7630 – Autonomous Robotics

Probabilistic Localisation

Principles of Probabilistic Localisation

Particle Filters for Localisation

Kalman Filter for Localisation

Based on material from R. Triebel, R. Kästner, R. Siegwart, D. Scaramuzza

The Bayes Filter

$$= p(x_t \mid u_1, z_1, \ldots, u_t, z_t)$$

(Bayes) $$= \eta\ p(z_t \mid x_t, u_1, z_1, \ldots, u_t) p(x_t \mid u_1, z_1, \ldots, u_t)$$

(Markov) $$= \eta\ p(z_t \mid x_t) p(x_t \mid u_1, z_1, \ldots, u_t)$$

(Tot. prob.) $$= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_1, z_1, \ldots, u_t, x_{t-1})$$
$$p(x_{t-1} \mid u_1, z_1, \ldots, u_t) dx_{t-1}$$

(Markov) $$= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, u_t) dx_{t-1}$$

(Markov) $$= \eta\ p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) p(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1}) dx_{t-1}$$

The Bayes Filter

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

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

Bayes Filter Variants

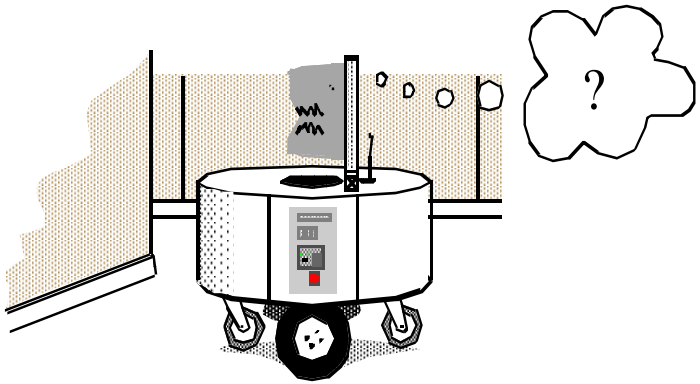
The Bayes filter principle is used in:

- Kalman filters
- Particle filters (Monte-Carlo-Localization)
- Hidden Markov models
- Dynamic Bayesian networks
- Partially Observable Markov Decision Processes (POMDPs)

LOCALISATION

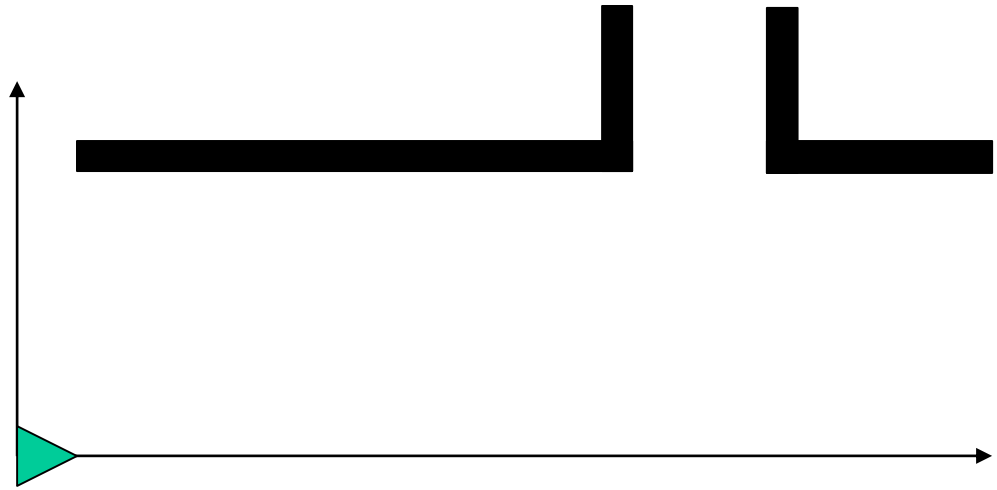
The localisation problem

- Consider a mobile robot moving in a known environment.
- As it starts to move, say from a precisely known location, it might keep track of its location using odometry.
- However, after a certain movement the robot will get very uncertain about its position => update using an observation of its environment
- Observation leads also to an estimate of the robot position which can then be fused with the odometric estimation to get the best possible update of the robots actual position.



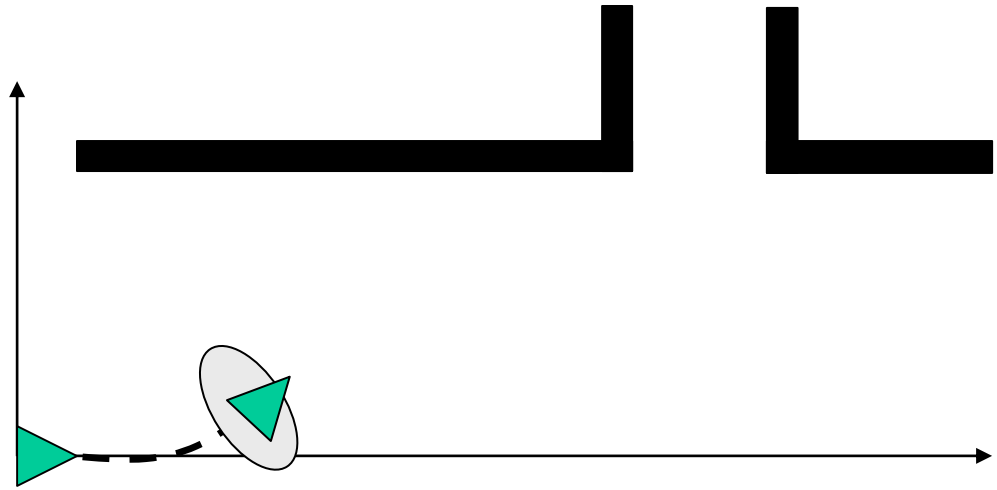
Description of the probabilistic localization problem

- Consider a mobile robot moving in a known environment.



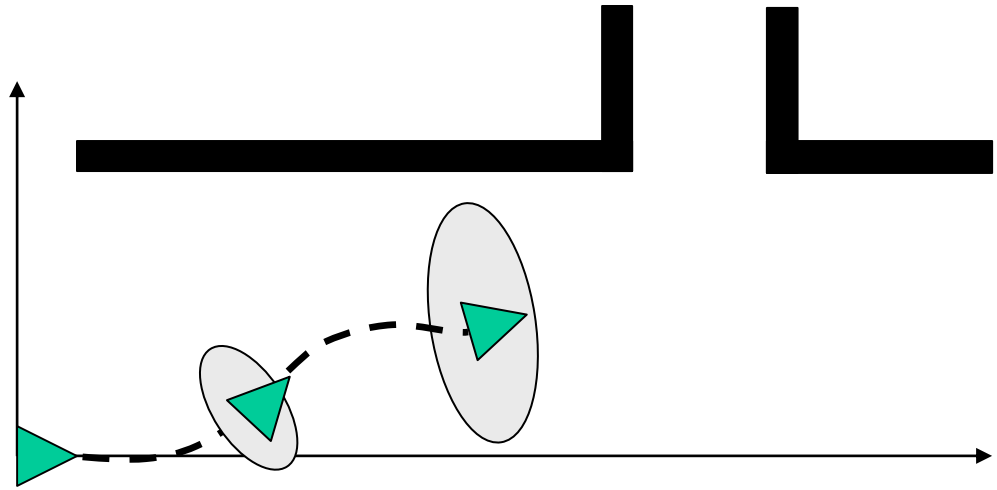
Description of the probabilistic localization problem

- Consider a mobile robot moving in a known environment.
- As it starts to move, say from a precisely known location, it can keep track of its motion using odometry.



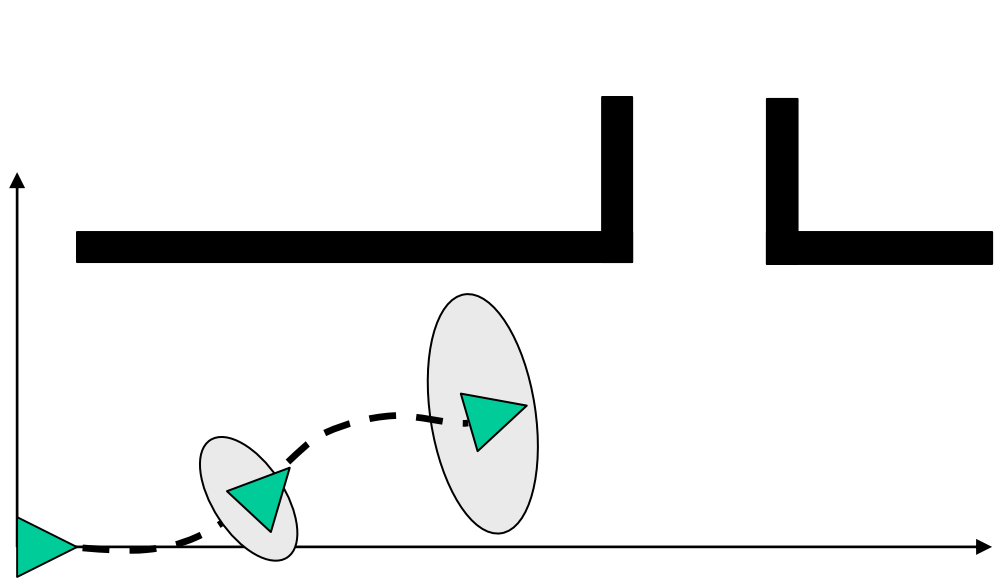
Description of the probabilistic localization problem

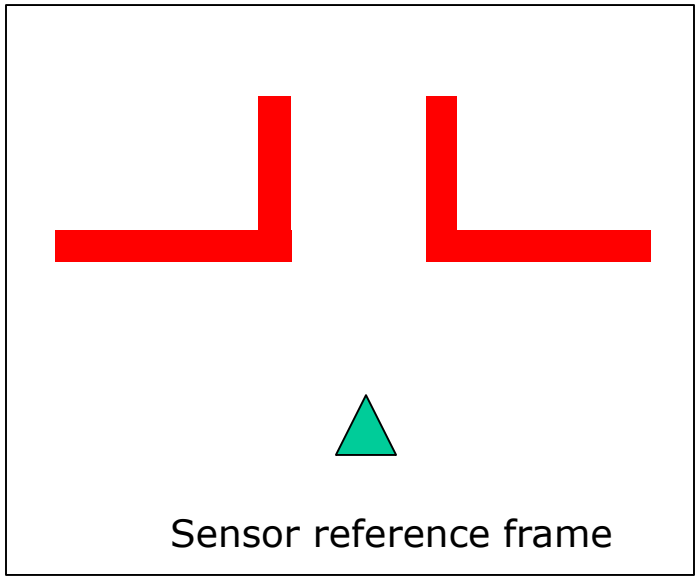
- Consider a mobile robot moving in a known environment.
- As it starts to move, say from a precisely known location, it can keep track of its motion using odometry.
- Due to odometry uncertainty, after some movement the robot will become very uncertain about its position.



Description of the probabilistic localization problem

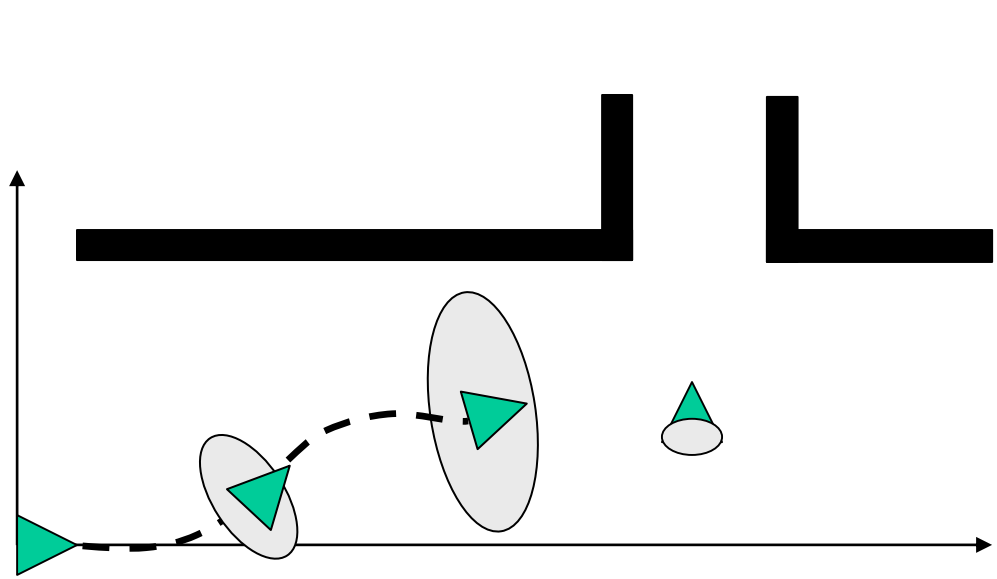
- Consider a mobile robot moving in a known environment.
- As it starts to move, say from a precisely known location, it can keep track of its motion using odometry.
- Due to odometry uncertainty, after some movement the robot will become very uncertain about its position.
- To keep position uncertainty from growing unbounded, the robot must localize itself in relation to its environment map. To localize, the robot might use its on-board exteroceptive sensors (e.g. ultrasonic, laser, vision sensors) to make observations of its environment

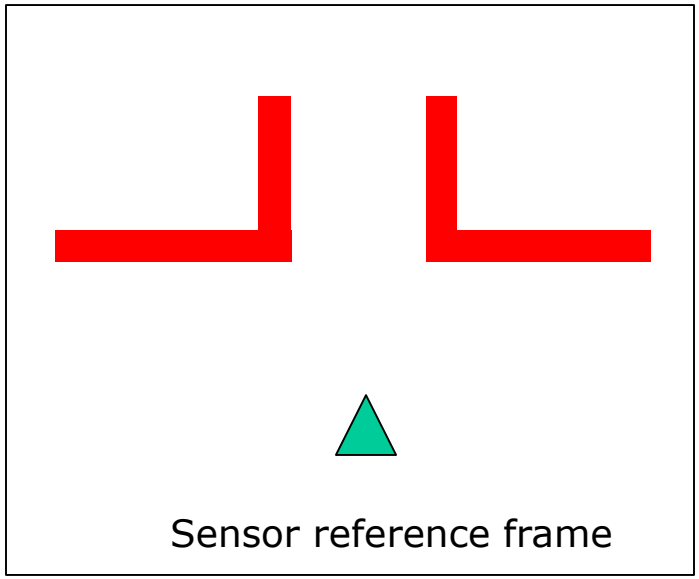




Description of the probabilistic localization problem

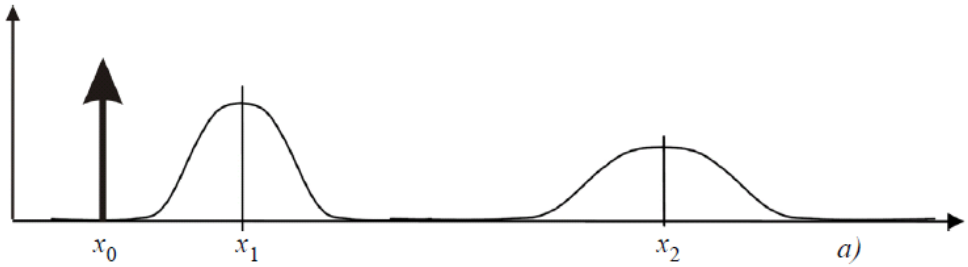
- Consider a mobile robot moving in a known environment.
- As it starts to move, say from a precisely known location, it can keep track of its motion using odometry.
- Due to odometry uncertainty, after some movement the robot will become very uncertain about its position.
- To keep position uncertainty from growing unbounded, the robot must localize itself in relation to its environment map. To localize, the robot might use its on-board exteroceptive sensors (e.g. ultrasonic, laser, vision sensors) to make observations of its environment
- The robot updates its position based on the observation. Its uncertainty shrinks.



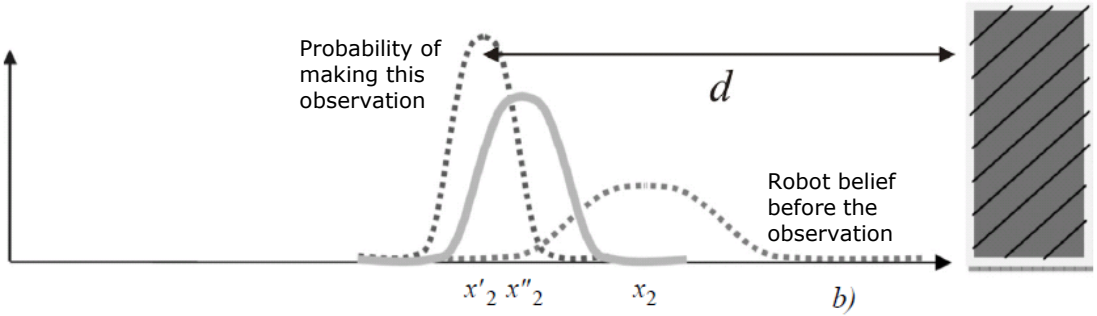


Action and perception updates

- In robot localization, we distinguish two update steps:
  1. ACTION update:
     - the robot moves and estimates its position through its **proprioceptive** sensors.
       During this step, the robot uncertainty grows.



  2. PERCEPTION update:
     - the robot makes an observation using its **exteroceptive** sensors and correct its position by opportunely combining its belief before the observation with the probability of making exactly that observation.
       During this step, the robot uncertainty shrinks.



The solution to the probabilistic localization problem

Solving the probabilistic localization problem
consists in solving separately the Action and Perception updates

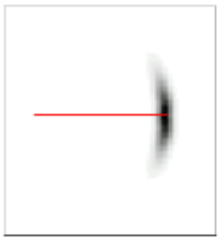
The solution to the probabilistic localization problem

A probabilistic approach to the mobile robot localization problem is a method able to compute the probability distribution of the robot configuration during each Action and Perception step.

The ingredients are:

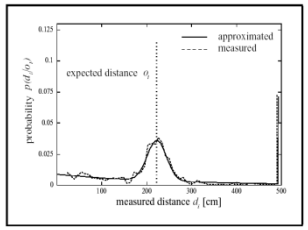
1. The initial probability distribution $p(x)_{t=0}$

2. The statistical error model of the proprioceptive sensors (e.g. wheel encoders)

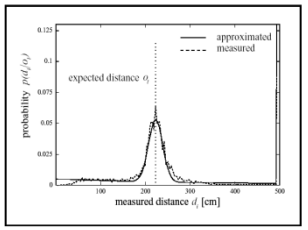


$$\Sigma_{\Delta} = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}$$

3. The statistical error model of the exteroceptive sensors (e.g. laser, sonar, camera)



Ultrasound.



Laser range-finder.

4. Map of the environment
(If the map is not known a priori then the robots needs to build a map of the environment and then localizing in it. This is called SLAM, Simultaneous Localization And Mapping)

The solution to the probabilistic localization problem

How do we solve the Action and Perception updates?

- Action update uses the Theorem of Total probability (or convolution)

$$p(x) = \int_y p(x|y)p(y)dy$$

- Perception update uses the Bayes rule

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Action update

- In this phase the robot estimates its current position $\overline{bel}(x_t)$ based on the knowledge of the previous position $bel(x_{t-1})$ and the odometric input u_t

- Using the Theorem of Total probability, we compute the robot's belief after the motion as

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

which can be written as a *convolution*

$$\overline{bel}(x_t) = p(x_t | u_t, x_{t-1}) * bel(x_{t-1})$$

Perception update

- In this phase the robot corrects its previous position (i.e. its former belief) by opportunely combining it with the information from its exteroceptive sensors

$$p(x_t \mid z_t) = \frac{p(z_t \mid x_t)\, p(x_t)}{p(z_t)}$$

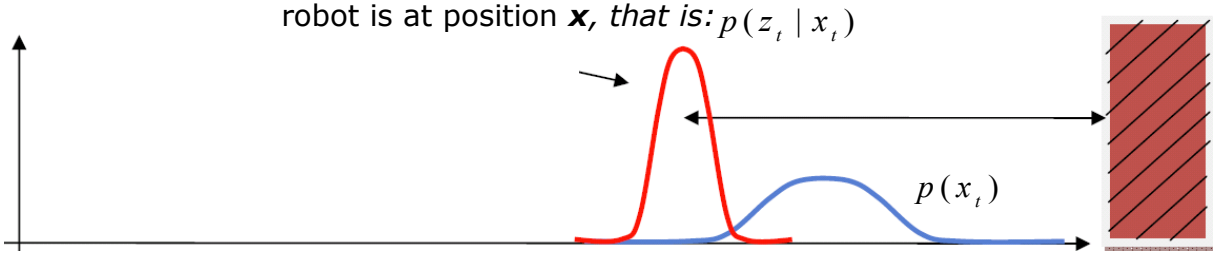
$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\, \overline{bel}(x_t)$$

where η is a normalization factor which makes the probability integrate to 1.

