Maastricht University

Department of Data Science and Knowledge Engineering

Autonomous Robotic Systems

Master Course

Assignment
Mobile Robot Localization with Kalman Filter

Localization

"Using sensory information to locate the robot in its environment is the most fundamental problem to providing a mobile robot with autonomous capabilities." [Cox '91]

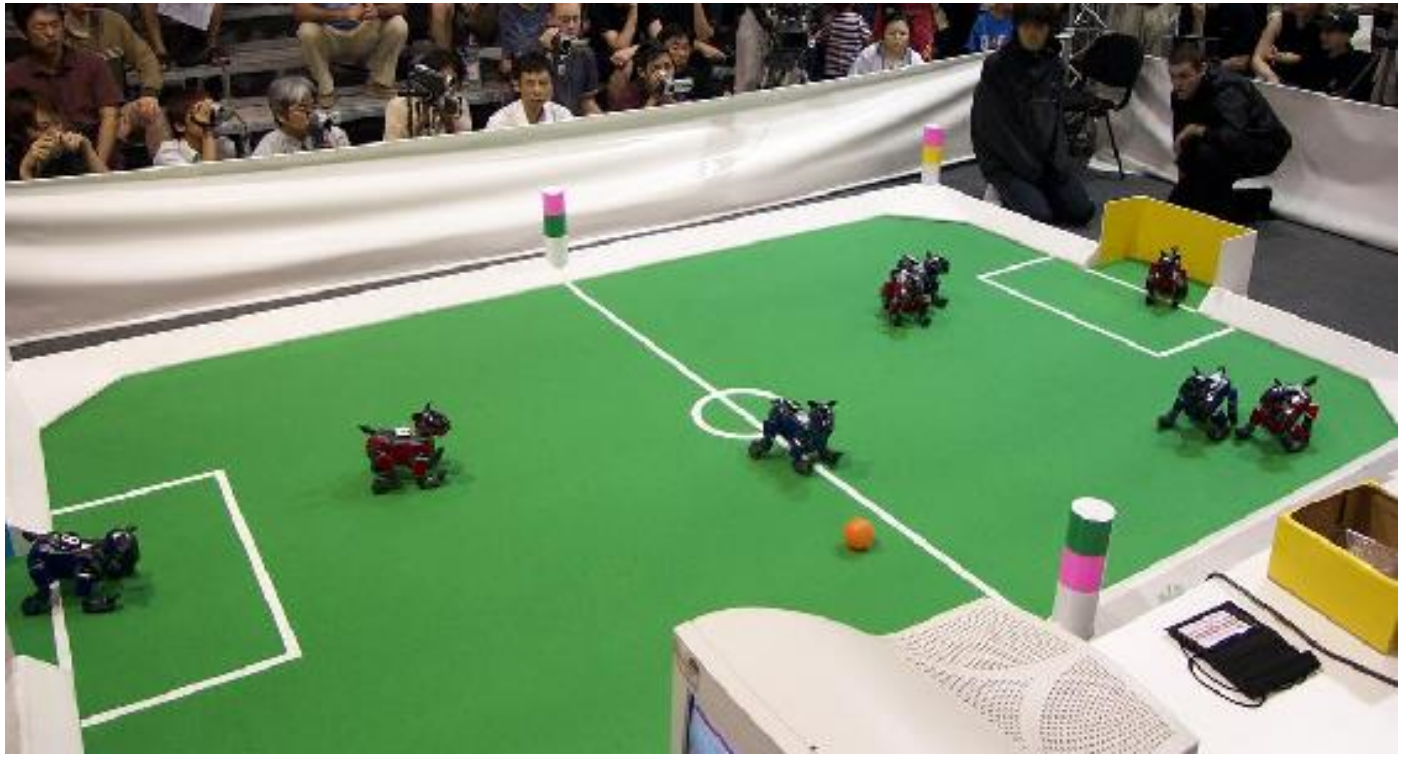
- **Given**
  - Map of the environment.
  - Sequence of sensor measurements.
- **Wanted**
  - Estimate of the robot's pose (position and orientation).
- **Pose estimation through Bayes Filter**

Localization can involve

- Self-localization of a robot
  - find own pose on map
  - = find coordinate transformation between global coordinate system of map and own local coordinate system centred at robot
- Localization of other objects' pose e.g. for object tracking and manipulation

→ Fundamental problems of robotic systems

Landmark-based Localization



- Localize yourself with six landmarks
- Localize ball and other robots

Localization → obtain own trajectory

Victoria Park Data Set [courtesy by E. Nebot]

