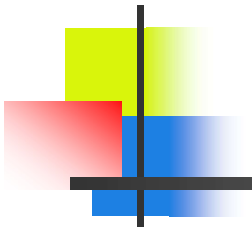


# Localization

Mobile robot localization is the problem of determining the pose of a robot relative to a given map of the environment.

# Taxonomy of Localization Problem 1

- Local vs. Global Localization
  - Position tracking
    - initial robot pose is known
    - pose uncertainty is approximated by a unimodal distribution (e.g., a Gaussian)
    - uncertainty is local
  - Global localization
    - initial robot pose is unknown
    - cannot assume boundedness of the pose error, thus unimodal prob. distribution is inappropriate
  - Kidnapped robot problem
    - during operation, robot can get kidnapped and teleoperated to some other location
    - more difficult than the global localization
    - robot might believe it knows where it is while it does not

# Taxonomy of Localization Problem 2

- Static vs. Dynamic Environment
  - Static environment:
    - the only variable (state) is the robot's pose
  - Dynamic environment:
    - objects other than the robot whose locations or configurations change over time, such as people, daylight, movable furniture, doors
    - Solution:
      - dynamic entities might be included in the state vector
      - sensor data can be filtered so as to eliminate the damaging effect of unmodeled dynamics

# Taxonomy of Localization Problem 3

- Passive vs. Active
  - Passive localization: the localization module only observes the robot operating
  - Active localization: controls the robot so as to minimize the localization error and/or the costs
    - coastal navigation
    - direct the robot to move into a room in a symmetric corridor to eliminate ambiguity

- Single-Robot vs. Multi-Robot
  - One robot's belief can be used to bias another robot's belief

# Localization: Iconic vs. Feature-Based

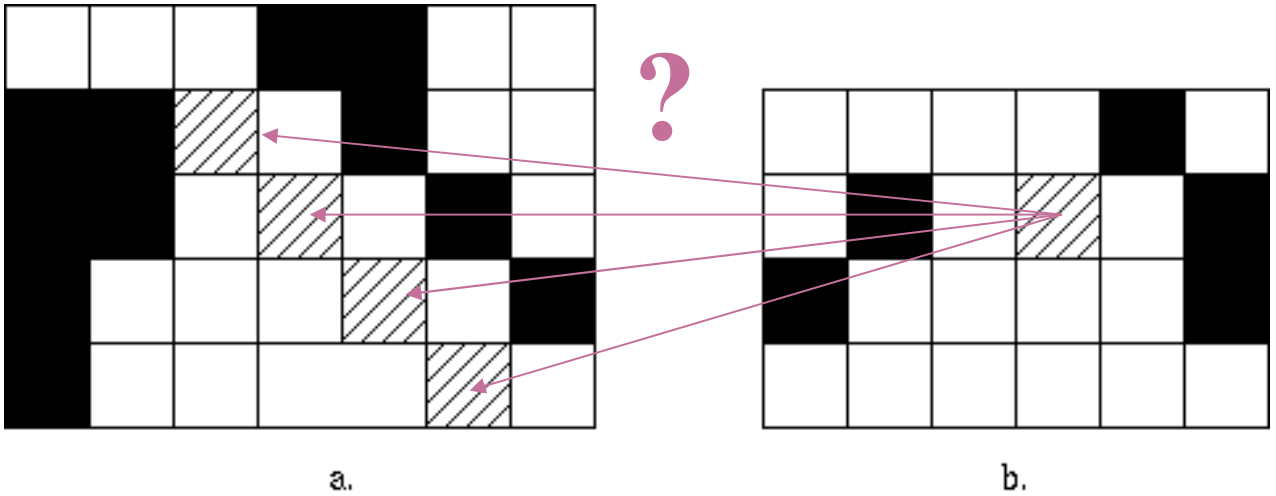
- Localization methods:
  - Iconic method:
    - Use raw sensor data directly
    - Would match sensor readings to data fused from previous sensor measurements in occupancy grid
    - Used a lot in current metric map making
  - Feature-based method:
    - Use features extracted from sensor data
    - E.g., extract a corner from sonar data or occupancy grid, then compare how corner moved in next step
    - Conceptually similar to distinctive places
    - Good for topological map making

# Continuous Localization and Mapping (Iconic)

- Use exteroception sensors (e.g., laser, sonar, vision, etc.)
  - To eliminate problems with proprioceptive techniques (e.g., shaft encoder)
- Match current perception with world with past observations (i.e., map)
  - This is NOT easy!
- Once true position of robot is known with respect to the map, current perception added to the map (called "registration")

# Matching Current Perception with Previous Map

- If robot hasn't moved far, a large portion of grid should match what is already in the global map
- BUT: could be error in the sensing
- The shaft encoders provide a set of possible poses (x, y, θ)



# Tradeoffs in Localization

- Localization attempts to balance competing interests:
  - Localization attempts to determine proper:
    - Frequency of updates (more could be better, but also incur more noise)

- Tradeoffs between:
  - Localizing after every sensor reading
  - Localizing after *n* sensor updates have been fused
  - The choice of *n* is done by trial and error

# Feature-Based Localization

- Two types:
  - (Similar to continuous localization and mapping): robot extracts feature, then tries to find feature in next sensor update
  - (Extension of topological navigation): robot localizes itself relative to topological features
- Sensor uncertainty: still plays large role
- Use gateways, plus topological map, to determine position (or set of possible positions)

# Comparison of Two Methods

- Shaffer et al. concluded that:
  - Iconic methods are more accurate
  - Iconic methods impose fewer restrictions on the environment
  - Feature-based algorithms are faster (ignoring the cost of feature extraction)
  - Feature-based algorithms can handle poor initial estimates well

- No technique handles a dynamic environment
  - Localization to a priori map cannot allow a large number of discrepancies between map and current state

# Our Focus: Two Types of Localization Problems

- Global position estimation – figure out where the robot is, but we don't know where the robot started, given a priori map

- Local position tracking – figure out where the robot is, given that we know where the robot started

# Basic Idea of Markov Localization

- A variant of the Bayes filter

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

Algorithm Markov_localiztion($bel(x_{t-1}), u_t, z_t, m$):

*for all* $x_t$ *do*

$$bel'(x_t) = \int p(x_t \mid u_t, x_{t-1}, m) bel(x_{t-1}) dx$$

$$bel(x_t) = \eta \, p(z_t \mid x_t, m) bel'(x_t)$$

*endfor*

*return* $bel(x_t)$

# Revisit Bayes Filters

z = observation
u = action
x = state

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

**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**
$$= \eta \; P(z_t \mid x_t) \; P(x_t \mid u_1, z_1, \ldots, u_t)$$

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

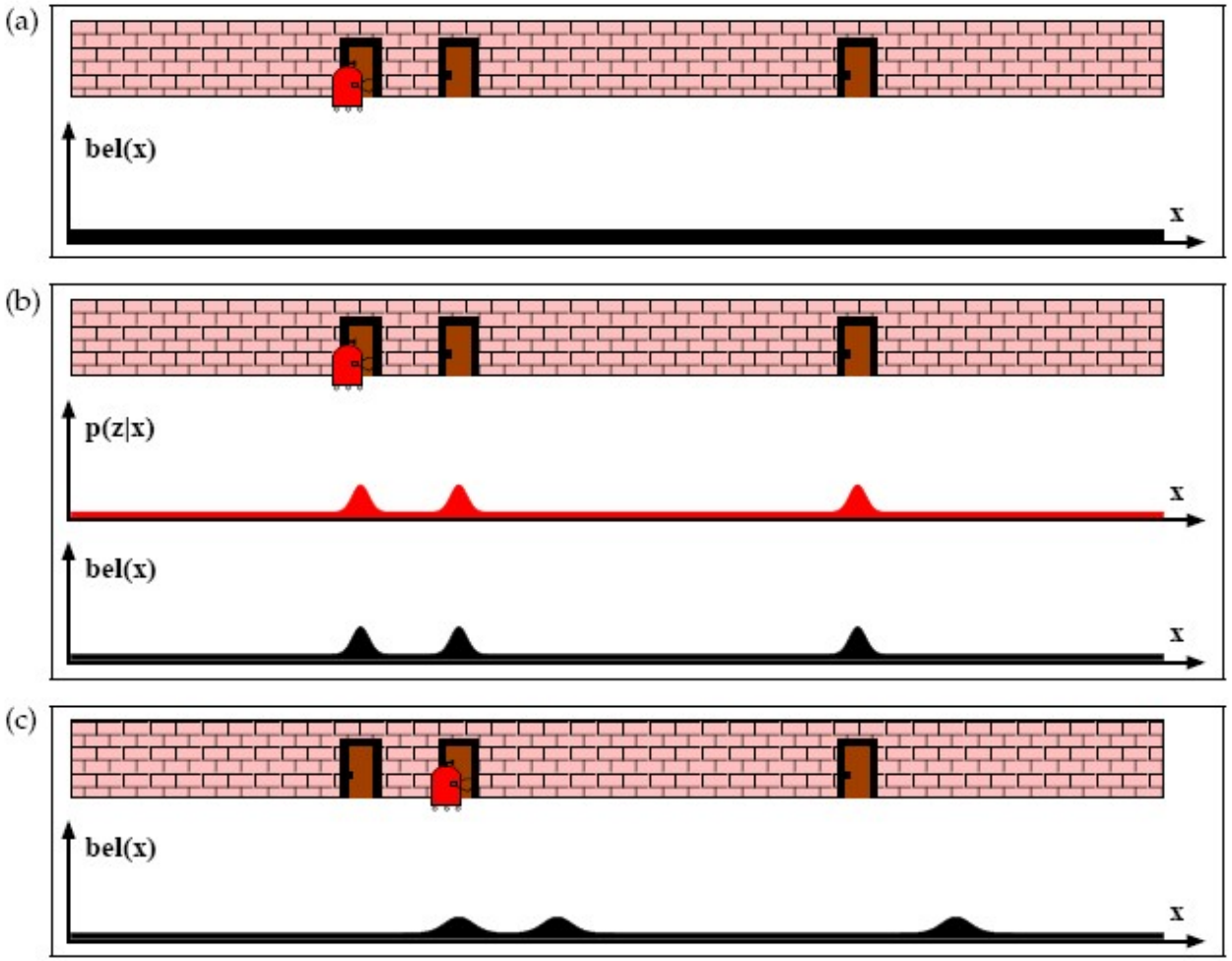
$$= \eta \; P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1}) \; Bel(x_{t-1}) \, dx_{t-1}$$

# Markov Localization

- Markov localization addresses all three localization problems:
  - Position tracking
    - $bel(x_0) = 1$ if $x_0 = x'_0$ (known initial position)
    - Or Gaussian distribution centered around $x'_0$
  - Global localization
    - Uniform distribution over the space of all legal poses
    - $bel(x_0) = 1/|X|$
  - Kidnapped robot

# Illustration of the ML Algorithm (1)

- Key idea: compute a probability distribution over all possible positions in the environment, representing the likelihood that the robot is in a particular location




1. Initial position unknown: uniform distribution: $\mathrm{bel}(x_0)$

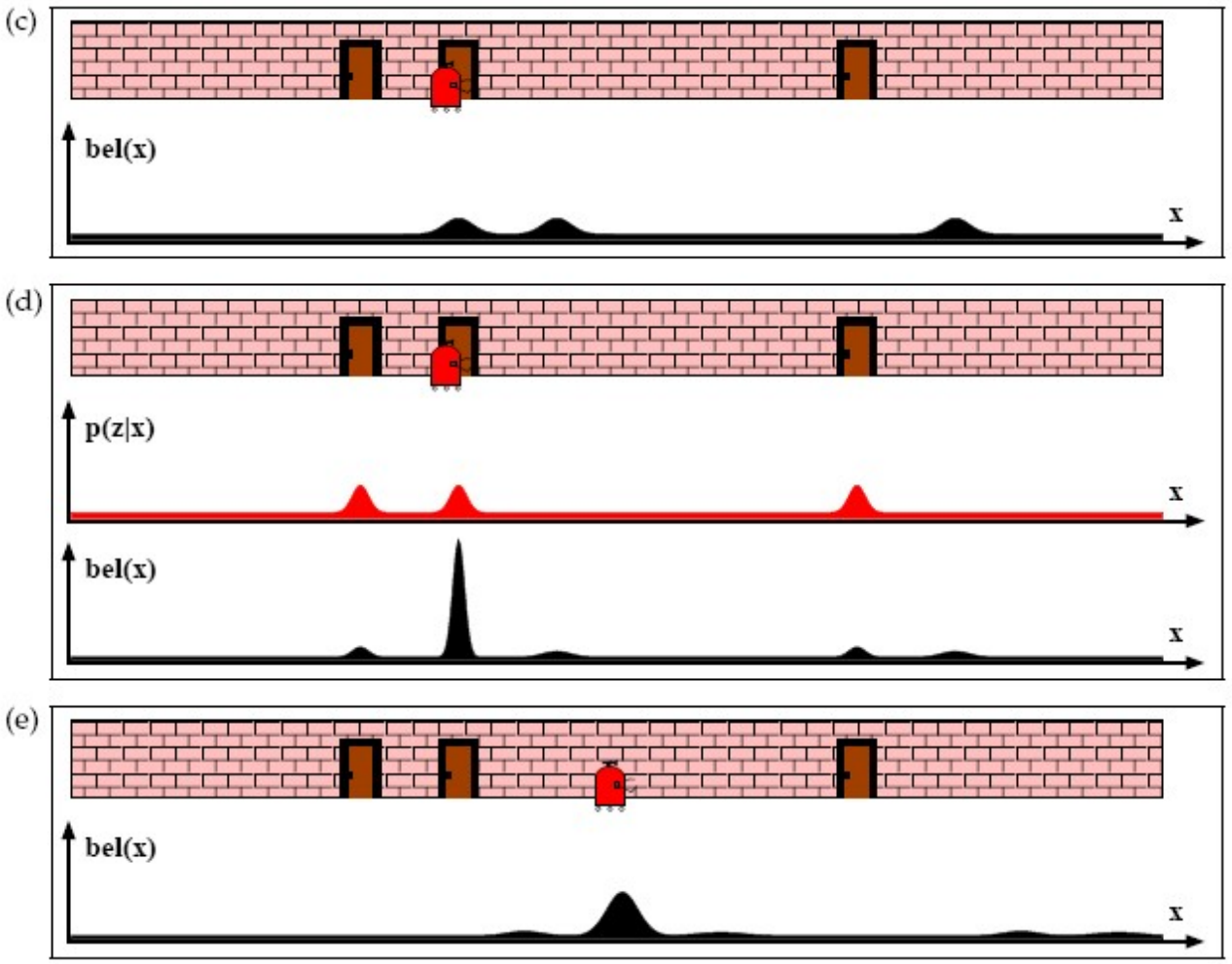
2. Door sensed: raising probabilities for places close to door

   multiply its belief $\mathrm{bel}(x_0)$ by $\mathrm{p}(z_t \mid x_t, m)$

3. Moved forward: shifting belief distribution accordingly

   motion: $\mathrm{p}(x_t \mid u_t, x_{t-1})$

# Illustration of the ML Algorithm (2)




3. Moved forward: shifting belief distribution accordingly

motion: $p(x_t \mid u_t, x_{t-1})$

4. Door sensed: raising probabilities $p(z_t \mid x_t)$ for places close to door, multiplied to current belief

5. robot's belief after having moved further down the hallway

# What does "Markov" Mean?

- "Markov" means the system obeys the Markov Property

- Markov Property:
  - the conditional probability of the future state is dependent only on the current state and independent of the past states