Perception update

- Explanation of $bel(x_t) = \eta \cdot p(z_t \mid x_t) \overline{bel}(x_t)$

This is the probability of observing ***z*** *given that the* robot is at position ***x***, *that is:* $p(z_t \mid x_t)$



$$p(x_t \mid z_t) = \frac{p(z_t \mid x_t)\, p(x_t)}{p(z_t)}$$

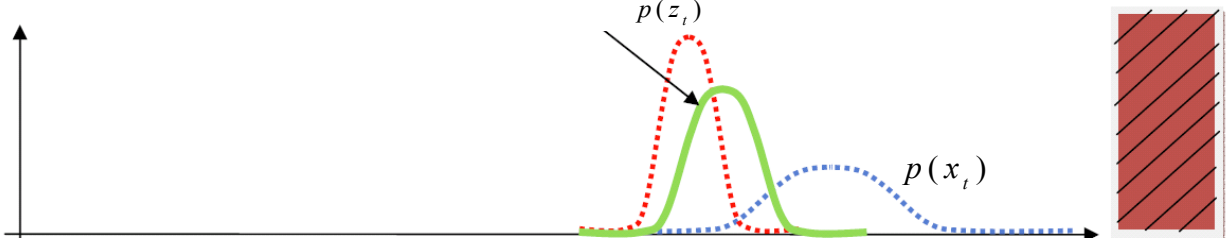


Illustration of probabilistic bap based localization

Initial probability distribution $p(x)_{t=0}$

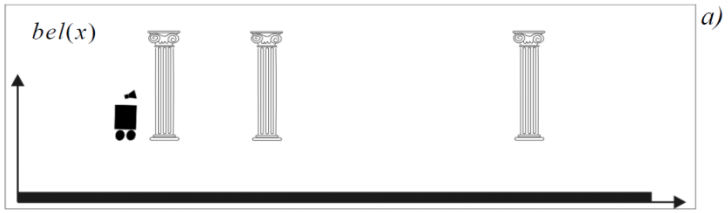


Illustration of probabilistic bap based localization

Initial probability distribution

$$p(x)_{t=0}$$

$$p(z_t \mid x_t)$$

Perception update

$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\overline{bel}(x_t)$$

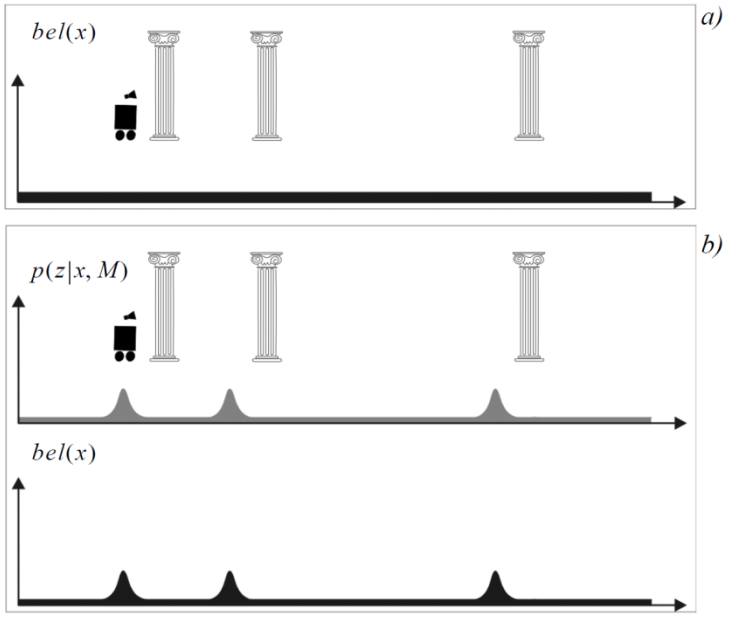
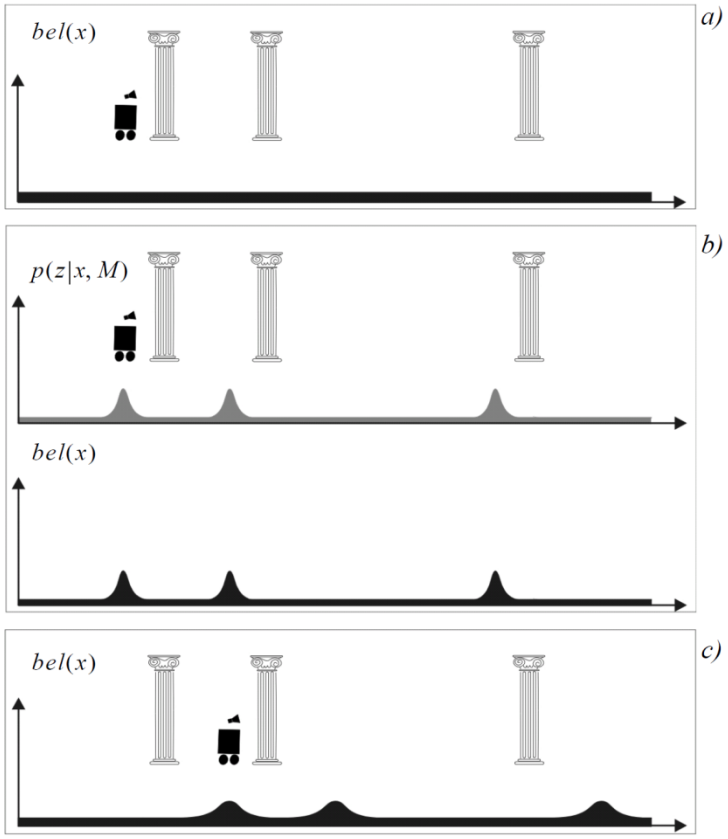


Illustration of probabilistic bap based localization

Initial probability distribution

$$p(x)_{t=0}$$



$$p(z_t \mid x_t)$$

Perception update

$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\overline{bel}(x_t)$$

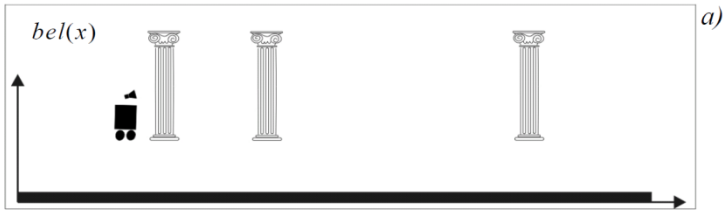
Action update

$$\overline{bel}(x_t) = p(x_t|u_t, x_{t-1})*bel(x_{t-1})$$

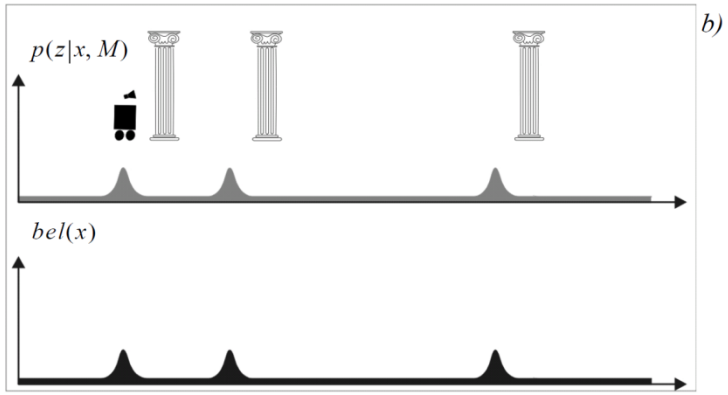
Illustration of probabilistic bap based localization

Initial probability distribution

$$p(x)_{t=0}$$



$$p(z_t \mid x_t)$$

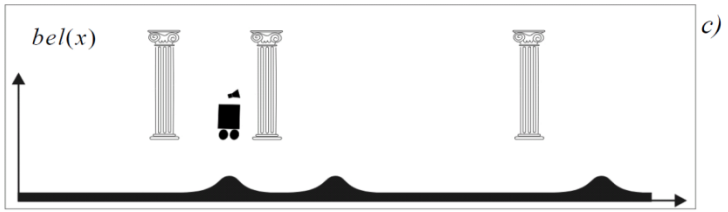


Perception update

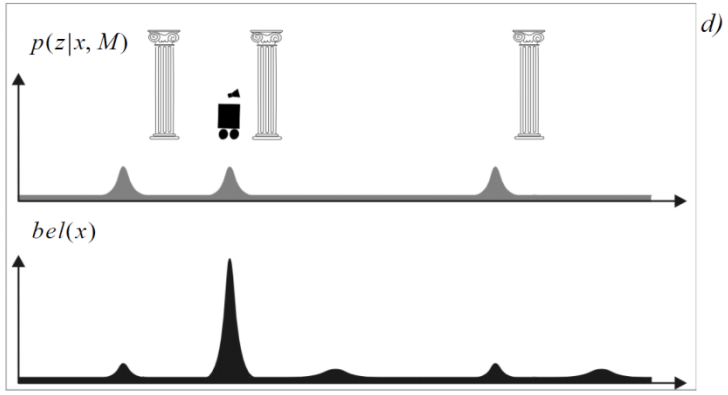
$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\overline{bel}(x_t)$$

Action update

$$\overline{bel}(x_t) = p(x_t \mid u_t, x_{t-1}) * bel(x_{t-1})$$



$$p(z_t \mid x_t)$$



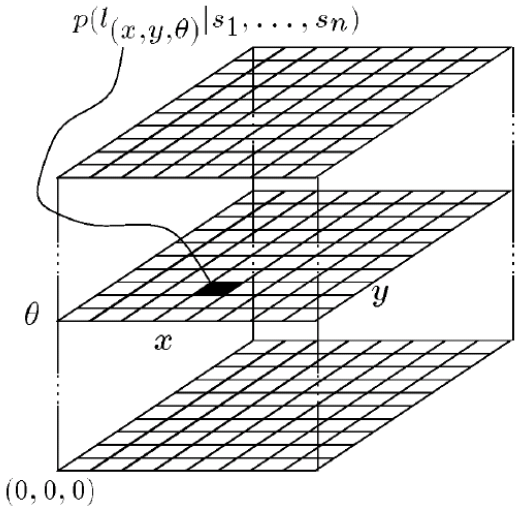
Perception update

$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\overline{bel}(x_t)$$

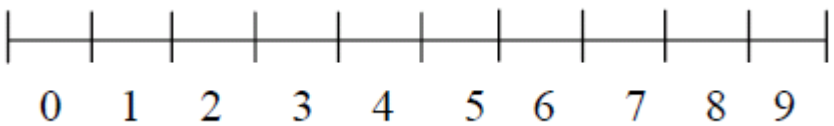
MARKOV LOCALISATION

Markov localization

- Markov localization uses a grid space representation of the robot configuration.

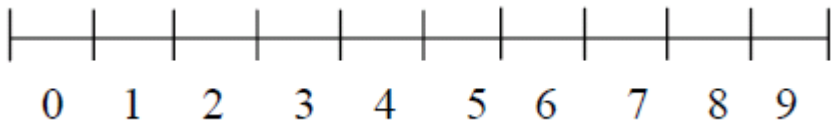


- For sake of simplicity let us consider a robot moving in a one dimensional environment (e.g. moving on a rail). Therefore the robot configuration space can be represented through the sole variable x.
- Let us discretize the configuration space into 10 cells

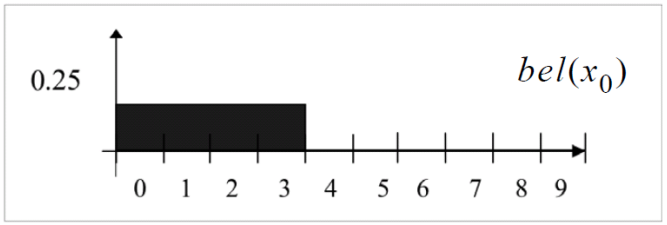


Markov localization

- Let us discretize the configuration space into 10 cells



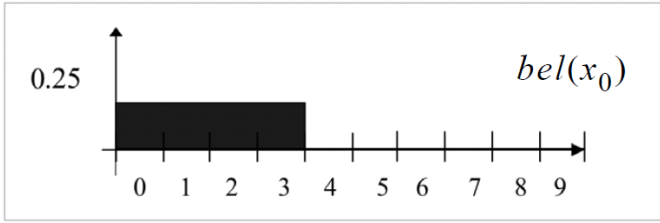
- Suppose that the robot's initial belief is a uniform distribution from 0 to 3. Observe that all the elements were normalized so that their sum is 1.



a)

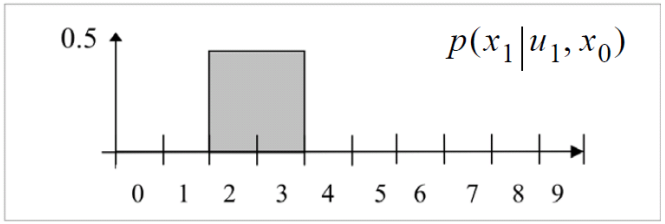
Markov localization

- Initial belief distribution



a)

- **Action phase:**
  Let us assume that the robot moves forward with the following statistical model



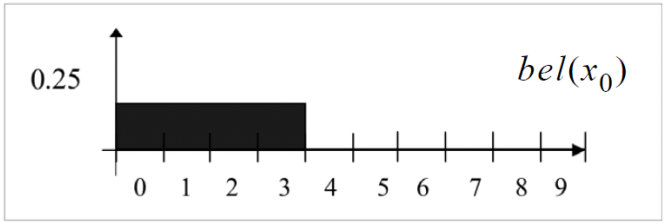
b)

- This means that we have 50% probability that the robot moved 2 or 3 cells forward.
- Considering what the probability was before moving, what will the probability be after the motion?

Markov localization: Action update

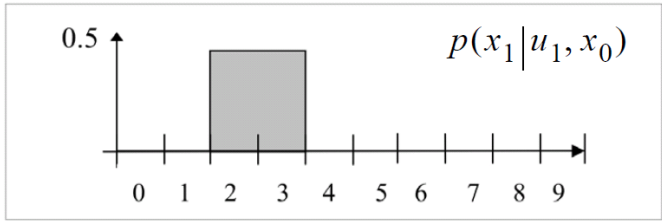
- The solution is given by the convolution of the two distributions

$$\overline{bel}(x_1) = p(x_1|u_1, x_0) * bel(x_0) = \sum_{x_0=0}^{3} p(x_1|u_1, x_0) bel(x_0)$$



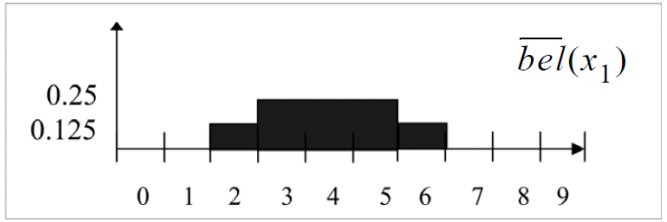
a)

*



b)

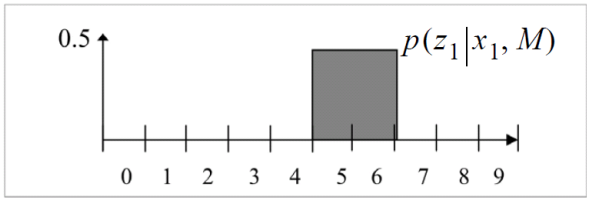
=



c)

Markov localization: Perception update

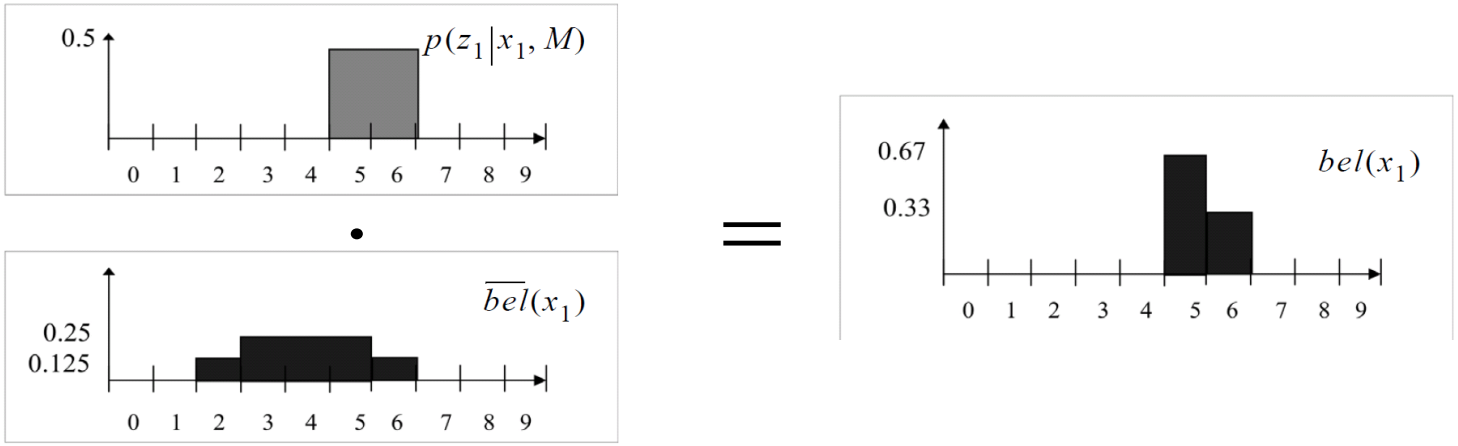
- Let us now assume that the robot uses its onboard range finder and measures the distance from the origin. Assume that the statistical error model of the sensors is



This plot tells us that the distance of the robot from the origin can be equally 5 or 6 units.

- What will the final robot belief be after this measurement? The answer is again given by the Bayes rule:

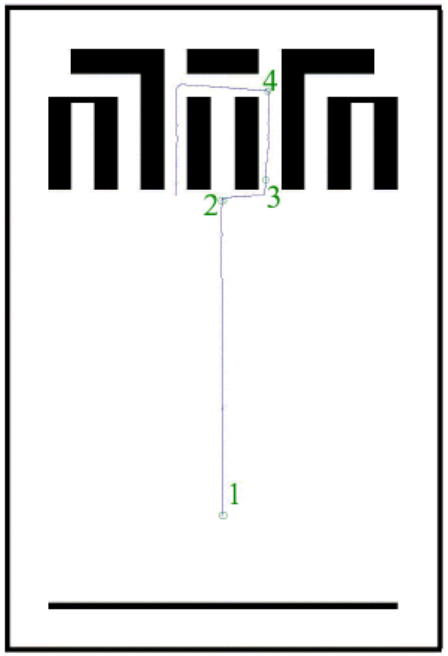
$$bel(x_t) = \eta \cdot p(z_t \mid x_t)\overline{bel}(x_t)$$



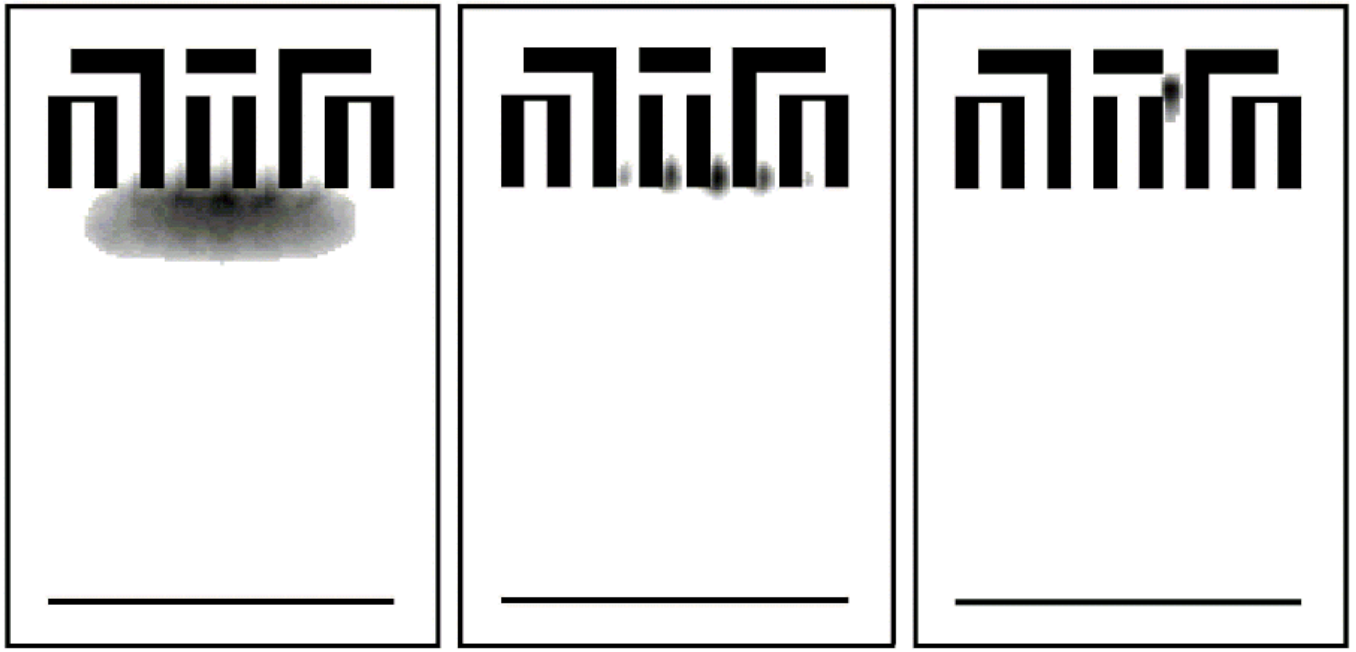
Grid-based Representation - Multi Hypothesis

- Grid size around 20 cm^2.

