CS3027,CS5059: Localisation[1]

N. Oren
n.oren@abdn.ac.uk

University of Aberdeen

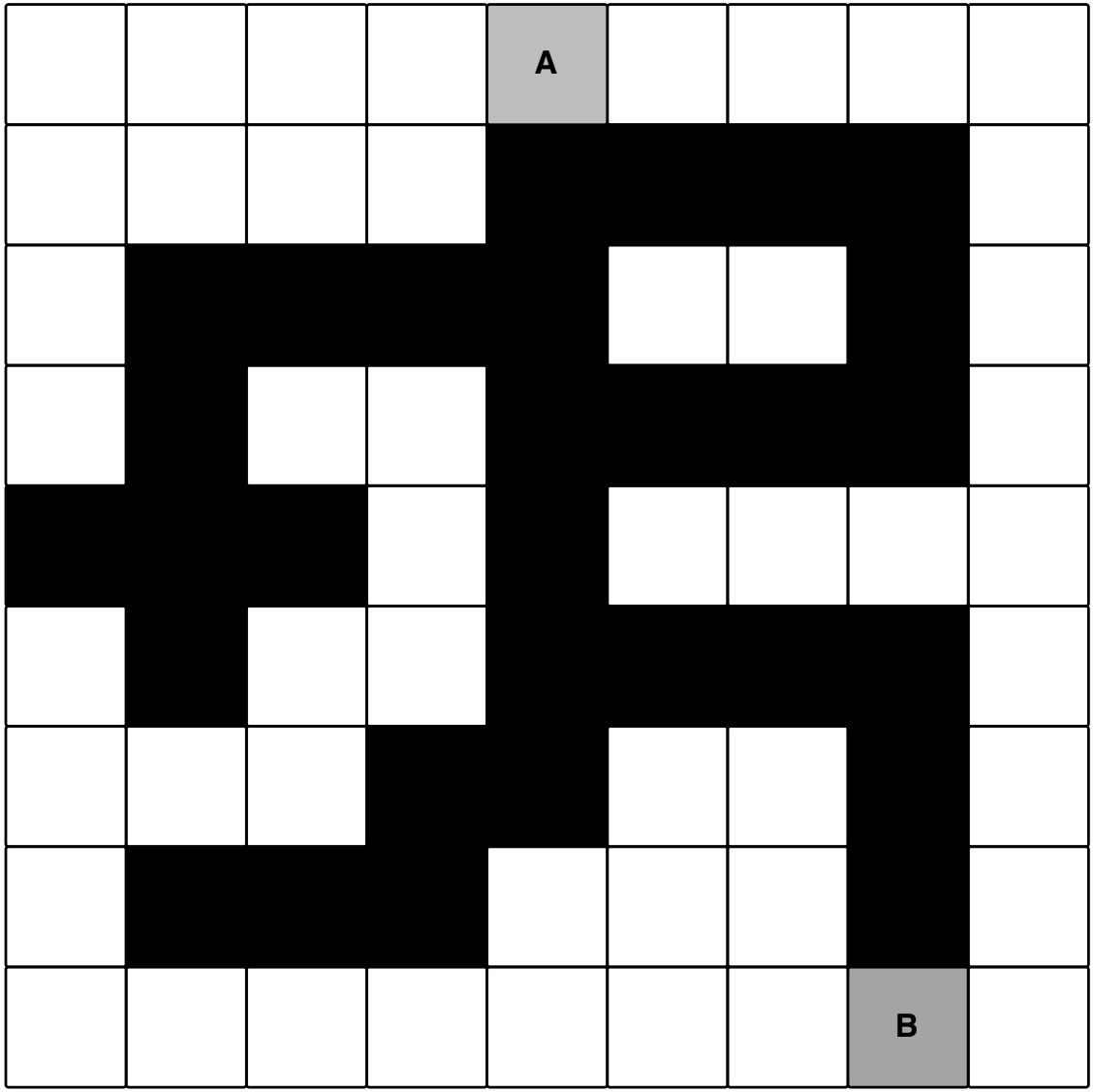
[1] Some slides and images are from Probabilistic Robotics, Thrun et al.

Navigation

- `http://www.youtube.com/watch?v=ysueHeSvg1Q`
- Successful navigation typically requires the robot to
  - perceive (interpret sensor data)
  - localise (determine its position in the environment)
  - utilise cognition (decide how to act to achieve its goals)
  - perform motor control (alter motor outputs to achieve trajectory)

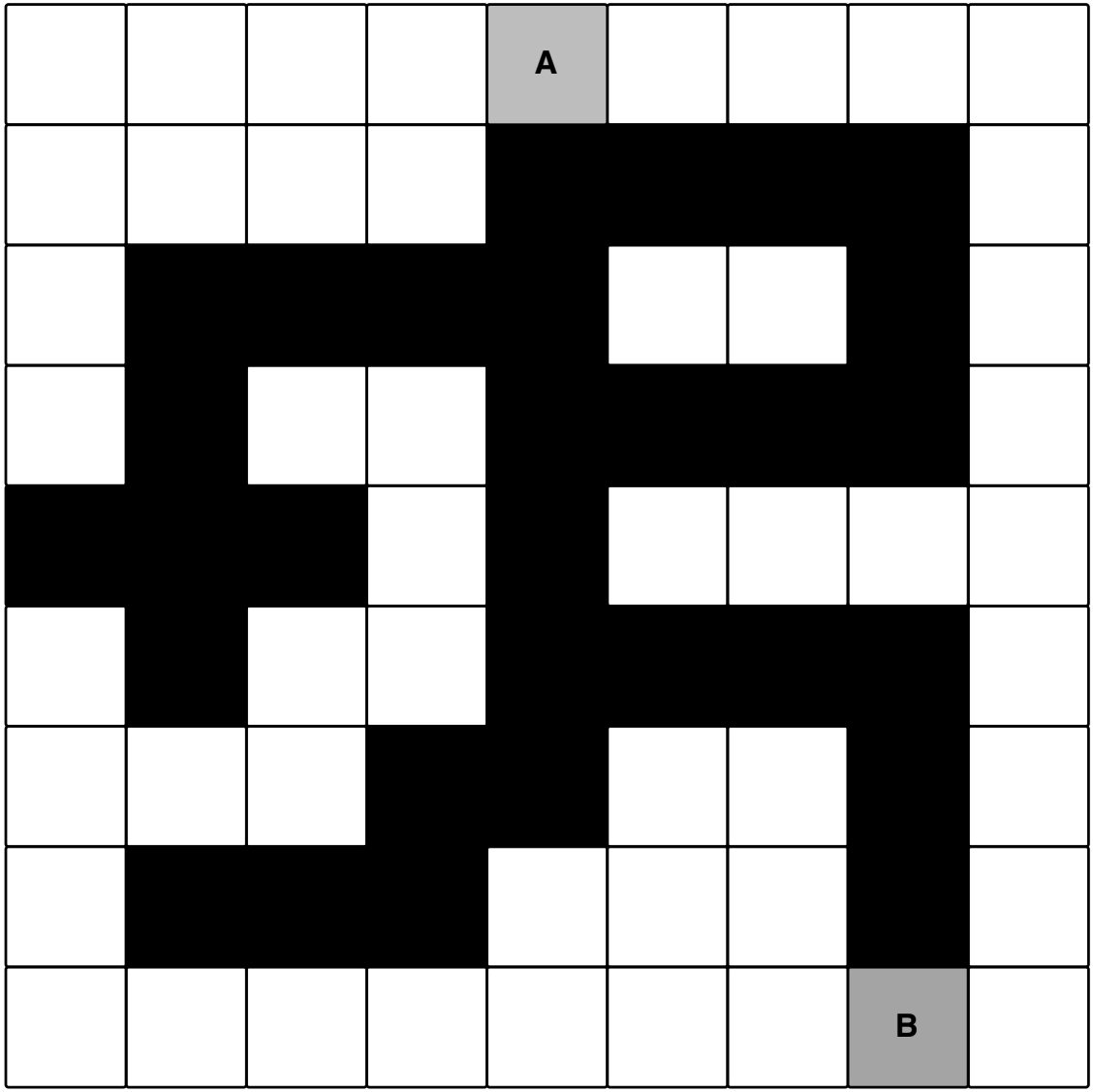
Localisation

- Does the robot need to know where it is to travel from A to B?



Localisation

- Does the robot need to know where it is to travel from A to B?
- Sensors are needed to navigate without hitting obstacles; detect goal location
- Follow left wall



Behaviour Based Navigation

- "Following the left wall" is an example of behaviour based navigation.
- BBN: do not create maps, but design behaviours that together result in desired motion.
- Avoids explicit reasoning about localisation and position, as well as explicit path planning.
- It works in some environments, but not in others.
- Can be inefficient
- Relatively simple to implement
- BBN is often environment or location specific.

Map Based Navigation

- Unlike behaviour based navigation, map based navigation asks how goals can be achieved given a known map.
- "Simplest" form of the problem: given a known starting position, how can we tell where the robot is at every point in time?
  - We can use wheel odometry or dead reckoning.
  - Is this enough?
  - Are there other alternatives?
- How do we represent the map?
- How do we represent the robot's position in the map?

Problem Categories

- Global localization — "Kidnapped robot problem"
  - The robot is not told its initial position
  - Position must be estimated from scratch
- Position tracking
  - Initial position is known
  - Errors in odometry must be taken into account when movement occurs

Localisation Techniques

- Localisation based on external inputs (sensors, beacons and landmarks)
- Odometry
- Map based localisation (with no access to external sensors or artificial landmarks)
  - Probabilistic map based localisation

Sensors, beacons and landmarks

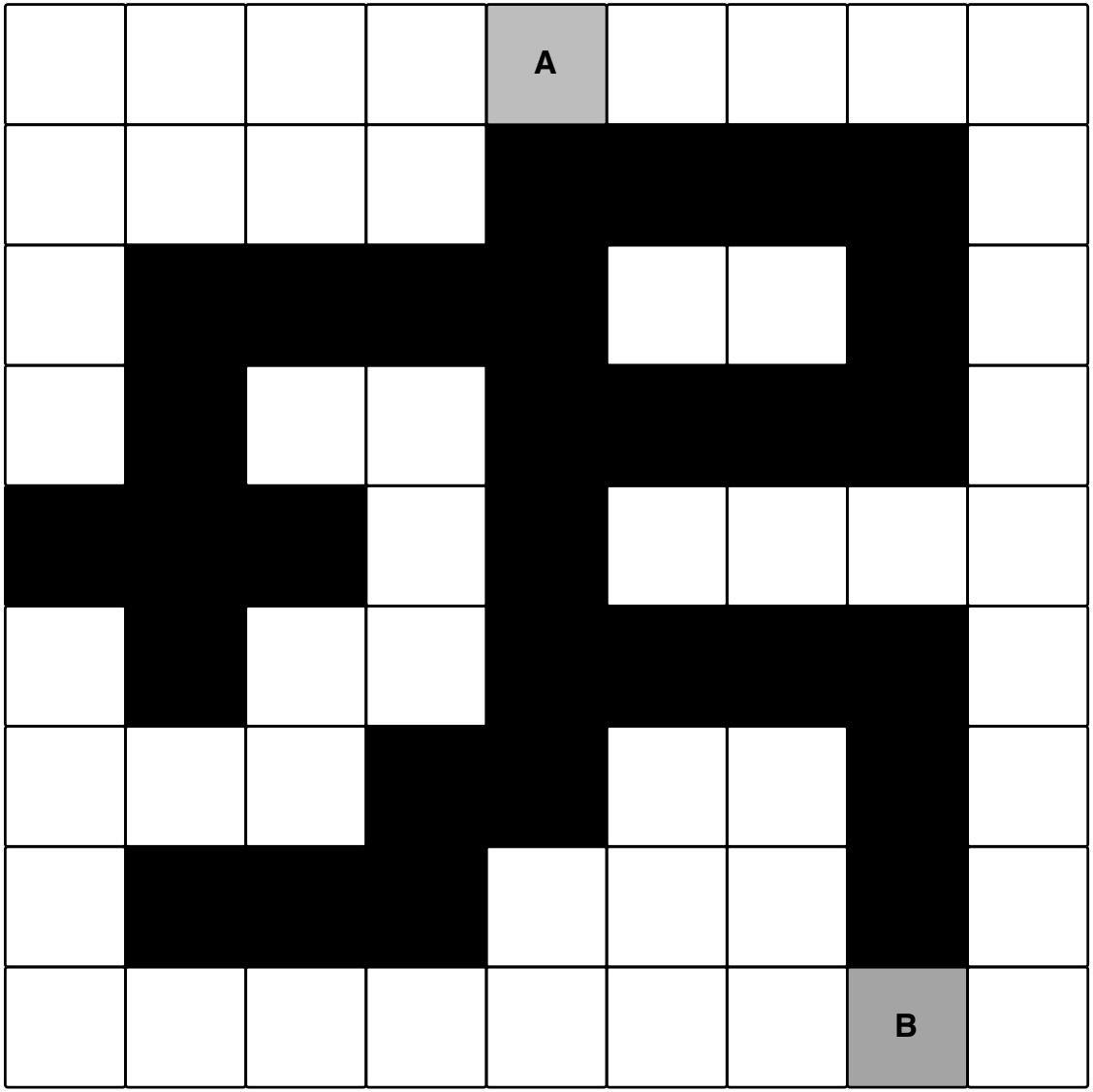
- A robot can measure distance/orientation to a set of known beacons
- An external camera with a known position can detect the robot(s) and compute its position (particularly if multiple cameras are used).
- Robots can detect unique markers on the ground or ceiling placed at a known map position.

Odometry

- Given the amount each robot wheel has turned, determine where the robot is.
- Dead reckoning: given a velocity estimate, determine where the robot is in relation to previous position.

Map based localisation

- Identify position given unique features observed on a known map.
- E.g. given that you see paths North, South, East and West of you



Localisation Challenges

- GPS doesn't solve everything
  - Doesn't work indoors
  - Might need to know relative distance to objects which move (other people)
  - Errors at the meter scale means finer grain localisation might be necessary (e.g. to avoid driving off a cliff).
- We may need to both localise and build a map of the environment (SLAM).
- So we must integrate odometry and sensors into the localisation process.
  - Sensor noise
  - Sensor aliasing — sensors return non-unique perceptions

Sensor Noise

- Sensor noise is mostly influenced by the environment, e.g. due to different illumination levels, as well as by the measurement process itself (e.g. errors in timings of ToF sensors, interference etc).
- Solutions:
  - Take multiple readings
  - Model sensor noise
  - perform sensor fusion

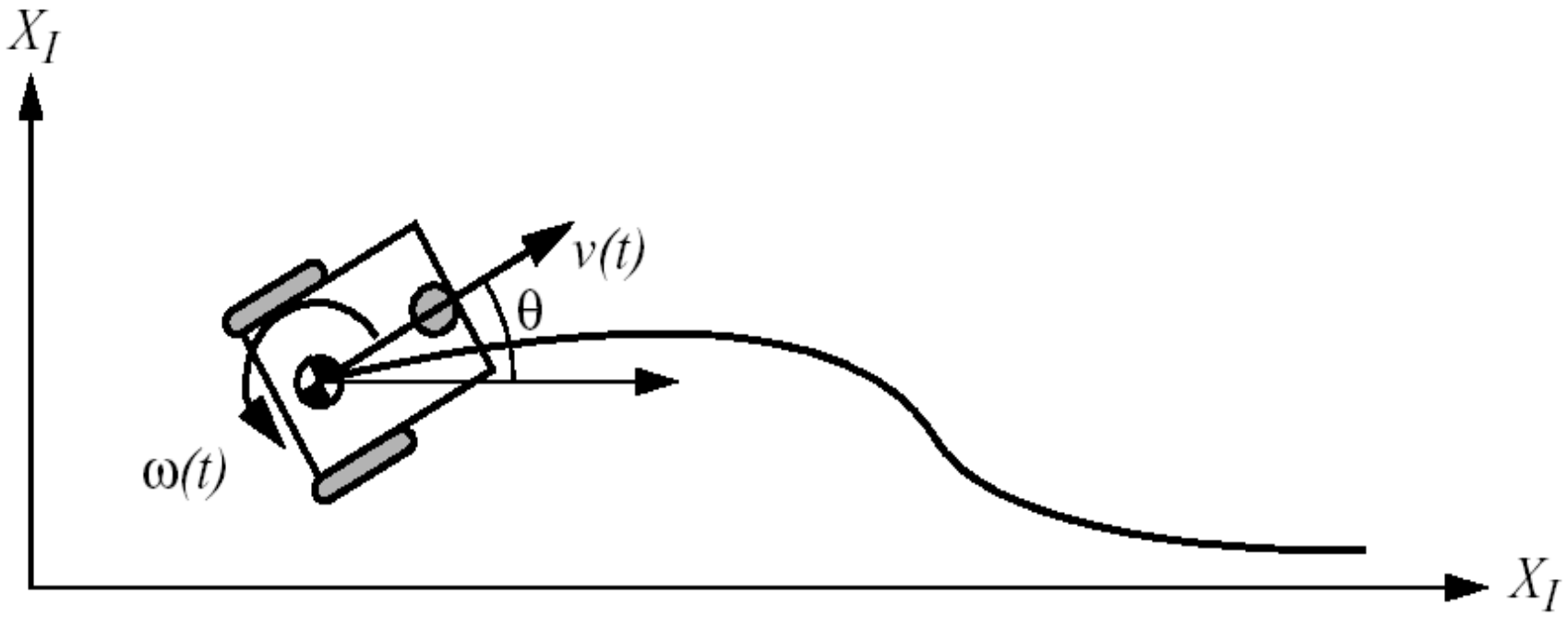
Effector Noise

- A single action taken by the robot can have slightly different results — robot effectors introduce uncertainty about the future state.
- In odometry, position is updated based on wheel sensors (encoders) only.
- In dead reckoning, heading sensors are also taken into account.
- Movement is integrated to get the robot's position.
- Approach is simple, but integration of errors mean they grow over time.
- The use of additional sensors can reduce the accumulated errors, but this cannot prevent them growing more slowly.

Errors

- Three broad types of geometric errors:
  - Range error: Computed path length (distance) of robot movement (due to sum of wheel movement).
  - Turn error: Computed turn amount due to robot movement (due to difference of wheel movement).
  - Drift error: difference in the error of the wheels leads to an error in angular orientation.
- Turn and drift errors dominate range errors.
- Major error sources:
  - Limited resolution during integration (of time, of measurements).
  - Wheel misalignment (deterministic)
  - Unequal wheel diameter (deterministic)
  - Wheel slip, variations in wheel contact point etc.
- See UMBmark for robot calibration procedures.

A simple error model



$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

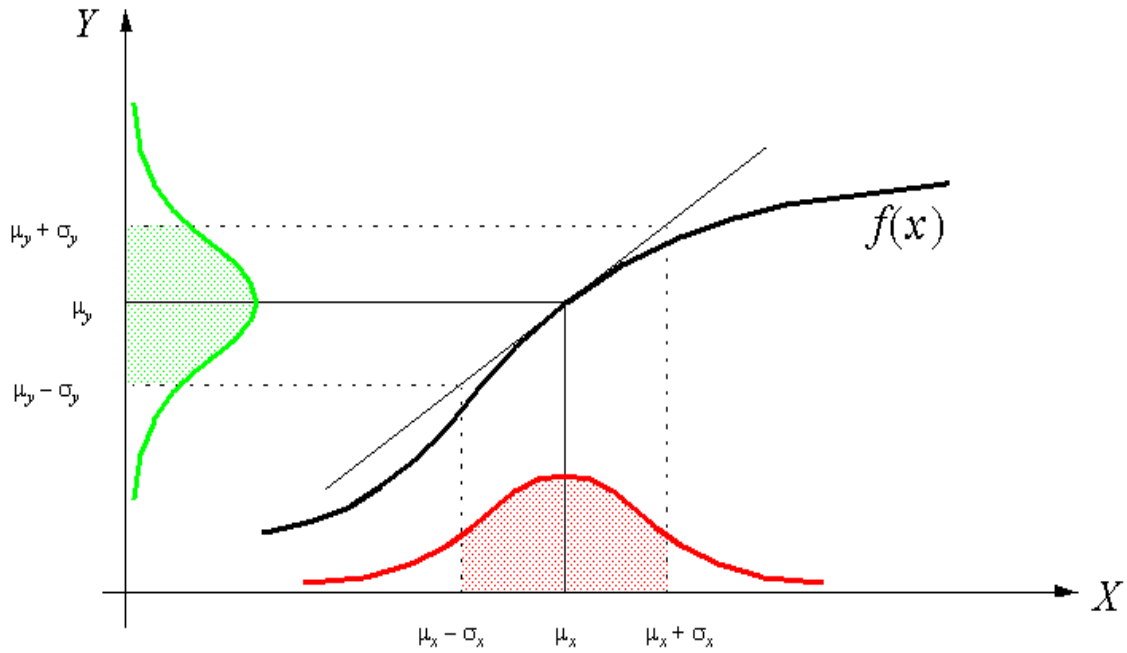
- Given an initial pose p, we estimate position by integrating movement — summing incremental travel distances.

