Introduction to

Simultaneous Localization and Mapping

Spring Semester Course 2019

CS284

Course Convener:

Prof Laurent Kneip

lkneip@ shanghaitech.edu.cn

Disclaimer

- Some of the lecturing material is naturally taken from publically available online material from other groups. Sources include:
  - Autonomous Systems Lab ETH Zurich
  - Research School of Engineering, ANU
  - Informatik Department, Uni Freiburg
  - MIT
  - …
- By using the present material you confirm that
  - You use it only internally and for the purpose of education
  - You are aware that the lecture material may originate from other sources, even if explicit reference is occasionally missing

Where are we at?

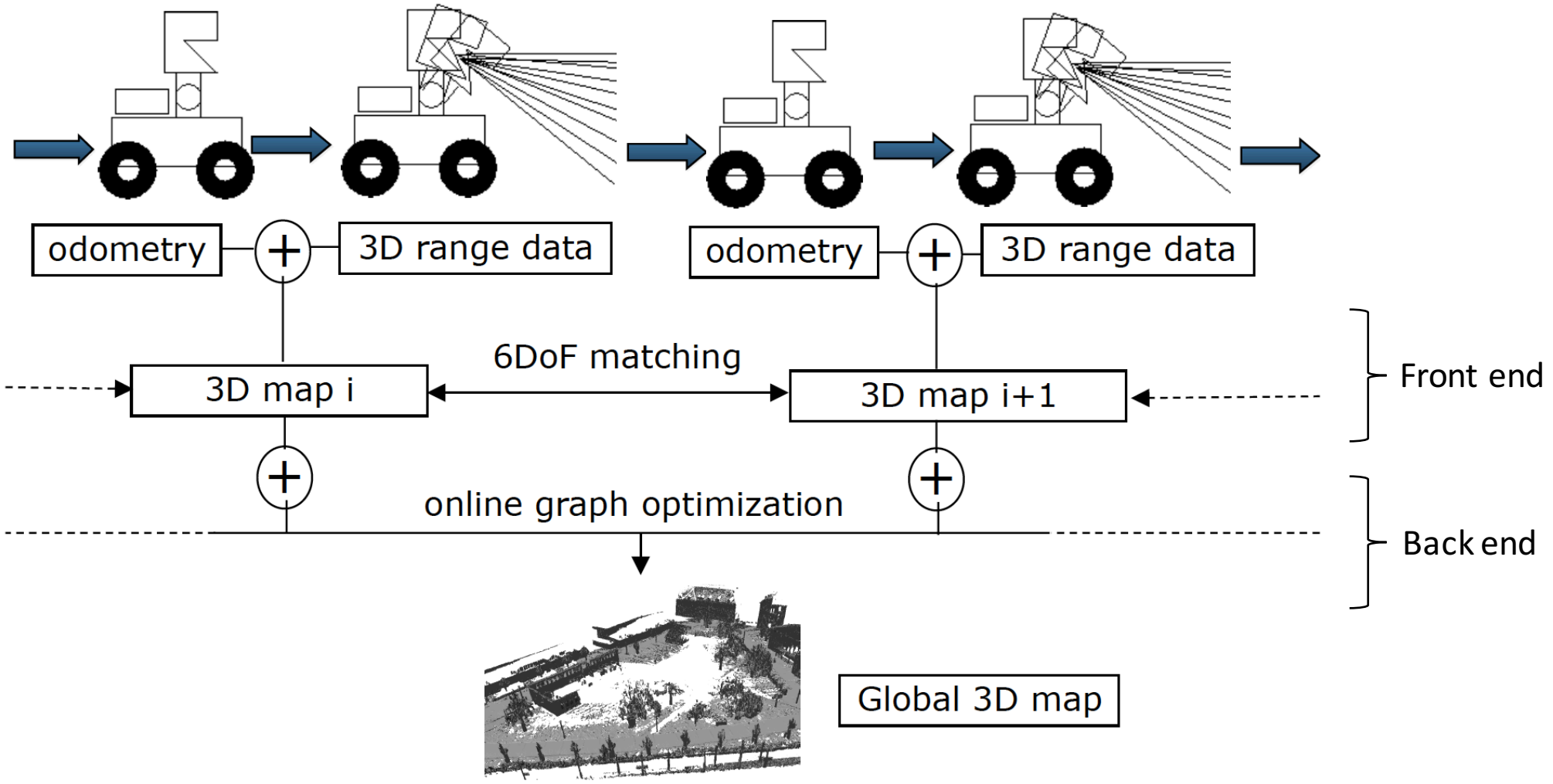
- Today: Front-end choices, SLAM as a graphical optimization problem

Week	Tuesday course	Thursday course
18th February	Introduction (what is SLAM, curriculum, course structure (homeworks, projects), course webpage	Homogeneous coordinates, Transformations, etc.
25th February	Filtering methods (GF, PF)	Filtering methods (GF,PF)
4th March	Pose-only SLAM, Graph SLAM	Introduction to the projects
11th March	Camera as a sensor, camera calibration	Introduction into Visual SLAM, feature extraction, tracking, and matching
18th March	Feature extraction, tracking, and matching	Case study: MonoSLAM (why filter?)
25th March	Bootstrapping: The Relative Pose Problem	Full visual odometry pipeline, Non-linear optimization, Bundle adjustment
1st April	Non-linear optimization, Bundle adjustment	Place recognition, loop closure
8th April	Place recognition, loop closure, the absolute pose problem	Case study: ORB SLAM
15th April	Dense Tracking and Mapping (DTAM), Photometric vs geometric errors, semi-dense opt.	RGBD SLAM, Iterative closest points (Laser-point cloud registration), TSDF
22nd April	SLAM with Stereo and Multi-Perspective cameras	Midterm exam
29th April	Visual-Inertial SLAM	Project discussions/buffer
6th May	Dynamic and Multi-body SLAM	Project discussions/buffer
13th May	Semantic SLAM	Project Presentations

Starting point

- Robot providing odometry
- Equipped with
  - Planar Laser scanner
  - Pitching planar Laser scanner
  - Velodyne
  - …

Starting point



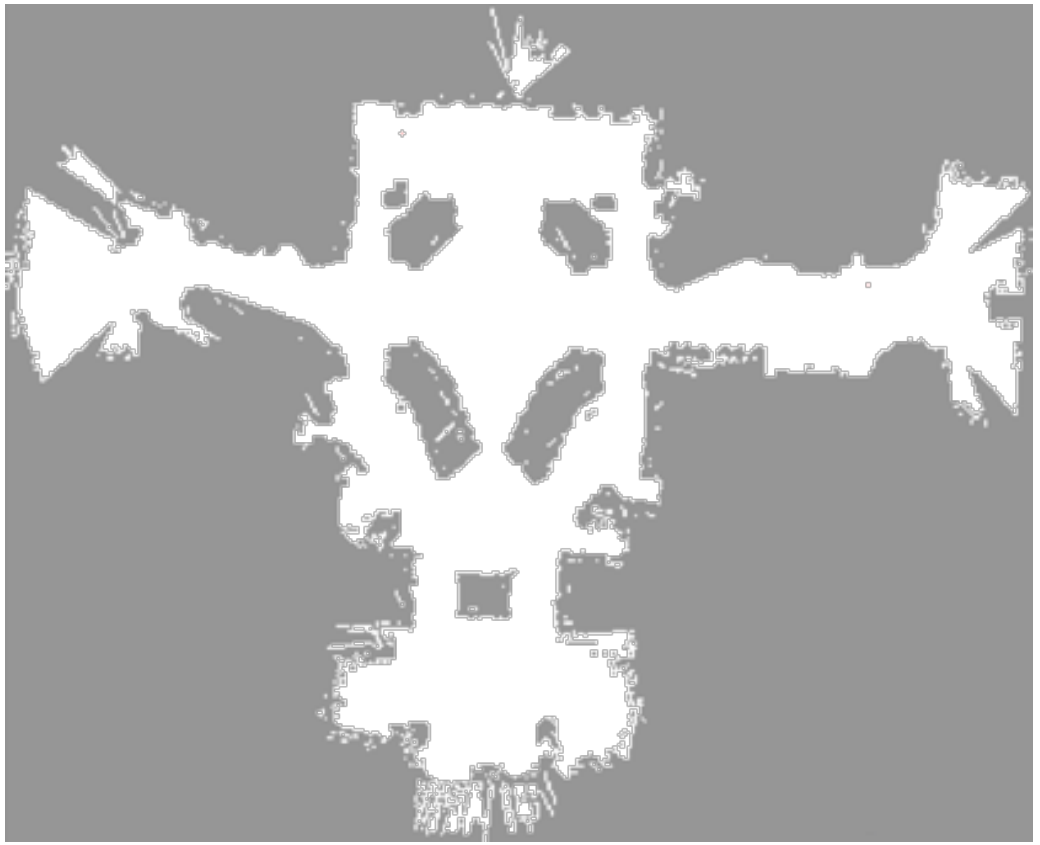
Starting point

- Simple proposal for a SLAM algorithm:

```
while (robot.isMoving()) {
        S_i = getNewScan();
        T_{i-1,i} = ICP(S_{i-1},S_i);          //find relative pose
        T_{W,i} = T_{W,i-1} T_{i-1,i};         //update absolute pose
        M = [M   T_{W,i}S_i];                  //extend map by new points
}
```

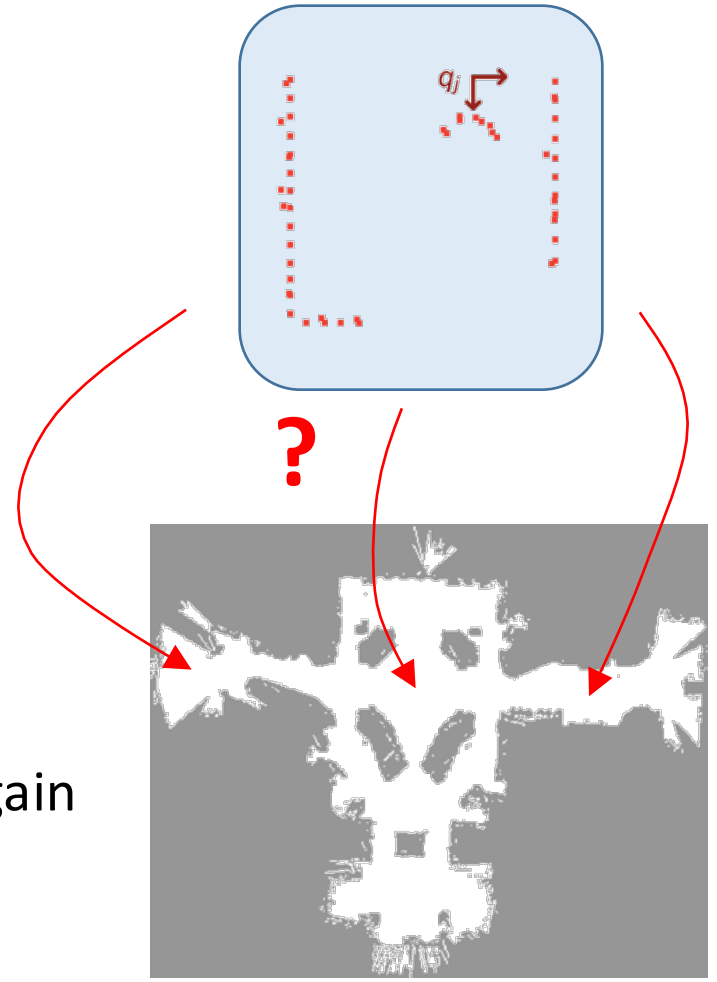
Starting point

- Result: A map of the environment
  - Let’s still assume this to be of very quality for the moment