Courtesy of W. Burgard



Path of the robot

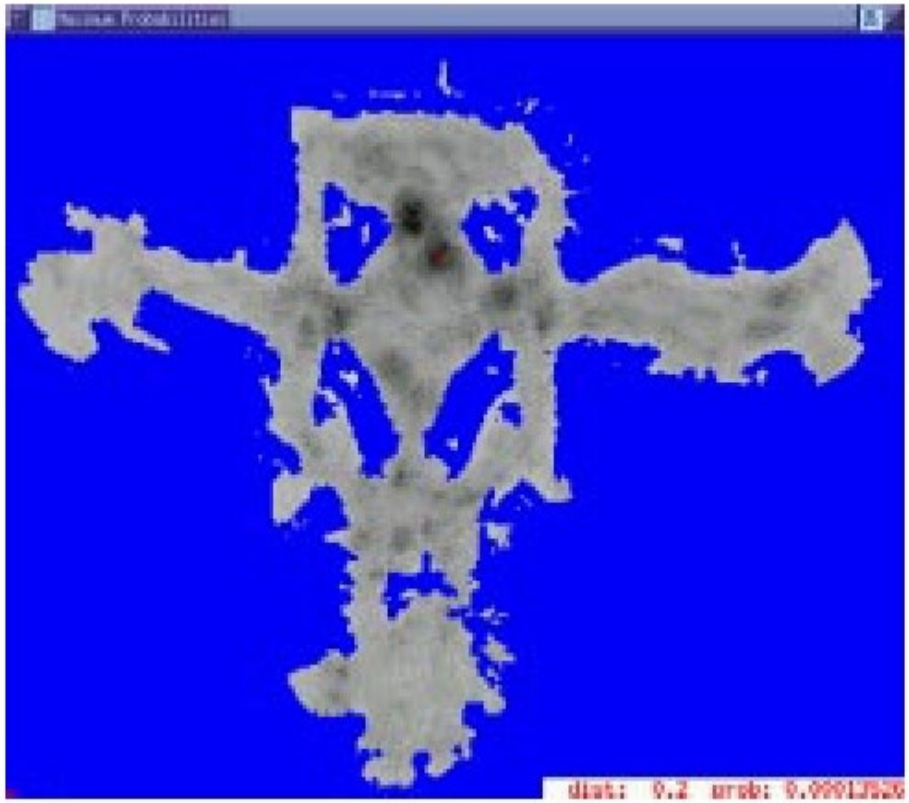


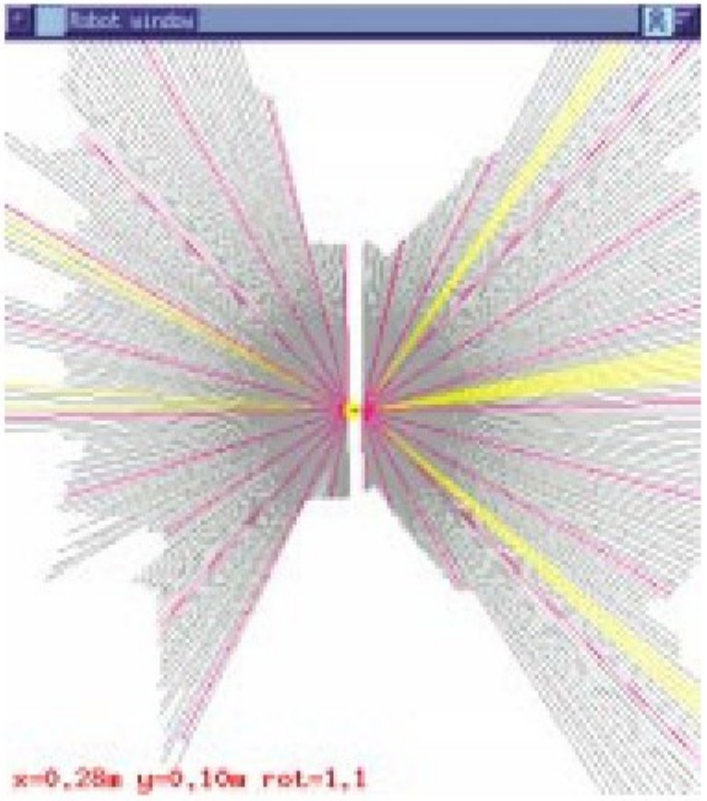
Belief states at positions 2, 3 and 4

Markov Localization: Case Study 2 – Grid Map (5)

- Example 2: Museum
  - Laser scan 1

Courtesy of W. Burgard

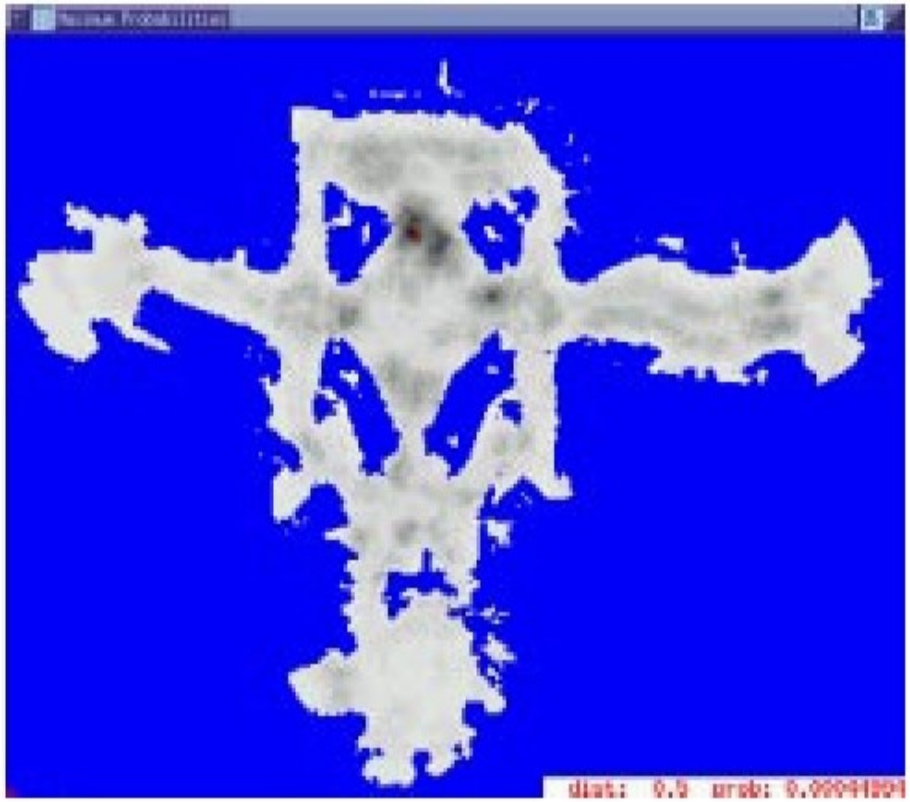


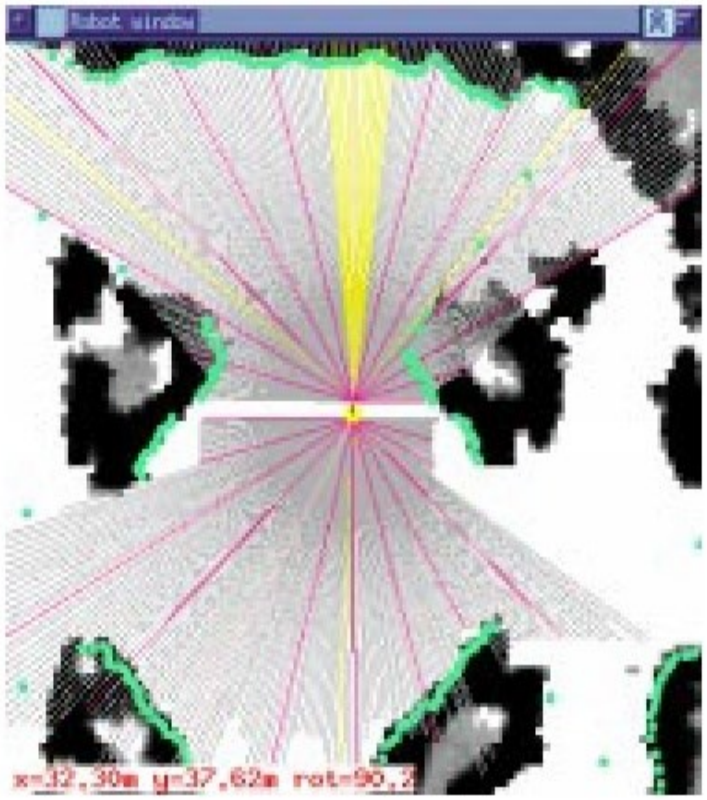


Markov Localization: Case Study 2 – Grid Map (6)

- Example 2: Museum
  - Laser scan 2

Courtesy of W. Burgard

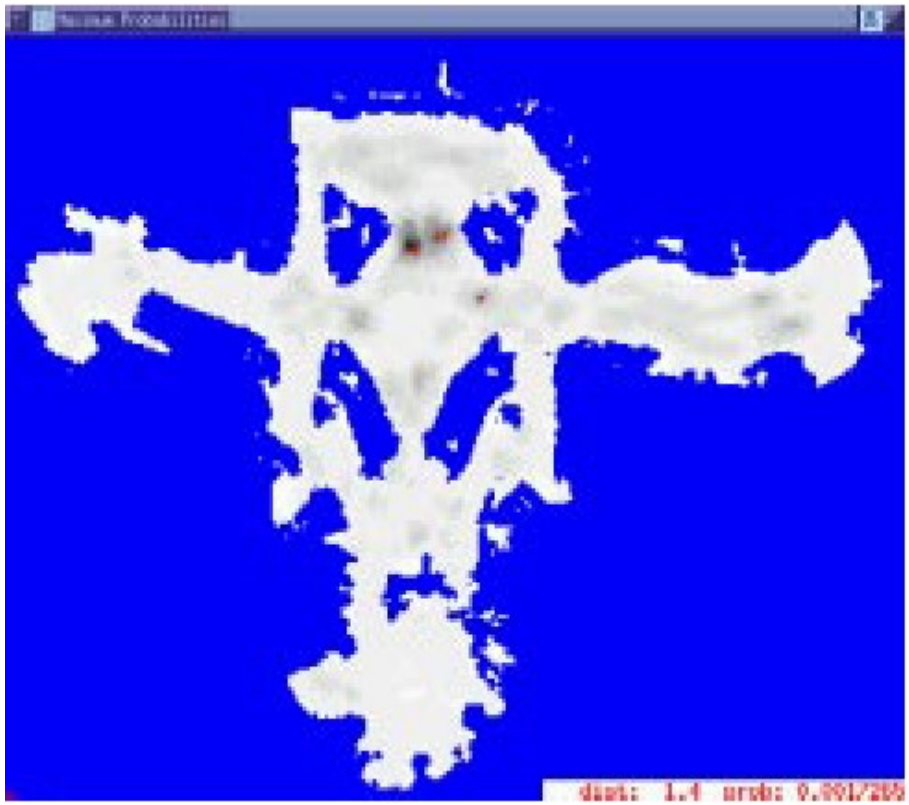


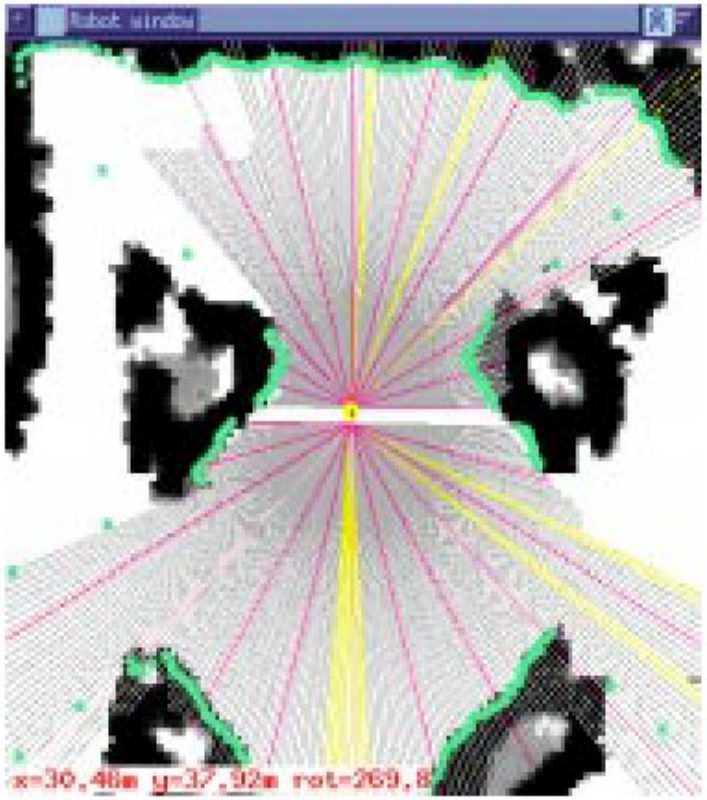


Markov Localization: Case Study 2 – Grid Map (7)

- Example 2: Museum
  - Laser scan 3

Courtesy of W. Burgard

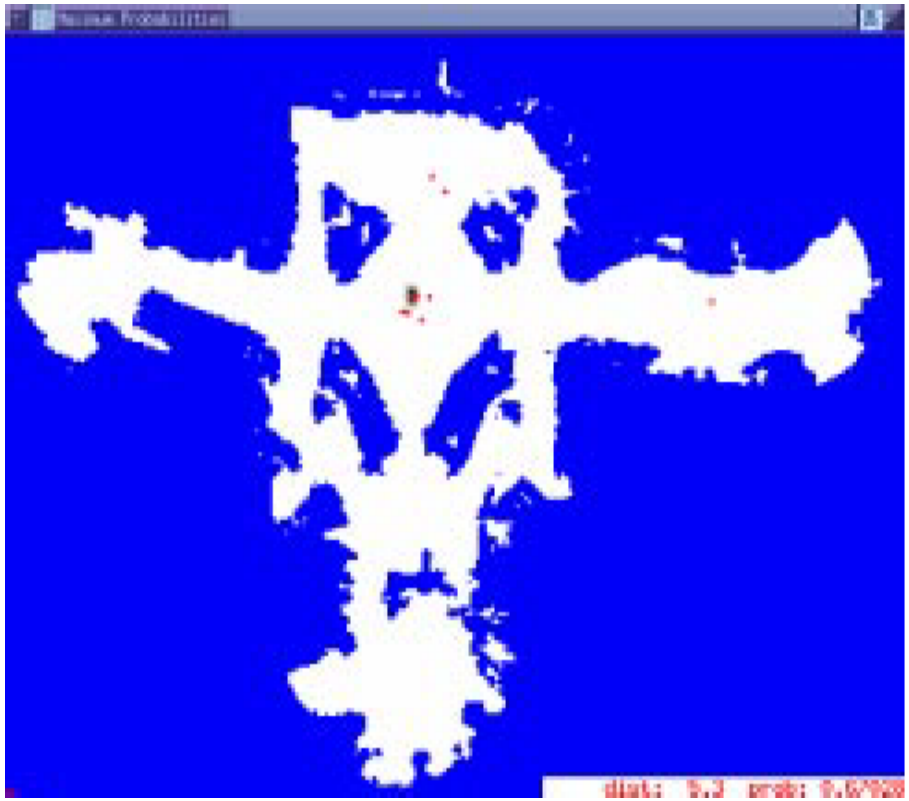


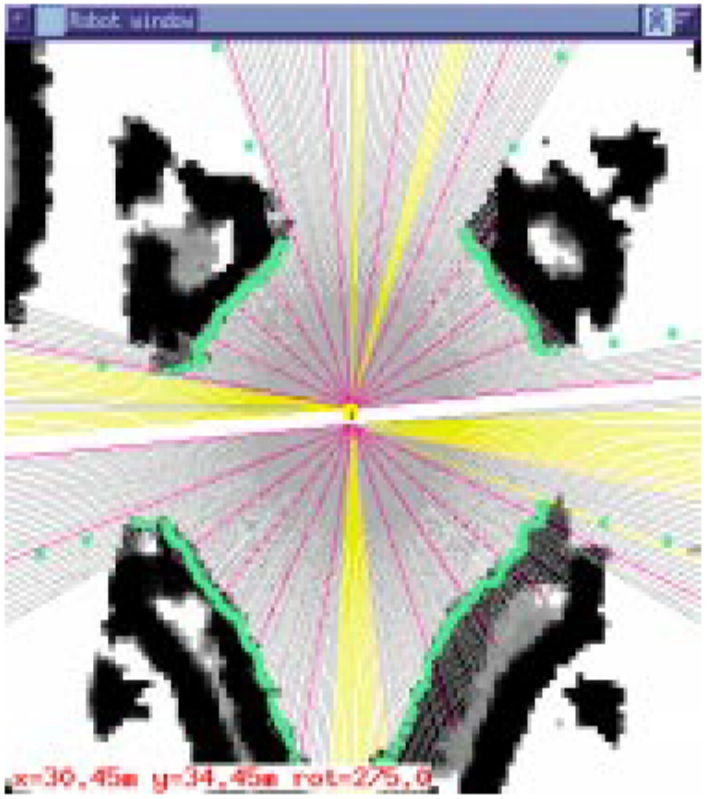


Markov Localization: Case Study 2 – Grid Map (8)

- Example 2: Museum
  - Laser scan 13

Courtesy of W. Burgard

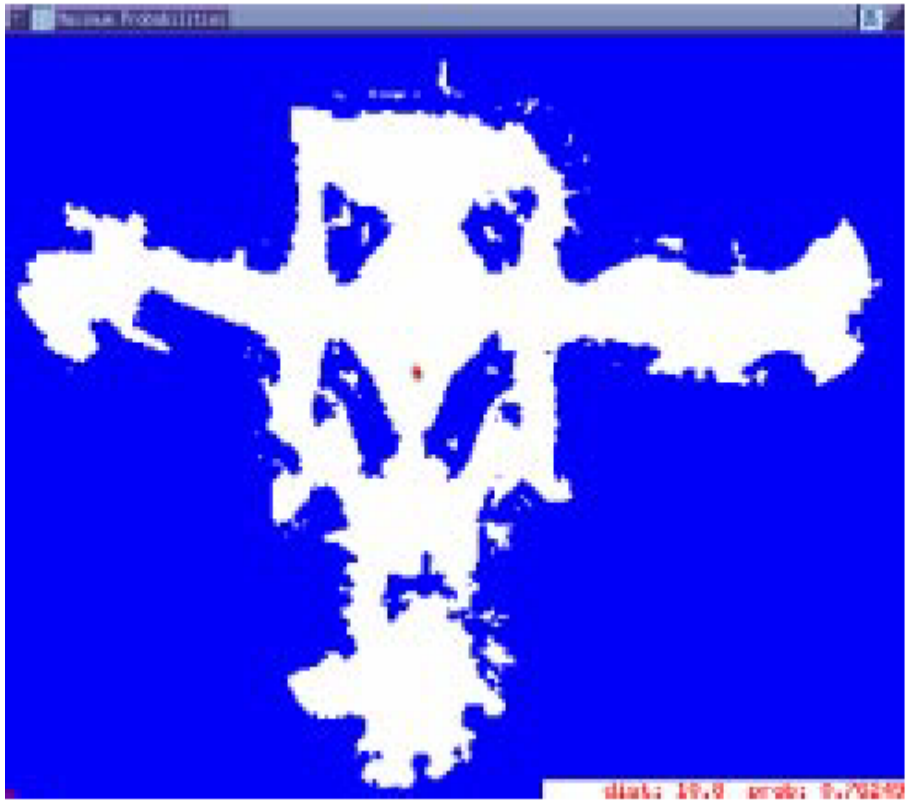


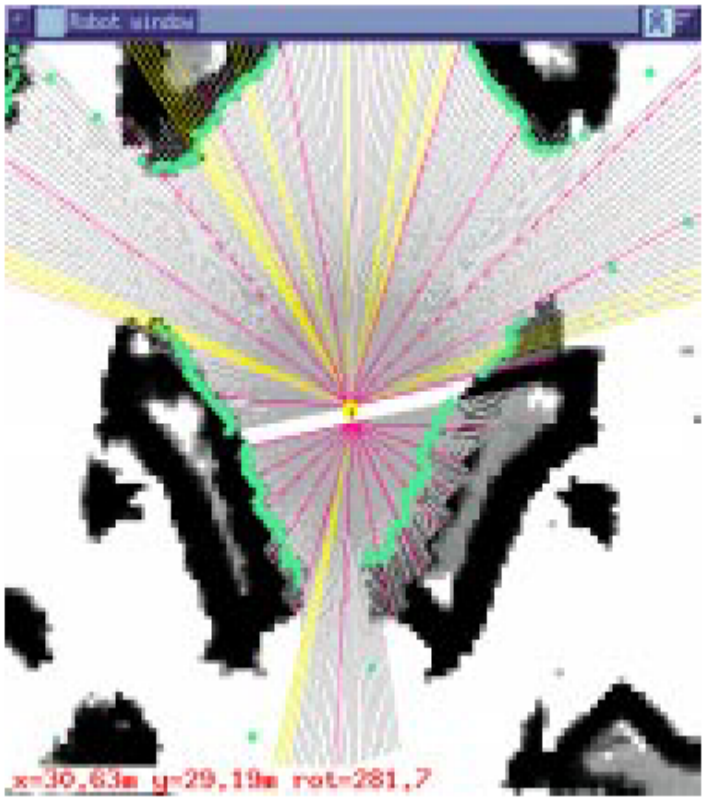


Markov Localization: Case Study 2 – Grid Map (9)