An aside

- Given n inputs with known probability distributions, and m outputs, what are the probability distributions of the outputs if they depend (via function f_i) on the inputs?

$$C_Y = F_X C_X F_X^T$$

- C_X, C_Y are covariance matrices.
- $F_X = \nabla f$ is the Jacobian
- Result is obtained from a Taylor expansion of f_i



A simple error model

- For a fixed discrete sampling interval Δt, incremental travel distances $(\Delta x, \Delta y, \Delta\theta)$ per interval are

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2) \qquad \Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \qquad \Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

- $\Delta s_r, \Delta s_l$ are right and left wheel interval travel distances. b is the inter-wheel distance.

A simple error model

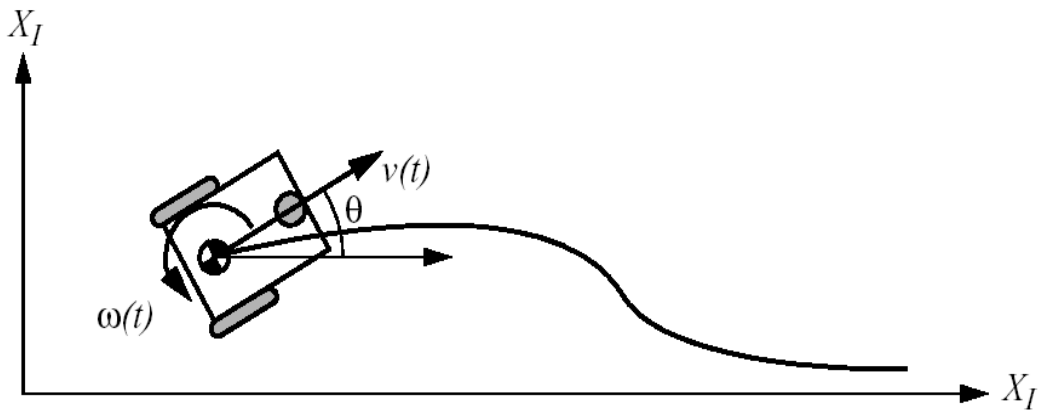
$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}$$



A simple error model

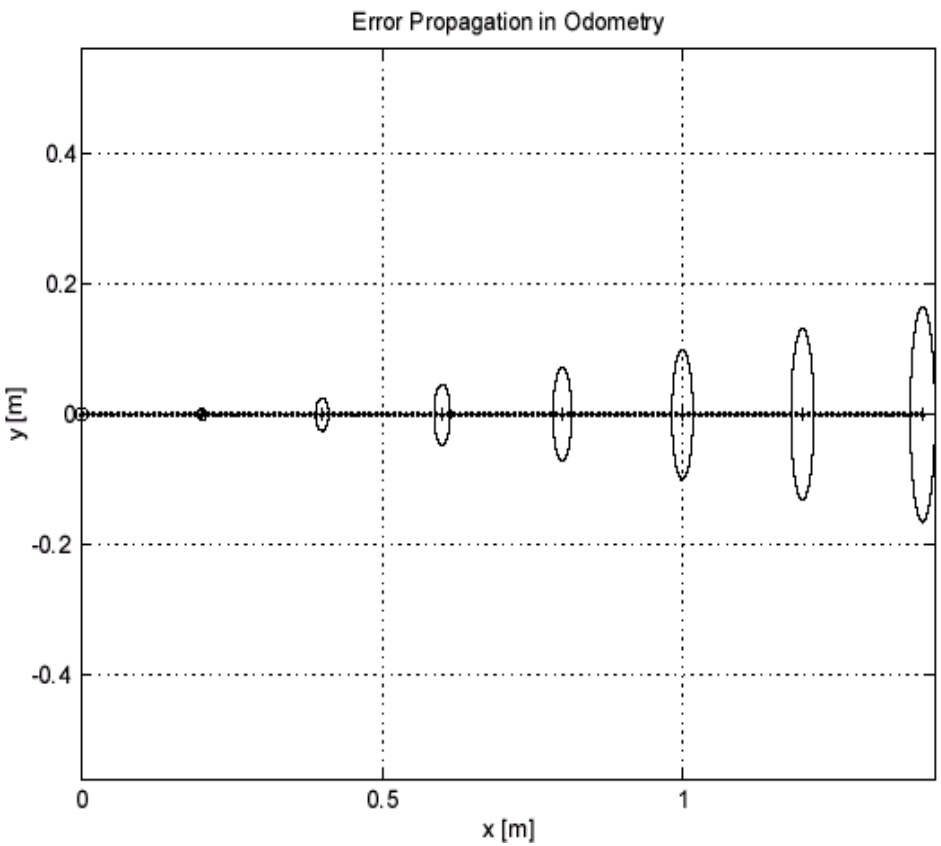
$$\Sigma_{\Delta} = covar(\Delta s_r, \Delta s_l) = \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}$$

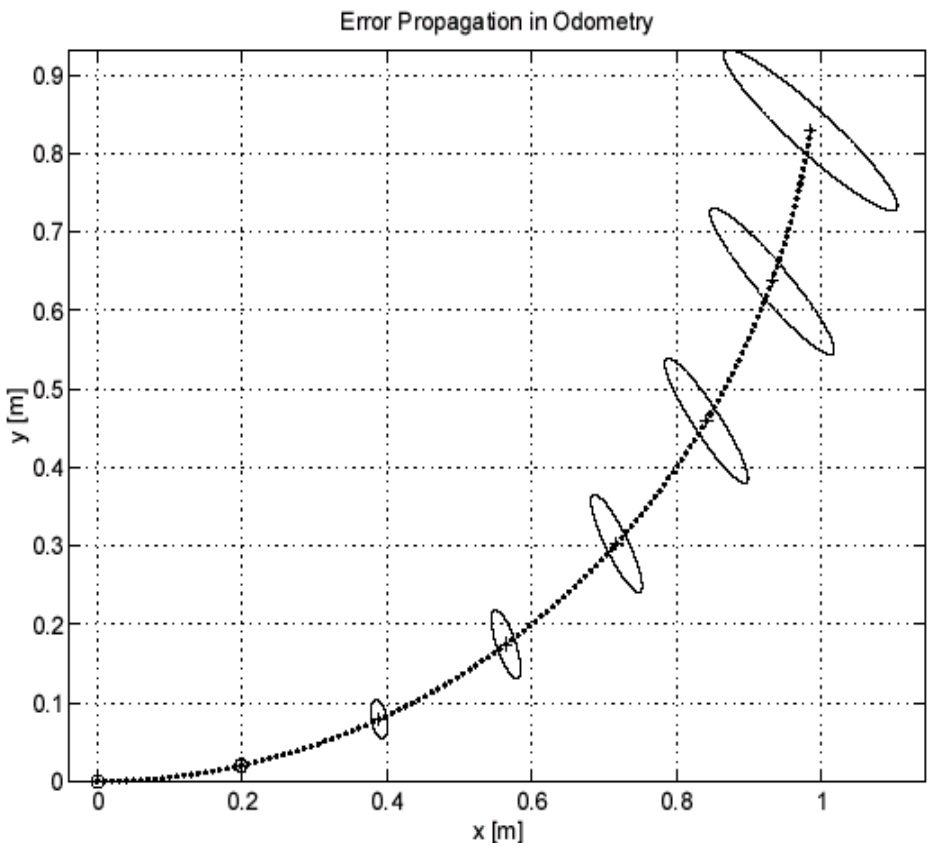
$$\Sigma_{p'} = \nabla_p f \cdot \Sigma_p \cdot \nabla_p f^T + \nabla_{\Delta_{rl}} f \cdot \Sigma_{\Delta} \cdot \nabla_{\Delta_{rl}} f^T$$

$$F_p = \nabla_p f = \nabla_p (f^T) = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta + \Delta\theta/2) \\ 0 & 1 & \Delta s \cos(\theta + \Delta\theta/2) \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_{\Delta_{rl}} = \begin{bmatrix} \frac{1}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{\Delta s}{2b}\sin\left(\theta + \frac{\Delta\theta}{2}\right) & \frac{1}{2}\cos\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2b}\sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{2}\sin\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{\Delta s}{2b}\cos\left(\theta + \frac{\Delta\theta}{2}\right) & \frac{1}{2}\sin\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{\Delta s}{2b}\cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix}$$

Error Model plot





- Note: errors perpendicular to movement direction are larger.
- Ellipse is not perpendicular to movement direction.
- More complex error models exist.

Representation

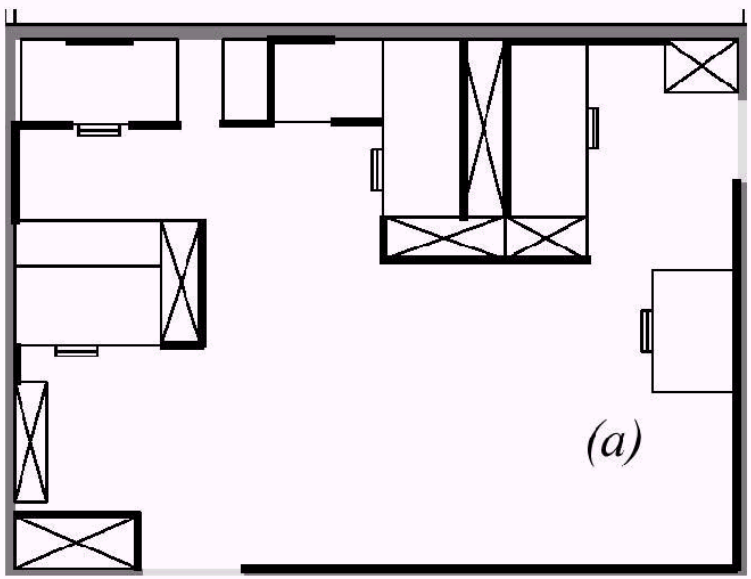
- Fundamental difference between various map-based localisation systems lies in the way the robot represents the map and its location therein.
- Map representation
  - What aspects of the environment are contained in the map?
  - At what level of resolution?
- Location representation
  - Does the robot identify a single unique current position as its position?
  - A set of possible positions?
  - How are these ranked?
- Tradeoffs between architectural, computational complexities and localisation accuracy.

Environment Representation

- Decisions regarding environmental representation can impact choices for robot position representation.
  - Precision of the map must match the precision required to achieve robot goals.
  - Precision of the map and features must match precision and data types returned by sensors.
  - Map representation complexity has direct impact on complexity of reasoning.
- Representations can be continuous or discrete.

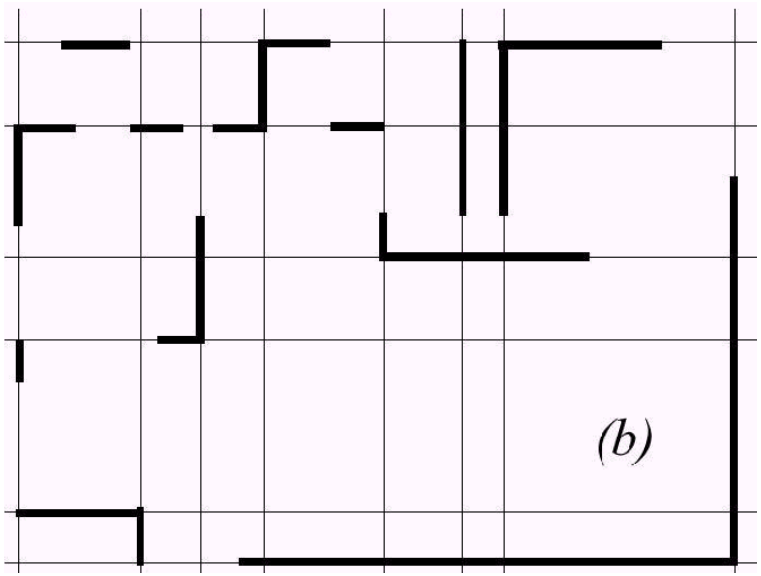
Continuous Representations

- Polygons represent all obstacles in environment.
- Closed world assumptions: if no obstacles are shown on map, there is no obstacle.
- Approximations required to represent real world areas (e.g. trees).
- Typically done in 2D only
- Abstract away from things like colour, texture etc to reduce complexity.



Continuous Representations

- Straight line representation captures only what laser range finder can detect.
- Bounded regions on straight lines make computation easy as robot can detect lines
- Simplification of real world (e.g. no textures of walls).
- Position for single-hypothesis position captured by a single continuous point in space providing high accuracy.
- For multiple hypotheses, one can either utilise encompass regions of the map
- Or provide a discretised set of possible points

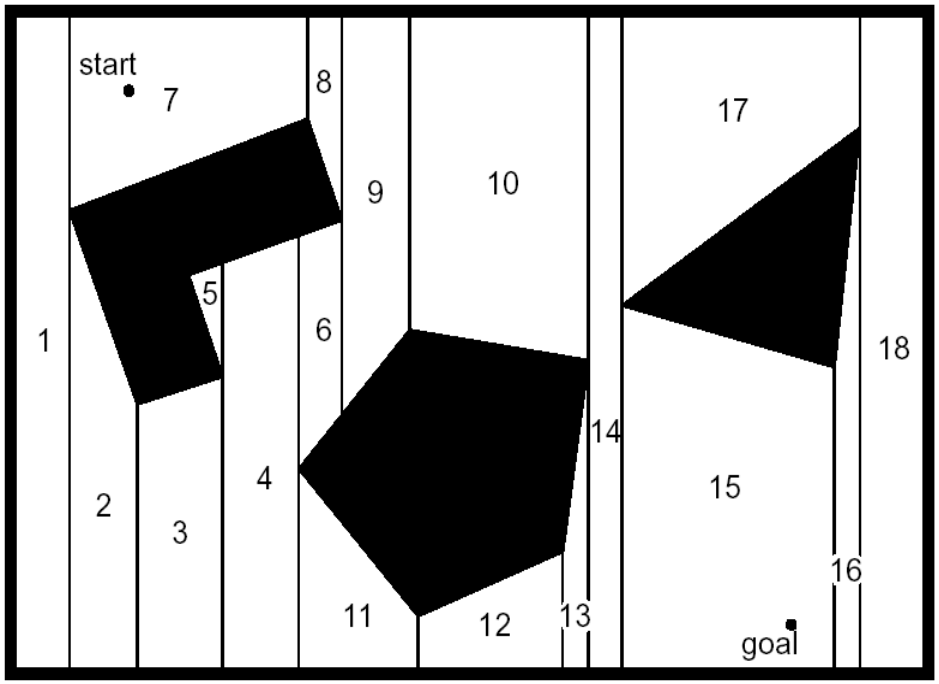


Continuous Representations Summary

- Continuous maps provide(potentially) high accuracy for environment and robot position.
- Disadvantage is potentially high computational cost.
- Mitigated by abstracting and capturing only relevant details, together with the closed world assumption.
- The most common abstraction involves decomposition
- Aim of an abstracted map is to capture useful features, discarding all others.
- Ideal decomposition reduces complexity while yielding results as good as or better than a full detail map (due to computational savings).

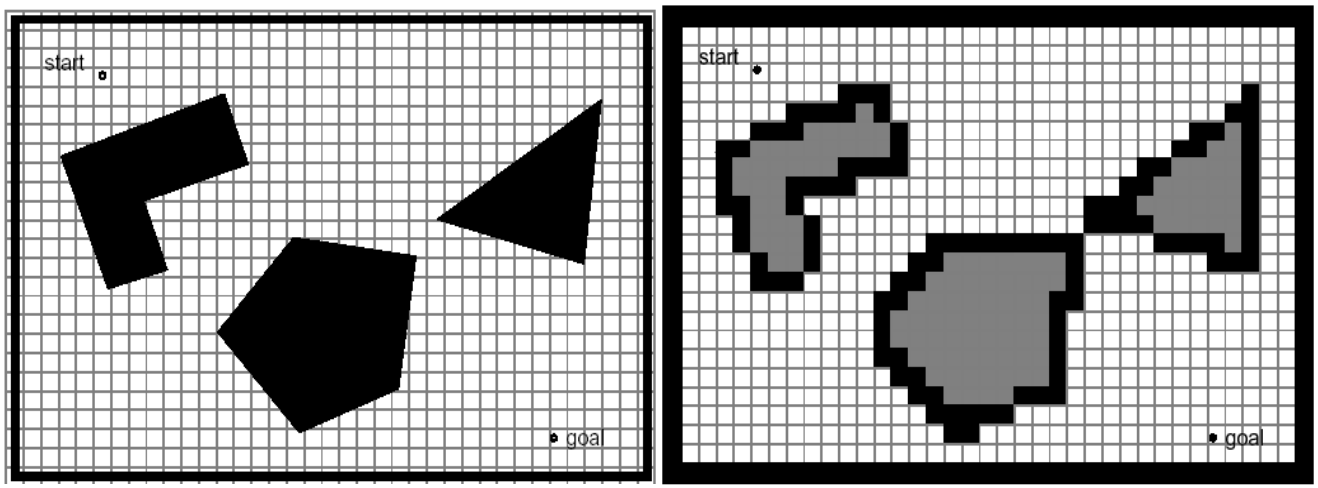
Exact Decomposition

- Tessellate space into areas of free space.
- Space efficient; each area is stored as a single "node".
- Core assumption: location within free space does not matter; the ability to traverse to adjacent areas is important.
- Resulting graph captures adjacencies.
- Not useful in many situations.
- Algorithm for performing exact decomposition?



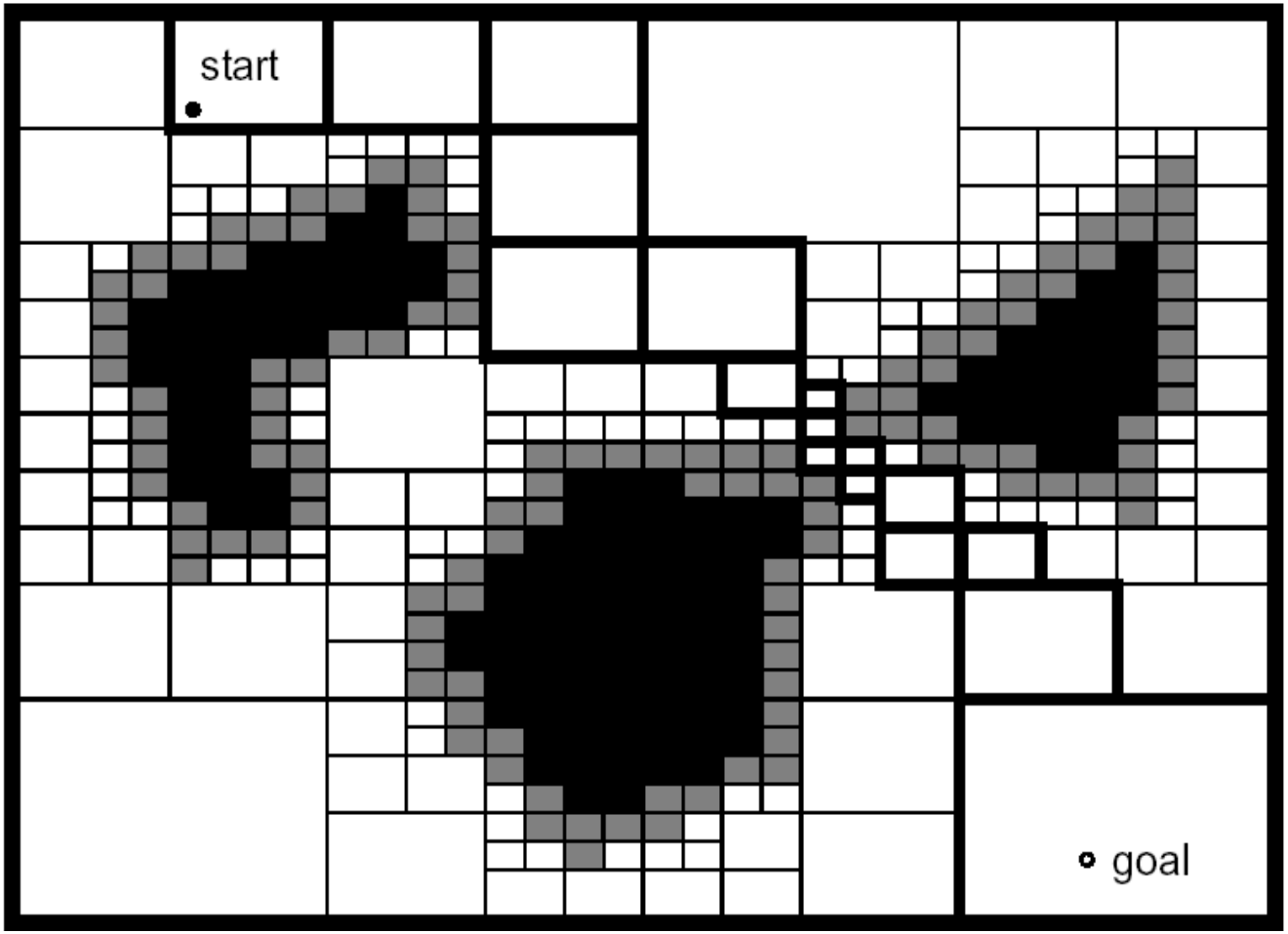
Fixed Decomposition

- Tesselate continuous environment into a discrete approximation.
- Narrow passageways can be lost.
- Fixed decomposition sound (we can determine if no route exists).
- Fixed decomposition is not complete (if a route exists, it may not be found).



