Mobile robot control 2021: Tutorial #2
Algorithms for robotics

MAY 7TH 2021

Bob Hendrikx

Mechanical engineering, Control Systems Technology

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Contents

Robot algorithms and examples in practice:

- *Localization*
- *Feature detection and tracking*
- *Robot motion planning and control*

- **Goal**: provide an overview of algorithms and techniques used for mobile robot control in practice

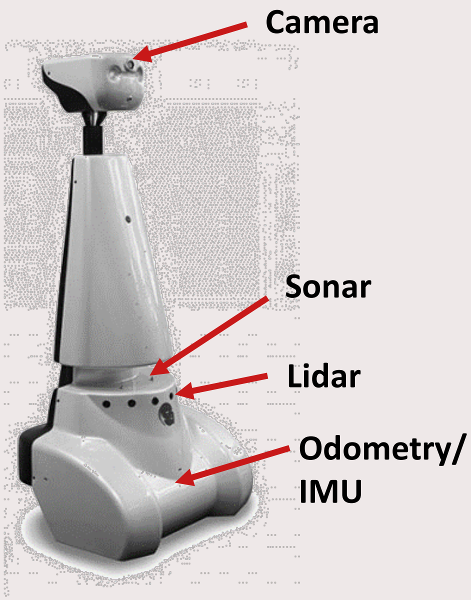
Robot localization

- Robots use ***proprioceptive*** *sensors* for local motion sensing
- Combined with ***exteroceptive*** sensors to *associate* with *external world* in which task is defined

Localization means:

- *Making* ***associations*** *between* ***sensor-data features*** *and* ***objects***
- **Infer** the **location** of *things* based on this **sensor data**

What **algorithms** can we apply to this problem?



TU/e

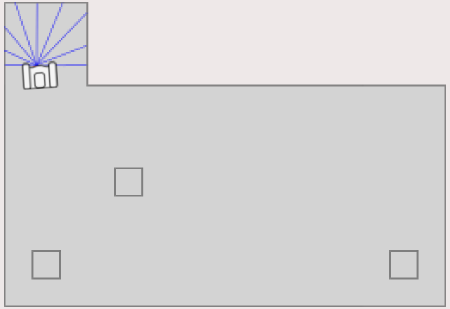
Robot localization

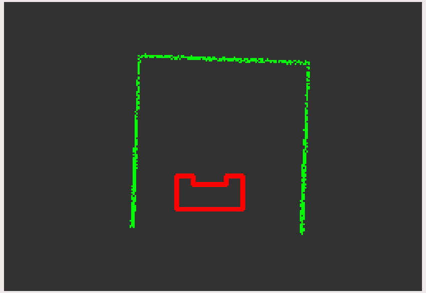
- *Making **associations** between **sensor-data features** and **objects***
- **Infer** the **location** of *things* based on this **sensor data**

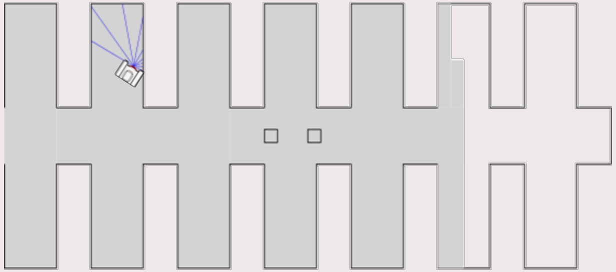
'Classical' localization formulation:
*"How to **infer** the **robot pose** from **sensor data**?"*

This is challenging because:
- We often cannot directly *sense* the robot pose
- What we can *sense* is obscured by *noise*
- What we sense does not uniquely determine the robot pose
- Dynamic objects are not on the map

Is every localization problem the same?







Classical taxonomy of localization problem

- **Tracking** - keeping track of the robot pose **starting** from **known location**
  - Scan matching / Kalman filters / Particle filters
- **Global localization** – Finding the robot pose **without initial knowledge**
  - Particle filters / Multiple hypothesis kalman filters
- **Kidnapped robot problem** – **Changing** the robot pose **without informing** it
  - Heuristic solutions

