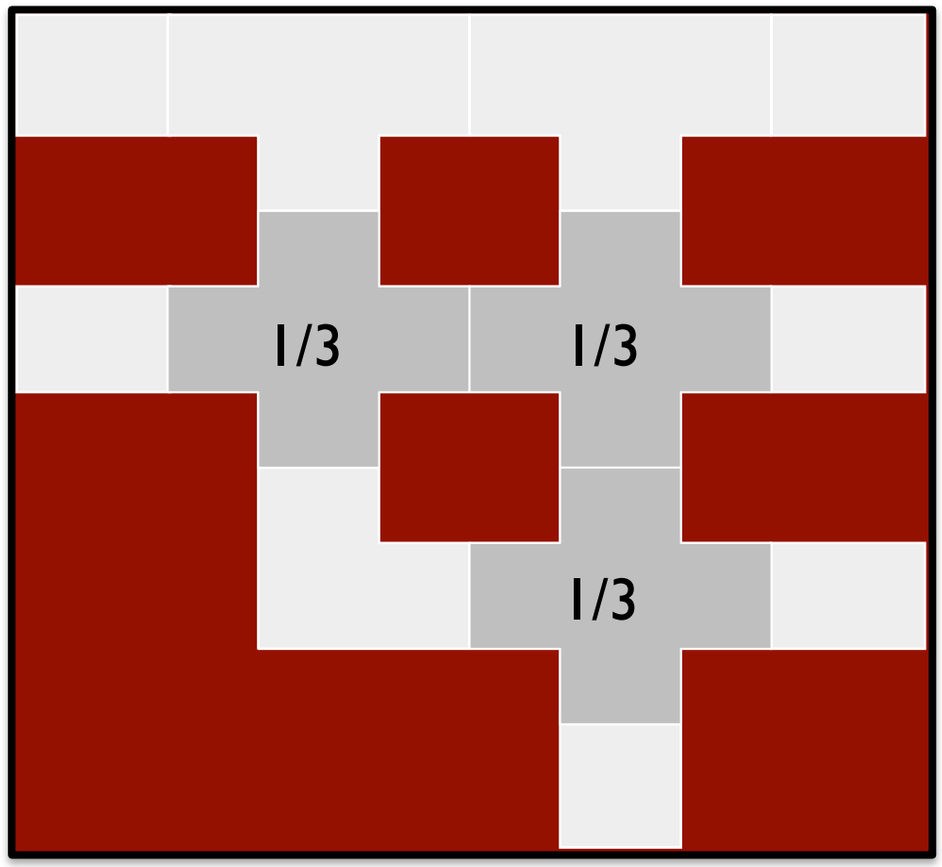
Example. The robot moves in a 4x4 grid world. The red regions indicate obstacles. The forward move may overshoot or undershoot probabilistically. Each cell has a shaped boundary that provides the observations enabling the robot to localize.



Action

Belief [0 0 0 0 0 0 0 0 0 1 0 0]

Belief [0 0 9/20 0 0 0 9/20 0 0 1/10 0 0]



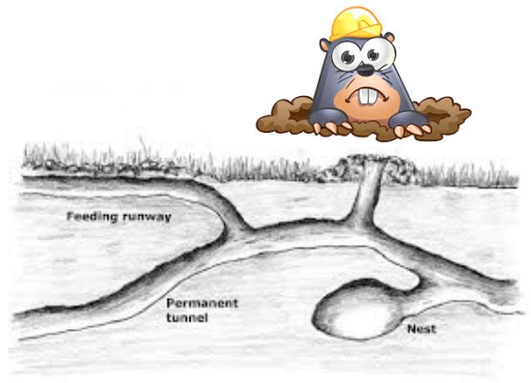
Observation:

Belief [1/12 1/12 1/12 ... 1/12]

Belief [0 0 0 0 0 1/3 1/3 0 0 1/3 0 0]

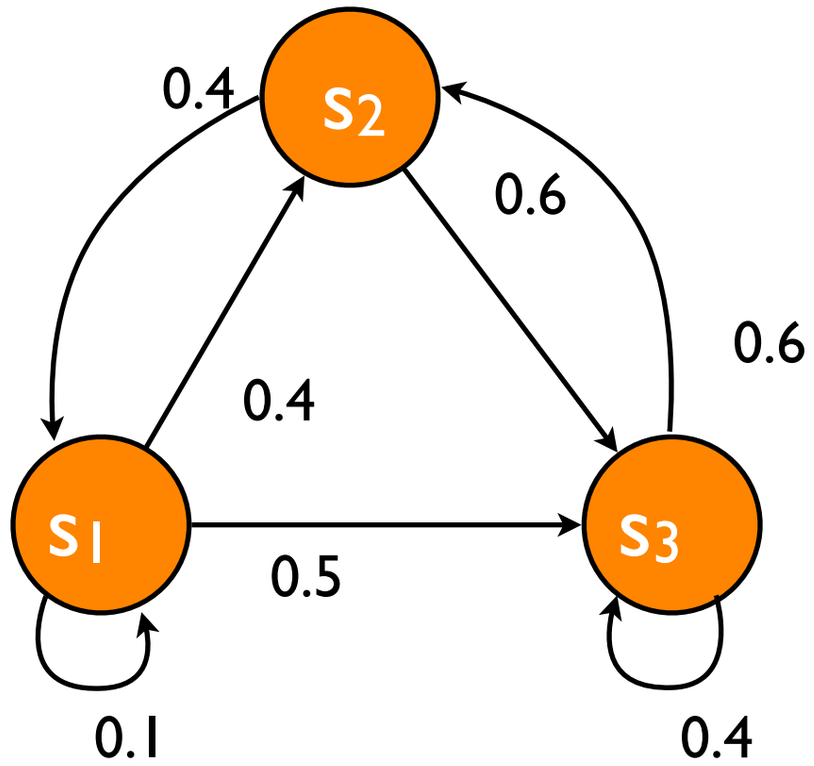
Question. If we get another observation , how does the belief change?

Example. A mole has burrowed a network of underground tunnels, with 3 openings at ground level. We are interested in modeling the sequence of mole locations, i.e., openings at which the mole will poke its head out of the ground. Assume that the mole's next location depends only on its current location.



- States s_1, s_2, s_3
- Transition probabilities T

	S_1	S_2	S_3
S_1	0.1	0.4	0.5
S_2	0.4	0	0.6
S_3	0	0.6	0.4



Question. Let $\mathbf{p}^t = (p_1^t, p_2^t, p_3^t)$ be the probability distribution for the mole's location at time t. Suppose that $\mathrm{p}^0 = (1,0,0)$. What is $\mathbf{p}^1$?

$p^1 = (0.1,0.4,0.5)$

In general,

$$p(S_t = s') = \sum_{s \in S} p(S_t = s' | S_{t-1} = s) p(S_{t-1} = s)$$

motion model

or in the matrix form,

$$\mathbf{p}^t = \mathbf{p}^{t-1} T$$

Let us apply it for a few time steps:

- $\mathbf{p}^0 = [1,0,0]$
- $\mathbf{p}^1 = \mathbf{p}^0 T = [0.1,0.4,0.5]$
- $\mathbf{p}^2 = \mathbf{p}^1 T = [0.17,0.34,0.49]$
- $\mathbf{p}^3 = \mathbf{p}^2 T = \ldots$
- $\ldots$

Remember the motion model for MDP,
T(s,a,s') = p(s' l s, a)? It's similar here, without explicitly conditioning on the action.

p	1	0	0

×

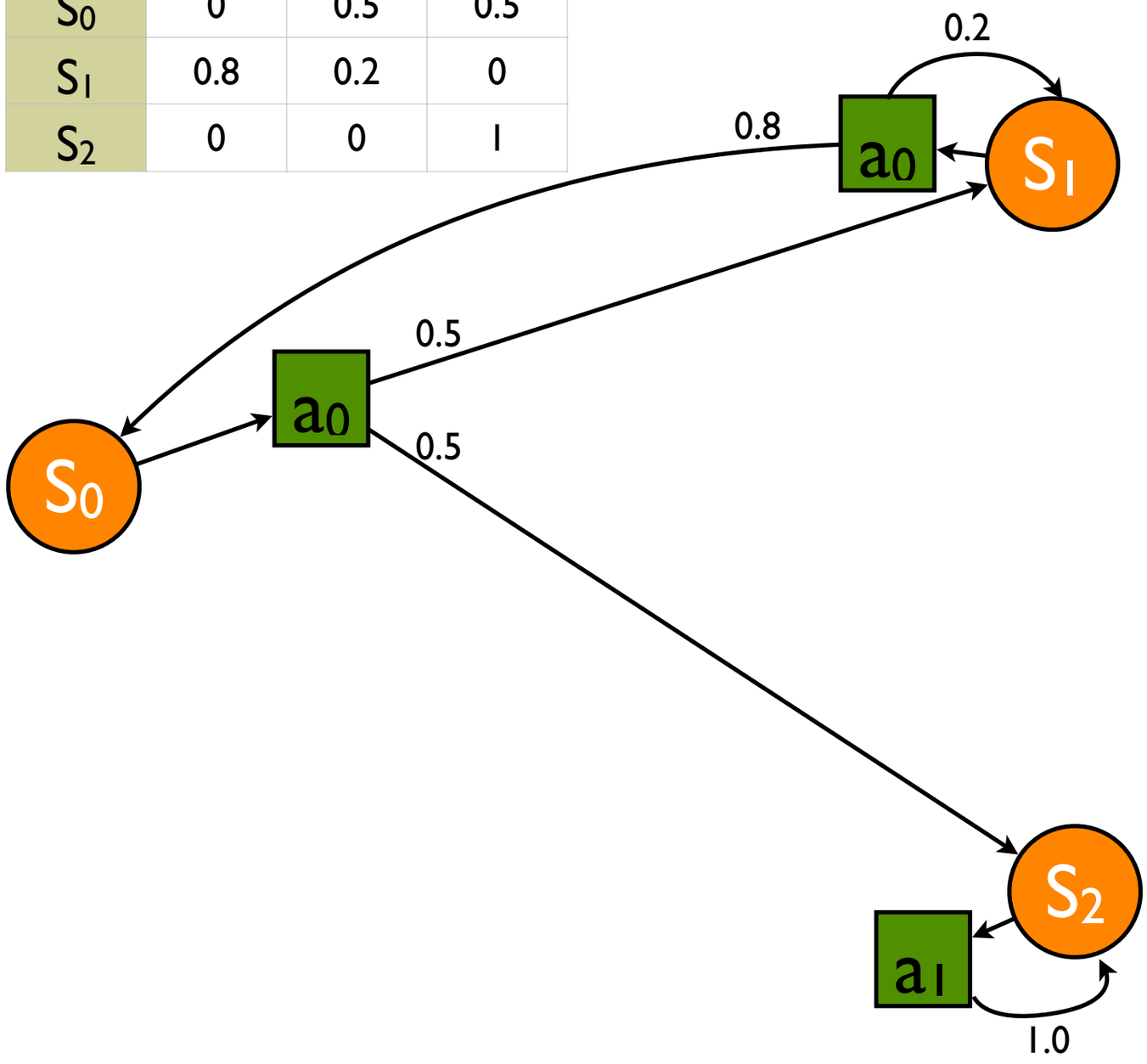
	S_1	S_2	S_3
S_1	0.1	0.4	0.5
S_2	0.4	0	0.6
S_3	0	0.6	0.4

Prediction. Using only the motion model, we know, more precisely, we can predict where the mole is at time t, with some uncertainty.