FieldLab Robotics in Roermond

ROBOMOTIVE
Robot cells,
Robot vision

AGV R
Automatic
Guided
Vehicles

madolex
Machine-
shop,
Productie
Management
Software

LIOF
Innovation
-network

Maastricht University
Artificial
intelligence
Machine
Learning

Fontys
Logistics

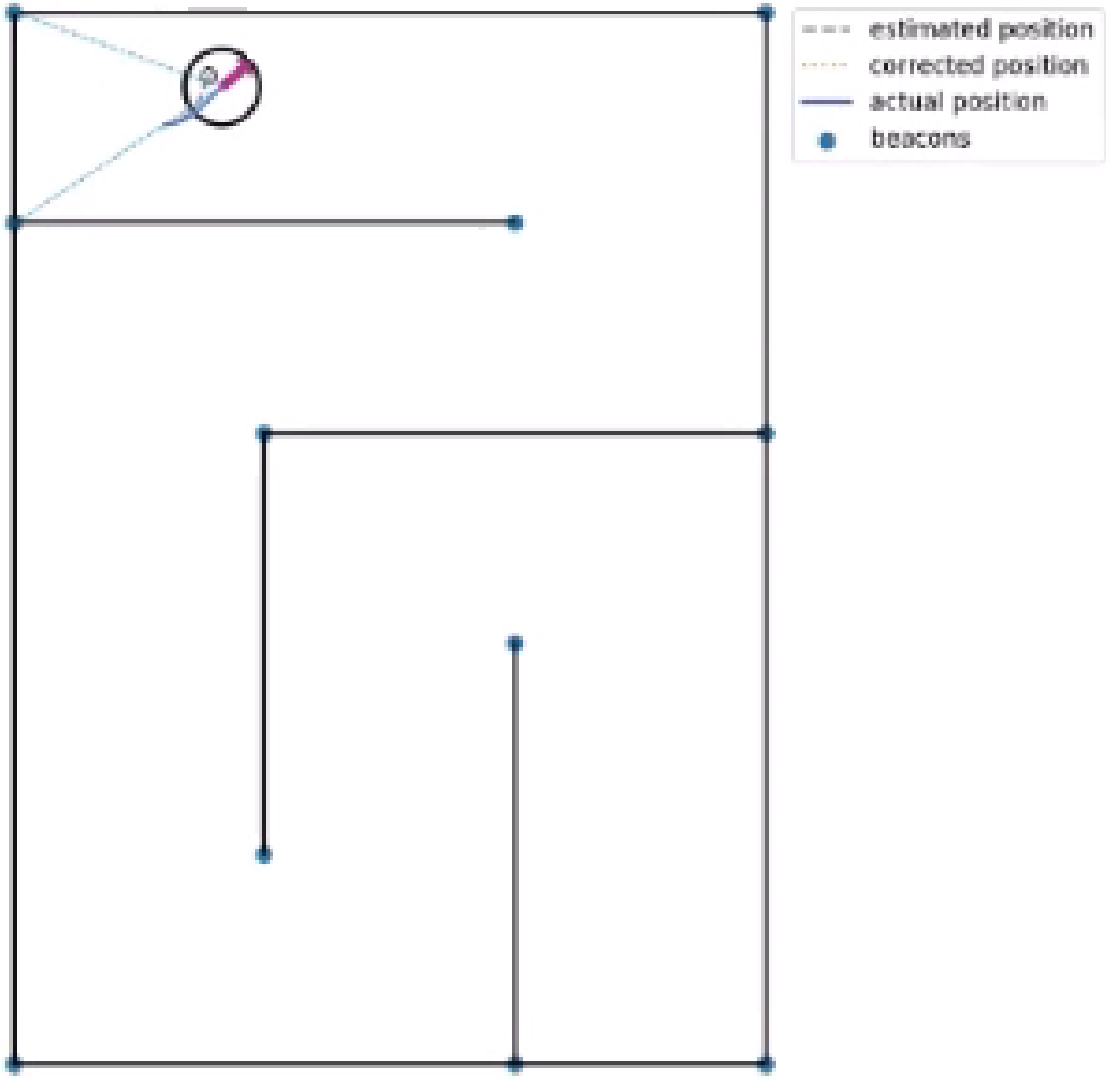
ACCERION
Robot vision
localization

https://www.1limburg.nl/robots-krijgen-eigen-plekje-roermond

Robomotive: eye-hand coordination

AGVR → autonomous fork lifter

Goal of assignment: demonstrate self-localization of mobile robot with Kalman Filter



Localization requires map

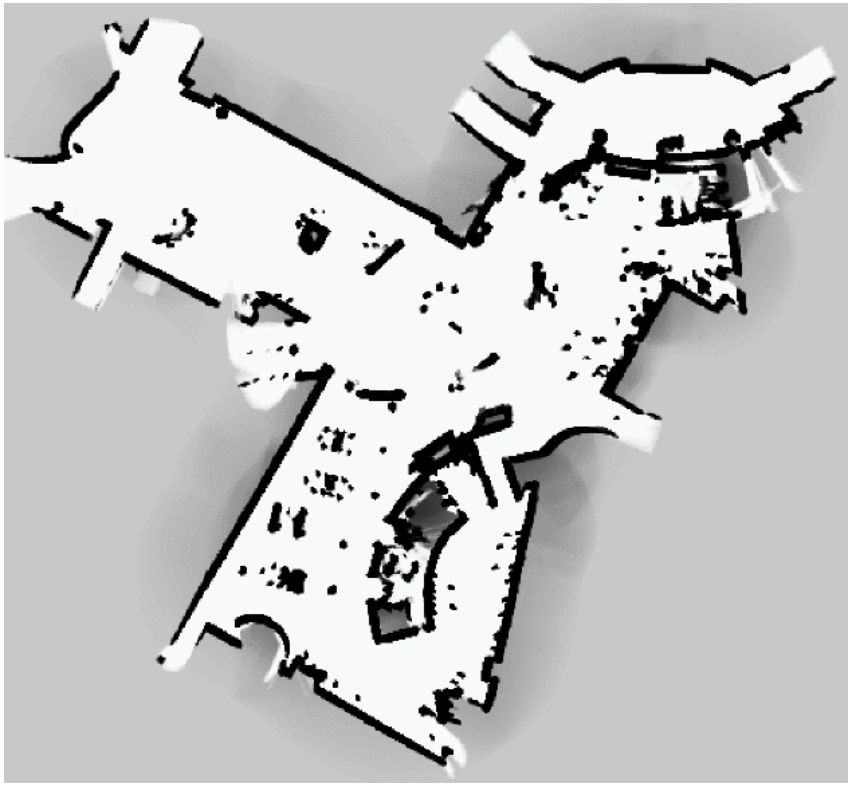
- Formally, a map m is a list of objects in the environment along with their properties:

$$m = \{m_1, m_2, \ldots, m_N\}$$

- Here N is the total number of objects in the environment, and each m_n specifies a property.
- Location-based maps: index n corresponds to a specific location. In planar maps often $m_{x,y}$ is used.

- Feature-based maps: n is feature index. Value of m_n contains properties of feature and Cartesian location of feature.

Example location-based map: Occupancy grid map

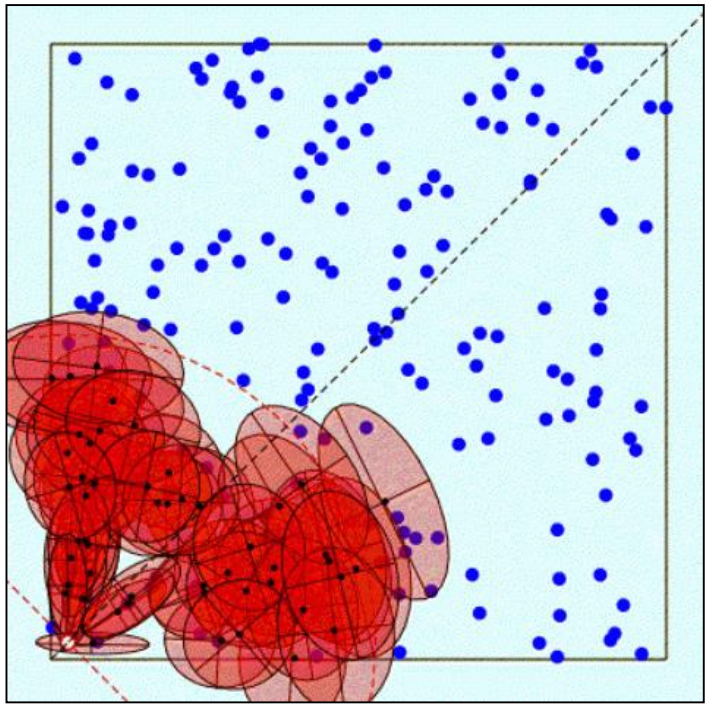


Scales badly to 3D in naïve implementation

- Each grid unit stores propability that space is occupied
- E.g. 1 = 100% certain that grid unit is occupied,
- 0 = 100% certain that grid unit is free
- Values between 0 and 1 mean lower confidence
- 0.5 = no evidence

