Localization

USI Robotics

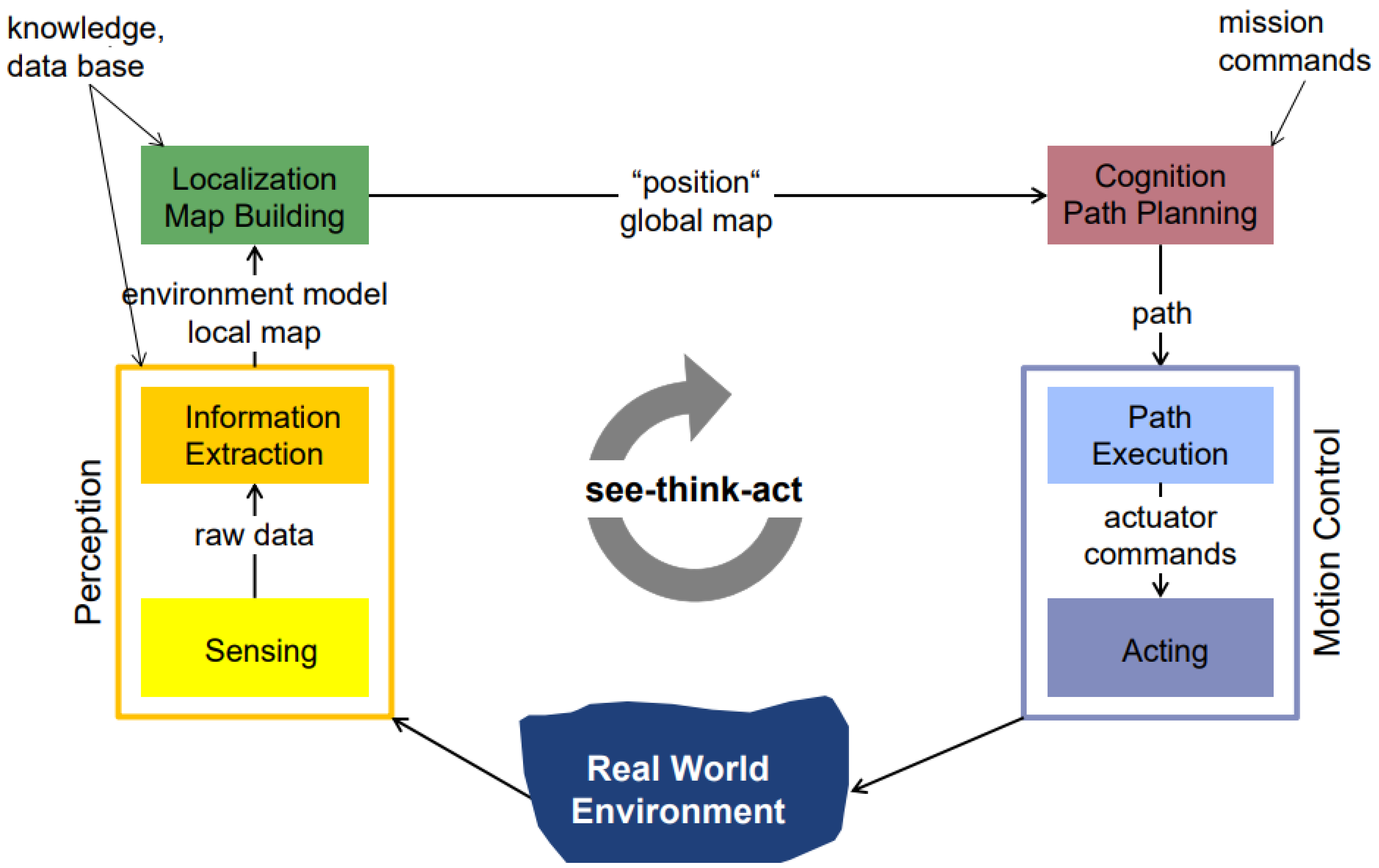
Alessandro Giusti, IDSIA

Slides credit:

- Roland Siegwart, Margarita Chli, Juan Nieto, Nick Lawrance, ETH
- Sebastian Thrun: http://www.probabilistic-robotics.org/
- Lynne Parker: http://web.eecs.utk.edu/~leparker

References:

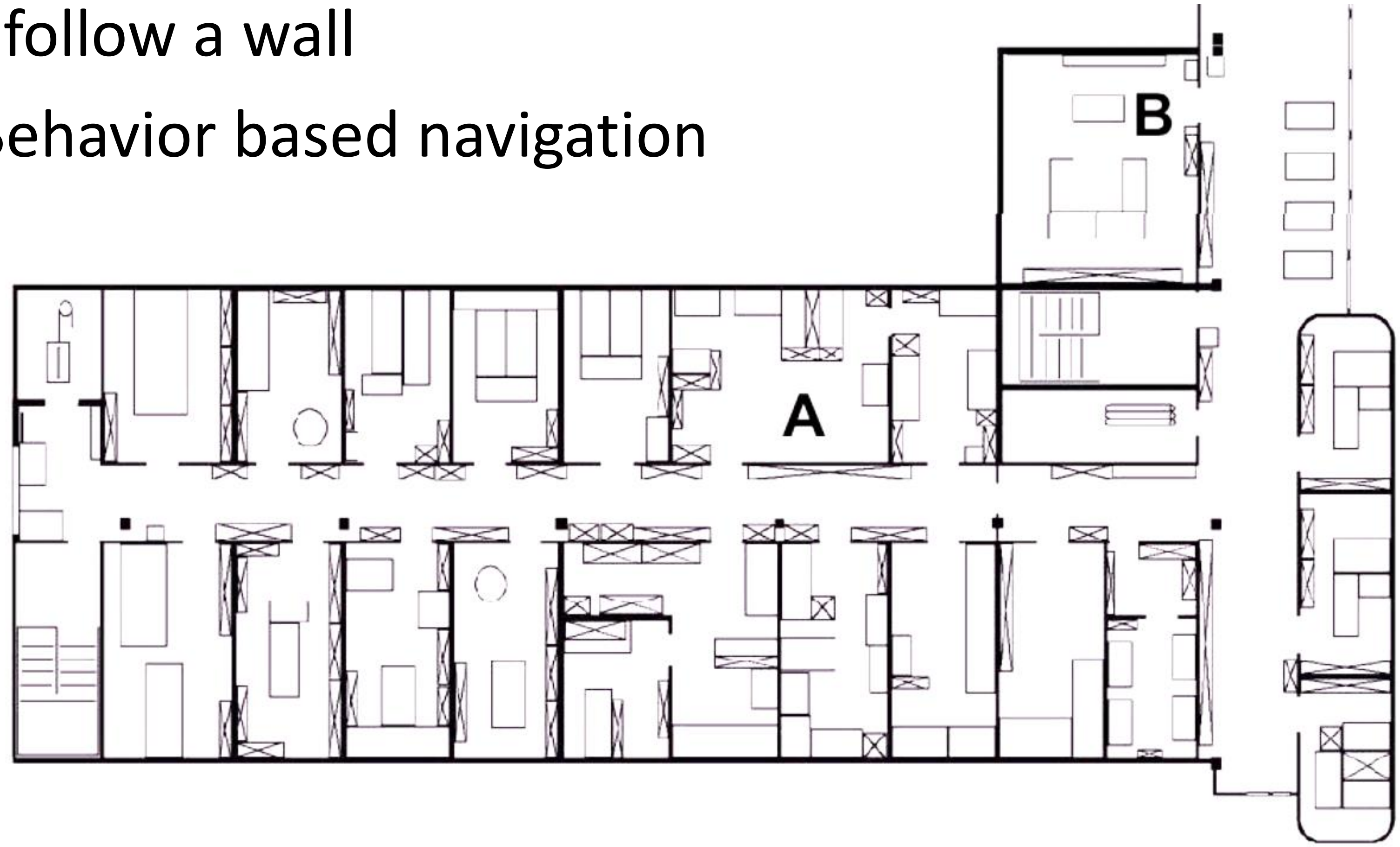
- Ben-Ari, Mondada: Elements of Robotics (Chapter 8) https://link.springer.com/book/10.1007%2F978-3-319-62533-1
- Siegwart et al: Autonomous Mobile Robots (Chapter 5)


knowledge, data base
mission commands
Localization Map Building
"position" global map
Cognition Path Planning
environment model local map
path
Perception
Information Extraction
raw data
Sensing
see-think-act
Path Execution
actuator commands
Acting
Motion Control
Real World Environment

We don’t always need localization

To go from A to B, a robot may just follow a wall

→ Behavior based navigation



Map-based Localization

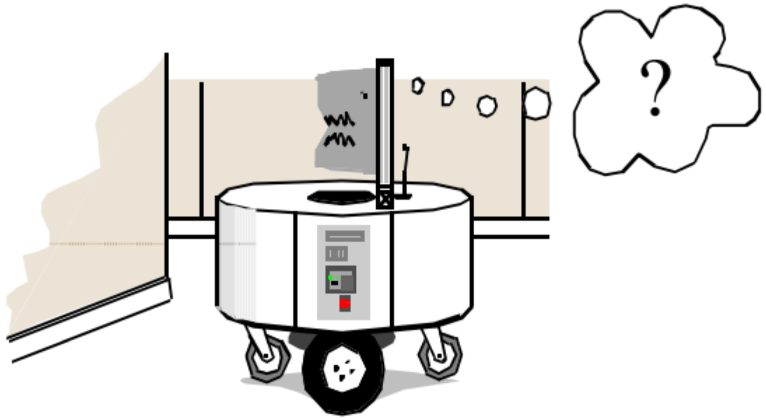
Assume the map is known. Where am I on the map?

- How to solve this?
- How to represent the map?
- How to represent the position of the robot in the map?

Definitions

Global Localization
The robot is not told its initial position, its position must be estimated from scratch. (*Kidnapped robot problem*)

Position Tracking
A robot knows its initial position and only has to compensate for odometry errors



How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where in the map the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where in the map the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

We focus on these (real autonomy without requiring infrastructure)

How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where in the map the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

Why not just onboard sensors?

Why not just onboard sensors?

Sensor noise / errors

- due to environment
  - e.g. mirrors confuses laser sensor
  - multipath echos confuse sonars
  - differently-colored walls affect Thymio proximity sensors
- due to sensing modality
  - inherent imprecision
  - e.g. interference between sonars

Aliasing

Non-uniqueness of sensors readings is the norm for robots (not for humans)! Even with multiple sensors, there is a many-to-one mapping from poses to robot perceptual inputs.

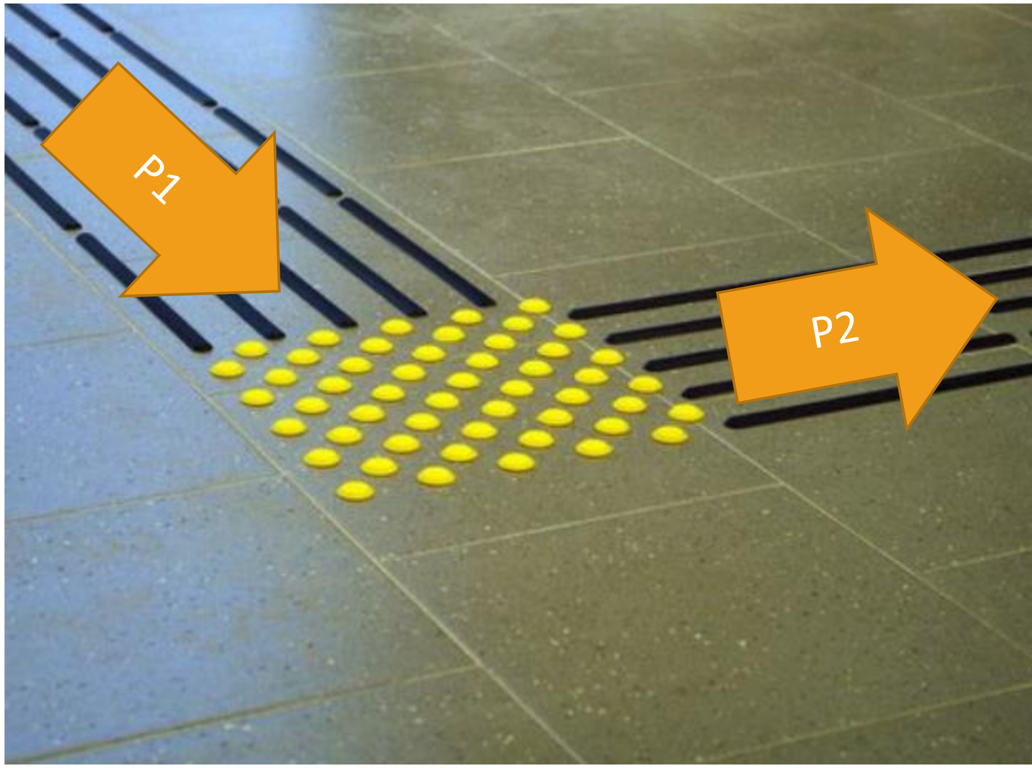
The amount of information perceived by the sensors is generally insufficient to identify the robot's position from a single reading.

Localization is usually based on a series of readings: sufficient information is recovered by the robot over time

Aliasing example

If my only sensor is my feet and I just woke up, I have no way to know if I am in pose P1 or in pose P2, even if I have the map.

But if I start walking I will orient myself by combining odometry and new sensor readings.



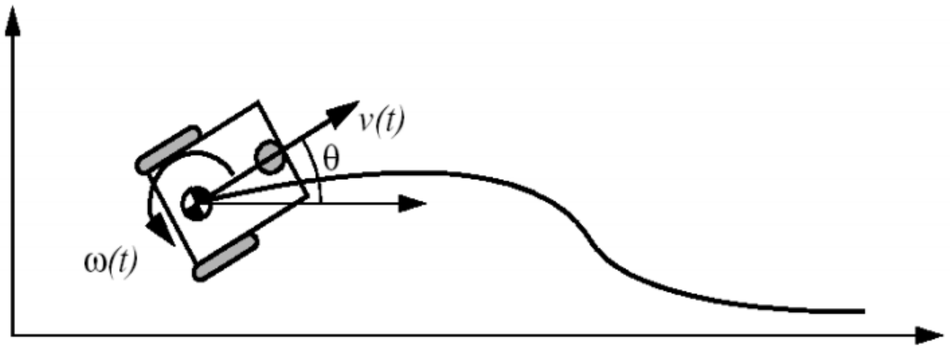
How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where in the map the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

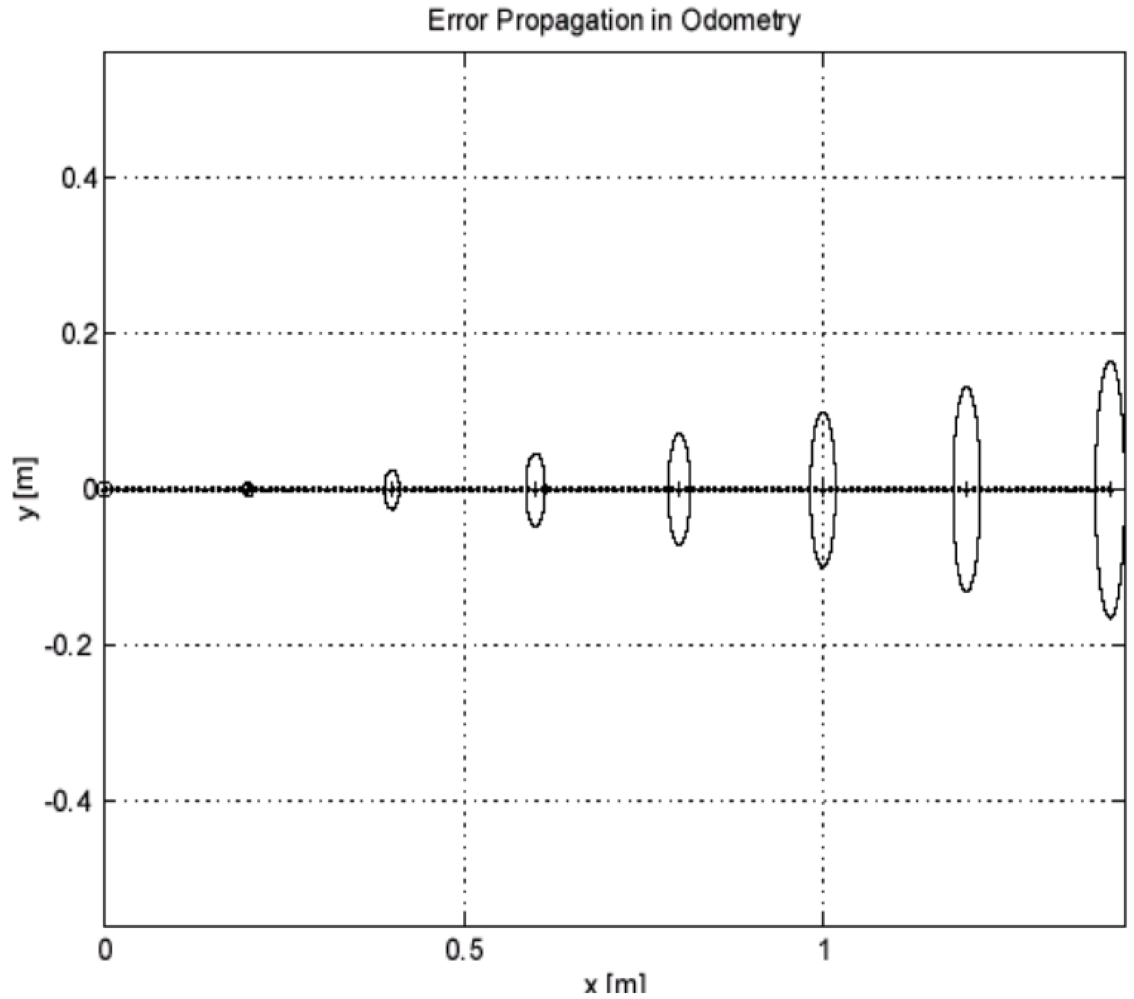
Why not just odometry?

Why not just odometry?

- Consider a differential drive robot where the speed of each wheel is measured with some error
  - Errors for left and right wheels are independent
- Robot motion is recovered by integrating proprioceptive sensor velocity readings (wheel encoders)

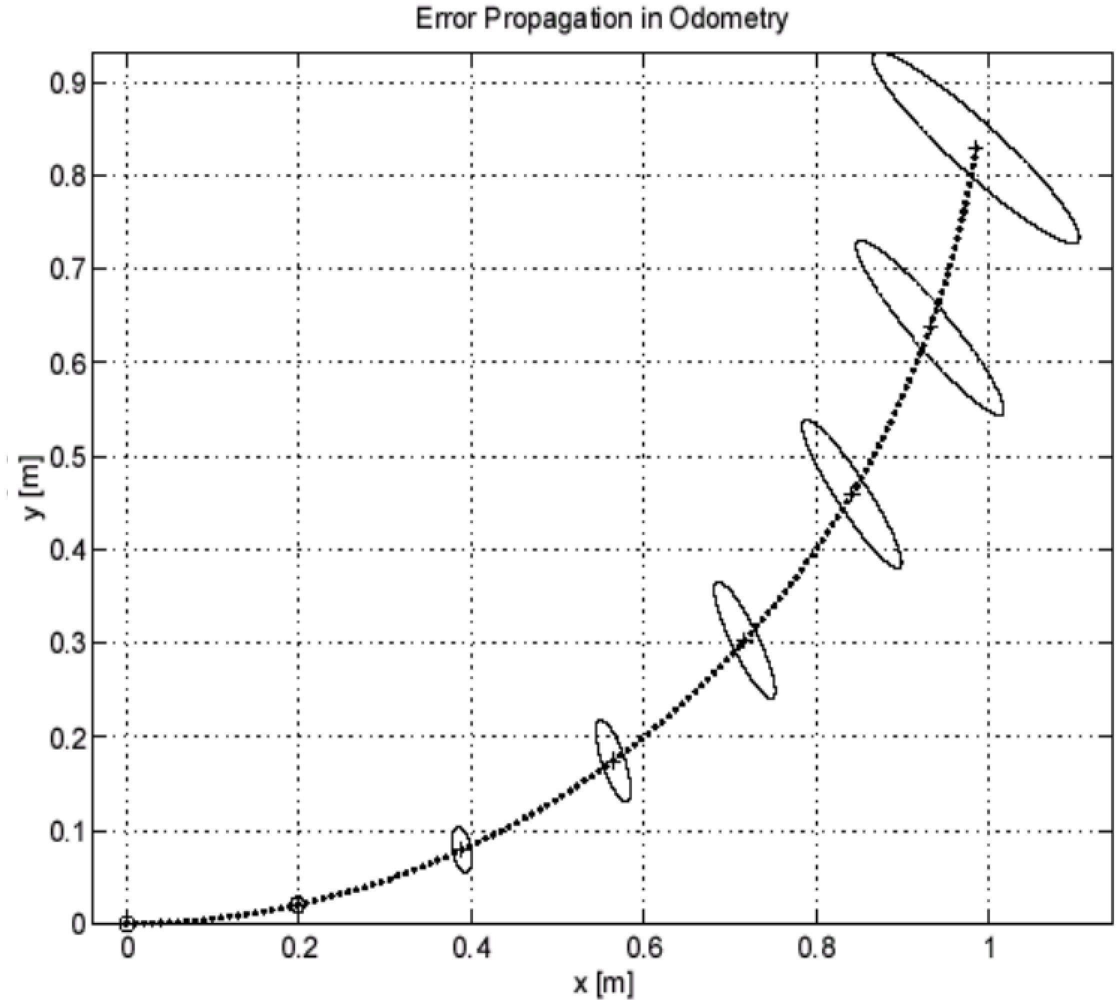


Odometry: Growth of Pose Uncertainty for Straight Motion



ellipses represent uncertainty in position

Odometry: Growth of Pose Uncertainty for Curvilinear Motion



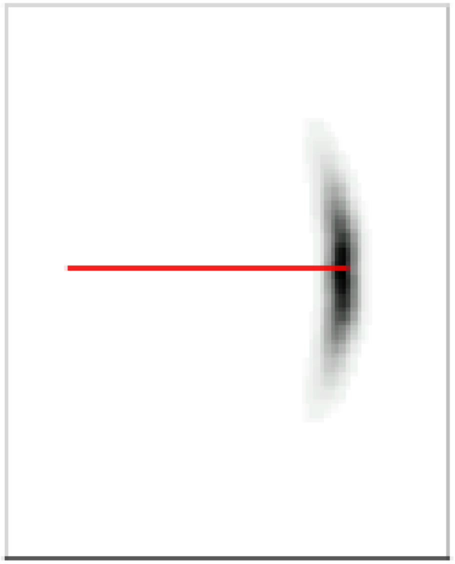
ellipses represent uncertainty in position