The global localization problem

- Problem formulation
  - Given: a new scan and an existing map of the environment
  - Find: robot position and orientation
- Synonyms:
  - Robot kidnapping problem
  - Absolute pose estimation
  - …
- Relevant in the following scenarios:
  - Robot is switched off, then moved, and then switched on again
  - Map captured by one robot is reused by another/new one
  - Tracking fails but robot still moves



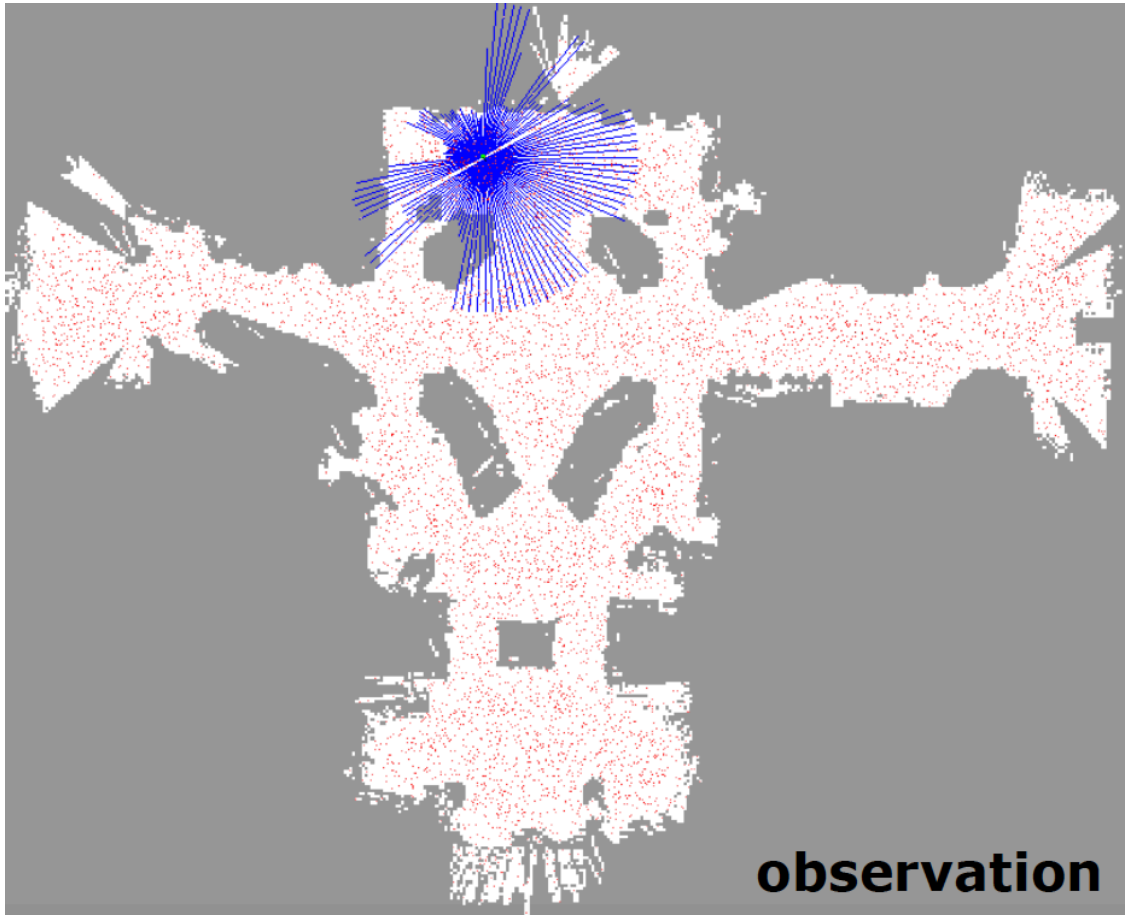
Global localization: initial pose?

- Since we already have a map, the initial pose cannot be initialized to zero translation and identity rotation
- The initial position can be “anywhere”
- The robot is moving and capturing measurements over time

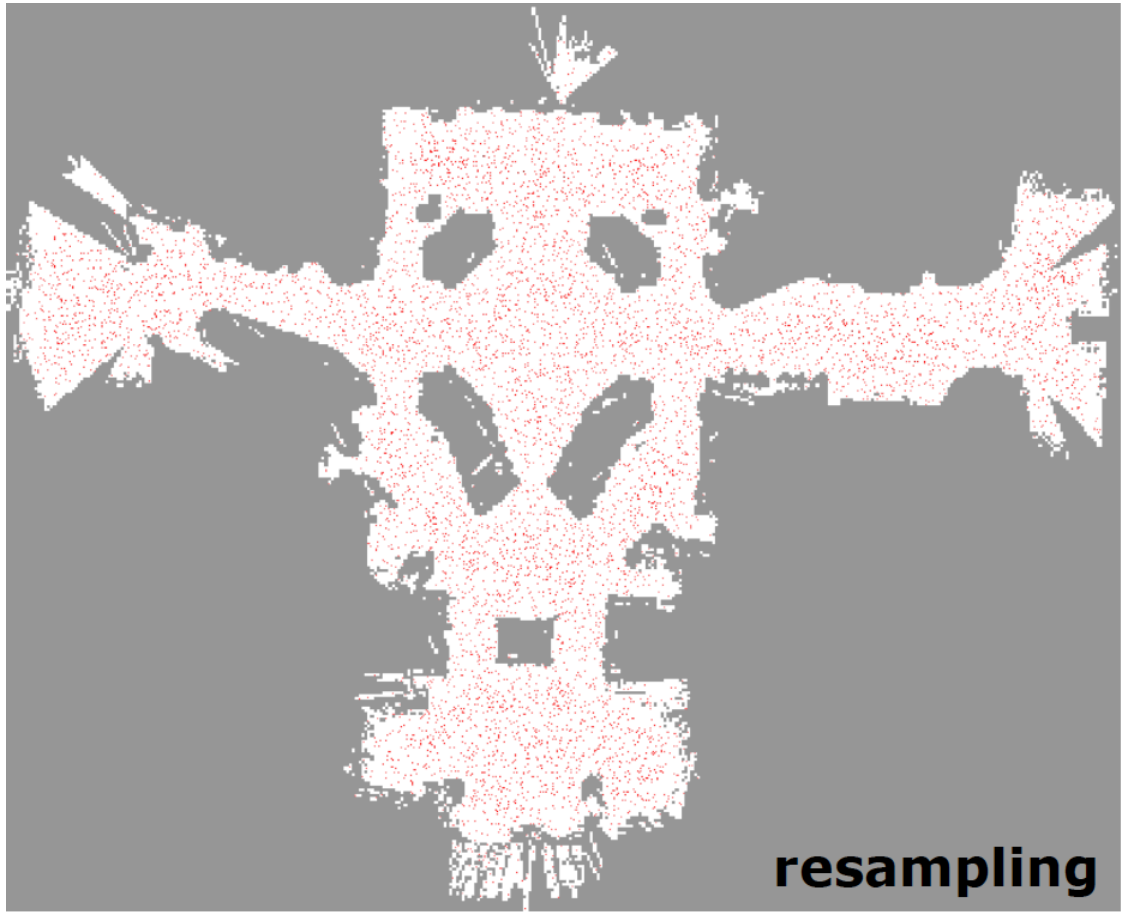
- → Need probability density distribution
- → Need filter for estimating this distribution
  - Filters perform incremental estimation of complete probability density distribution

Can be anywhere → Complicated distribution!
Use particle set/filter!