Example: feature-based map: Landmark-based

- Highly efficient representation for sparse features
- Feature coordinates and properties are being stored
- Requires feature extraction (which can fail) e.g. corner detection (see computer vision)



- More information on 3D maps in later lecture

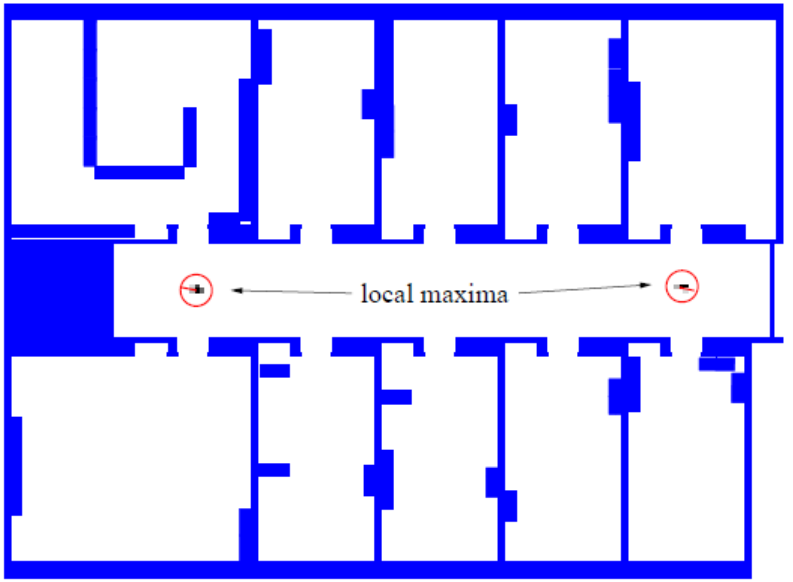
Known vs. unknown correspondence

- Known correspondence: all features have unique labels/identifiers
  - E.g. bar code, unique color code, unique building on map
- Unknown correspondence: features do not have distinct properties
  - E.g. doors in corridor look alike
  - Requires extra work to solve correspondence problem

Local vs. Global localization

- Local localization
  - Initial position is known = position tracking
  - Risk: if given initial position is wrong …
- Global localization
  - Initial position is unknown
  - Similarities in map might make it difficult to identify position
  - More difficult than local localization

What can standard Kalman Filter do?



Kidnapped robot problem

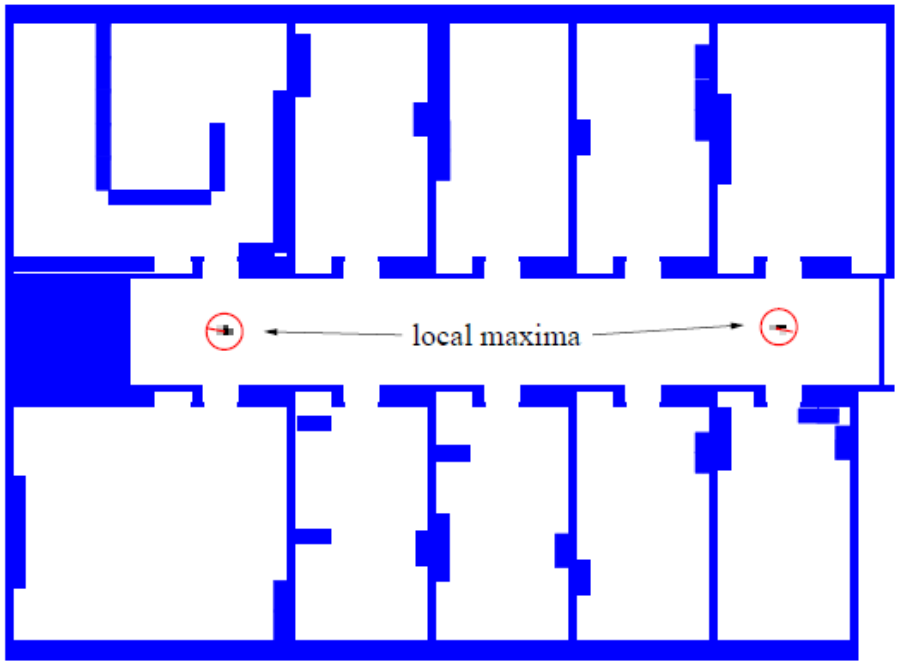
- While localization is running, robot is blind-folded, picked up and placed at random location.
- Considered most difficult
- Shows ability of localization algorithm to recover from failure
- Many efficient localization algorithms cannot solve this problem
- *How about Kalman Filter?*

Static vs dynamic environments

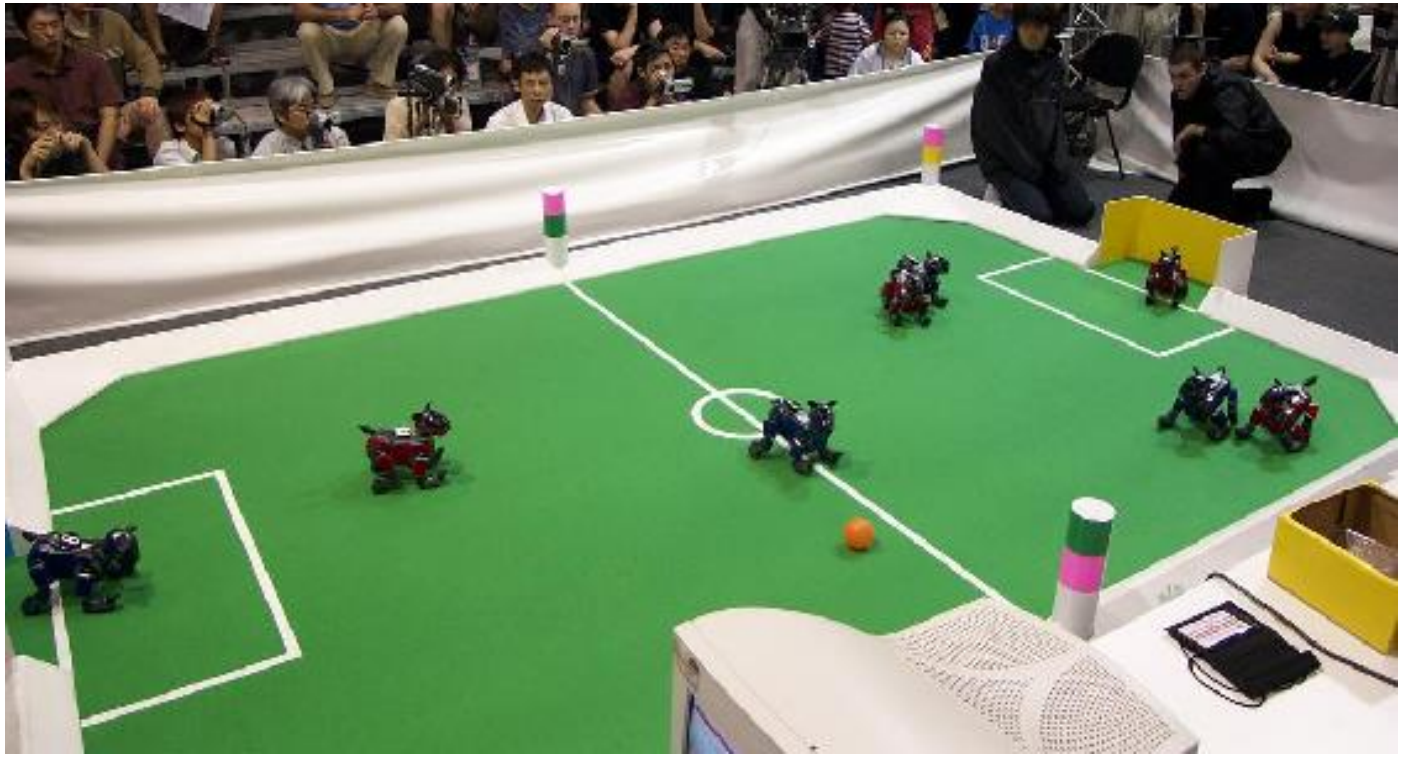
- Static environment: only robot is moving
  - All changes in observations are caused only by change in robot pose
  - Easier
- Dynamic environment: environment changes over time
  - E.g. people moving in room
  - More challenging
  - If room looks differently, is it not obvious if this is a different room or if robot revisits same location but room has changed
  - Might require that environment is part of the model (state vector and transitions)