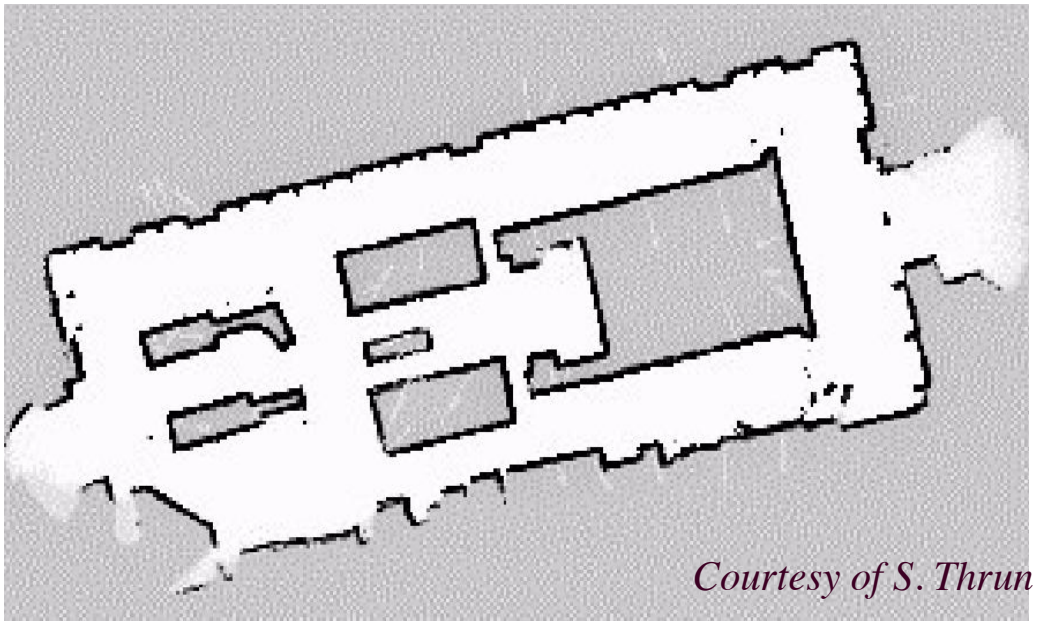
Adaptive Cell Decomposition

- Areas close to obstacles are sampled at finer scales.
- Suggest an algorithm for performing such decomposition?



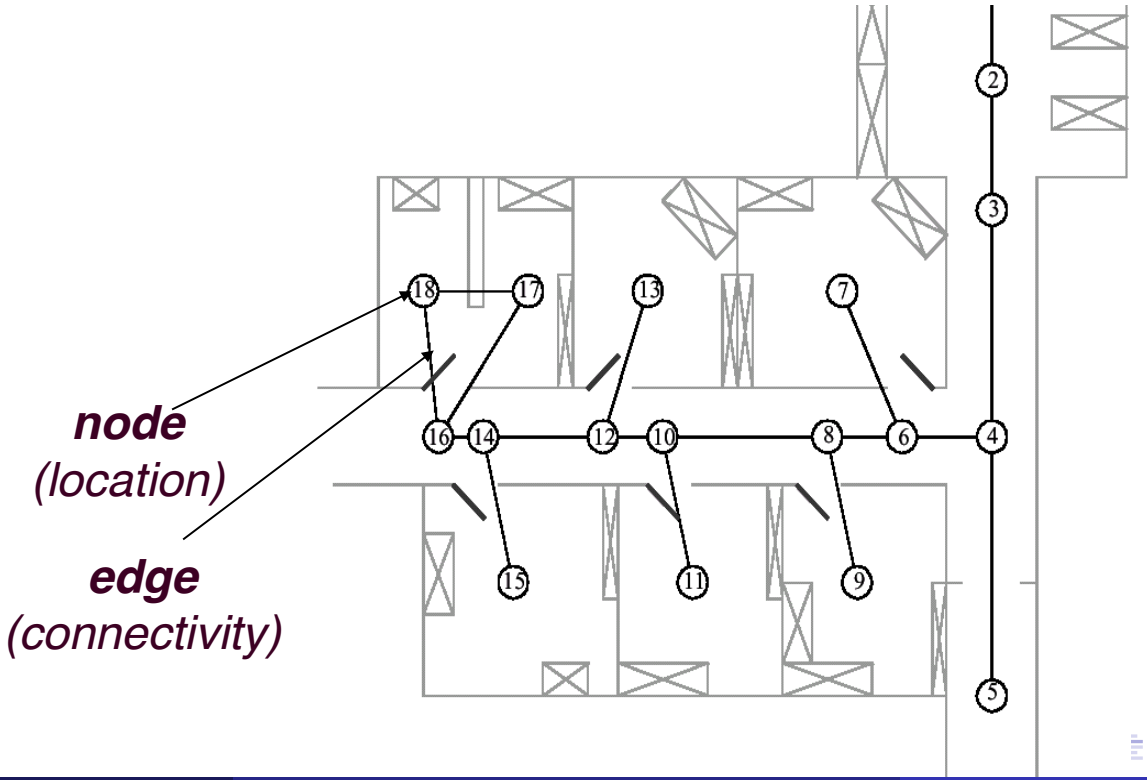
Occupancy grid

- Environment is a grid
- Each cell is filled or empty.
- Using a range sensor, cell values increment when a cell is "struck" (shown occupied by the ranger).
- Above a threshold, the cell is marked as filled.
- Ranger passing through a cell reduces its value, accounting for transient objects.



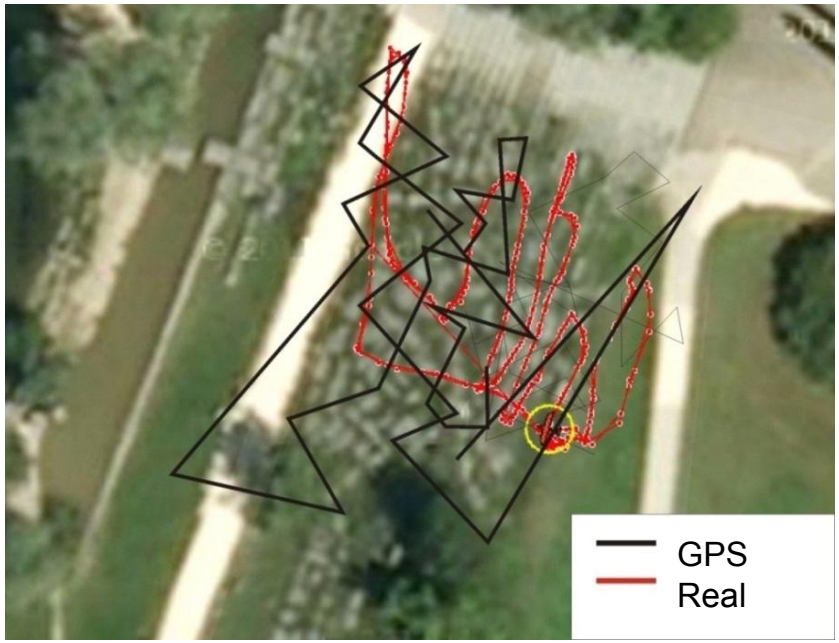
Topological representation

- Map is represented as a graph identifying adjacent areas.
- Nodes are not of a fixed size or specification of free space.
- Nodes are areas that the robot can detect it is entering or leaving using its sensors, e.g. SIFT features, unique ranges, etc.
- Robot must have means to travel between nodes.
- including distance information on the edges can help with reasoning.



Challenges

- Differentiate between permanent and transient obstacles (improvements in computer vision?)
- Most mapping techniques assume that the environment is free of moving objects when mapping is performed.
- How does one localise in the presence of moving objects (e.g. robotic tour guide surrounded by people)?
- How do we localise in very open spaces?



Map Representation

http://www.youtube.com/watch?v=wV8frjLqtIA

Belief Representation

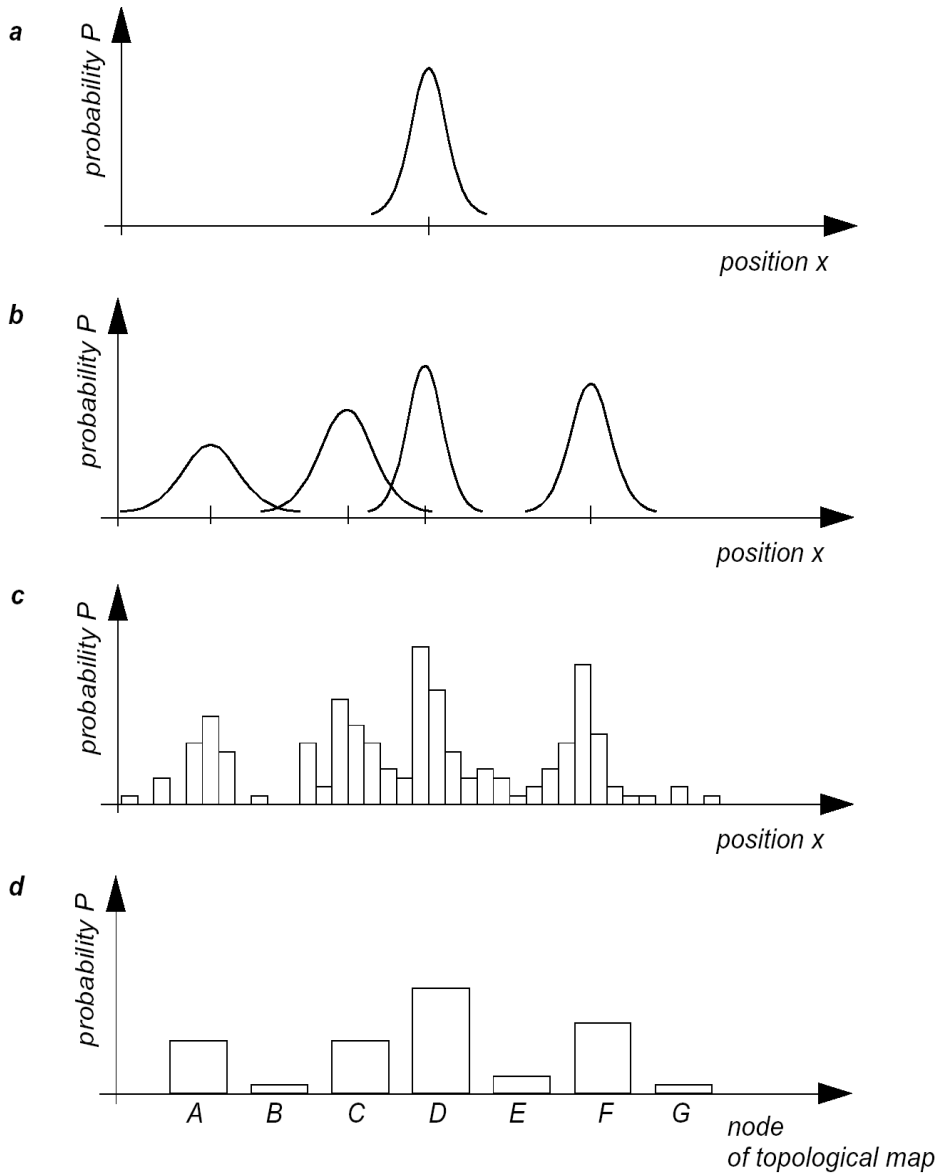
- How do we represent robot position on a map?

a Continuous map with single hypothesis probability distribution.

b Continuous map with multiple hypotheses probability distribution

c Discretised map with probability distribution

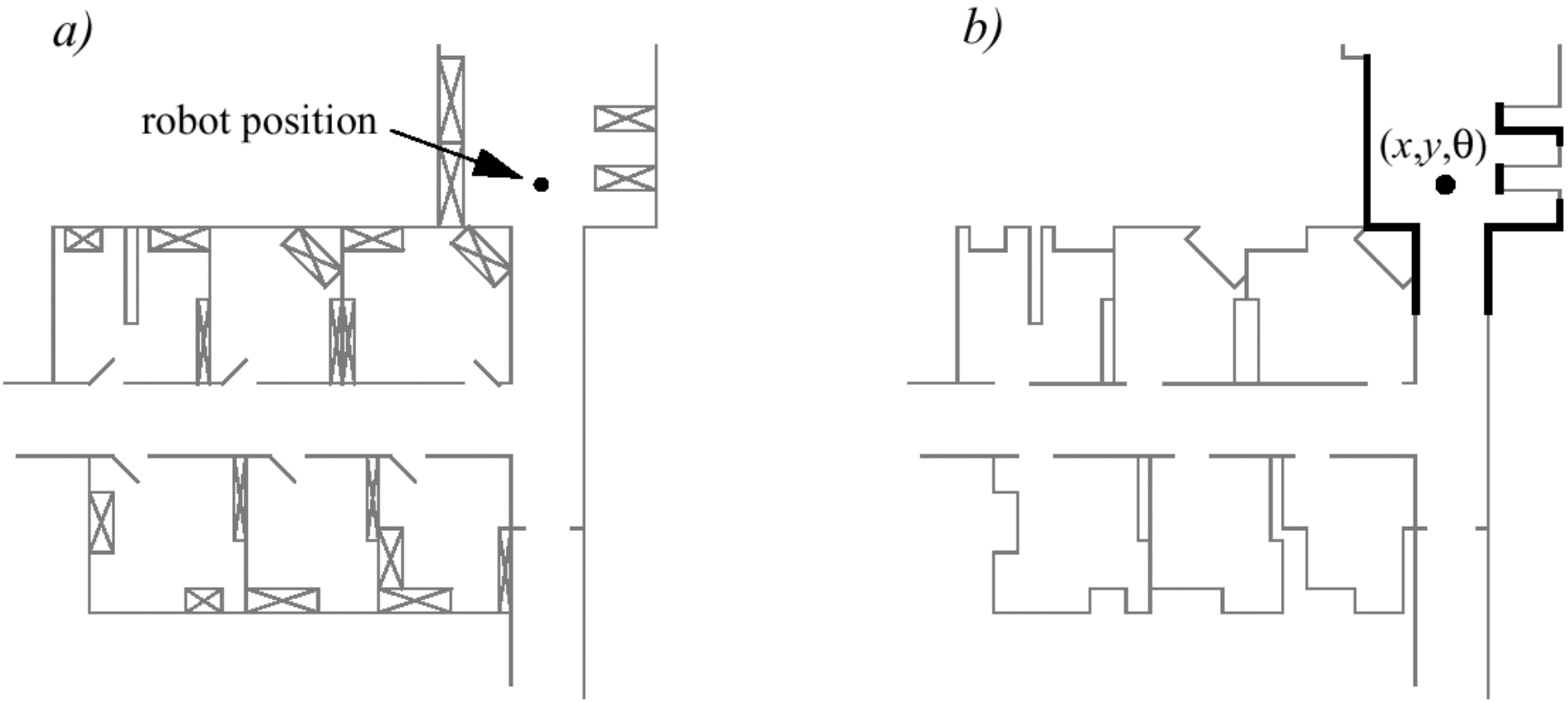
d Discretised topological map with probability distribution



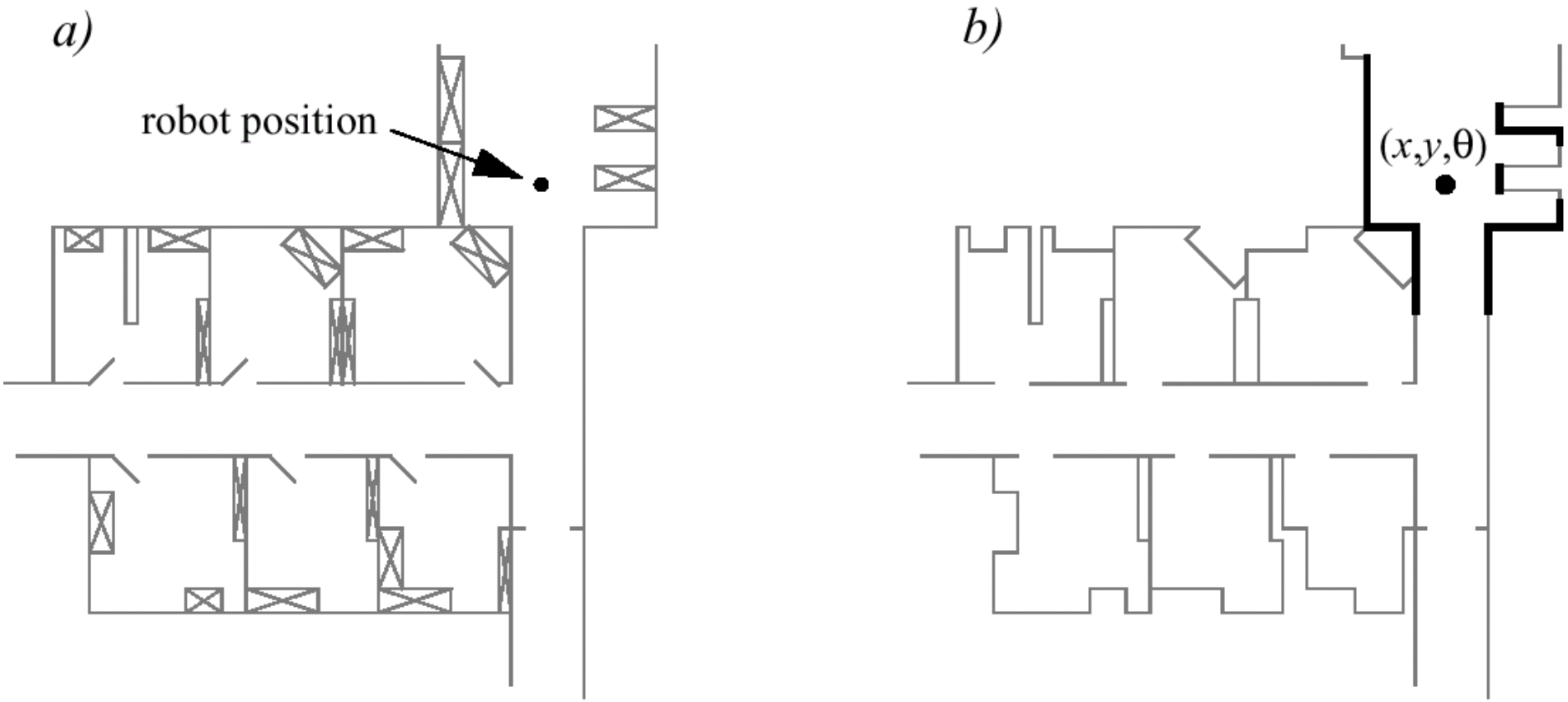
Single Hypothesis Belief

- Can be based on a real map; geometric map; discrete (tessellated) map or topological map.
- In a discrete map, resolution equal to cell size.
- In a topological map, localisation identifies a single topological node.
- Main advantage of a single hypothesis belief is that there is no position ambiguity; robot believes its belief is correct and acts on this belief.
- Updates must also result in a single unique position.
- It is challenging (if not impossible) to generate a single hypothesis when motion takes place, making this approach difficult.