- For the purpose of robot localization:
  - Future sensor readings are conditionally independent of past readings, given the true current position of the robot
  - We don't have to save all the prior sensor data and apply it each time we update beliefs on the robot's location

# Markov Localization (Con't.)

- Bel($x$) represents the robot's belief that it is at position $x$
  - Bel($x$) is a probability distribution, centered on the correct position

- Two probabilistic models are used to update Bel($x$):
  - Action (or motion model): represents movements of robot
  - Perception (or sensing model): represents likelihood that robot senses a particular reading at a particular position

1: Algorithm Markov_localization$(bel(x_{t-1}), u_t, z_t, m)$:

2: *for all* $x_t$ *do*

3: $bel'(x_t) = \int p(x_t \mid u_t, x_{t-1}, m) bel(x_{t-1})\, dx$ prediction

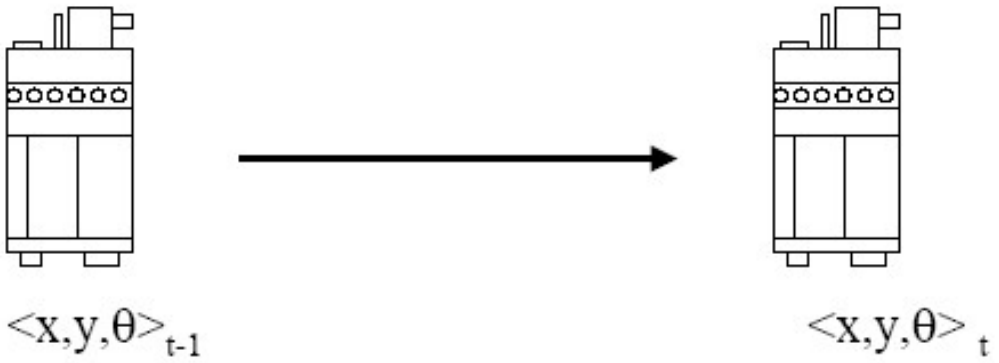
4: $bel(x_t) = \eta\, p(z_t \mid x_t, m) bel'(x_t)$ correction

5: *endfor*

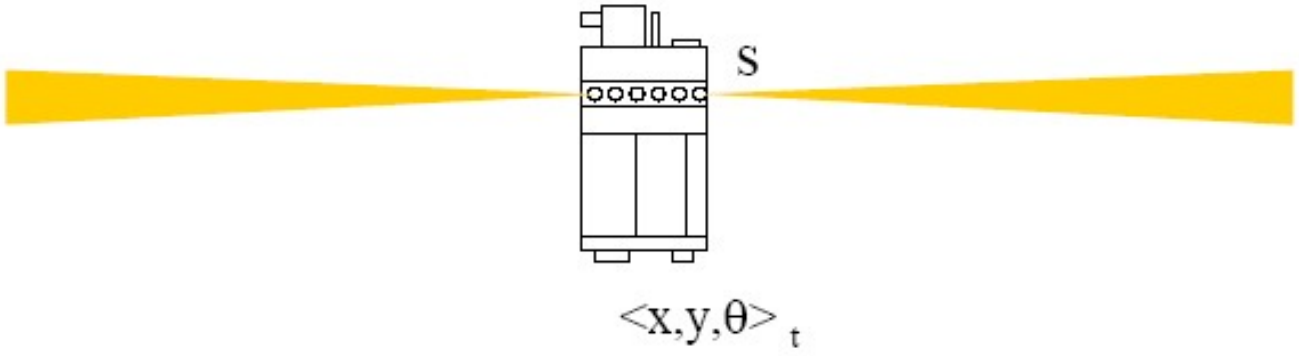
6: *return* $bel(x_t)$

# Common Terminology

- Prediction Phase: Applying motion model




- Measurement Phase: Applying sensor model




Slide adapted from Dellaert presentation "19-Particles.ppt"

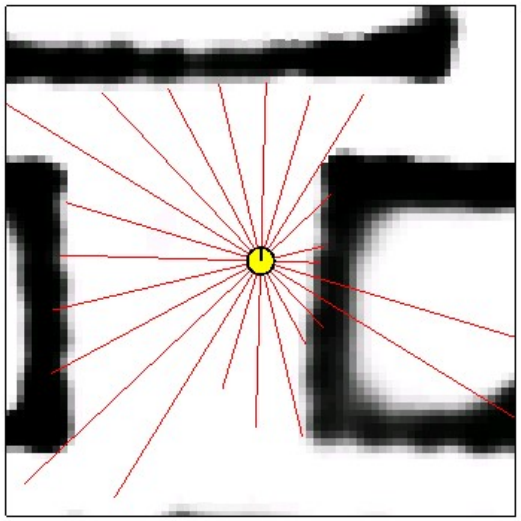
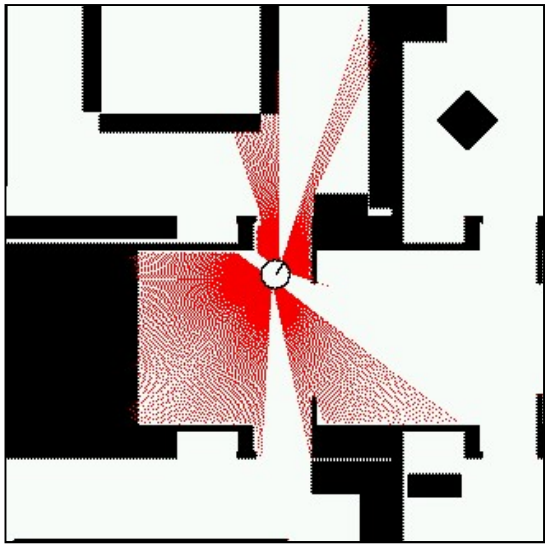
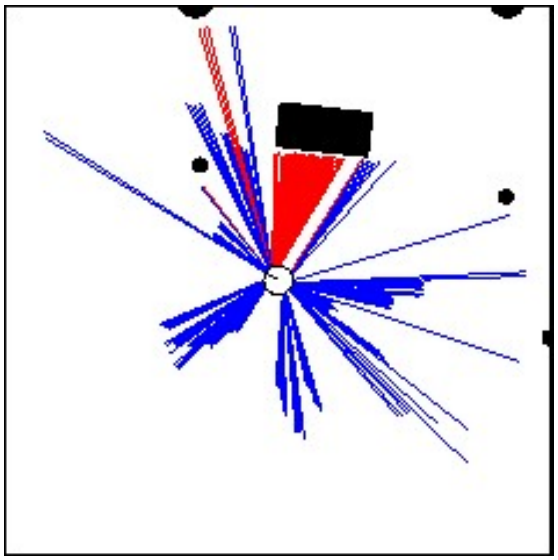
# Probabilistic Sensor Models

- Beam-based
- Scan-based
- Landmarks

# Sensors for Mobile Robots

- Contact sensors: Bumpers
- Internal sensors
  - Accelerometers (spring-mounted masses)
  - Gyroscopes (spinning mass, laser light)
  - Compasses, inclinometers (earth magnetic field, gravity)
- Proximity sensors
  - Sonar (time of flight)
  - Radar (phase and frequency)
  - Laser range-finders (triangulation, tof, phase)
  - Infrared (intensity)
- Visual sensors: Cameras
- Satellite-based sensors: GPS

# Proximity Sensors

- The central task is to determine $P(z|x)$, i.e., the probability of a measurement $z$ given that the robot is at position $x$.
- **Question**: Where do the probabilities come from?
- **Approach**: Let's try to explain a measurement.

# Beam-based Sensor Model

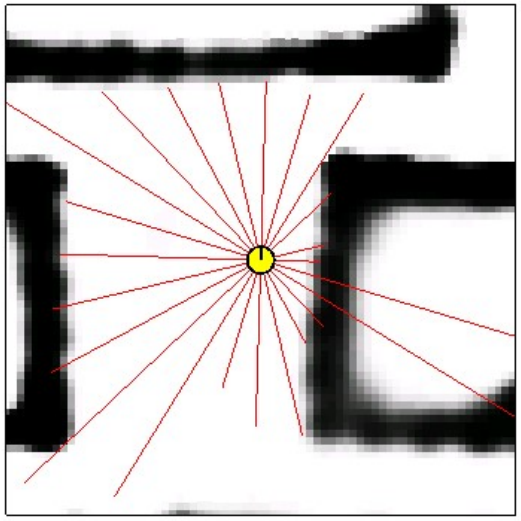
- Scan $z$ consists of $K$ measurements.

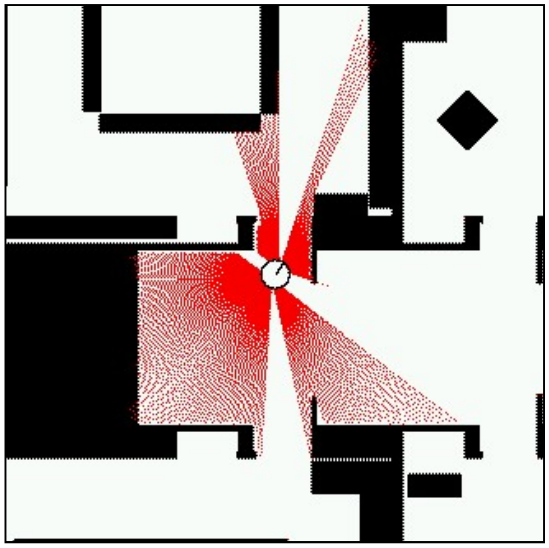
$$z = \{z_1, z_2, \ldots, z_K\}$$

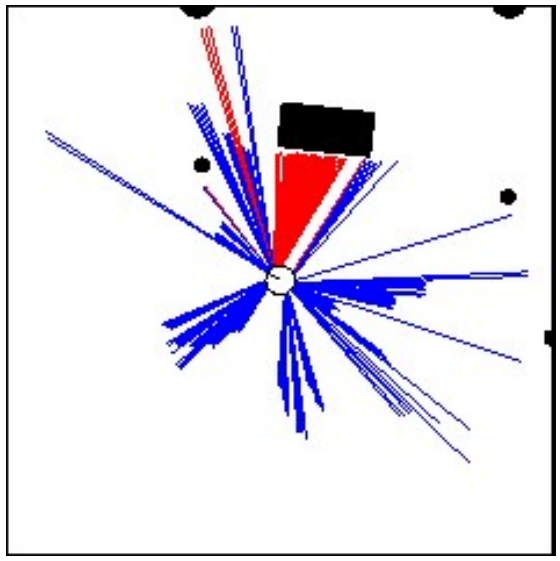
- Individual measurements are independent given the robot position.

$$P(z \mid x, m) = \prod_{k=1}^{K} P(z_k \mid x, m)$$

# Beam-based Sensor Model



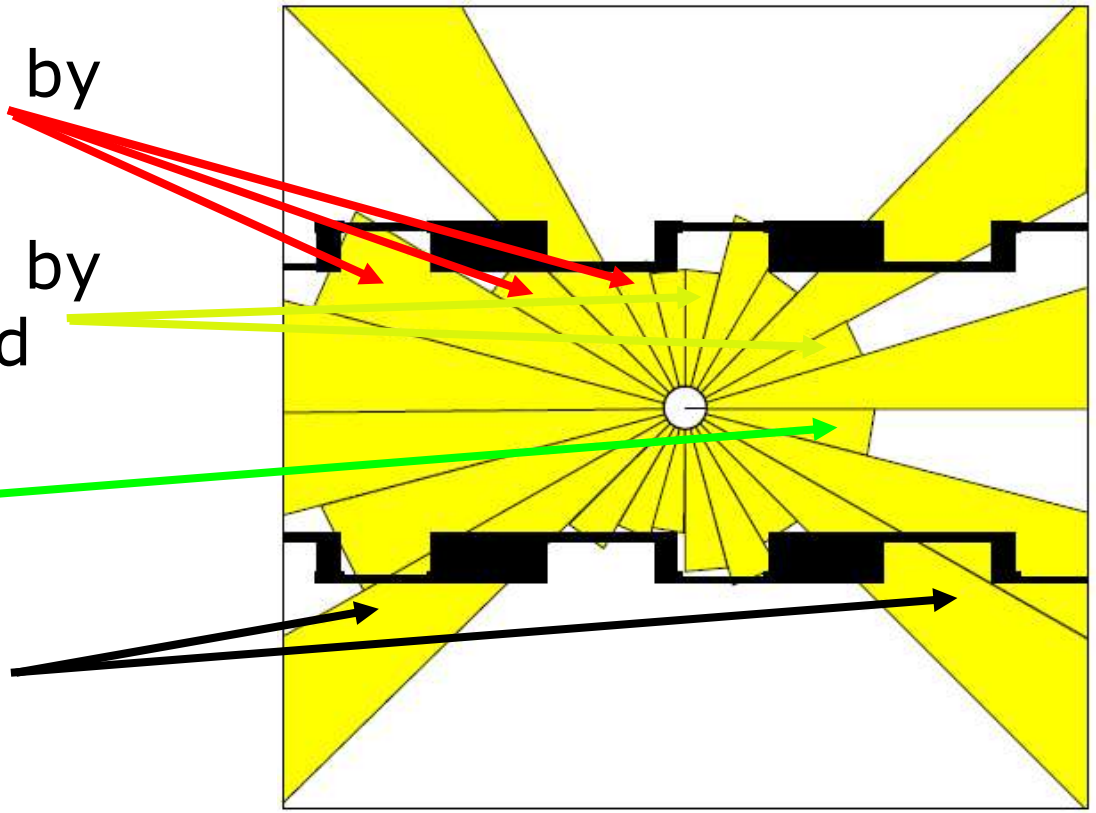




$$P(z \mid x, m) = \prod_{k=1}^{K} P(z_k \mid x, m)$$

# Typical Measurement Errors of an Range Measurements

1. Beams reflected by obstacles
2. Beams reflected by persons / caused by crosstalk
3. Random measurements
4. Maximum range measurements