Passive vs active localization

- Passive localization
  - Localization algorithm is running in the background and cannot control movement of robot
- Active localization
  - Localization algorithm can control robot
  - Can be used to resolve uncertainties



Single-robot vs Multi-robot localization



- Robots can increase accuracy of own pose estimate by localizing and using pose information of other robots

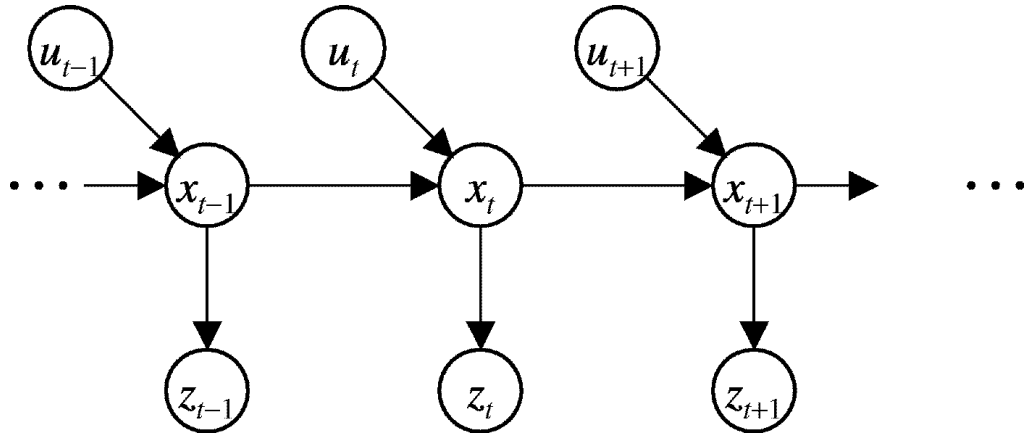
Assignment focus on

- Feature-based map
- Known correspondence
- Local localization
- Passive localization
- Single-robot localization
- Static environment

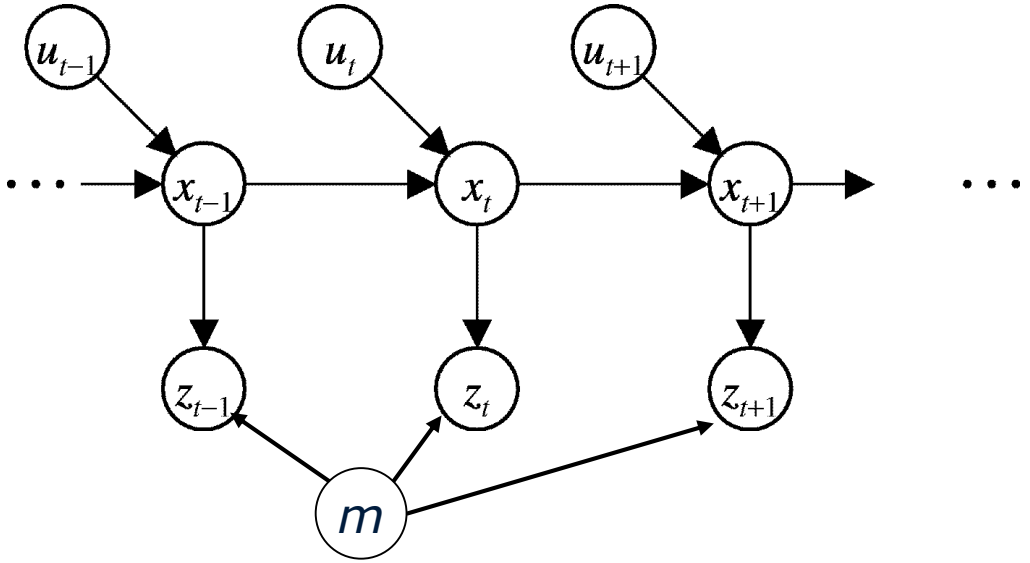
→ Pose tracking

Graphical representation of

Bayes Filter:



Markov Localization:



1: **Algorithm Bayes_filter**($bel(x_{t-1}), u_t, z_t$)**:**
2: for all x_t do
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: endfor
6: return $bel(x_t)$

1: **Algorithm Markov_localization**($bel(x_{t-1}), u_t, z_t, m$)**:**
2: *for all* x_t *do*
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}, m)\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t, m)\ \overline{bel}(x_t)$
5: *endfor*
6: *return* $bel(x_t)$

LOCALIZATION WITH KALMAN FILTER

Bayes Filter Reminder

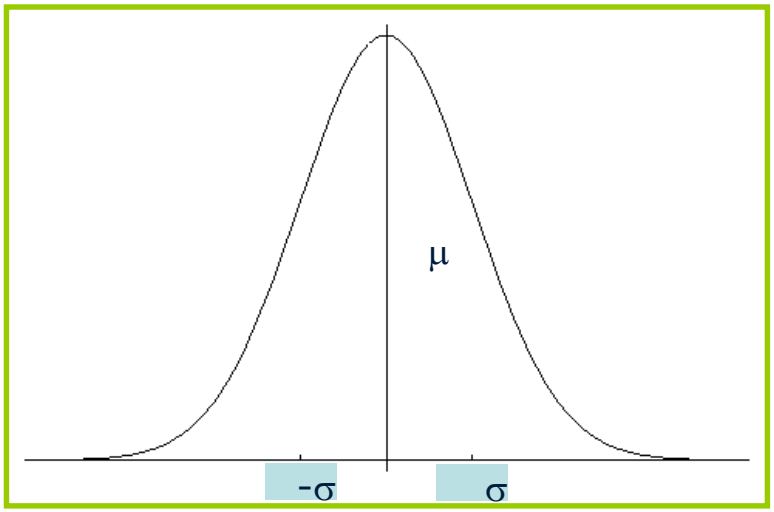
- Prediction

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

- Correction

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

What if *bel(x) would be modelled by a* Gaussian?