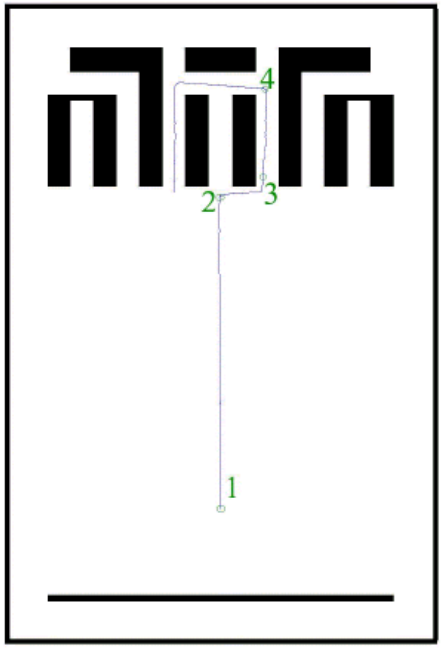
Single Hypothesis Belief



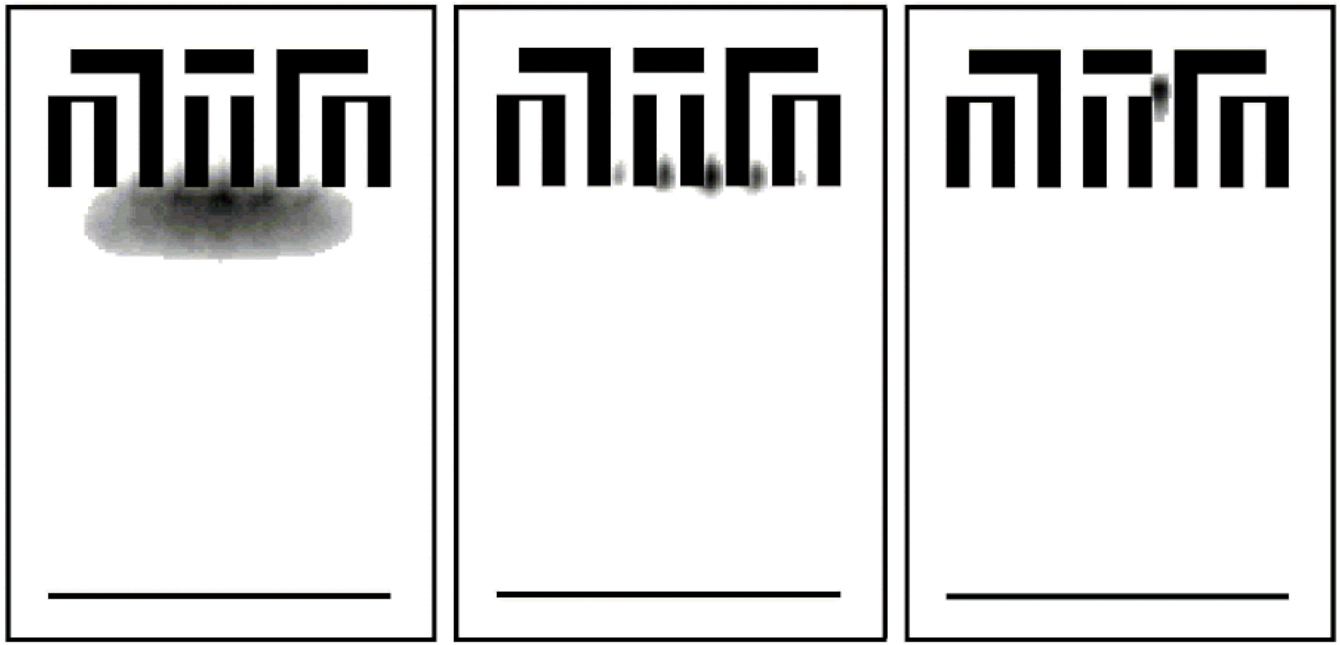
Single Hypothesis Belief



Multi Hypothesis Belief



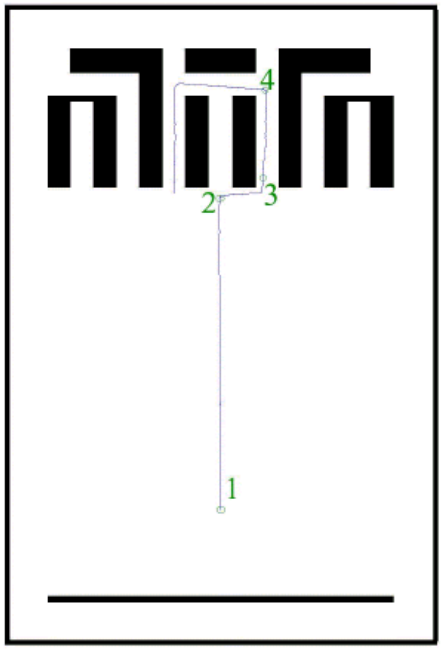
Path of the robot



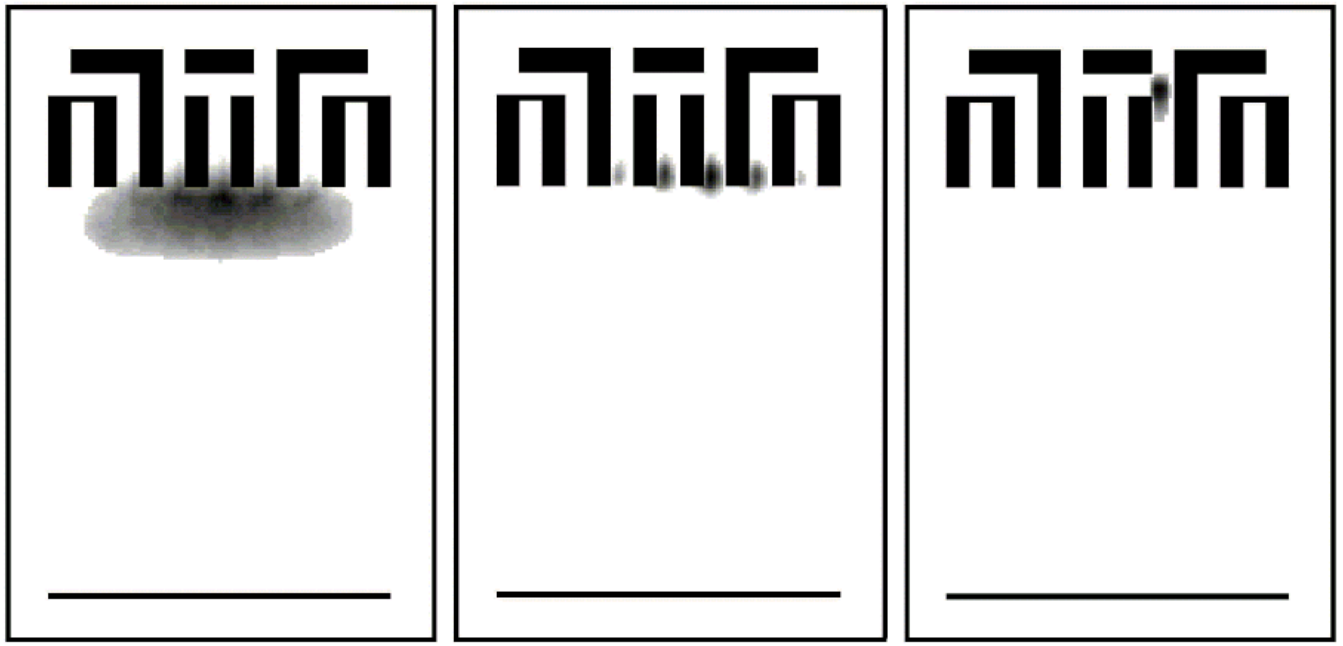
Belief states at positions 2, 3 and 4

- Robot tracks a potentially infinite set of positions.
- Advantages:
  - Allows for explicit modelling of uncertainty regarding position, in the presence of partial sensor information.
  - Also allows robot to explicitly model its uncertainty in position.
  - Robot can then choose a path to minimise future position uncertainty (e,g. by driving past landmarks).

Multi Hypothesis Belief



Path of the robot



Belief states at positions 2, 3 and 4

- Robot tracks a potentially infinite set of positions.
- Disadvantages:
  - Multiple decisions possible for the robot; which action should it perform?
  - Computationally very expensive (N possible positions in the world means belief state is of size 2^N).

Probabilistic Map Based Localisation — Overview

- As a robot moves, odometry allows it to track its location.
- But uncertainty increases due to errors.
- observations are made to help update its position.
- These observations are fused with the odometric observations to get the best possible estimate of position.

1D example

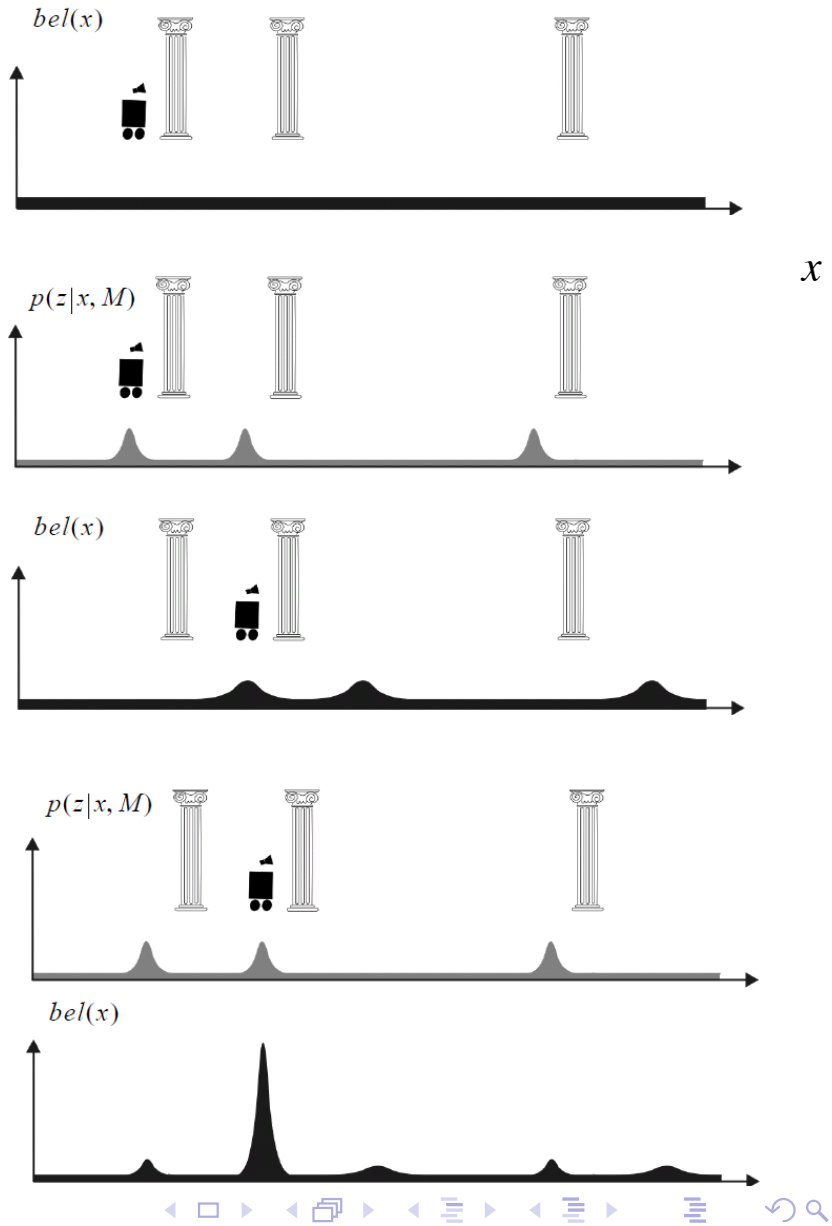
The robot is placed somewhere in the environment but is not told its location

The robot queries its sensors and finds it is next to a pillar

The robot moves one meter forward.
To account for inherent noise in robot motion the new belief is smoother

The robot queries its sensors and again it finds itself next to a pillar

Finally, it updates its belief by combining this information with its previous belief



Probabilistic Robotics

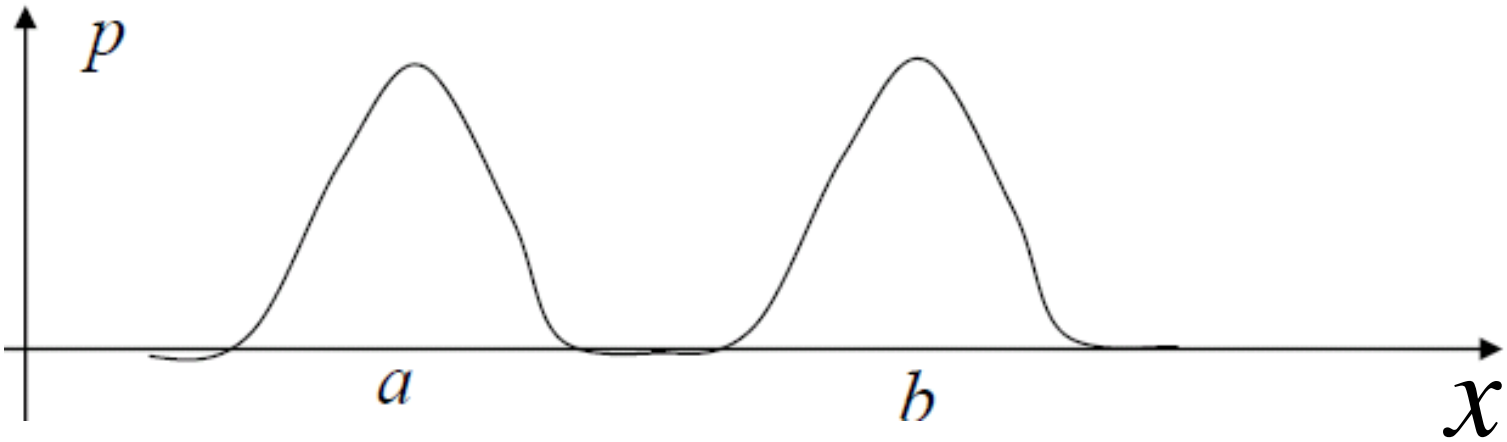
- Due to error, we can only compute the probability that a robot is in some given configuration.
- Probabilistic robotics utilises probability theory to represent the robot's configuration as a probability distribution over all possible poses.
- Such a probability distribution is referred to as a belief.

Sample Distributions

- No information about robot configuration — all states are equally likely.
- Note that the probability distribution must always satisfy

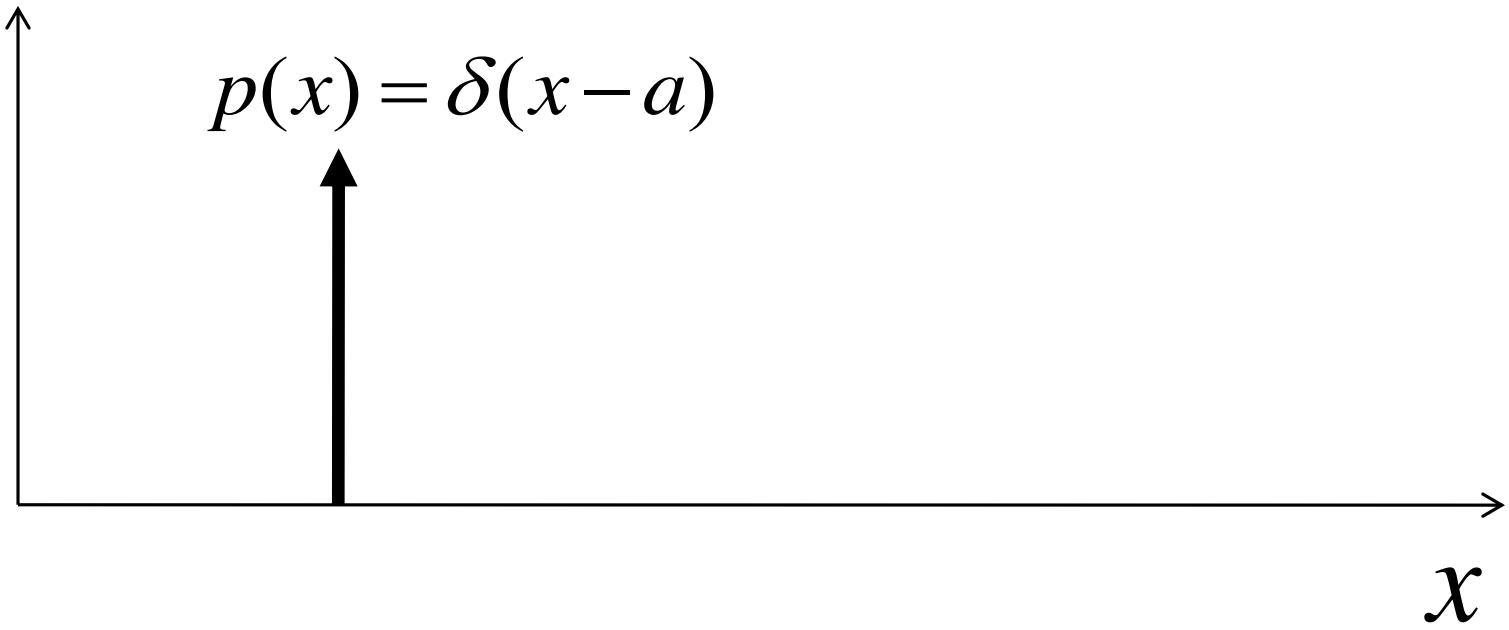
$$\int_{-\infty}^{\infty} p(x)dx = 1$$

Sample Distributions



- A multi-modal distribution identifies likely regions where the robot can be.

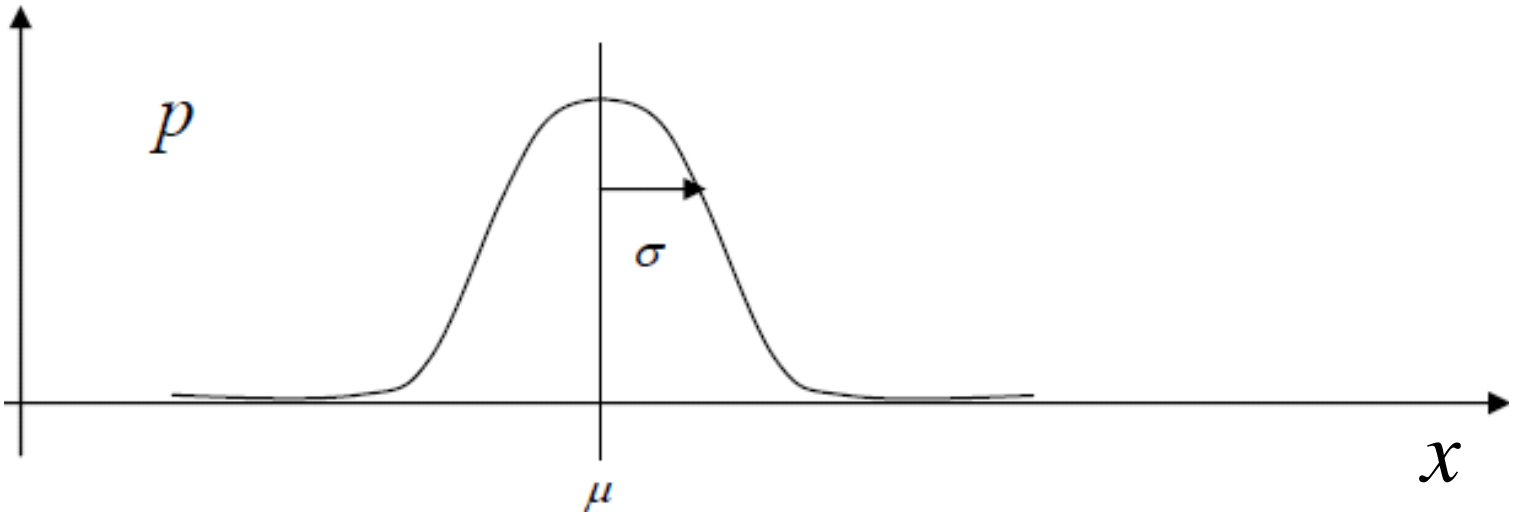
Sample Distributions



- A Dirac distribution identifies where a robot is with full certainty (i.e. $p(a) = 1$) for location a).
- Normally represented with an arrow.
- Dirac function is as follows:

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

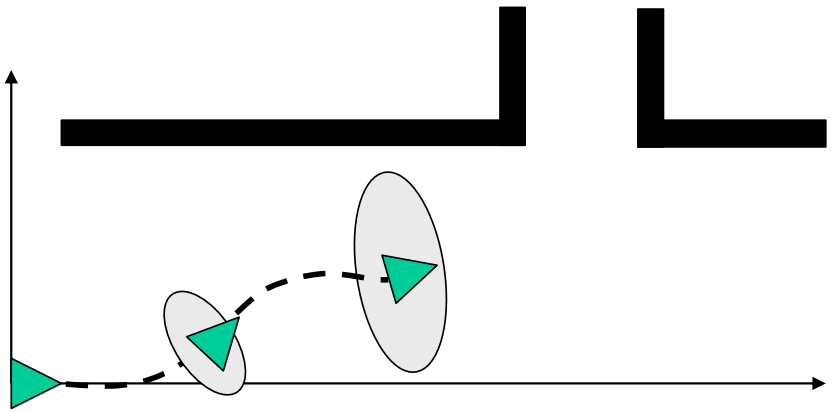
Sample Distributions



- Gaussian distribution is characterised by a mean μ and standard deviation σ
- Abbreviated $N(\mu, \sigma)$

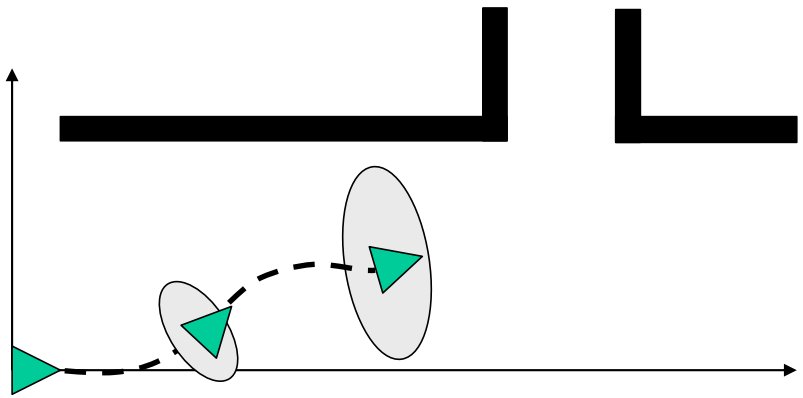
The Problem

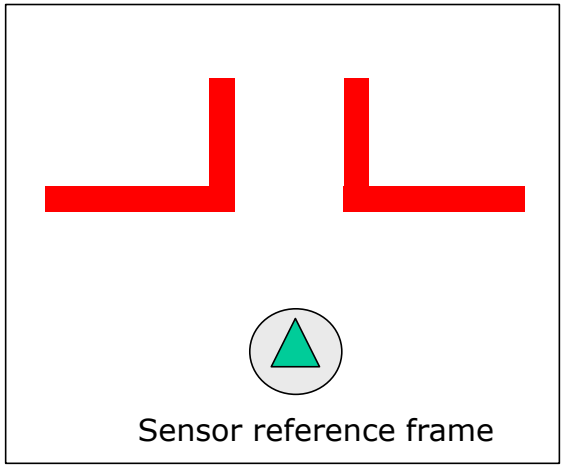
- Assume a robot moves in a known environment, tracking its motion via odometry.
- As more movement takes place, uncertainty increases.