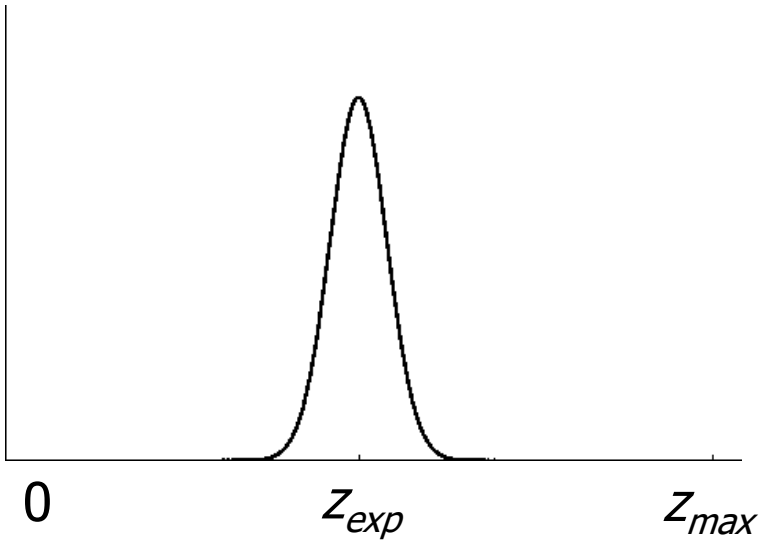


# Proximity Measurement

- Measurement can be caused by …
  - a known obstacle.
  - cross-talk.
  - an unexpected obstacle (people, furniture, …).
  - missing all obstacles (total reflection, glass, …).
- Noise is due to uncertainty …
  - in measuring distance to known obstacle.
  - in position of known obstacles.
  - in position of additional obstacles.
  - whether obstacle is missed.

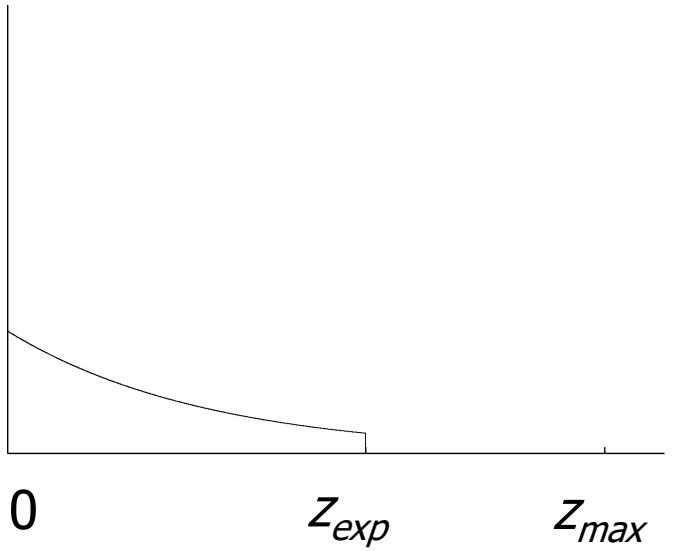
# Beam-based Proximity Model

## Measurement noise




$$P_{hit}(z \mid x,m) = \eta \frac{1}{\sqrt{2\pi b}} e^{-\frac{1}{2}\frac{(z-z_{\exp})^2}{b}}$$

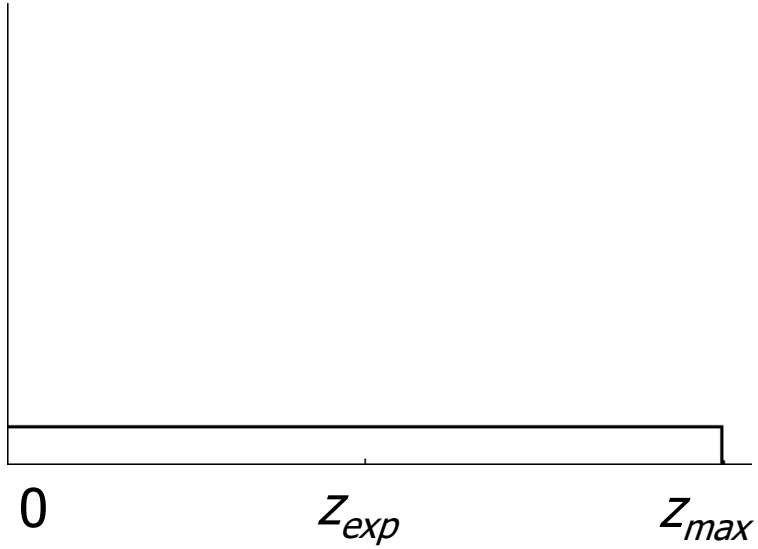
## Unexpected obstacles




$$P_{\text{unexp}}(z \mid x,m) = \begin{cases} \eta\ \lambda\ \mathrm{e}^{-\lambda z} & z < z_{\exp} \\ 0 & otherwise \end{cases}$$

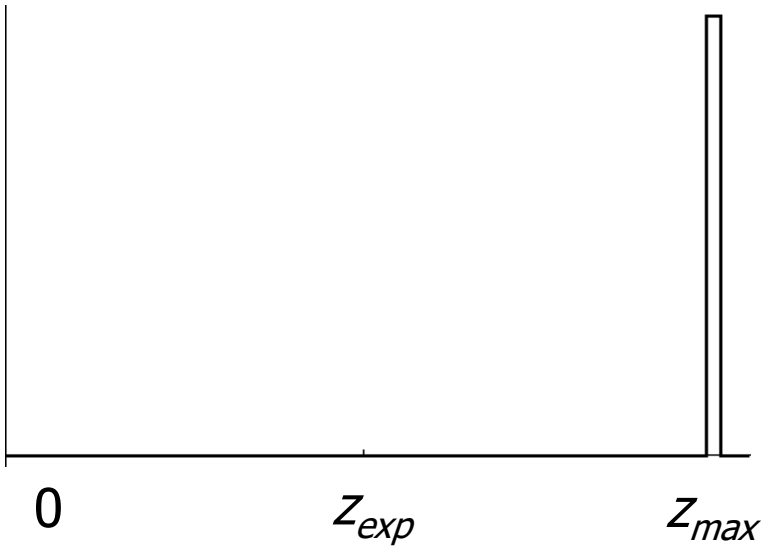
# Beam-based Proximity Model

Random measurement



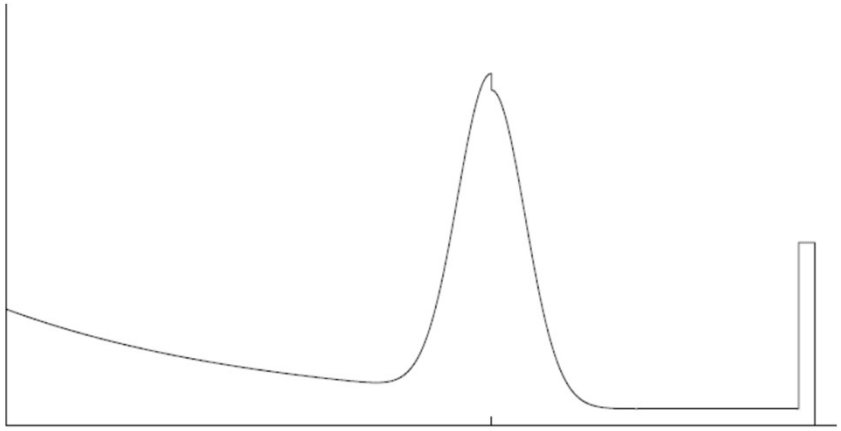

$$P_{rand}(z \mid x, m) = \eta \frac{1}{z_{\max}}$$

Max range




$$P_{\max}(z \mid x, m) = \eta \frac{1}{z_{small}}$$

# Resulting Mixture Density

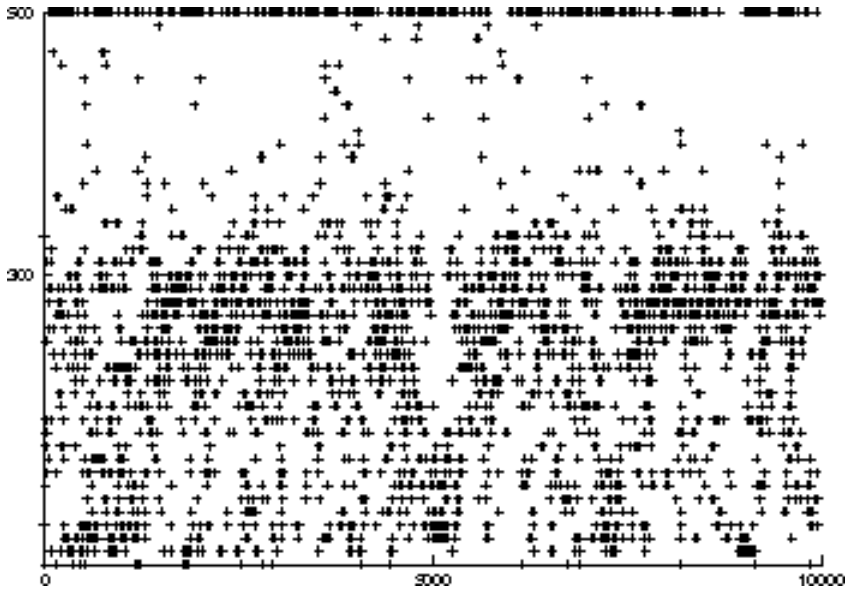


$$P(z \mid x,m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\text{max}} \\ \alpha_{\text{rand}} \end{pmatrix}^{T} \cdot \begin{pmatrix} P_{\text{hit}}(z \mid x,m) \\ P_{\text{unexp}}(z \mid x,m) \\ P_{\text{max}}(z \mid x,m) \\ P_{\text{rand}}(z \mid x,m) \end{pmatrix}$$

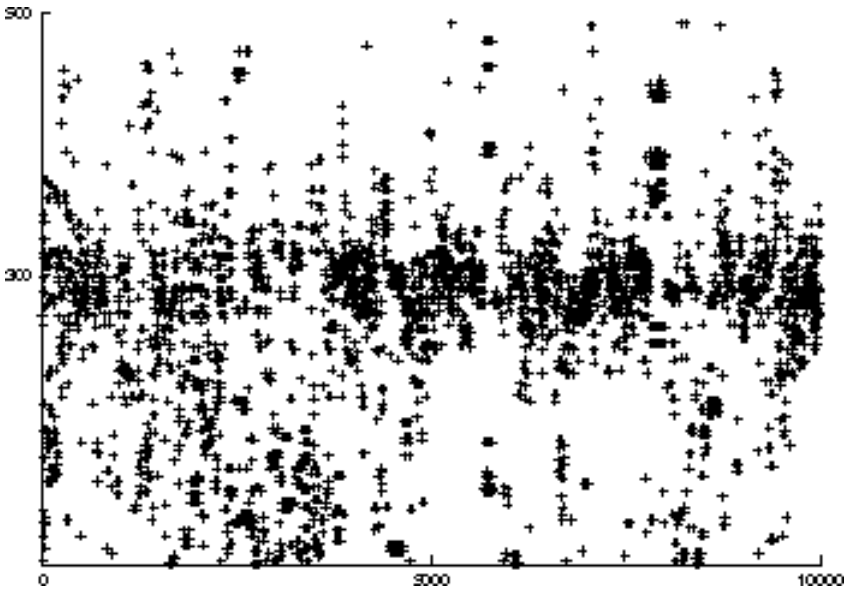
How can we determine the model parameters?

# Raw Sensor Data

Measured distances for expected distance of 300 cm.




Sonar




Laser

# Approximation

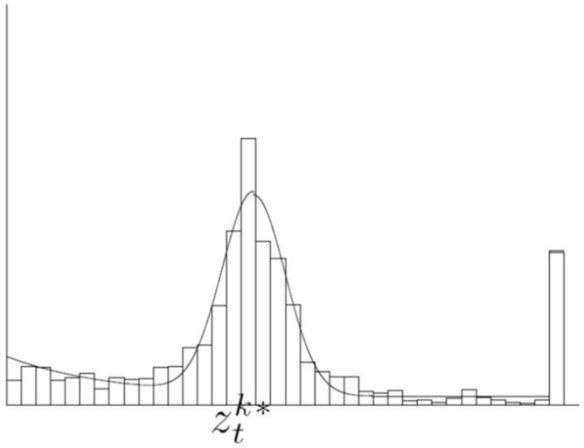
- Maximize log likelihood of the data

$$P(z \mid z_{\text{exp}})$$

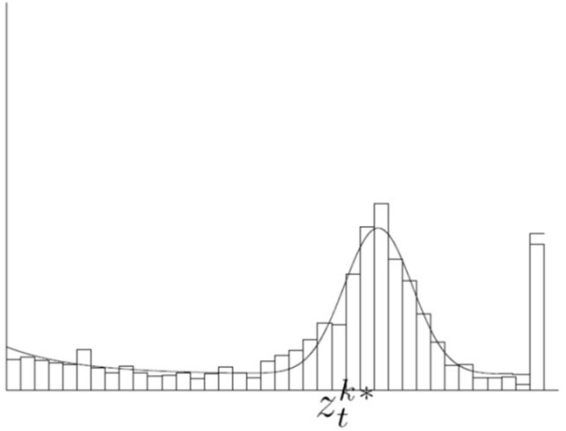
- Search space of n-1 parameters.
  - Hill climbing
  - Gradient descent
  - Genetic algorithms
  - …

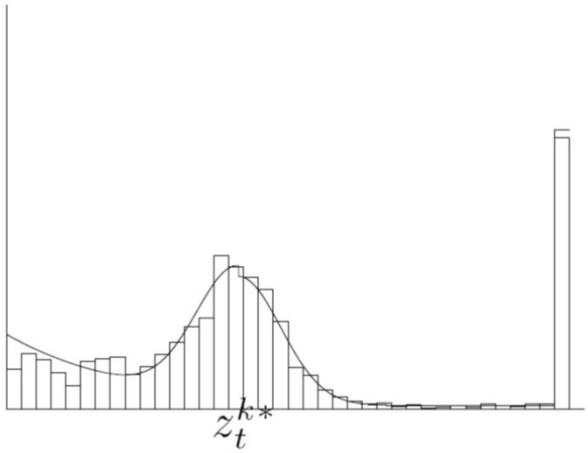
- Deterministically compute the n-th parameter to satisfy normalization constraint.