- Example 2: Museum
  - Laser scan 21

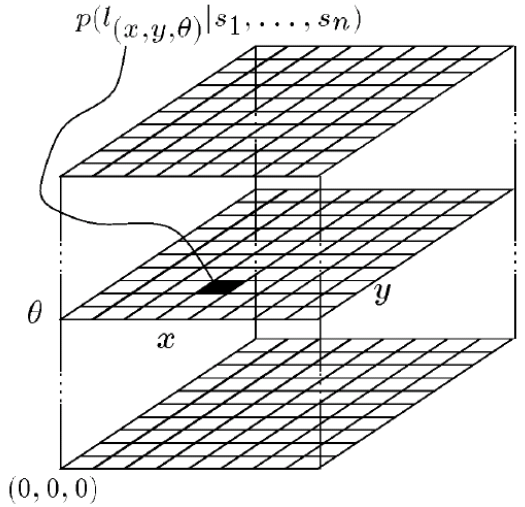
Courtesy of W. Burgard





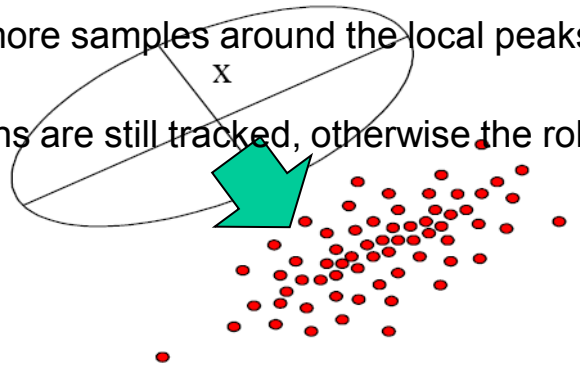
Drawbacks of Markov localization

- In the more general planar motion case the grid-map is a three-dimensional array where each cell contains the probability of the robot to be in that cell. In this case, the cell size must be chosen carefully.
- During each prediction and measurement steps, all the cells are updated. If the number of cells in the map is too large, the computation can become too heavy for real-time operations.
- The convolution in a 3D space is clearly the computationally most expensive step.
- As an example, consider a 30x30 m environment and a cell size of 0.1 m x 0.1 m x 1 deg. In this case, the number of cells which need to be updated at each step would be 30x30x100x360=32.4 million cells!
  Fine fixed decomposition grids result in a huge state space
  - Very important processing power needed
  - Large memory requirement



Drawbacks of Markov localization

- **Reducing complexity**
  - Various approached have been proposed for reducing complexity
    - One possible solution would be to increase the cell size at the expense of localization accuracy.
    - Another solution is to use an adaptive cell decomposition instead of a fixed cell decomposition.
- **Unimodal distribution / Kalman Filter**
  - Use a single Normal distribution to represent the distribution
  - Assumes measurement and action process are also normally distributed
  - Closed form solution.
- **Randomized Sampling / Particle Filter**
  - The main goal is to reduce the number of states that are updated in each step
  - Approximated belief state by representing only a 'representative' subset of all states (possible locations)
  - E.g update only 10% of all possible locations
  - The sampling process is typically weighted, e.g. put more samples around the local peaks in the probability density function
  - However, you have to ensure some less likely locations are still tracked, otherwise the robot might get lost

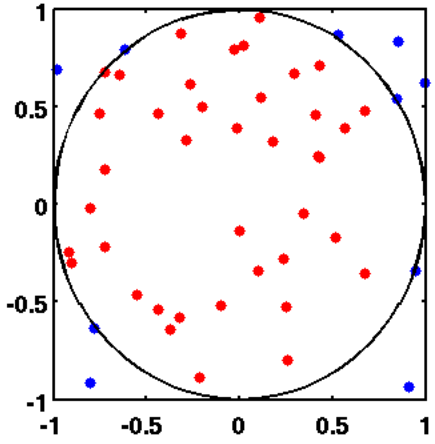


probability distribution (ellipse) as particle set (red dots)

PARTICLE FILTER FOR LOCALISATION

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

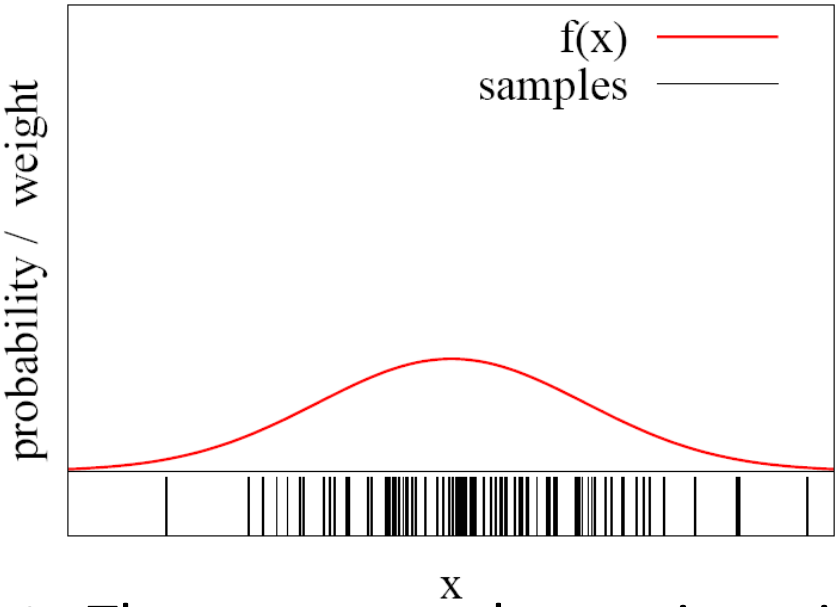


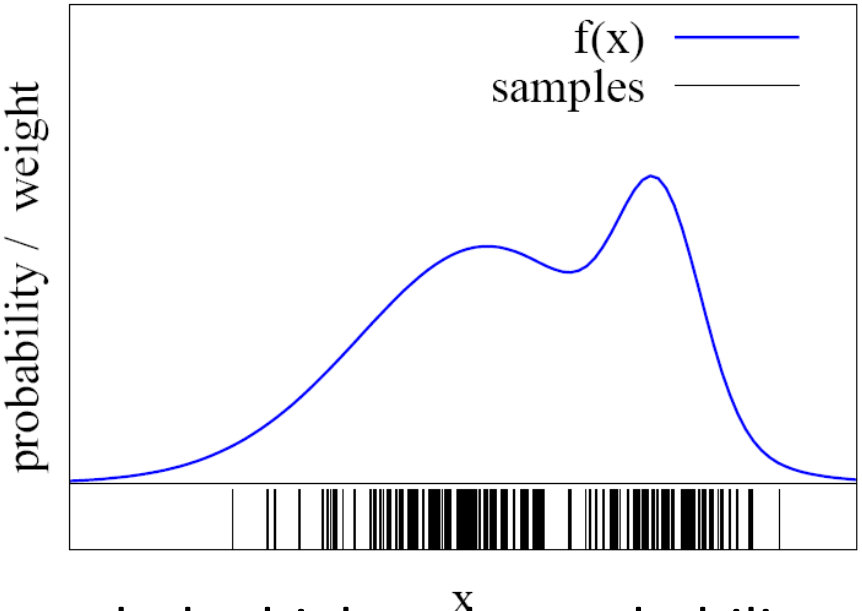
Non-Parametric Representation

- Probability distributions (e.g. a robot‘s belief) can be represented:
  - 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?

Inversion sampling

Rejection sampling

Importance sampling

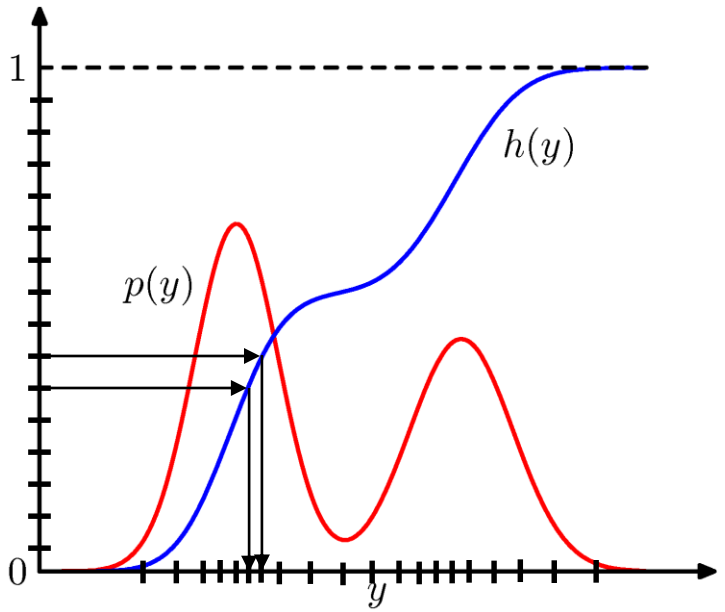
SAMPLING FROM A DISTRIBUTION

Sampling from a Distribution

- There are several approaches:
  - Probability transformation
    - Uses inverse of the c.d.f h
  - Rejection Sampling
  - Importance Sampling
  - …
- Probability transformation:
  - Sample uniformly in [0,1]
  - Transform using h-1
- But:
  - Needs calculation of h and its inverse

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

"Cumulative distribution Function"



Rejection Sampling

Simplification:

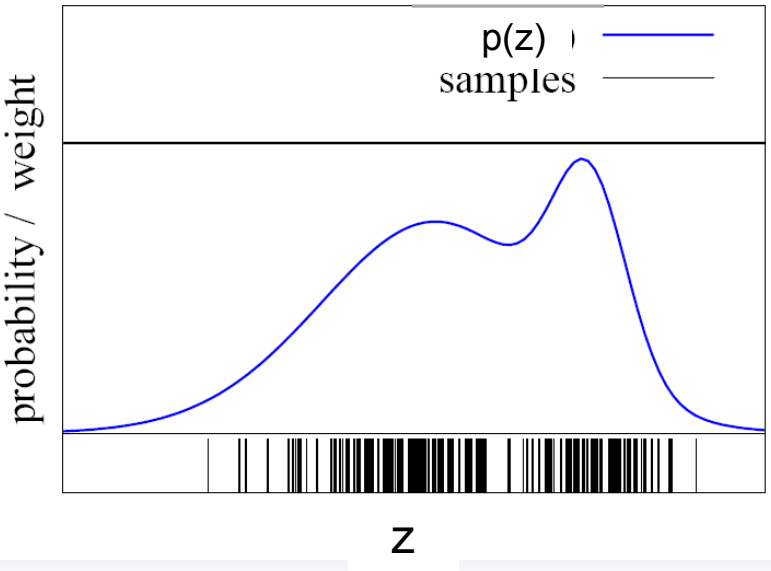
- Assume $p(z) < 1$ for all z
- Sample z uniformly
- Sample c from $[0, 1]$

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

  keep the sample

  otherwise:

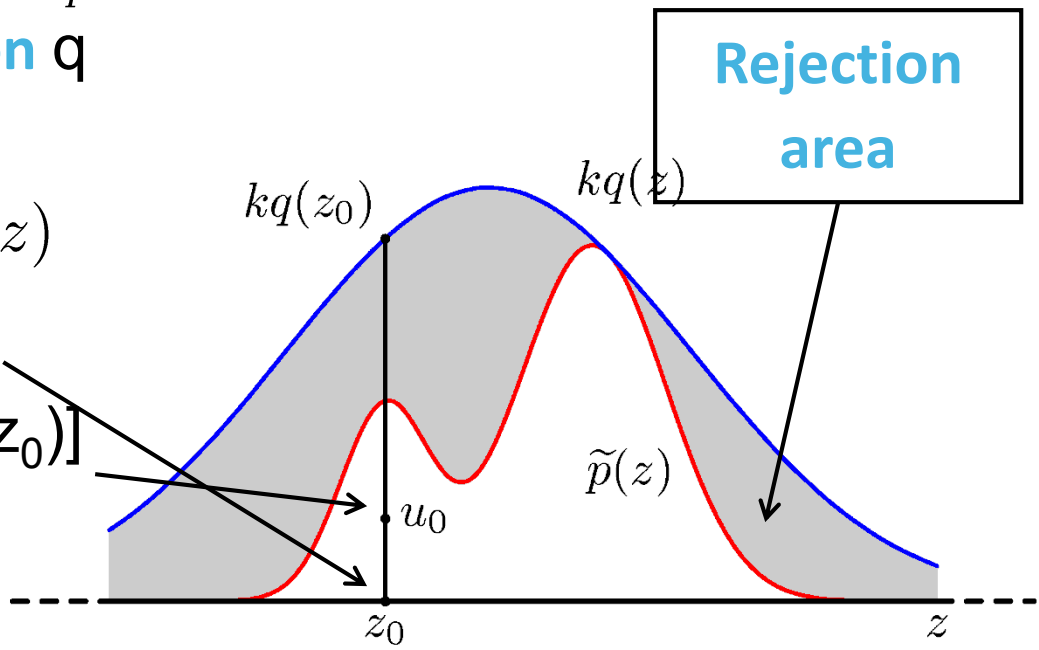
  reject the sample



Rejection Sampling

2. General case: $p(z) = \frac{1}{Z_p}\tilde{p}(z)$ (unnormalized)

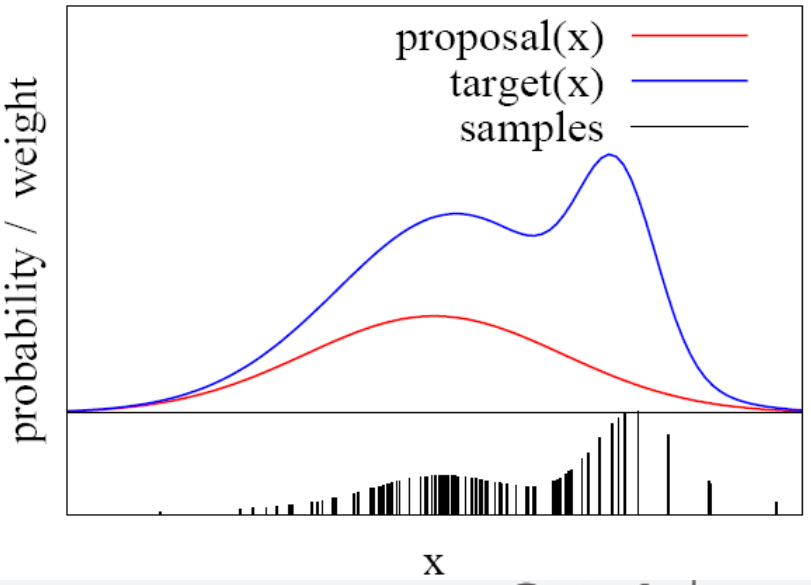
- Find **proposal distribution** q
  - Easy to sample from q
- Find k with $kq(z) \geq \tilde{p}(z)$
- Sample uniformly from q
- Sample unif. from [0,kq(z_0)]
- Reject if $u_0 > \tilde{p}(z_0)$



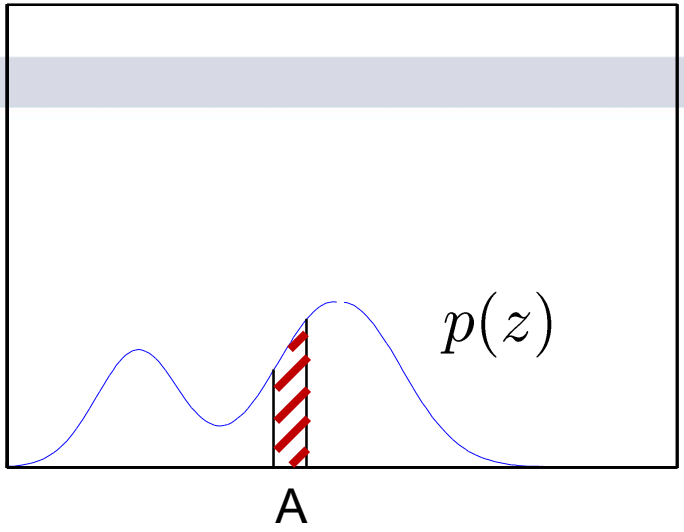
But: Rejection sampling is inefficient.