Univariate

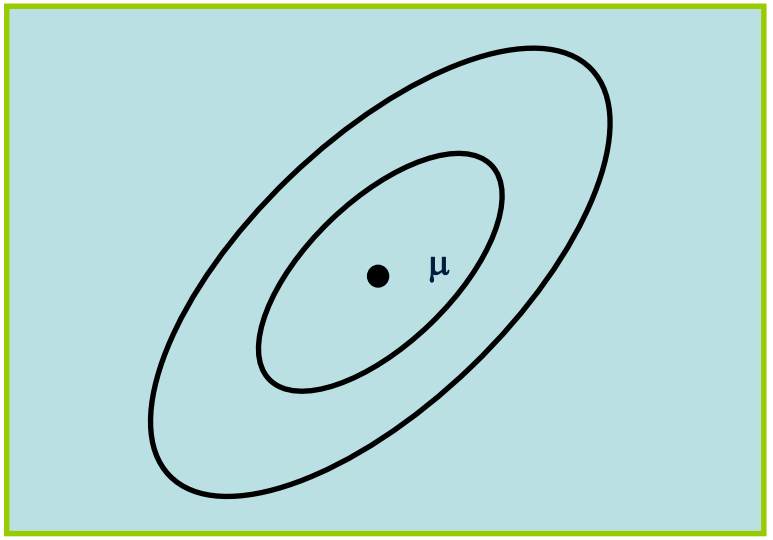
$$p(x) \sim N(\mu, \sigma^2):$$

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

Multivariate

$$p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}):$$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$



Properties of Gaussians

$$\left.\begin{aligned} X &\sim N(\mu, \sigma^2) \\ Y &= aX + b \end{aligned}\right\} \Rightarrow \quad Y \sim N(a\mu + b, a^2\sigma^2)$$

$$\left.\begin{aligned} X_1 &\sim N(\mu_1, {\sigma_1}^2) \\ X_2 &\sim N(\mu_2, {\sigma_2}^2) \end{aligned}\right\} \Rightarrow p(X_1) \cdot p(X_2) \sim N\left(\frac{{\sigma_2}^2}{{\sigma_1}^2 + {\sigma_2}^2} \mu_1 + \frac{{\sigma_1}^2}{{\sigma_1}^2 + {\sigma_2}^2} \mu_2, \quad \frac{1}{{\sigma_1}^{-2} + {\sigma_2}^{-2}} \right)$$

$$\left.\begin{aligned} X &\sim N(\mu, \Sigma) \\ Y &= AX + B \end{aligned}\right\} \Rightarrow \quad Y \sim N(A\mu + B, A\Sigma A^T)$$

$$\left.\begin{aligned} X_1 &\sim N(\mu_1, \Sigma_1) \\ X_2 &\sim N(\mu_2, \Sigma_2) \end{aligned}\right\} \Rightarrow p(X_1) \cdot p(X_2) \sim N\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2} \mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2} \mu_2, \quad \frac{1}{\Sigma_1^{-1} + \Sigma_2^{-1}} \right)$$

- We stay in the "Gaussian world" as long as we start with Gaussians and perform only linear transformations.

What if we assume linear models → Discrete Kalman Filter

Estimates the state x of a discrete-time controlled process that is governed by the linear stochastic difference equation

$p(x_t|x_{t-1}, u_t) \rightarrow$ $x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$

with a measurement

$p(z_t|x_t) \rightarrow$ $z_t = C_t x_t + \delta_t$

Components of a Kalman Filter

A_t Matrix (nxn) that describes how the state evolves from t-1 to t without controls or noise.

B_t Matrix (nxl) that describes how the control u_t changes the state from t-1 to t.

C_t Matrix (kxn) that describes how to map the state x_t to an observation z_t.

ε_t δ_t Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

Kalman Filter Algorithm

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

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

Correction:

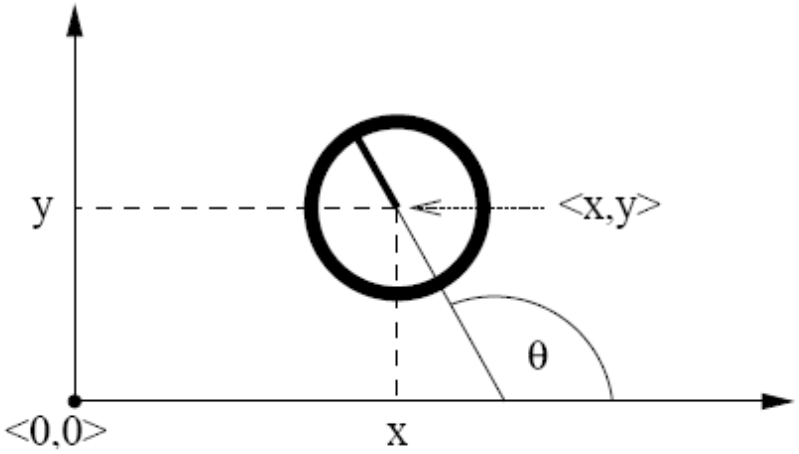
4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$

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

7. Return μ_t, Σ_t

Coordinate Systems

- In general the configuration of a robot can be described by six parameters.
- Three-dimensional Cartesian coordinates plus three Euler angles pitch, roll, and tilt.
- Throughout this section, we consider robots operating on a planar surface.
- The state space of such systems is three-dimensional (x,y,θ).



Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

State

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

Initialize with known pose

Covariance

$$\Sigma_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

Initialize with small values corresponding to estimated error in initial position

Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

A, B, u?