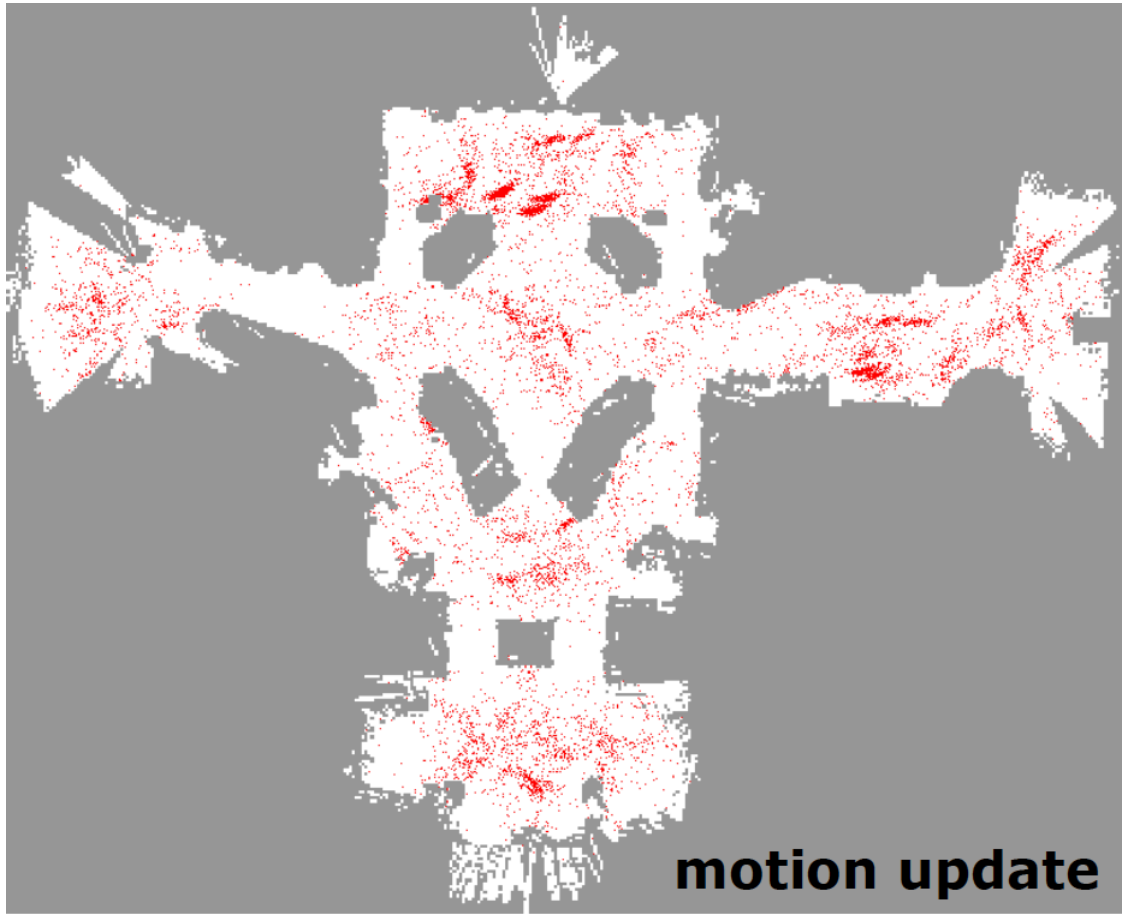
Particle Filter Example



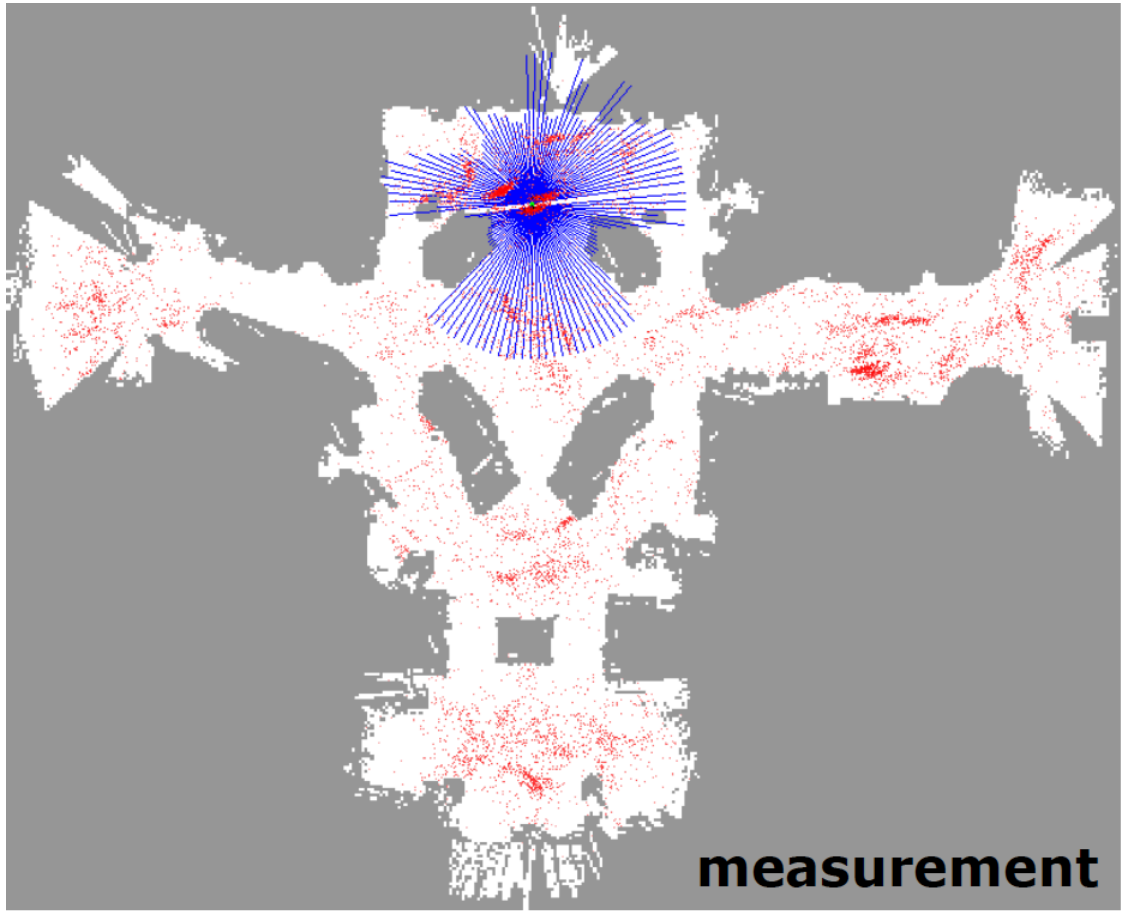
Particle Filter Example



Particle Filter Example

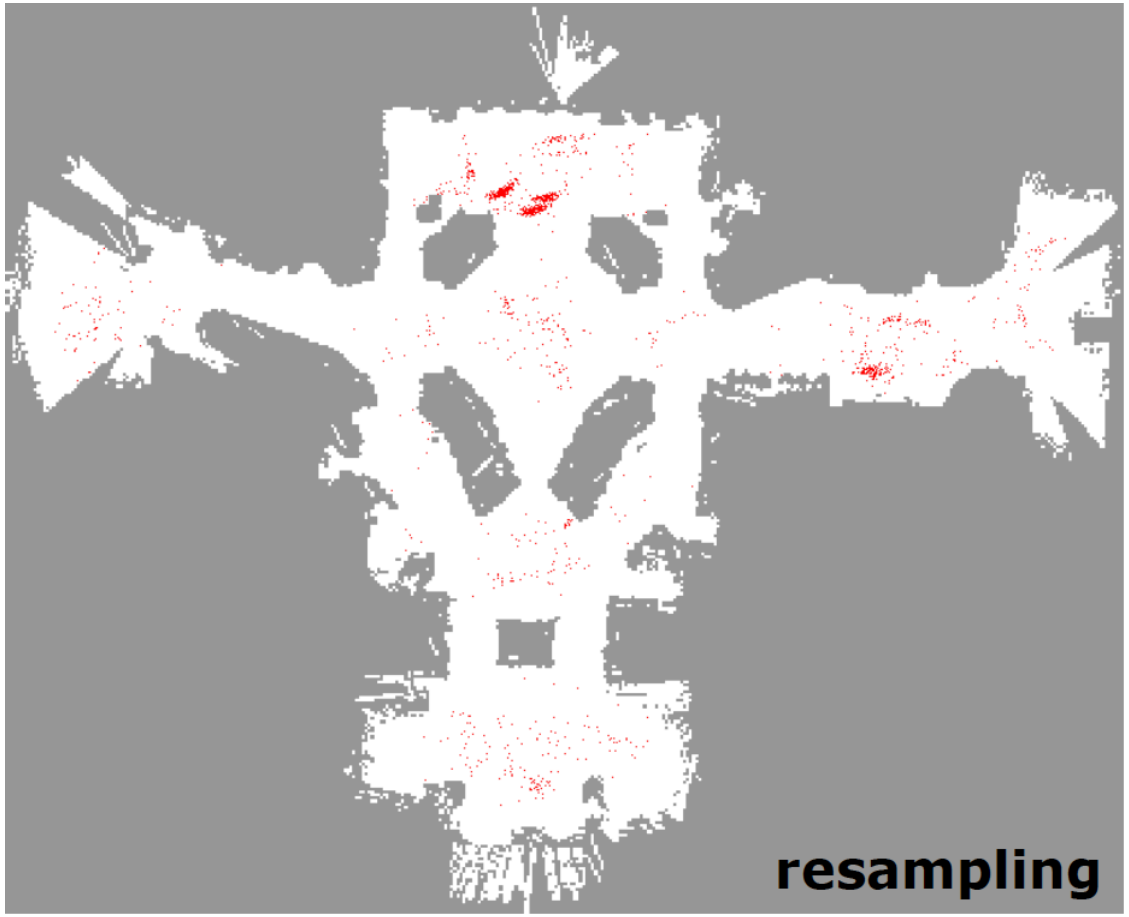


Particle Filter Example

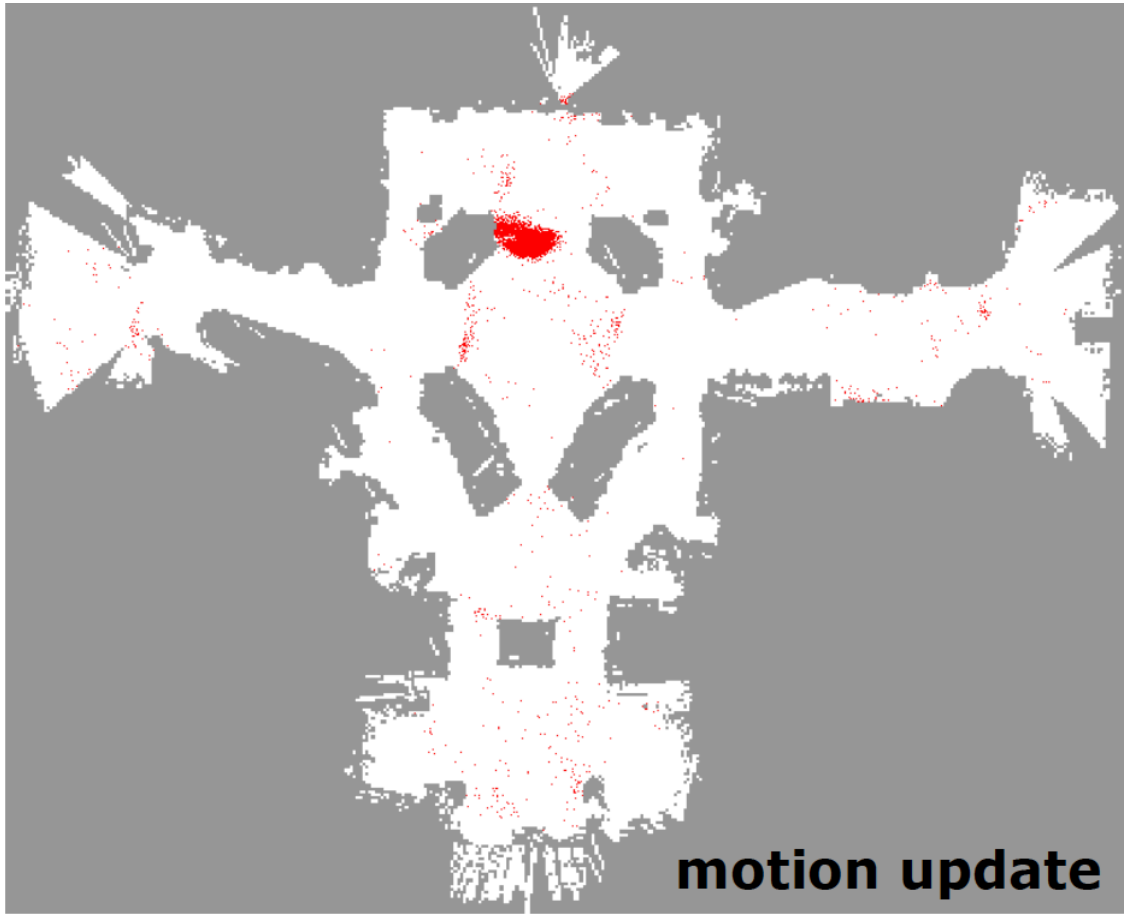


Particle Filter Example

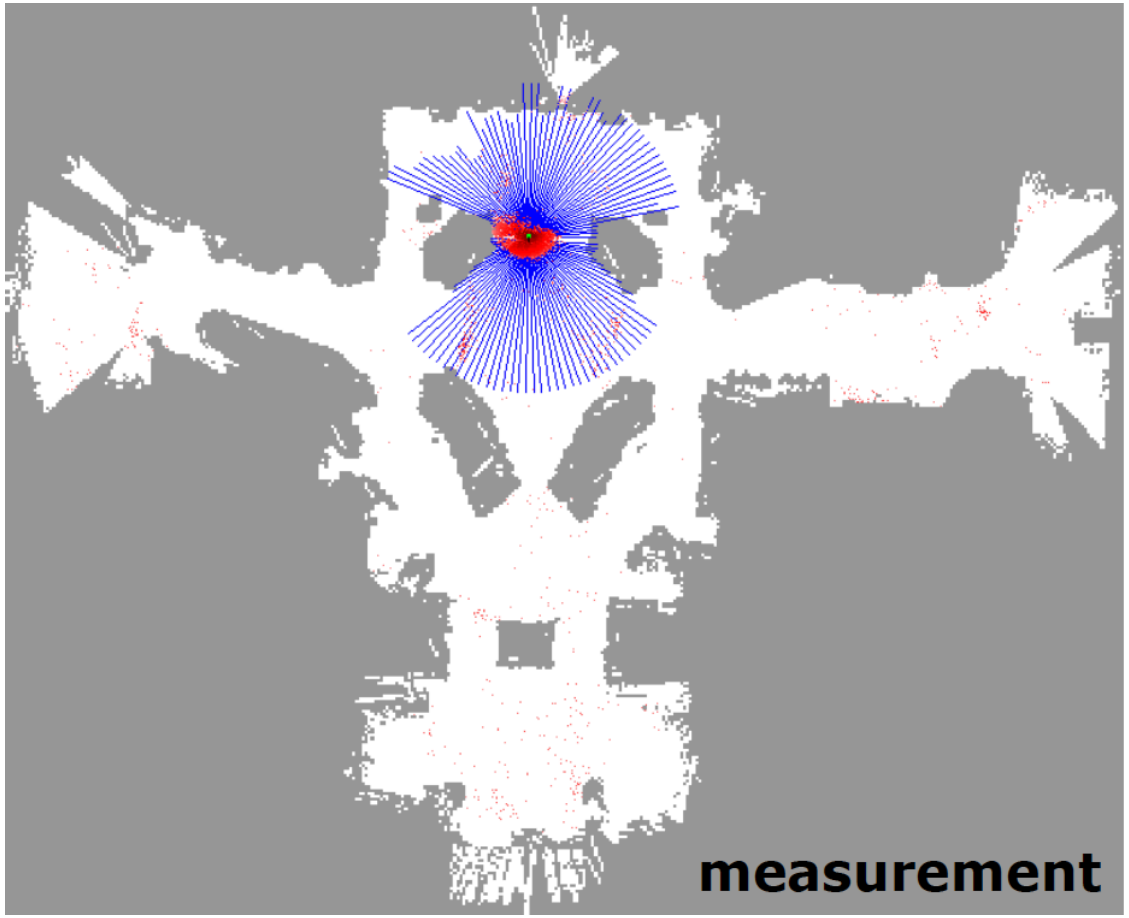
Particle Filter Example



Particle Filter Example

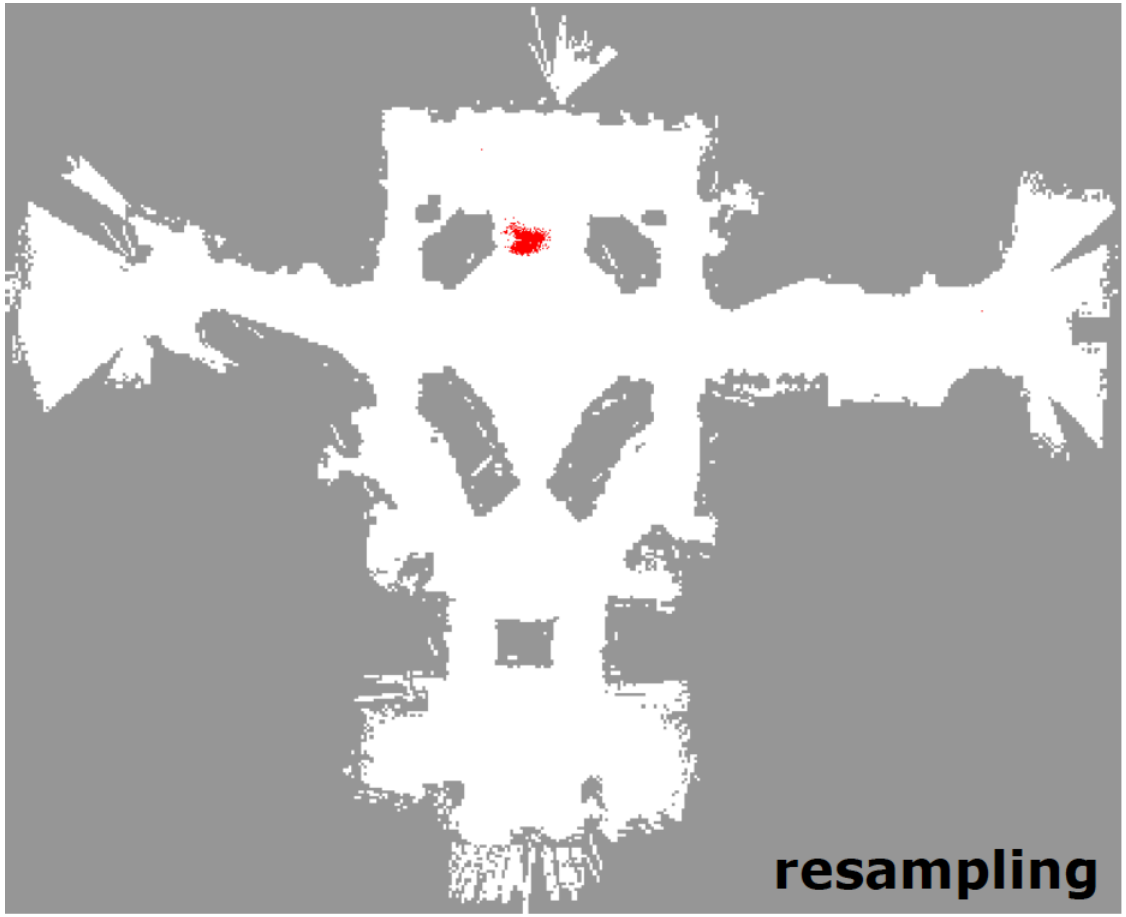


Particle Filter Example

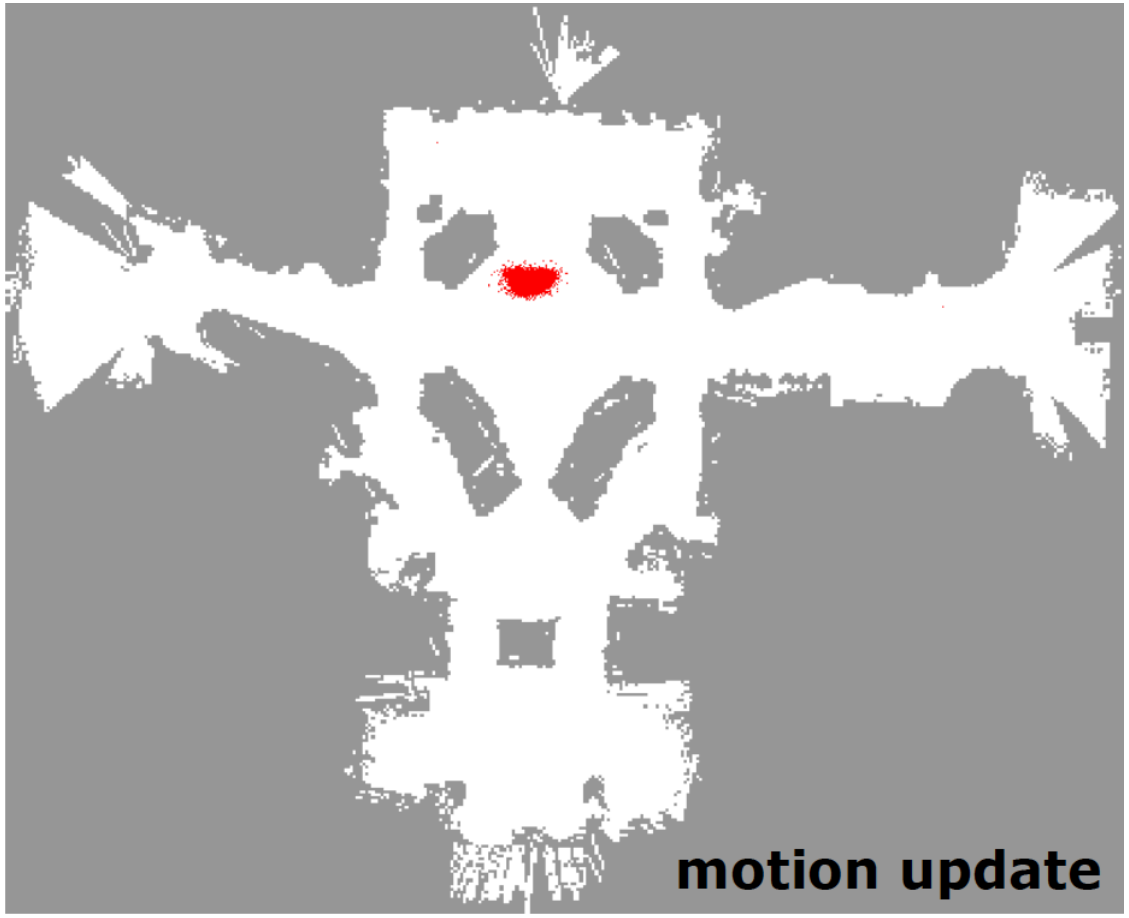


Particle Filter Example

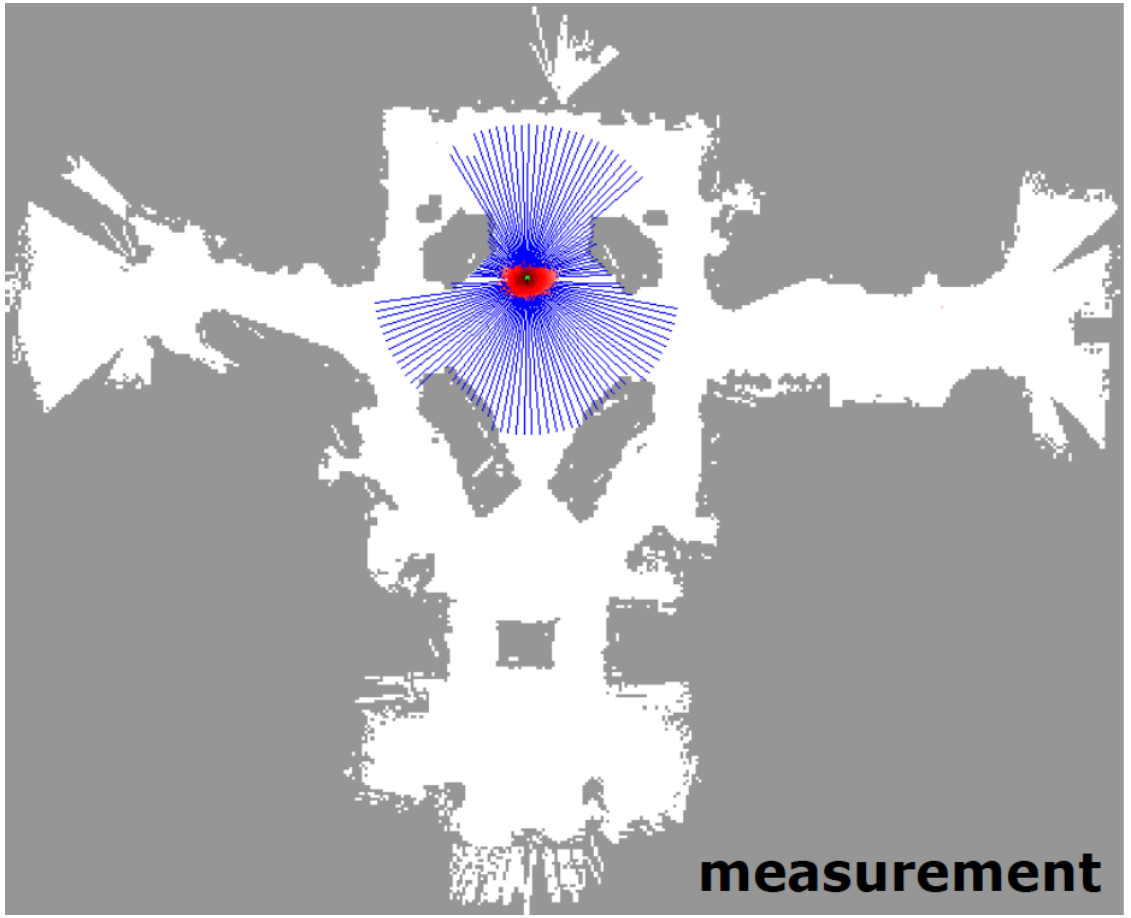
Particle Filter Example



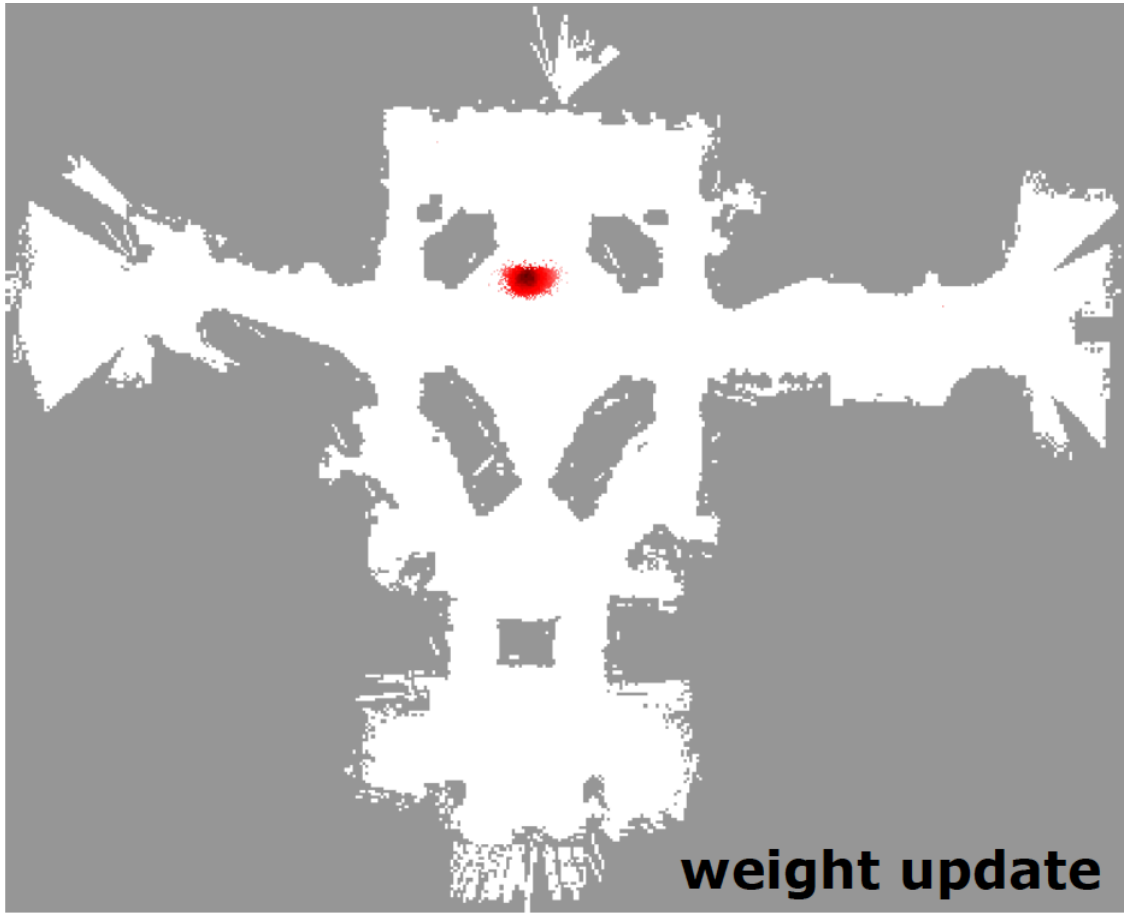
Particle Filter Example



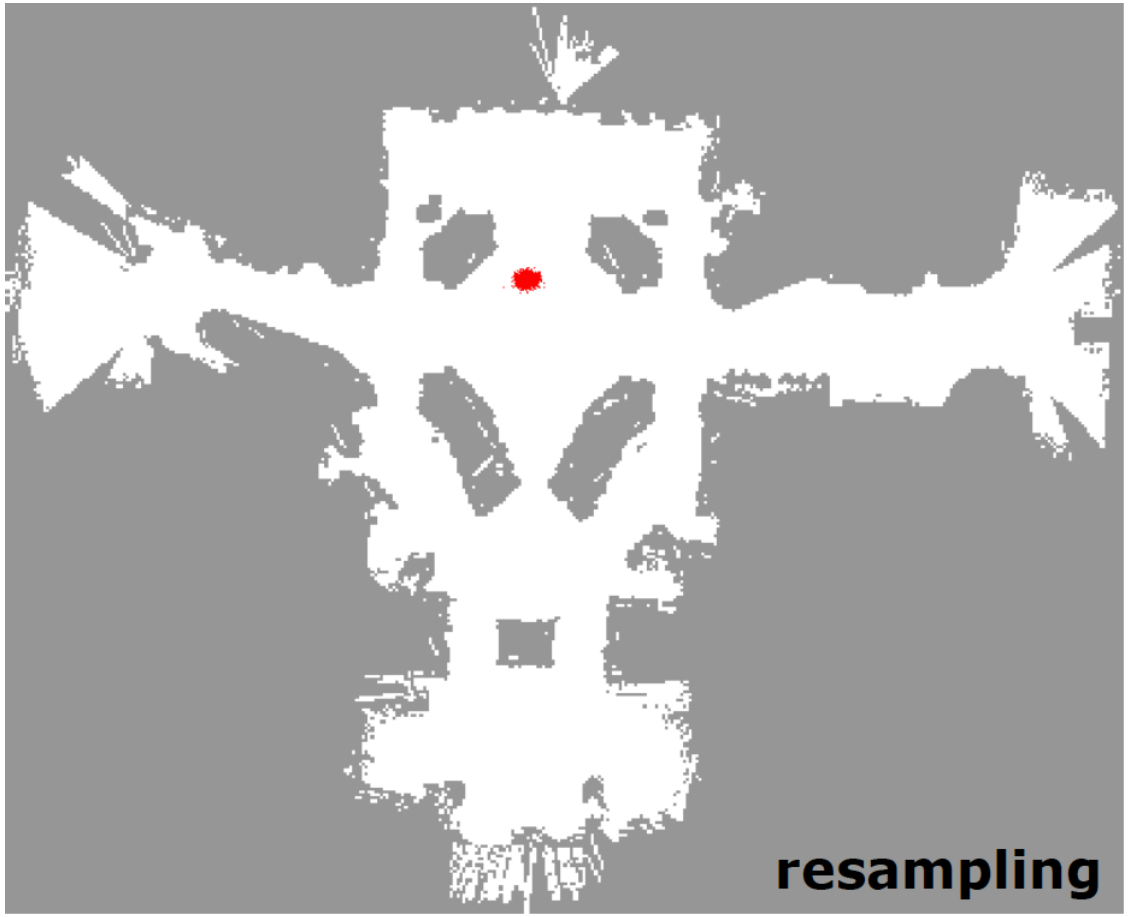
Particle Filter Example



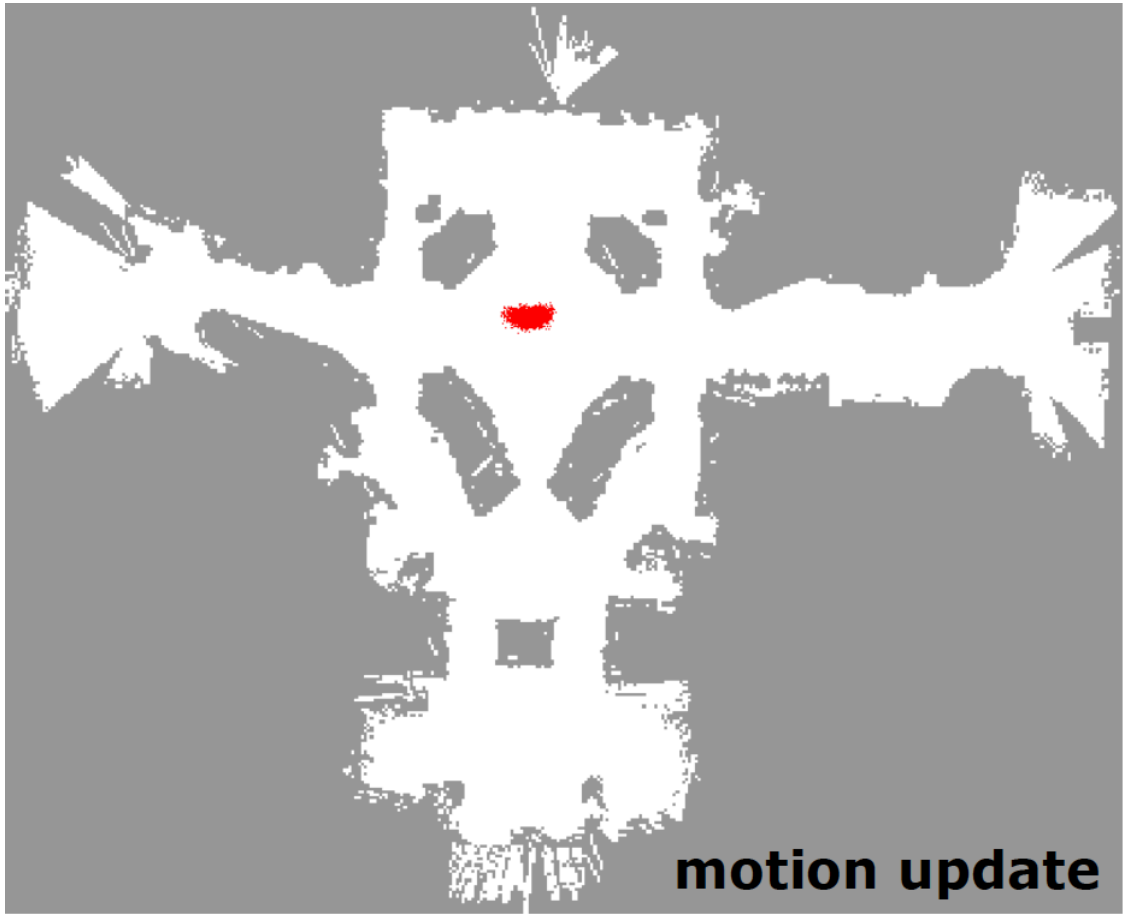
Particle Filter Example



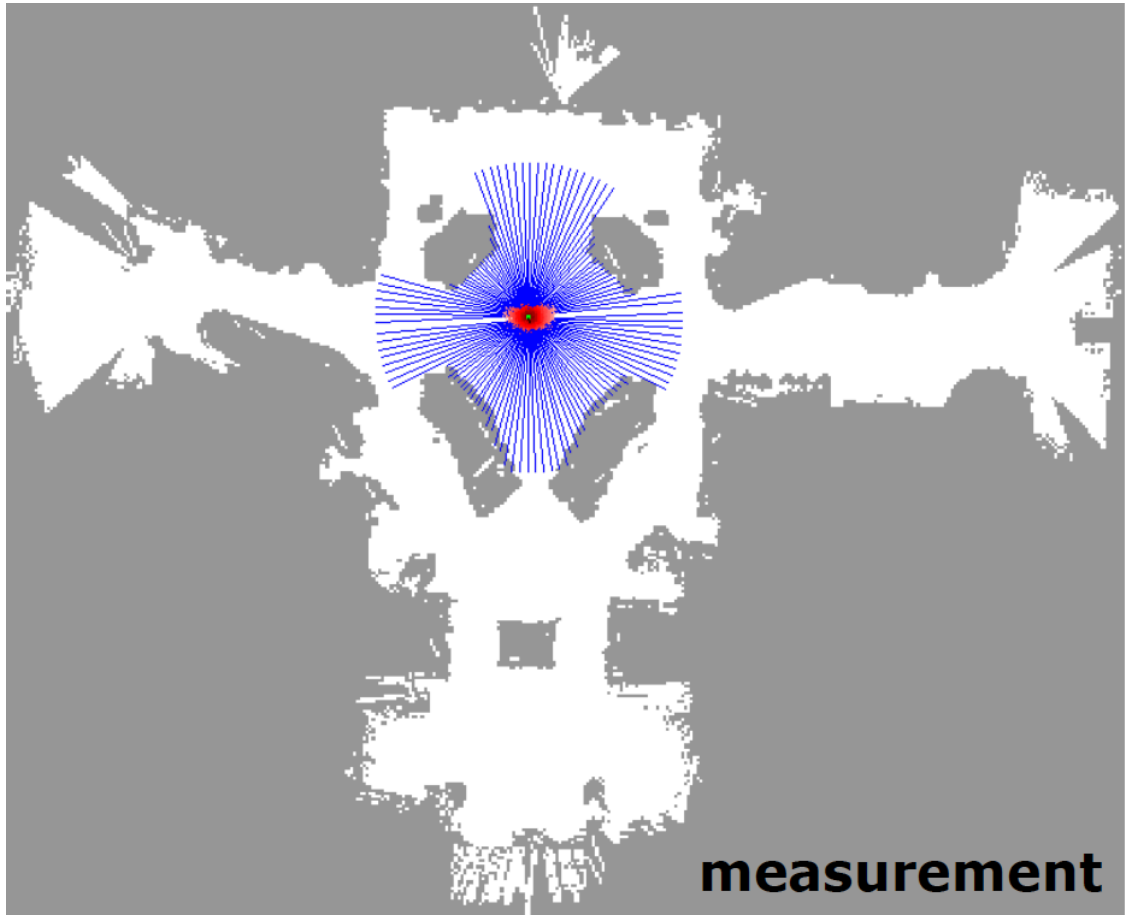
Particle Filter Example



Particle Filter Example

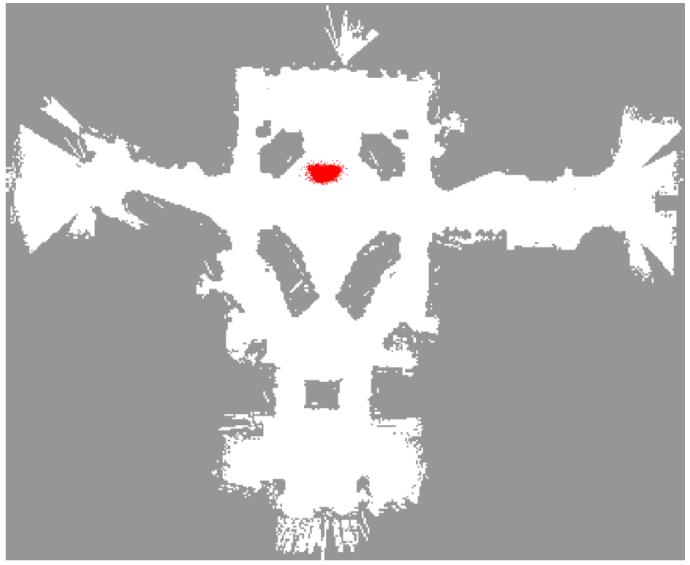


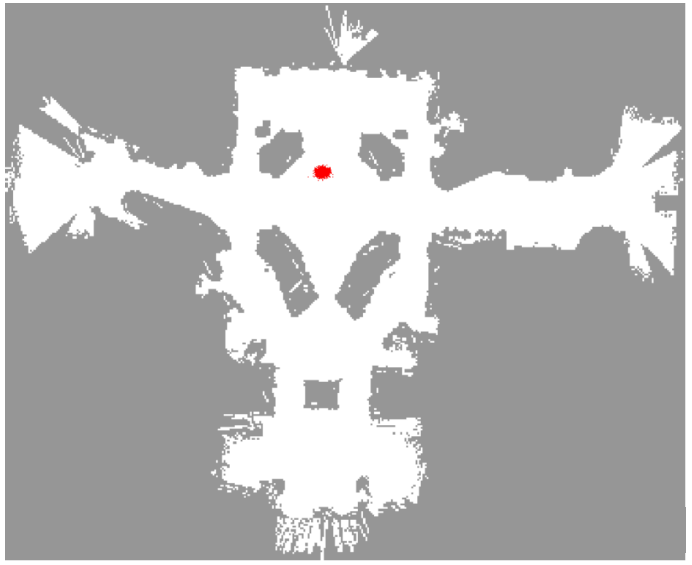
Particle Filter Example

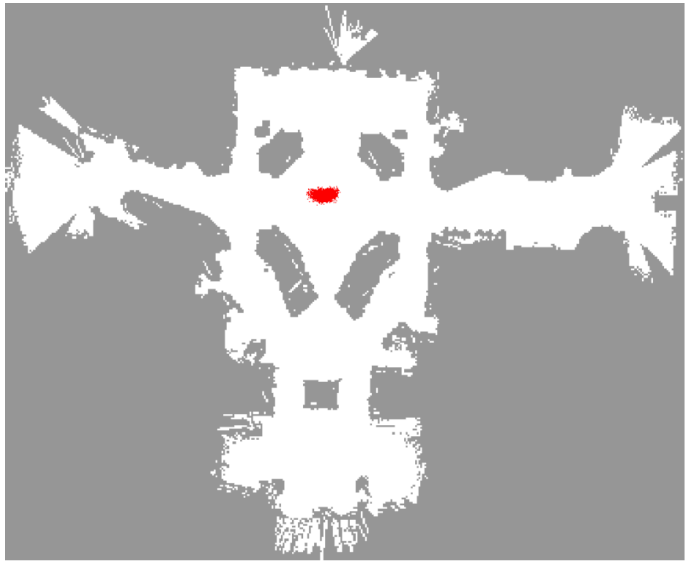