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** $I(x \in A)$

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

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

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

Approximation with samples drawn from q:

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

An alternative to EKF for non-linear functions,

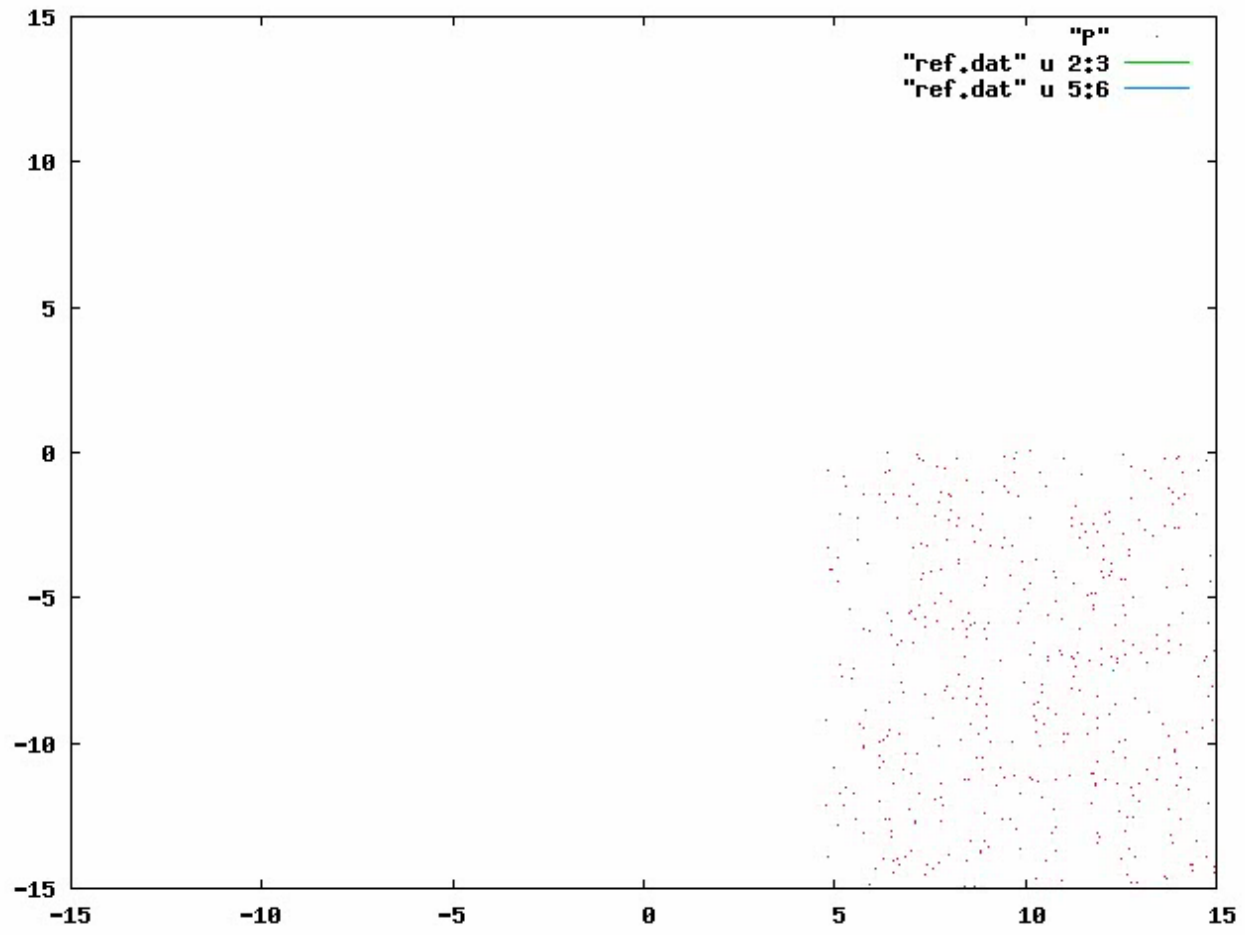
Non-gaussian noise

APPLICATION: THE PARTICLE FILTER

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

Example



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** $m = 1$ **do** M
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** $m = 1$ **do** M
   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

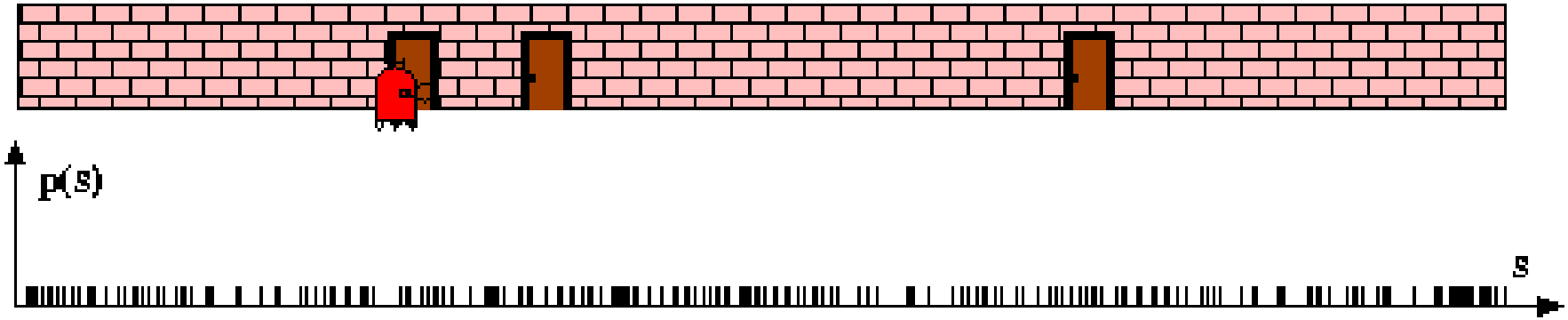
APPLICATION TO LOCALISATION

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:

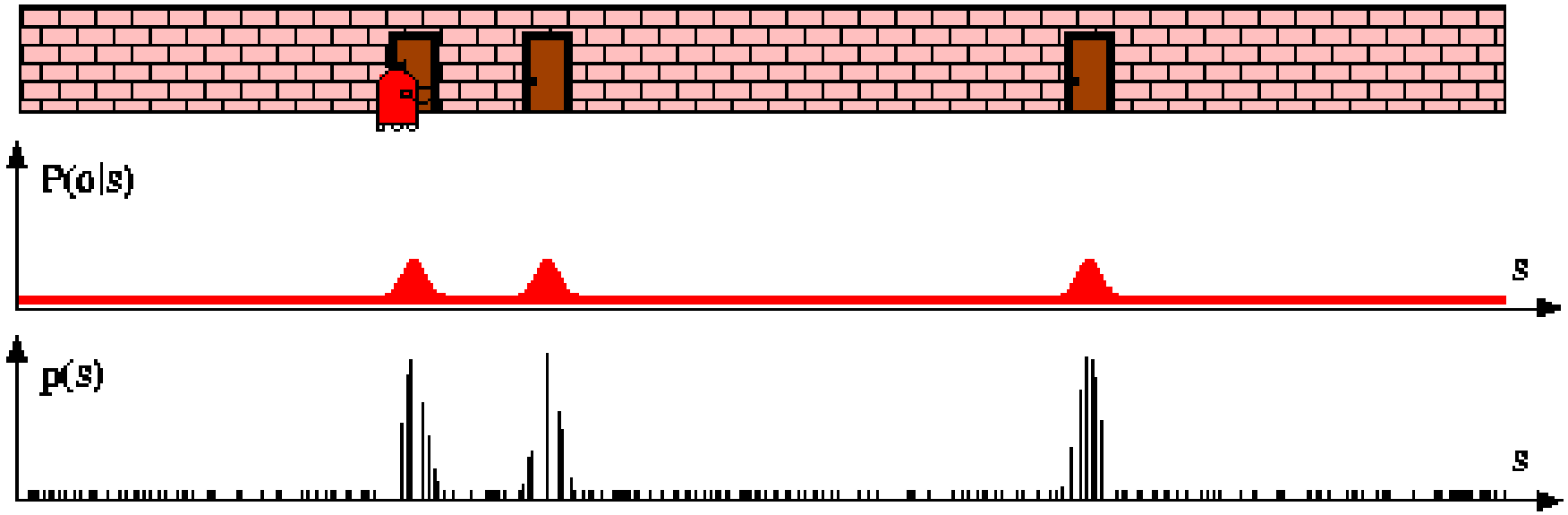
Monte-Carlo Localization

Back to Our Example …



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

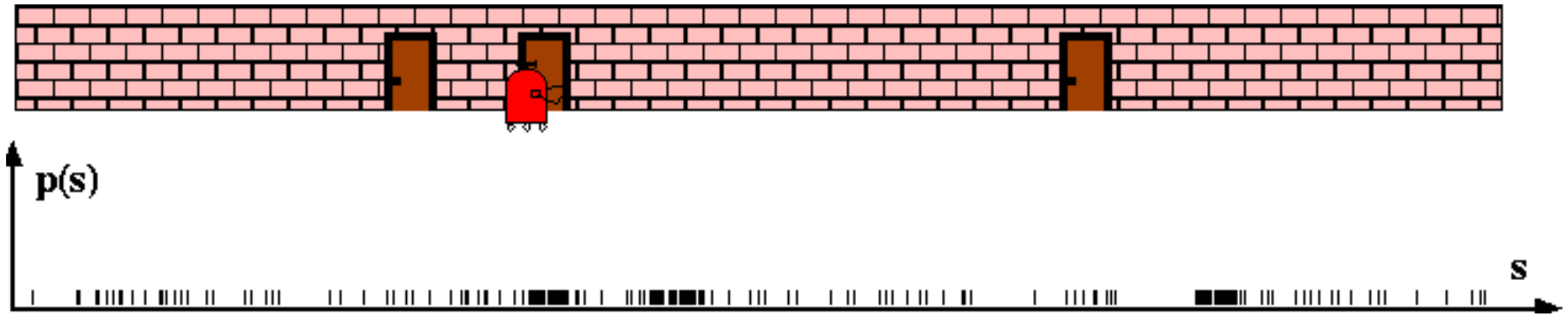
Sensor Information



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

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

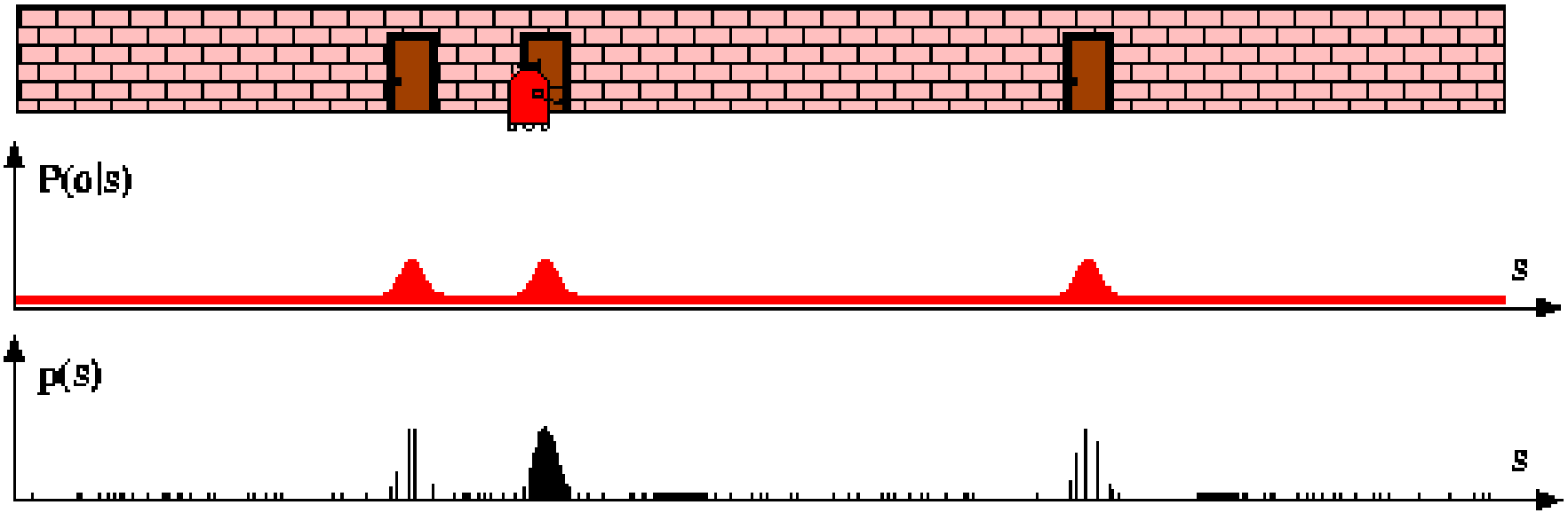
Robot Motion



After resampling and applying the motion model the particles are distributed more densely at three locations.

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

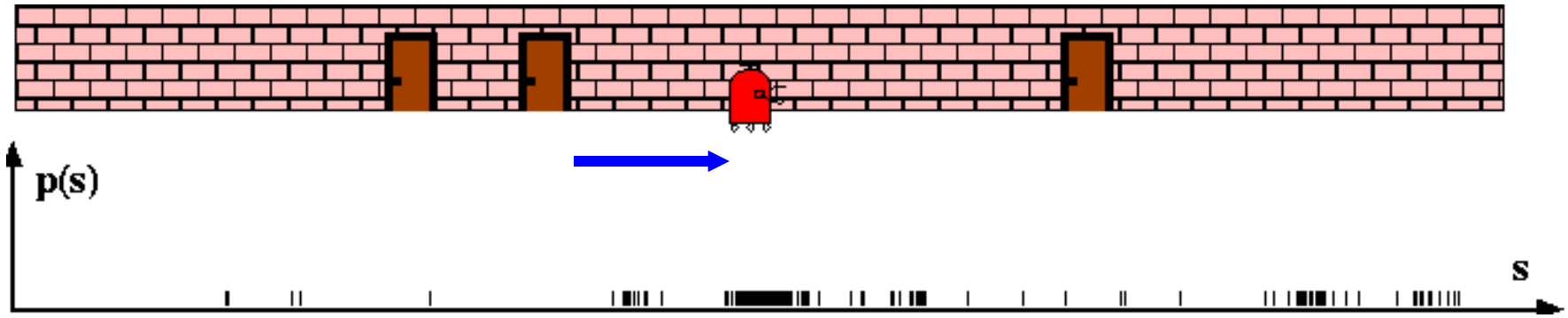
Sensor Information



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

1. $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2. **for** $m = 1$ **do** M
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** $m = 1$ **do** M
   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

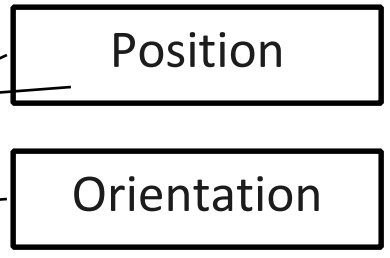
Resampling

Sampling from Proposal

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

- This can be done in the following ways:
  - Adding the motion vector to each particle directly (this assumes perfect motion)
  - Sampling from the motion model $p(x_t \mid u_t, x_{t-1}^{[m]})$, e.g. for a 2D motion with translation velocity v and rotation velocity w we have:

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



Sampling from Motion Model

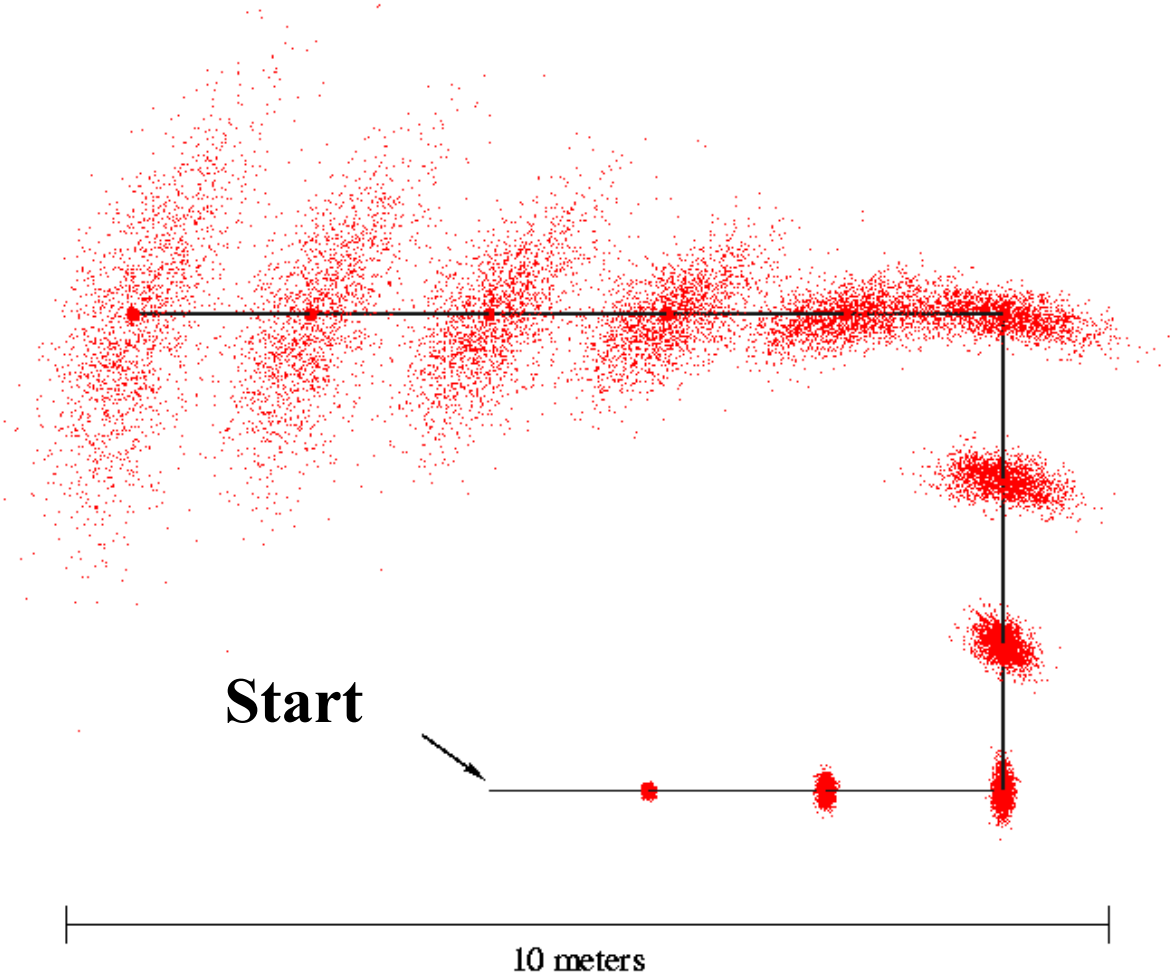
Algorithm sample_motion_model_vel$(\mathbf{u}_t, \mathbf{x}_{t-1})$:

1. $\hat{v} \leftarrow v_t + \text{sample}(\alpha_1|v_t| + \alpha_2|w_t|)$
2. $\hat{w} \leftarrow w_t + \text{sample}(\alpha_3|v_t| + \alpha_4|w_t|)$
3. $\hat{\gamma} \leftarrow \text{sample}(\alpha_5|v_t| + \alpha_6|w_t|)$
4. $x_t \leftarrow x_{t-1} - \frac{\hat{v}}{\hat{w}} \sin\theta_{t-1} + \frac{\hat{v}}{\hat{w}} \sin(\theta_{t-1} + \hat{w}\Delta t)$
   $y_t \leftarrow y_{t-1} + \frac{\hat{v}}{\hat{w}} \cos\theta_{t-1} - \frac{\hat{v}}{\hat{w}} \cos(\theta_{t-1} + \hat{w}\Delta t)$
5. $\theta_t \leftarrow \theta_{t-1} + \hat{w}\Delta t + \hat{\gamma}\Delta t$
6. $\mathbf{x}_t = (x_t, y_t, \theta_t)^T$
7. return

$\alpha_1, \ldots, \alpha_6$: robot specific parameters

$\text{sample}(\sigma^2)$: zero-mean, Gaussian sampling with variance σ^2

Motion Model Sampling (Example)



Computation of Importance Weights

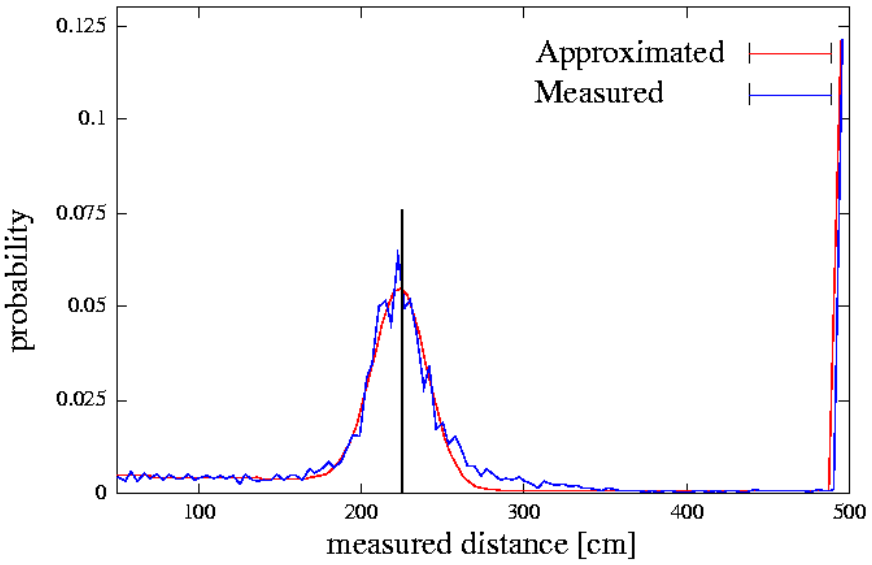
$$w_t^{[m]} \leftarrow p(z_t \mid x_t^{[m]})$$

- 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: we need importance sampling.