However, the bad news is that the uncertainty seems to increase over time. The information loss is particularly obvious if we compare p^0 and p^1. Further, the information loss evolves in a complex manner that depends on the motion model. It does not have to increase monotonically.

Question. What is $\mathbf{p}^t$ as $t \to \infty$? What do you learn about the evolution of information loss over time?

	S_0	S_1	S_2
S_0	0	0.5	0.5
S_1	0.8	0.2	0
S_2	0	0	1



Assume that every time the mole surfaces, we may be able to see it with some error. We have three observations, z_1, z_2, and, z_3, each corresponding to sighting the mole at one of the three opening, respectively. It's dark out there. Our observations are noisy with the following probabilities M:

	Z_1	Z_2	Z_3
S_1	0.6	0.2	0.2
S_2	0.2	0.6	0.2
S_3	0.2	0.2	0.6

Observation. Suppose that an at time t, we receive an observation z. What does it tell us about where the mole is?

That depends ons $p(z \mid s)$, the relationship between the observation z and the underlying state s. For an ideal sensor, there is one-to-one correspondence between z and s. In general, we use the observation model $p(z \mid s)$ and apply the Bayes' rule:

$$p(S_t = s|Z_t = z) = \frac{p(Z_t = z|S_t = s)p(S_t = s)}{p(Z_t = z)}$$

$$= \eta p(Z_t = z|S_t = s)p(S_t = s)$$

posterior

observation model

prior

Suppose that our current belief on the mole's location is $\mathbf{p} = [0.1, 0.4, 0.5]$. Given an observation z_2, how do we update our belief?

p
0.1
0.4
0.5

$\otimes$

	Z_1	Z_2	Z_3
S_1	0.6	0.2	0.2
S_2	0.2	0.6	0.2
S_3	0.2	0.2	0.6

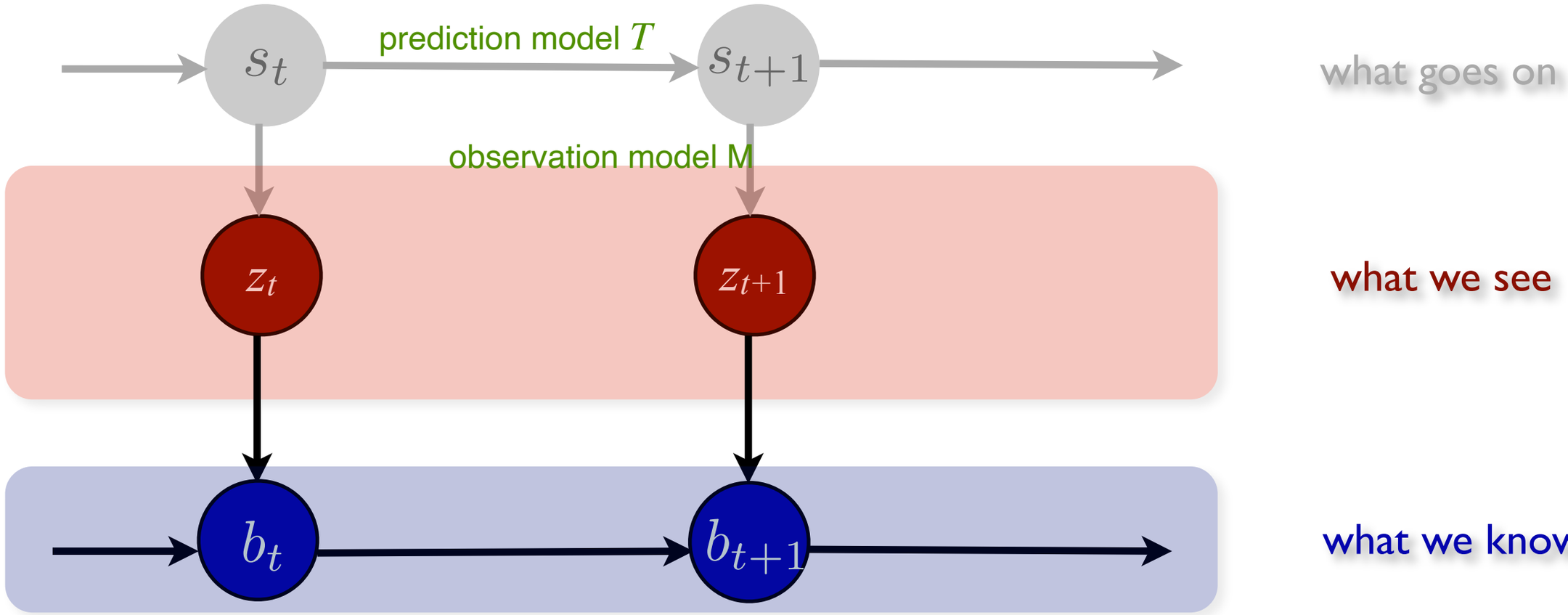
According to the Bayes' rule, we perform element-wise multiplication and then normalize: $\mathbf{p} = \eta\,[0.02, 0.24, 0.10]$.

Using the motion model and the observation model, we can estimate the robot's **state** from noisy **observations**.

Hidden Markov model. A hidden Markov model (HMM) consists of five basic elements.

- S is a set of states,
- Z is a set of observations,
- $T_{s,s'} = p(s' \mid s)$ is a probabilistic state-transition model,
- $M_{s,z} = p(z \mid s)$ is a probabilistic observation model,
- $\pi(s) = p(s)$ is an initial state distribution.

Filtering. Recursive state estimation, usually called filtering, maintains recursively a probability distribution on the current state, given a history of observations.



- Start with initial distribution π.
- Prediction update: $\mathbf{p} \leftarrow \mathbf{p}\,T$. $\quad p_j \leftarrow \sum_i p_i T_{ij}$
- Observation update for observation z: $\mathbf{p} \leftarrow \eta \mathbf{p} \otimes M_z$. $\quad p_i \leftarrow \eta\, p_i\, M_{i,z}$ $\quad O(|S|)$
- Repeat the 2 previous steps at each time step t.

$O(|S|^2)$

Observation update is also called measurement update or correction update.

Formally, given a motion model T and a observation model M, we want to deduce the probability distribution $p(s_t)$ from the observation history $z_{1:t} = (z_1, z_2, \ldots, z_t)$:

$$p(s_t|z_{1:t}) = \frac{p(z_{1:t}, s_t)}{p(z_{1:t})}$$

To calculate $p(z_{1:t})$, we must condition on the state history $s_{1:t} = (s_1, s_2, \ldots, s_t)$, as we do not observe the states directly.

$$p(z_{1:t}) = \sum_{s_{1:t}} p(z_{1:t}|s_{1:t})p(s_{1:t})$$

Since there are $|S|^t$ terms in the sum, it is not practical to do so in brute force.

What we know

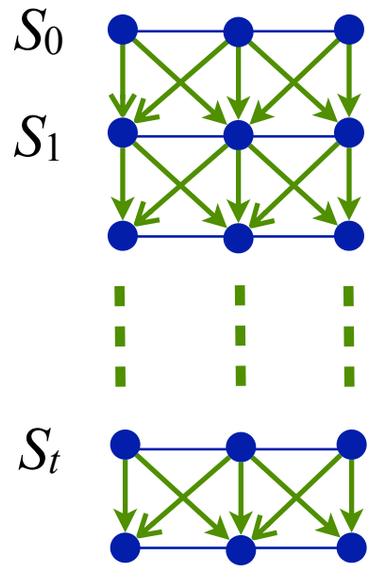
$T_{s,s'} = p(S_t = s' \mid S_{t-1} = s)$

$M_{s,z} = p(Z_t = z \mid S_t = s)$

$z_{1:t}$

What we want

$p(s_t \mid z_{1:t})$



Marginalization ("sum"): $p(x) = \int p(x, y)\,\mathrm{d}y$

Conditioning ("product"): $p(x, y) = p(x \mid y)p(y)$

Independence: $p(x \mid y) = p(x)$

"Case analysis": $p(x) = \int p(x \mid y)p(y)\,\mathrm{d}y$

Bayes' rule("swap"): $p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}$

Forward algorithm. We use the idea of dynamic programming again. Define

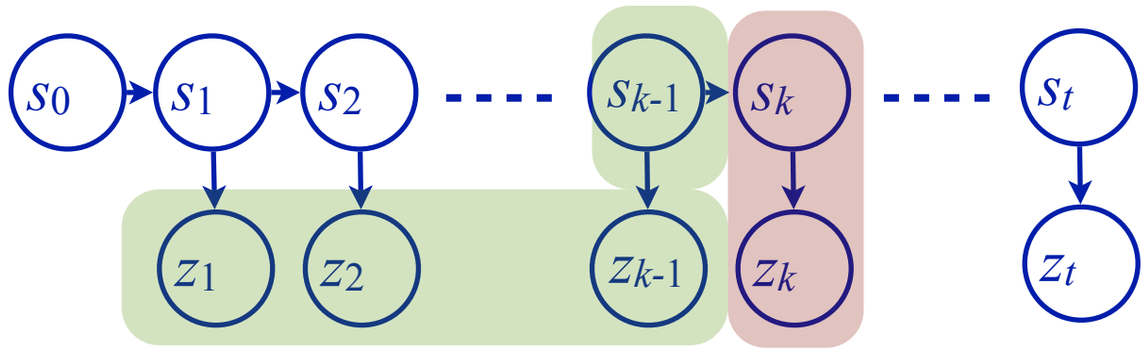
$$\alpha_k(s_k) = p(z_{1:k}, s_k)$$

To initialize,

$$\begin{aligned}
\alpha_1(s_1) &= p(z_1, s_1) \\
&= \sum_{s_0} p(z_1, s_1|s_0)p(s_0) \\
&= \sum_{s_0} p(s_1|s_0)p(z_1|s_0, s_1)p(s_0) \\
&= \sum_{s_0} p(s_1|s_0)p(z_1|s_1)p(s_0) \\
&= \sum_{s_0} T_{s_0,s_1} M_{s_1,z_1} \pi(s_0) \\
&= M_{s_1,z_1} \sum_{s_0} T_{s_0,s_1} \pi(s_0)
\end{aligned}$$

We then calculate α_k recursively in terms of α_{k-1}:

$$\begin{aligned}
\alpha_k(s_k) &= p(z_{1:k}, s_k) \\
&= p(z_{1:k-1}, z_k, s_k) \\
\text{marginalizing} \quad &= \sum_{s_{k-1}} p(z_{1:k-1}, s_{k-1}, z_k, s_k) \\
\text{definition} \quad &= \sum_{s_{k-1}} p(z_k, s_k | z_{1:k-1}, s_{k-1}) p(z_{1:k-1}, s_{k-1}) \\
\text{independence (Markov assumption)} \quad &= \sum_{s_{k-1}} p(z_k, s_k | s_{k-1}) p(z_{1:k-1}, s_{k-1}) \\
&= \sum_{s_{k-1}} p(s_k | s_{k-1}) p(z_k | s_k) p(z_{1:k-1}, s_{k-1}) \\
&= \sum_{s_{k-1}} T_{s_{k-1}, s_k} M_{s_k, z_k} \alpha_{k-1}(s_{k-1}) \\
&= M_{s_k, z_k} \sum_{s_{k-1}} T_{s_{k-1}, s_k} \alpha_{k-1}(s_{k-1})
\end{aligned}$$