# Approximation Results

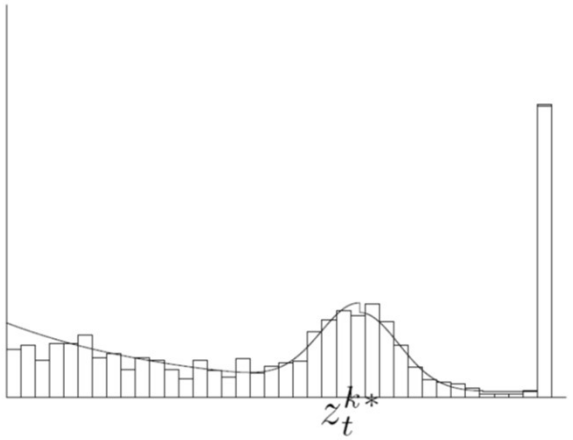



Laser



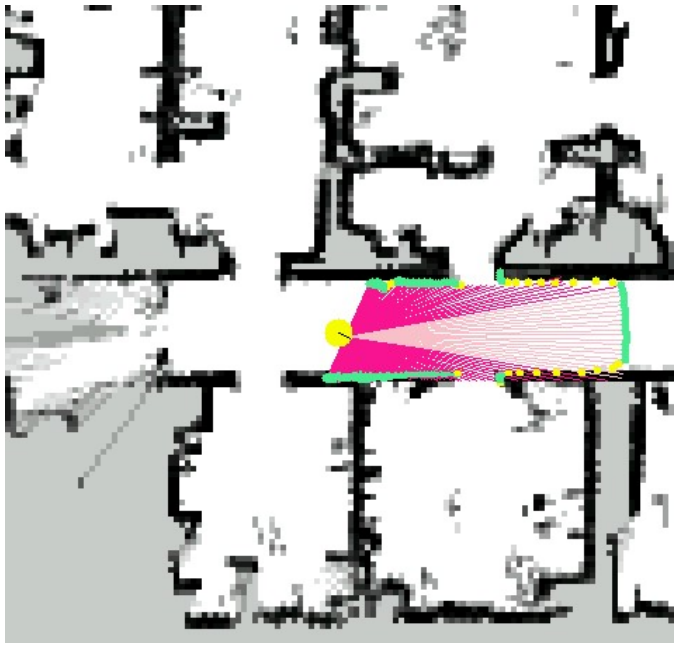

Sonar



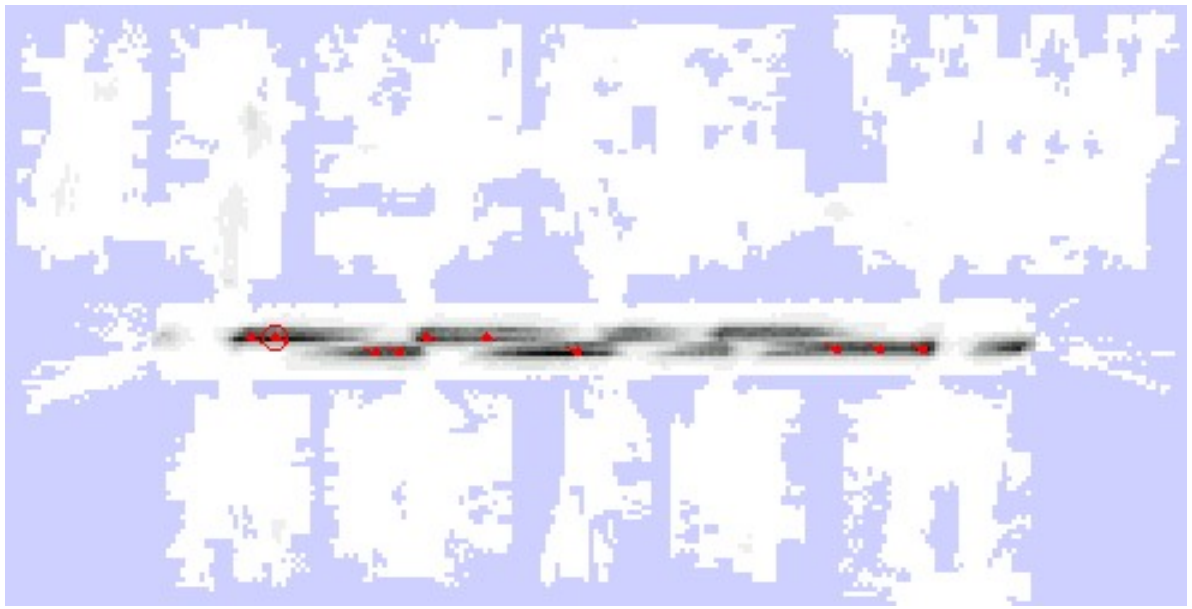

300cm

400cm

# Example



*z*



*P(z|x,m)*

# Scan-based Model

- Beam-based model is …
  - not smooth for small obstacles and at edges.
  - not very efficient.

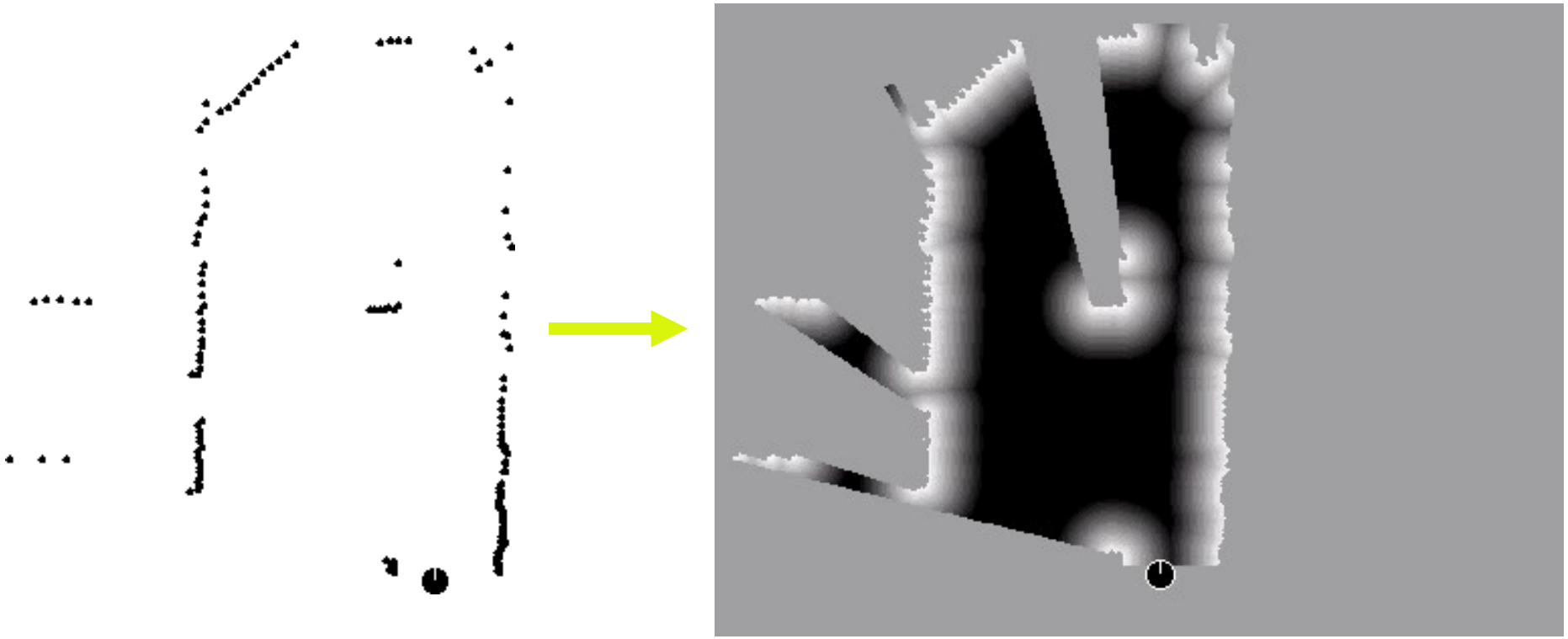
- Idea: Instead of following along the beam, just check the end point.

# Scan-based Model

- Probability is a mixture of …
  - a Gaussian distribution with mean at distance to closest obstacle,
  - a uniform distribution for random measurements, and
  - a small uniform distribution for max range measurements.
- Again, independence between different components is assumed.

# Scan Matching

- Extract likelihood field from scan and use it to match different scan.

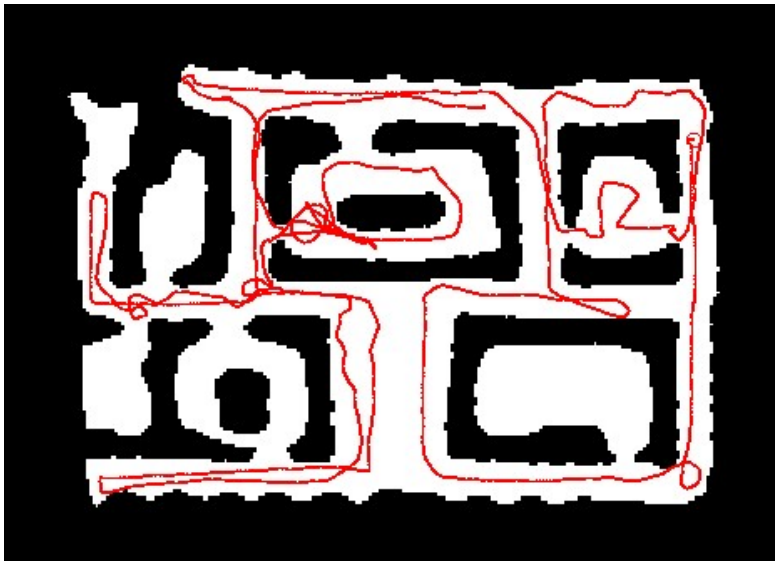


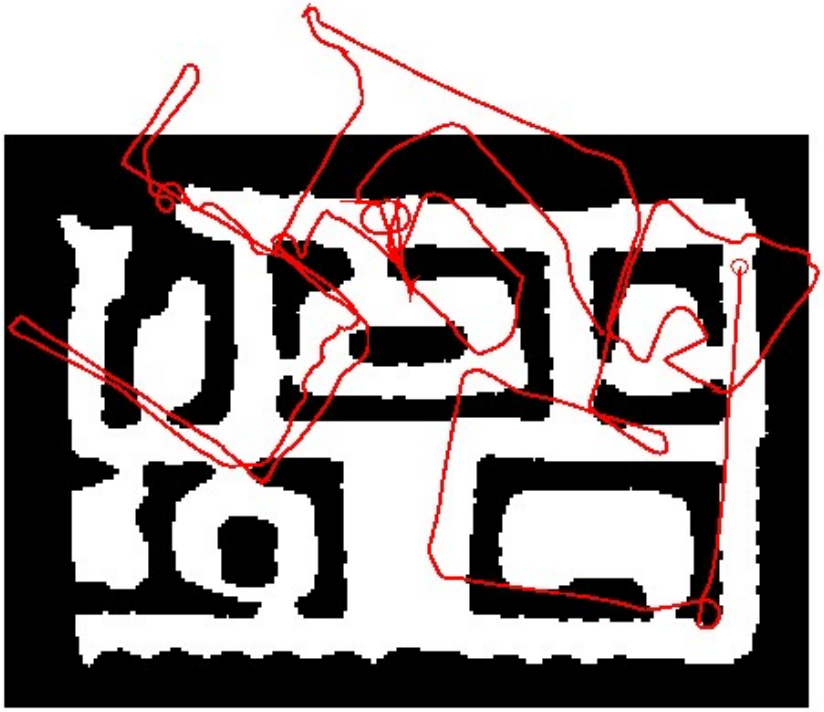
# Properties of Scan-based Model

- Highly efficient, uses 2D tables only.
- Smooth w.r.t. to small changes in robot position.
- Allows gradient descent, scan matching.
- Ignores physical properties of beams.

# Robot Motion

- Robot motion is inherently uncertain.
- How can we model this uncertainty?



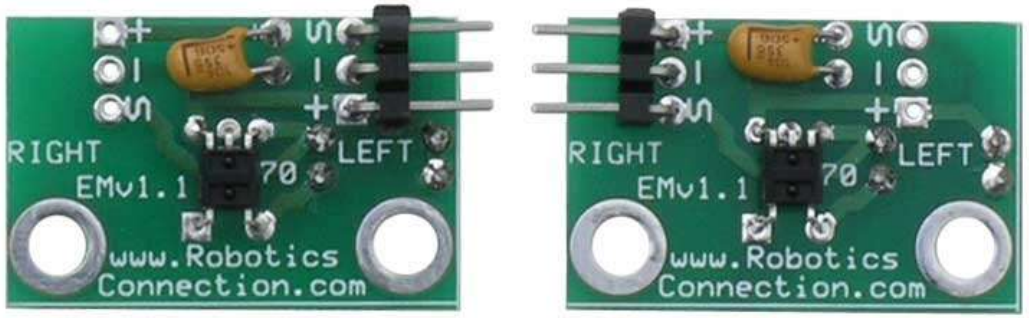


# Probabilistic Motion Model

- In practice, one often finds two types of motion models:
  - **Odometry-based**
  - **Velocity-based** (**dead reckoning**)
- Odometry-based models are used when systems are equipped with wheel encoders.
- Velocity-based models have to be applied when no wheel encoders are given.
- They calculate the new pose based on the velocities and the time elapsed.

# Example Wheel Encoders

These modules require +5V and GND to power them, and provide a 0 to 5V output. They provide +5V output when they "see" white, and a 0V output when they "see" black.



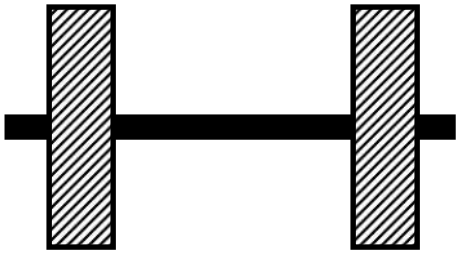

These disks are manufactured out of high quality laminated color plastic to offer a very crisp black to white transition. This enables a wheel encoder sensor to easily see the transitions.