The Problem

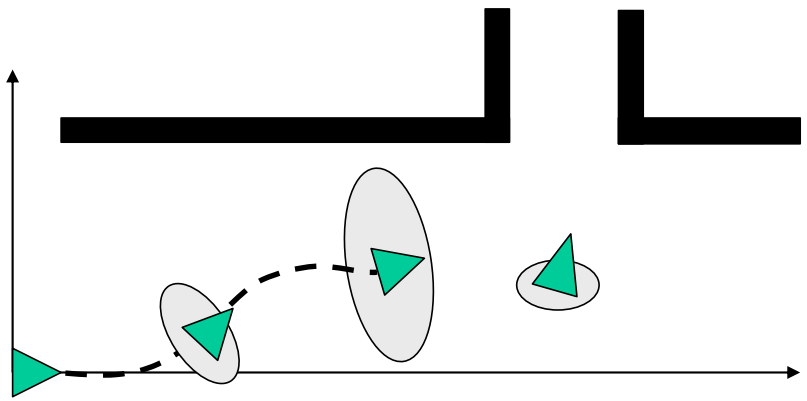
- Assume a robot moves in a known environment, tracking its motion via odometry.
- As more movement takes place, uncertainty increases.
- Utilising its exteroceptive sensors, observations regarding the environment are made.

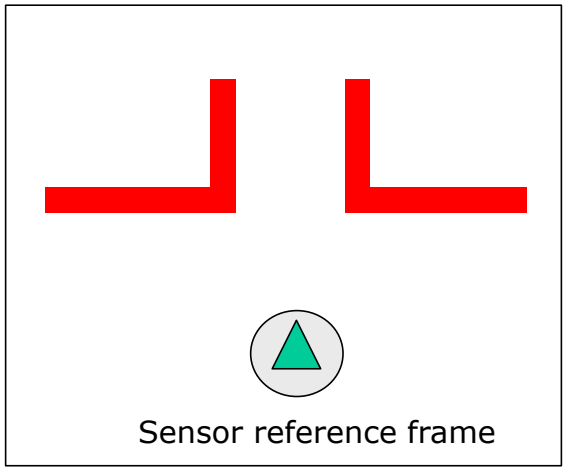




The Problem

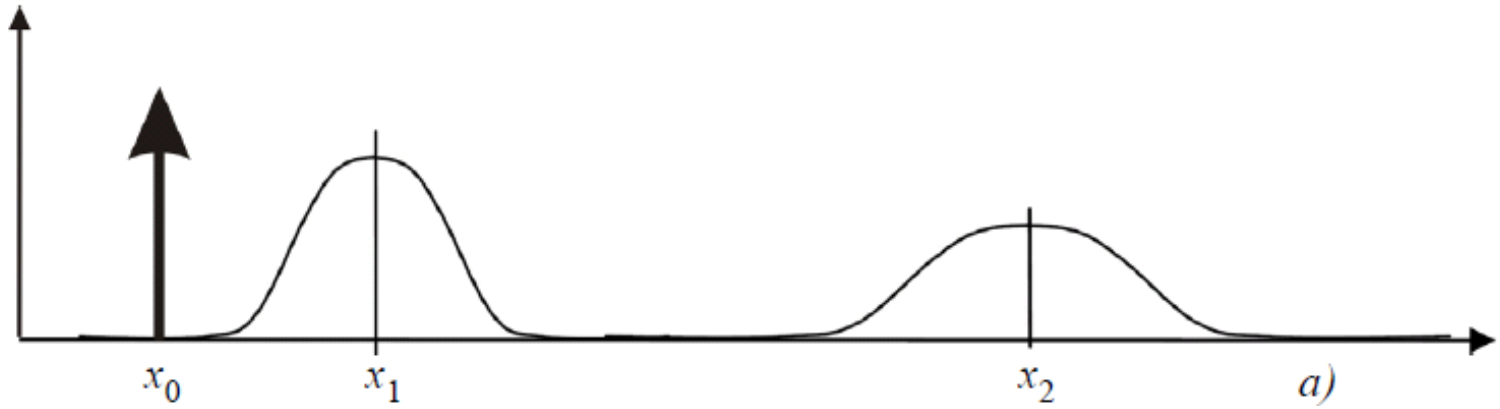
- Assume a robot moves in a known environment, tracking its motion via odometry.
- As more movement takes place, uncertainty increases.
- Utilising its exteroceptive sensors, observations regarding the environment are made.
- These observations shrink the uncertainty.



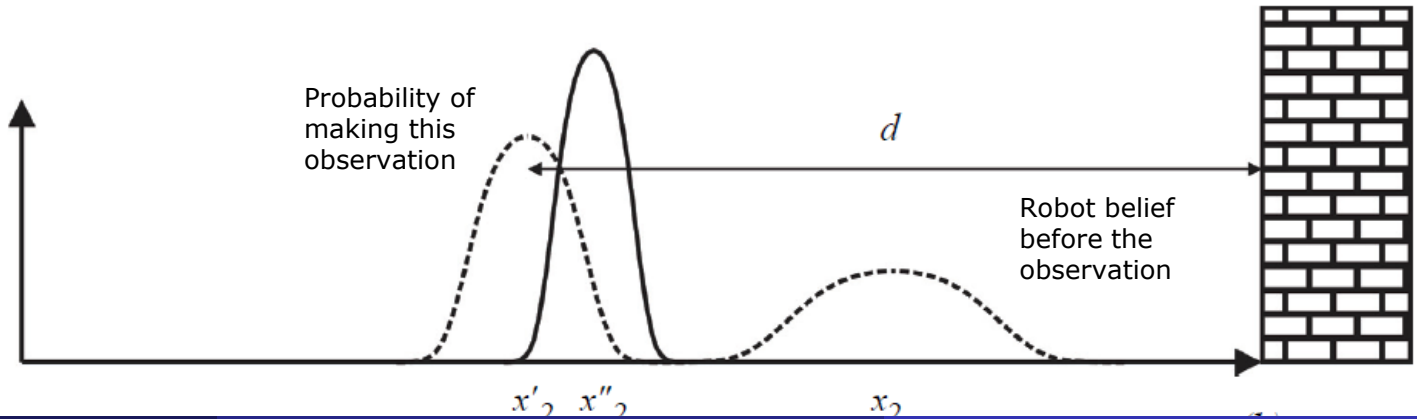


Updates

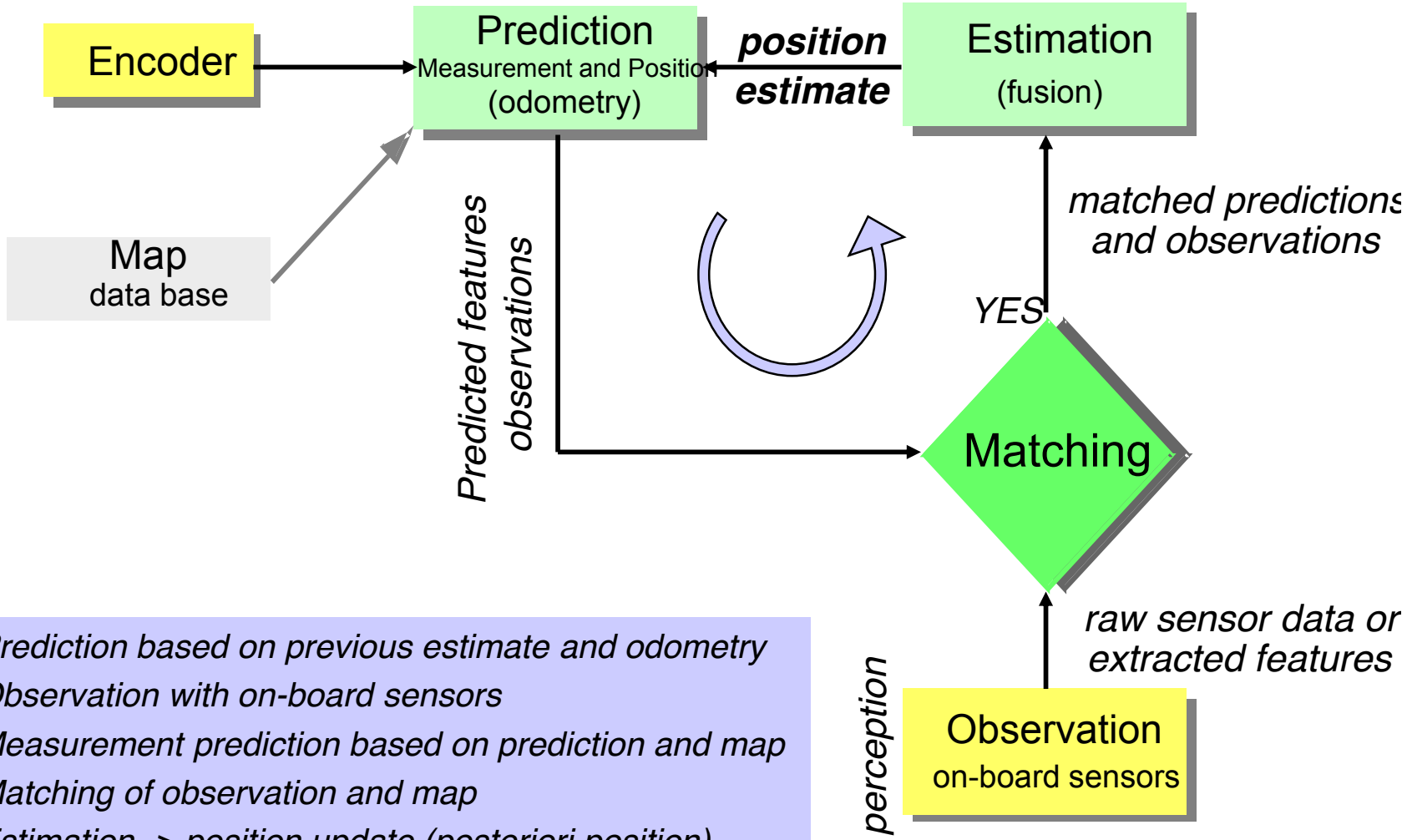
- We can identify two distinct steps in localisation:
  - Action (or prediction) update — the robot moves and estimates position using proprioception; uncertainty grows.



  - Perception (or measurement) update — observations are made using exteroceptive sensors. Uncertainty shrinks are prior beliefs are combined with probability of actually making the observation.



Localisation



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)*

Probabilistic Robot Localisation: Inputs

- The probabilistic approach computes the probability distribution of the robot pose during each action and perception step.
- Inputs:
  - Initial probability distribution $p(x)_{t=0}$
  - Statistical error model of proprioceptive sensors

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

  - Statistical error model of the exteroceptive sensors
  - Map of the environment

Performing Updates

- Action update uses theorem of total probability

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

- Perception update uses Bayes rule

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

Action Update

- $bel(x_t)$ is computed based on belief in the previous position $bel(x_{t-1})$ and the proprioceptive input u_t.
- Utilises the Markov assumption: current robot position depends only on the previous position (and odometric input).

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

Perception Update

- The robot corrects its previous position by combining it with information from exteroceptive sensors

$$p(x_t|z_t) = \frac{p(z_t|x_t)p(x_t)}{p(z_t)}$$

- Since all possible sensor inputs must sum to 1, we can utilise a normalisation factor

$$bel(x_t) = \eta p(z_t|x_t)bel(x_t)$$

Algorithm

1: **for all** x_t **do**
2: $bel(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$ ▷ Prediction Update
3: $bel(x_t) = \eta p(z_t|x_t) bel(x_t)$ ▷ Measurement Update
4: **end for**
5: **return** $bel(x_t)$

- This algorithm is also called the Bayes filter.

Example

Initial probability distribution

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

$$p(z_t \mid x_t)$$

Perception update

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

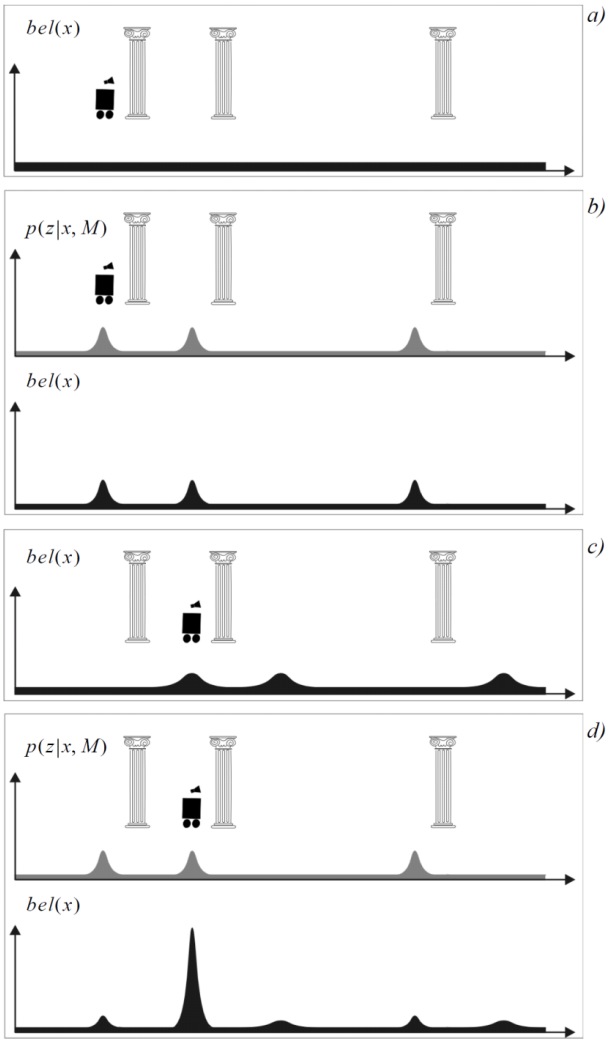
Action update

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

$$p(z_t \mid x_t)$$

Perception update

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

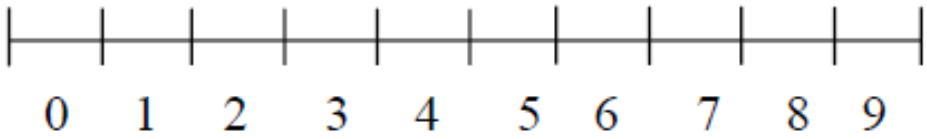


Approach

- Markov approach
  - The configuration space is divided into discrete cells with each cell containing the probability of the robot being in the cell.
  - During action and perception, all cells are updated (the sensor model distribution is also discrete).
- Kalman approach
  - We assume that both robot configuration and sensor distributions are continuous and Gaussian.
  - Only μ and Σ^2 need to be updated — computationally efficient.

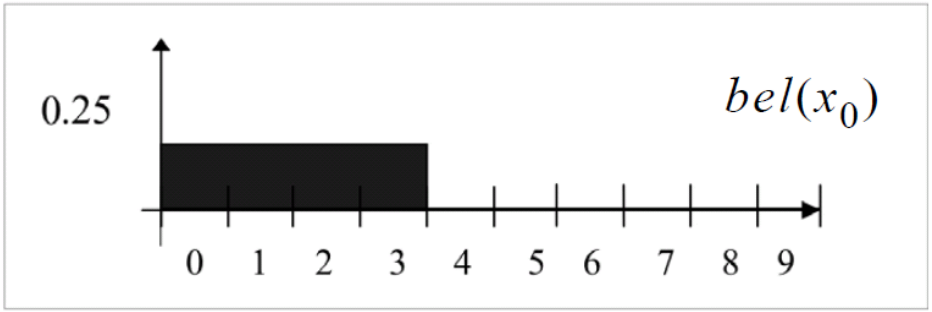
Markov Localisation

- We utilise a grid space to represent robot configuration (one dimension for each of x, y, θ).
- Approach best illustrated in 1 dimension — consider a robot able to move only in the x direction.
- We discretise the space into 10 cells.

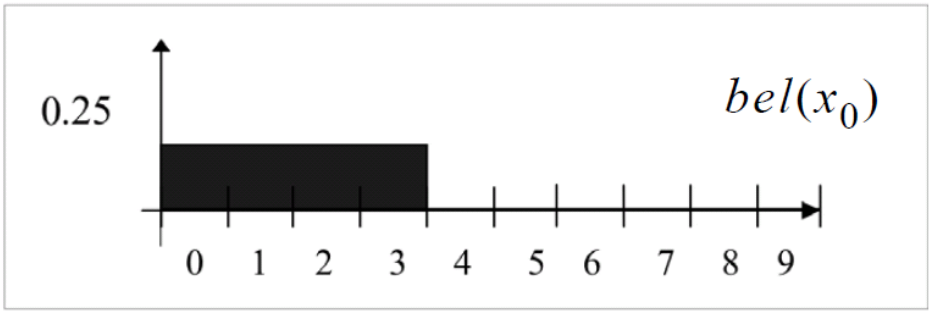


Markov Localisation

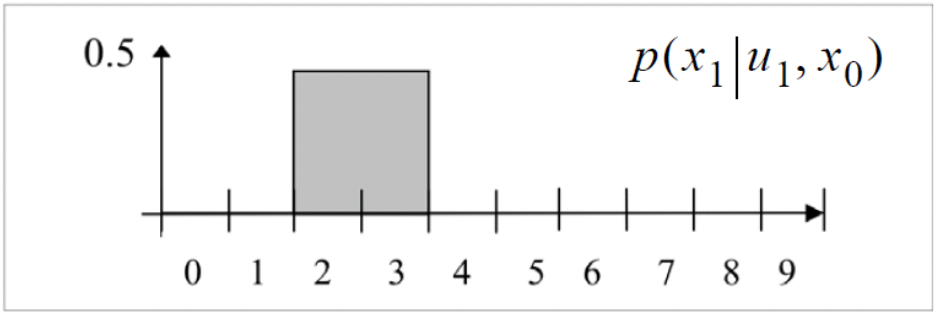
- Assume that the robot initially believes it is either in cell 0,1,2 or 3 (with equal likelihood).