To use the forward algorithm for filtering, note that

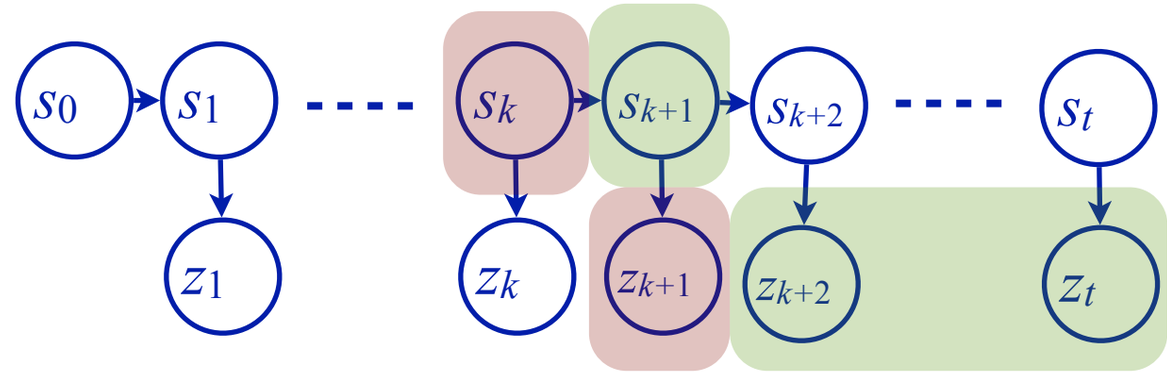
$$p(s_t|z_{1:t}) = \frac{p(z_{1:t}, s_t)}{p(z_{1:t})} = \eta \alpha_t(s_t)$$

and

$$1/\eta = p(z_{1:t}) = \sum_{s_t} p(z_{1:t}, s_t) = \sum_{s_t} \alpha_t(s_t)$$

Backward algorithm. Similarly, define

$$\beta_k(s_k) = p(z_{k+1:t}|s_k)$$



To initialize,

$$\beta_t(s_t) = 1 \quad \text{for all } s_t \in S\,.$$

We then recurse backwards and calculate β_k recursively in terms of β_{k+1}:

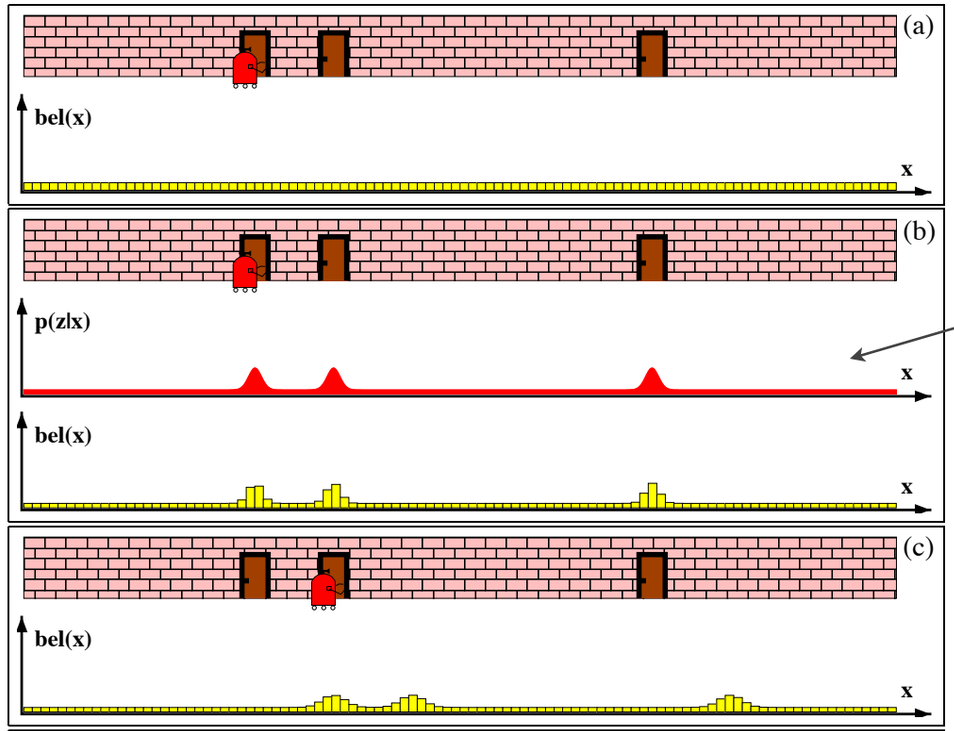
$$\beta_k(s_k) = \sum_{s_{k+1}} T_{s_k, s_{k+1}} M_{s_{k+1}, z_{k+1}} \beta_{k+1}(s_{k+1})$$

Example. Localization with HMM.

initial belief

observation update

prediction update

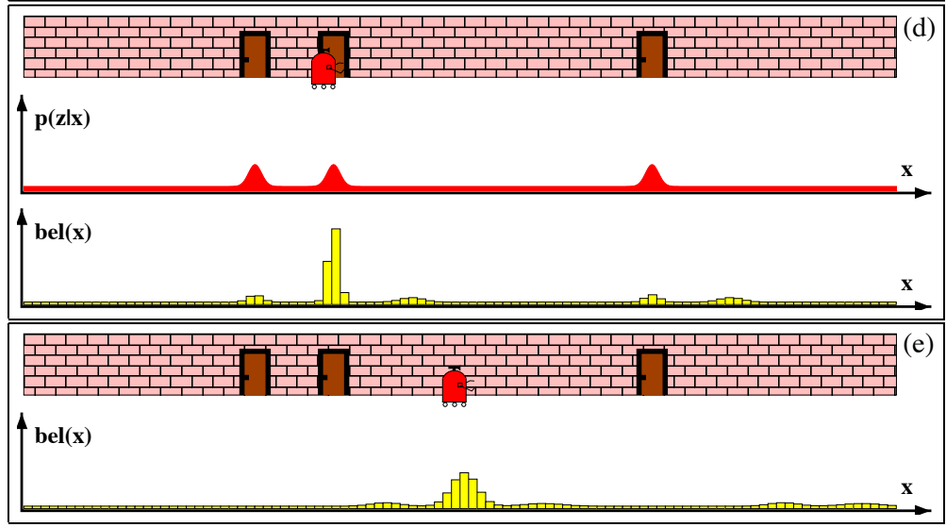


observation function

z = “door”

observation update

prediction update



...

Question. After a prediction update, the distribution appears more "spread out". Why? Does this always happen in general?

Question. After an observation update, the distribution appears more concentrated? Why? Does this always happen in general?

Related HMM inference questions. In addition to filtering, α_k and β_k can help to answer several related inference questions, given a history of observations $(z_1, z_2, \ldots, z_t)$:

- Filtering. Compute the probability distribution of S_t at current time t.
  Example. Where is the mole now?
- Prediction. Compute the probability distribution of S_k for some $k > t$.
  Example. Where will the mole be at a later time k?
- Smoothing. Compute the probability distribution of S_k for some $k < t$.
  Example. Where was the mole at an earlier time k?
- Decoding. Determine the most likely state history $(s_1, s_2, \ldots, s_t)$.

For prediction, we take two steps:

- Perform filtering to calculate $p(s_t \mid z_{1:t})$.
- Set $p(s_t \mid z_{1:t})$ as the initial probability distribution of the corresponding Markov chain and project forward for k - t steps, using the motion model T.

For smoothing,

$$p(s_k|z_{1:t}) = \frac{p(s_k, z_{1:t})}{p(z_{1:t})}$$

$$= \frac{p(s_k, z_{1:k}, z_{k+1:t})}{p(z_{1:t})}$$

$$\text{definition} = \frac{p(s_k, z_{1:k})p(z_{k+1:t}|s_k, z_{1:k})}{p(z_{1:t})}$$

$$\text{independence} = \frac{p(s_k, z_{1:k})p(z_{k+1:t}|s_k)}{p(z_{1:t})}$$

$$\text{definition} = \eta\alpha_k(s_k)\beta_k(s_k)$$

Basyesian filter. HMM assumes a discrete state space. HMM filtering takes time $O(|S|^2)$ per step in the worst case, where $|S|$ is the number of states. Just as the MDP, it suffers from the "curse of dimensionality". Further, what shall we do if S is continuous?

HMM filtering is an instance of Bayesian filtering. To handle large discrete state spaces or continuous spaces, we need computationally efficient representations, but the underlying idea of filtering remains the same:

- Prediction update

Replace by $\int$ for continuous space.

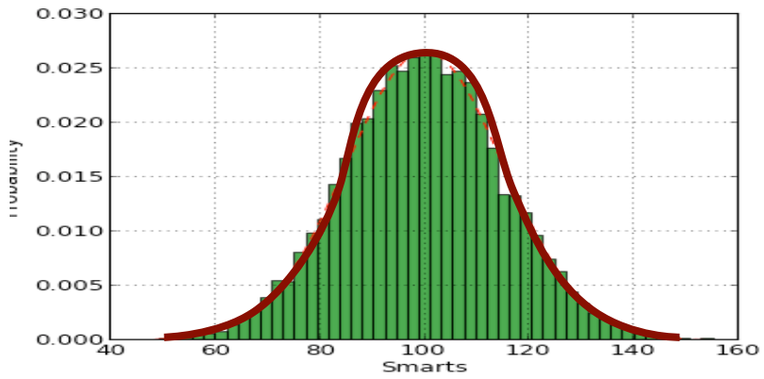
$$p(S_t = s') = \sum_{s \in S} p(S_t = s' | S_{t-1} = s) p(S_{t-1} = s)$$

- Observation update

$$p(S_t = s | Z_t = z) = \eta p(Z_t = z | S_t = s) p(S_t = s)$$

Representing continuous probability distributions.

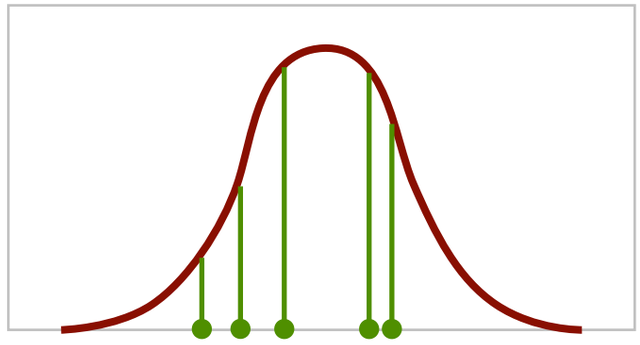
There are three main approaches:



- Histogram. The histogram representation discretizes the state space uniformly as a grid and approximates the probability within each grid cell. This leads to an HMM that we have just seen.
- Particles. The particle representation samples a set of states from the state space and approximates the probability mass at the samples points:

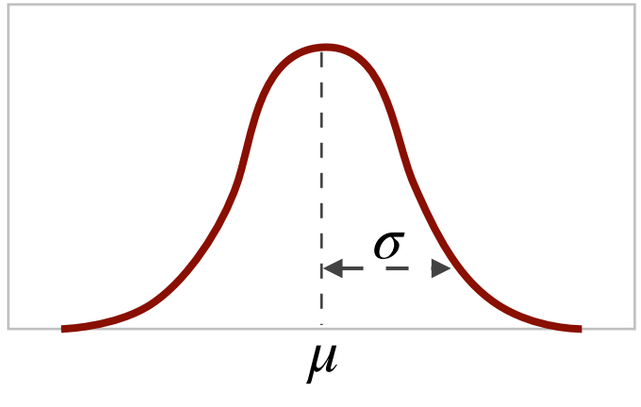
$$\{(s_1, p_1), (s_2, p_2), \ldots, \}$$

sampled state particle weight

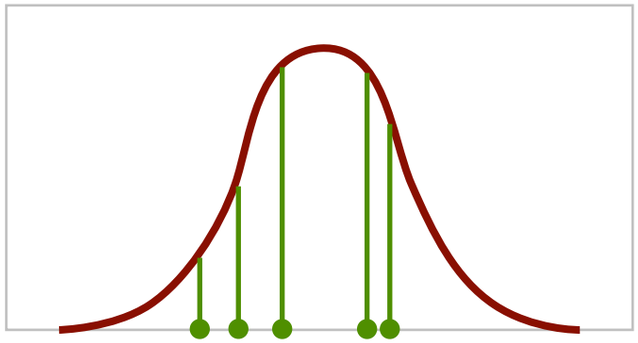


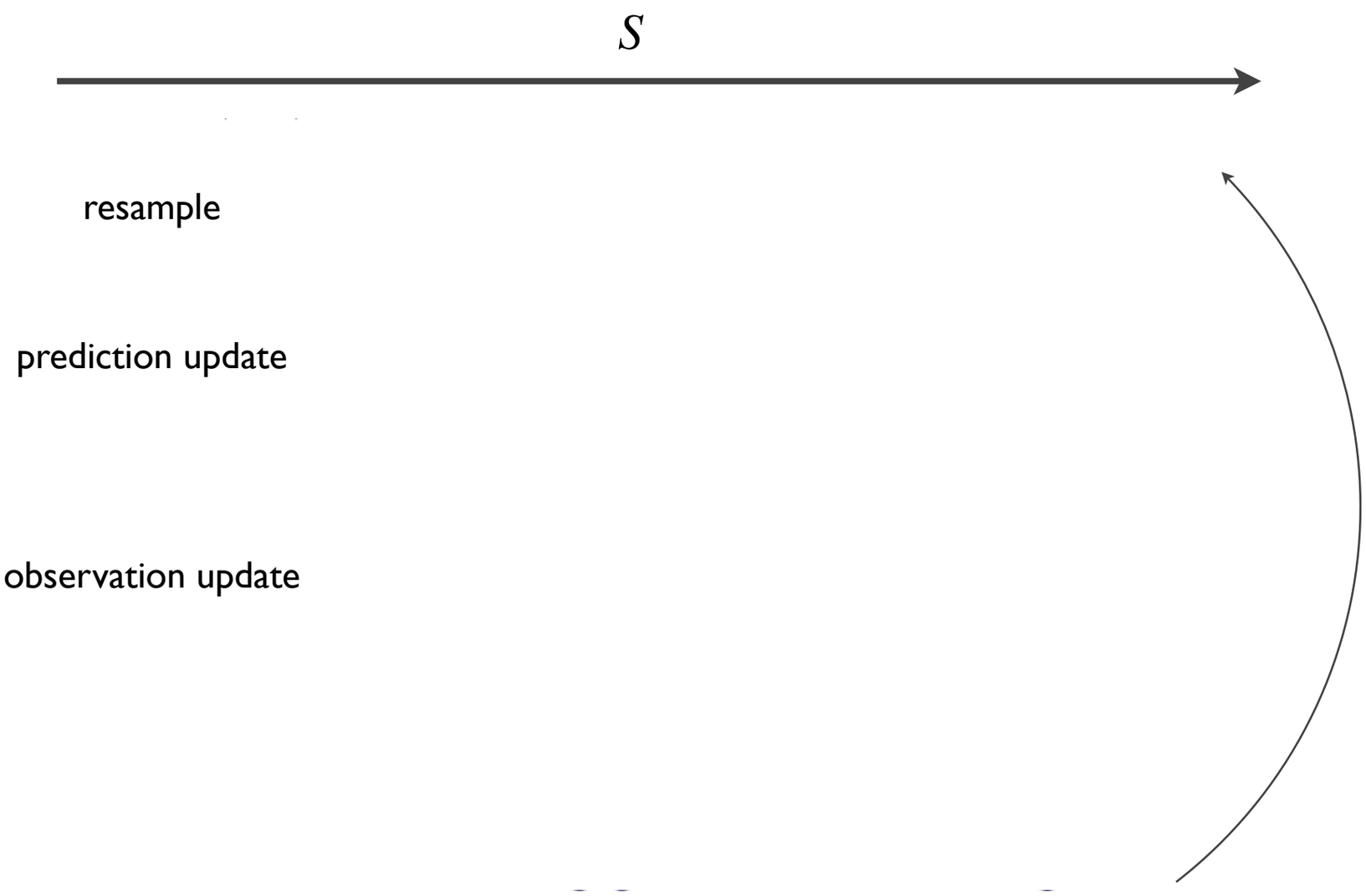
Sampling overcomes the "curse of dimensionality", just as it does for planning, through PRM, EST, RRT, ….

- Parametric representations. Approximate the underlying probability distribution as a continuous parametric function. The most common, important instance is the Gaussian distribution with parameters
  - mean μ
  - covariance matrix Σ.

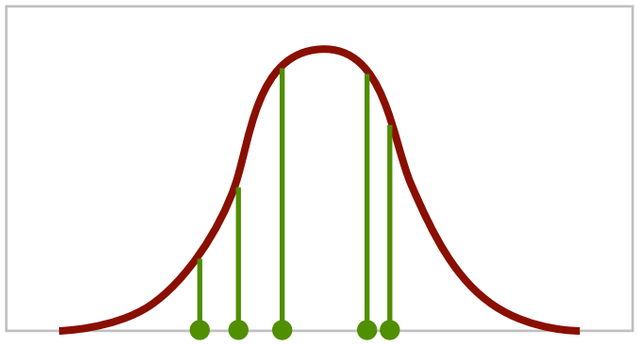


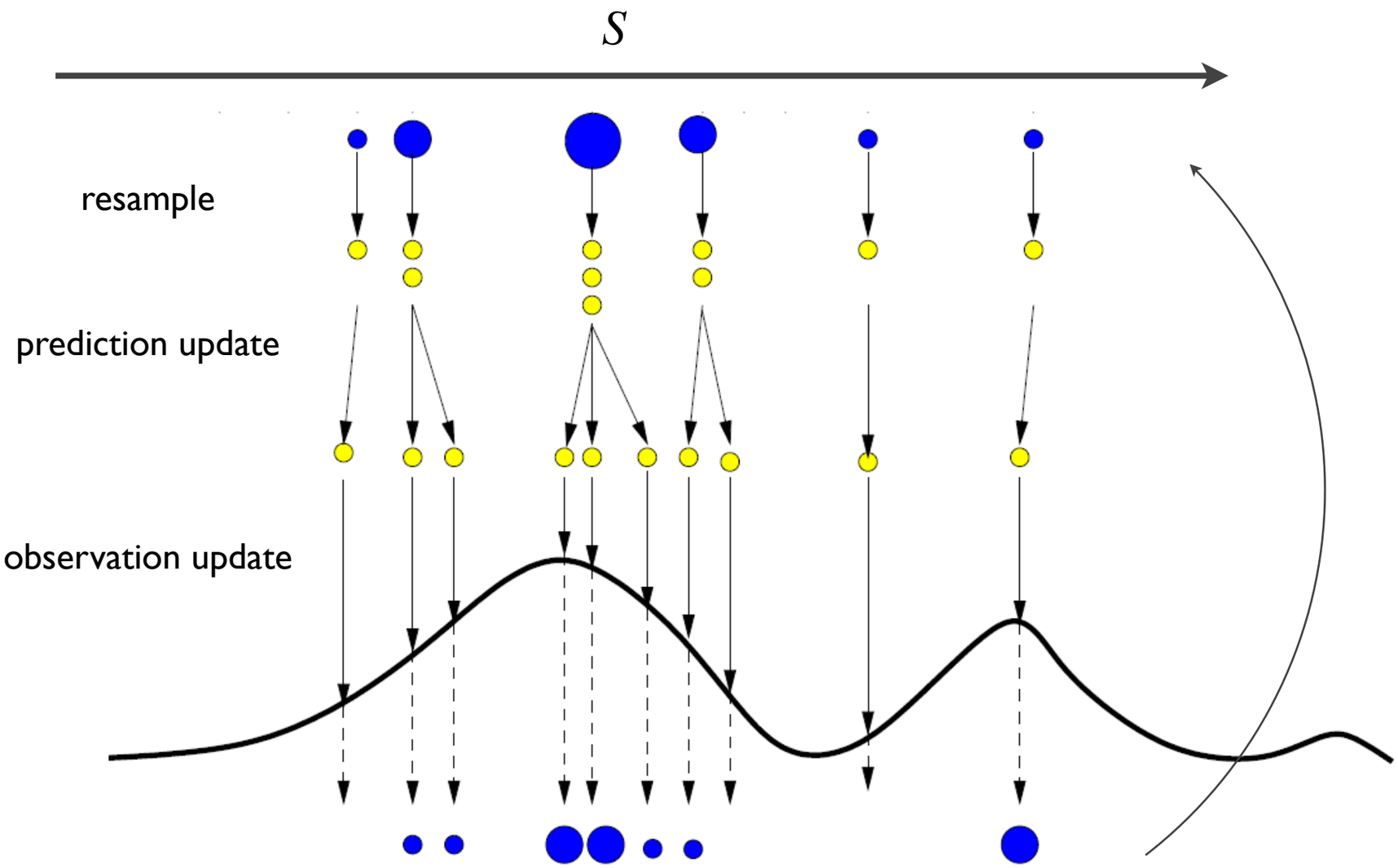
Particle filter. Sequential importance resampling (SIR) is among the most commonly used particle filter. It represents a continuous probability distribution as a set of weighted particles.



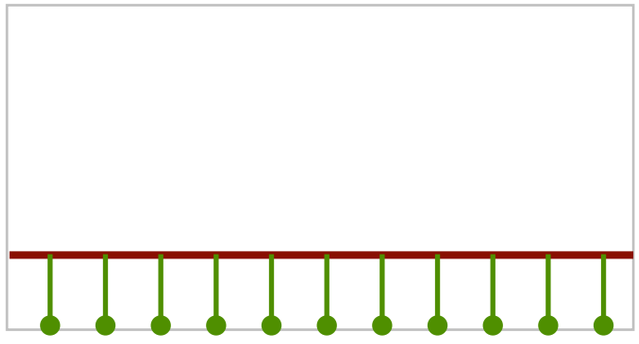


Particle filter. Sequential importance resampling (SIR) is among the most commonly used particle filter. It represents a continuous probability distribution as a set of weighted particles.