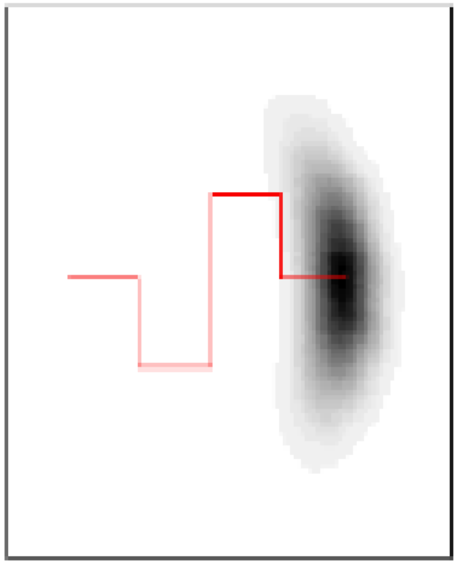
Notes on the previous plots

We only show errors in position, but also theta is affected by errors

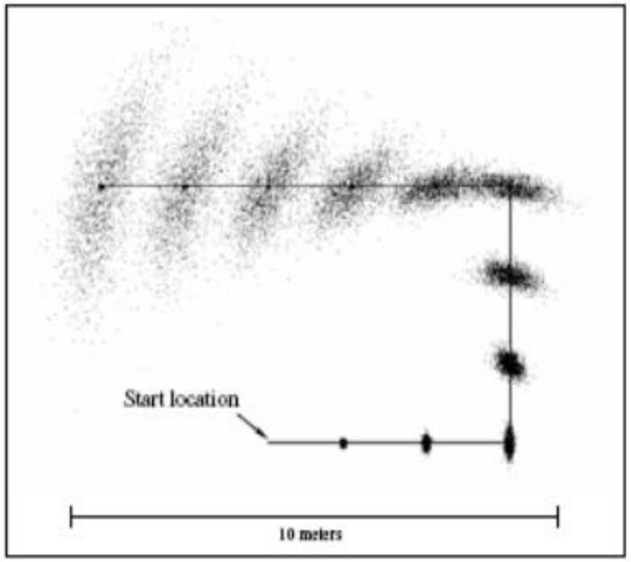
- not represented directly
- shown indirectly by the error ellipses being wider perpendicular to the direction of motion

Note: in practice, errors are not ellipse-shaped





[Fox, Thrun, Burgard, Dellaert, 2000]



(From Fox, et al, AAAI-99)

Sources of odometry errors

- Limited resolution during integration (time increments, measurement resolution)
- Misalignment of the wheels (deterministic)
- Unequal wheel diameter (deterministic)
- Variation in the contact point of the wheel (non deterministic)
- Unequal floor contact (slippage, non planar …) (non deterministic)

Error types

- Deterministic (systematic) errors can be eliminated by proper calibration of the system.
- Non deterministic (non systematic) errors are random and unpredictable. They have to be described by error models and will always lead to uncertain position estimates.

How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where in the map the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

We focus on these (real autonomy without requiring infrastructure)

How to localize?

- **External sensors** (e.g. motion tracking system, a camera looking at the robot...)
- **External beacons** or artificial landmarks (GPS, lighthouses)
- **Onboard sensors** to understand where $\boxed{\text{in the map}}$ the robot is
- **Odometry** to estimate how the pose changes in time (using known control inputs or proprioceptive sensors)

We focus on these (real autonomy without requiring infrastructure)

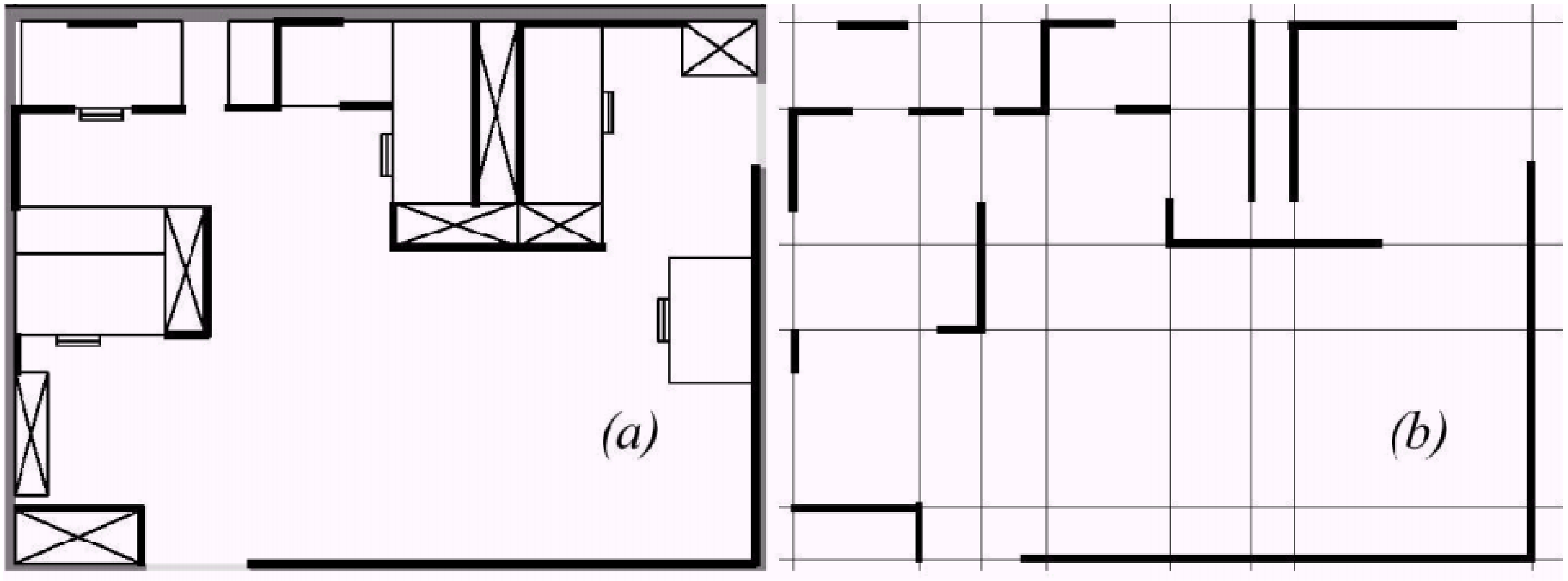
Map Representations

How to represent the map?

Map representation

- Map precision vs. application
  The precision of the map should match the precision with which the robot needs to achieve its goals.
- Features precision vs. map precision
  The precision of the map and the type of features represented should match the precision and data types returned by the robot's sensors.
- Precision vs. computational complexity
  The complexity of the map representation has a direct impact on the computational complexity of reasoning about mapping, localization, and navigation
- Two primary choices:
  - Continuous Representation
  - Decomposition (Discretization)

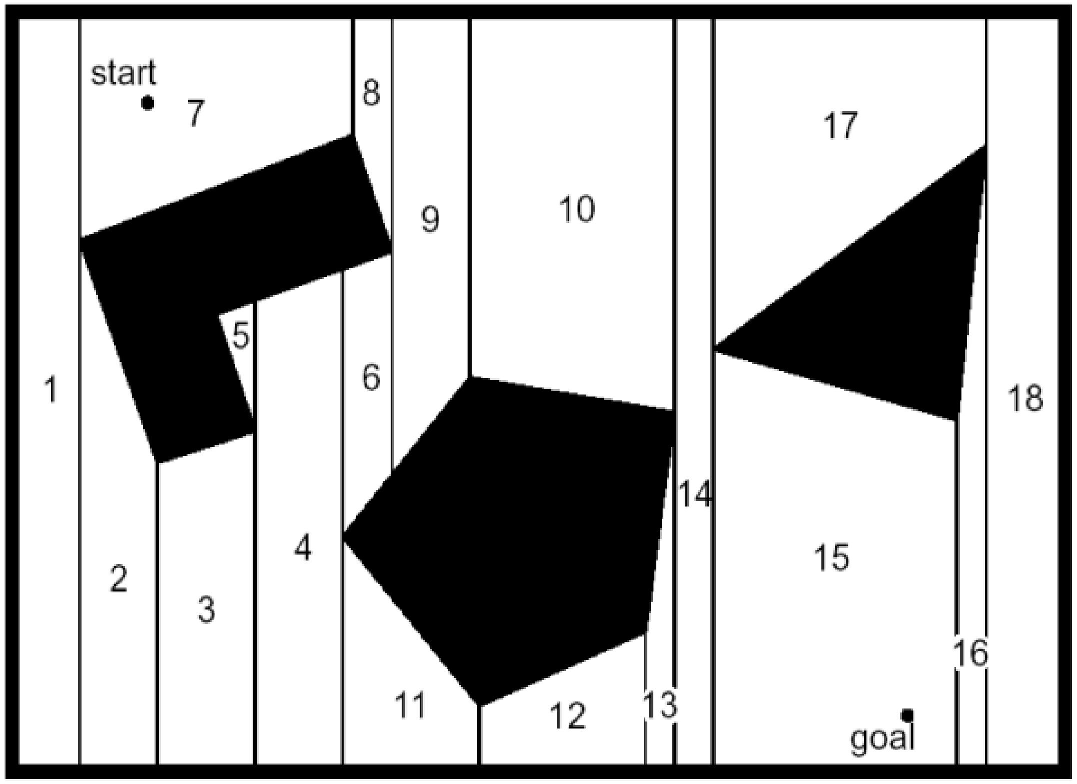
Continuous line-based



a) Architecture map
b) Set of lines

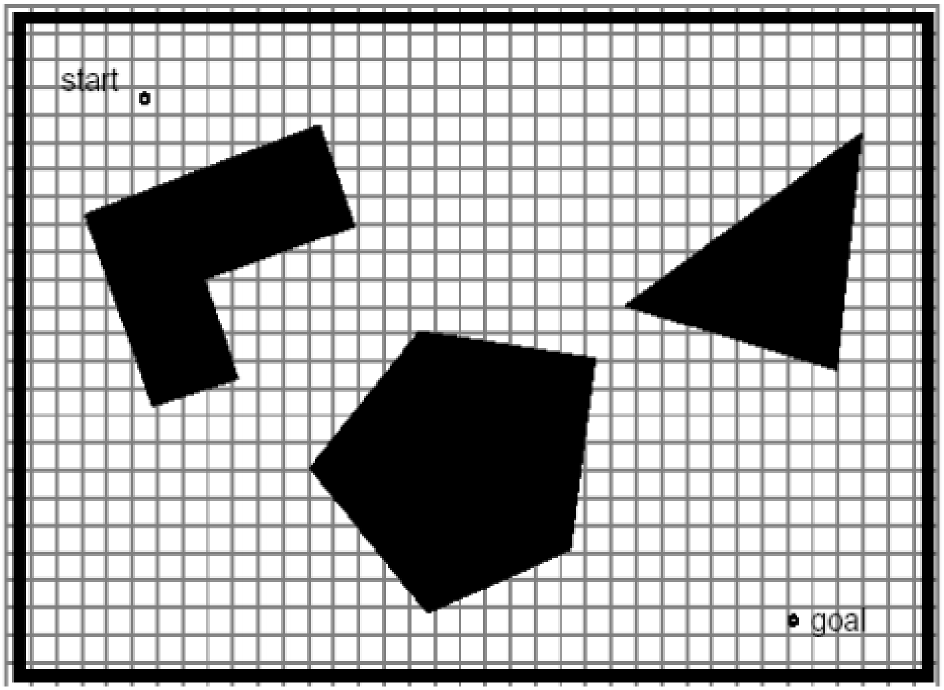
Exact cell decomposition with Polygons

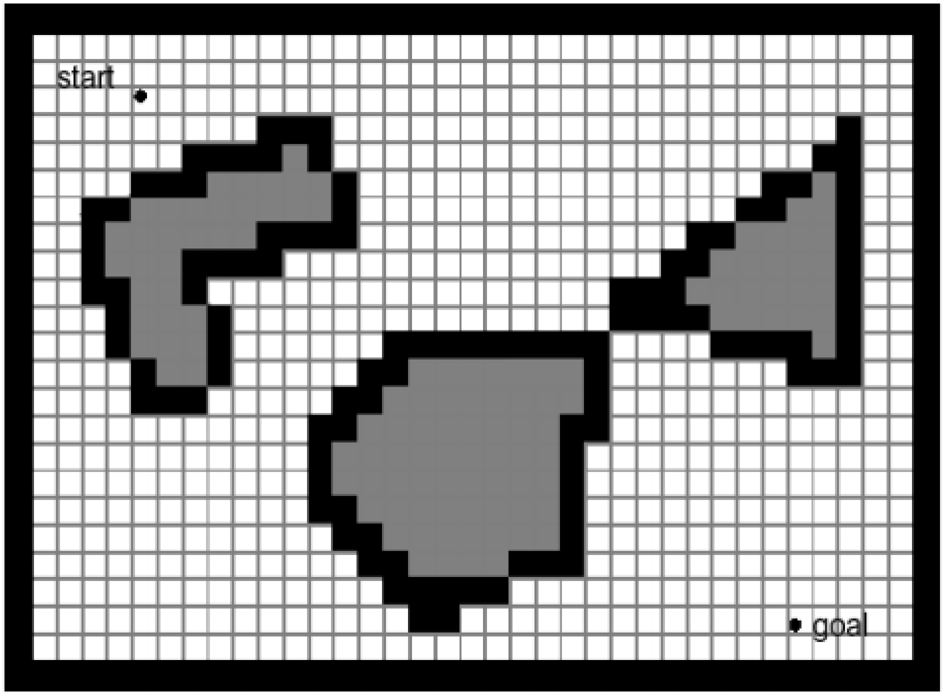
Represent world with set of polygonal cells + adjacency



Approximate cell decomposition

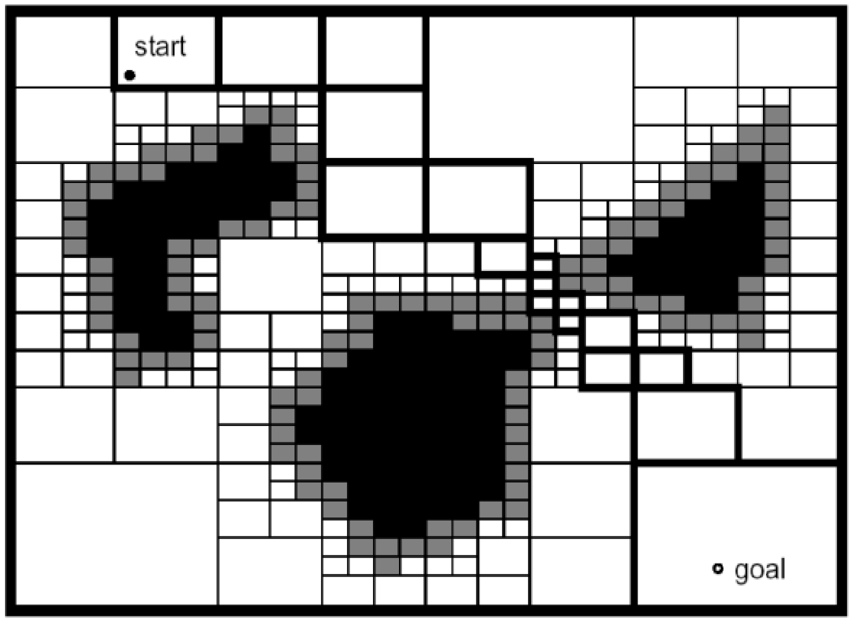
- Regular cell tiling (squares, hexes)
- Any cell that overlaps an obstacle (even in part) is marked as obstacle
- Note: some narrow passages may disappear, depending on:
    - grid size
    - grid offset





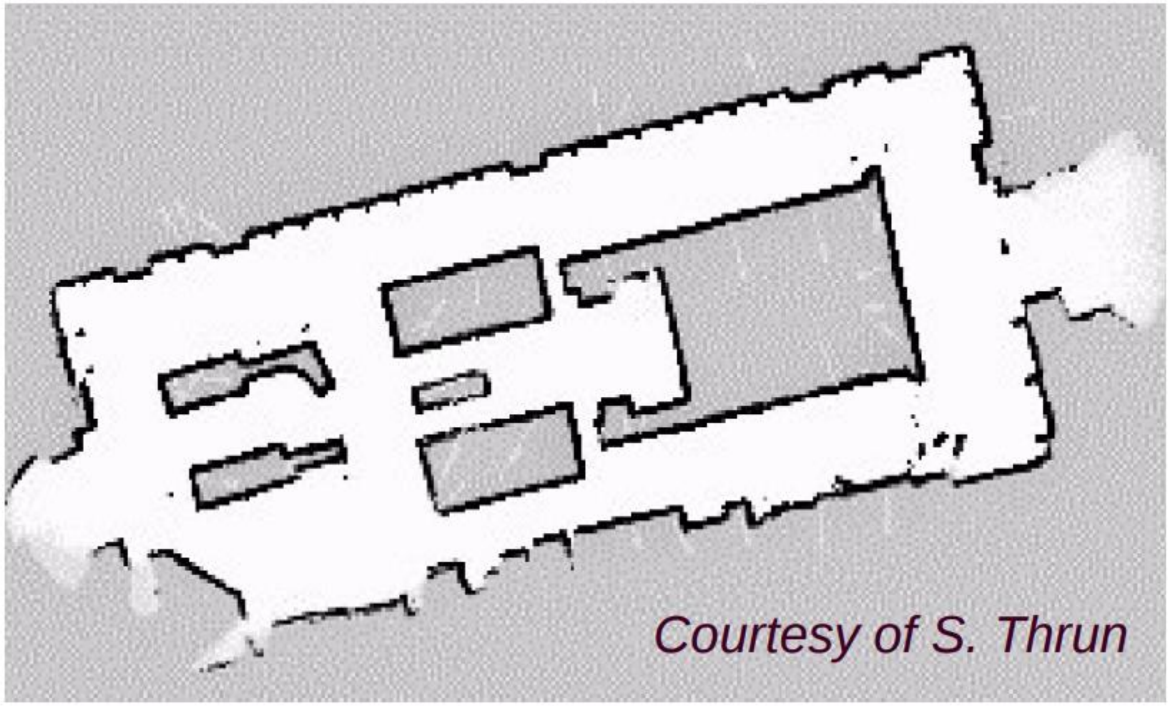
Adaptive cell decomposition

- Increases resolution close to obstacle borders (where more resolution is needed). E.g. quadtree
- Limits memory usage (even by a few orders of magnitude)
- Can handle huge areas while keeping very high resolution for obstacles
- Very effective for sparse maps