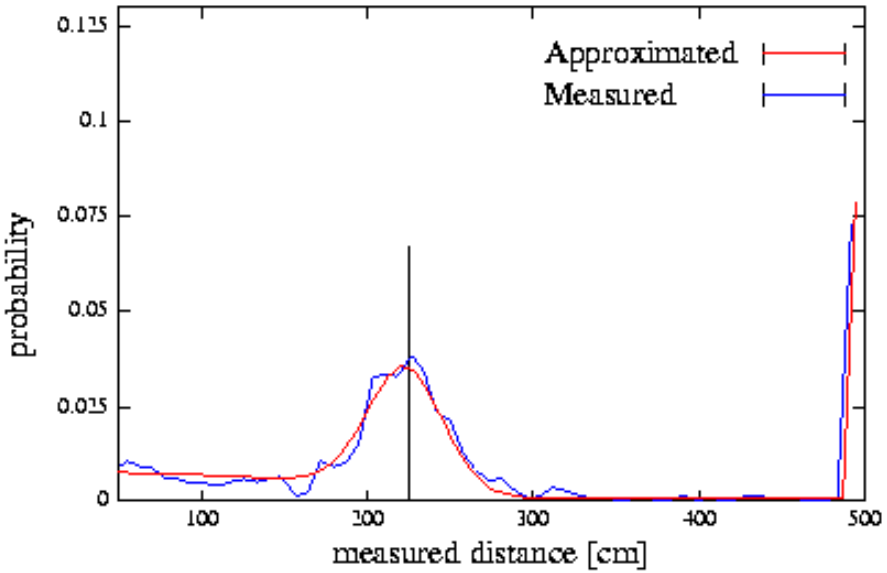
$$\square = \frac{f(x_t^{[m]})}{g(x_t^{[m]})} \propto \frac{p(z_t \mid x_t^{[m]}) p(x_t^{[m]} \mid u_t, x_{t-1}^{[m]}) \mathrm{Bel}(x_{t-1}^{[m]})}{p(x_t^{[m]} \mid u_t, x_{t-1}^{[m]}) \mathrm{Bel}(x_{t-1}^{[m]})} = \square$$

Proximity Sensor Models

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



Laser sensor



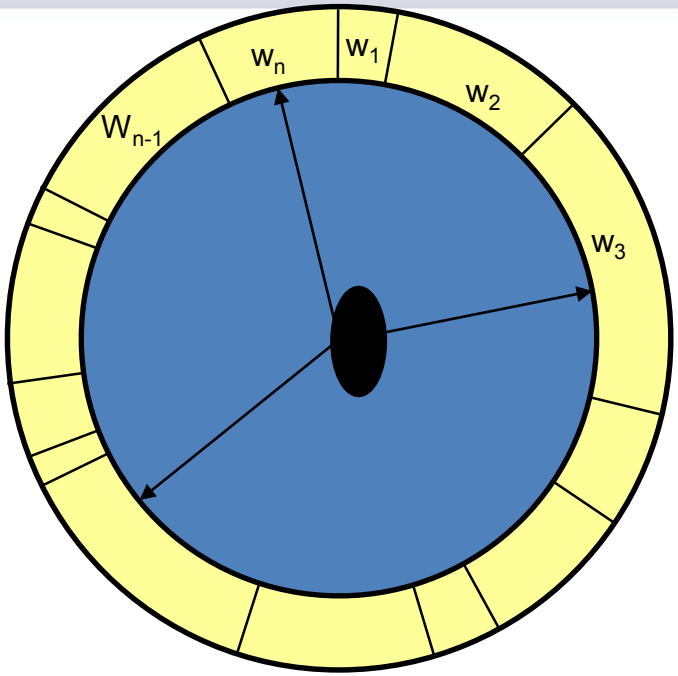
Sonar sensor

Resampling

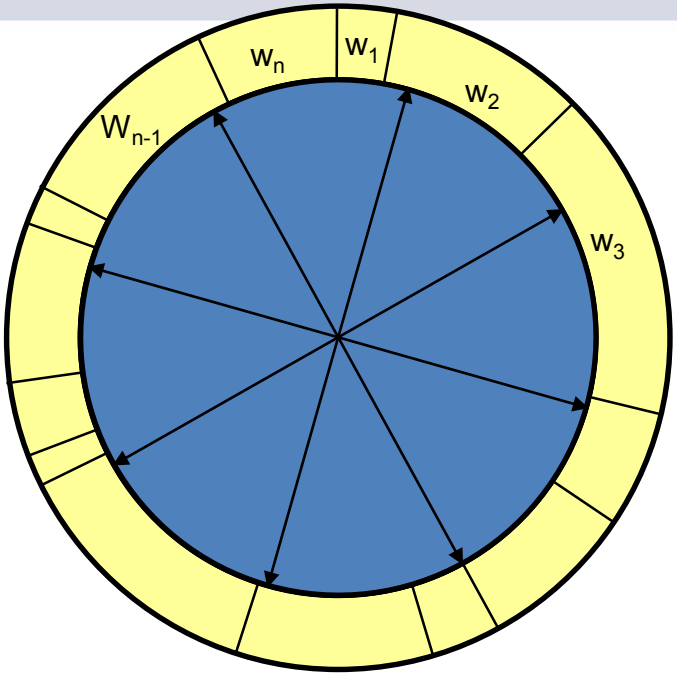
for $m = 1$ **to** M **do**
draw i with prob. $\propto w_t^{[i]}$
$\mathcal{X}_t \leftarrow \mathcal{X}_t \cup x_t^{[i]}$

- 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 $\mathcal{X}_t$.

Resampling

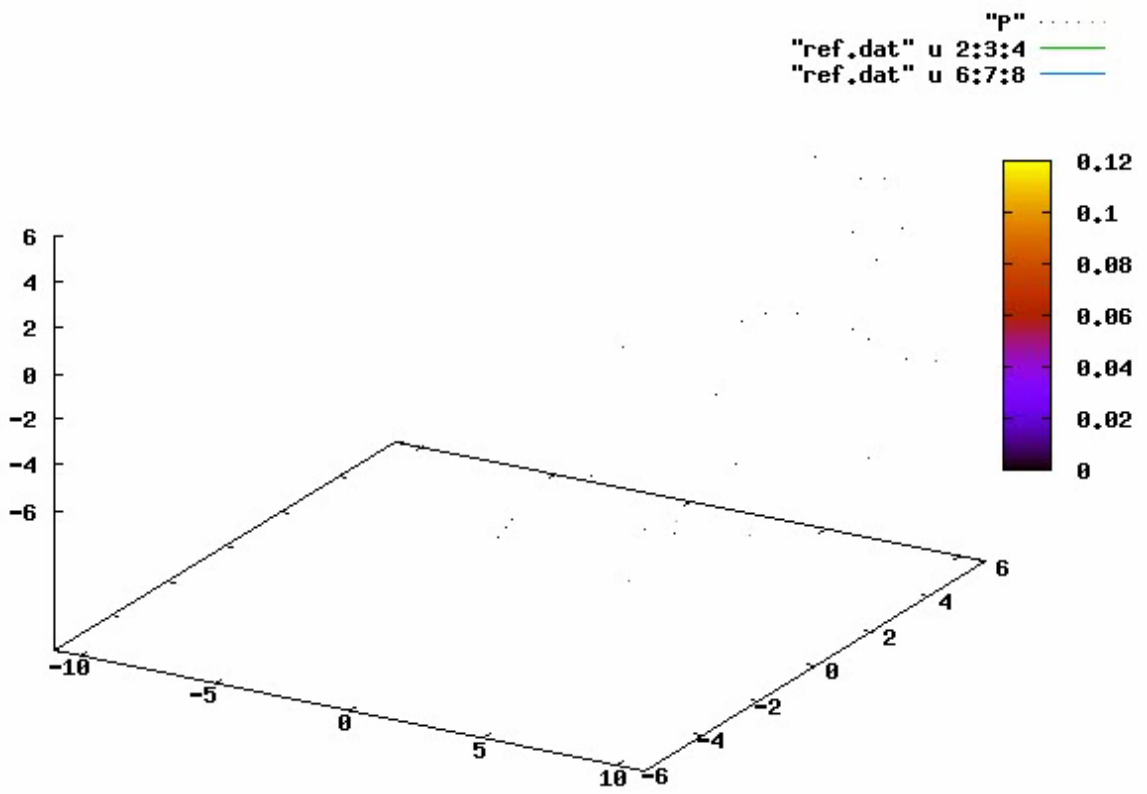


- "Roulette wheel"
- Binary search: O(n log n)



- Stochastic universal sampling
- Systematic resampling
- Linear time complexity
- Easy to implement, low variance

Example


"P"
"ref.dat" u 2:3:4
"ref.dat" u 6:7:8
0.12
0.1
0.08
0.06
0.04
0.02
0
6
4
2
0
-2
-4
-6
-10
-5
0
5
10
-6
-4
-2
0
2
4
6

Limitations

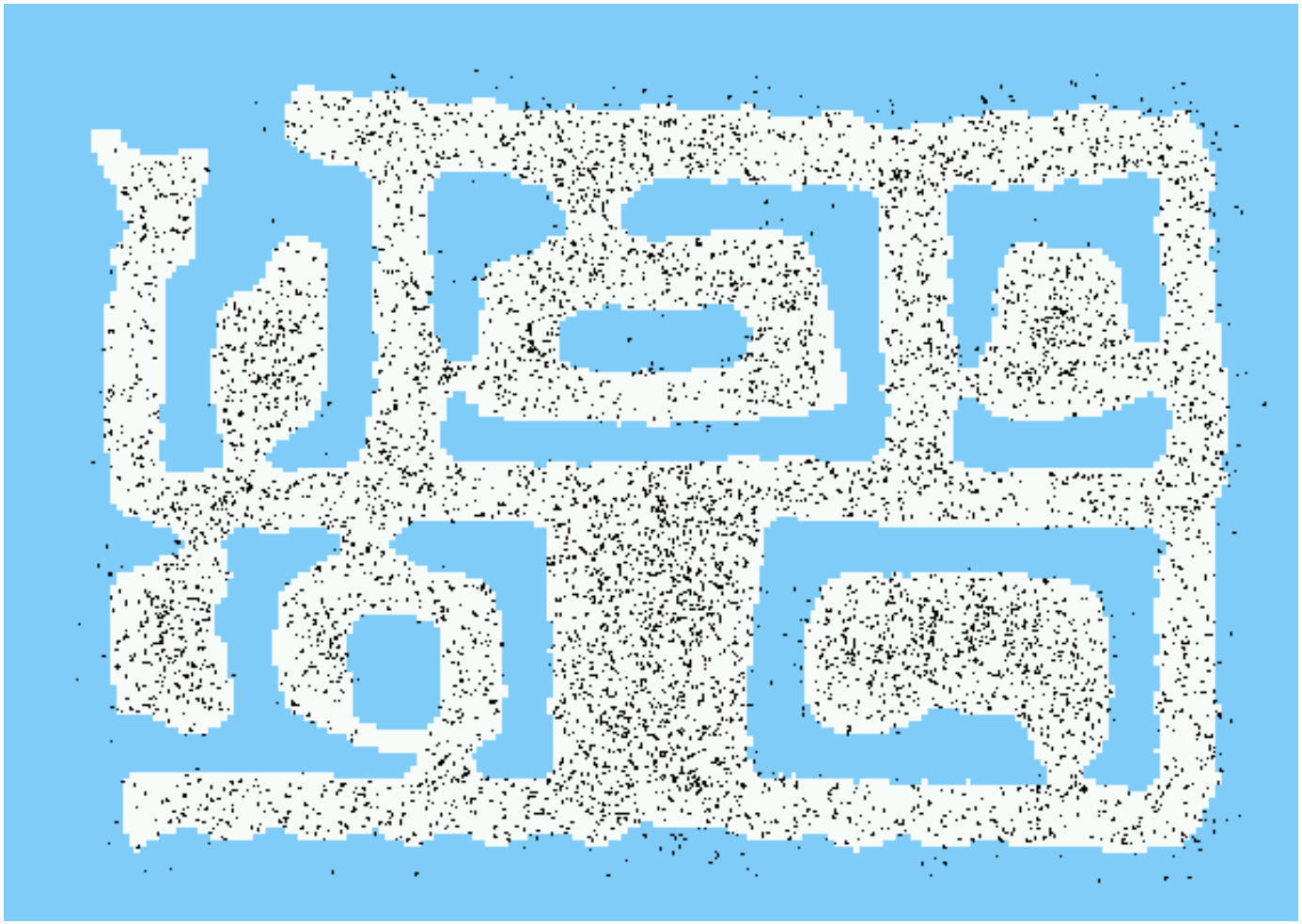
- The approach described so far is able to
  - track the pose of a mobile robot and to
  - globally localize the robot.

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

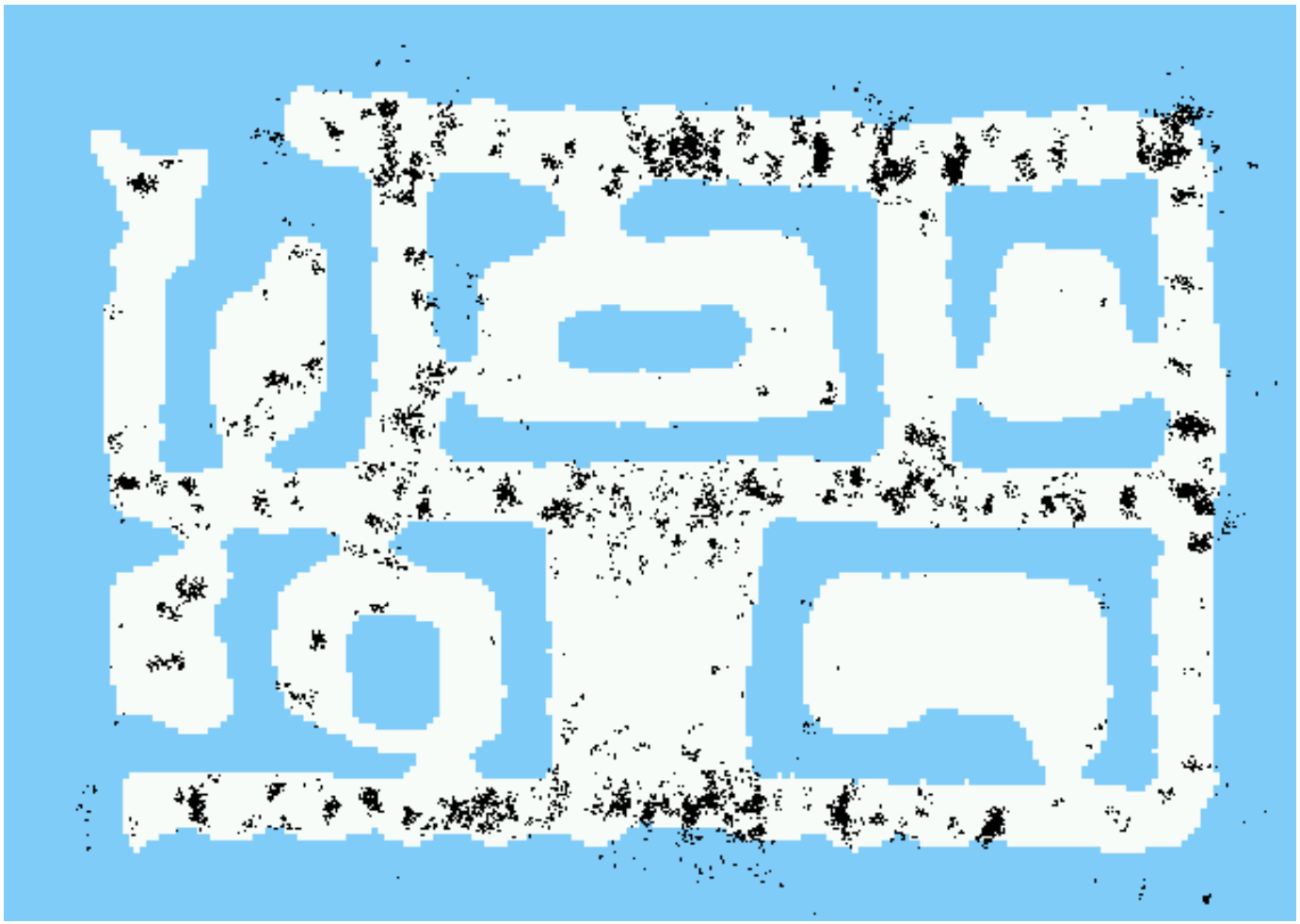
The Kidnapped Robot Problem

- Possible Approaches:
- Randomly insert samples (the robot can be teleported at any point in time).
- Insert random samples proportional to the average likelihood of the particles (the robot has been teleported with higher probability when the likelihood of its observations drops).

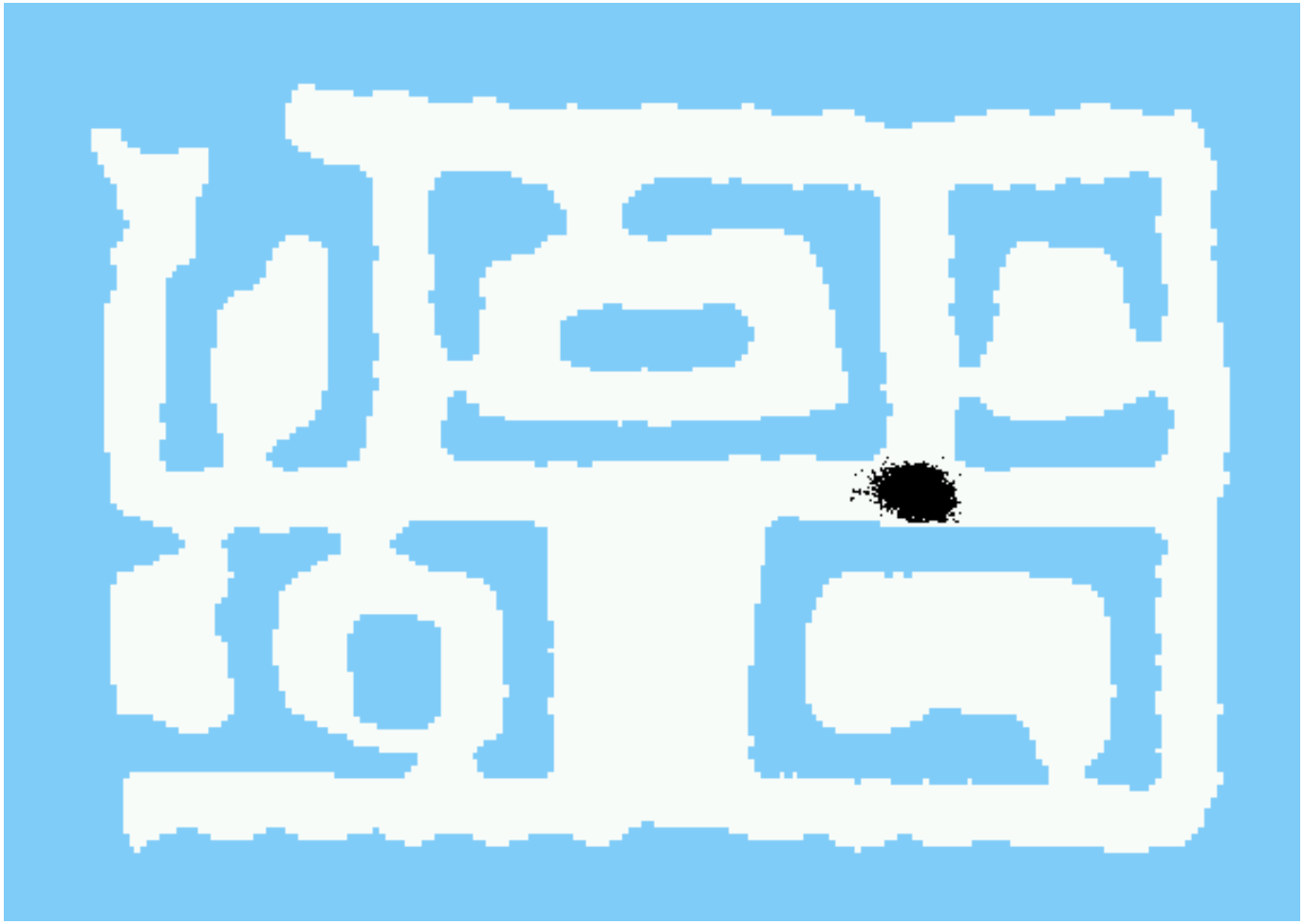
Initial Distribution



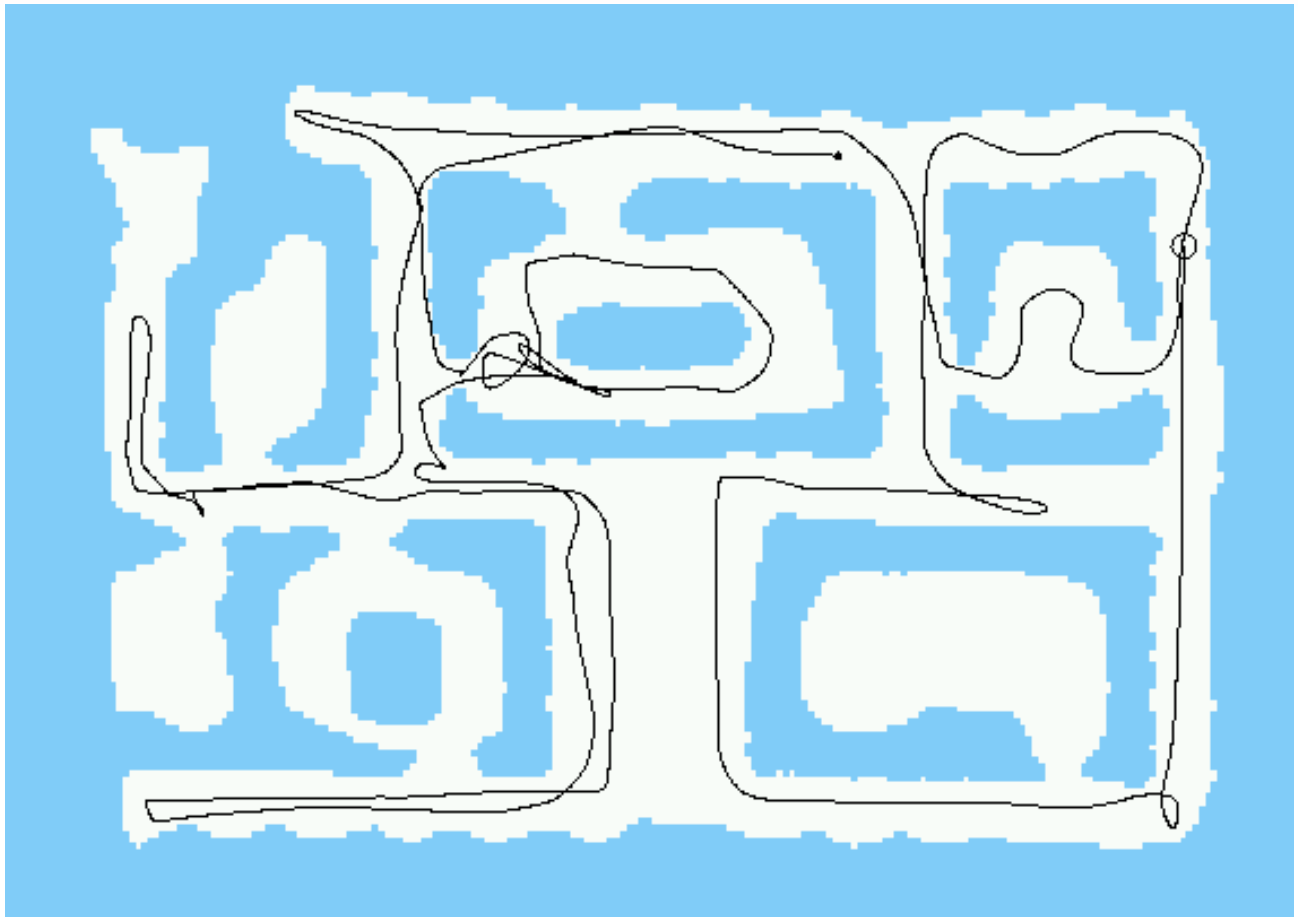
After Ten Ultrasound Scans



After 65 Ultrasound Scans



Estimated Path



KALMAN FILTER FOR LOCALISATION

Markov ⇔ Kalman Filter Localization

- Markov localization
  - localization starting from any unknown position
  - recovers from ambiguous situation.
  - However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space (grid). The required memory and calculation power can thus become very important if a fine grid is used.
- Kalman filter localization
  - tracks the robot and is inherently very precise and efficient.
  - However, if the uncertainty of the robot becomes to large (e.g. collision with an object) the Kalman filter will fail and the position is definitively lost.