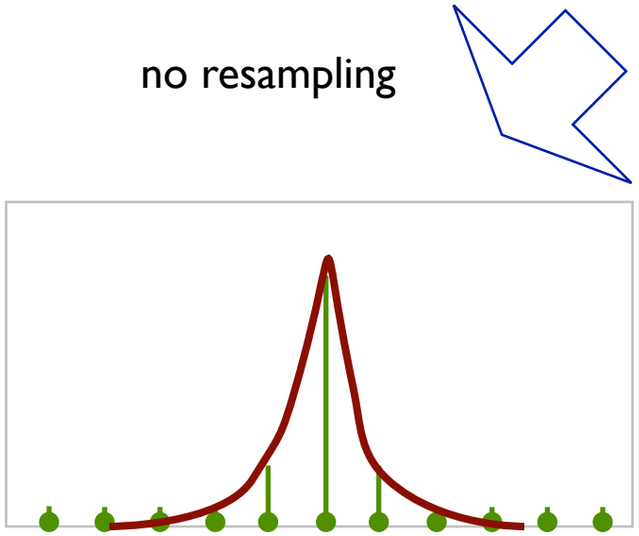




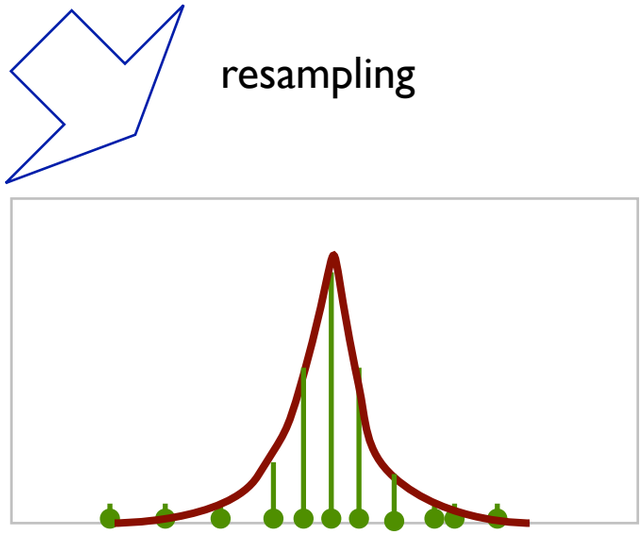
Why is the resampling step useful? Place articles at high-probability states.



no resampling



resampling



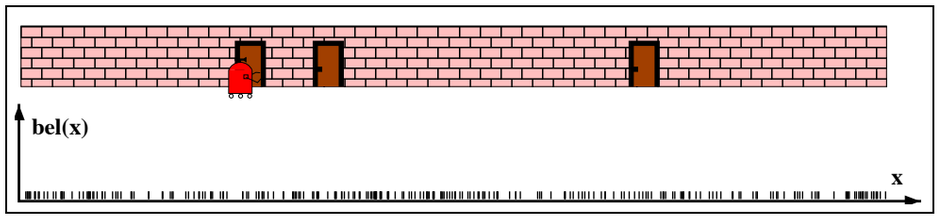
SIR is summarized below:

- Initialize by sampling a set X of N states from S according to the initial distribution π. $X \leftarrow \{(s_i, 1/N) \mid i=1, 2, ..., N\}$.
- For each $s_i \in X$, $O(cN)$
  prediction update: sample a new state s_i' according to $p(s_i' \mid s_i)$; $O(c)$
  observation update for observation z: $p_i' = p(z \mid s_i')$; $O(1)$
  Normalize p_i' so that $\sum_{i=1}^{N} p_i' = 1$;
  $X \leftarrow \{(s_i', p_i') \mid i=1, 2, ..., N\}$.
- Perform importance resampling: choose a new set X' of N points from X so that $s_i' \in X$ is sampled with probability p_i'. $O(N)$
  $X \leftarrow \{(s, 1/N) \mid s \in X'\}$.
- Repeat the 2 previous steps at each time step t.

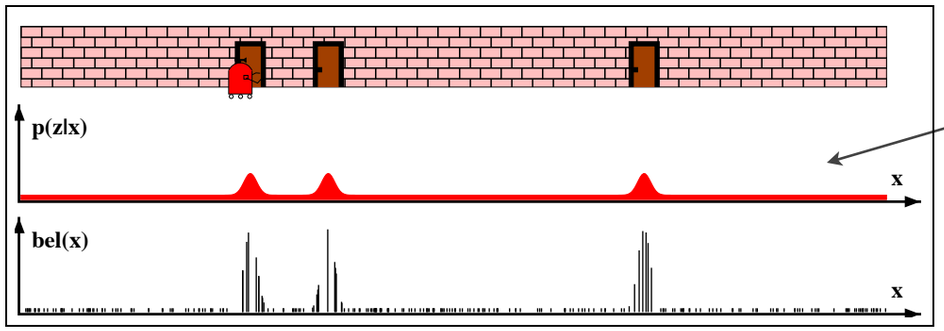
The particle filter approximates a **complex multi-modal** probability distribution over a high-dimensional discrete state space or a continuous state space with N particles. As $N \rightarrow \infty$, the particle approximation approaches the true underlying probability distribution asymptotically.

Example. An illustrative example for localization with particle filtering.

initial belief



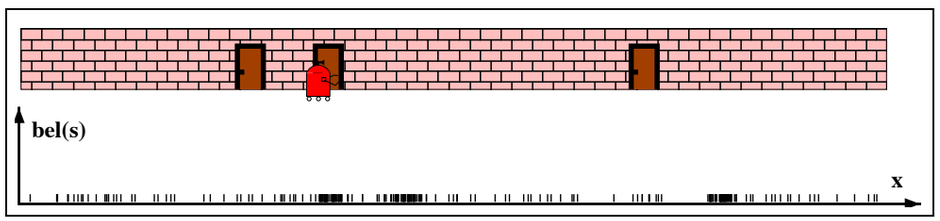
observation update



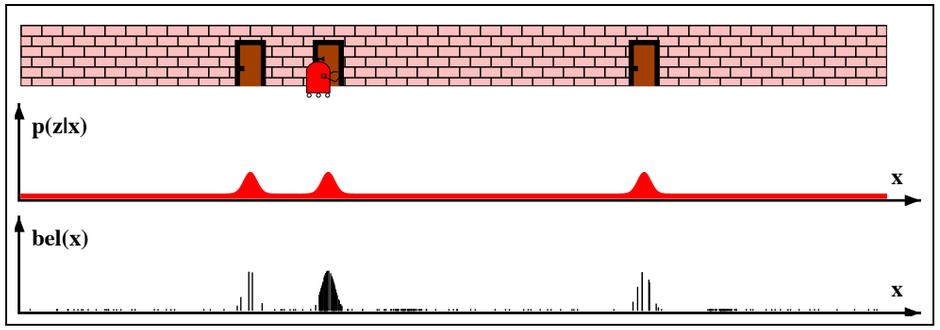
observation function

z = "door"

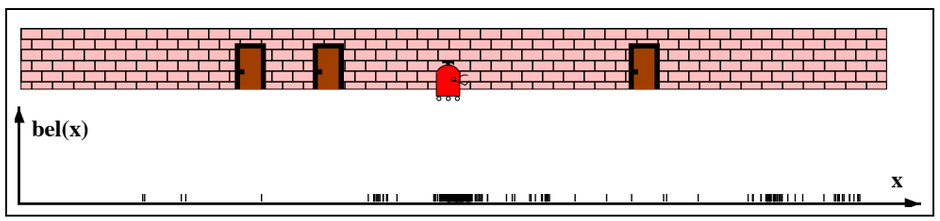
prediction update
resample



observation update

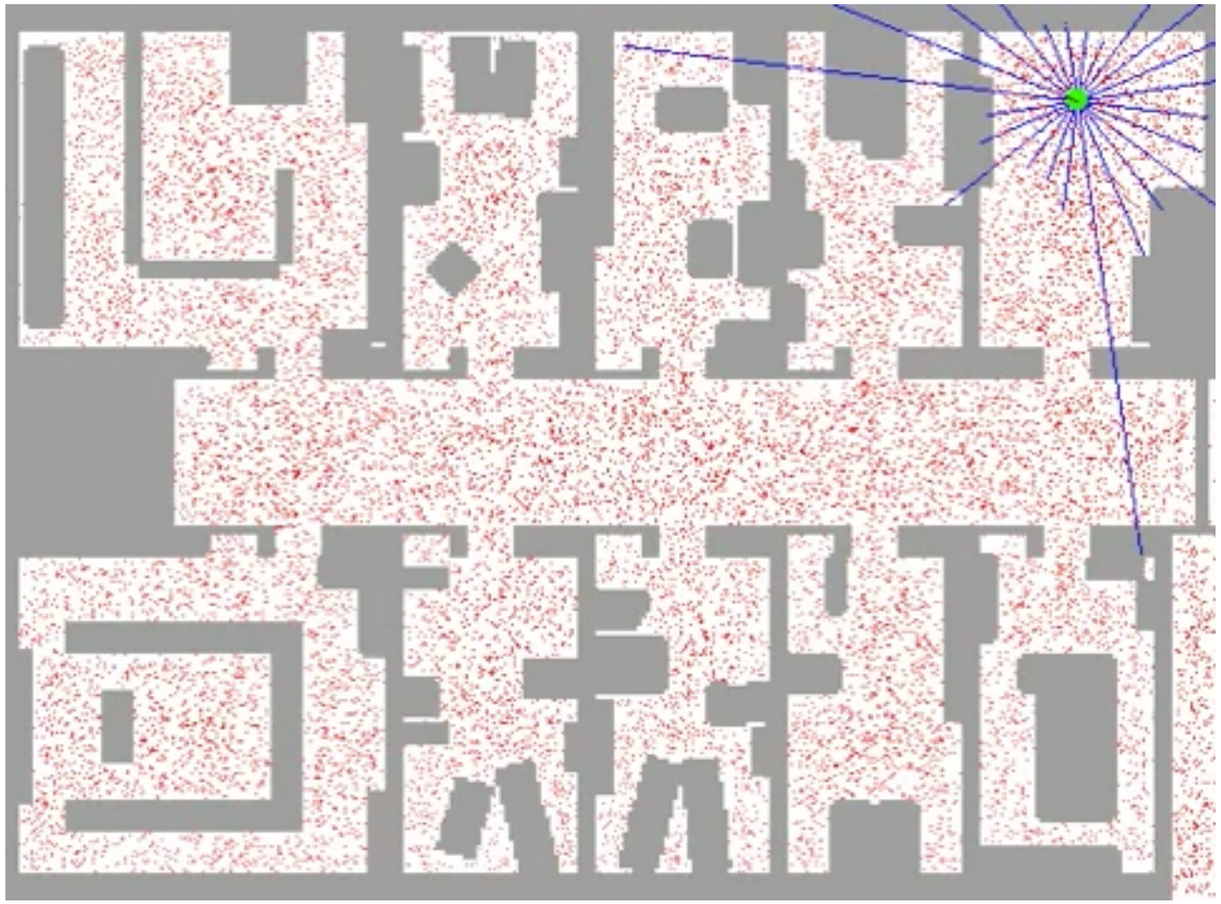


prediction update
resample



...

Example. LIDAR localization with particle filtering.

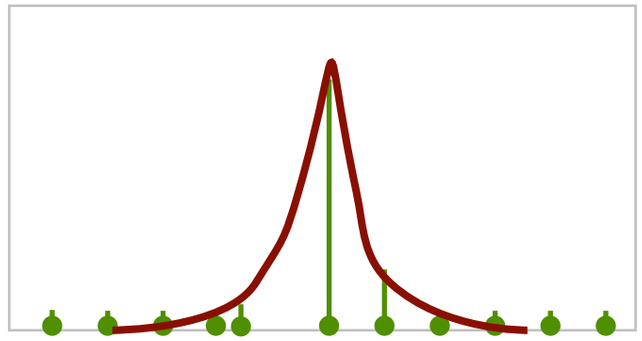


The particle filter can approximate arbitrary an arbitrary probability distribution. It is usually used for global localization when we know not have good prior knowledge of the robot's initial location, in other words, the initial state distribution on the robot's location is nearly uniform.

The number of particles is a key parameter of the algorithm. More particles improve the representation accuracy, but also increase the computational cost. In practice, there are two common causes of particle filtering failure:

- insufficient particles
- particles not well distributed to capture the underlying distribution

The resampling step in SIR attempt to address the second issue, but it is not perfect and introduces other difficulties.