Action



- Now assume the robot's forward movement causes it to either move 2 or 3 cells forward (with equal likelihood)

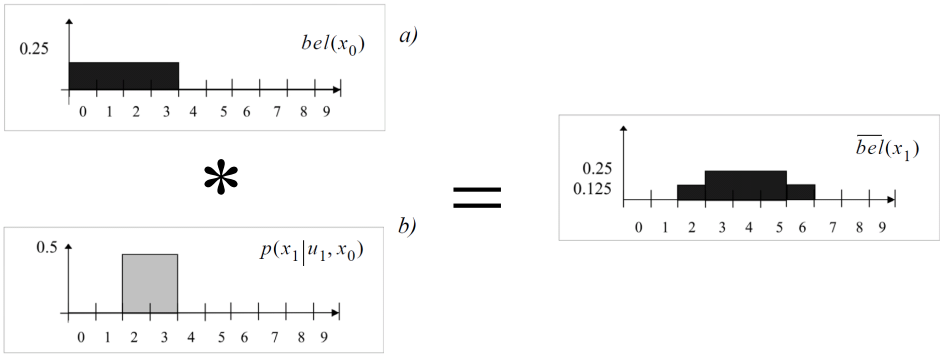


- Where is the robot after moving?

Action Update

- We cross-correlate the two distributions to obtain the solution

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



Action Update

- Essentially, we are using the theorem of total probability

$$p(x_1 = 2) = p(x_0 = 0)p(u_1 = 2) = 0.125\,,$$

$$p(x_1 = 3) = p(x_0 = 0)p(u_1 = 3) + p(x_0 = 1)p(u_1 = 2) = 0.25$$

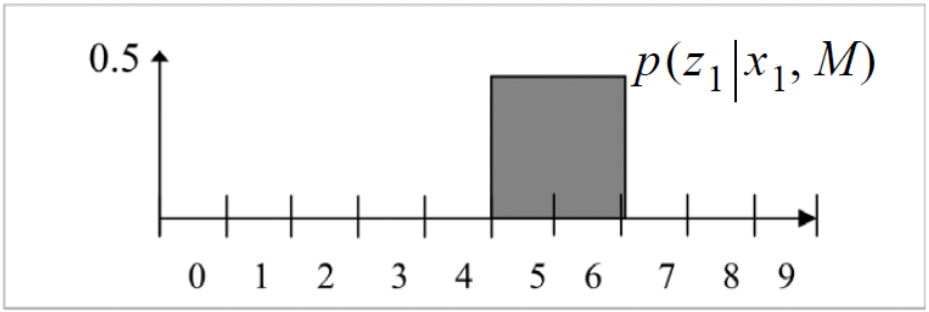
$$p(x_1 = 4) = p(x_0 = 1)p(u_1 = 3) + p(x_0 = 2)p(u_1 = 2) = 0.25$$

$$p(x_1 = 5) = p(x_0 = 2)p(u_1 = 3) + p(x_0 = 3)p(u_1 = 2) = 0.25$$

$$p(x_1 = 6) = p(x_0 = 3)p(u_1 = 3) = 0.125$$

Perception Update

- The robot uses its sensors to measure the distance from the origin.



- In other words, the robot is either in cell 5 or 6.
- What is the final robot belief after this measurement?

Perception Update

$$bel(x_t) = \eta p(z_t|x_t) bel(x_t)$$

$$bel(x_1 = 5) = \eta p(z_1 = 5|x_1 = 5) p(x_1 = 5) = \eta * 0.5 * 0.25 = 0.666$$
$$bel(x_1 = 6) = \eta p(z_1 = 6|x_1 = 6) p(x_1 = 6) = \eta * 0.5 * 0.125 = 0.333$$
$$\eta = \frac{1}{(0.5)(0.25)+(0.5)(0.125)} = \frac{1}{0.1875} \approx 5.33$$

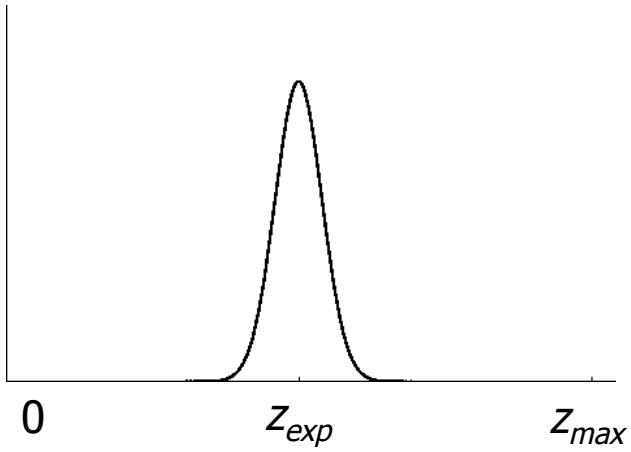
Real World Perception Updates

- How do we compute $p(z_t|x_t)$?
  - Beam sensor model — consider how the beam moves through the map
  - Likelihood field model — consider only the sensor endpoints interacting with a likelihood field
- These are heuristics that have been found to work well in real life.
- Other models exist, often more computationally expensive but may be more accurate.

Beam Sensor Model

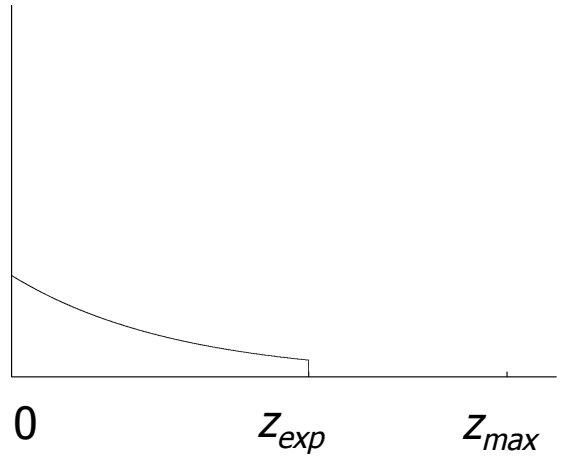
- Considers 4 possible beam types. Here, η is a normalising constant used to make sure all probabilities sum to 1.

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

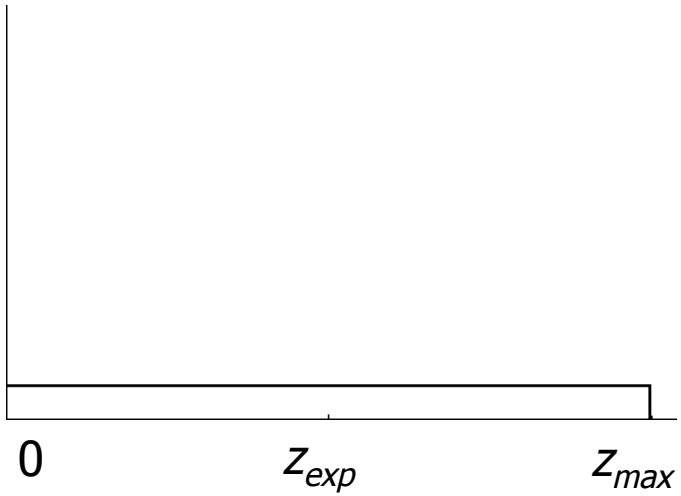


$$P_{\text{unexp}}(z \mid x,m) = \begin{cases} \eta\, \lambda\, \mathrm{e}^{-\lambda z} & z < z_{\exp} \\ 0 & otherwise \end{cases}$$

Beam Sensor Model

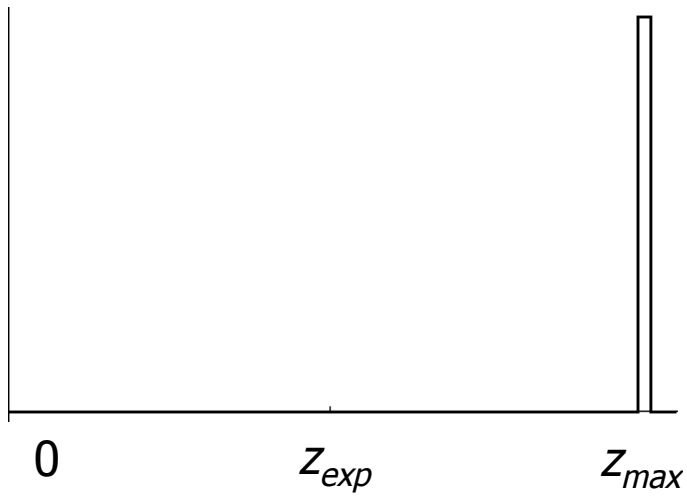
- Considers 4 possible beam types. Here, η is a normalising constant used to make sure all probabilities sum to 1.

Random measurement



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

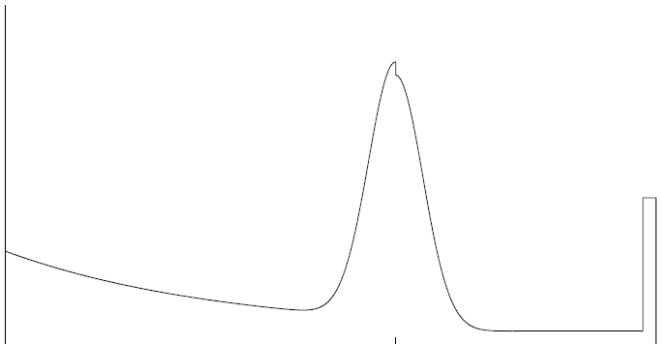
Max range



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

Beam Sensor Model

- Considers 4 possible beam types. Here, η is a normalising constant used to make sure all probabilities sum to 1.

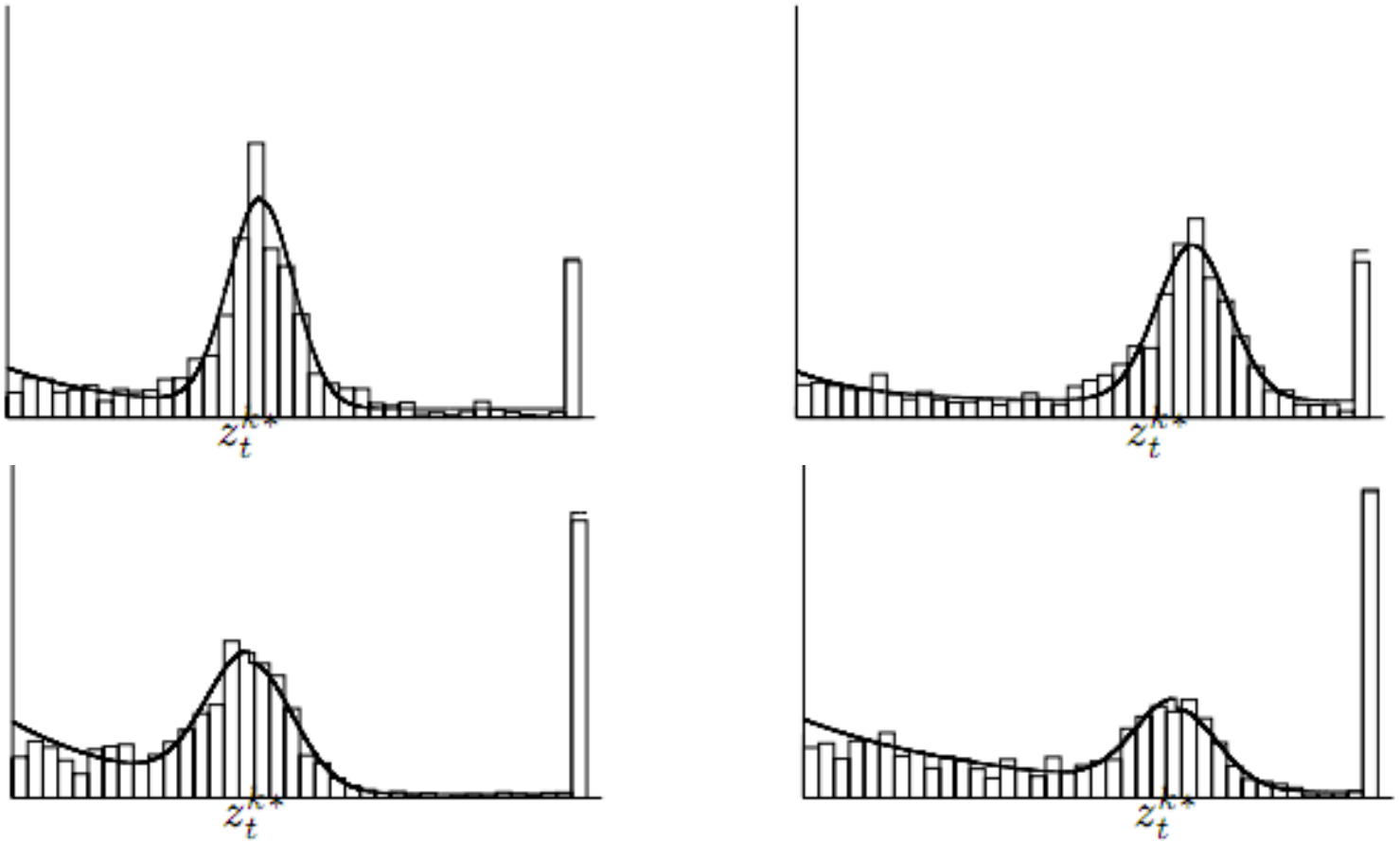


$$P(z \mid x,m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\text{max}} \\ \alpha_{\text{rand}} \end{pmatrix}^{T} \cdot \begin{pmatrix} P_{\text{hit}}(z \mid x,m) \\ P_{\text{unexp}}(z \mid x,m) \\ P_{\text{max}}(z \mid x,m) \\ P_{\text{rand}}(z \mid x,m) \end{pmatrix}$$

Beam Sensor Model

- Parameters are fitted based on actual sensor measurements.

Laser



300cm

400cm

Sonar

Beam Sensor Model

- Assumes independence between beams.
- Models physical causes for measurements.
  - Mixture of densities for causes.
  - Independence between causes.
- Implementation
  - Learn parameters based on real data.
  - Ideally different models should be learned for different angles at which sensor beam hits obstacle (but not often done).
  - Expected distances are computed using ray tracing.

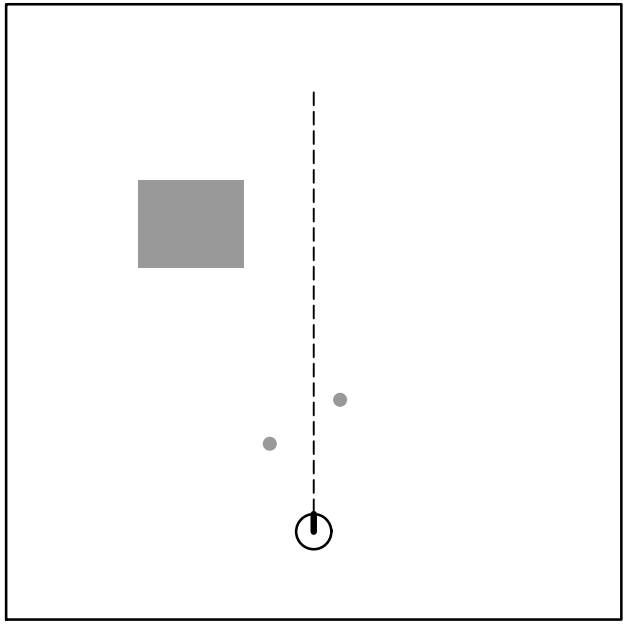
Likelihood field Model

- Also known an beam endpoint model or scan-based model.
- Ad hoc algorithm — doesn't give a physics based generative model of probability.
- Works well in practice.
- Basic idea: rather than following along the beam, check what's going on at the endpoint.

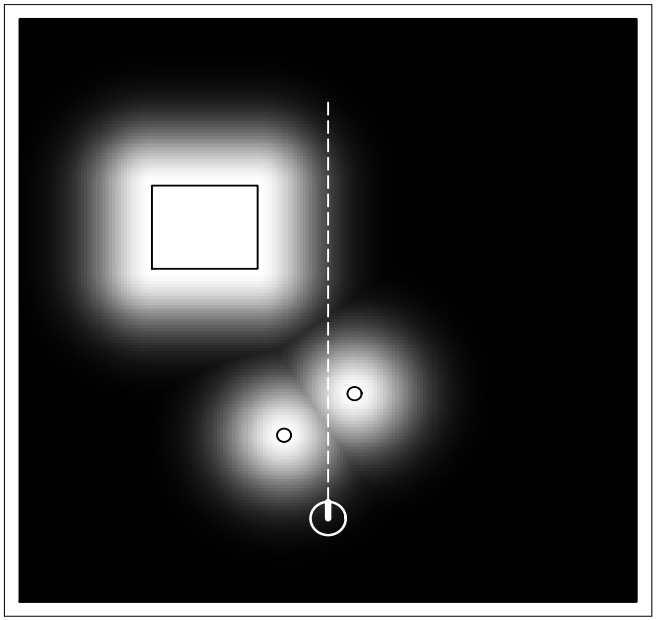
$$p(z|x_t) = N(d^2, \sigma) + p_{random}/z_{max} + p_{max}/z_{small}$$

- Where d is the distance from the point sensor reported to the nearest obstacle.
- We can precompute a likelihood field using $N(d^2, \sigma)$ for the map.

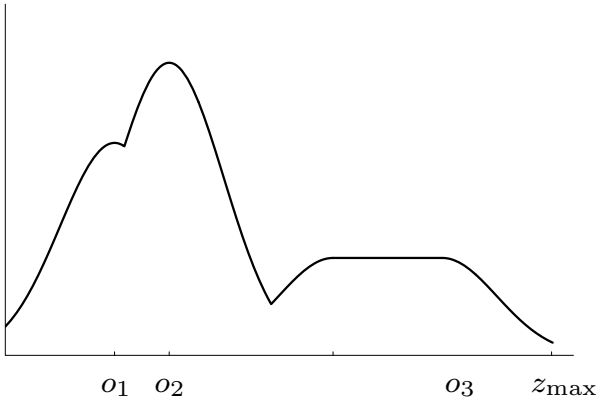
Likelihood field Model

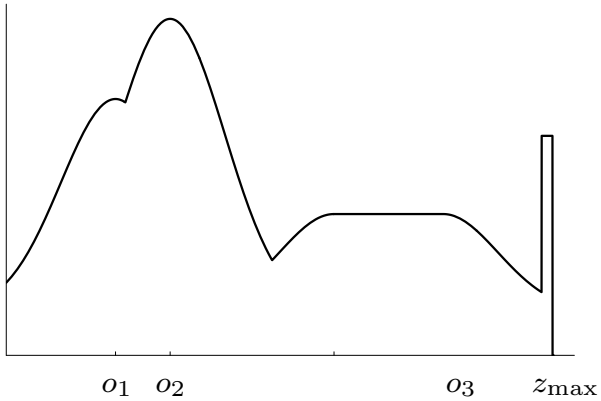


(a) $p_{\text{hit}}(z_t^k \mid x_t, m)$



(b) $p(z_t^k \mid x_t, m)$





Likelihood field Model

- No explicit modelling of people or other dynamic factors that can cause short readings.
- No modelling of beam — treats sensor as if it can see through walls.

Gaussian Update — Weaknesses

- In the general case, the grid is a three dimensional structure capturing robot pose and the probability that the robot is in the cell.
- The size of grid must be carefully selected — for a 30×30m environment with a 0.1m and 1 degree cell size, there are ≈ 32 million cells.
- Updates become computationally expensive.
- Memory requirements increase.