→ *Need to understand how to control robot and how to estimate effect of environment and control*

→ *Motion model*

→ *We already derived and implemented a first motion model. Remember?*

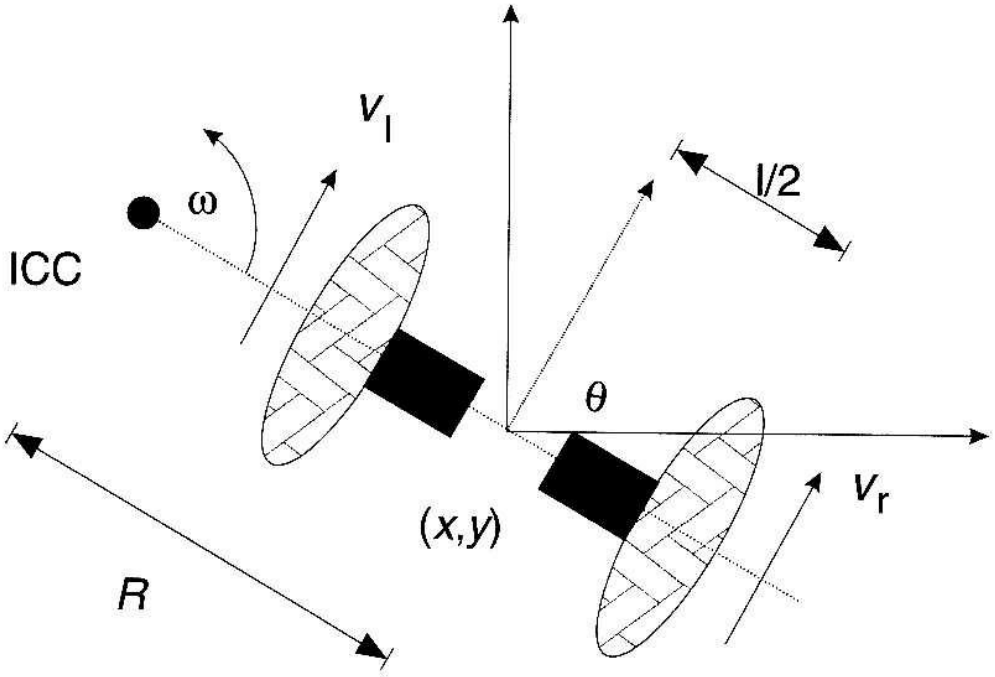
Probabilistic Motion Models

- To implement the Bayes Filter, we need the transition model $p(x \mid x', u)$.
- The term $p(x \mid x', u)$ specifies a posterior probability, that action u carries the robot from x' to x.
- In this section we will specify, how $p(x \mid x', u)$ can be modeled based on the motion equations.
- Allow modeling of motion errors.

MOTION MODELS

How to derive and implement motion model for wheeled robot with differential drive?

- You actually already implemented a motion model for your mobile robot simulator:
- Remember?
- Now we assume direct control over translation and rotation velocity



Dead Reckoning

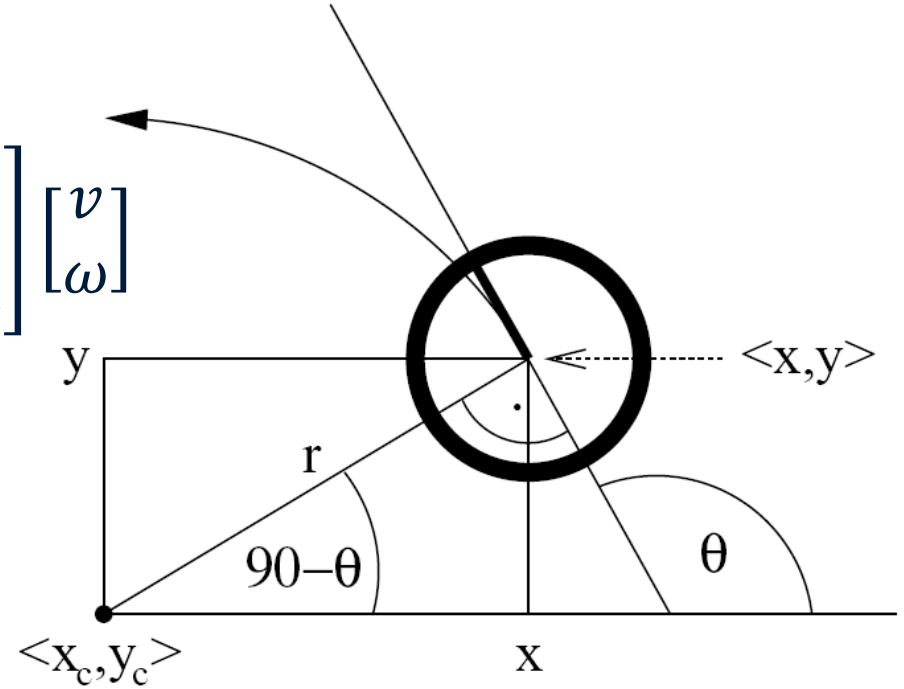
- Derived from “deduced reckoning.”
- Mathematical procedure for determining the present location of a vehicle.
- Achieved by calculating the current pose of the vehicle based on its velocities and the time elapsed.

→ Velocity-based motion model

Velocity-Based Model

An extremely simplified version of this model assumes independence of rotation and translation
= for each time step, we first apply translation to update x,y and then apply rotation to update orientation of robot

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \delta t \cos\theta & 0 \\ \delta t \sin\theta & 0 \\ 0 & \delta t \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$



Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \delta t \cos\theta & 0 \\ \delta t \sin\theta & 0 \\ 0 & \delta t \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

so

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

$$u = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

$$B = \begin{bmatrix} \delta t \cos\theta & 0 \\ \delta t \sin\theta & 0 \\ 0 & \delta t \end{bmatrix}$$

Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

How about R?

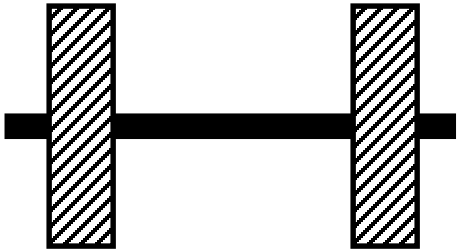
$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

R is covariance matrix defining noise of motion model ε

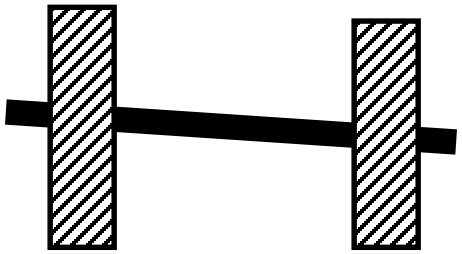
$$R_t = \begin{bmatrix} \sigma_{Rx}^2 & 0 & 0 \\ 0 & \sigma_{Ry}^2 & 0 \\ 0 & 0 & \sigma_{R\theta}^2 \end{bmatrix}$$

Initialize with small values corresponding to estimated error in motion model (in real robot has to be obtained through experiments)

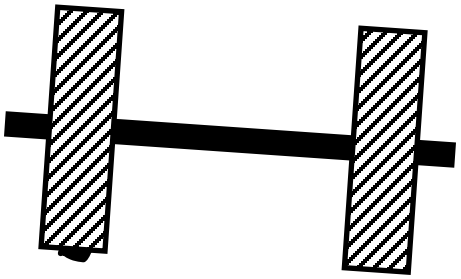
Reasons for Motion Errors



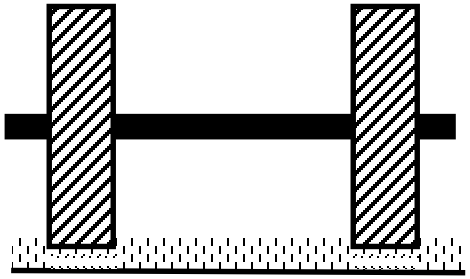
ideal case



different wheel diameters



bump



carpet

and many more …

Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

How to include sensor information?