Source: http://www.active-robots.com/

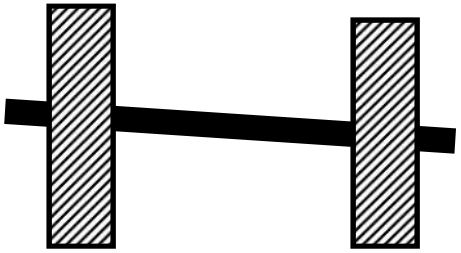
# Dead Reckoning

- Derived from “deduced reckoning.”
- Mathematical procedure for determining the present location of a vehicle.
- Achieved by calculating the current pose of the vehicle based on its velocities and the time elapsed.

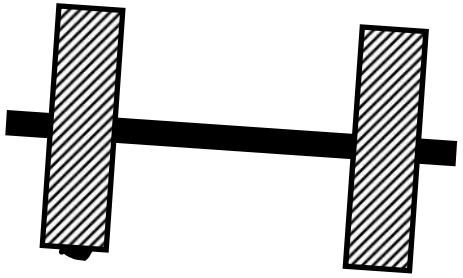
# Reasons for Motion Errors



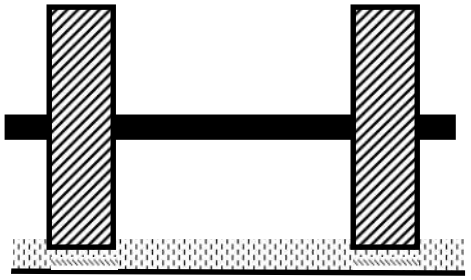
ideal case



different wheel diameters



bump



carpet

and many more …

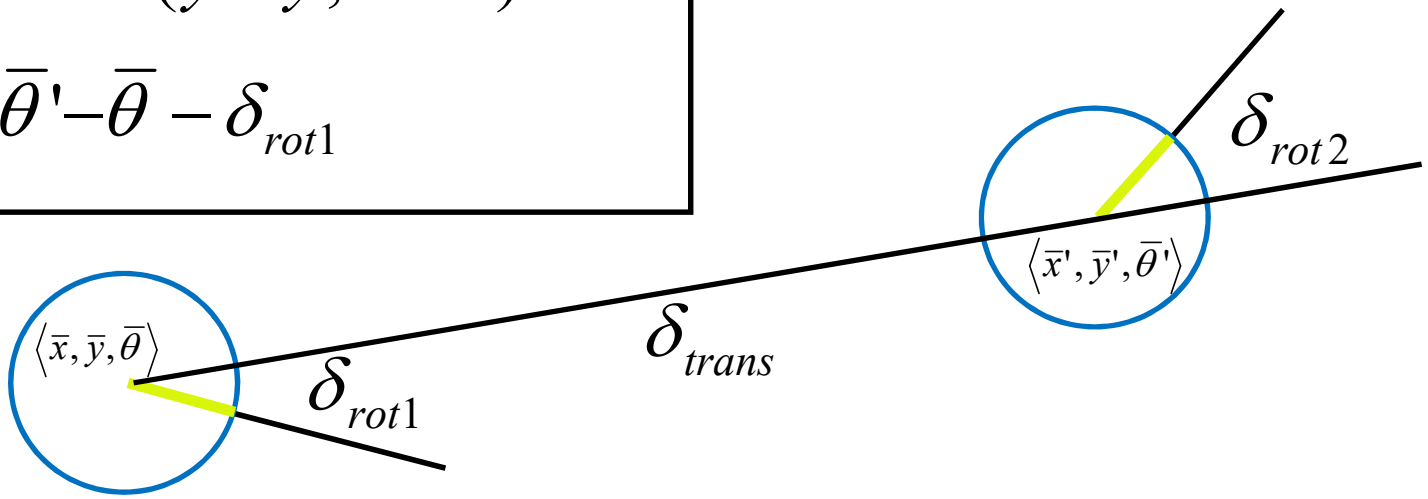
# Odometry Model

- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
- Odometry information $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

$$\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$$

$$\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$$

$$\delta_{rot2} = \bar{\theta}'-\bar{\theta} - \delta_{rot1}$$



# Noise Model for Odometry

- The measured motion is given by the true motion corrupted with noise.

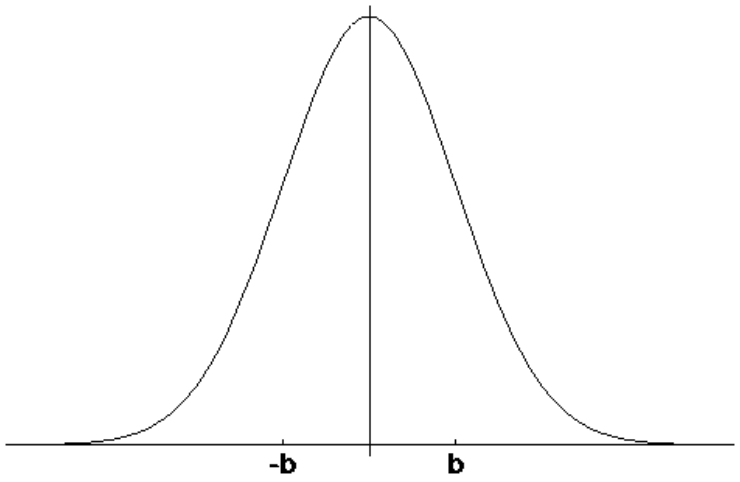
$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$

$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|}$$

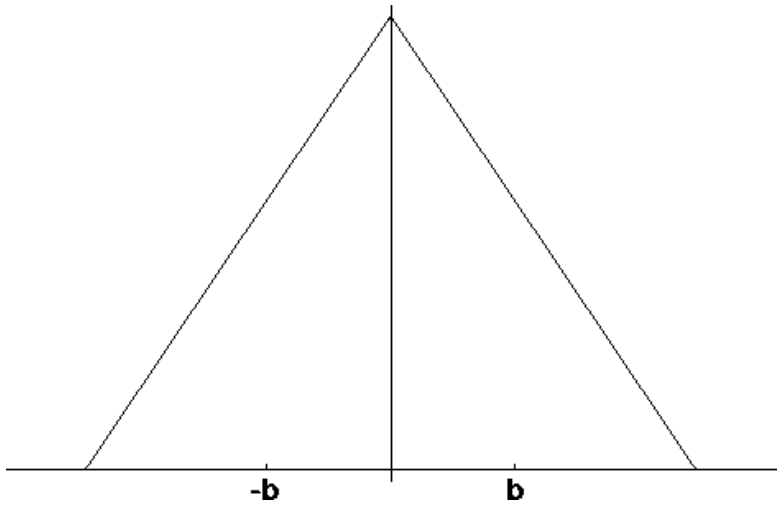
# Typical Distributions for Probabilistic Motion Models

Normal distribution




$$\varepsilon_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$$

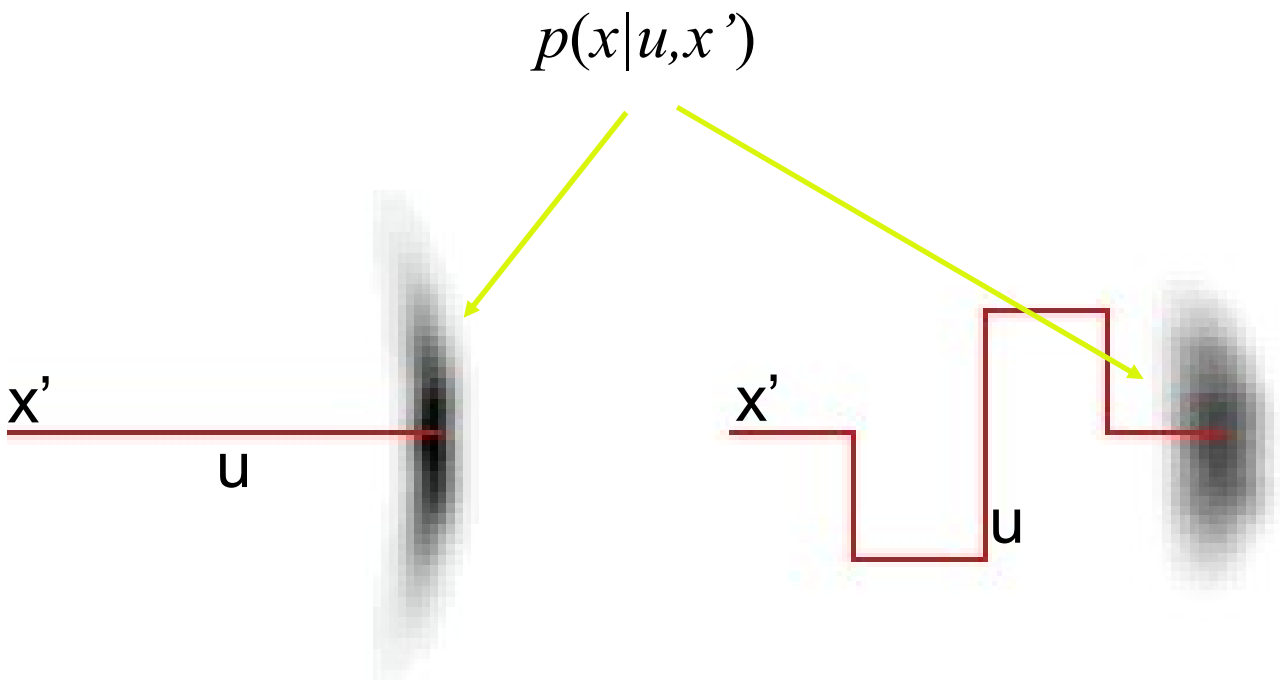
Triangular distribution



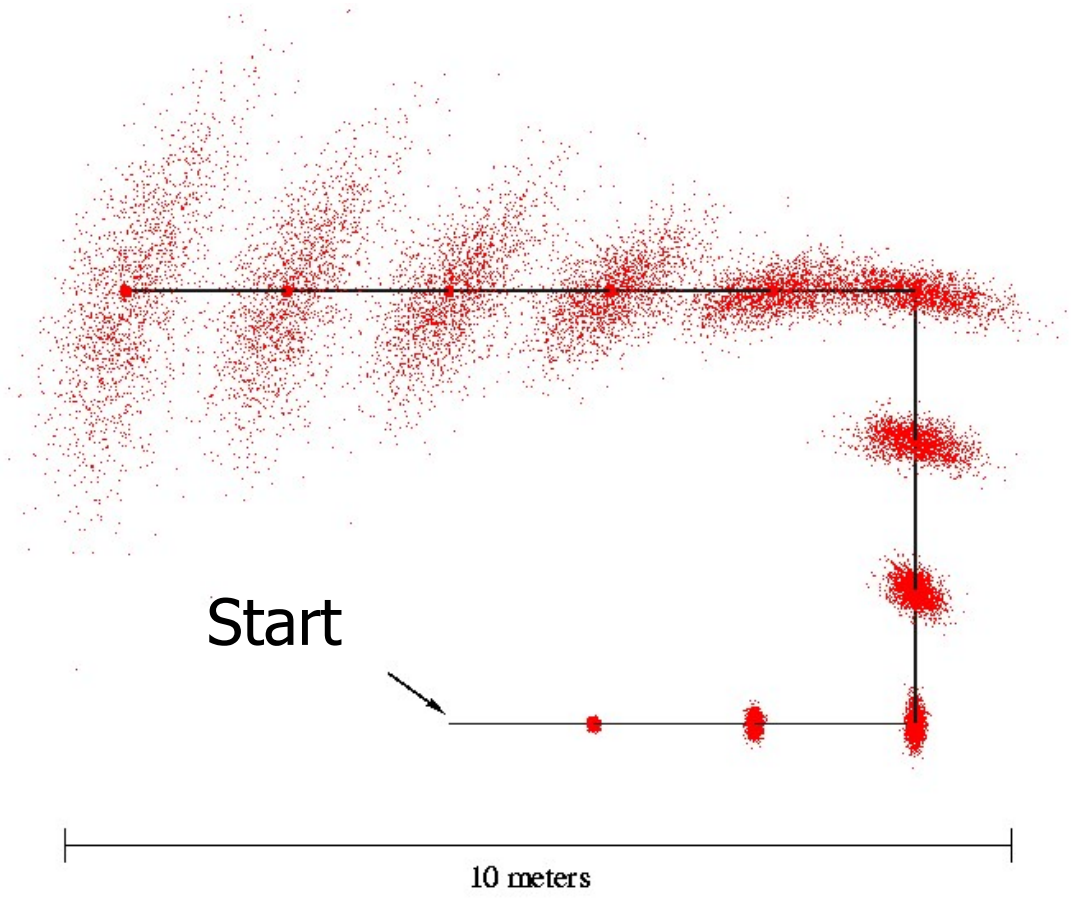

$$\varepsilon_{\sigma^2}(x) = \begin{cases} 0 \text{ if } |x| > \sqrt{6\sigma^2} \\ \dfrac{\sqrt{6\sigma^2} - |x|}{6\sigma^2} \end{cases}$$

# Application

- Repeated application of the sensor model for short movements.
- Typical banana-shaped distributions obtained for 2d-projection of 3d posterior.