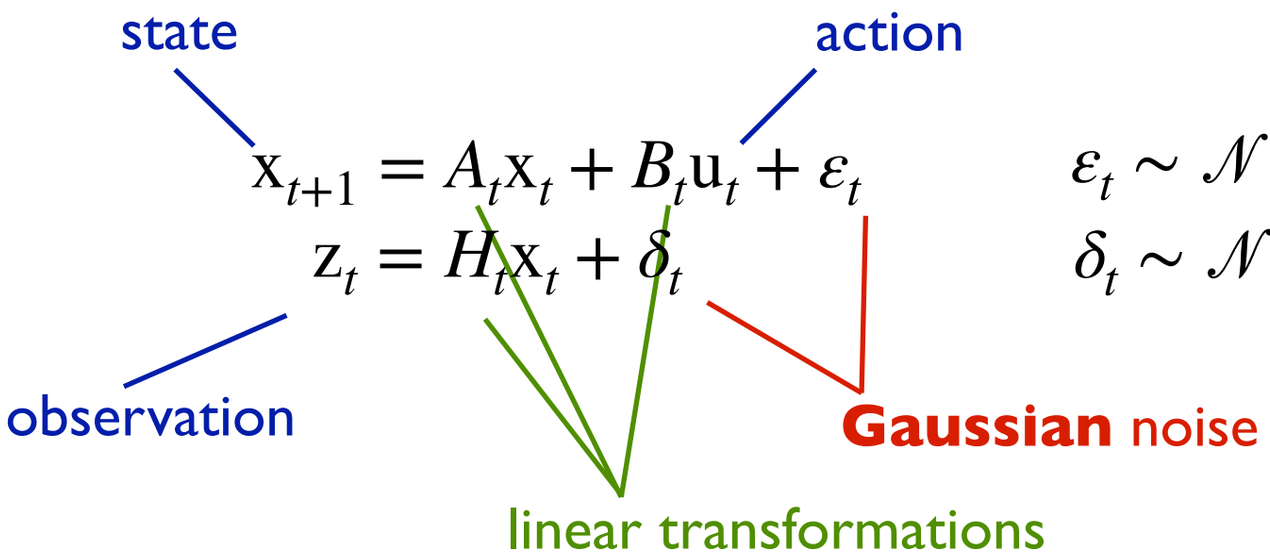
Summary.

- The HMM is a discrete probability model that allows us to answer various inferences questions about the hidden states from a history of observations, including filtering, prediction, smoothing, … For robotics, the most important and common question is filtering, i.e., estimating the current state.
- The MDP assumes that the current state is known exactly and provides the actions to control the robot. The HMM assumes that the current state is not known exactly and provides the inference mechanism to reason about the hidden states. As discrete state-space models, both suffer from the "curse of dimensionality".
- The particle filter is a powerful filtering algorithm widely used in practice. The SIR algorithm approximates a continuous probability distribution adaptively as a set of weighted particles and approaches the true underlying probability distribution asymptotically as the number of particles increases.

Kalman filtering. The Kalman filter makes the following key assumptions on the system dynamic model and the observation model:

- they are **linear**;
- they have **Gaussian** noise.

As a result, the probability distribution over the state is always Gaussian. This makes the Kalman filter very efficient. At the same time, it is also the main reason for the limitations.



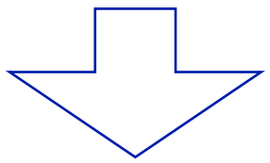
The equations above specify the probabilistic state-transition model $p(x_{t+1} \mid x_t, u_t)$ conditioned on the action u_t and the observation model $p(z_t \mid x_t)$.

Following the conventions, we write these equations in terms of random variables (RVs). They serve the purpose of specifying the state-transition and observation models.

The state-transition model is conditioned on the action u_t . It is not required, but is more general. Similarly, we can have $p(s_{t+1} \mid s_t, a_t)$ for HMM filtering or particle filtering. u_t or a_t must be known at each time step t.

Example. A robot moves in an one-dimensional space. At time t, the state is (x_t, v_t), where x_t and v_t are the position and the speed of the robot, respectively. The robot can apply a force to control the acceleration a_t. The speedometer on board measures v_t, but not x_t.

$$
\begin{aligned}
x_{t+1} &= x_t + v_t \\
v_{t+1} &= v_t + a_t + \phi_t \\
z_t &= v_t + \psi_t
\end{aligned}
$$

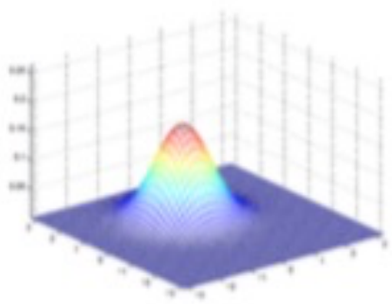


$$
\begin{pmatrix} x_{t+1} \\ v_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_t \\ v_t \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} a_t + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \phi_t
$$

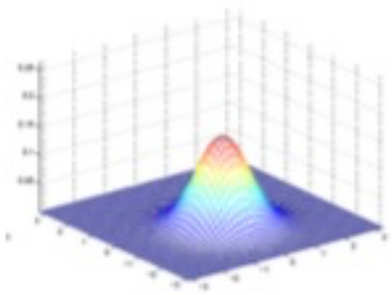
$$
z_t = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} x_t \\ v_t \end{pmatrix} + \psi_t
$$

Our objective is to calculate $p(\mathrm{x}_t \mid \mathrm{u}_{0:t-1}, \mathrm{z}_{1:t})$, which is Gaussian with mean μ_t and covariance matrix Σ_t :

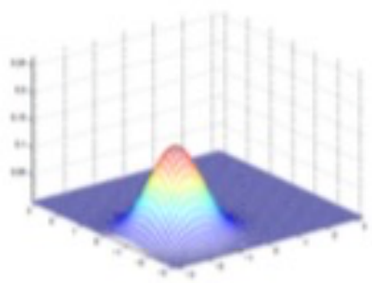
$$p(\mathrm{x}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathrm{x}-\mu)^T \Sigma^{-1} (\mathrm{x}-\mu) \right\}$$



- $\mu = [1; 0]$
- $\Sigma = [1\ \ 0; 0\ \ 1]$

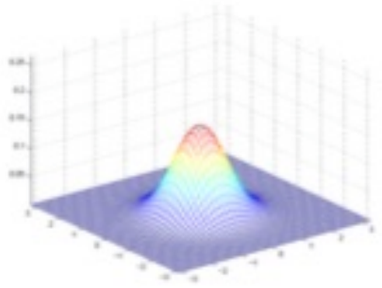


- $\mu = [-.5; 0]$
- $\Sigma = [1\ \ 0; 0\ \ 1]$

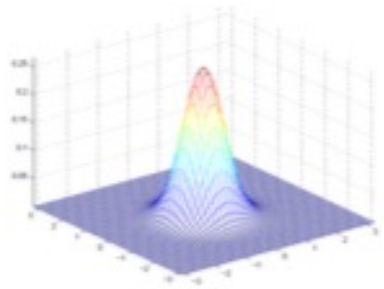


- $\mu = [-1; -1.5]$
- $\Sigma = [1\ \ 0; 0\ \ 1]$

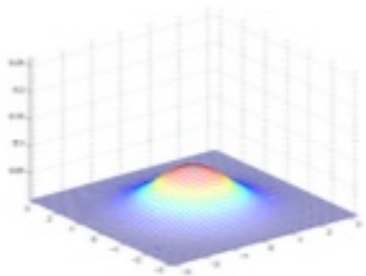
Translate



- $\mu = [0; 0]$
- $\Sigma = [1\ 0\ ; 0\ 1]$

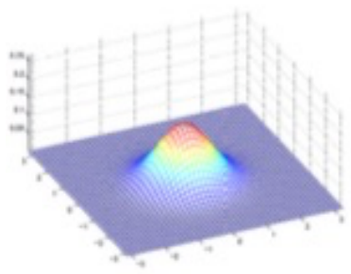


- $\mu = [0; 0]$
- $\Sigma = [.6\ 0\ ; 0\ .6]$

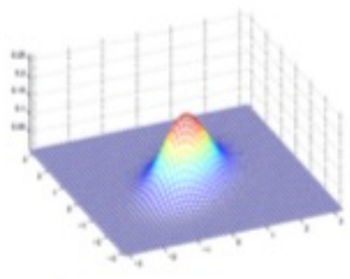


- $\mu = [0; 0]$
- $\Sigma = [2\ 0\ ; 0\ 2]$

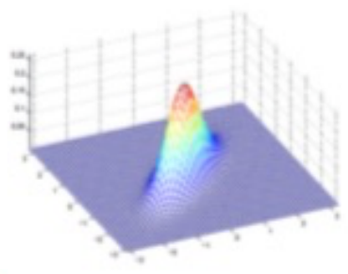
Change the spread.



- $\mu = [0; 0]$
- $\Sigma = [1\ \ 0; 0\ \ 1]$

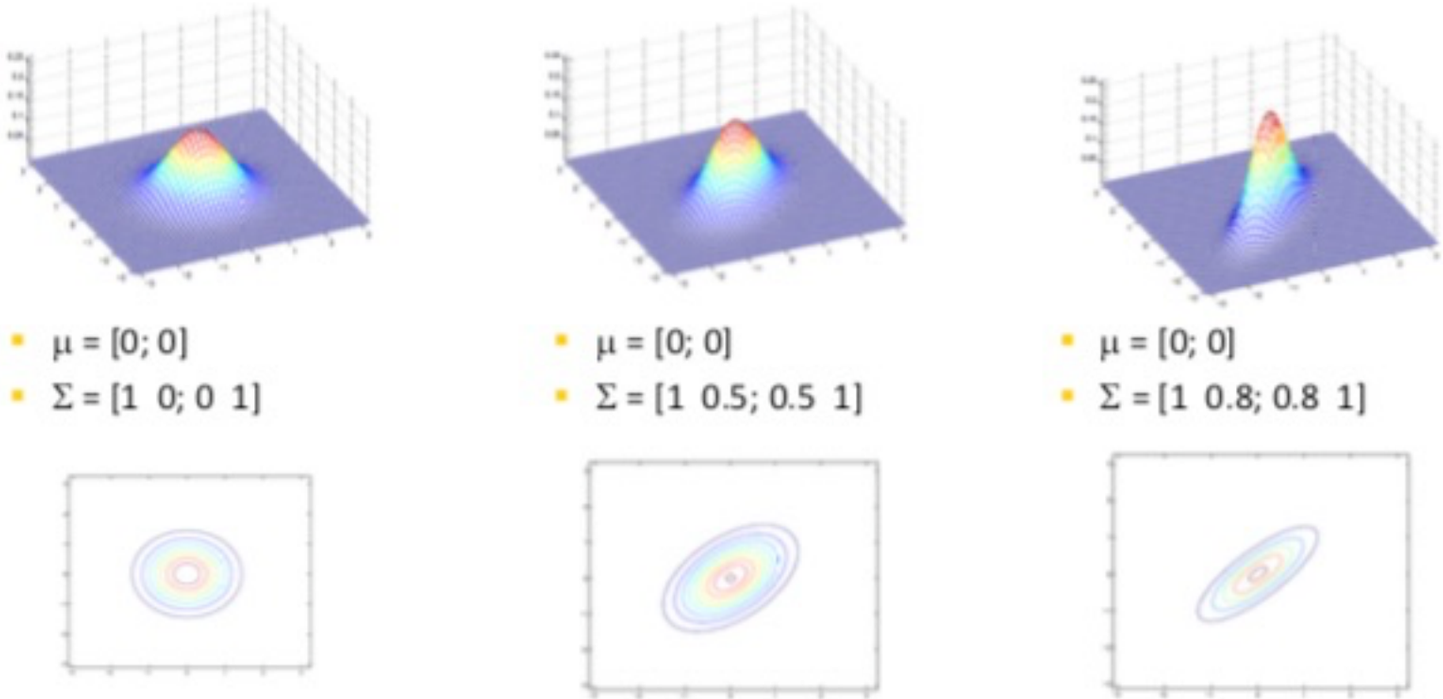


- $\mu = [0; 0]$
- $\Sigma = [1\ \ 0.5; 0.5\ 1]$



- $\mu = [0; 0]$
- $\Sigma = [1\ \ 0.8; 0.8\ \ 1]$

Change the spread and rotate.