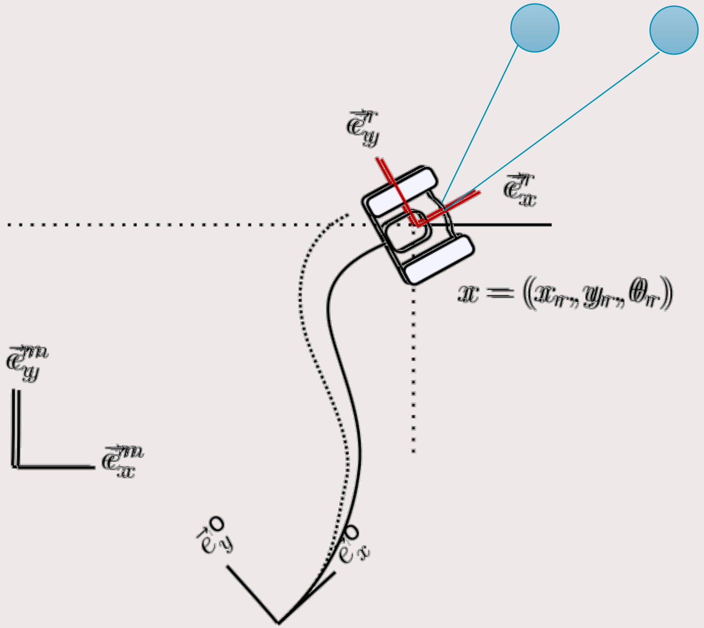
*All are **inference** and **data association** problems – just different levels of **prior knowledge***

Robot pose

- $x = (x_r, y_r, \theta_r)$ w.r.t. a reference frame
- *Convention: First translate – then rotate in place*

$$T = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}$$

- *Odometry provides a drifted pose…*
  … w.r.t. wherever the robot was turned on
- *Sensors can help eliminate drift by using a map*



Working with odometry

- Convert **odometry** to **relative poses** at sample times

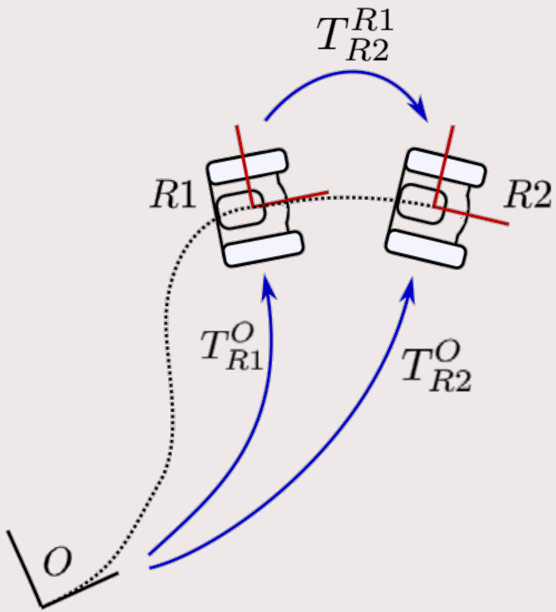
- Pre-multiply with inverse odometry at t1, to obtain the **relative pose** between time instant **t1** and **t2**:

$$T^O_{R2} = T^O_{R1} T^{R1}_{R2}$$

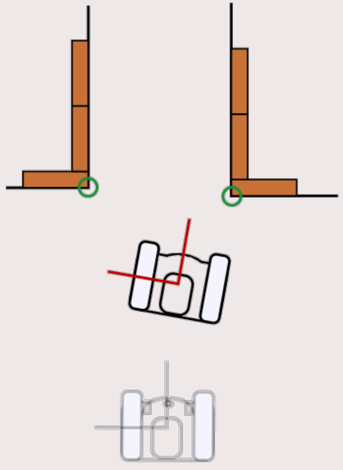
$$(T^O_{R1})^{-1} T^O_{R2} = (T^O_{R1})^{-1} T^O_{R1} T^{R1}_{R2} = T^{R1}_{R2}$$

- If we know the robot pose at time t1 on the map, we can easily obtain an odometry estimate for t2

$$T^M_{R2} = T^M_{R1} T^{R1}_{R2}$$



Eliminating drift using the map

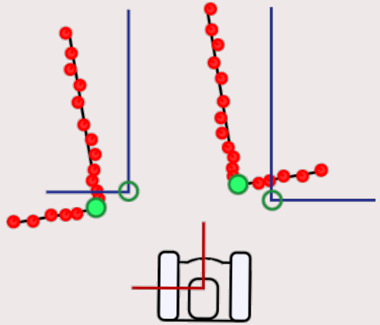


- The location in the world (top) will not match the odometry perfectly (bottom)

- Can we use the laserscan to correct for this?
- Find the correction that transforms the scan to the map, and use this to correct the robot pose in the map!