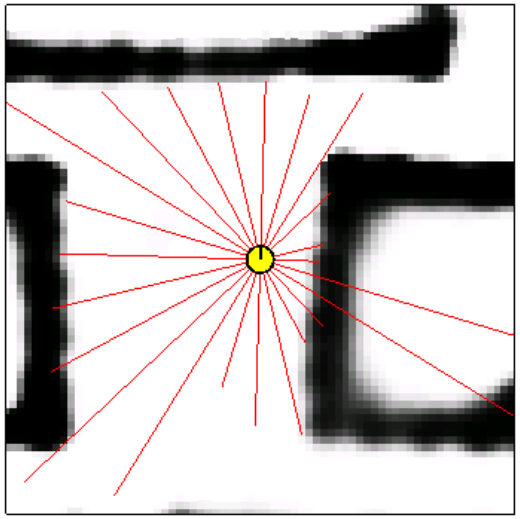
$$z_t = C_t x_t + \delta_t$$

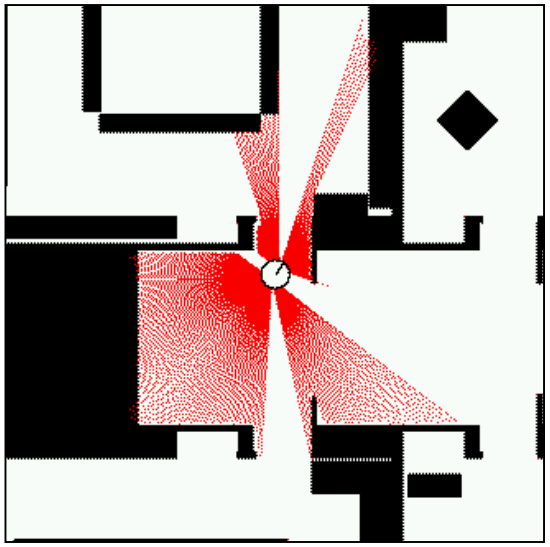
Model assumes that state causes sensor readings. Here we need to estimate state from sensor readings = inverse

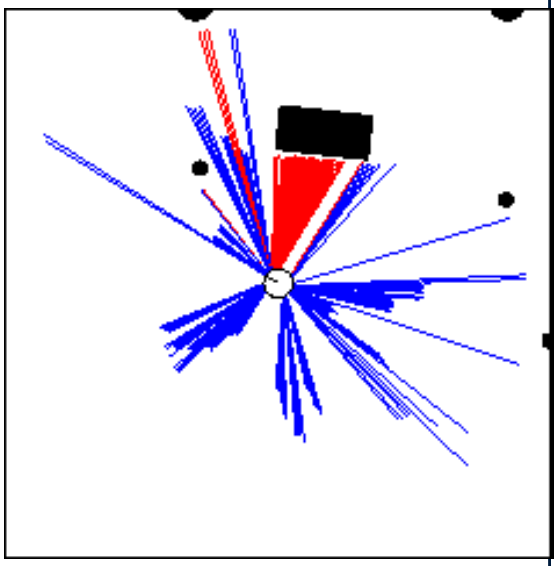
How to get *C* and *z*?

→ Need a sensor model

(Simplified) Beam-based Sensor Model







Omnidirectional sensor that emits beams in a directions. From reflection of beam, sensor estimates distance and bearing (direction) of observed feature.
E.g. range finder or stereo camera

Feature-based measurements

If we denote the range by r, the bearing by ϕ, and the signature by s, the feature vector is given by a collection of triplets

$$f(z_1) = \{f_t^1, f_t^2, \ldots\} = \left\{ \begin{pmatrix} r_t^1 \\ \Phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \Phi_t^2 \\ s_t^2 \end{pmatrix}, \ldots \right\}$$

The measurement vector for a noise-free landmark sensor is easily specified by the standard geometric laws. We will model noise in landmark perception by independent Gaussian noise on the range, bearing, an the signature. The resulting measurement model is formulated for the case where the i-th feature at time t corresponds to the j-th landmark in the map. As usual, the robot pose is given by $x_t = (x \;\; y \;\; \theta)^T$.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \text{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \\ s_j \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_r^2} \\ \varepsilon_{\sigma_\phi^2} \\ \varepsilon_{\sigma_s^2} \end{pmatrix}$$

The atan2 Function

- Extends the inverse tangent and correctly copes with the signs of x and y.

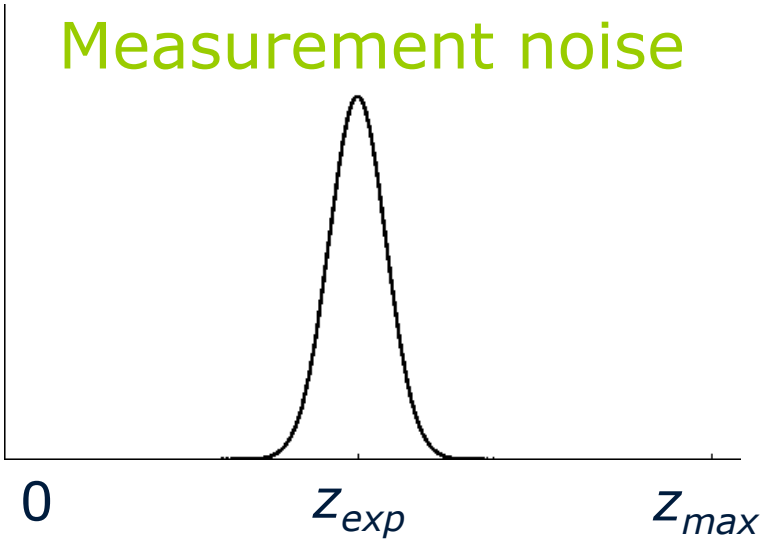
$$\text{atan2}(y,x) = \begin{cases} \text{atan}(y/x) & \text{if } x > 0 \\ \text{sign}(y)\ (\pi - \text{atan}(|y/x|)) & \text{if } x < 0 \\ 0 & \text{if } x = y = 0 \\ \text{sign}(y)\ \pi/2 & \text{if } x = 0, y \neq 0 \end{cases}$$

Feature-based measurements

- For assignment make things simple: calculate exact pose of robot from landmarks within sensor range and add noise afterwards to mimic sensor noise
- If we observe several features, we use the average pose estimated over all features

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

(Simplified) Beam-based Proximity Model



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

- We assume Gaussian noise
- In a real-world application, this noise would have to be estimated through experimentation

Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

How to include sensor information?

$$z_t = C_t x_t + \delta_t$$

$$z_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \qquad C_t = I$$

Pose tracking with KF

1. Algorithm **Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

Prediction:

2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

Correction:

4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7. Return μ_t, Σ_t

How about Q?

$$z_t = C_t x_t + \delta_t$$

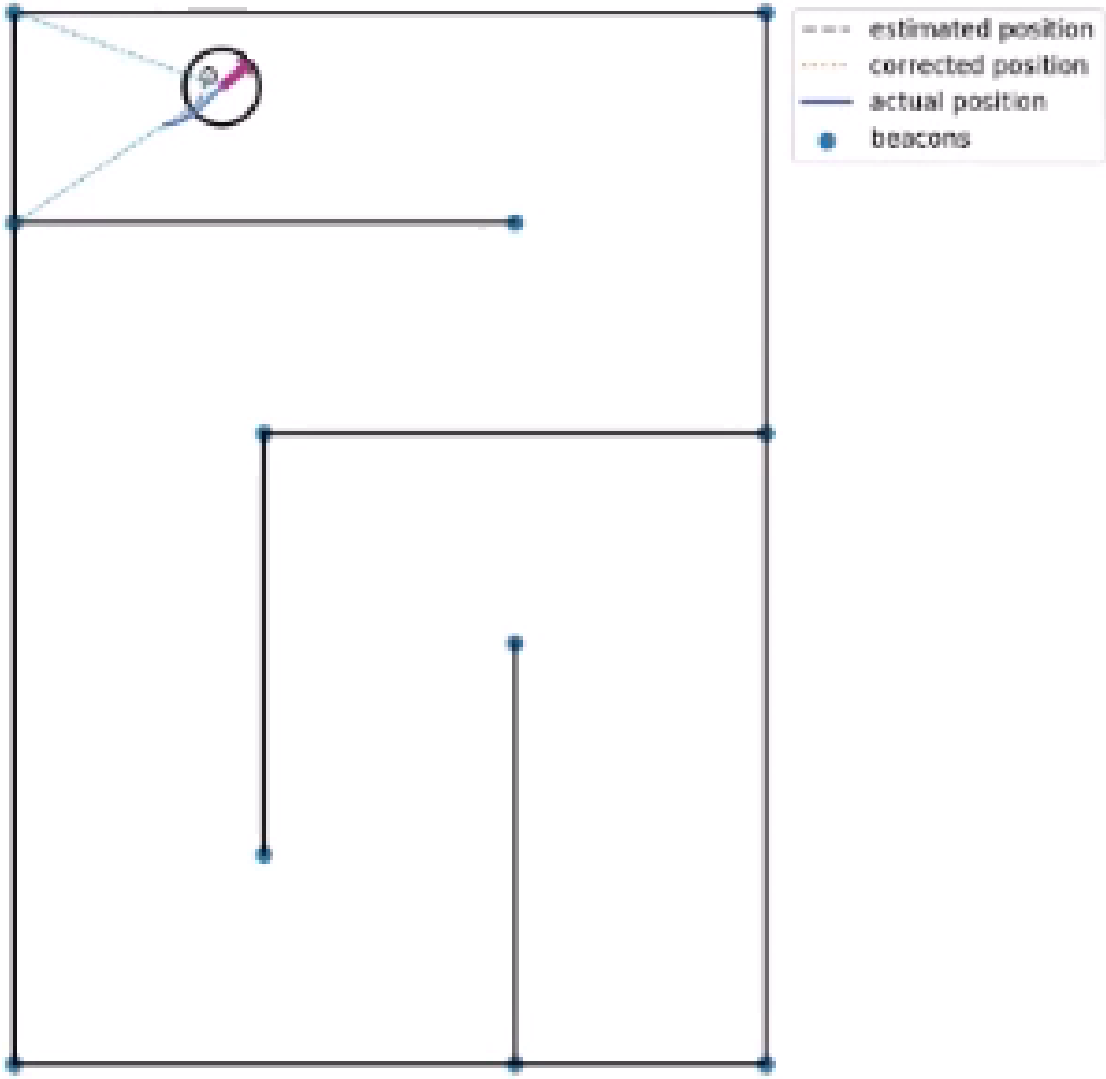
Q is covariance matrix defining noise of sensor model δ

$$Q_t = \begin{bmatrix} \sigma_{Qx}^2 & 0 & 0 \\ 0 & \sigma_{Qy}^2 & 0 \\ 0 & 0 & \sigma_{Q\theta}^2 \end{bmatrix}$$

Initialize with small values corresponding to estimated error in motion model (in real robot has to be obtained through experiments)

ASSIGNMENT MOBILE ROBOT LOCALIZATION

Goal of assignment: demonstrate self-localization of mobile robot with Kalman Filter



More instructions

- Point-based features
- No walls, no collision with features
- Omnidirectional sensor for feature detection
- Limited sensor range
- Bearing and distance estimate
- Known correspondence
- Velocity-based motion model ($u = (v, \omega)^T$)
- Control robot with key board (W=increment v, S=decrement v, A=decrement ω, D=increment ω, X=stop)
- Track pose with Kalman Filter

Visualization

Forward

Solid line = actual robot trajectory

Ellipse: show intermediate estimates of position and covariance

Dotted line = estimated robot trajectory

Feature = black point

Draw green line between robot and feature, if feature is in sensor range

Hand in

- Documented code (Python, C++, C, Java, Matlab)
  - Make sure that each group member codes something, add names to code (who did what?)
  - Upload plain files to Student portal (or zip archives)
- Short demonstration of localization in class (in two weeks)
  - Allow user (me!) to drive robot inside field with landmarks
  - Demonstrate what happens if no landmarks are visible
  - Demonstrate what happens if 3 landmarks are visible
  - **Make sure to show proper visualization** so we can see output of covariance matrices

Deadlines

- Deadline: Tuesday March 24th before the lecture
- Implementation and experiments take time: **START NOW!**
- Discuss today (Localization, simulator, approach, experiments, work distribution)
- Prepare questions until next time

Groups

- Groups of ideally 3 students (not more than 3, not less than 2)
- Do not forget to register again to the student portal. Use same group number as before.
- If you wish to switch groups (for a good reason) tell me first

Plagiarism

- This is a group assignment
- Help other members of your group
- Do not copy and hand in code or reports from other groups
- You can use libraries for calculating intersections between lines, between lines and circles, and for visualization
- **You must not use a library for Kalman Filter implementation**
- Write your own software

Write simple software

- No need to use (a lot of) objects
- Use functions
- Do not distribute code over too many files
- Avoid complex constructions and data structures

Now

Do not run away!

- Register to new assignment on student portal to group with same group number as last time
- Start working on assignment: make plans
- There is plenty of reason to discuss!
- I move around and test your mobile robot assignment. **Have simulator running so we do not loose time!**

Do not run away!