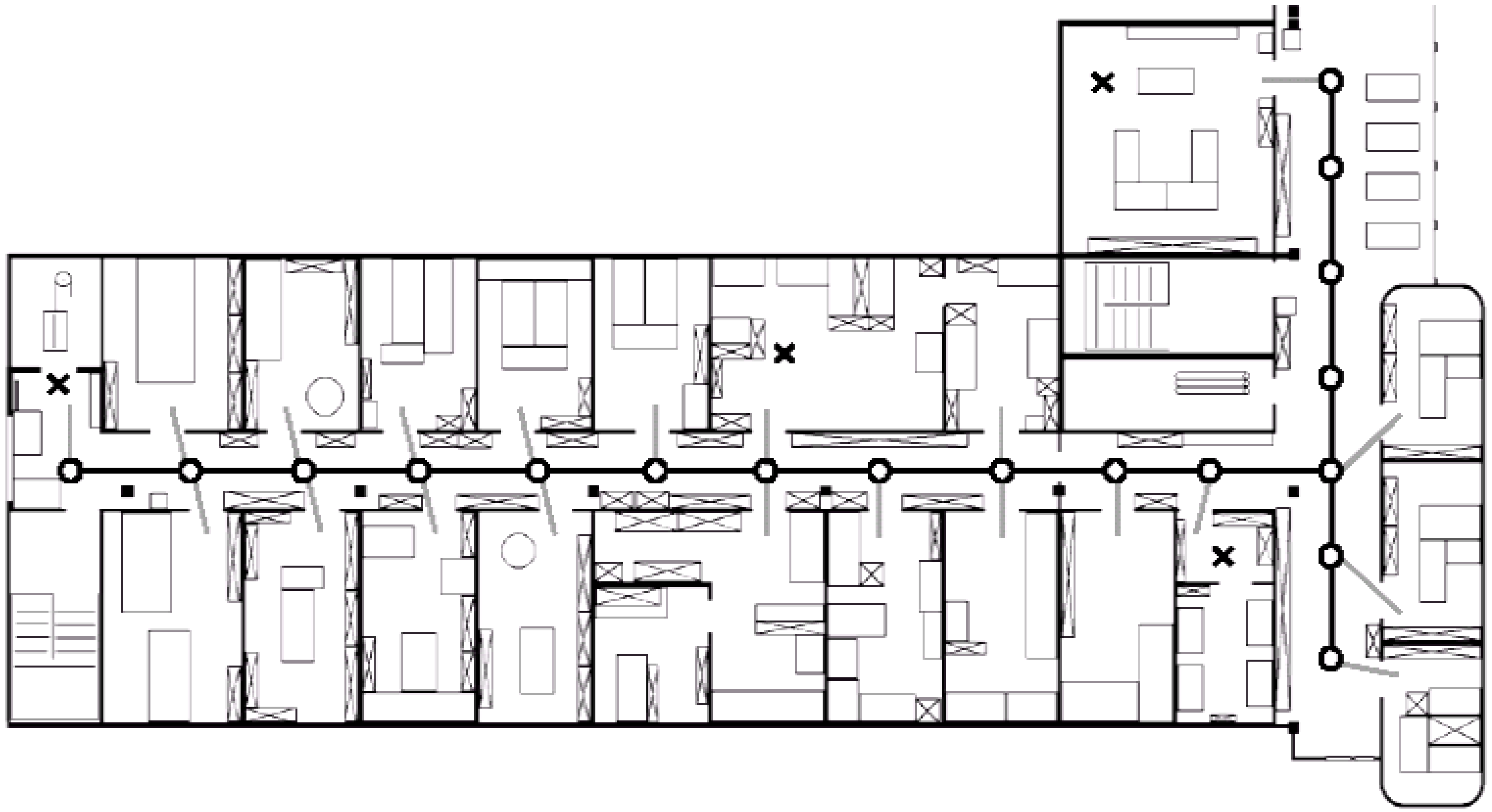


Building a grid map

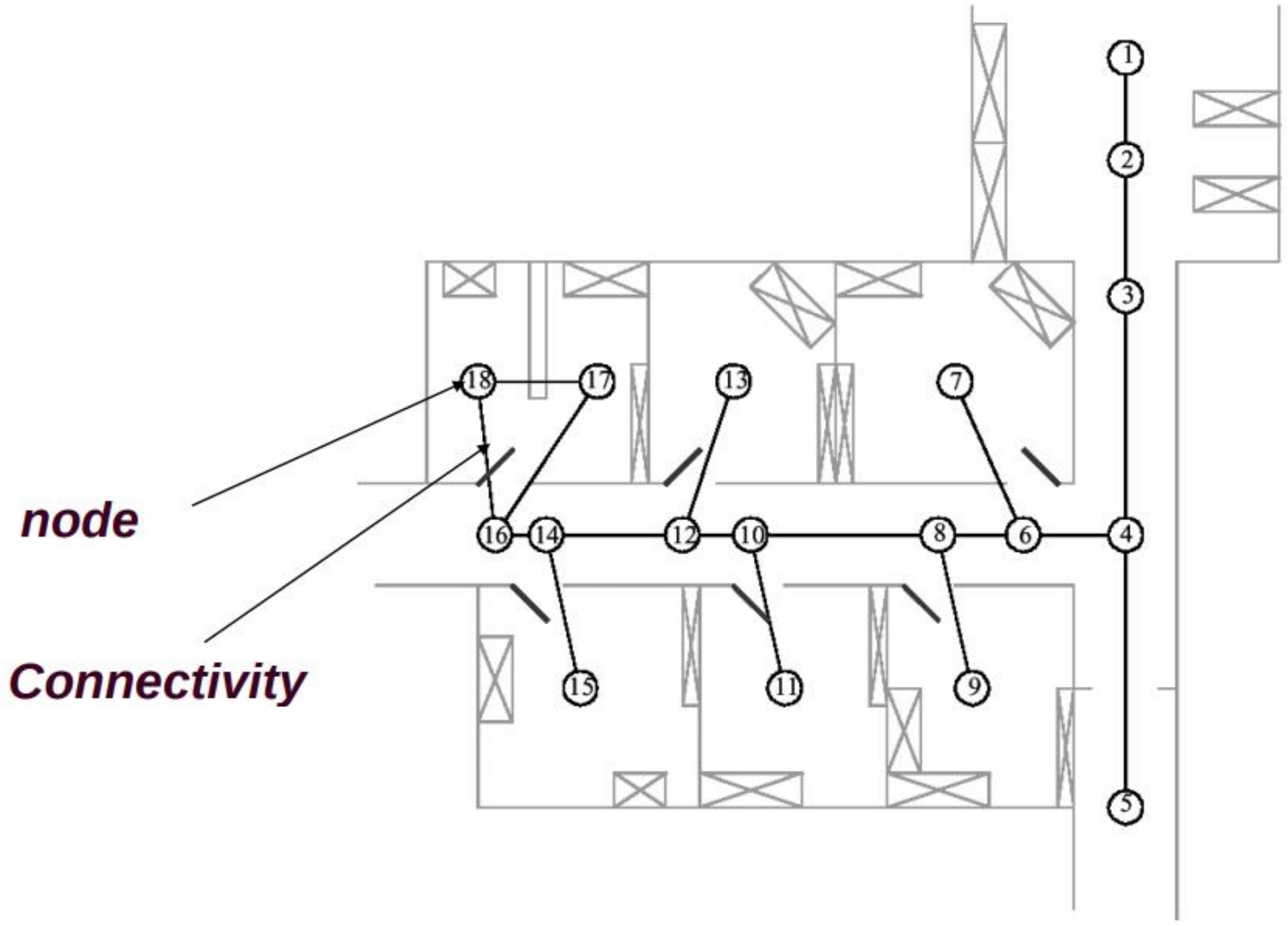
- Cells initialized gray
- A cell hit by a ranging measurement is made darker (becomes black when many rays hit that cell)
- A cell crossed by a ranging measurement is made brighter (becomes white when many rays cross that cell without being blocked)
- May change over time (e.g. dynamic obstacles)



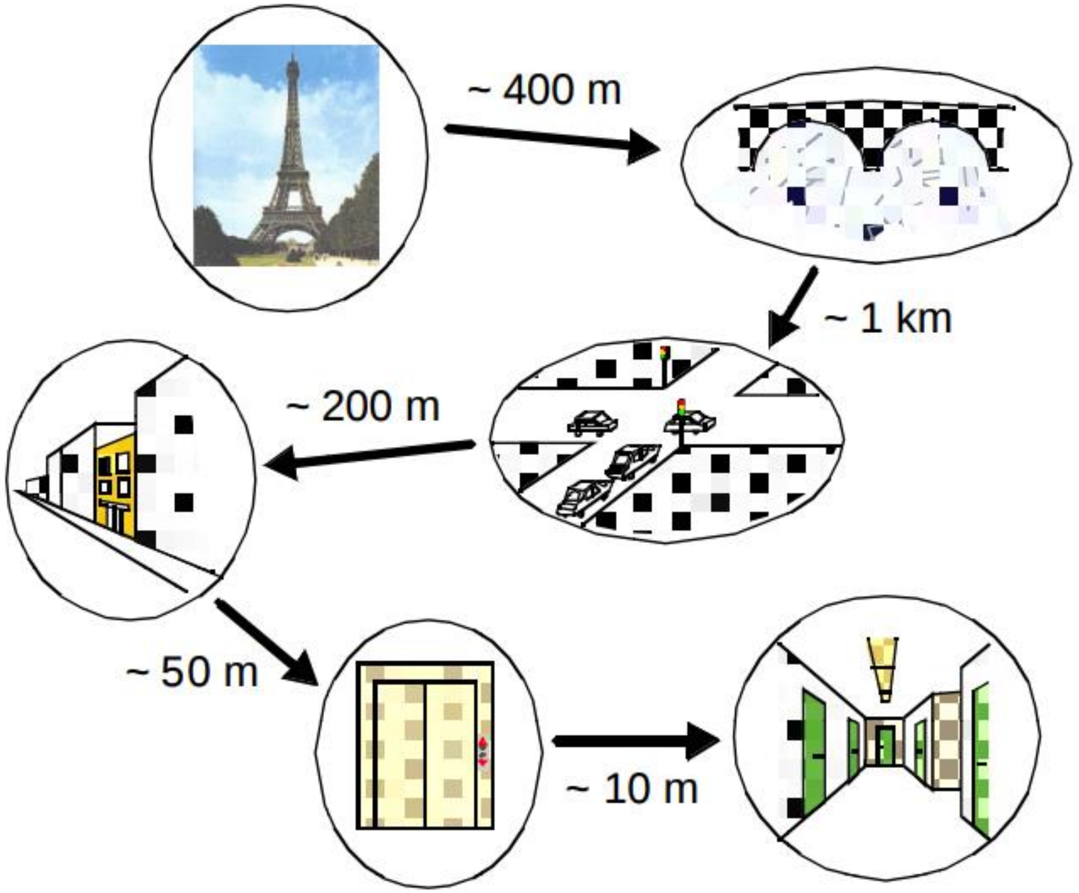
Topological decomposition



Topological decomposition



Topological decomposition

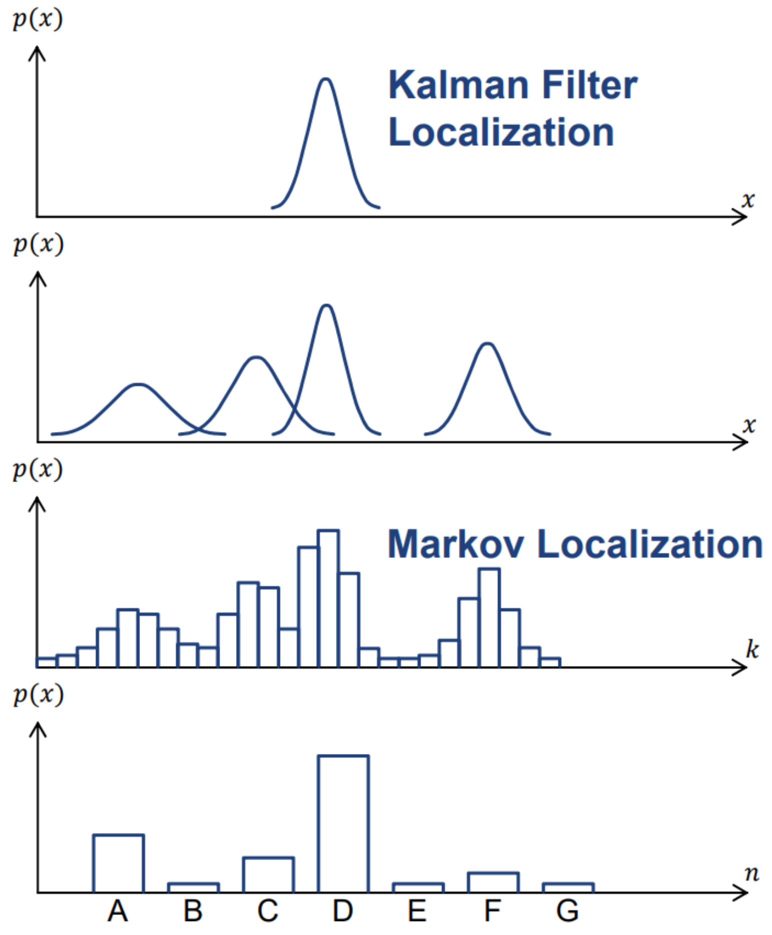


Belief representation

How to represent the belief about the robot pose?

Representing the robot belief about its pose

- **Continuous** map with **single hypothesis**
- **Continuous** map with **multiple hypotheses**
- **Discretized** metric map (grid with probability distribution, **multiple hypoteses**)
- **Discretized** topological map (nodes with probability distribution , **multiple hypoteses**)



Continuous vs Discrete Belief Representaiton

Continuous

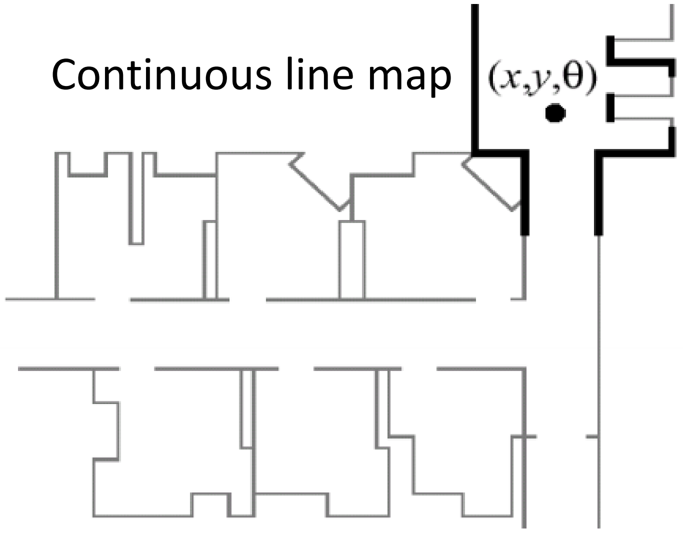
- Precision bound by sensor data
- Typically single hypothesis (may get lost if hypothesis is very wrong)
- Compact representation and efficient

Discrete

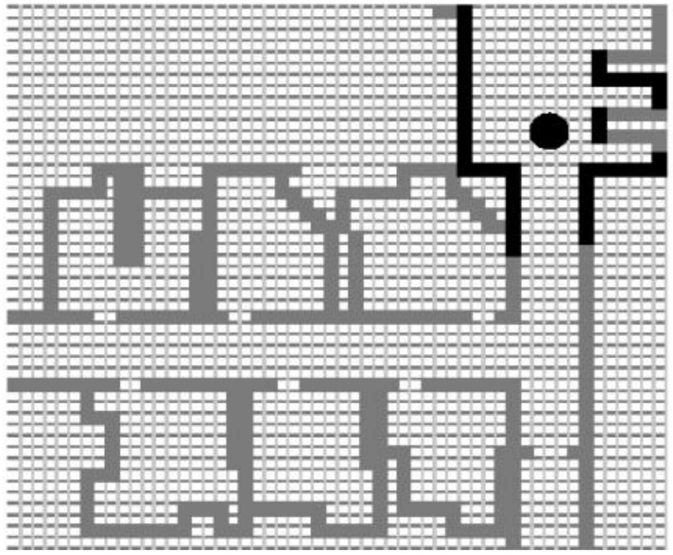
- Precision bound by resolution of discretization
- Typically multiple hypothesis pose estimate (won’t get lost)
- Memory and processing requirements

Single-hypothesis examples

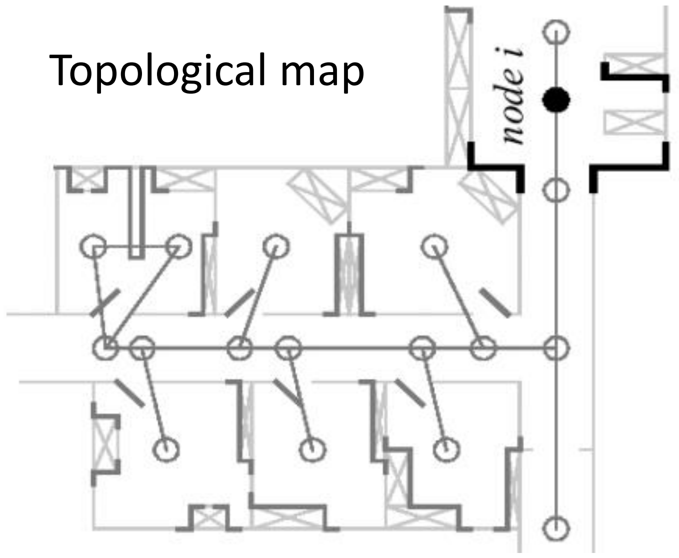
Continuous line map



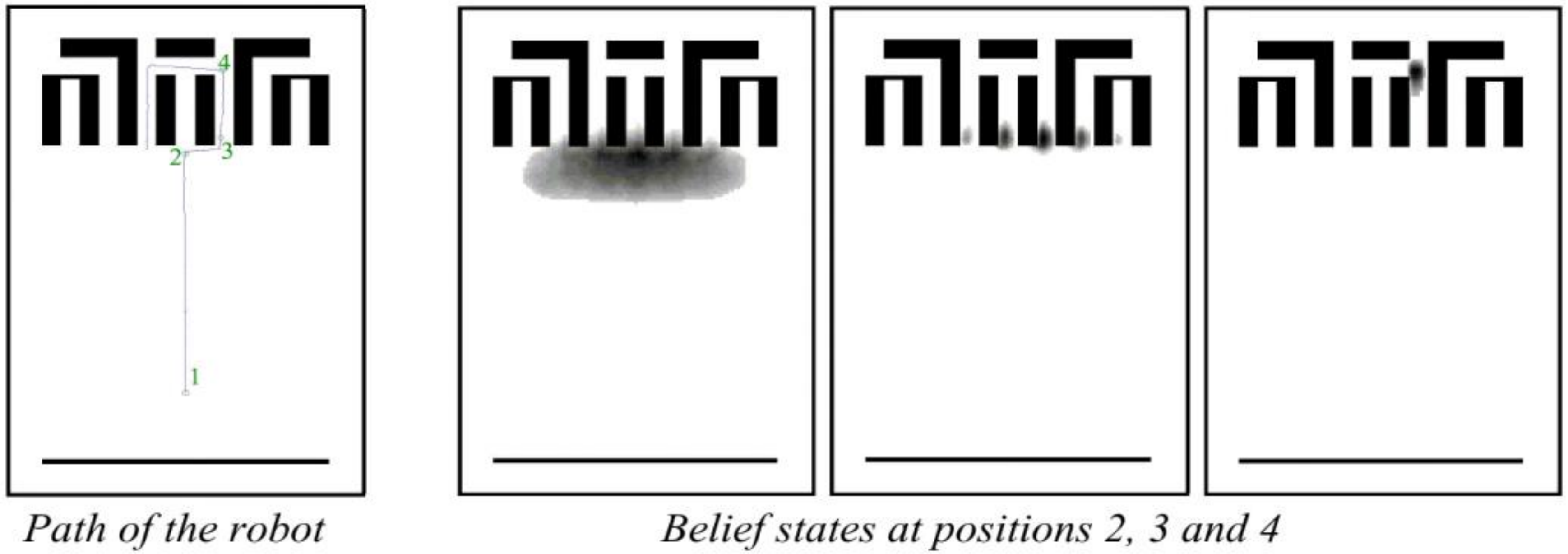
Grid map



Topological map



Multi Hypotesis on a grid map



- Grid size around 20 sq cm
- Clouds represent possible robot locations
- Darker coloring means higher probability

Main idea of map-based localization

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.
- Odometric information leads to an estimate of the robot's position, which can then be fused with the sensor observations to get the best possible update of the robot's actual position.

Ingredients

- Probability theory

  error propagation, sensor fusion
- Belief representation

  discrete / continuous (map/position)
- Motion model

  odometry model
- Sensing

  measurement model

Two approaches to localization

Markov

- Maintains multiple estimates of robot position
- Localization can start from any unknown position
- Can recover from ambiguous situations
- However, to update the probability of all positions within the state space requires a discrete representation of the space (grid);
- if a fine grid is used (or many estimates are maintained), the computational and memory requirements can be large.