- But how do we do this?
- Possibility: extract **point features** and do **point registration**

- E.g.: use a **split-and-merge** procedure to extract **corner points** and find the correction that **minimizes the squared distance** between **scan** and **map**



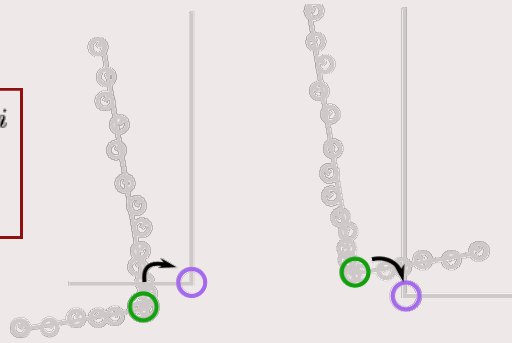
Basic feature extraction sketch

```
segments = [(p1,pend)]
While true:
        newsegments =[]
        for segment in segments[]:
                for point in segment.pointrange()
                        if distance(segment, point) > threshold
                                newsegments.update(segment, point)
                endfor
        endfor
        if newsegments = []:
                return segments
        else:
                segments.update(newsegments)
endwhile
```

Point registration in 2D

- Minimize the distance over $t = (x, y)$ and θ for corresponding points $p_i,\ m_i$

$$\min_{t,\theta} \sum_{i=1}^{N} (R(\theta)p_i + t - m_i)^T (R(\theta)p_i + t - m_i)$$



First find center-of-mass of points:

$$c_m = \frac{1}{N}\sum_i \begin{bmatrix} m_i^x \\ m_i^y \end{bmatrix}, \quad c_p = \frac{1}{N}\sum_i \begin{bmatrix} p_i^x \\ p_i^y \end{bmatrix}$$

Rotation matrix can be obtained through Singular Value Decomposition:

$$H = \sum_{i=1}^{N} (p_i - c_p)(m_i - c_m)^T$$

$$[U, S, V] = \mathrm{svd}(H), \quad R = VU^T$$

Translation part becomes:

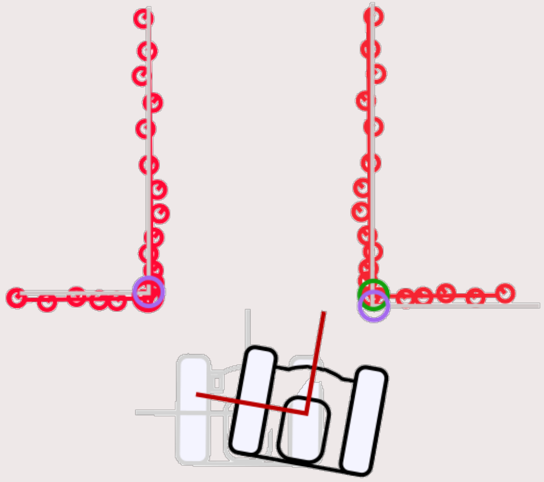
$$t = c_m - Rc_p$$

Feature matching variants are used often in practice (e.g. iterative-closest-point), but have limitations:

- What will happen if we have only one point?
- What will happen if we match wrong points?
- How can we incorporate knowledge of old pose uncertainty and sensor uncertainty?

Common strategies:

- Represent multiple hypotheses and throw away those that are unlikely
- Use a probablisitic framework to represent measurement uncertainty and robot pose uncertainty



TU/e

Modeling uncertainty

Continuous representation

- Model robot pose as multivariate Gaussian over x, y, theta
- Model odometry and measurement uncertainties as Gaussian white noise
- Use a Kalman filter to fuse odometry and laser -> "**recursive prediction – correction**"

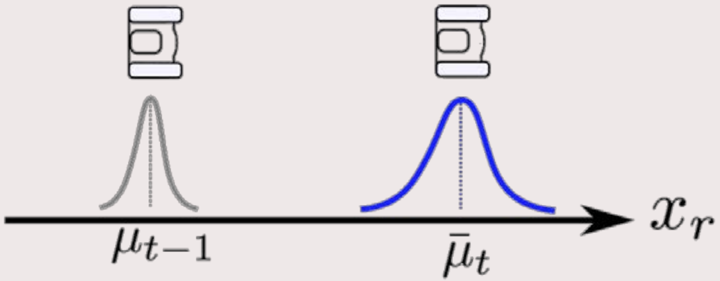
Discrete / sampled representation

- Model robot pose as multiple distinct hypotheses
- Evaluate the likelihood of the hypotheses given the measurements
- Create new hypotheses as needed and remove unlikely ones

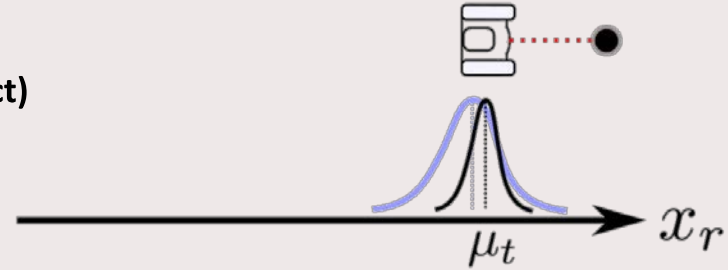
Q: Which of these models is most adequate for the problem we are solving?