Recommended reading: Introduction to the Kalman Filter (Welch & Bishop):
http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

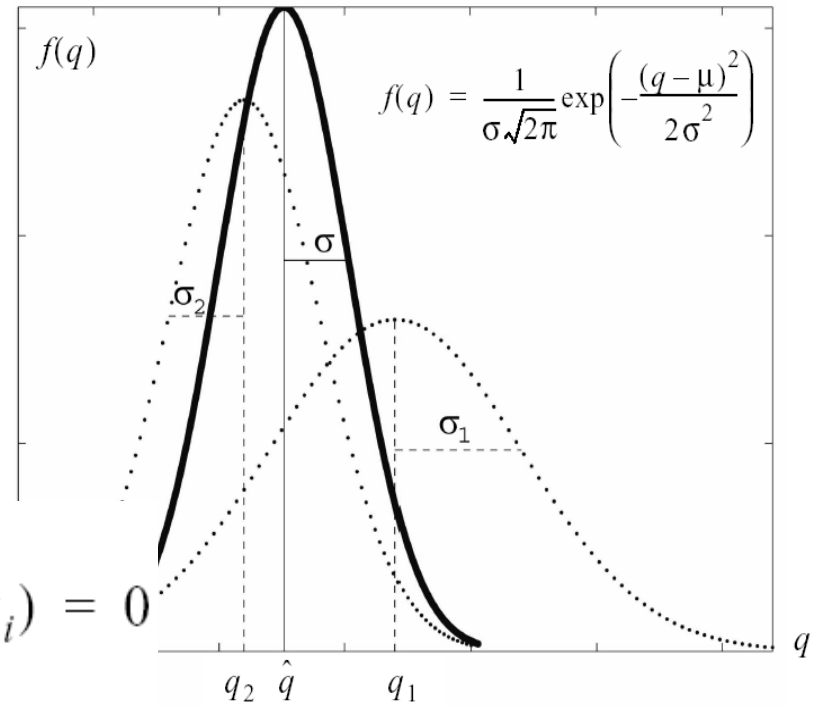
Introduction to Kalman Filter (1)

- Two measurements

$$\hat{q}_1 = q_1 \text{ with variance } \sigma_1^2$$

$$\hat{q}_2 = q_2 \text{ with variance } \sigma_2^2$$

- $S = \sum_{i=1}^{n} w_i (\hat{q} - q_i)^2$ -square



- $\frac{\partial S}{\partial \hat{q}} = \frac{\partial}{\partial \hat{q}} \sum_{i=1}^{n} w_i (\hat{q} - q_i)^2 = 2 \sum_{i=1}^{n} w_i (\hat{q} - q_i) = 0$

- After some calculation and rearrangements

$$\hat{q} = q_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}(q_2 - q_1)$$

Georgia Tech Lorraine

Introduction to Kalman Filter (2)

- In Kalman Filter notation

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1}(z_{k+1} - \hat{x}_k)$$

$$K_{k+1} = \frac{\sigma_k^2}{\sigma_k^2 + \sigma_z^2} \;;\; \sigma_k^2 = \sigma_1^2 \;;\; \sigma_z^2 = \sigma_2^2$$

$$\sigma_{k+1}^2 = \sigma_k^2 - K_{k+1}\sigma_k^2$$

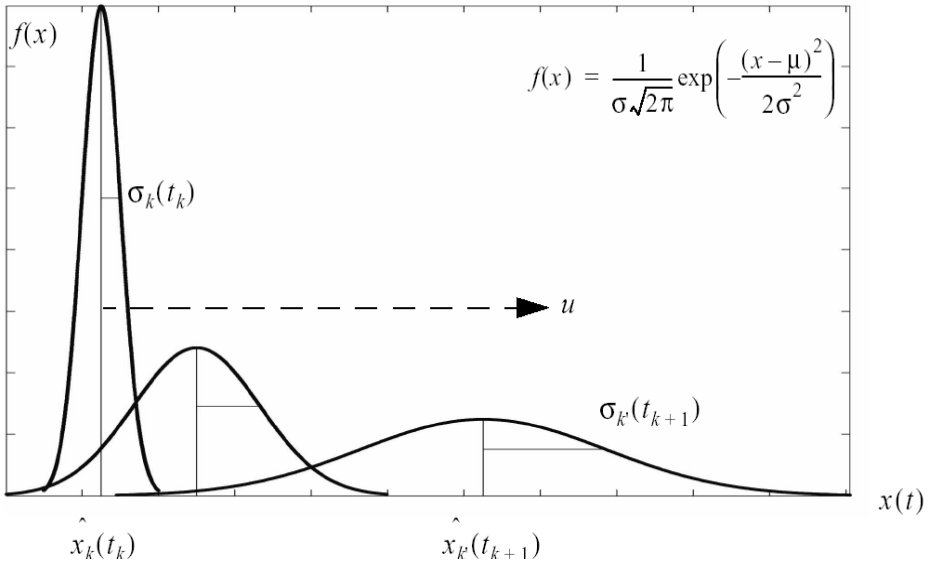
Introduction to Kalman Filter (3)

- Dynamic Prediction (robot moving)

$$\frac{dx}{dt} = u + w$$

u = velocity

w = noise



- Motion

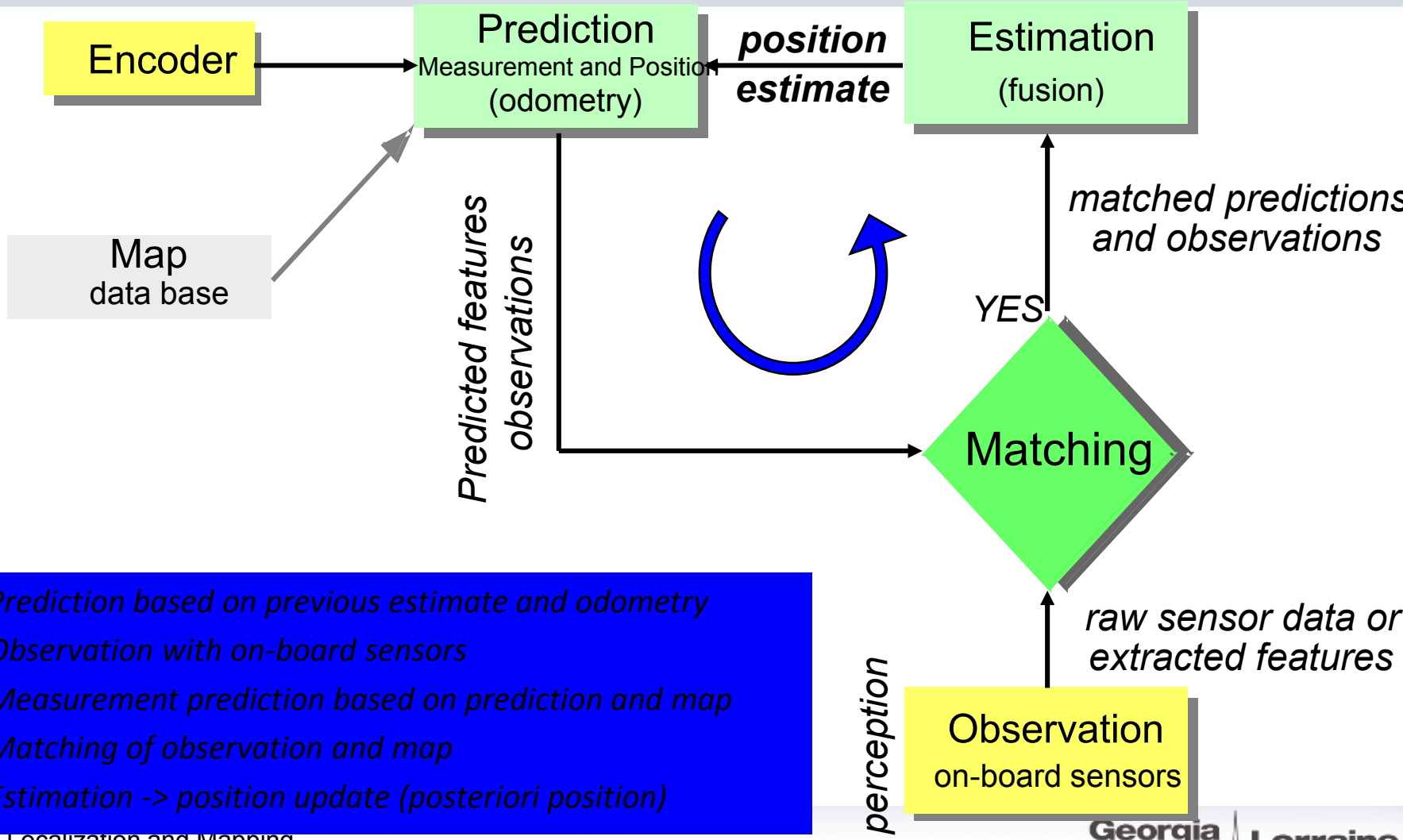
$$\hat{x}_{k'} = \hat{x}_k + u(t_{k+1} - t_k)$$
$$\sigma^2_{k'} = \sigma^2_k + \sigma^2_w[t_{k+1} - t_k]$$

$$\hat{x}_{k+1} = \hat{x}_{k'} + K_{k+1}(z_{k+1} - \hat{x}_{k'})$$
$$= [\hat{x}_k + u(t_{k+1} - t_k)] + K_{k+1}[z_{k+1} - \hat{x}_k - u(t_{k+1} - t_k)]$$

- Combining fusion and dynamic prediction

$$K_{k+1} = \frac{\sigma^2_{k'}}{\sigma^2_{k'} + \sigma^2_z} = \frac{\sigma^2_k + \sigma^2_w[t_{k+1} - t_k]}{\sigma^2_k + \sigma^2_w[t_{k+1} - t_k] + \sigma^2_z}$$

The Five Steps for Map-Based Localization



1. *Prediction based on previous estimate and odometry*
2. *Observation with on-board sensors*
3. *Measurement prediction based on prediction and map*
4. *Matching of observation and map*
5. *Estimation -> position update (posteriori position)*

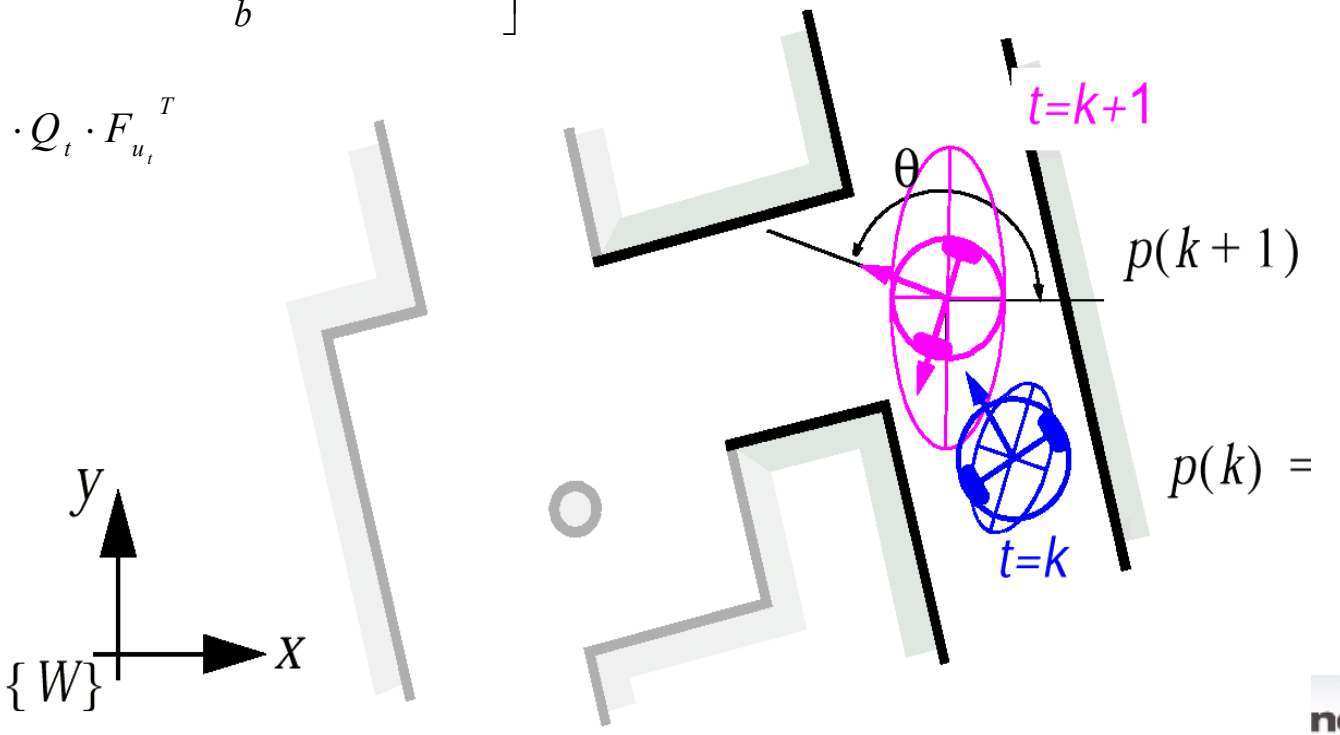
Kalman Filter Localization: Prediction update

$$\hat{x}_t = f(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}$$

$$Q_t = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix}$$

Odometry

$$P_t = F_{x_{t-1}} \cdot P_{t-1} \cdot F_{x_{t-1}}^T + F_{u_t} \cdot Q_t \cdot F_{u_t}^T$$

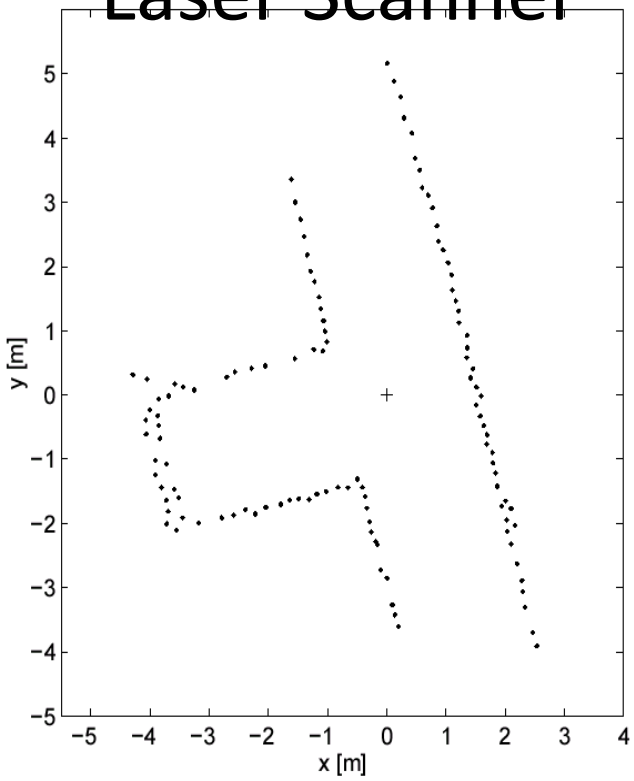


Kalman Filter for Mobile Robot Localization: Observation

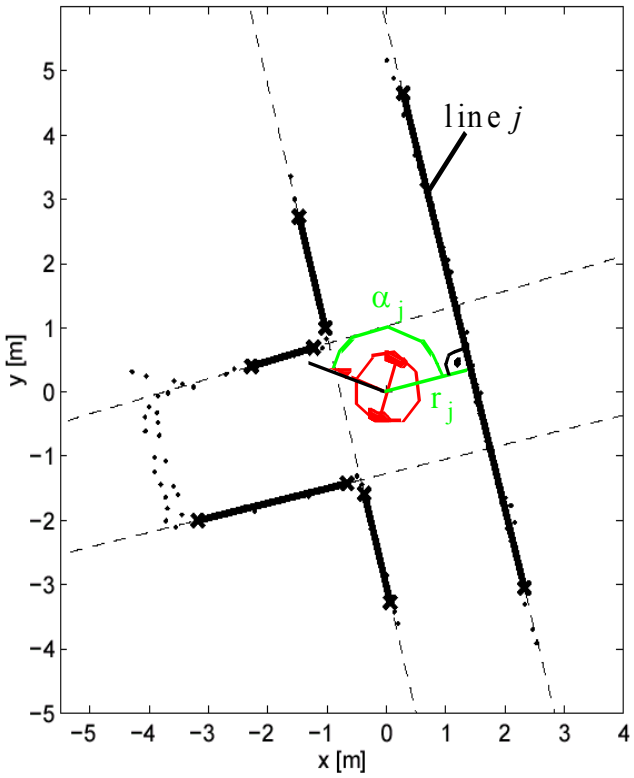
- The second step it to obtain the observation Z (measurements) from the robot's sensors at the new location
- The observation usually consists of a set n_0 of single observations z_j extracted from the different sensors signals. It represents *features* like *lines*, *doors* or *any kind of landmarks*.
- The parameters of the targets are usually observed in the sensor frame {S}.
  - Therefore the observations have to be transformed to the world frame {W} or
  - the measurement prediction have to be transformed to the sensor frame {S}.
  - This transformation is specified in the function h_j (see later).