What happens in later iterations?

- Particle set is “condensed”/”focused”
- Distribution is
  - is much less complicated
  - can be approximated by local a Gaussian
- Still a global localization problem, but with “good” priors

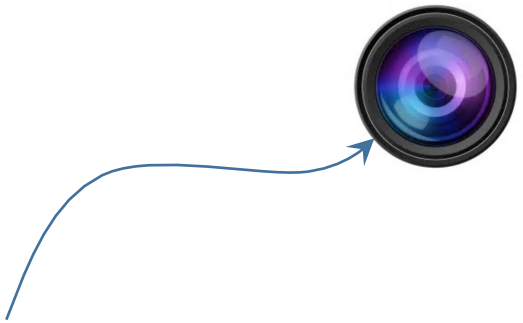






Local vs Global Localization

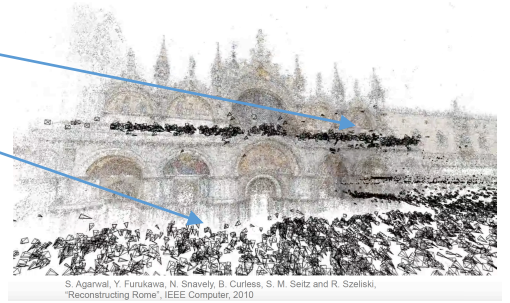
- Local localization
  - We only track the pose with respect to the previous pose
  - A relative problem
  - Requires only sensor tracking
  - Drifts over time



- Global localization
  - We want to know where we are w.r.t. all prior poses/the map
  - An absolute problem
  - Drift-free under the assumption of a sufficiently accurate map

With ”good”/”focussed” prior

?



S. Agarwal, Y. Furukawa, N. Snavely, B. Curless, S. M. Seitz and R. Szeliski, "Reconstructing Rome", IEEE Computer, 2010

A simple solution

- Just do ICP between the local scan and the entire map
- Why not?
  - Does not take any motion priors into account
  - Does not respect statistical aspects