Kalman filtering update μ_t and Σ_t recursively:

- At time 0, we have μ_0, Σ_0.
- For $t = 1,2,\ldots,$

  prediction update

$$\mu_{t+1}^{-} = A_t\mu_t + B_t\mathrm{u}_t$$
$$\Sigma_{t+1}^{-} = A_t\Sigma_t A_t^T + Q_t$$

The control action does not affect the covariance, i.e., the uncertainty. This is a unique property of linear systems.

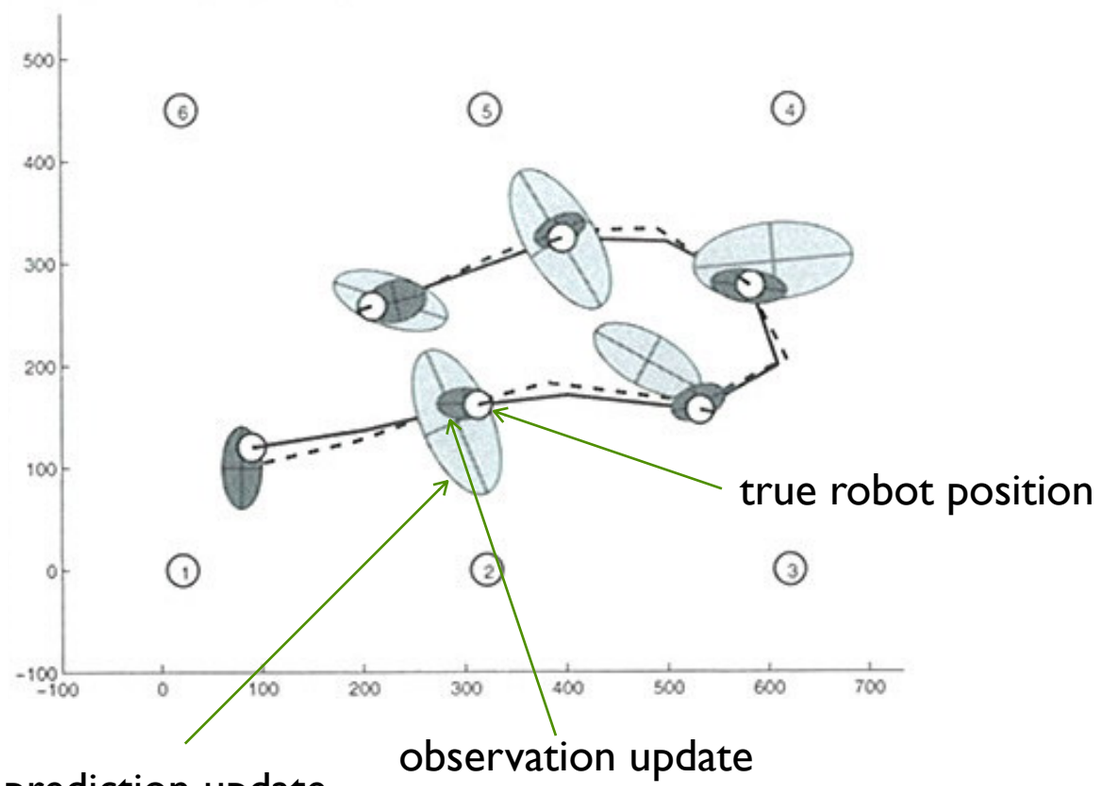
  observation update

$$\mu_{t+1} = \mu_{t+1}^{-} + K_{t+1}(z_{t+1} - H_{t+1}\mu_{t+1}^{-})$$
$$\Sigma_{t+1} = (I - K_{t+1}H_{t+1})\Sigma_{t+1}^{-}$$

  where $K_{t+1} = \Sigma_{t+1}^{-}H_{t+1}^T(H_{t+1}\Sigma_{t+1}^{-}H_{t+1}^T + R_{t+1})^{-1}$ is the Kalman gain.

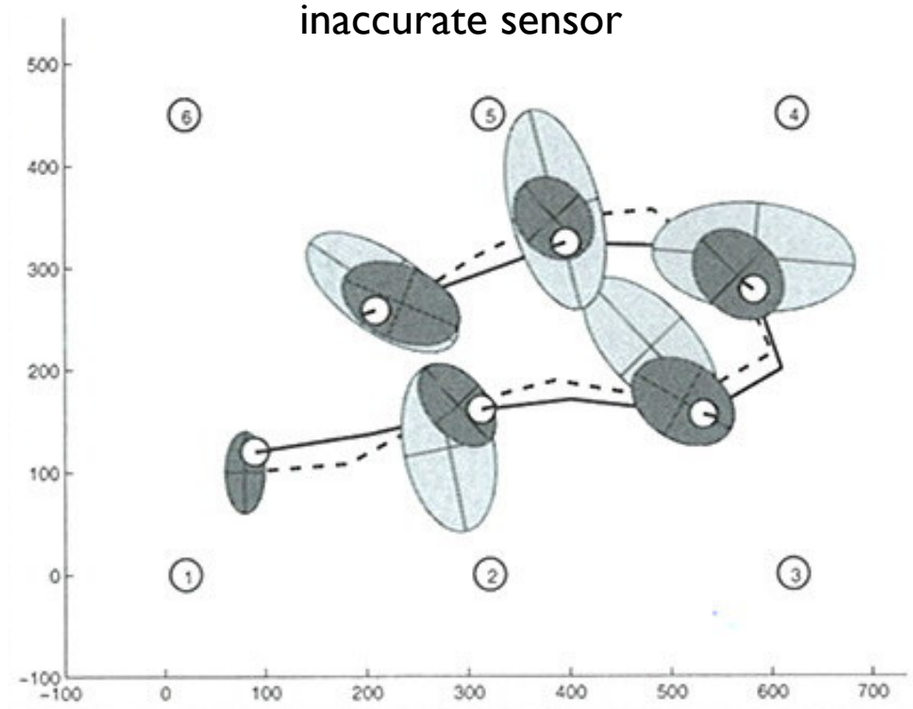
$$\mathrm{x}_{t+1} = A_t\mathrm{x}_t + B_t\mathrm{u}_t + \varepsilon_t$$
$$\mathrm{z}_t = H_t\mathrm{x}_t + \delta_t$$
$$\varepsilon_t \sim \mathcal{N}(0,Q_t)$$
$$\delta_t \sim \mathcal{N}(0,R_t)$$

accurate sensor



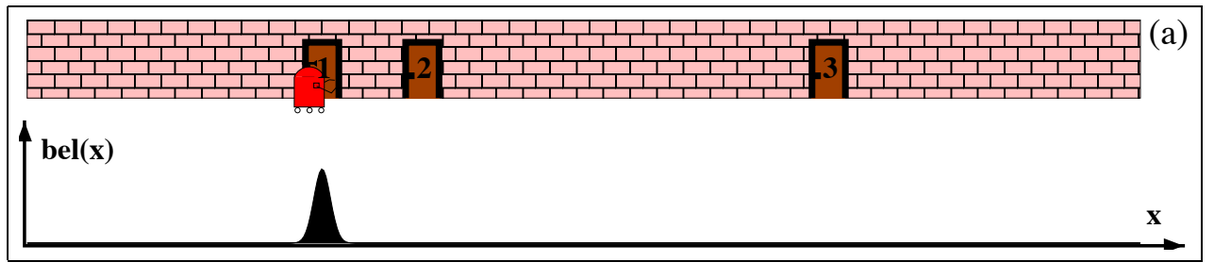
With an accurate sensor, the state distribution is sharper after the observation update.

inaccurate sensor

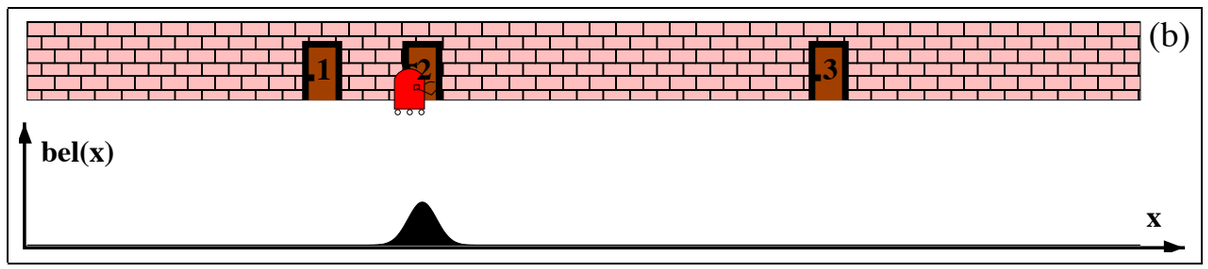


Example. An illustrative example for localization with Kalman filtering.