Kalman

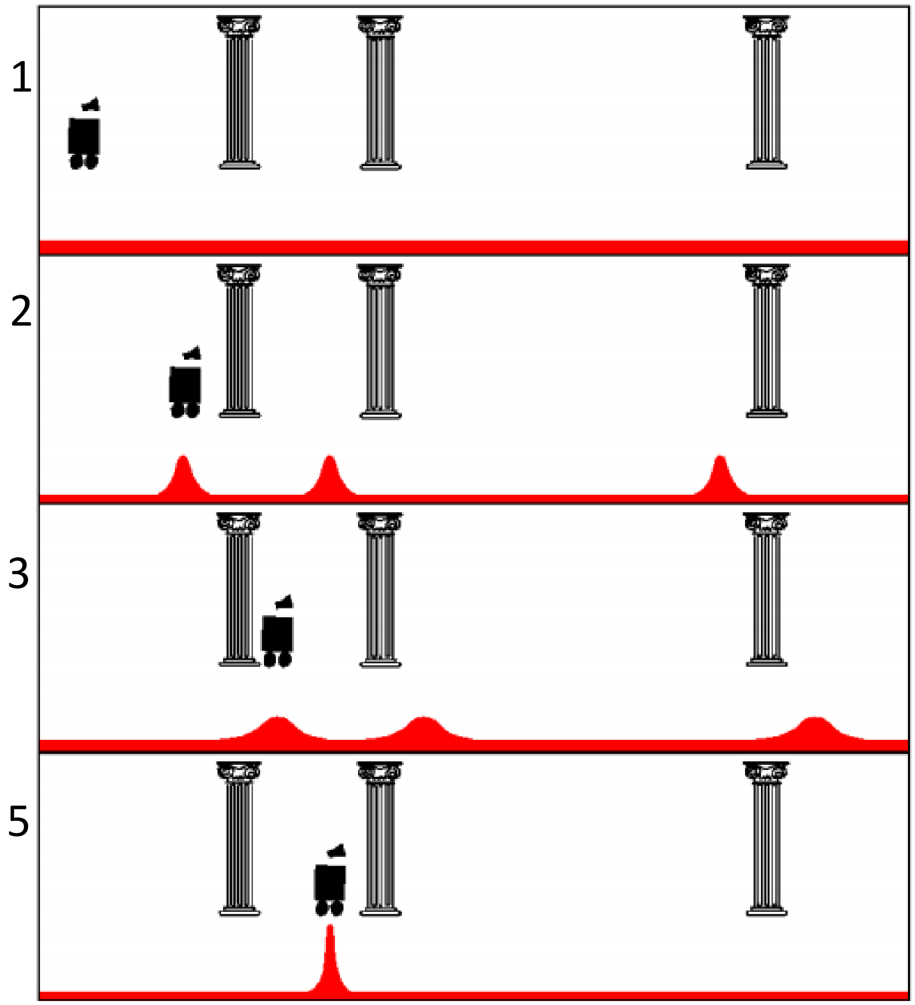
- Single estimate of robot position
- Requires known starting position of robot
- Tracks the robot and can be very precise and efficient
- However, if the uncertainty of the robot becomes too large (e.g. due collision with an object) the Kalman filter will fail and the robot becomes “lost”

Markov localization

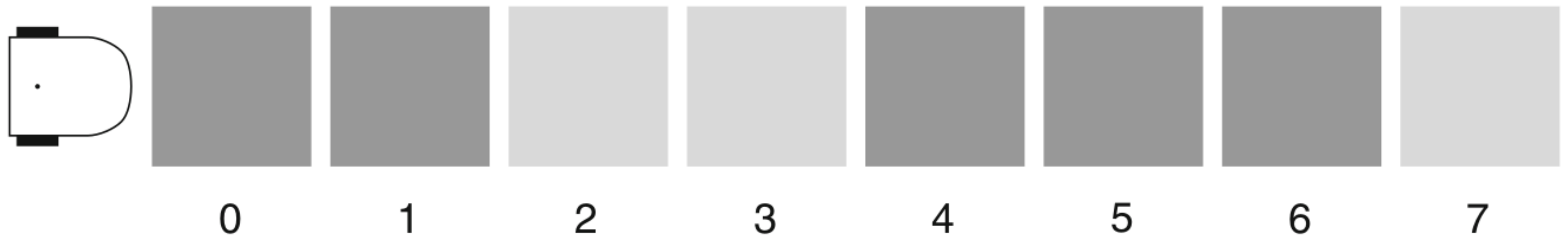
- Markov localization uses an explicit, discrete representation for the probability of all robot states (poses)
- This is usually done by representing the environment by a grid or a topological graph with a finite number of possible states.
- During each update, the probability for each state (element) of the entire space is updated
- This probability distribution represents the likelihood that the robot is in a particular pose

General idea

1. Start
   No knowledge at start, thus we have a uniform probability distribution.
2. SEE: Robot perceives first pillar
   Seeing only one pillar, the probability being at pillar 1, 2 or 3 is equal.
3. ACT: Robot moves
   Action model enables estimation of the new probability distribution based on the previous one and the motion.
4. ACT: Robot move again (not pictured)
5. SEE: Robot perceives second pillar
   Based on all prior knowledge, the probability of being at pillar 2 becomes dominant



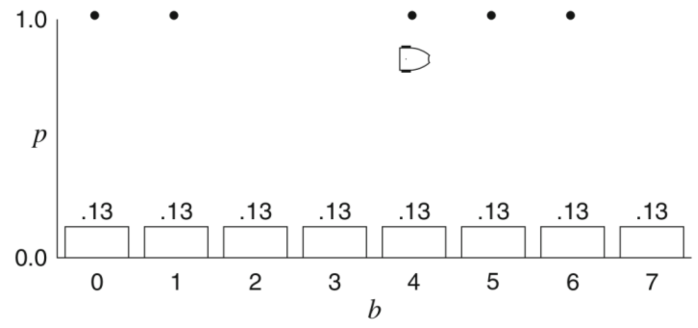
Example on a 1D state space: without uncertainty



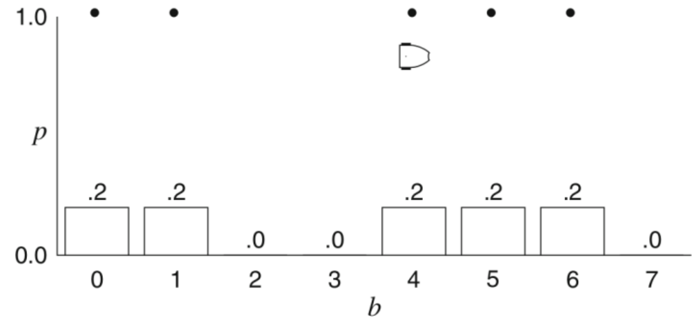
Thymio does not know its initial position (global localization problem), but:

- it can move right perfectly (action model, ACT)
- it can sense whether it’s on a dark or light cell, perfectly (perception model, SEE)

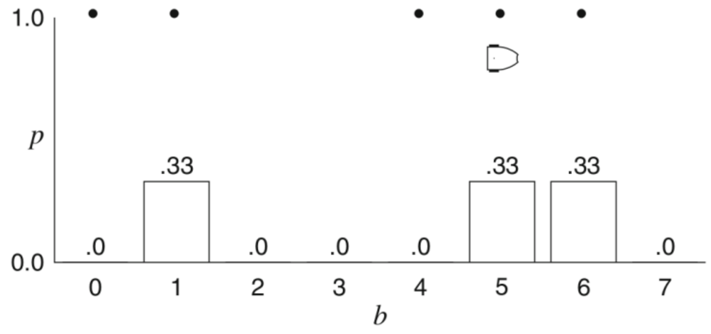
1. Initial belief



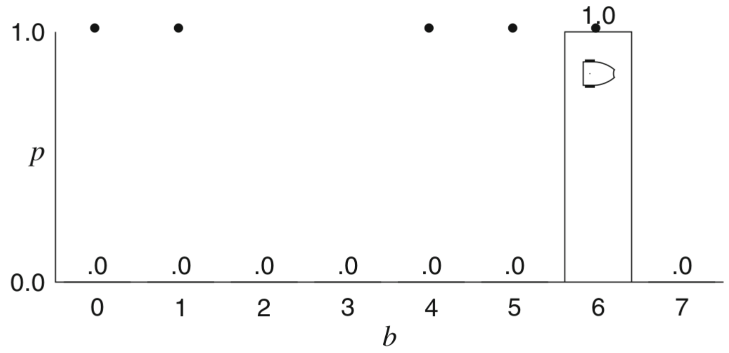
2. After sensing a dark floor (see)



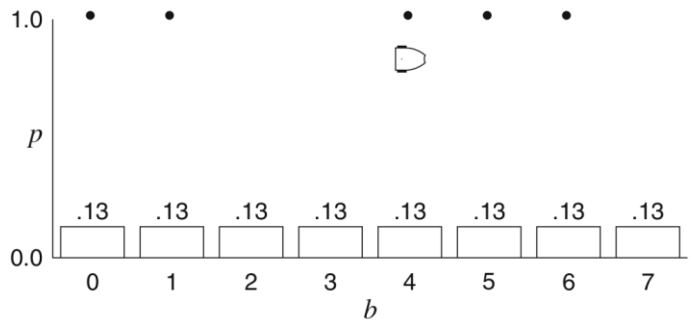
3. After moving 1 cell right (act)
and sensing a dark floor (see)



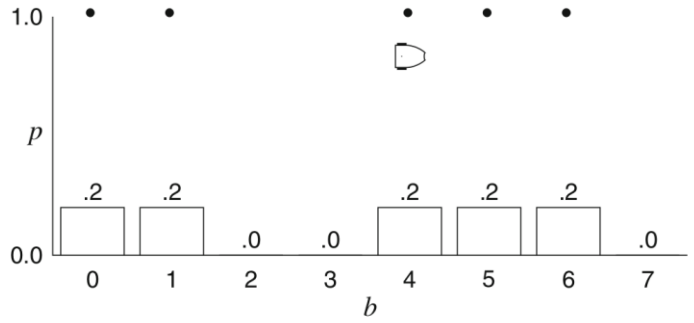
4. After moving 1 cell right (act)
and sensing a **dark** floor (see)



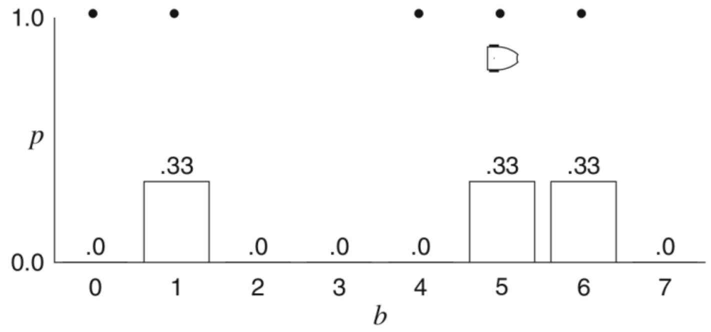
1. Initial belief



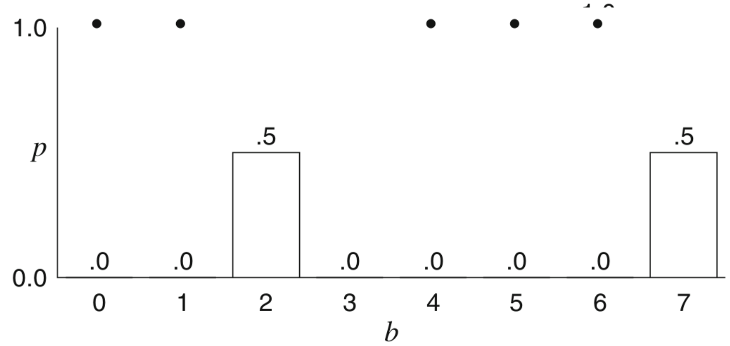
2. After sensing a dark floor (see)



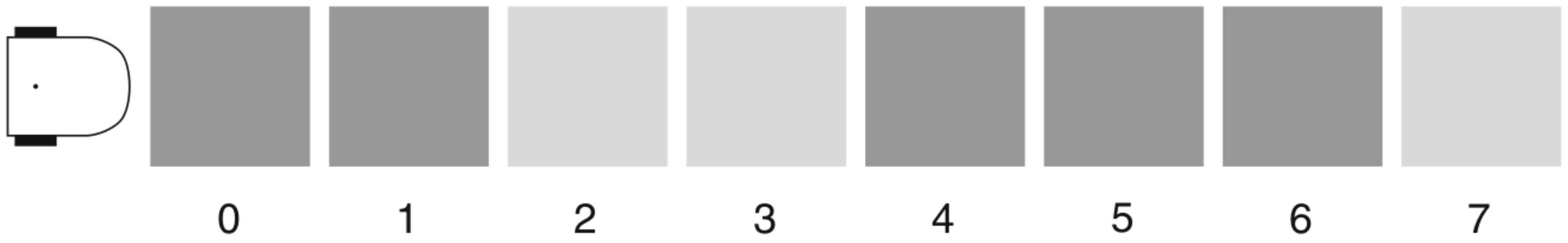
3. After moving 1 cell right (act)
and sensing a dark floor (see)



4. After moving 1 cell right (act)
and sensing a **light** floor (see)

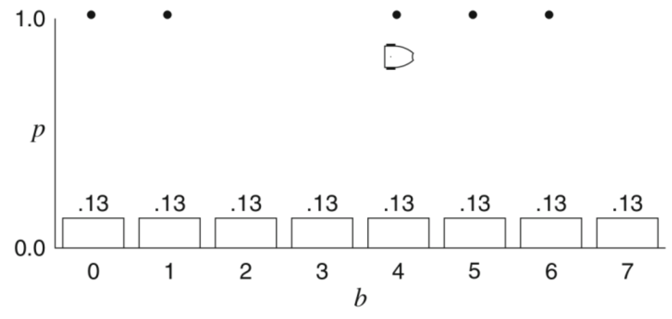


Example on a 1D state space: with sensing uncertainty

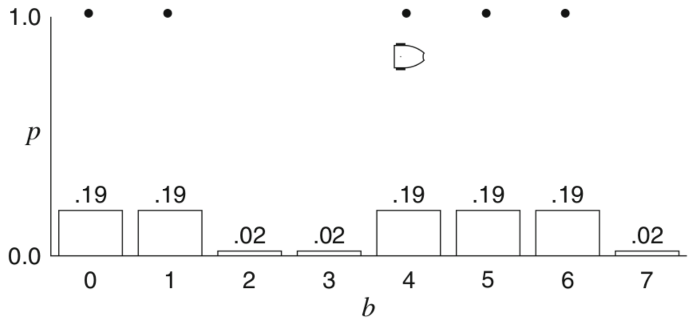


- The thymio does not know its initial position (global localization problem)
- It can move right perfectly (action model).
  The world is circular, cell 0 is to the right of cell 7
- It can sense whether it's on a dark or light cell (perception model), however:
  - If the robot is on a dark cell:
    - it will correctly detect a dark cell with probability 0.9
    - it will mistakenly detect a light cell with probability 0.1
  - If the robot is on a light cell:
    - it will correctly detect a light cell with probability 0.9
    - it will mistakenly detect a dark cell with probability 0.1

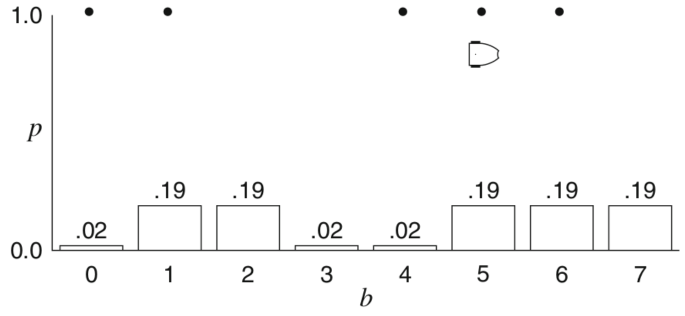
Act



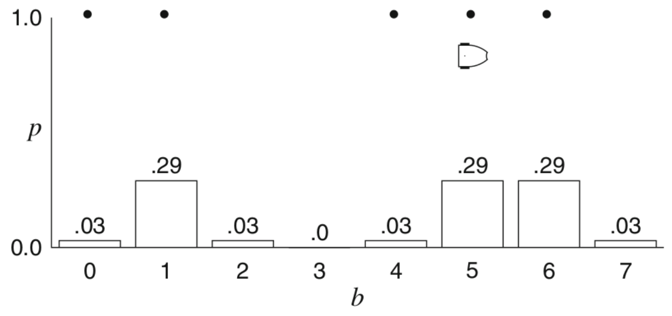
See



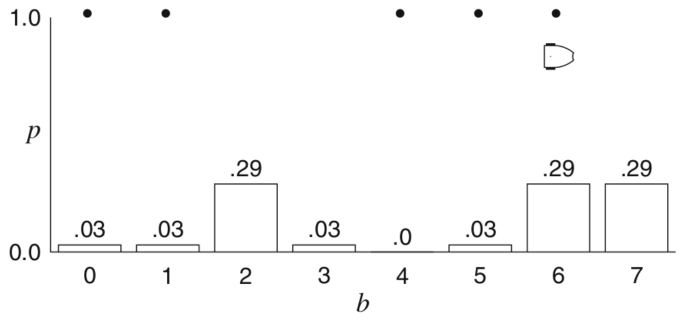
Act



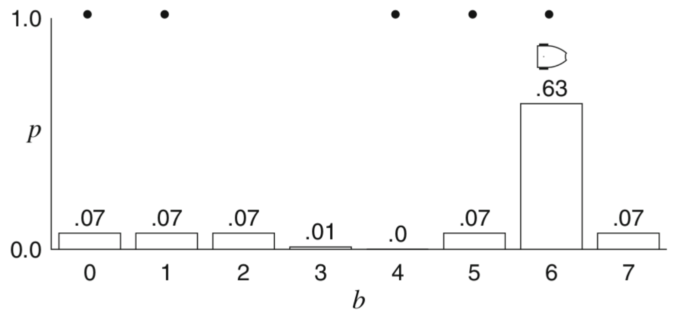
See



Act



See



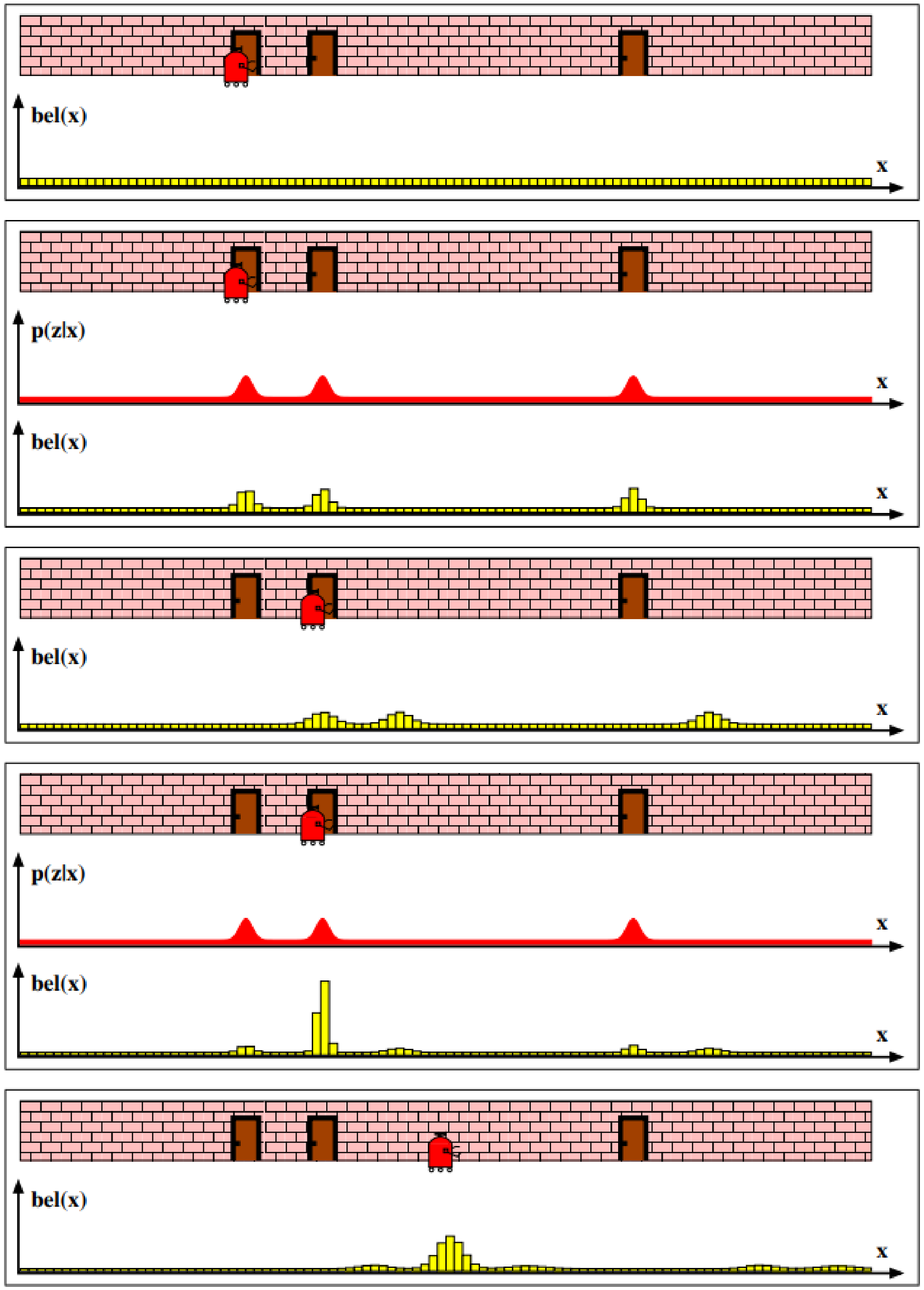
A similar 1D scenario on a finer grid

Notation:

- Position or pose:
  x or *l*

- Sensor input:
  z or *i*

- Action:
  o



Eq. 5.21 on Autonomous Mobile Robots

Sensor model (SEE)
(updating the belief after reading a sensor)

We just got measurement *i* from our sensor. **For each pose *l* of the state space**, we update our belief of being there as follows

input (sensor reading)

probability to get measurement *i* if we are at *l*

My previous belief of being at *l* (before looking)

pose (aka state or location)

$$p(l|i) = \frac{p(i|l)p(l)}{p(i)}$$

The probability of being at *l* given that I got measurement *i*

normalization factor so that sum over all l equals 1.