  - Does not guarantee the motion is smooth

- In practice:
  - Still want to use a filter to assure the utilization of control inputs and the enforcement of temporal smoothness

Different filter realizations

- The Bayes filter is a framework for recursive state estimation
- There are different realizations
- Different properties
  - Linear vs. non-linear models for motion and observation models
  - Gaussian distributions only?
  - Parametric vs. non-parametric filters
  - …
- Examples:
  - Gaussian filters (KF, EKF, for linear and linearized models)
  - Particle filter (non-parametric, for arbitrary models, requires sampling)

Filter review: Recursive Bayes filter

- Problem definition:
  - Estimate the state x of a system given observations z and controls u (Note: x groups ego-motion and structure related unknowns)
  - Find

$$p(x \mid z, u)$$

Recursive Bayes filter

$$
\begin{aligned}
bel(x_t) &= p(x_t \mid z_{1:t}, u_{1:t}) && \leftarrow \text{Definition of belief} \\
&= \eta\, p(z_t \mid x_t, z_{1:t-1}, u_{1:t})\; p(x_t \mid z_{1:t-1}, u_{1:t}) && \leftarrow \text{Bayes' rule} \\
&= \eta\, p(z_t \mid x_t)\; p(x_t \mid z_{1:t-1}, u_{1:t}) && \leftarrow \text{Markov assumption} \\
&= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) \\
&\qquad\qquad p(x_{t-1} \mid z_{1:t-1}, u_{1:t})\; dx_{t-1} && \leftarrow \text{Law of total probability} \\
&= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\; p(x_{t-1} \mid z_{1:t-1}, u_{1:t})\; dx_{t-1} && \leftarrow \text{Markov assumption} \\
&= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\; p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1})\; dx_{t-1} && \leftarrow \text{Markov assumption} \\
&= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\; \underline{bel(x_{t-1})}\; dx_{t-1}
\end{aligned}
$$

Recursive term

Prediction and Correction steps

- Bayes filter can be written as two-step process
- Prediction step (motion model)

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

- Correction step (sensor, measurement, observation model)