# Sampling from Our Motion Model

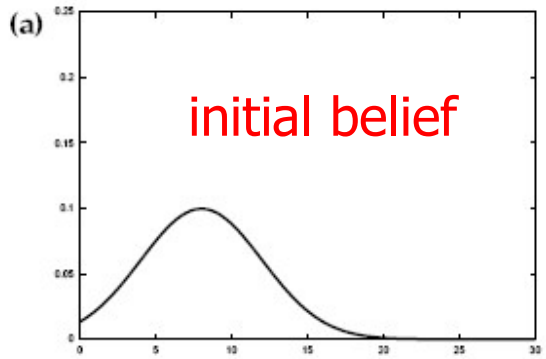


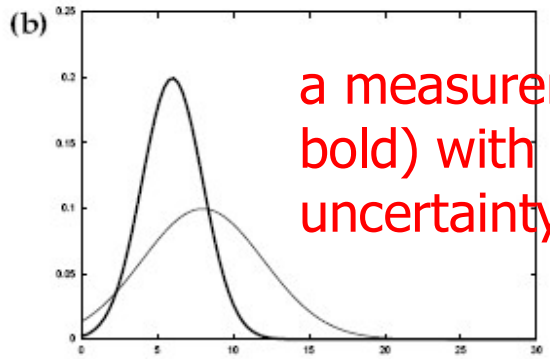
# Markov Localization's Implementation

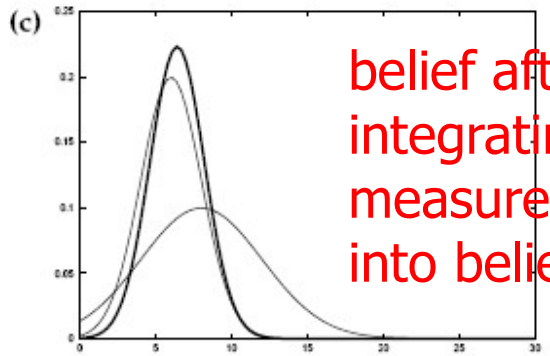
- Depending on how one chooses to represent density Bel($x$), various algorithms are available to implement Markov Localization:
  - Kalman filter
    - both the motion and the sensor model are described using a Gaussian density
    - maintains a single estimate of the robot's position
  - Grid-based Markov Localization
    - deal with multi-modal and non-Gaussian densities at a fine resolution
    - computational overhead vs. accuracy
  - Sampling-based methods
    - density is represented by a set of samples that are randomly drawn from it

# Implementation with Kalman Filter

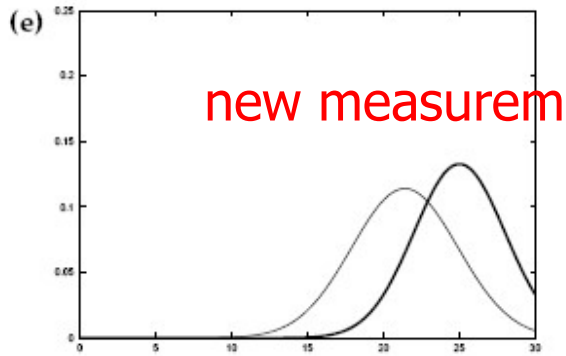
- KF maintains a single estimate of the robot's position

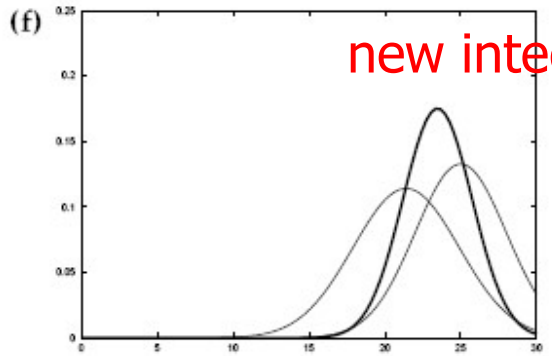


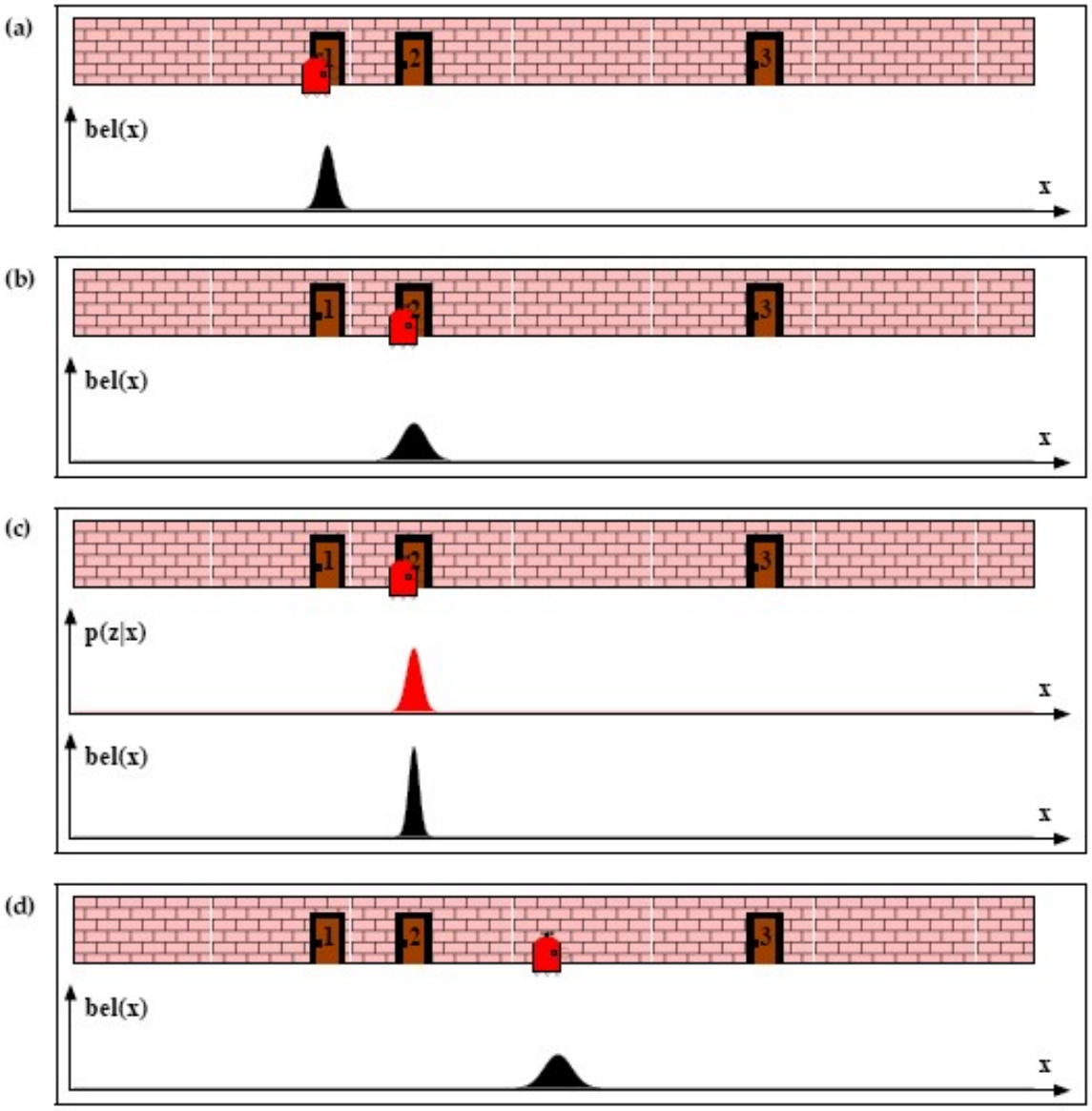







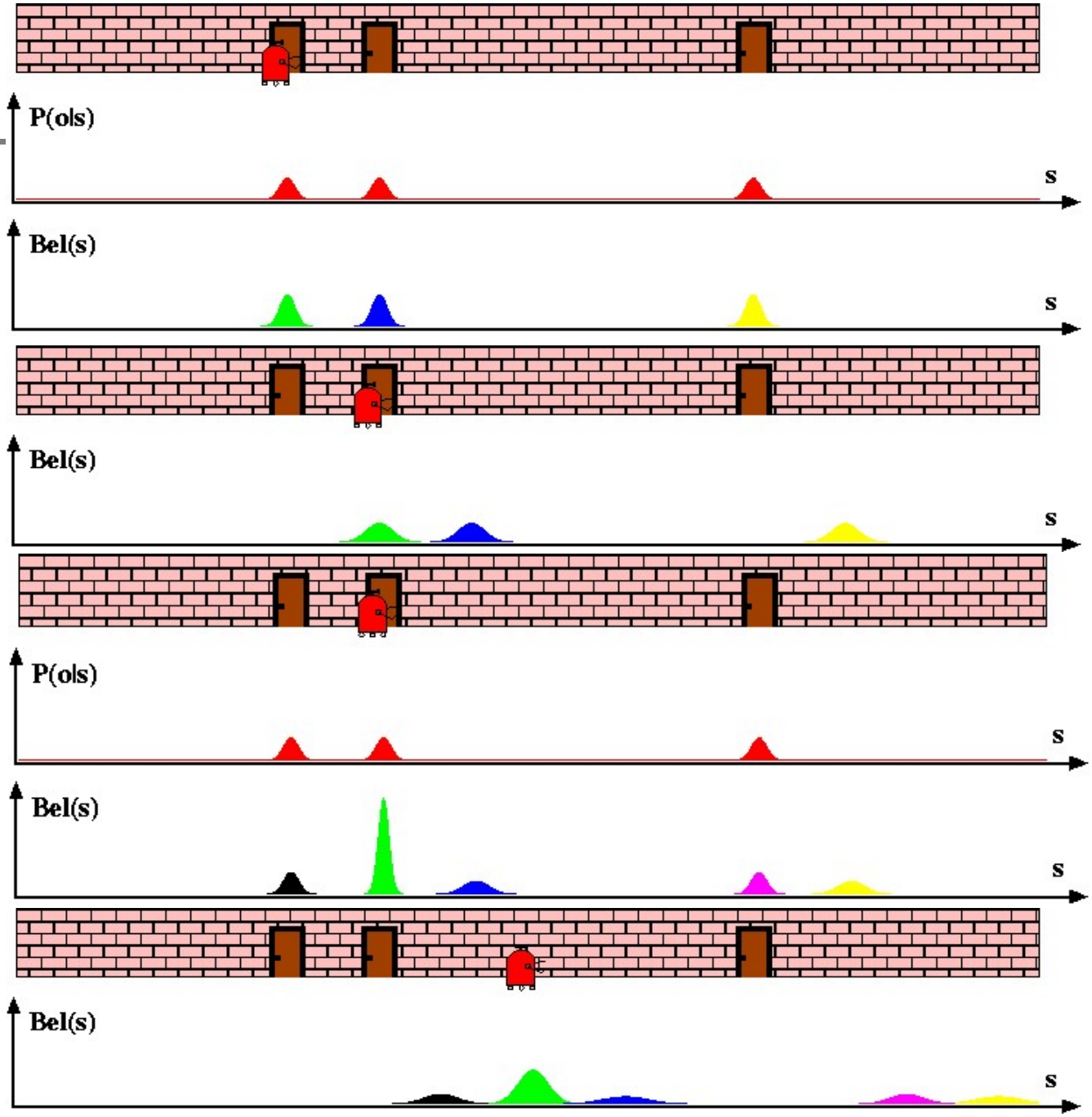



# Previous Example




- It's only applicable to position tracking problems
- It works well only if the position uncertainty is small
- Multi-hypothesis tracking will solve the problem at the cost of increased computational complexity

# Multi-Hypothesis Tracking Example



# Grid Localization

- Approximates posterior using a histogram filter over a grid decomposition of pose space
- A common granularity in indoor environment is 15cm for x- and y-dimensions, and 5 degrees for rotational dimension
- Grid resolution:
  - Coarse: topological representation
  - Fine: metric representation

# Coarse-Grained

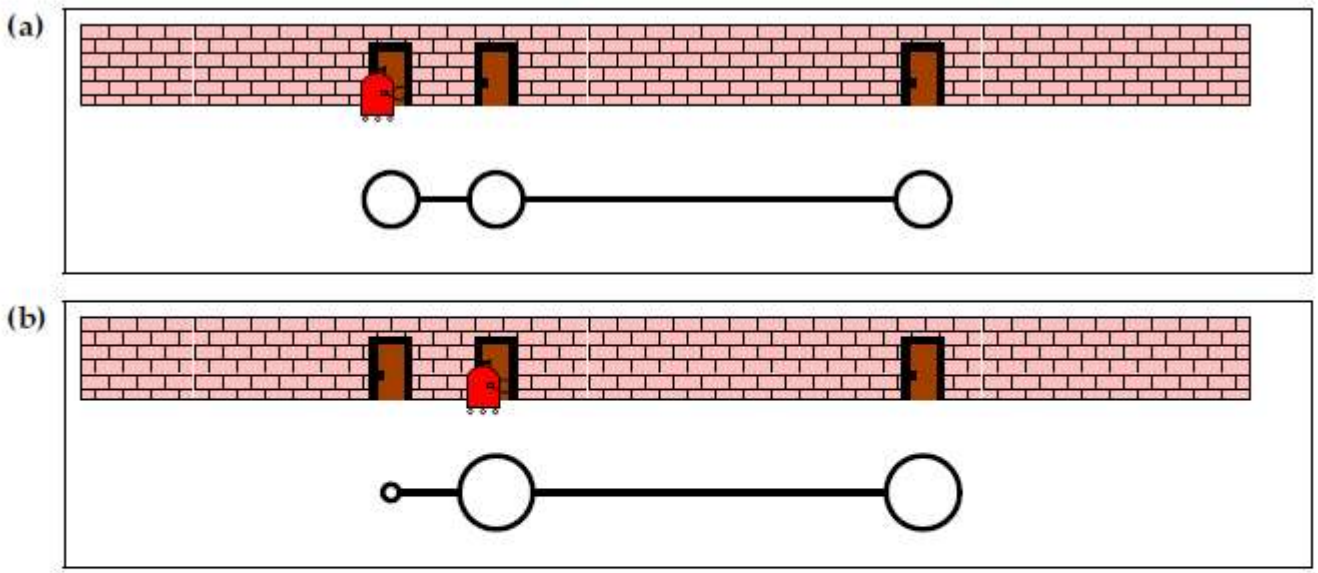



**Figure 8.5** Application of a coarse-grained, topological representation to mobile robot localization. Each state corresponds to a distinctive place in the environment (a door in this case). The robot's belief $bel(x_t)$ of being in a state is represented by the size of the circles. (a) The initial belief is uniform over all poses. (b) shows the belief after the robot made one state transition and detected a door. At this point, it is unlikely that the robot is still in the left position.