Eq. 5.22 on Autonomous Mobile Robots

Action model (ACT)
(updating the belief after moving)

We just got measurement *o* from our wheel encoder (or IMU, or we sent a command *o* to an encoder-less wheel, like on a thymio). **For each pose** *l* **of the state space**, we update our belief of being there at time *t* as follows

probability of reaching *l*, given that we were at *l′* in the previous timestep and we observed o

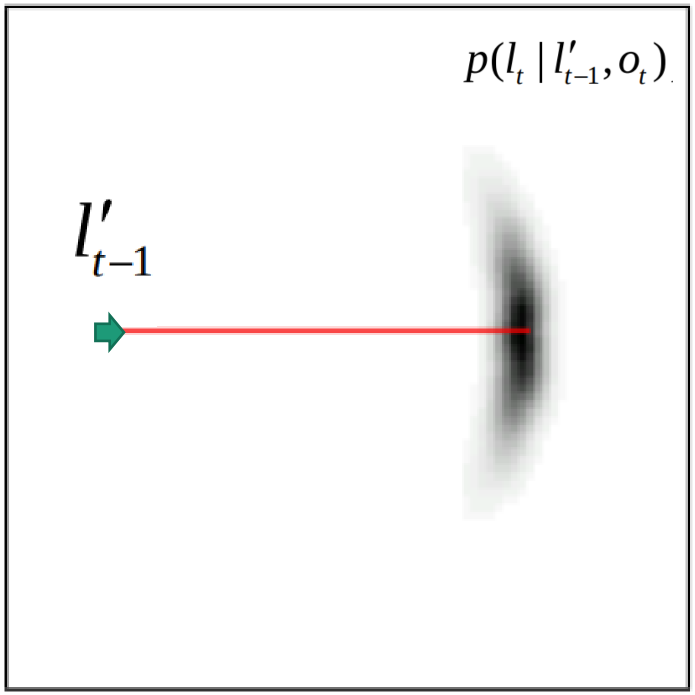
$$p(l_t \mid o_t) = \int p(l_t \mid l'_{t-1}, o_t)\, p(l'_{t-1})\, dl'_{t-1}$$

Sum over all possible positions *l′* we may have had at *t*-1, weighted by the belief we were in fact there

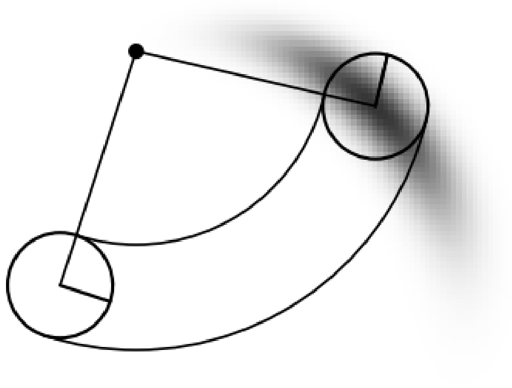
Example $p(l_t \mid l'_{t-1}, o_t)$

Probability map for for a single l'_{t-1} pose

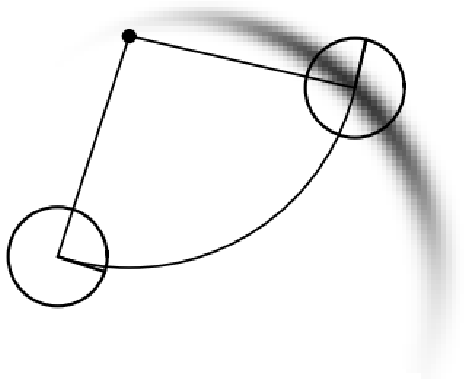
o is a straight motion
(measured or commanded)



o is a curve
(measured or commanded)



diff drive
robot



bicycle with
uncertain steering
and good wheel
encoder

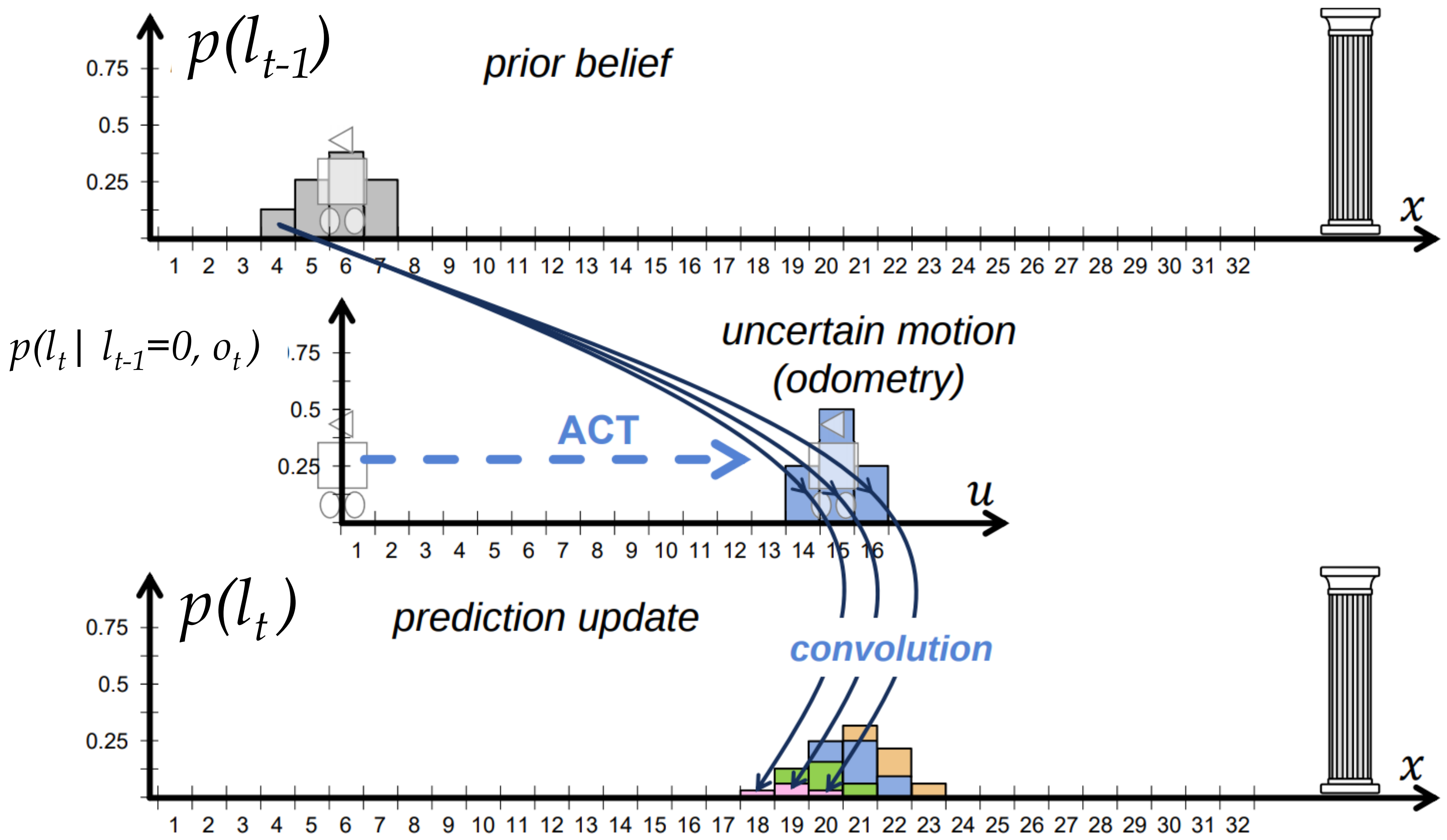
The Markov Assumption

ACT: $$p(l_t \mid o_t) = \int p(l_t \mid l'_{t-1}, o_t)\, p(l'_{t-1})\, dl'_{t-1}$$

SEE: $$p(l|i) = \frac{p(i|l)p(l)}{p(i)}$$

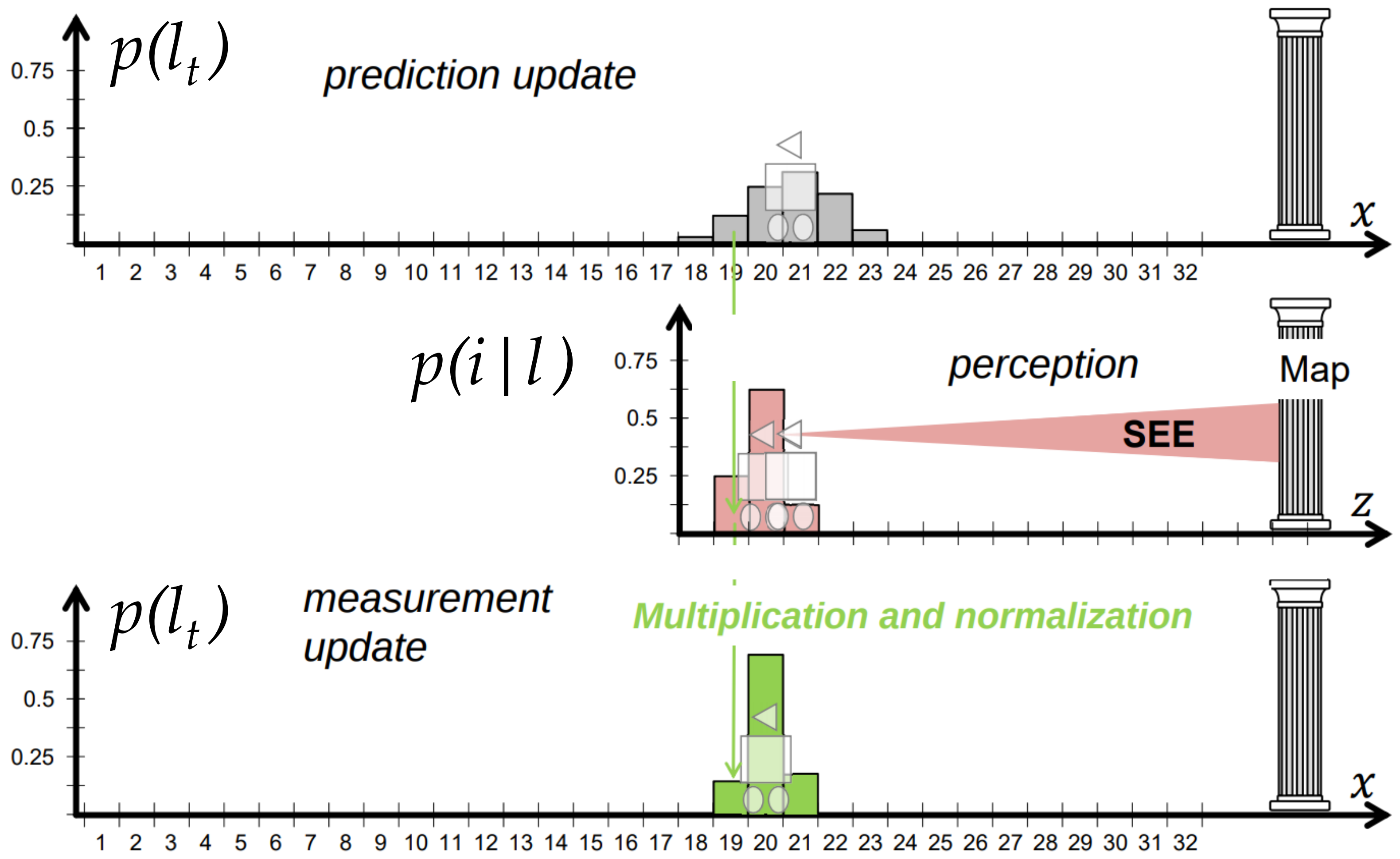
The current belief only depends on the previous belief about the state, and the most recent action *o* and perception *i*

ACT step

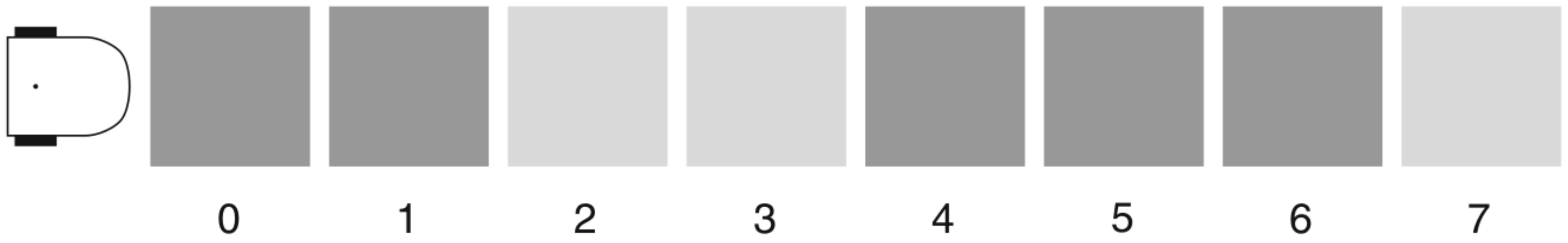


$p(l_{t-1})$
prior belief
0.75
0.5
0.25
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
x
$p(l_t \mid l_{t-1}=0, o_t)$
uncertain motion (odometry)
ACT
u
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$p(l_t)$
prediction update
convolution

SEE step


$p(l_t)$
prediction update
0.75
0.5
0.25
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
$p(i \mid l)$
perception
Map
SEE
z
$p(l_t)$
measurement update
Multiplication and normalization

Example on a 1D state space: with sensing uncertainty **and** motion uncertainty

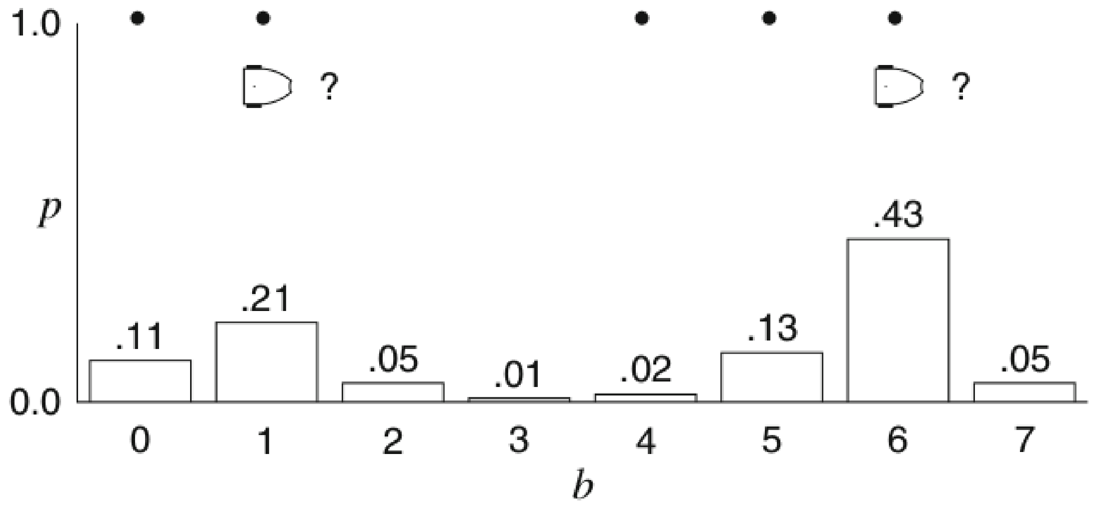


Thymio does not know its initial position (global localization problem), but:

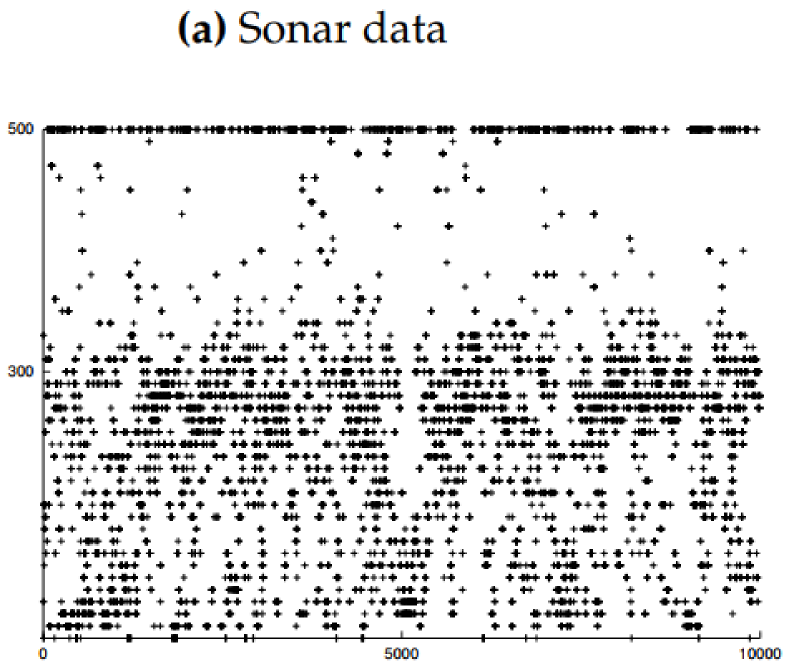
- when it moves right:
  - With probability 0.1 it will actually stay where it is
  - With probability 0.8 it will move one tile to the right
  - With probability 0.1 it will move two tiles
- It can sense whether it’s on a dark or light cell (perception model), however:
  - If the robot is on a dark cell:
    - it will correctly detect a dark cell with probability 0.9
    - it will mistakenly detect a light cell with probability 0.1
  - If the robot is on a light cell:
    - it will correctly detect a light cell with probability 0.9
    - it will mistakenly detect a dark cell with probability 0.1

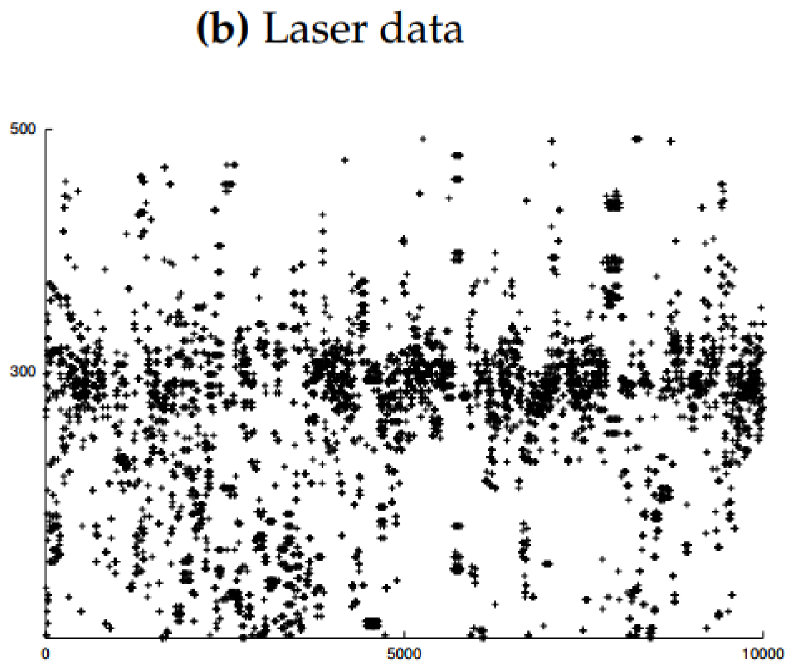
Table 8.2 Localization with uncertainty in sensing and motion, sensor = after multiplying by the sensor uncertainty, norm = after normalization, right = after moving right one position

	Position	0	1	2	3	4	5	6	7
	Dark cell?	●	●			●	●	●	
	Initial	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13
SEE (i=dark)	Sensor	0.11	0.11	0.01	0.01	0.11	0.11	0.11	0.01
	Norm	0.19	0.19	0.02	0.02	0.19	0.19	0.19	0.02
ACT (o=right)	Right	0.05	0.19	0.17	0.04	0.04	0.17	0.19	0.17
SEE (i=dark)	Sensor	0.05	0.17	0.02	0.00	0.03	0.15	0.17	0.02
	Norm	0.08	0.27	0.03	0.01	0.06	0.25	0.28	0.03
ACT (o=right)	Right	0.06	0.12	0.23	0.05	0.01	0.07	0.23	0.25
SEE (i=dark)	Sensor	0.05	0.10	0.02	0.01	0.01	0.06	0.21	0.02
	Norm	0.11	0.21	0.05	0.01	0.02	0.13	0.43	0.05



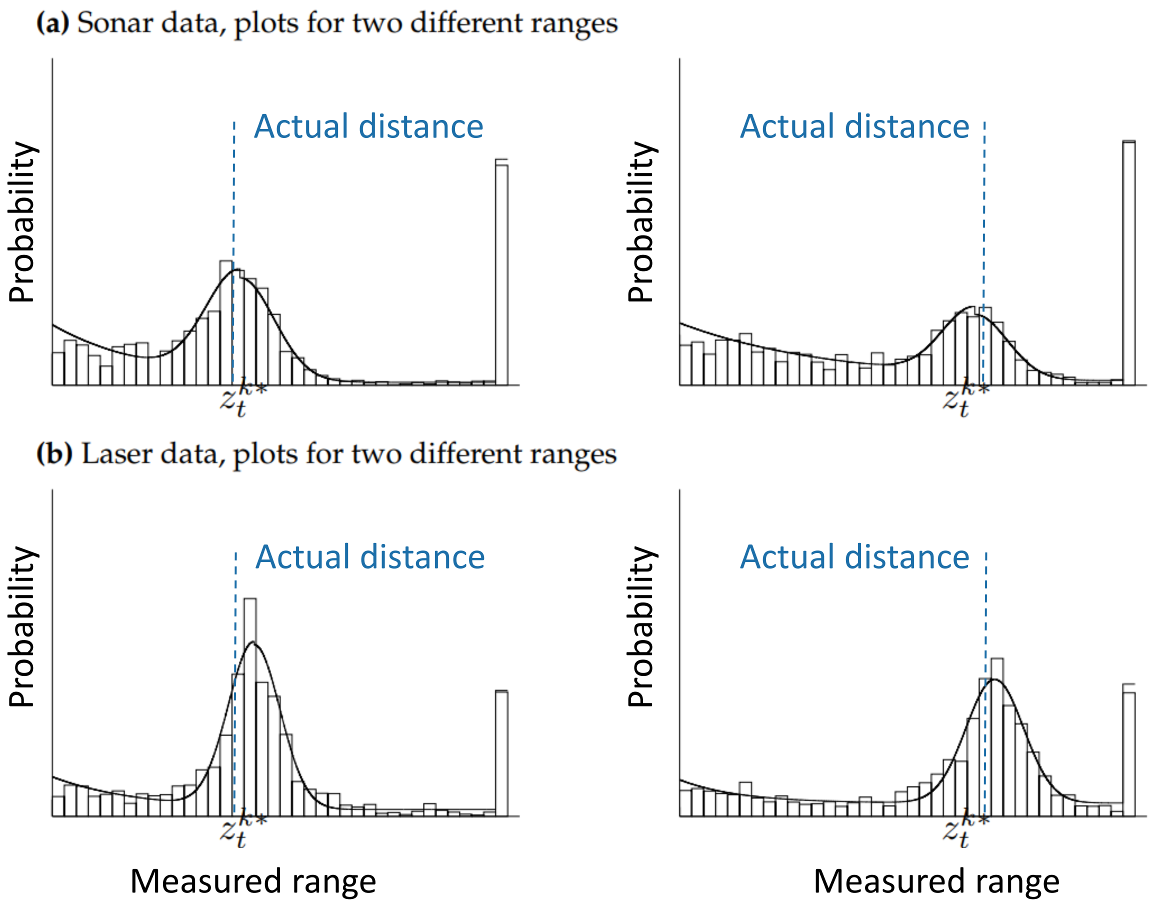
How real range sensors behave





Typical data obtained with (a) a sonar sensor and (b) a laser-range sensor in an office environment for a “true” range of 300 cm and a maximum range of 500 cm