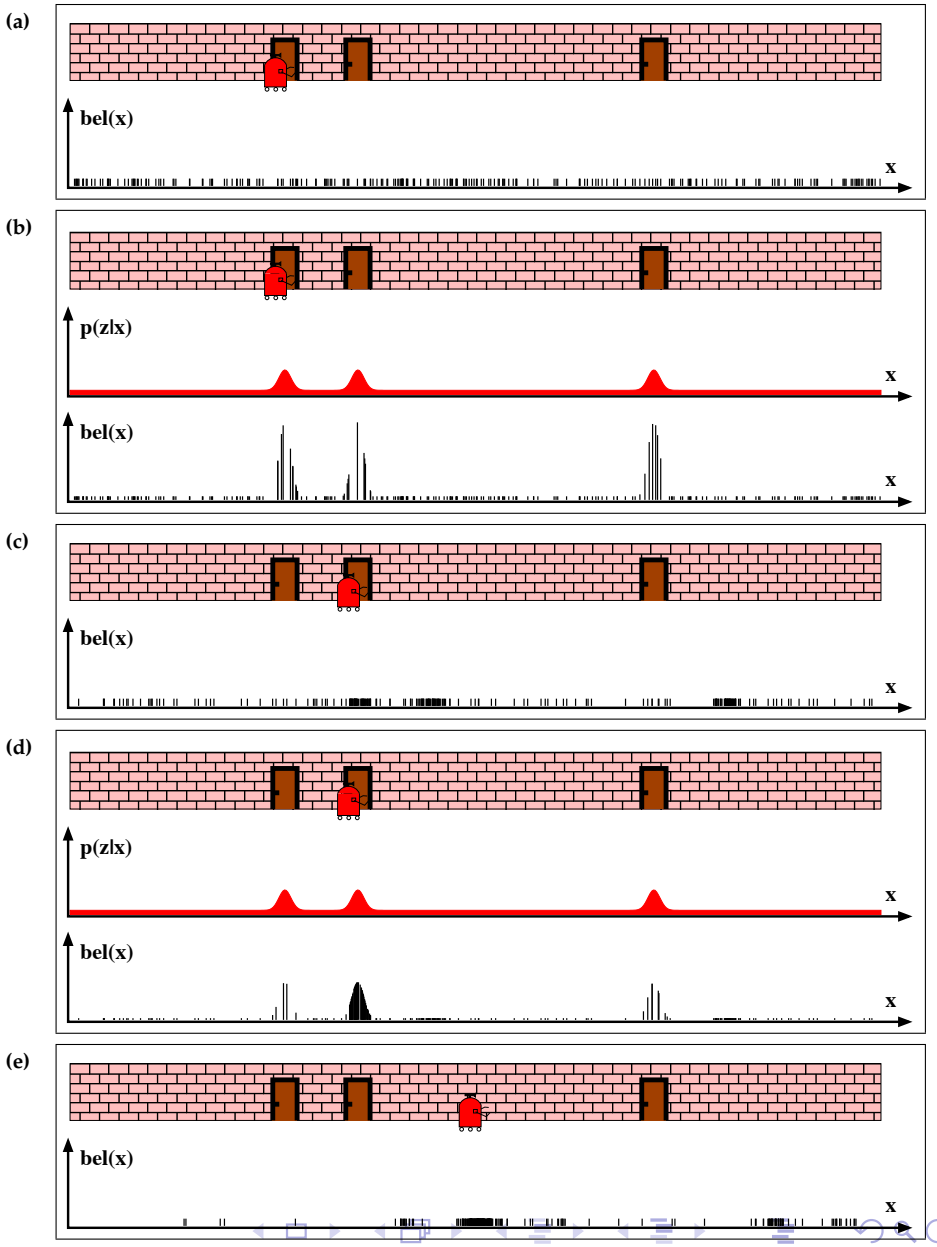
Particle Filters/Monte Carlo Localisation

- Instead of representing every possible robot state through a complete and correct belief state, we can construct approximate belief states.
- This is done by using a subset of the complete set of possible locations that should be considered.
- The belief state is encoded through a set of m particles distributed within the state space.
- Particles are moved according to the action update probability.
- A weight is associated with each particle according to its likelihood with regards to the perception model.
- A new set of particles is generated by sampling from the weighted particle set.

Particle Filters

Require: $X_{t-1} = \{x_{t-1}^1, \ldots, x_{t-1}^M\}$

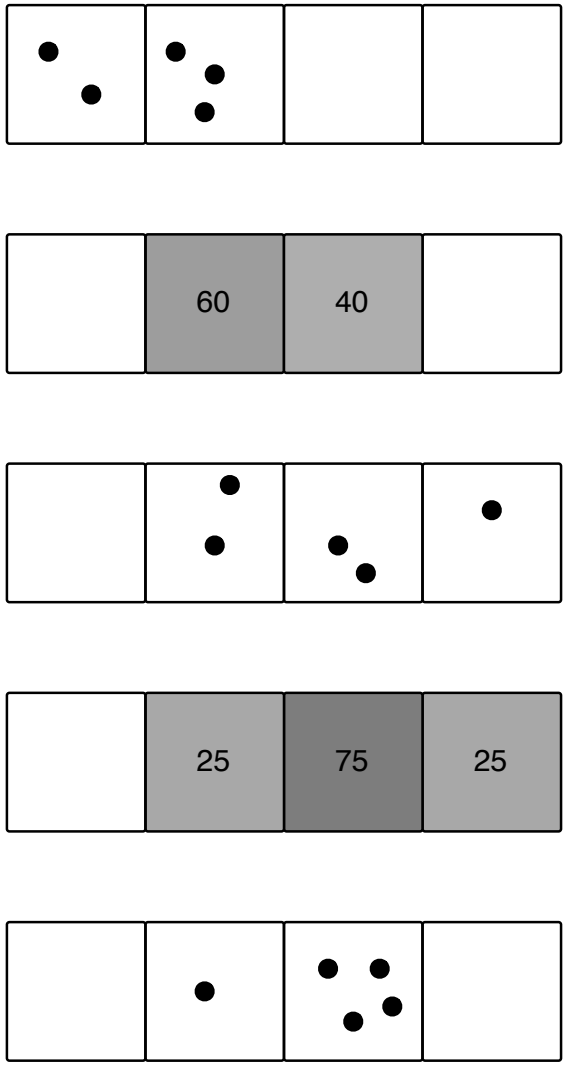
1: $\bar{X}_t = X_t = \emptyset$
2: **for all** m=1 to M **do**
3: x_t^m =update x_{t-1}^m based on motion u_t
4: w_t^m =likelihood of obtaining sensor reading z_t given position x_t^m
5: $\bar{X}_t = \bar{X}_t + \langle x_t^m, w_t^m \rangle$
6: **end for**
7: **for all** m=1 to M **do**
8: select i with probability proportional to w_t^i
9: add x_t^i to X_t
10: **end for**
11: **return** X_t



Particle Filters — Example

Require: $X_{t-1} = \{x_{t-1}^1, \ldots, x_{t-1}^M\}$

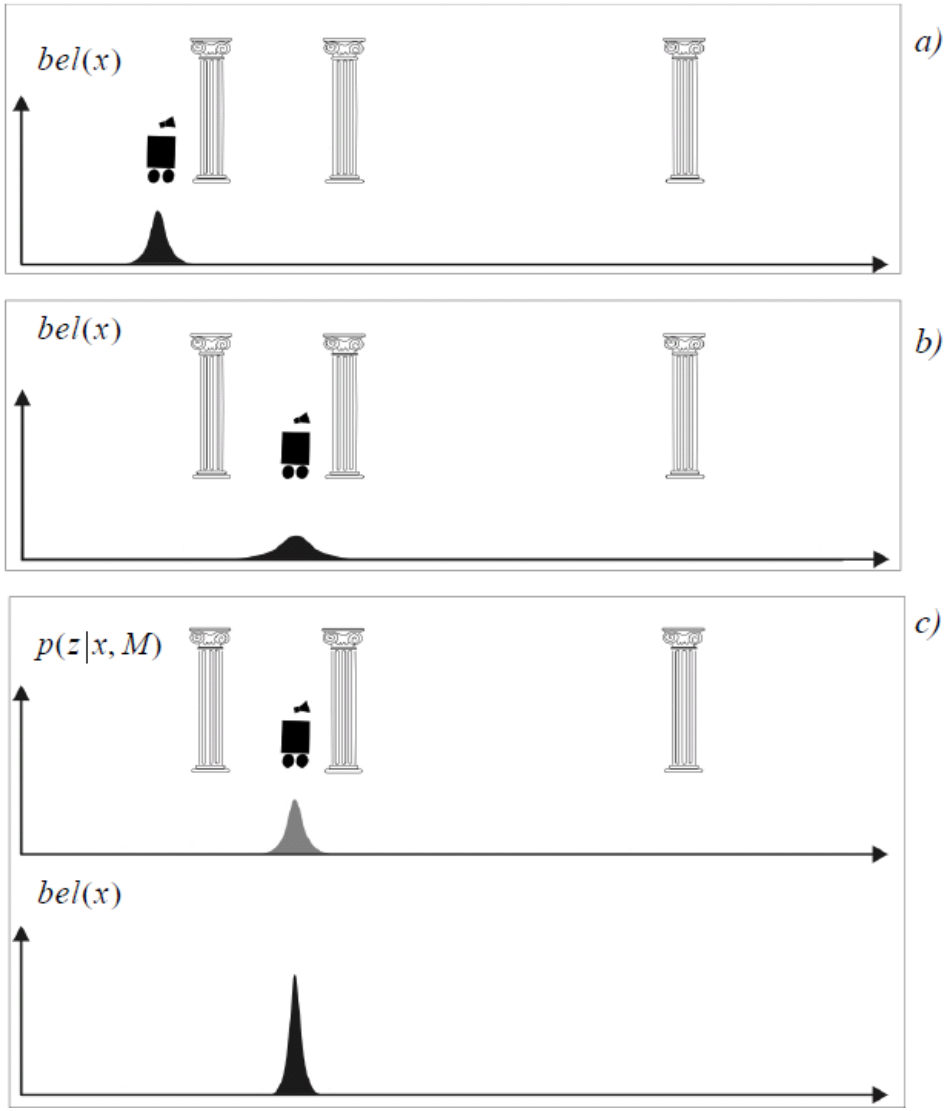
1: $\bar{X}_t = X_t = \emptyset$
2: **for all** m=1 to M **do**
3: x_t^m =update x_{t-1}^m based on motion u_t
4: w_t^m =likelihood of obtaining sensor reading z_t given position x_t^m
5: $\bar{X}_t = \bar{X}_t + \langle x_t^m, w_t^m \rangle$
6: **end for**
7: **for all** m=1 to M **do**
8: select i with probability proportional to w_t^i
9: add x_t^i to X_t
10: **end for**
11: **return** X_t



Particle Filter Notes

- Particle weights are often the cube of $p(z|t)$, summed over all beams.
- Much fewer particles are needed than grid points.
- Multiple particles can end up occupying the same grid point.
- If motion or sensors are totally accurate, we may be unable to find a particle to sample — noise is needed.
- Particle distributions end up mirroring full distribution.
- Additional particles drawn from a uniform distribution are often added to enable recovery from errors.
- `http://www.youtube.com/watch?v=8P5DhFRYxCY`

Kalman Filters



Linear Gaussian Functions

- Kalman Filters make use of the special properties of linear Gaussians.
- Given two Gaussian random variables

$$x_1 = N(\mu_1, \sigma_1^2) \qquad x_2 = N(\mu_2, \sigma_2^2)$$

- If $y = f(x_1, x_2) = Ax_1 + Bx_2$ then

$$\mu_y = A\mu_1 + B\mu_2$$

$$\sigma_y^2 = A^2\sigma_1^2 + B^2\sigma_2^2 \qquad A\Sigma_1 A^T + B\Sigma_2 B^T$$

Non-linear Gaussians

- If f is non-linear, y will not be Gaussian.
- However, we consider a first-order approximation by linearizing about f

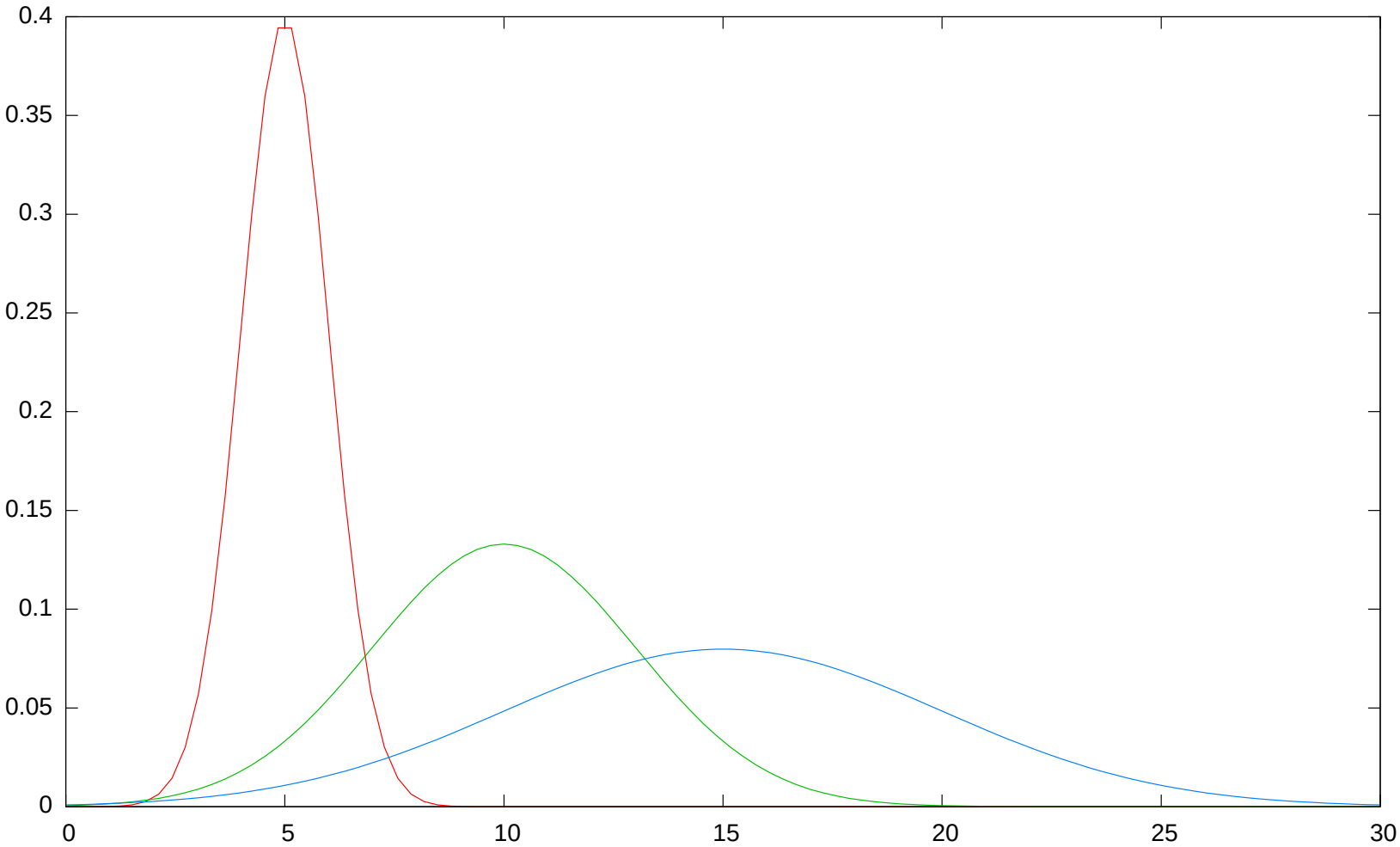
$$y \approx f(\mu_1, \mu_2) + F_{x_1}(x_1 - \mu_1) + F_{x_2}(x_2 - \mu_2)$$

- F_{x_1}, F_{x_2} are the Jacobians of f w.r.t x_1 and x_2

$$\mu_y = f(\mu_1, \mu_2)$$

$$\Sigma_y = F_{x_1}\Sigma_1 F_{x_1}^T + F_{x_2}\Sigma_2 F_{x_2}^T$$

1D case



Variance Increases over time

Prediction

- We have done the motion update.
- For prediction, let q denote robot position, $p_1(q)$ the motion update belief and $p_2(q)$ the exteroceptive belief. Assuming these are normally distributed,

$$p_1(q) = N(\hat{q}_1, \sigma_1^2) \qquad p_2(q) = N(\hat{q}_2, \sigma_2^2)$$

- Bayes rule tells us that the final distribution $p(q)$ after motion and measurement will be proportional to $p_1(q)p_2(q)$.
- Multiplying out we get

$$\frac{1}{\sigma_1 \sigma_2 2\pi} e^{-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2} - \frac{(q-\hat{q}_2)^2}{2\sigma_2^2}}$$

Prediction

$$\frac{1}{\sigma_1 \sigma_2 2\pi} e^{-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2} - \frac{(q-\hat{q}_2)^2}{2\sigma_2^2}}$$

- Can we rewrite this in the form of another Gaussian:

$$\Omega e^{-\frac{(q-\hat{q})^2}{2\sigma^2}}$$

Prediction

$$e^{-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2}-\frac{(q-\hat{q}_2)^2}{2\sigma_2^2}}$$

$$= e^{-0.5\frac{q^2(\sigma_1^2+\sigma_2^2)-2q(\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2)+(\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2)}{\sigma_1^2\sigma_2^2}}$$

$$= e^{-0.5\frac{q^2-2q\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}+\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}}$$

$$= e^{-0.5\frac{q^2-2q\alpha+\alpha}{\beta}}$$

Completing the Square

- A quadratic equation of the form $ax^2 + bx + c$ can be rewritten as $(x - h)^2 + k$
- Where $h = -b/2$ and $k = c - b^2/4$

$$= e^{-0.5\frac{(q-\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2})^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}} e^{-0.5\frac{\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2}-(\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2})^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}}$$

- The second term is a constant as it depends only on $\hat{q}_1$ and $\hat{q}_2$

Prediction

- Therefore we have

$$\Omega e^{-0.5\frac{(q-\frac{\hat{q_1}\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2})^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}} = \Omega e^{\frac{(q-\hat{q})^2}{2\sigma^2}}$$

- Where

$$\hat{q} = \frac{\hat{q_1}\sigma_2^2 + \hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2} \qquad \sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}$$

General Case

- In the general case we have

$$\hat{q} = q_1 + P(P+R)^{-1}(q_2 - q_1) \qquad \hat{P} = P - P(P+R)^{-1}P$$

- Where P and R are the covariances of q_1 and q_2
- and $\hat{q}$ is the final mean and $\hat{P}$ is the final covariance.
- Alternatively

$$\hat{q} = q_1 + K(q_2 - q_1) \qquad \hat{P} = P - K\Sigma_{IN}k^T$$

- $K = P(P+R)^{-1}$ is the Kalman gain, $q_2 - q_1$ is the innovation and $\Sigma_{IN} = P + R$ is the innovation covariance.

Kalman vs Markov

- In Markov localisation all sensor measurements are used to update the robot position likelihood in each individual belief state.
- In Kalman filtering only one belief state is updated.
- For Kalman filtering, the prediction update is an application of the Gaussian error motion model with respect to measured travel.
- Measurement update consist of 4 steps
  1. Observation: collect sensor data and extract features.
  2. Measurement prediction: based on location, generate features that are expected to be observed.
  3. Matching: computes the best match between extracted features and expected features.
  4. Estimation: fuses all the information provided by the matching to update the belief.

Prediction Update

- Mean position at time step t is based on the old location and movement due to the control input u_t

$$\hat{x}_t = f(x_{t-1}, u_t)$$

- We can compute the variance associated with the prediction using the total probability theorem applied to Gaussians

$$\hat{P}_t = F_x P_{t-1} F_x^T + F_u Q_t F_u^T$$

- We are only updating two values — mean and covariance.

Measurement Updates - Observation

- A set of sensor measurements z_t is obtained for the robot at time t
- z_t consists of n individual observations obtained from a sensor — z_t^i for $i = 1 \ldots n$.
- Each observation is typically specified in the sensor frame (i.e. the robot's frame of reference).
- We therefore transform our robot's position to the sensor reference frame via a function h.

Measurement Updates - Measurement Prediction

- Given a map M and the robot's predicted position $\hat{x}_t$ we generated predictions for observations $\hat{z}_t^j$.
- These identify what the robot expects to see if it was at position $\hat{x}_t$.
- If feature j is at m^j on the map then

$$\hat{z}_t^j = h^j(\hat{x}_t, m^j)$$

Measurement Updates - Matching

- We now have a set of actual observations positioned in the sensor space.
- And a set of predicted features, positioned in the sensor space.
- We seek to identify observations that match predicted features "well enough" to be used during the estimation step.
- I.e. "This observation is the robot's measurement of this predicted feature on the map".
- Aim is to assign from an observation z_t^i to a predicted observation $\hat{z}_t^j$
- For each prediction for which a corresponding observation is found, we calculate the difference between the predicted and observed measurements:

$$v_t^{ij} = [z_t^i - \hat{z}_t^j] = [z_t^i - h^j(\hat{x}_t, m^j)]$$

- This is a measure of innovation.

Measurement Updates - Matching

- The covariance of the innovation can be computed as

$$\Sigma_{IN_t}^{ij} = H^j \hat{P}_t {H^j}^T + R_t^i$$

- H^j is the Jacobian of h^j and R_t^i is the covariance of the actual observation z_t^i
- In other words, total covariance is the covariance due to the position plus covariance due to sensor noise.
- To identify correspondences, we pick a threshold g and ensure that

$${v^{ij}}_t^T (\Sigma_{IN_t}^{ij})^{-1} v_t^{ij} \leq g^2$$

- This is the Mahalanobis distance — measuring the distance of the test point from the origin (i.e. identical points) taking into account the shape of the covariance ellipsoid.
- If an observation succeeds once, it is an obvious match. If it falls into multiple potential observations, pick the based one (smallest distance).

Measurement Updates - Estimation

- Given position prediction $\hat{x}_t$ and observations z_t^i
- We stack all observations into a single vector z_t and create a composite innovation v_t.
- Create a stacked Jacobian H from all H^j
- Create a stacked measurement error vector $R_t = diag[R_t^i]$
- We can compute a composite innovation covariance matrix Σ_{IN_t}

$$x_t = \hat{x}_t + K_t v_t$$

$$P_t = \hat{P}_t - K_t \Sigma_{IN_t} K_t^T$$

Where

$$K_t = \hat{P}_t H_t^T \Sigma_{IN_t}^{-1}$$

What have we done?