# Grid with Fixed-Resolution

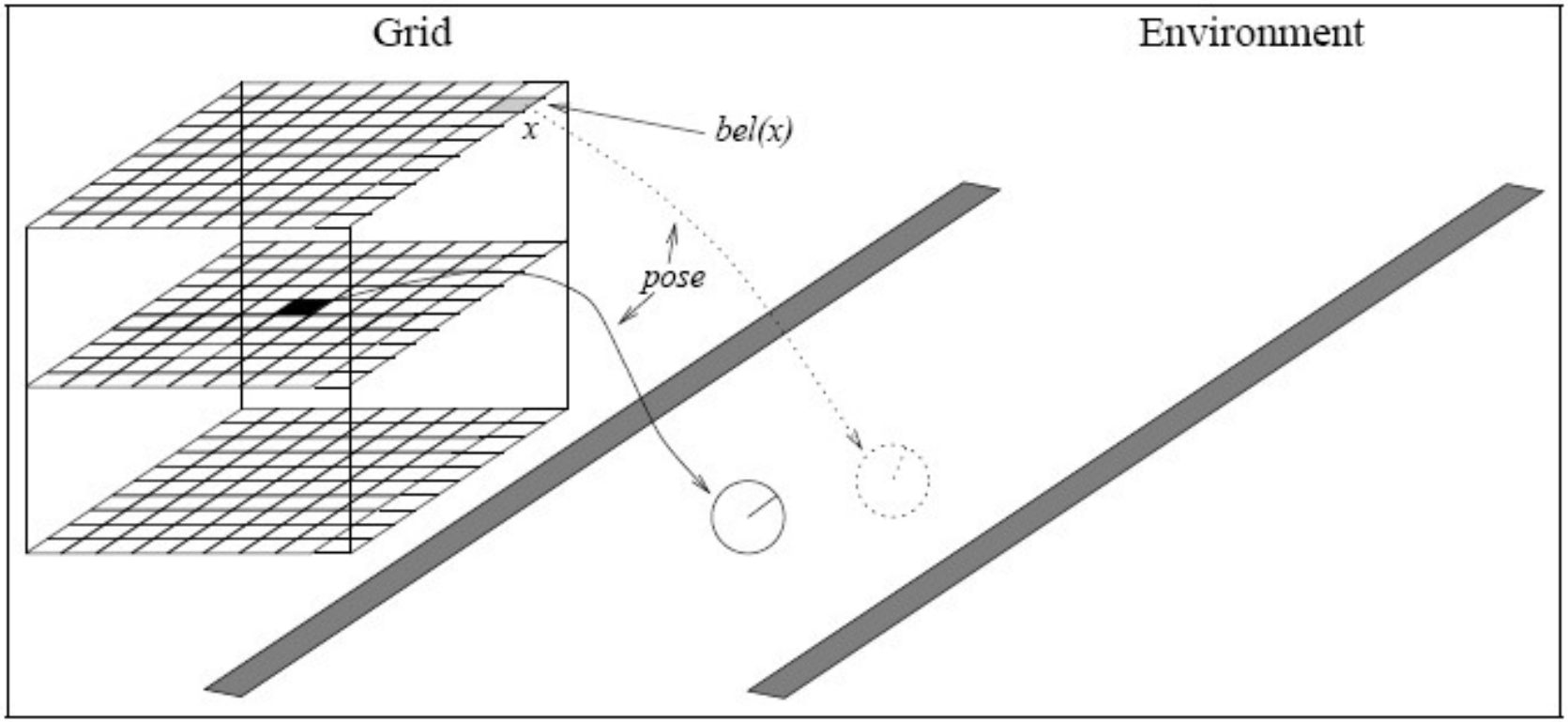
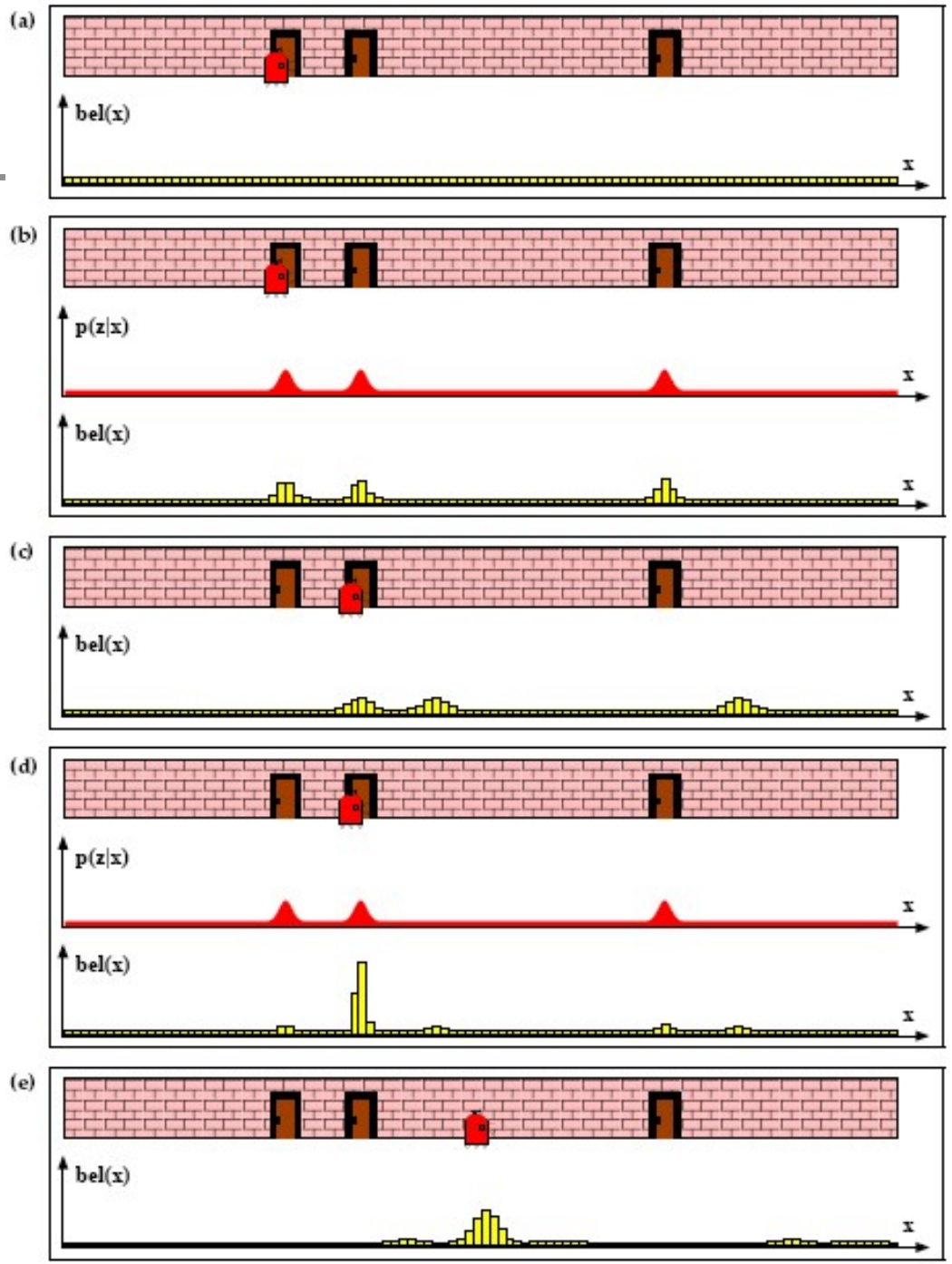



**Figure 8.2** Example of a fixed-resolution grid over the robot pose variables $x$, $y$, and $\theta$. Each grid cell represents a robot pose in the environment. Different orientations of the robot correspond to different planes in the grid (shown are only three orientations).

# Example



# Discrete Bayes Filter Algorithm

1. Algorithm **Discrete_Bayes_filter**( *Bel(x),d* ):
2. $\eta$=0
3. If *d* is a perceptual data item *z* then
4. For all *x* do
5. $Bel'(x) = P(z \mid x) Bel(x)$
6. $\eta = \eta + Bel'(x)$
7. For all *x* do
8. $Bel'(x) = \eta^{-1} Bel'(x)$
9. Else if *d* is an action data item *u* then
10. For all *x* do
11. $Bel'(x) = \sum_{x'} P(x \mid u, x')\ Bel(x')$
12. Return *Bel'(x)*

# Implementation (1)

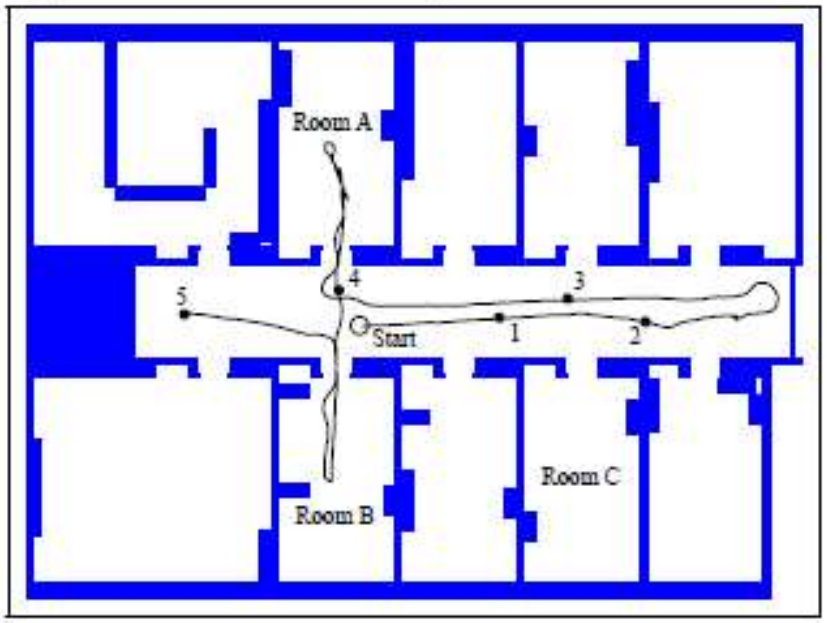
- To update the belief upon sensory input and to carry out the normalization one has to iterate over all cells of the grid.
- Especially when the belief is peaked (which is generally the case during position tracking), one wants to avoid updating irrelevant aspects of the state space.
- One approach is not to update entire sub-spaces of the state space.
- This, however, requires to monitor whether the robot is de-localized or not.
- To achieve this, one can consider the likelihood of the observations given the active components of the state space.

# Implementation (2)

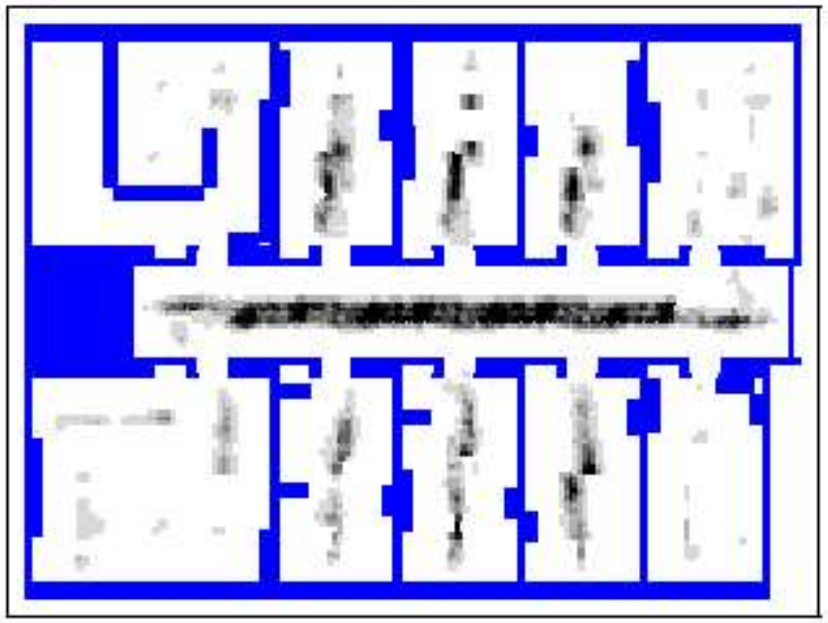
- To efficiently update the belief upon robot motions, one typically assumes a bounded Gaussian model for the motion uncertainty.
- This reduces the update cost from $O(n^2)$ to $O(n)$, where $n$ is the number of states.
- The update can also be realized by shifting the data in the grid according to the measured motion.