How real range sensors behave



Approximation of the beam model based on (a) sonar data and (b) laser range data. The sensor models depicted on the left were obtained by a maximum likelihood approximation to the data sets depicted in the previous slide

Markov localization for a planar robot

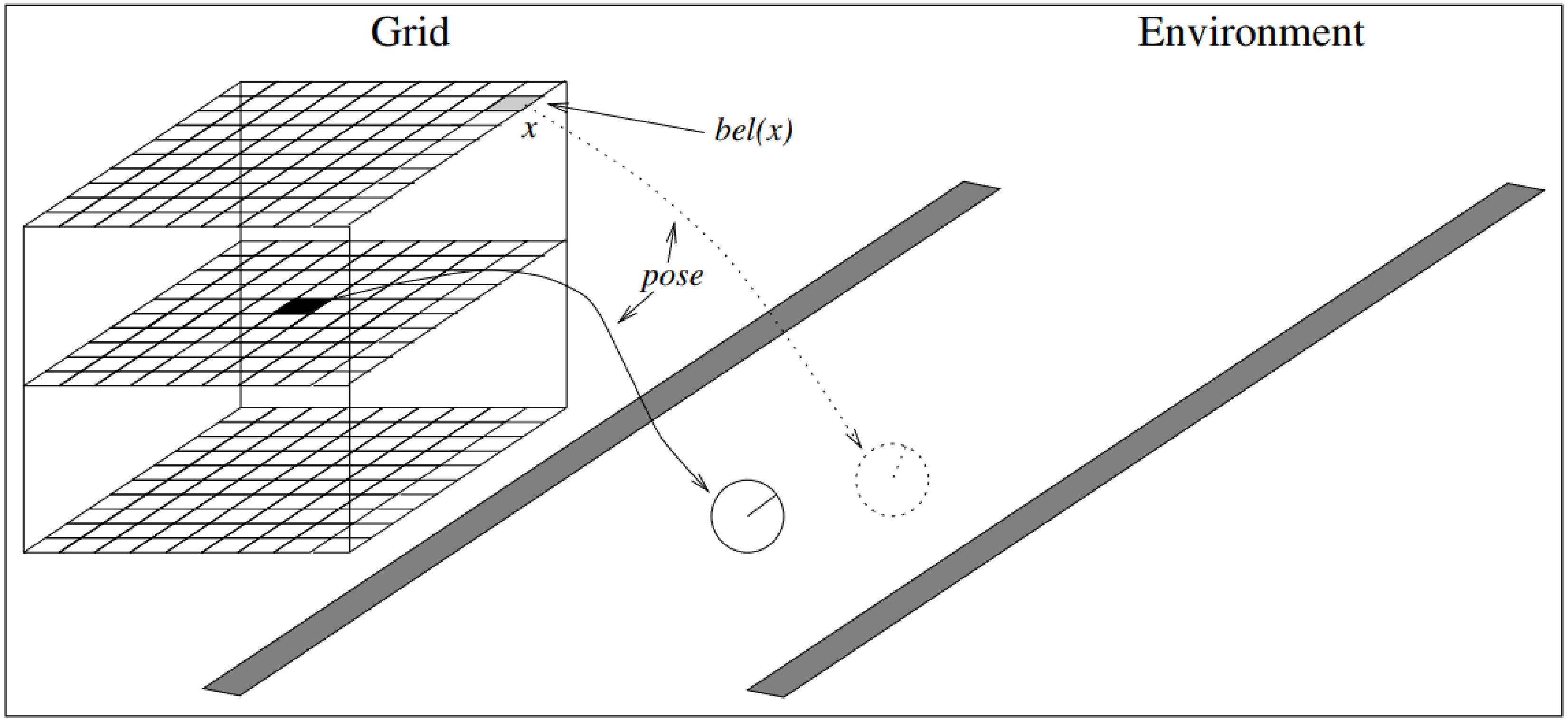
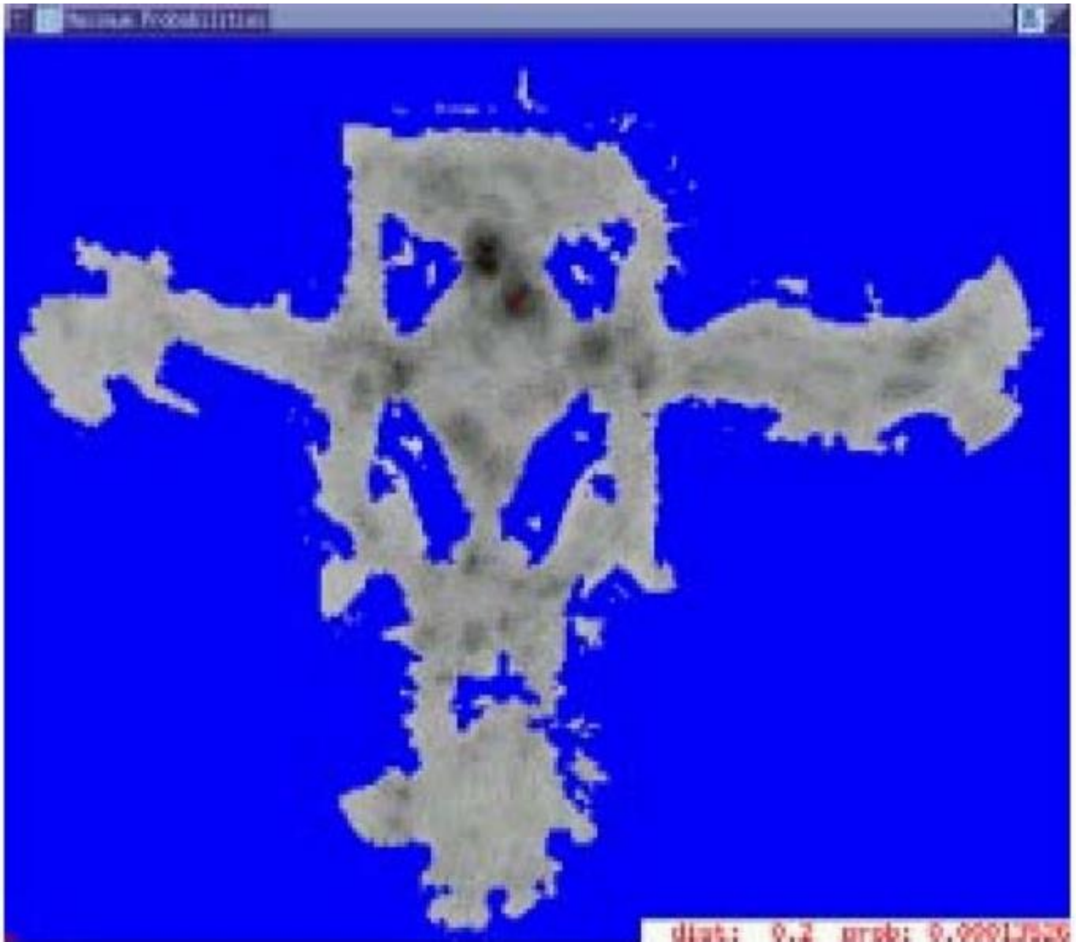
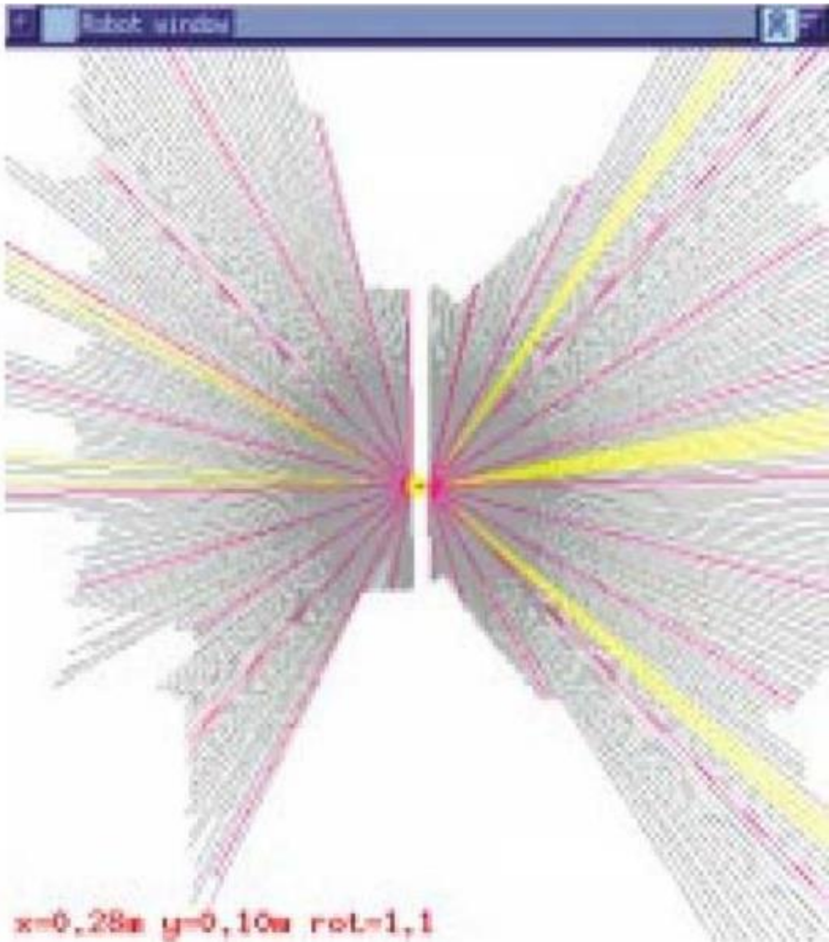


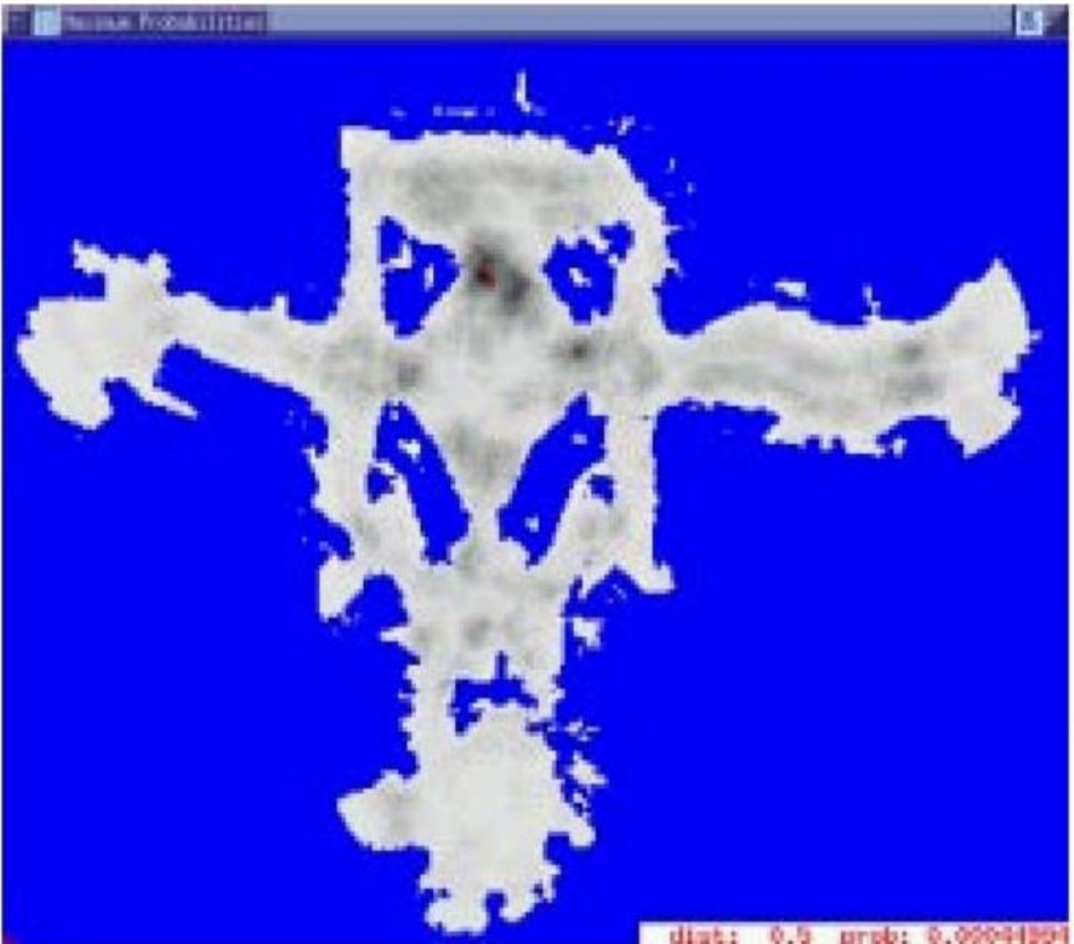
Figure 8.2 Example of a fixed-resolution grid over the robot pose variables x, y, and θ. Each grid cell represents a robot pose in the environment. Different orientations of the robot correspond to different planes in the grid (shown are only three orientations).