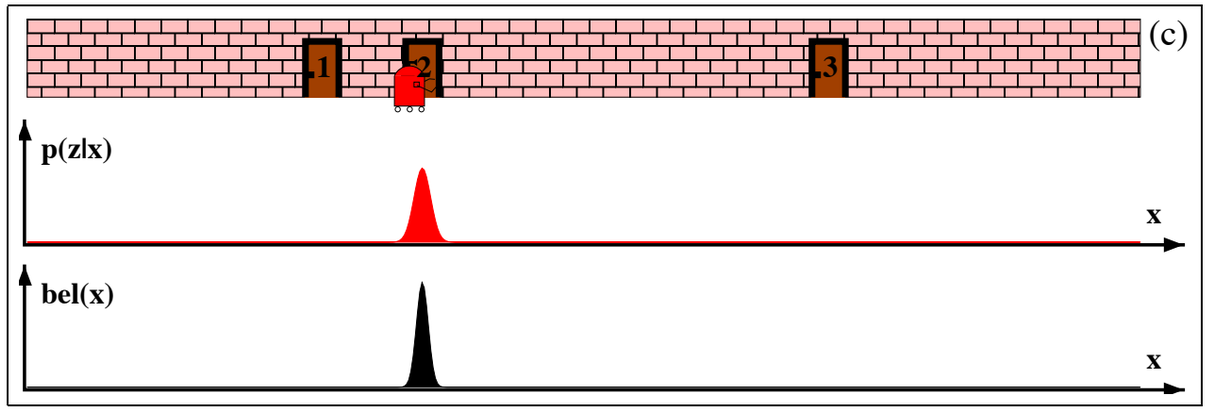
initial belief



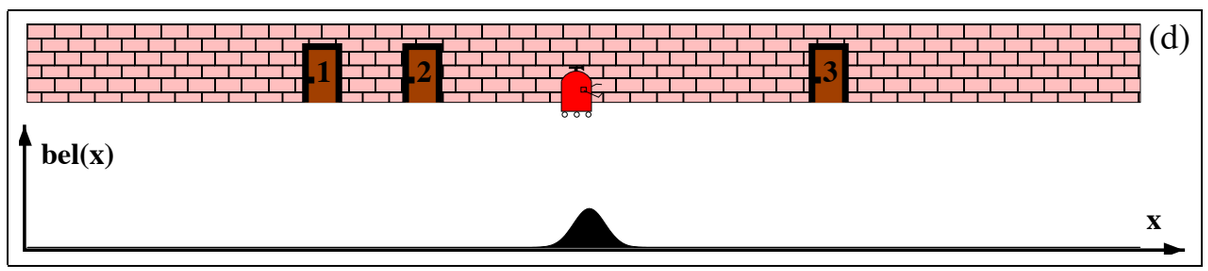
prediction update



observation update



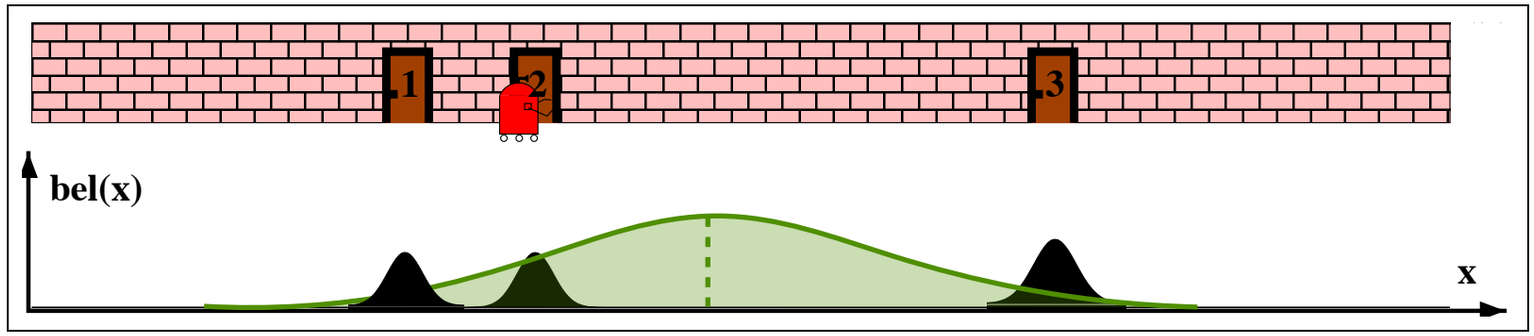
prediction update



...

Kalman filtering is $O(D^2)$, where D is the dimension of the state space S. In contrast, HMM filtering is $O(|S|^2)$. PF is $O(N)$, where N is the number of particles. In general, $D \ll N \ll |S|$. So Kalman filtering is the most efficient one computationally.

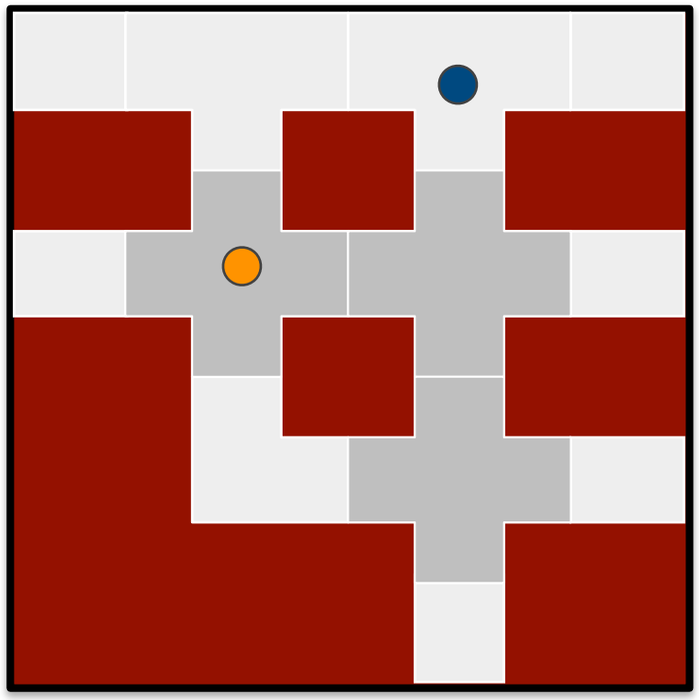
At the same time, the Gaussian assumption is restrictive. Kalman filtering does not perform well for complex, multi-modal distributions. Consider the example below. The ground-truth state distribution has three modes, each located at a door. The approximating Gaussian has its mean located in front the wall. Very likely, the robot will make the wrong decision as a result.



Localization with a map. What information does a map provide? How do we use a map for robot localization? The map contains spatial information about the environment. We use the map M to build the observation model $p(z \mid s, M)$:

observation

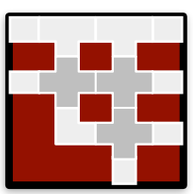
map



$p(z \mid s)$: $p($  $\mid$  ,  $) = 0,$

$p($  $\mid$,  $) = 1,$

Localization:

$p(s \mid z)$: $p($  $\mid$  ,  $) = 0,$

$p($  $\mid$  ,  $) = 1/3$

Localization variants. There are various localization tasks that differ in their assumptions:

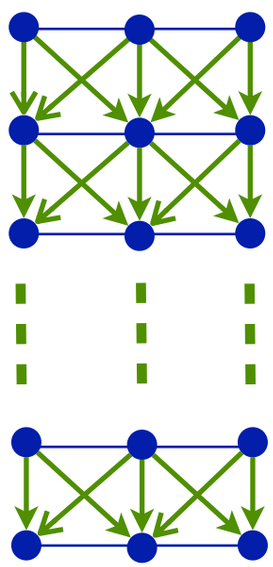
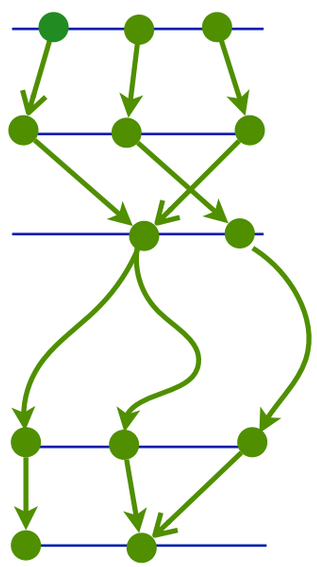
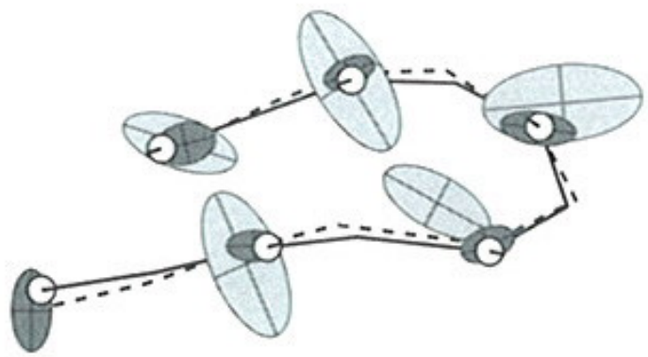
- tracking
- global localization
- active localization.

Tracking assumes good knowledge of the robot initial state. Further, the uncertainty remains small over time. If the underlying state distribution is well approximated as Gaussian, then Kalman filtering is a good solution.

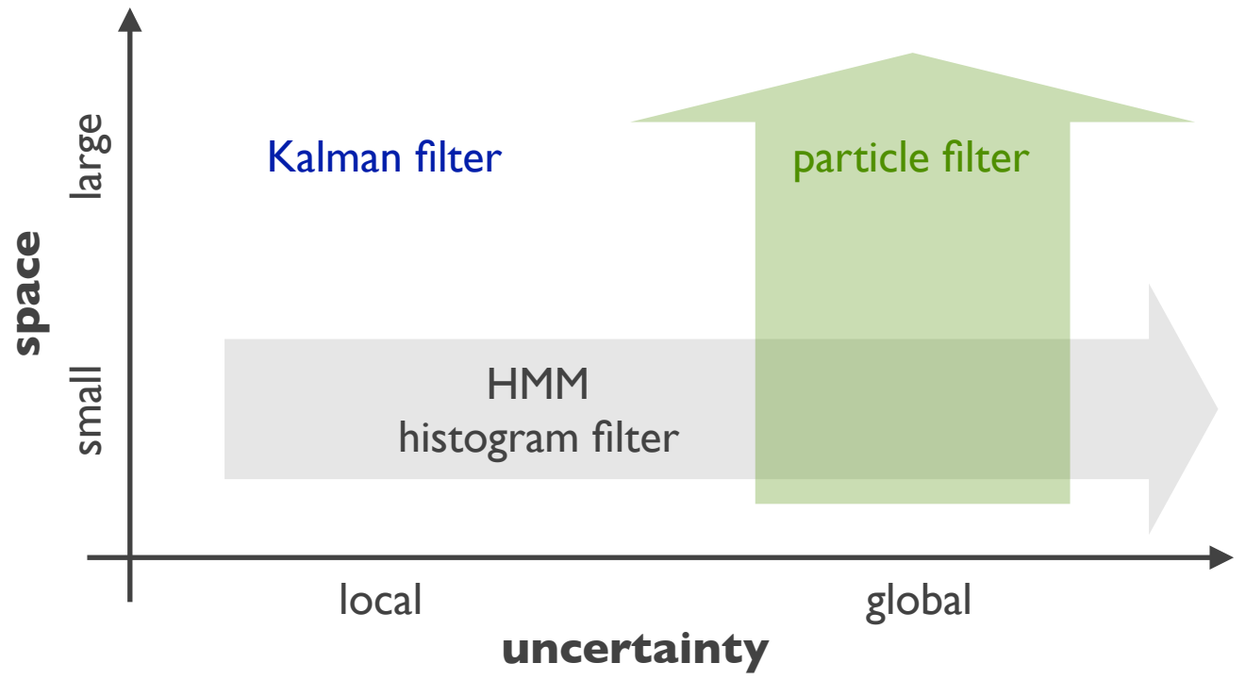
In global localization, the robot initial state may not be known accurately. The uncertainty on the robot state may become very large. Under these conditions, PF is the most common solution.

All filtering algorithms **passively** absorb information. The robot may actively explore in order to localize, e.g., by moving to a location with uniquely identifiable landmarks.

Summary.

	HMM, histogram filter	particle filter	Kalman filter
			
	● state	● sampled state	
uncertainty model	arbitrary state distribution	arbitrary state distribution	Gaussian
algorithm	dynamic programming	Monte Carlo sampling	parametric approximation closed form recursion
task	global localization	global localization	tracking
computational efficiency	✗	✓	✓✓
robustness	✓	✓✓	✗

Summary.



- A good choice for the filtering algorithm depends on the size of the state space, the action space, and the observation space. It also depends on the assumption on the scale of uncertainty.
- Histogram filter
  - To apply the histogram filter, we need to represent the state-transition model $T_{s,a,s'} = p(s' \mid s, a)$ and $M_{s,a,z} = p(z \mid a, s)$ explicitly as tables. It thus suffers from the "curse of dimensionality". It is practical only if the state space, the action space, and the observation space are all small.
  - For global localization in a small state space, the histogram filter is more efficient than the particle filter, through the use of dynamic programming.

- Particle filter
  - To overcome the curse of dimensionality, the particle filter uses probabilistic sampling. It approximates an arbitrary state distribution with a sufficiently large number of particles. It is also reasonably efficient.
  - The particle filter is relatively simple to implement and widely used in practice.
  - However, sampling results in statistical variance, manifested as particle depletion. This is a major difficulty and must be addressed carefully for the particle filter to work well in practice.

For robot localization in 2-D environments, the state space size is moderate. Nevertheless, the particle filter is sometimes preferred over the histogram filter, as it does not require explicit discretization of continuous systems and is easier to use.

- The Kalman filter approximates the underlying state distribution as a Gaussian distribution. It is very efficient because of the parametric approximation, but works well only if the Gaussian assumption is verified. It is often used for tracking when the initial robot state is known accurately and the uncertainty remains small.

Key concepts.

- Kalman filter