- The best measure of the robot's position is equal to its best prediction of position before sensing plus a correction in terms of the optimal weighting value K_t times the difference between z_t and the best prediction $\hat{z}_t$ at time t.
- We correct position based on a weighted combination of measurements, based on how far away they are from our expectations.

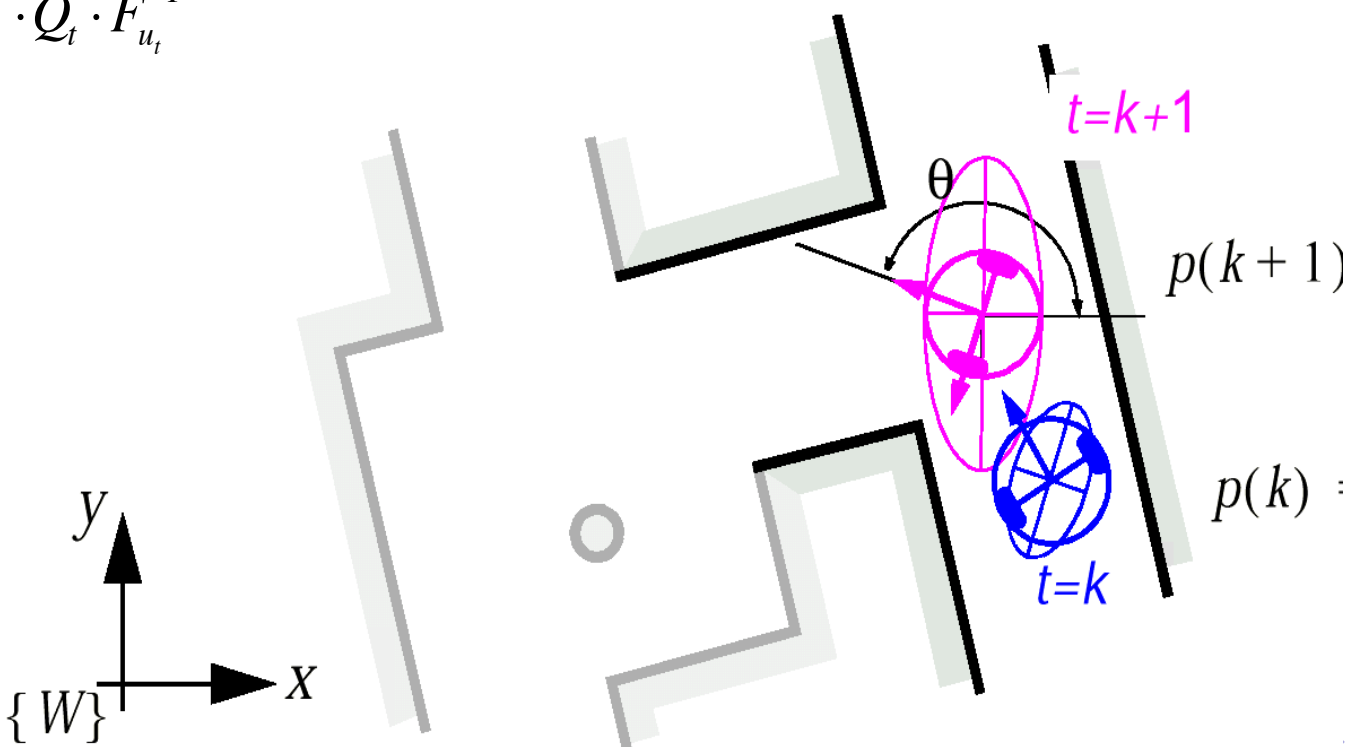
Differential Drive Robot — Prediction

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

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

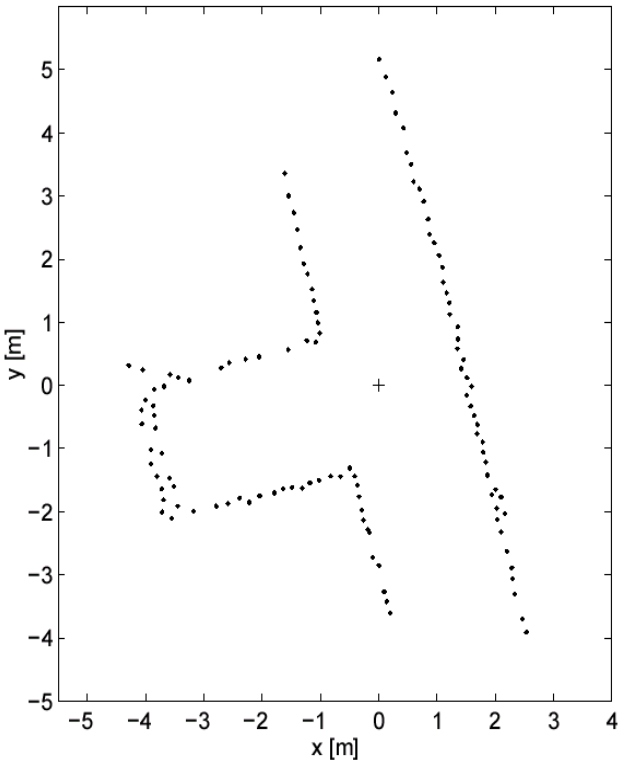
$$P_t = F_{x_{t-1}} \cdot P_{t-1} \cdot F_{x_{t-1}}^{\ T} + F_{u_t} \cdot Q_t \cdot F_{u_t}^{\ T}$$

Odometry

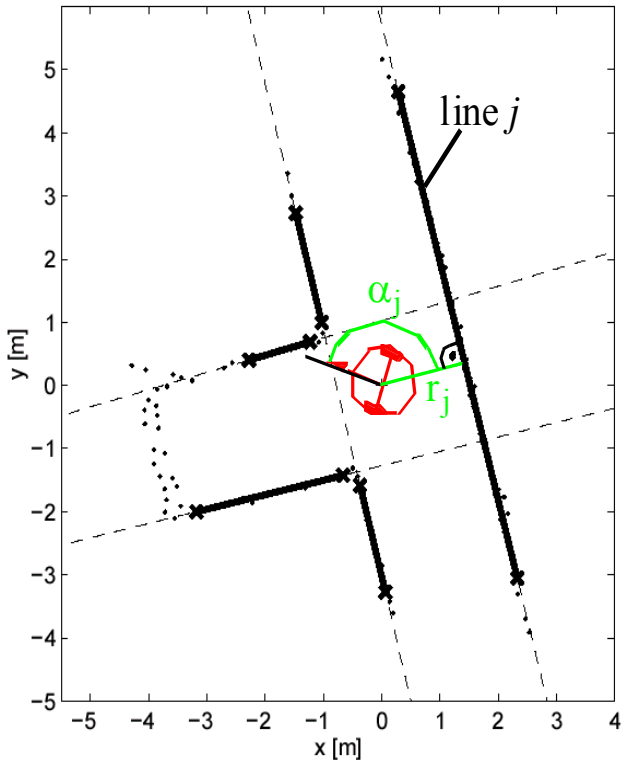


Differential Drive Robot — Observation

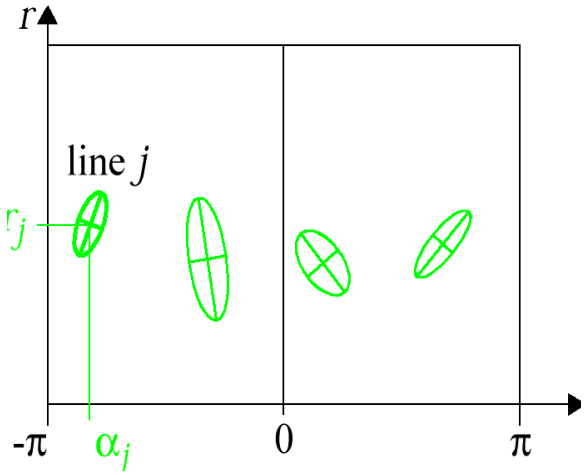
Raw Date of Laser Scanner



Extracted Lines



Extracted Lines in Model Space



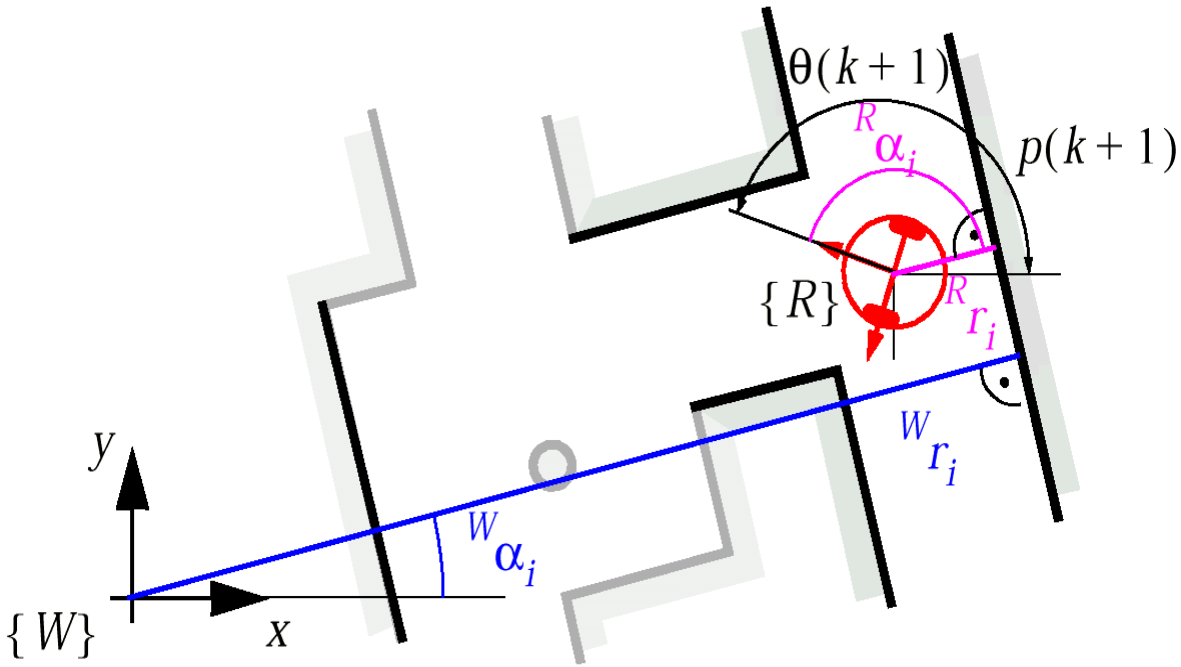
$$z_j = {}^{R}\begin{bmatrix} \alpha_j \\ r_j \end{bmatrix}$$

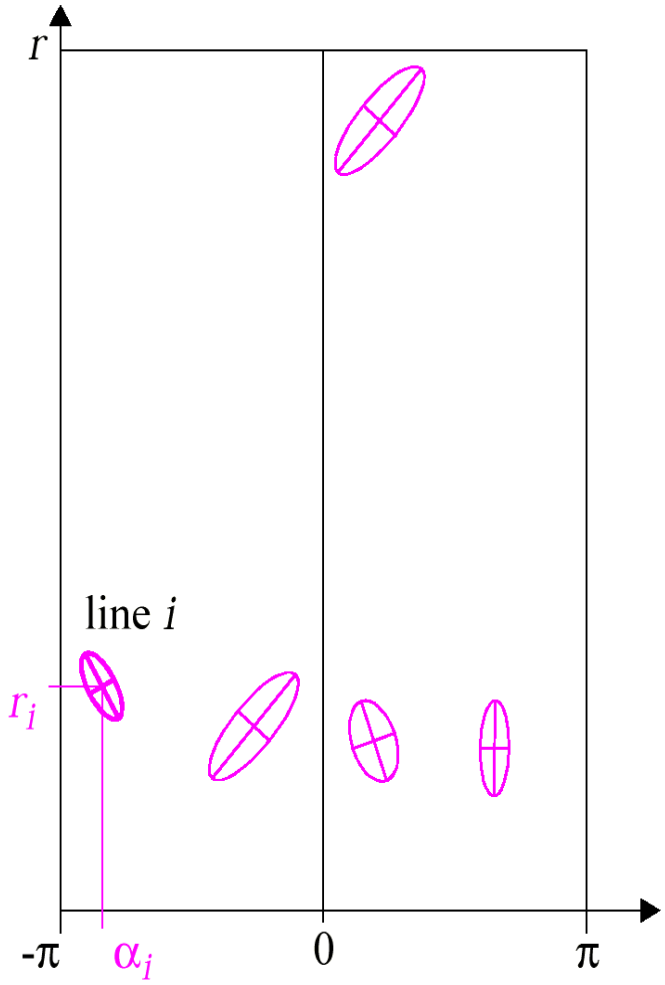
Sensor (robot) frame

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

Differential Drive Robot — Measurement

- For prediction, only the walls that are in the field of view of the robot are selected.
- This is done by linking the individual lines to the nodes of the path





Differential Drive Robot — Measurement

- 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 by

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

Differential Drive Robot — 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 an corresponding observation is found we calculate the innovation:

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

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

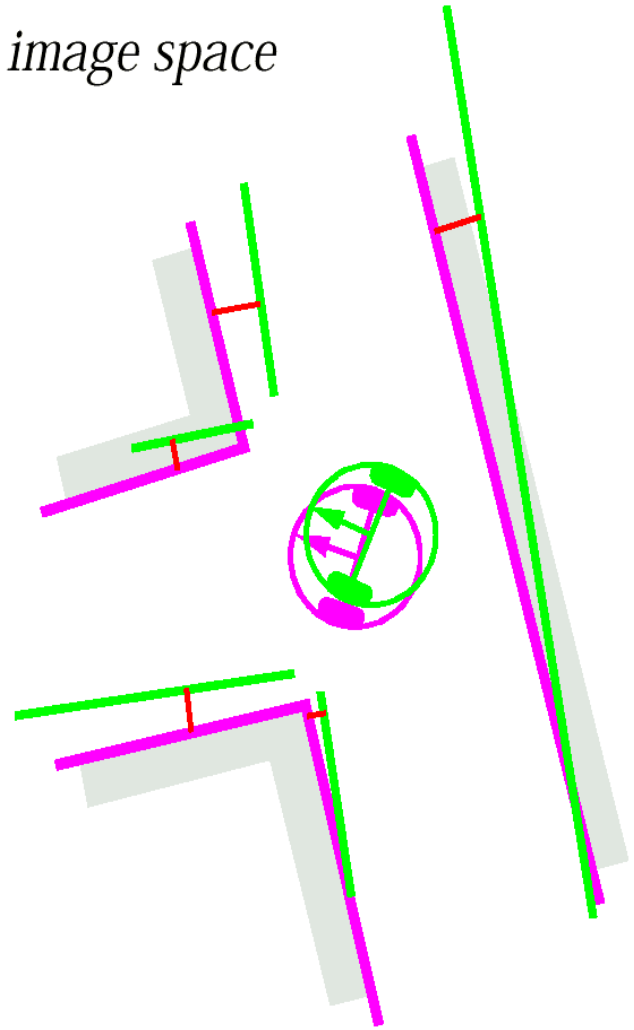
and its innovation covariance found by applying the error propagation law:

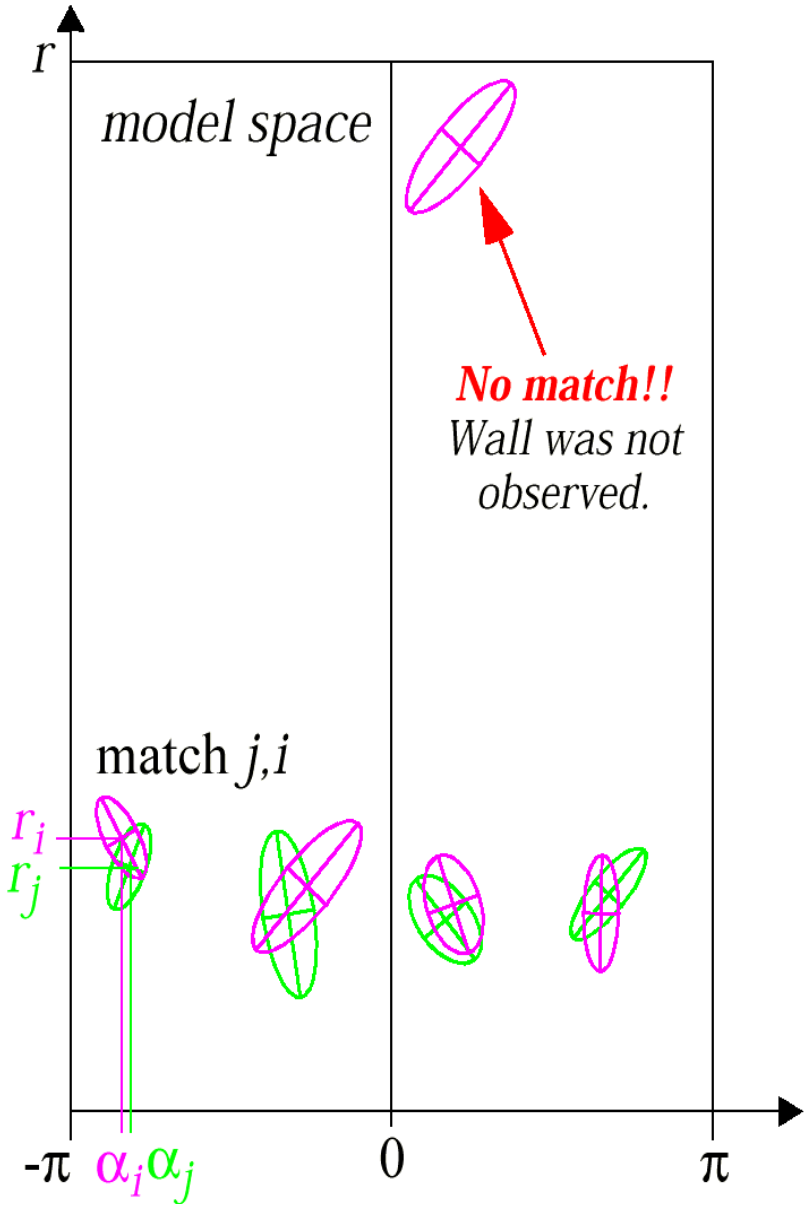
$$\Sigma_{IN,ij}(k+1) = \nabla h_i \cdot \Sigma_p(k+1|k) \cdot \nabla h_i^T + \Sigma_{R,i}(k+1)$$

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

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

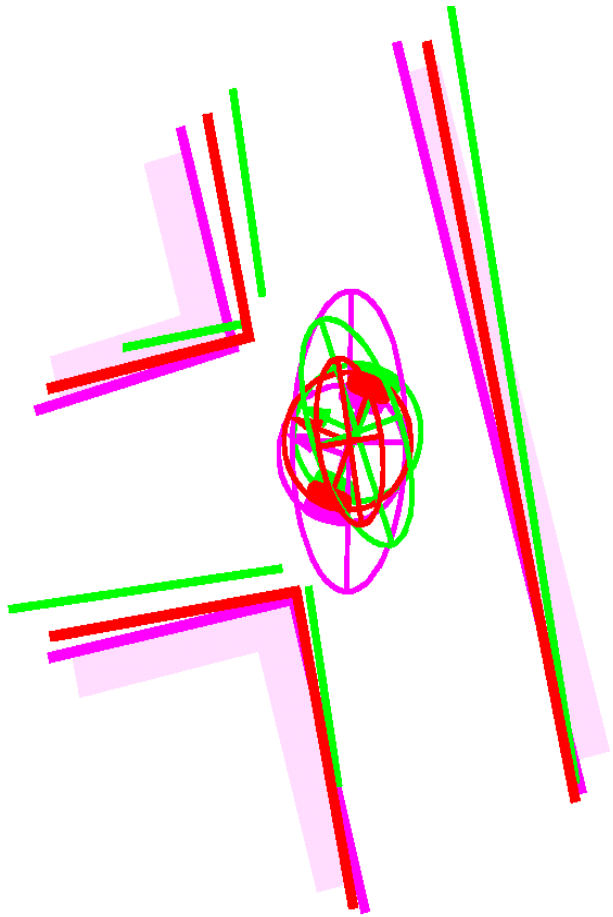
Differential Drive Robot — Matching





Differential Drive Robot — Matching

- Kalman filter estimation of the new robot position x_t :
  - 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)



https://www.youtube.com/watch?v=MELYZ5r5V1c

https://www.youtube.com/watch?v=XUW5jY9eOVI

Summary

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!