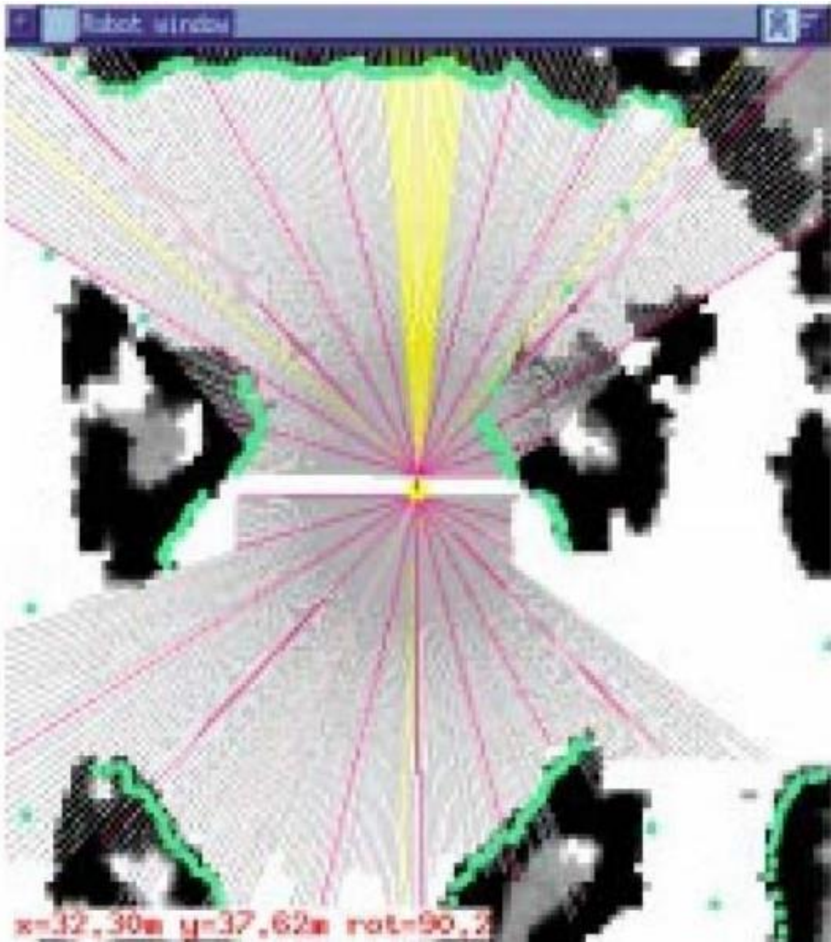
Markov Localization



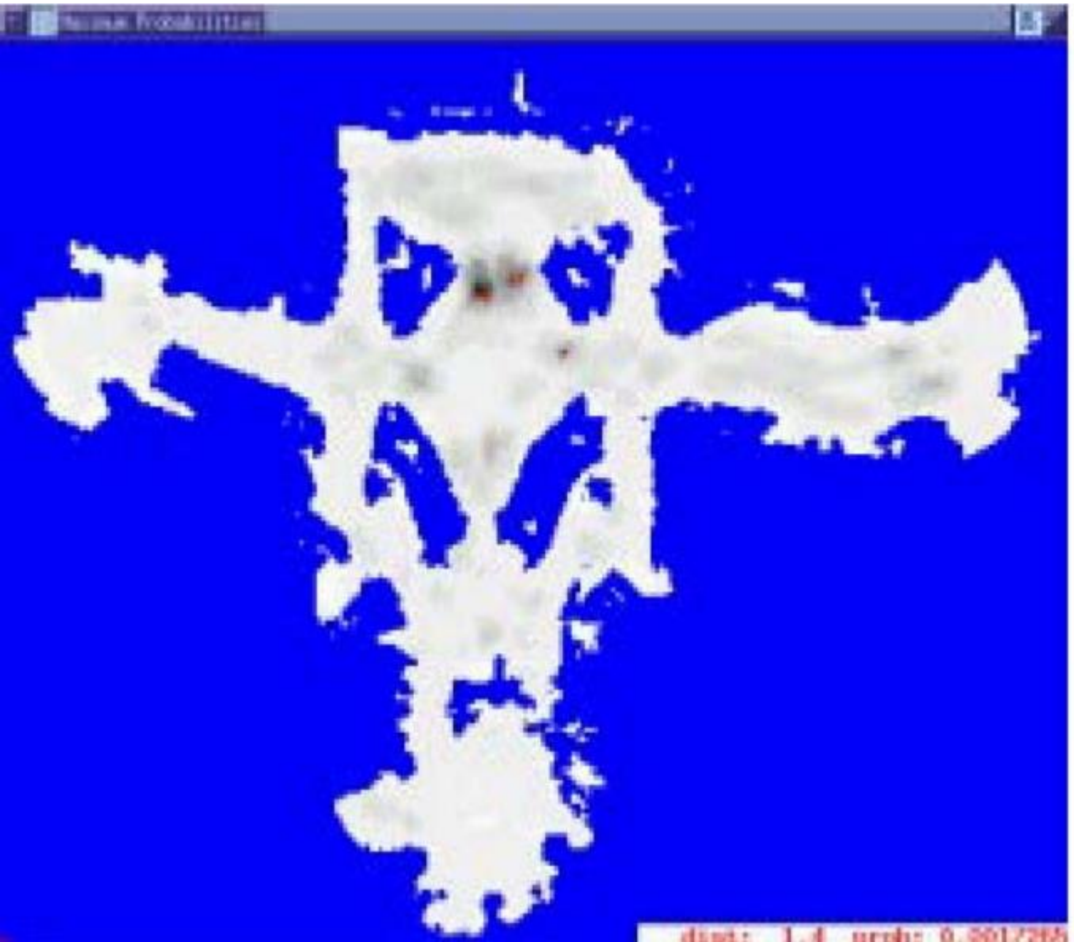


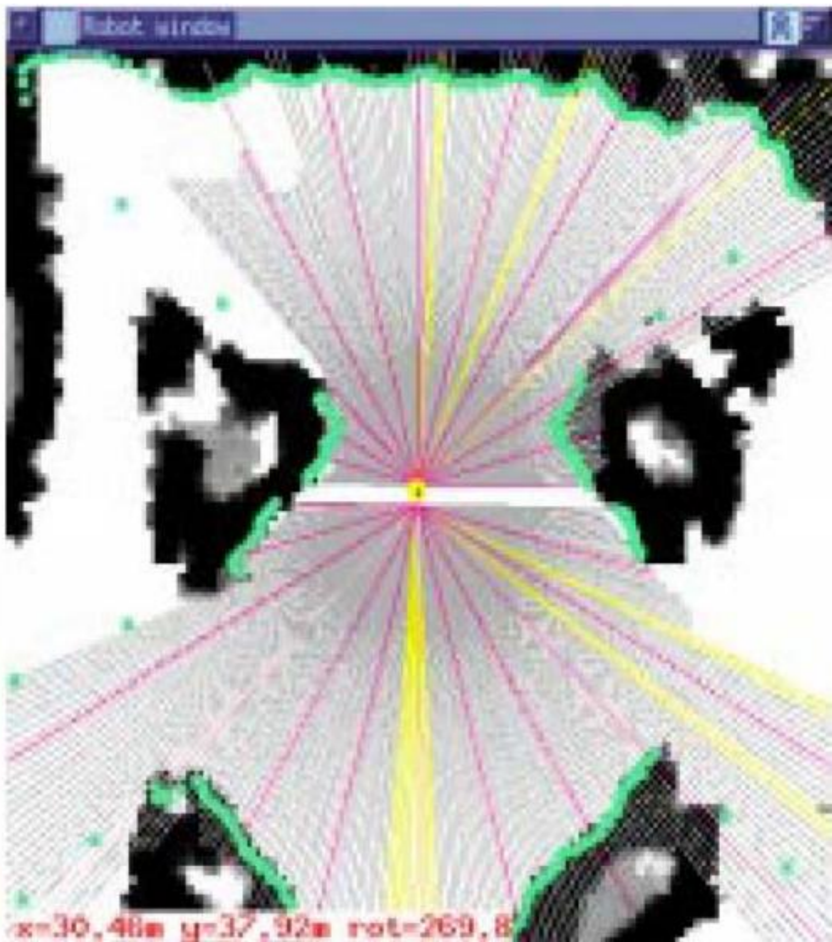
Markov Localization



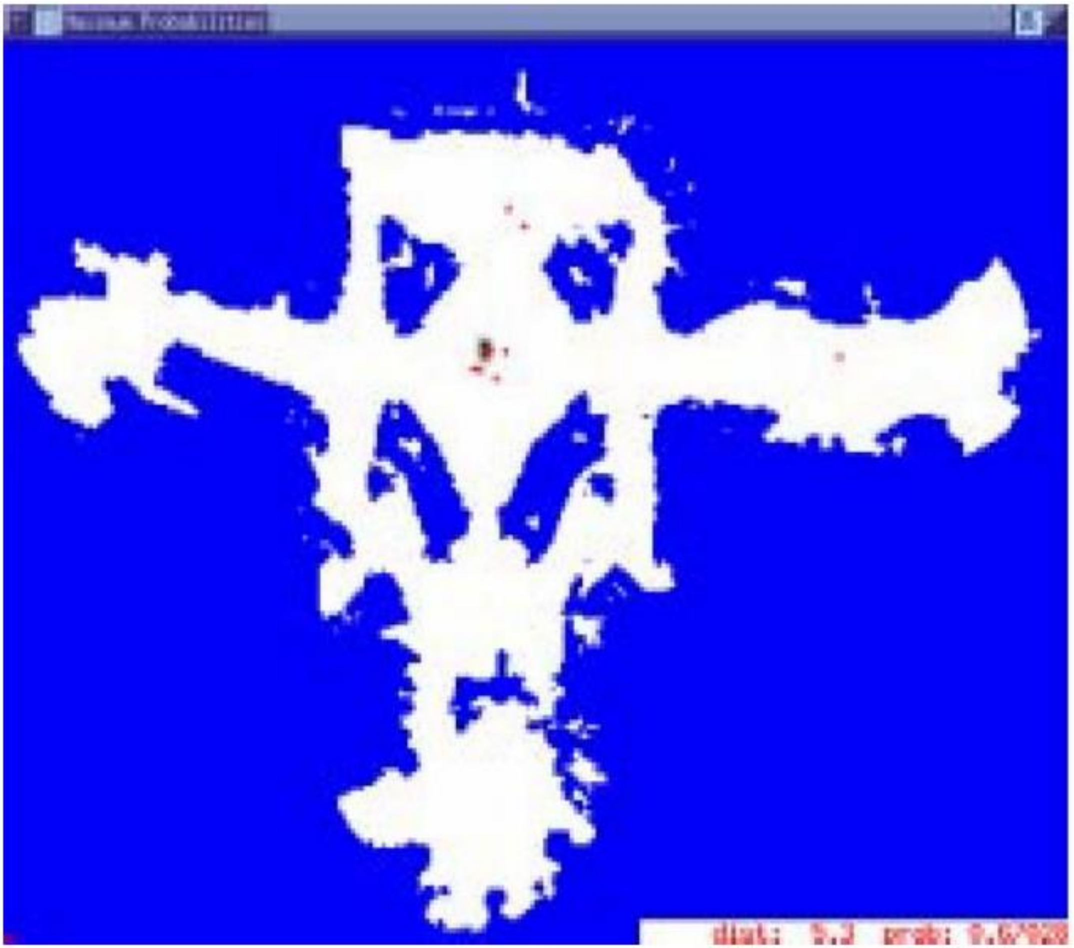


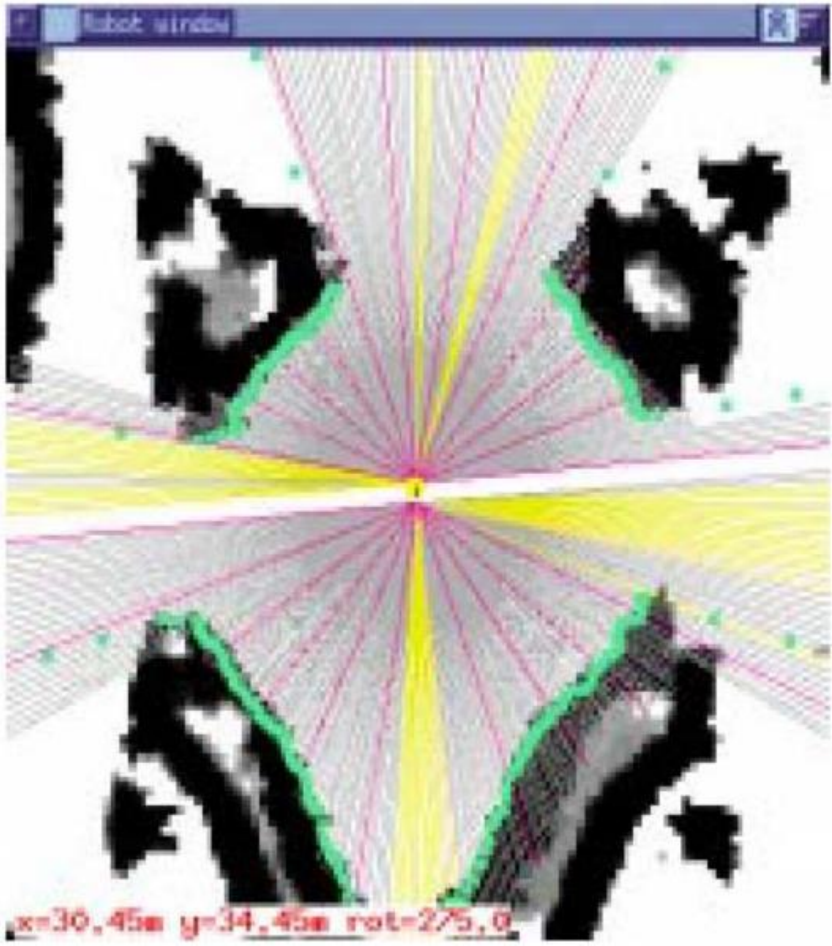
Markov Localization



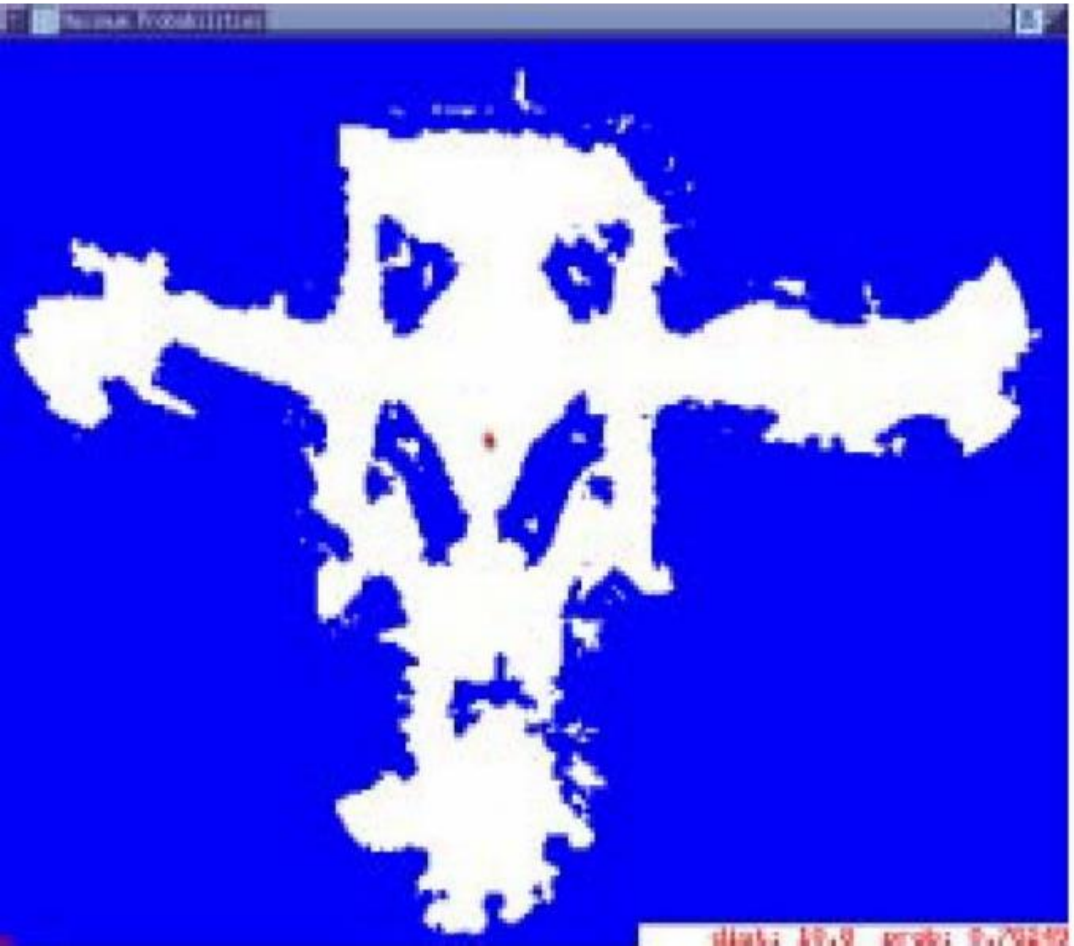


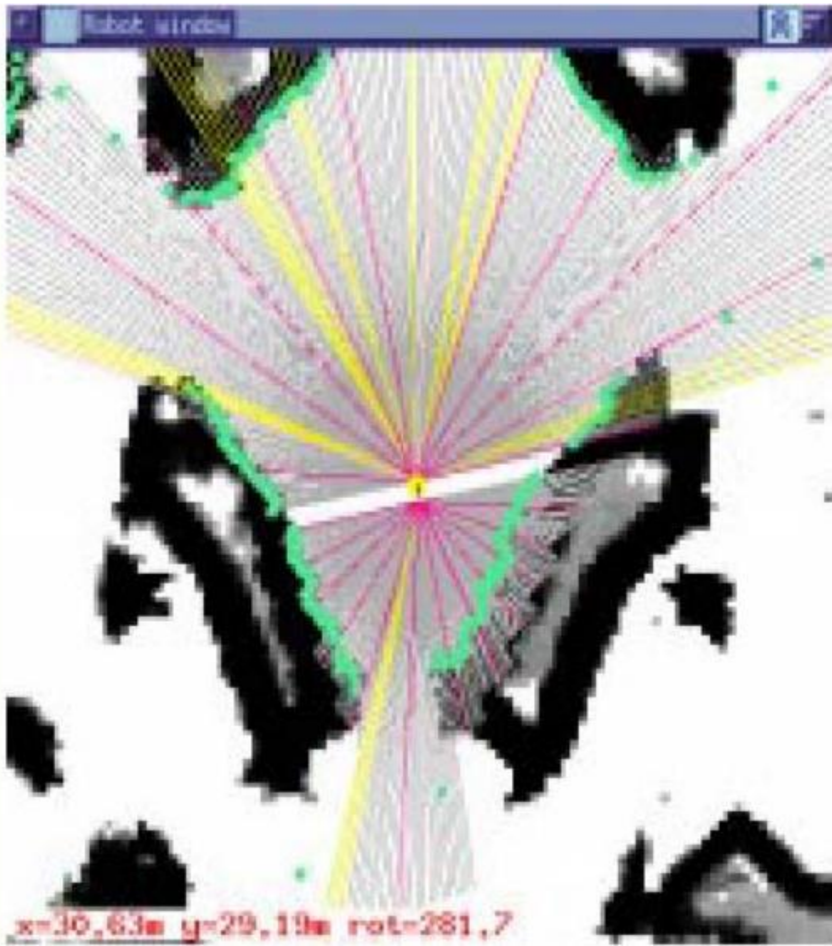
Markov Localization





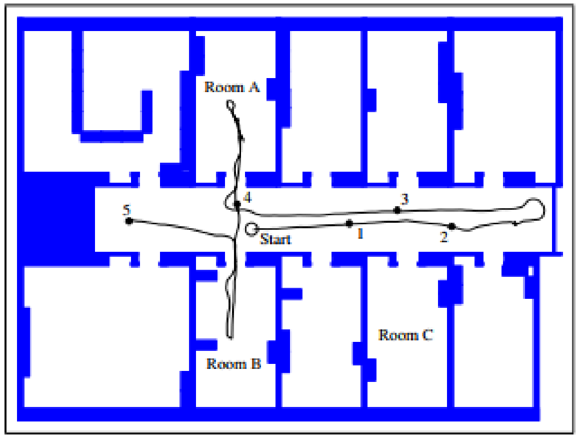
Markov Localization



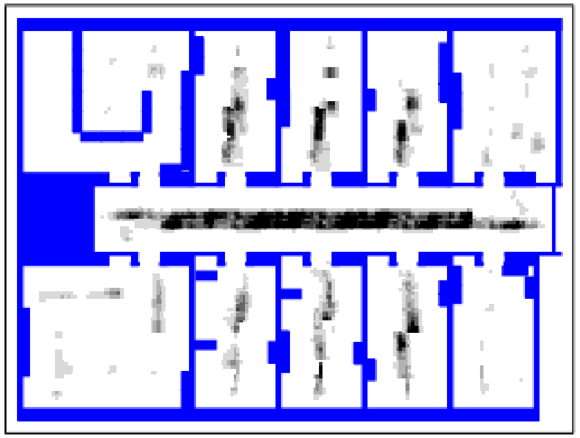


Global localization in an office environment using sonar data. (a) Path of the robot. (b) Belief as the robot passes position 1. (c) After some meters of robot motion, the robot knows that it is in the corridor. (d) As the robot reaches position 3 it has scanned the end of the corridor with its sonar sensors and hence the distribution is concentrated on two local maxima. While the maximum labeled I represents the true location of the robot, the second maximum arises due to the symmetry of the corridor (position II is rotated by 180deg relative to position I). (e) After moving through Room A, the probability of being at the correct position I is now higher than the probability of being at position II. (f) Finally the robot's belief is centered on the correct pose.

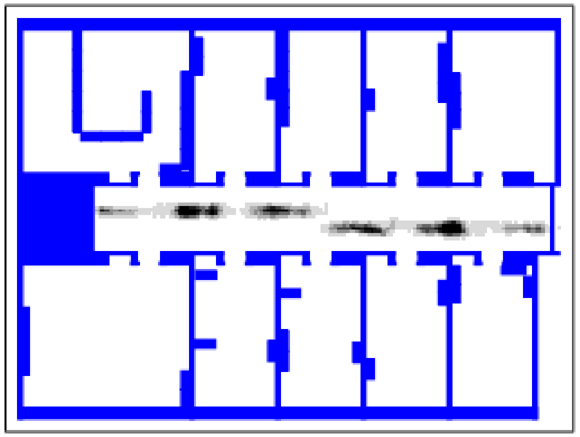
(a) Path and reference poses



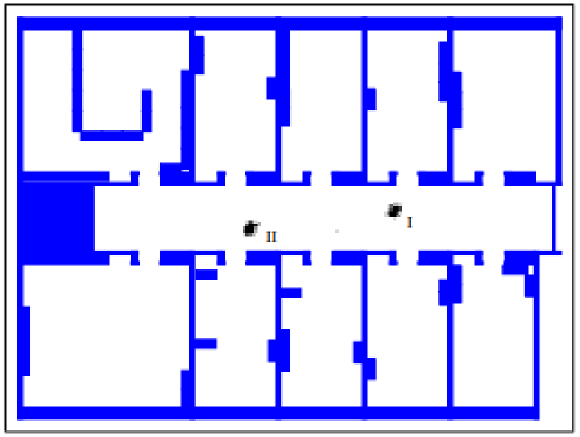
(b) Belief at reference pose 1



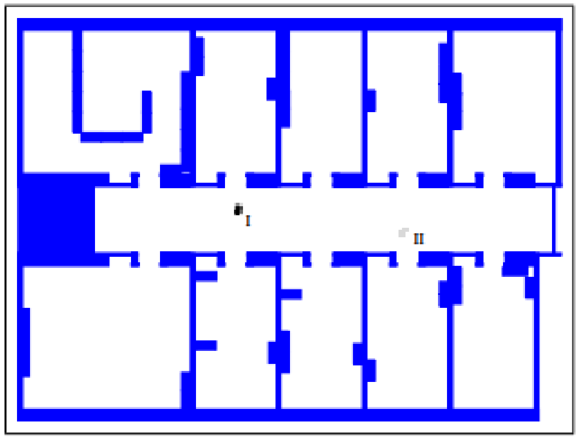
(c) Belief at reference pose 2



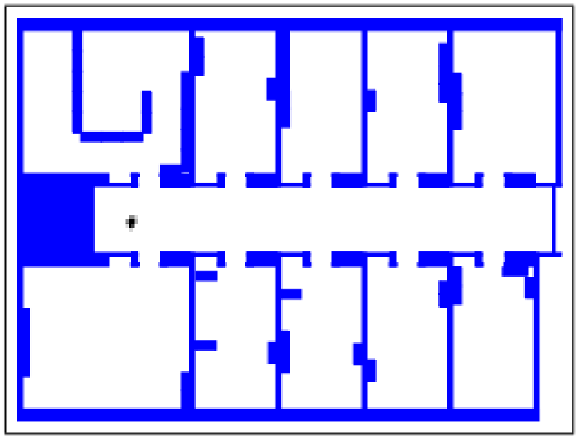
(d) Belief at reference pose 3

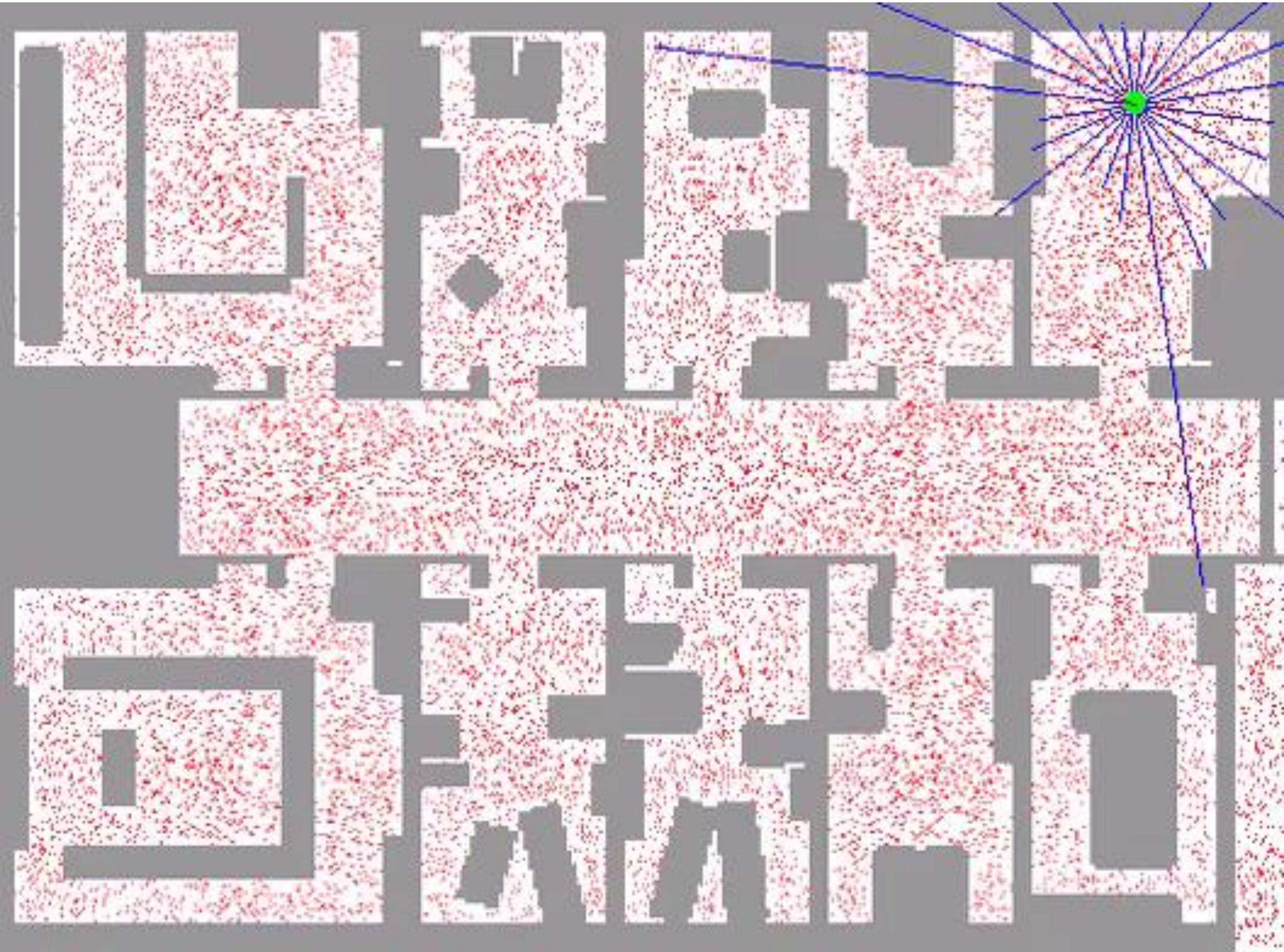


(e) Belief at reference pose 4



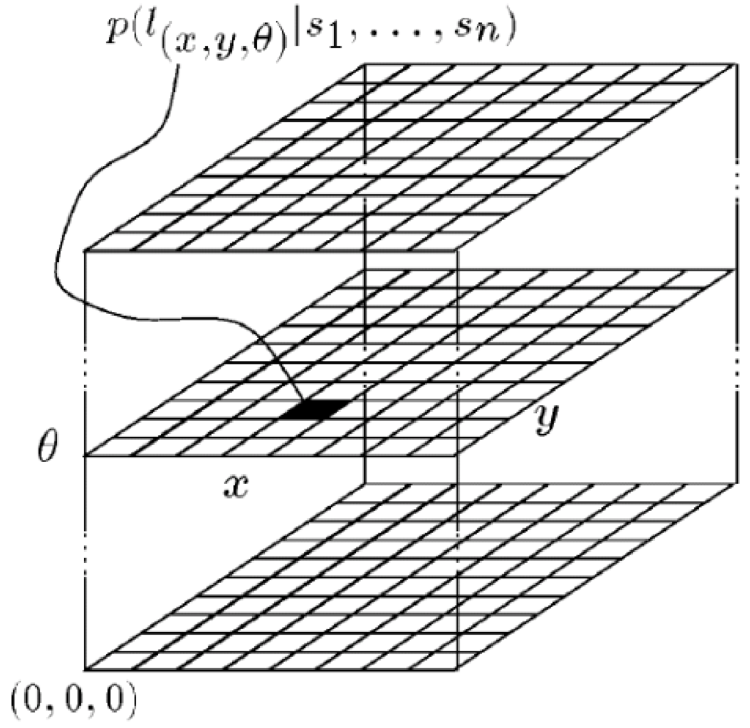
(f) Belief at reference pose 5





Drawbacks of Markov Localization

- For a planar robot pose is (x,y,θ), needs 3D grid map
  - Memory-heavy!
- Prediction and measurement steps will update all cells
  - Computationally heavy!
- Example:
  - 30x30 m environment
  - cell size 0.1 m x 0.1 m x 1 deg
  - 300 x 300 x 360 = 32.4 M cells!