Gaussian filtering with features: Extended Kalman filters

Motion model (predict)



Measurement model (correct)



Gaussians

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

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

Univariate

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

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

Multivariate

The data association problem

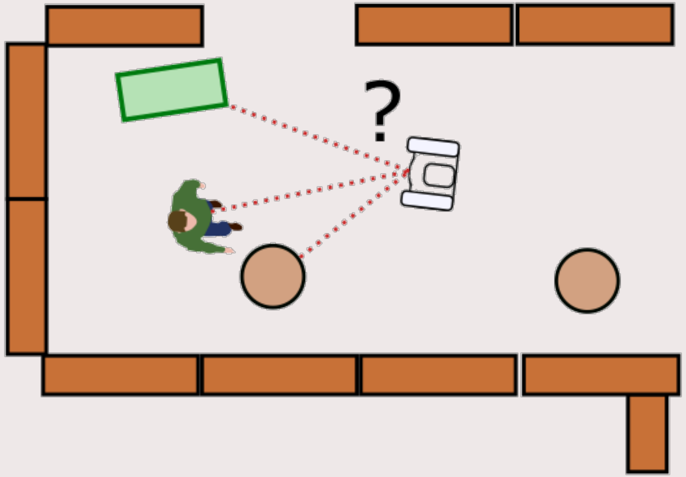
Problem so far: we assumed **known data associations**

Often we can retrieve the correct data association:
- **nearest neighbor**
- **Uncertainty-based** (choose not to make one)

Making a *wrong association* can be a big problem!

Multiple data association **hypotheses** give rise to **multimodal** probabilities!

How can we deal with this?



TU/e

Discrete representation: particle filters

Brute-force implementation of recursive filter

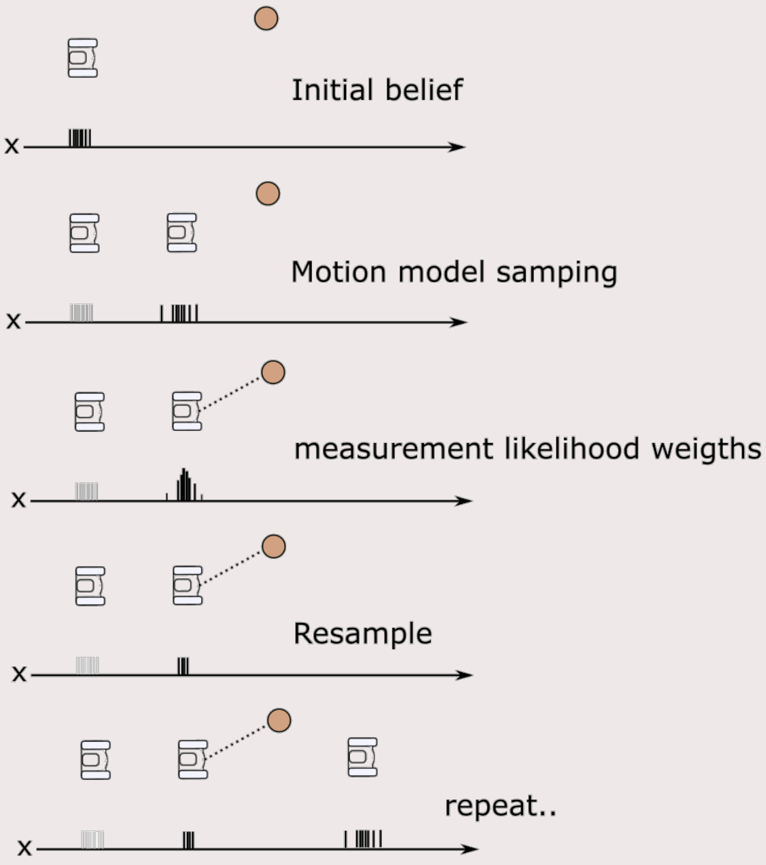
Represents the **belief** as **weighted particles** (often 100+)

Particles are discrete **hypotheses** about the state

Bayesian filter steps

- Particles get propagated according to **motion model**
- Particles get **likelihood weights** based on sensor information
- Requires a **stochastic resampling step** (tuning parameter)
- **Low weight** particles **removed**, **high weight** particles **cloned**

Successful in **low-dimensional** state spaces
Tuning: How many particles? How often resampling?



The right solution for the problem

We challenge you to **abstract the problem** using the right **models**

- Would scan / feature matching be adequate?
- Can continuous representations increase robustness?
- Or are discrete representations better suited?
- How many hypotheses do we need? 2? 500?

- We don’t expect you to implement all possible solutions
- Rather, think about how your robot can be **robust** and **explainable**

References

Elfring, J., Torta, E., Molengraft, M. v. d., (2021) Particle Filters: A Hands-On Tutorial https://www.mdpi.com/1424-8220/21/2/438

Thrun, S., Burgard, W.,, Fox, D. (2005). *robotics*. Cambridge, Mass.: MIT Press. ISBN: 0262201623 9780262201629 *Probabilistic*