# Markov Localization (Grid)

(a) Path and reference poses

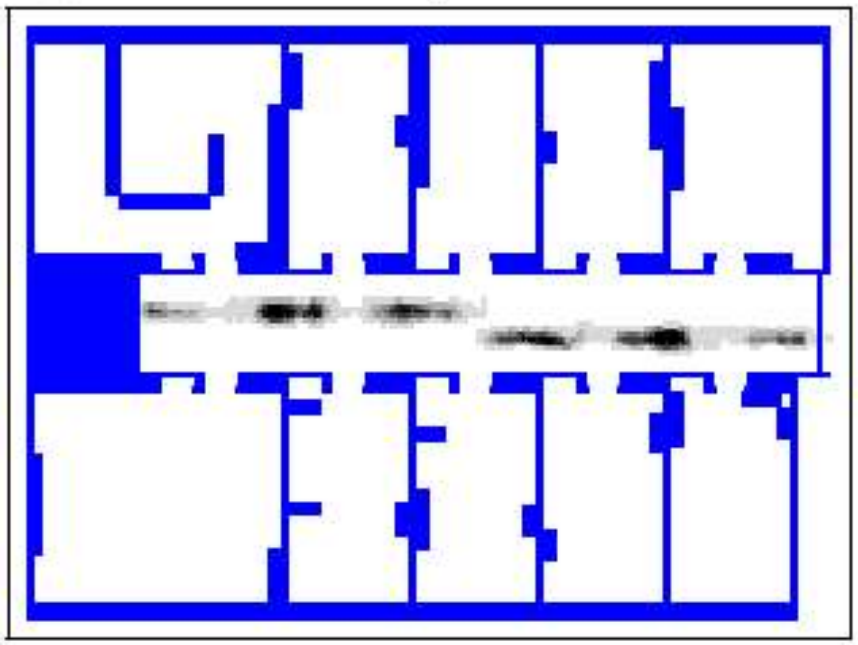



(b) Belief at reference pose 1

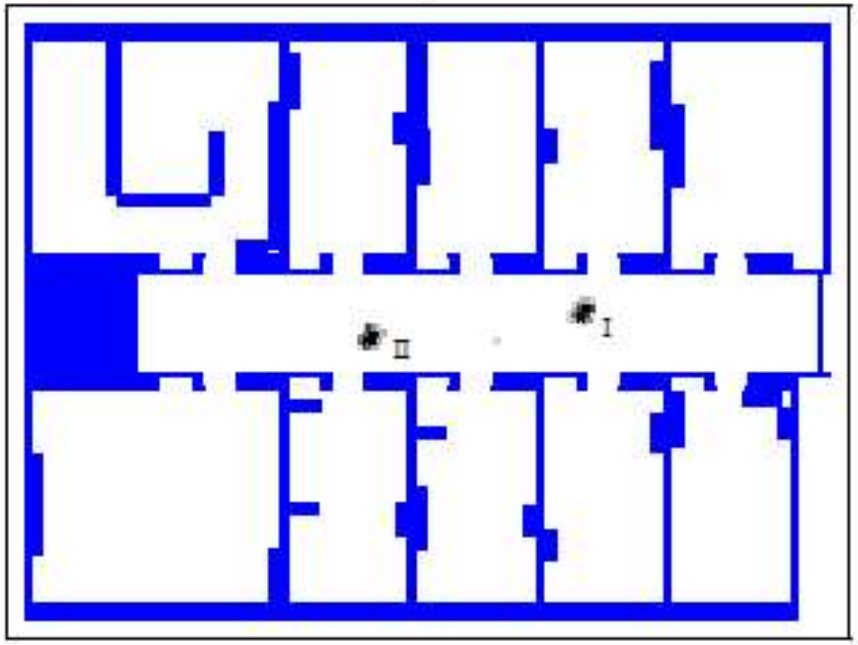


# Markov Localization (Grid)

(c) Belief at reference pose 2

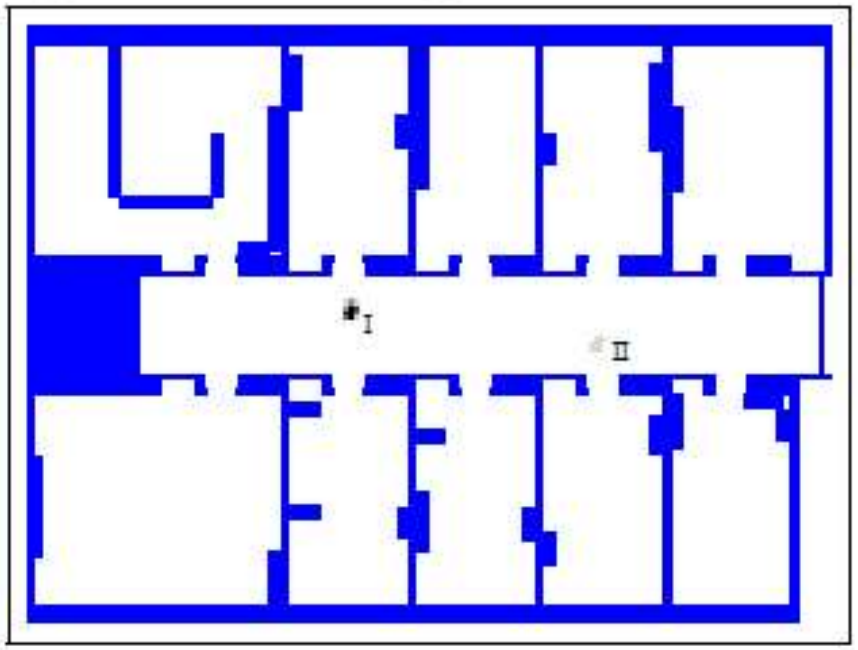


(d) Belief at reference pose 3

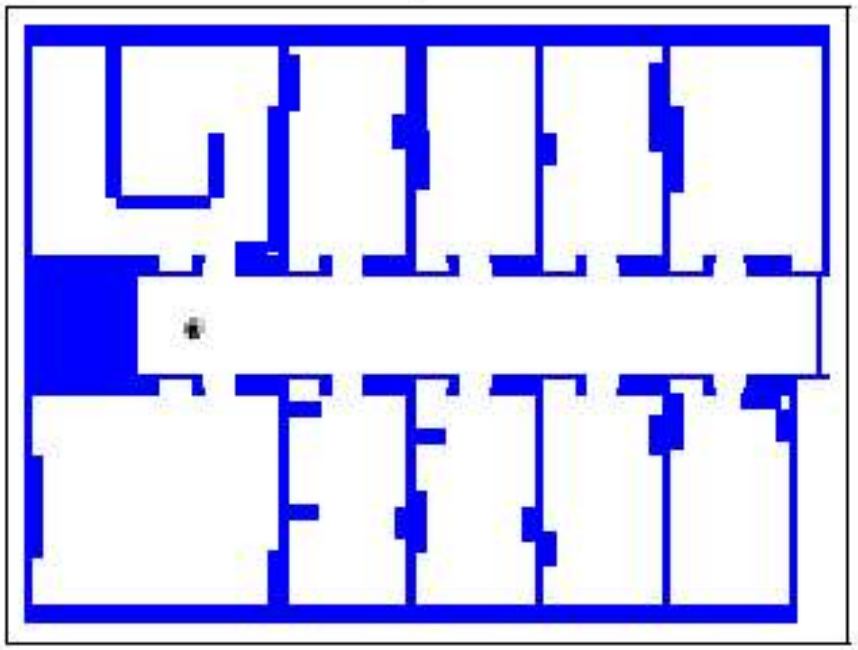


# Markov Localization (Grid)

(e) Belief at reference pose 4




(f) Belief at reference pose 5



# Sampling Based Method

- In which, we represent bel(x) by particles
- Estimation of non-Gaussian, nonlinear processes

- This algorithm is called Monte Carlo Localization (MCL)

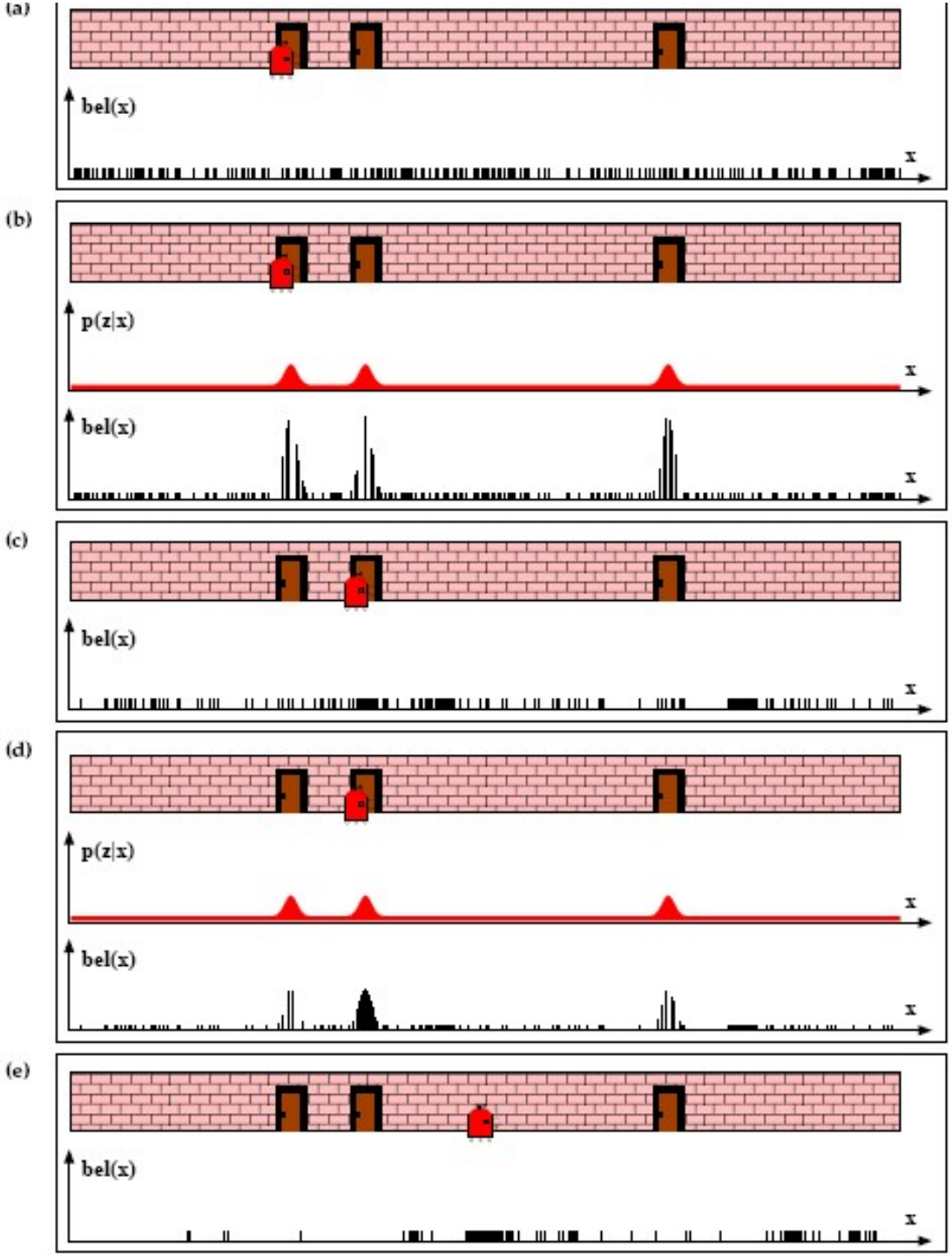
- It's applicable to both local and global localization problems

# Example

M population of particles

Each particle has an importance weight

Increased # of particles around locations



# Properties of MCL

- Not bound to a limited subset of distributions
- Increasing # of particles (M) increases accuracy
  - Trade off: accuracy vs. computation
  - Common strategy: keep sampling until the next pair (u, z) has arrived
  - M needs to be high enough

# "Localize" Proxy in Player/Stage

- The LocalizeProxy class is used to control a localize device, which can provide multiple pose hypotheses for a robot
  - Allows you to localize the robot using its sensors, such as laser, sonar, or even radio strength of signal
  - Localization drivers will estimate the pose of the robot by comparing observed sensor readings against a pre-defined map of the environment
  - E.g., amcl driver, this driver implements the Adaptive Monte-Carlo Localization (AMCL) algorithm described by Dieter Fox
  - "Monte Carlo Localization for Mobile Robots" by F. Dellaert, D. Fox, W. Burgard and S. Thrun, *AAAI* '99.

# AMCL Driver in Player/Stage

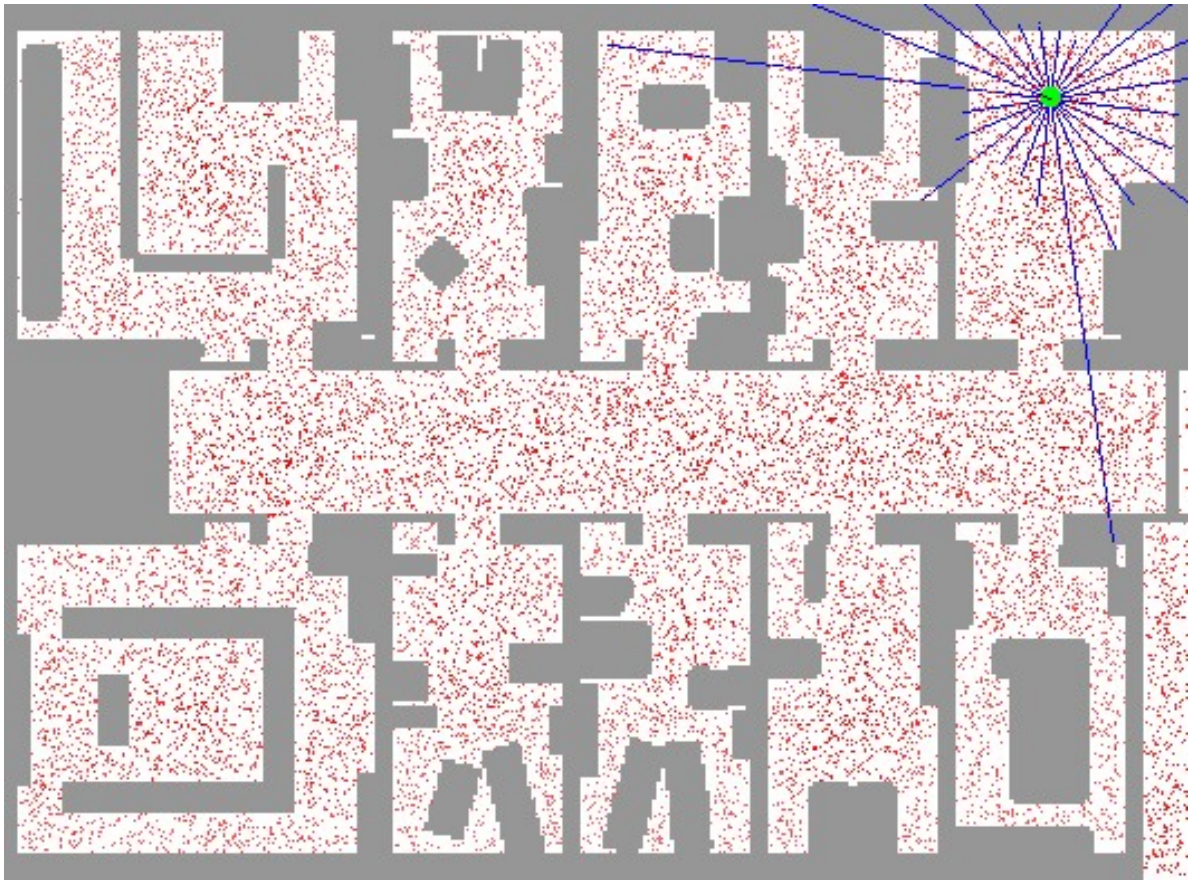
- The amcl driver maintains a probability distribution over the set of all possible robot poses, and updates this distribution using data from odometry, sonar and/or laser.

- Probability distribution is represented by a particle filter:
  - Adaptive: the number of particles can be dynamically adjusted
    - pose is uncertain → # of particles increases
    - pose is determined → # of particles decreases
    - tradeoff between speed and accuracy

- Features of simple MCL techniques:
  - unknown robot initial pose → usually converge to correct pose
  - incorrect robot initial pose, or robot becomes lost → will not converge to correct pose

# Monte Carlo Localization

- Developed by Fox, Burgard, Dellaert, Thrun (AAAI'99 article)



"Monte Carlo" refers to techniques that are stochastic / random / non-deterministic

http://robots.stanford.edu/movies/sca80a0.avi

# Adapting the Size of the Sample Set

- # of samples varies based on different situations:
  - During global localization: unknown robot's initial position → need lots of samples
  - During position tracking: robot's uncertainty is small → don't need as many samples

- MCL determines sample size "on the fly"
  - Compare P($x$) and P($x|z$) (i.e., belief before and after sensing) to determine sample size
  - The more divergence, the more samples that are kept

# More Movies from Dieter Fox


# Summary

- Robot localization using probabilistic models
  - Sensor models
  - Motion models
  - Markov localization: 3 different ways of implementation