Markov Localization: Using Randomized Sampling to Reduce Complexity (*Monte Carlo Localization*)

- A fine fixed decomposition grid results in a huge state space
- We want to reduce the number of states that are updated in each step

Randomized Sampling / Particle Filters

- Approximate belief state by keeping track of a subset of all states.
- The sampling process is weighted to keep 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

Algorithm

Randomly initialize N samples, each on a random pose l, each associated to a probability value $p(l) = 1/N$

Then iterate:

- SEE step for a given sensor reading i:
    - for every sample, update probability as $p(l) = p(i \mid l)\, p(l)$
    - normalize such that the total probability is 1
- ACT step for a given command (or odometry reading) o:
    - N times:
        - randomly draw a sample l' from the previous sample set, with a likelihood given by each sample's probability value $p(l')$
        - generate a new sample at pose l according to $p(l \mid l', o)$
    - assign every generated sample a probability value of $p = 1/N$

Improvement: after every step, add a few samples at random poses, so that we will eventually recover from a situation in which we are completely lost (i.e. no particles are close to our true pose)

Monte Carlo Localization: Action model (ACT)

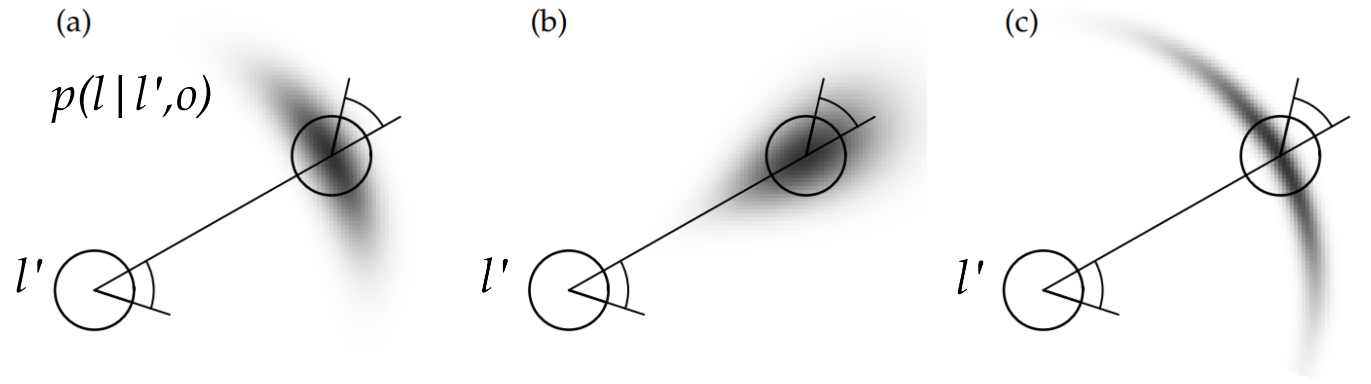


Figure 5.8 The odometry motion model, for different noise parameter settings.

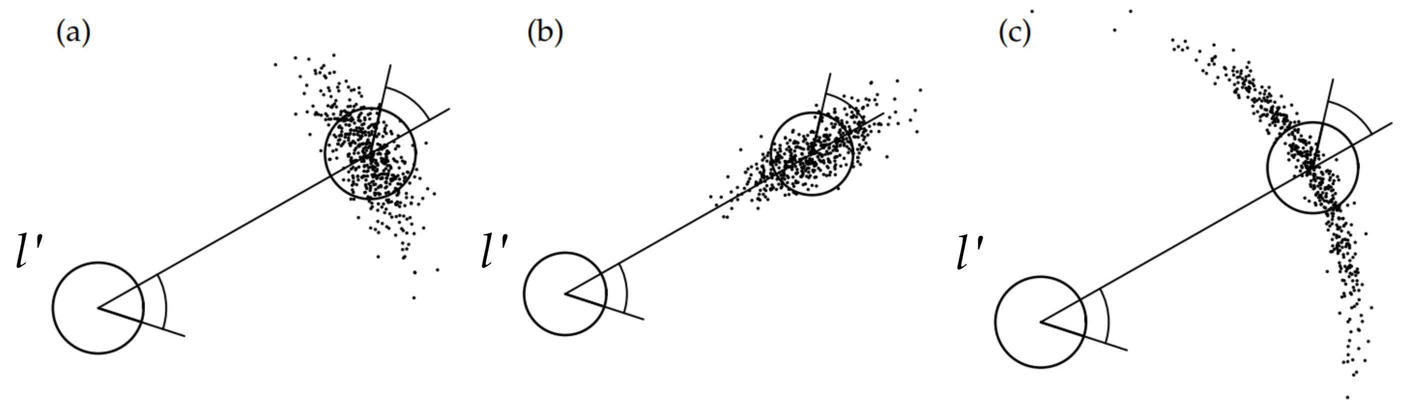


Figure 5.9 Sampling from the odometry motion model, using the same parameters as in Figure 5.8. Each diagram shows 500 samples.

Note: only x,y are represented but each sample has its own theta as well

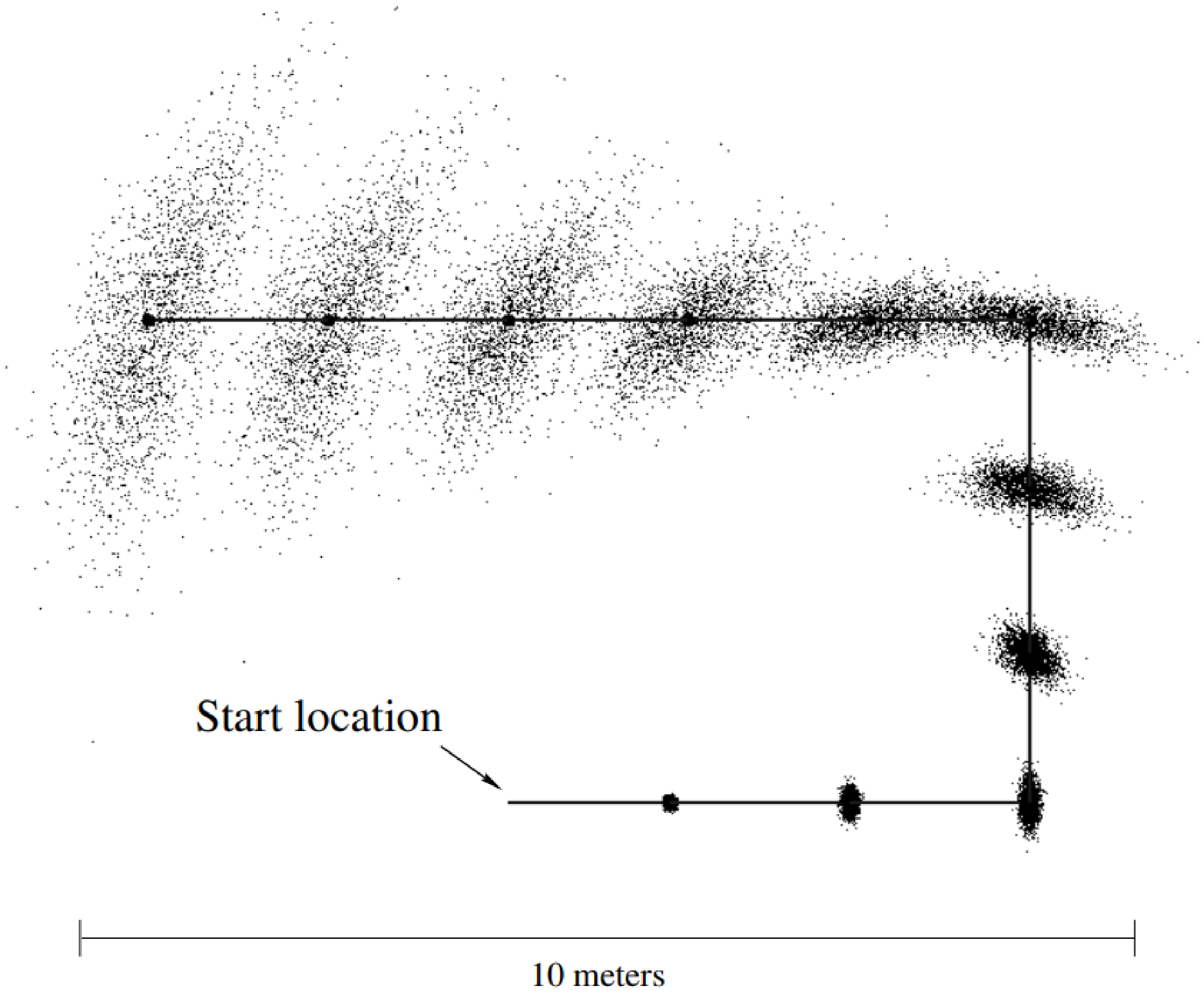
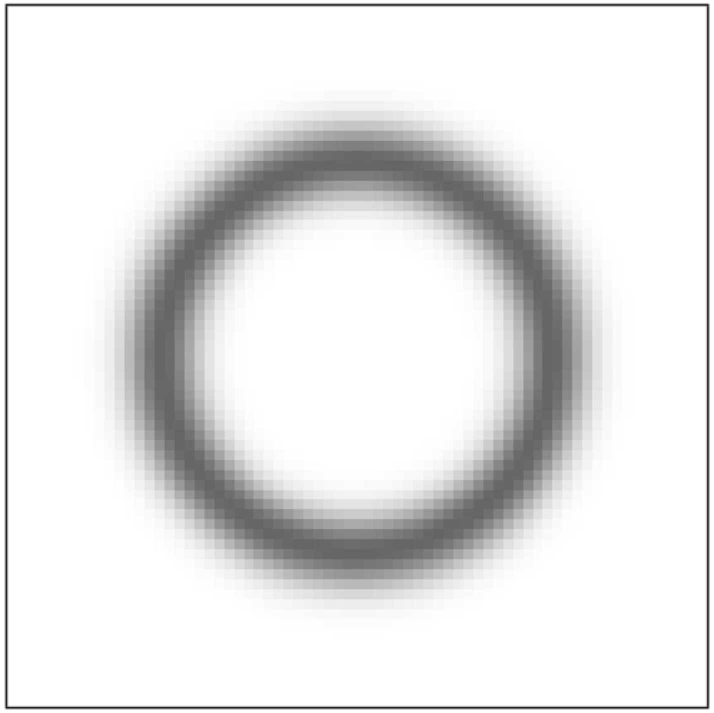


Figure 5.10 Sampling approximation of the position belief for a non-sensing robot. The solid line displays the actions, and the samples represent the robot's belief at different points in time.

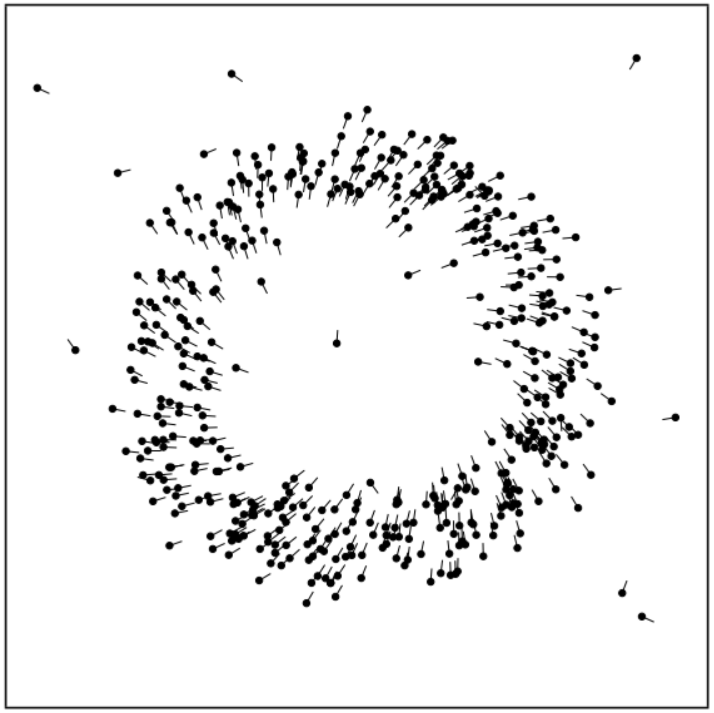
Landmark detection model

(a)



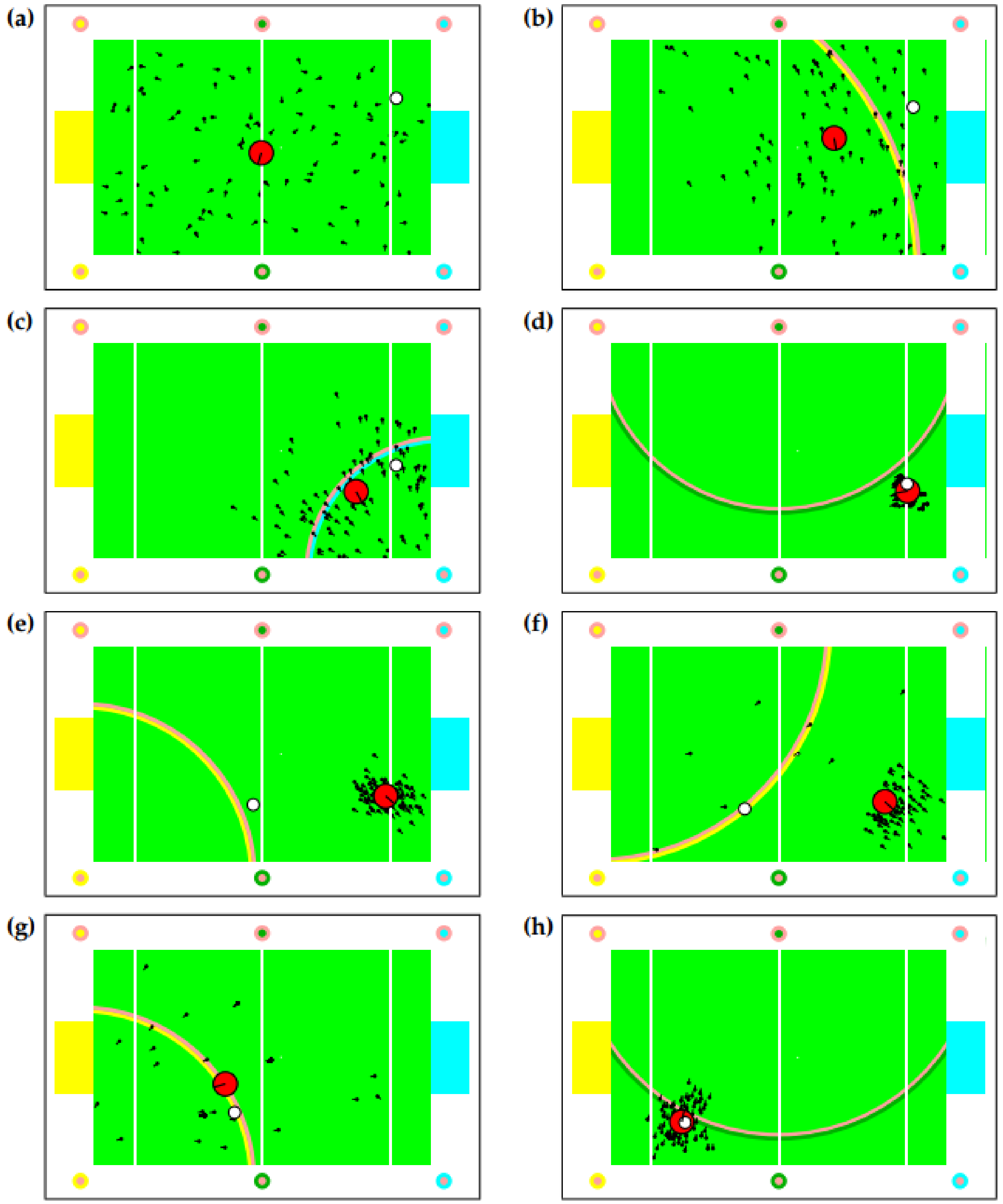
Posterior distribution of the robot's pose given that it detected a landmark in 5m distance and 30deg relative bearing (projected onto 2-D).

(b)



Sample robot poses generated from such a detection. The lines indicate the orientation of the poses.

Monte Carlo localization with random particles. Each picture shows a particle set representing the robot's position estimate (small lines indicate the orientation of the particles). The large circle depicts the mean of the particles, and the true robot position is indicated by the small, white circle. Marker detections are illustrated by arcs centered at the detected marker. The pictures illustrate global localization (a)–(d) and relocalization (e)–(h)



Two approaches to localization

Markov

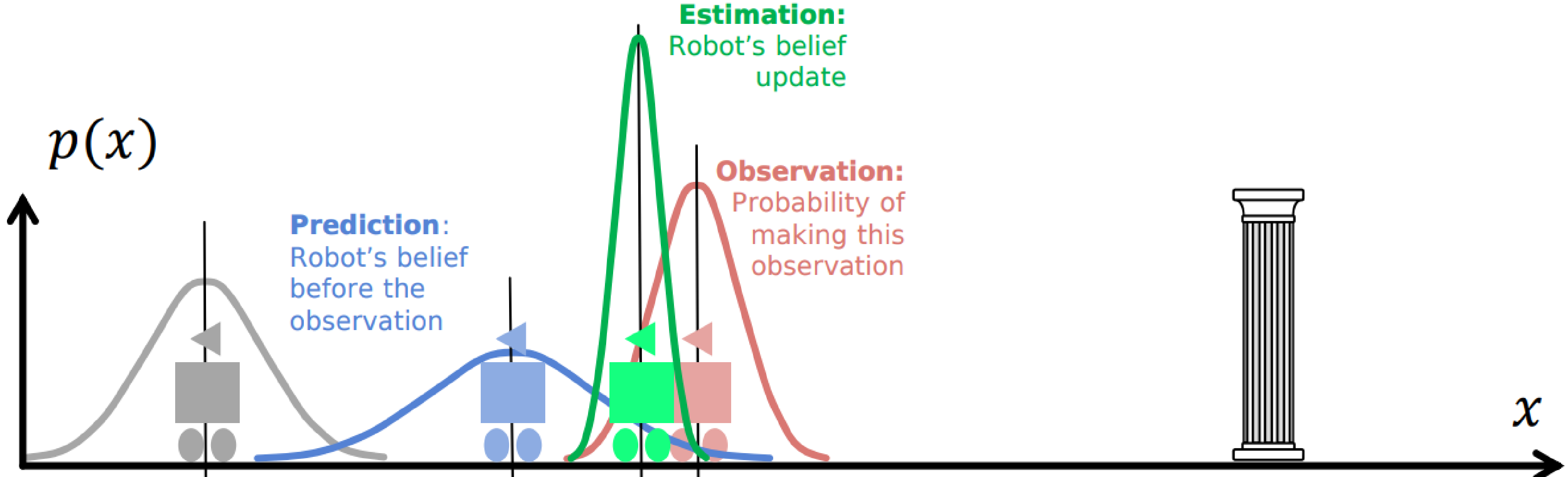
- Maintains multiple estimates of robot position
- Localization can start from any unknown position
- Can recover from ambiguous situations
- However, to update the probability of all positions within the state space requires a discrete representation of the space (grid);
- if a fine grid is used (or many estimates are maintained), the computational and memory requirements can be large.

Kalman

- Single estimate of robot position
- Requires known starting position of robot
- Tracks the robot and can be very precise and efficient
- However, if the uncertainty of the robot becomes too large (e.g. due collision with an object) the Kalman filter will fail and the robot becomes “lost”

Kalman approach (continuous, recursive and compact)

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


Estimation:
Robot's belief update
$p(x)$
Observation:
Probability of making this observation
Prediction:
Robot's belief before the observation
x