Observation

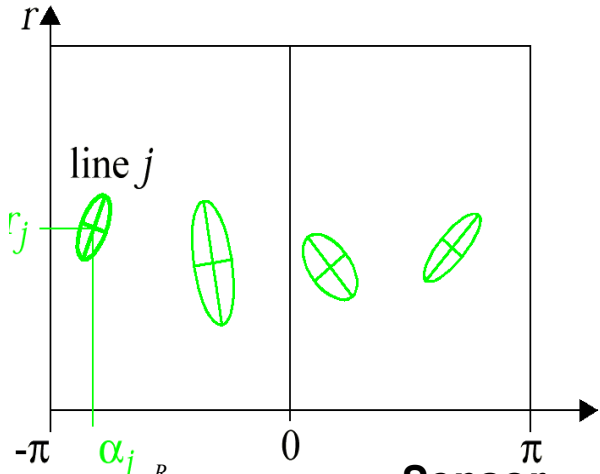
Raw Date of Laser Scanner



Extracted Lines



Extracted Lines in Model Space



Sensor (robot) frame

$$z_j = \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix}$$

$$\Sigma_{R,j} = \begin{bmatrix} \sigma_{\alpha\alpha} & \sigma_{\alpha r} \\ \sigma_{r\alpha} & \sigma_{rr} \end{bmatrix}_j$$

Measurement Prediction

- In the next step we use the predicted robot position $\hat{x}$ and the map to generate multiple predicted observations $\hat{z}_j$
- They have to be transformed into the sensor frame

$$\hat{z}_j = h_j(\hat{x})$$

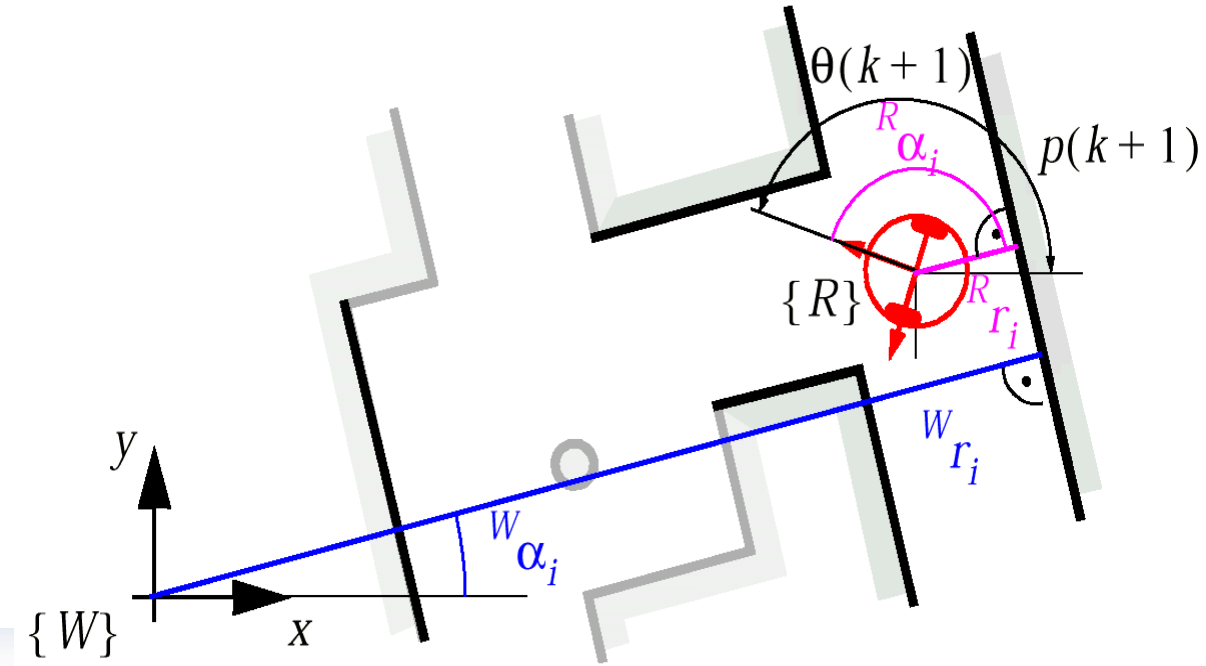
- We can now define the measurement prediction as the set containing all n_i predicted observations

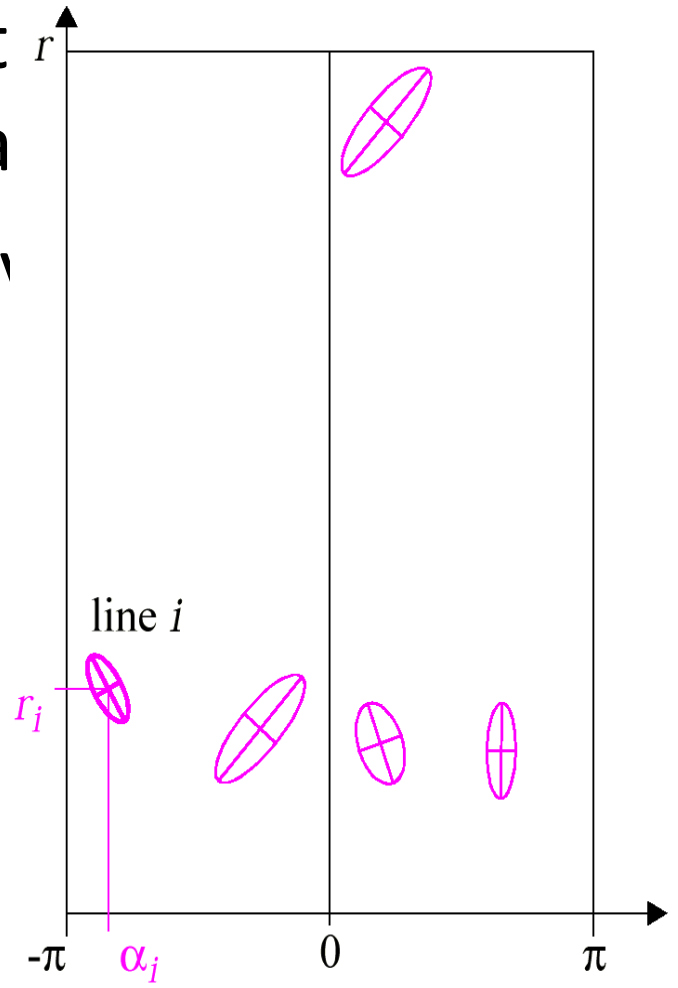
$$\hat{Z} = \left\{ \hat{z}_j \middle| (1 \leq j \leq n_i) \right\}$$

- The function h_j is mainly the coordinate transformation between the world frame and the sensor frame

Measurement Prediction

- For prediction, only the walls t
  the field of view of the robot a
- This is done by linking the indi





Measurement Prediction: *Example*

- The generated measurement predictions have to be transformed to the robot frame {*R*}

$$ {}^{W}z_{t,i} = {}^{W}\begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix} \rightarrow {}^{R}z_{t,i} = {}^{R}\begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix} $$

- According to the figure in previous slide the transformation is given by

$$ \hat{z}_i(k+1) = {}^{R}\begin{bmatrix} \alpha_{t,i} \\ r_{t,i} \end{bmatrix} = h_i(z_{t,i}, \hat{p}(k+1|k)) = \begin{bmatrix} {}^{W}\alpha_{t,i} - {}^{W}\hat{\theta}(k+1|k) \\ {}^{W}r_{t,i} - ({}^{W}\hat{x}(k+1|k)\cos({}^{W}\alpha_{t,i}) + {}^{W}\hat{y}(k+1|k)\sin({}^{W}\alpha_{t,i})) \end{bmatrix} $$

and its Jacobian

$$ \nabla h_i = \begin{bmatrix} \frac{\partial \alpha_{t,i}}{\partial \hat{x}} & \frac{\partial \alpha_{t,i}}{\partial \hat{y}} & \frac{\partial \alpha_{t,i}}{\partial \hat{\theta}} \\ \frac{\partial r_{t,i}}{\partial \hat{x}} & \frac{\partial r_{t,i}}{\partial \hat{y}} & \frac{\partial r_{t,i}}{\partial \hat{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -\cos {}^{W}\alpha_{t,i} & -\sin {}^{W}\alpha_{t,i} & 0 \end{bmatrix} $$

Kalman Filter for Mobile Robot Localization: Matching

- Assignment from observations $z_j(k+1)$ (gained by the sensors) to the targets z_t (stored in the map)
- For each measurement prediction for which $\hat{z}_i(k+1)$ ponding observation is found we calculate the innovation:

$$v_{ij}(k+1) = [z_j(k+1) - h_i(z_t, \hat{p}(k+1|k))]$$

$$= \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix} - \begin{bmatrix} {}^W\alpha_{t,i} - {}^W\hat{\theta}(k+1|k) \\ {}^W r_{t,i} - ({}^W\hat{x}(k+1|k)\cos({}^W\alpha_{t,i}) + {}^W\hat{y}(k+1|k)\sin({}^W\alpha_{t,i})) \end{bmatrix}$$

and its innovation covariance found by applying the error propagation law:

$$\Sigma_{IN,ij}(k+1) = \nabla h_i \cdot \Sigma_p(k+1|k) \cdot \nabla h_i^T + \Sigma_{R,i}(k+1)$$

- The validity of the correspondence between measurement and prediction can e.g. be evaluated through the Mahalanobis distance:

$$v_{ij}^T(k+1) \cdot \Sigma_{IN,ij}^{-1}(k+1) \cdot v_{ij}(k+1) \leq g^2$$

Matching

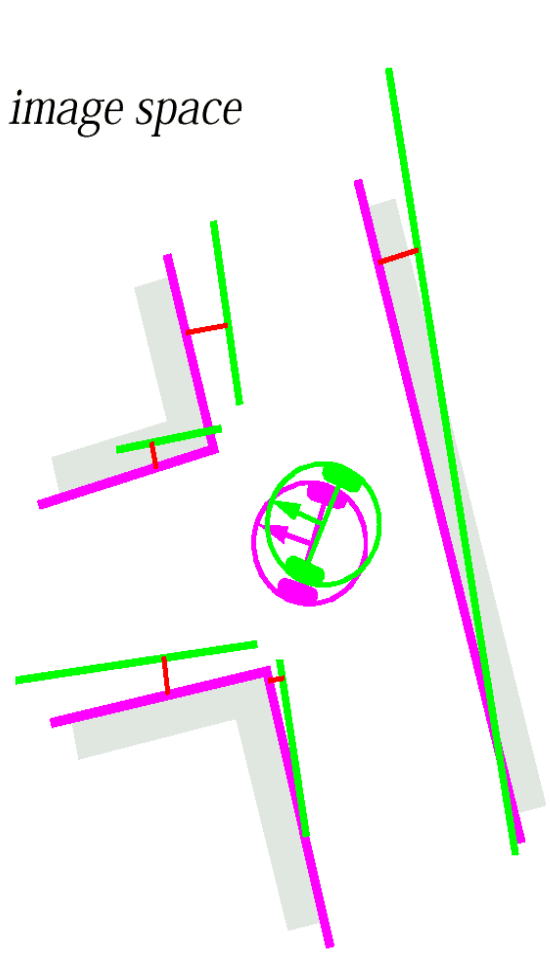
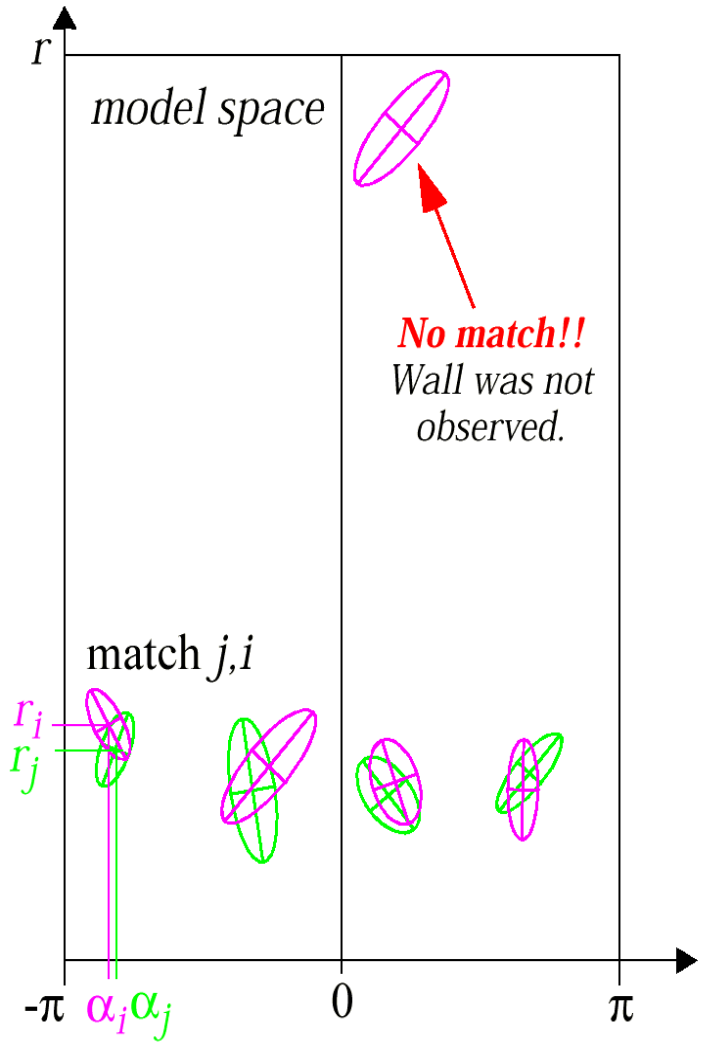


image space


r
model space
No match!!
Wall was not
observed.
match j,i
r_i
r_j
-π
$\alpha_i\alpha_j$
0
π

Matching: *Example*

- To find correspondence (pairs) of predicted and observed features we use the

$$v_{ij}(k+1) \cdot \Sigma^{-1}_{IN,\,ij}(k+1) \cdot v^{T}_{ij}(k+1) \le g^2$$

$$v_{ij}(k+1) = [z_j(k+1) - h_i(z_t, \hat{p}(k+1|k))]$$

$$= \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix} - \begin{bmatrix} {}^{W}\alpha_{t,\,i} - {}^{W}\hat{\theta}(k+1|k) \\ {}^{W}r_{t,\,i} - ({}^{W}\hat{x}(k+1|k)\cos({}^{W}\alpha_{t,\,i}) + {}^{W}\hat{y}(k+1|k)\sin({}^{W}\alpha_{t,\,i})) \end{bmatrix}$$

$$\Sigma_{IN,\,ij}(k+1) = \nabla h_i \cdot \Sigma_p(k+1|k) \cdot \nabla h_i^{T} + \Sigma_{R,\,i}(k+1)$$

Estimation: Applying the Kalman Filter

- Kalman filter gain:

$$K(k+1) = \Sigma_p(k+1|k) \cdot \nabla h^T \cdot \Sigma_{IN}^{-1}(k+1)$$

- Update of robot's position estimate:

$$\hat{p}(k+1|k+1) = \hat{p}(k+1|k) + K(k+1) \cdot v(k+1)$$

- The associated variance

$$\Sigma_p(k+1|k+1) = \Sigma_p(k+1|k) - K(k+1) \cdot \Sigma_{IN}(k+1) \cdot K^T(k+1)$$

Estimation: 1D Case $h_i(z_t, \hat{p}(k+1|k)) = z_t$

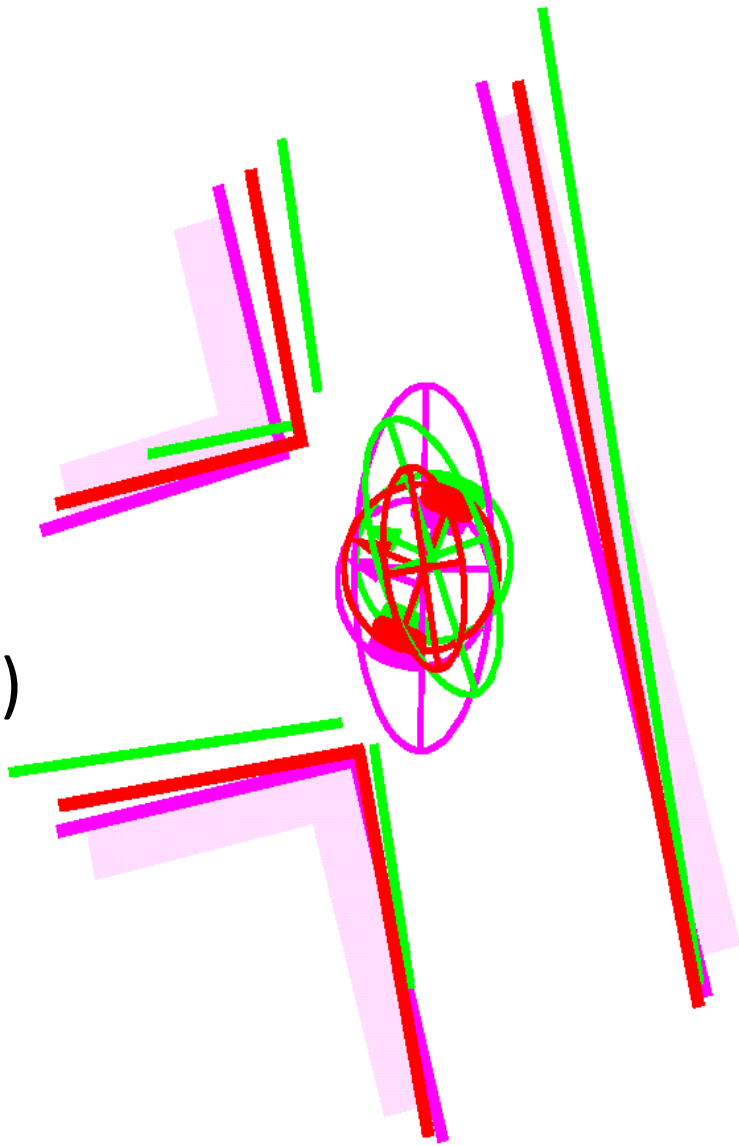
- For the one-dimensional case with
  we can show that the estimation corresponds to the Kalman filter for one-dimension presented earlier.

$$K(k+1) = \frac{\sigma_p^2(k+1|k)}{\sigma_{IN}^2(k+1)} = \frac{\sigma_p^2(k+1|k)}{\sigma_p^2(k+1|k) + \sigma_R^2(k+1)}$$

$$\begin{aligned}
\hat{p}(k+1|k+1) &= \hat{p}(k+1|k) + K(k+1) \cdot v(k+1) \\
&= \hat{p}(k+1|k) + K(k+1) \cdot [z_j(k+1) - h_i(z_t, \hat{p}(k+1|k))] \\
&= \hat{p}(k+1|k) + K(k+1) \cdot [z_j(k+1) - z_t]
\end{aligned}$$

Estimation

- Kalman filter estimation of the new robot position :
  - By fusing the prediction of robot position (magenta) with the innovation gained by the measurements (green) we get the updated estimate of the robot position (red)



Markov versus Kalman: summary

- Remember: both methods solve the convolution and Bayer rules for the action and the perception update respectively, but:

Markov Localization	Kalman Filter based Localization
• The configuration space is divided into many cells. The configuration space of a robot moving on a plane is 3D dimensional (x,y,θ). Each cell contains the probability of the robot to be in that cell. • The probability distribution of the sensors model is also discrete. • During Action and Perception, all the cells are updated	• The probability distribution of both the robot configuration and the sensor model is assumed to be continuous and **Gaussian**! • Since a Gaussian distribution only described through mean value μ and variance σ^2, we need only to update μ and σ!! Therefore the computational cost is very low!
• **PROS:** 1. any probability distribution and any statistical error model for the sensor can be considered 2. the robot can start from any unknown position and can recover from ambiguous situations 3. can be used for topological localization	• **PROS** At every Action and Perception update we need to update only μ, σ, therefore we need 4 equations: 2 during the Action and 2 during the Perception phase. • As we need to update only 2 quantities instead of many cells, the computational cost is very low.
• **CONS:** Every cell must be updated during Action and Perception. In some cases the computation can become too heavy for real-time operations. For example, a robot moving on a plane is described through (x,y,θ). If we have a 20x100 m^2 environment and the size of each cell is 0.1 m in x-y and 1 deg in θ, then the configuration space would sum up to $100x20x100x360 = 72 \cdot 10^6$ cells which need to be updated at each time!!!	• **CONS:** Only Gaussian distributions are considered and therefore if the robot probability distribution or the sensor model cannot be approximated by a normal distribution the Kalman filter cannot be adopted or will give poor results. It may also not converge! Furthermore it is not possible to recover from ambiguous situations or situations where the robot is completely lost!

CONCLUSION

Conclusion

- Probabilistic localisation is considered the standard way of localising in robotics
  - Uncertainty on the sensor data and motion process
- Variation around the Bayes filter based on workspace and distributions:
  - KF for linear gaussian processes (rare)
  - EKF for non-linear (not too much) gaussian processes
  - PF for generic processes and kidnapped robots
  - Discretized Markov Loc for exhaustive representations
  - UKF, combinations, variations, …
- Remember the importance of Data Association