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

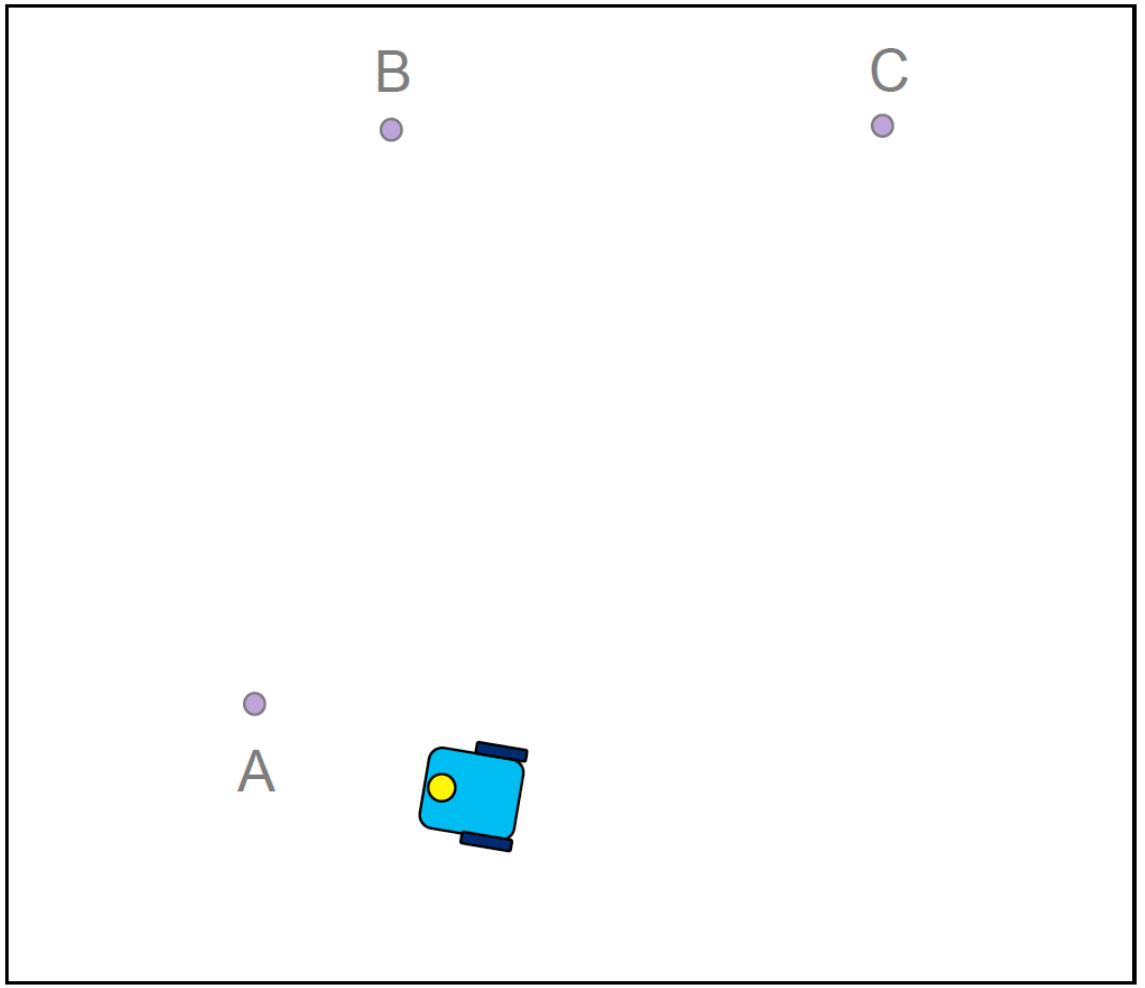
Filtering methods

- **Gaussian Filtering** (EKF, UKF)
  - Tracks a Gaussian belief of the state/landmarks
  - Assumes all noise is Gaussian
  - Follows the “predict/measure/update” approach
- Pros
  - Can run online
  - Works well for problems experiencing expected perturbations/uncertainty
- Cons
  - Unimodal estimate
  - States must be well approximated by a Gaussian
  - The vanilla implementation does not scale very well with larger maps

Gaussian filtering

- Use internal representations for
  - the positions of landmarks (also called “map”)
  - the camera pose

- Assumption:
  - Agent’s uncertainty at starting point is zero

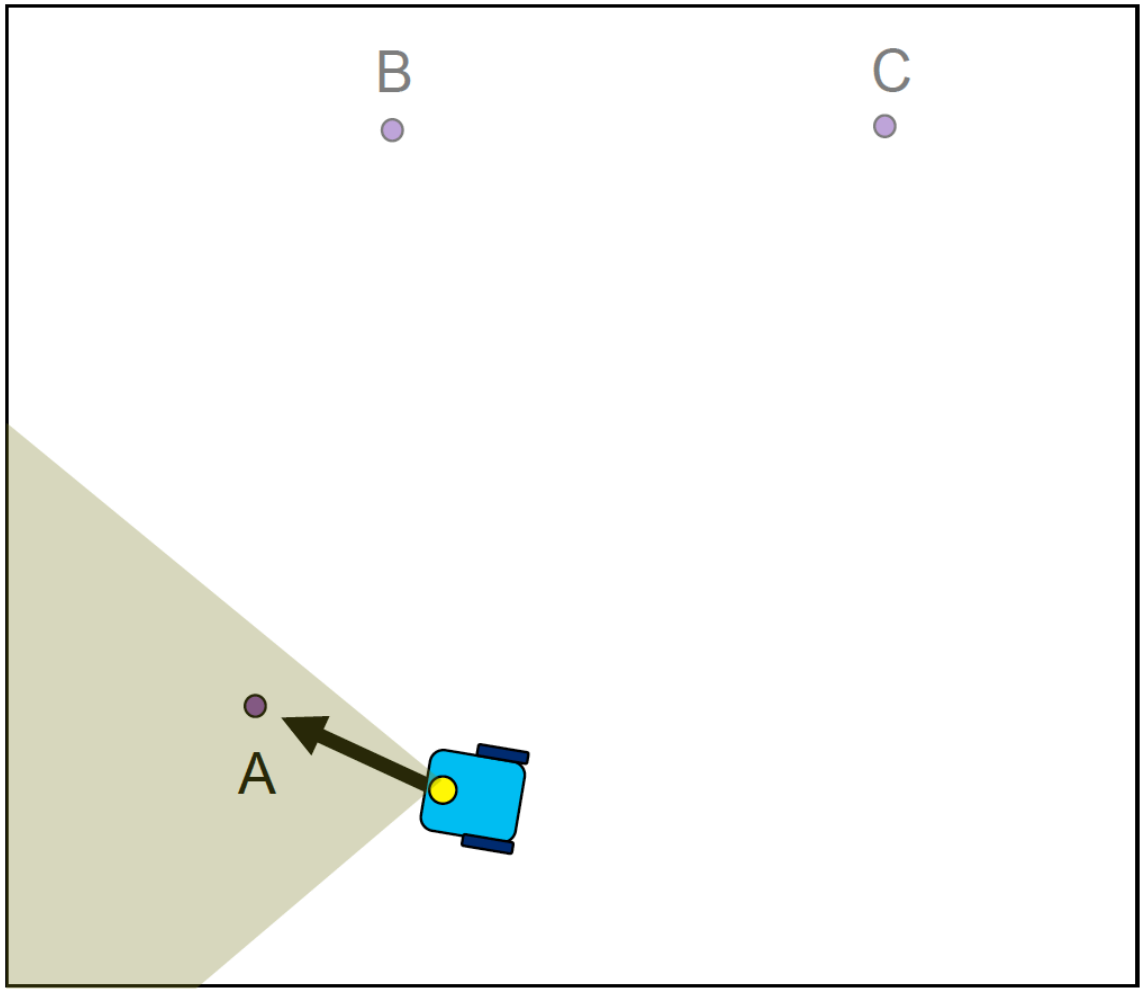


Start: Agent has zero uncertainty

Gaussian filtering

For every frame:

- **Predict** how body has moved
- **Measure**
- **Update** internal representation



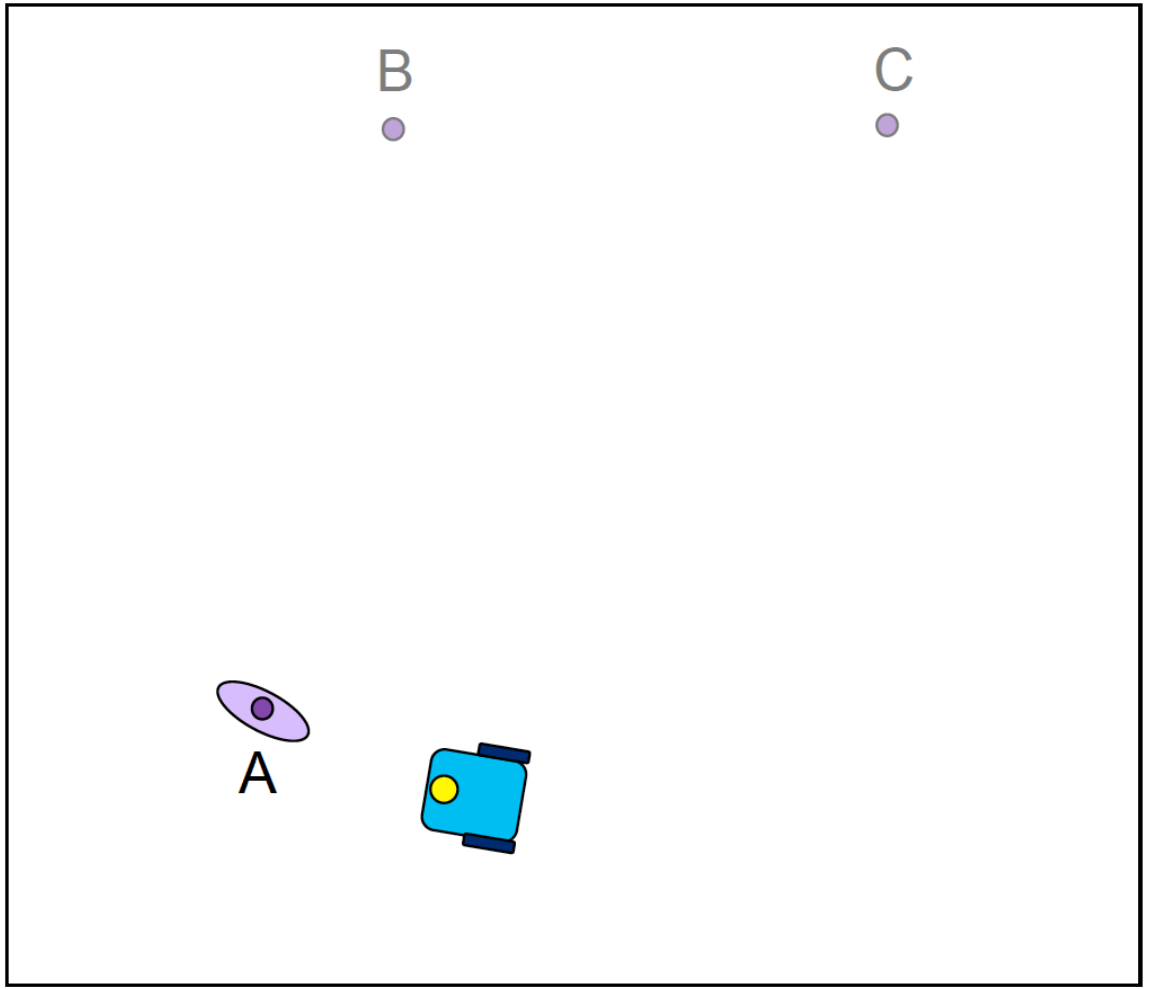
First measurement of feature A

Gaussian filtering

- The agent observes a feature which is mapped with an uncertainty related to the **measurement model**

For every frame:

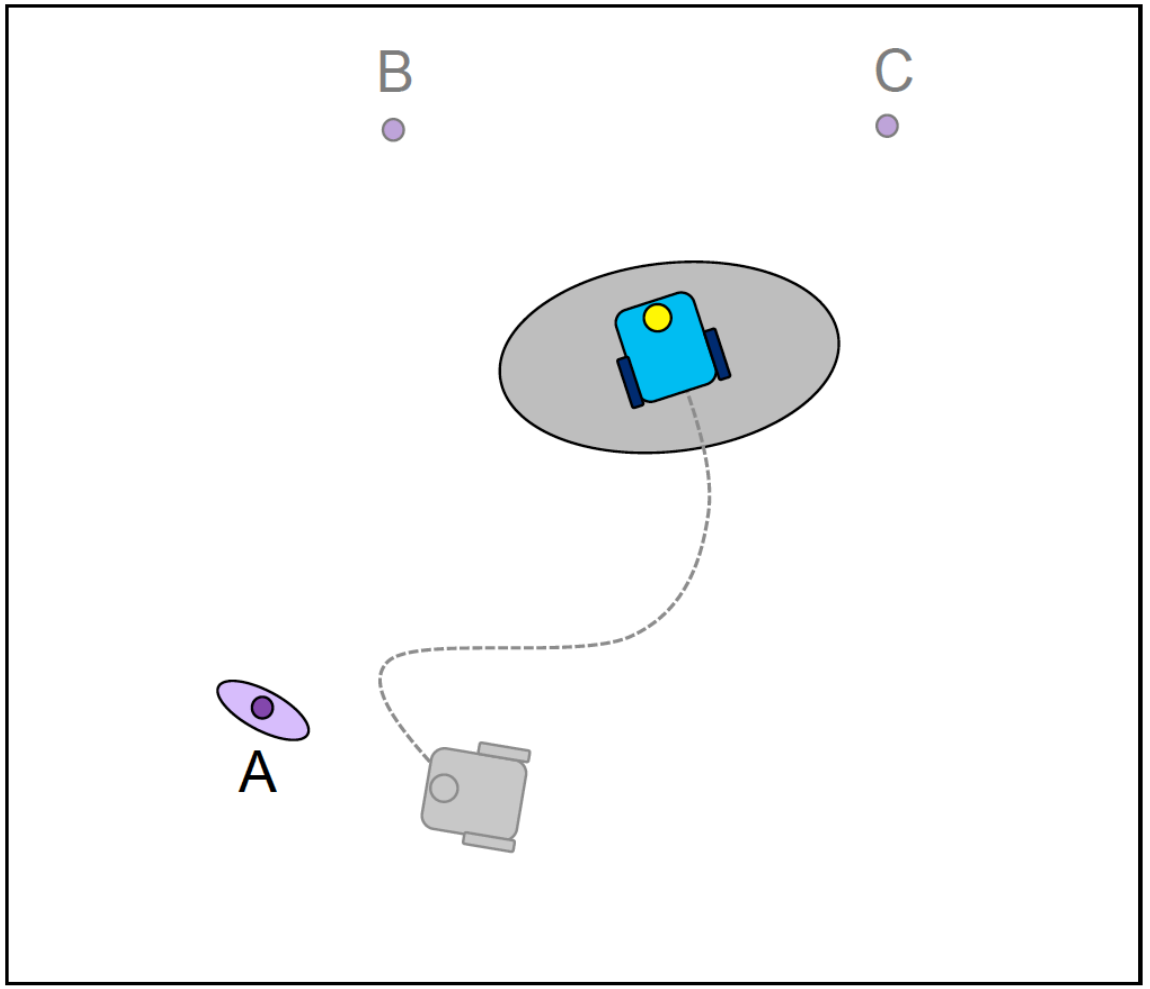
- **Predict** how body has moved
- **Measure**
- **Update** internal representation



Gaussian filtering

- As the agent moves, its pose uncertainty increases, obeying the agent's **motion model**

For every frame:
- **Predict** how body has moved
- **Measure**
- **Update** internal representation



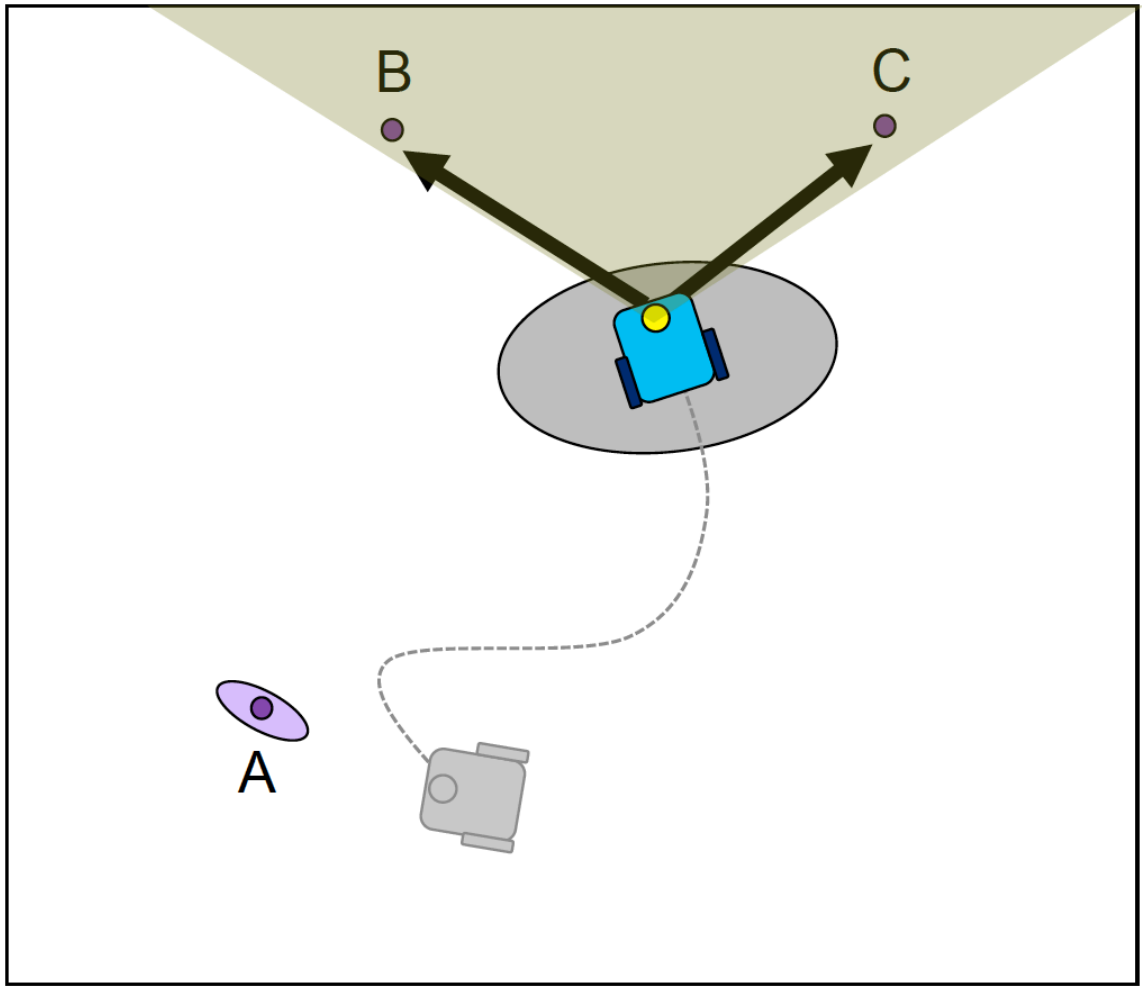
Agent moves forward: uncertainty grows

Gaussian filtering

- Agent observes two new features

For every frame:

- **Predict** how body has moved
- **Measure**
- **Update** internal representation

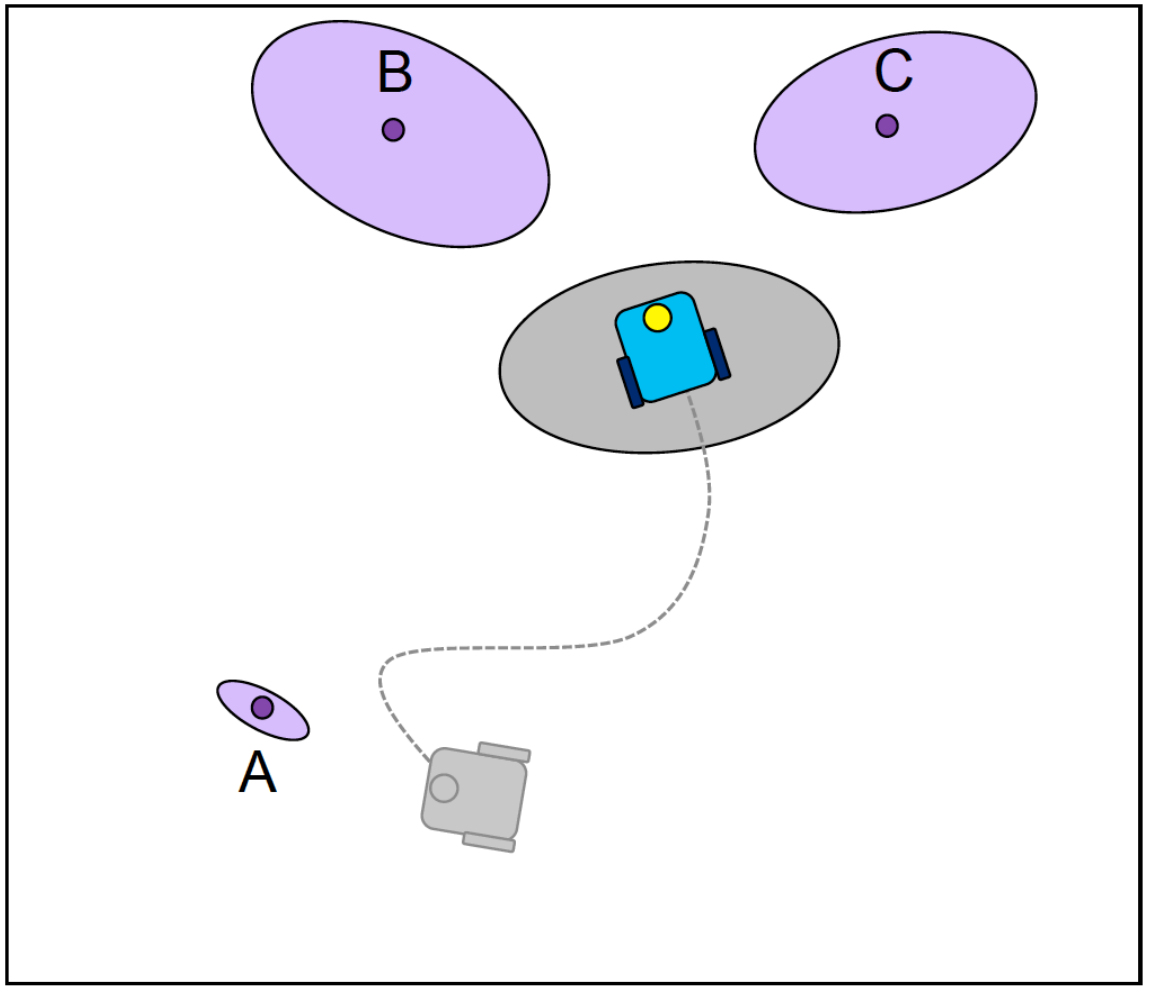


Agent makes first measurement of B & C

Gaussian filtering

- Their position uncertainty results from the **combination** of the measurement error with the body pose uncertainty
  → Map becomes **correlated** with the body pose uncertainty

For every frame:
- **Predict** how body has moved
- **Measure**
- **Update** internal representation



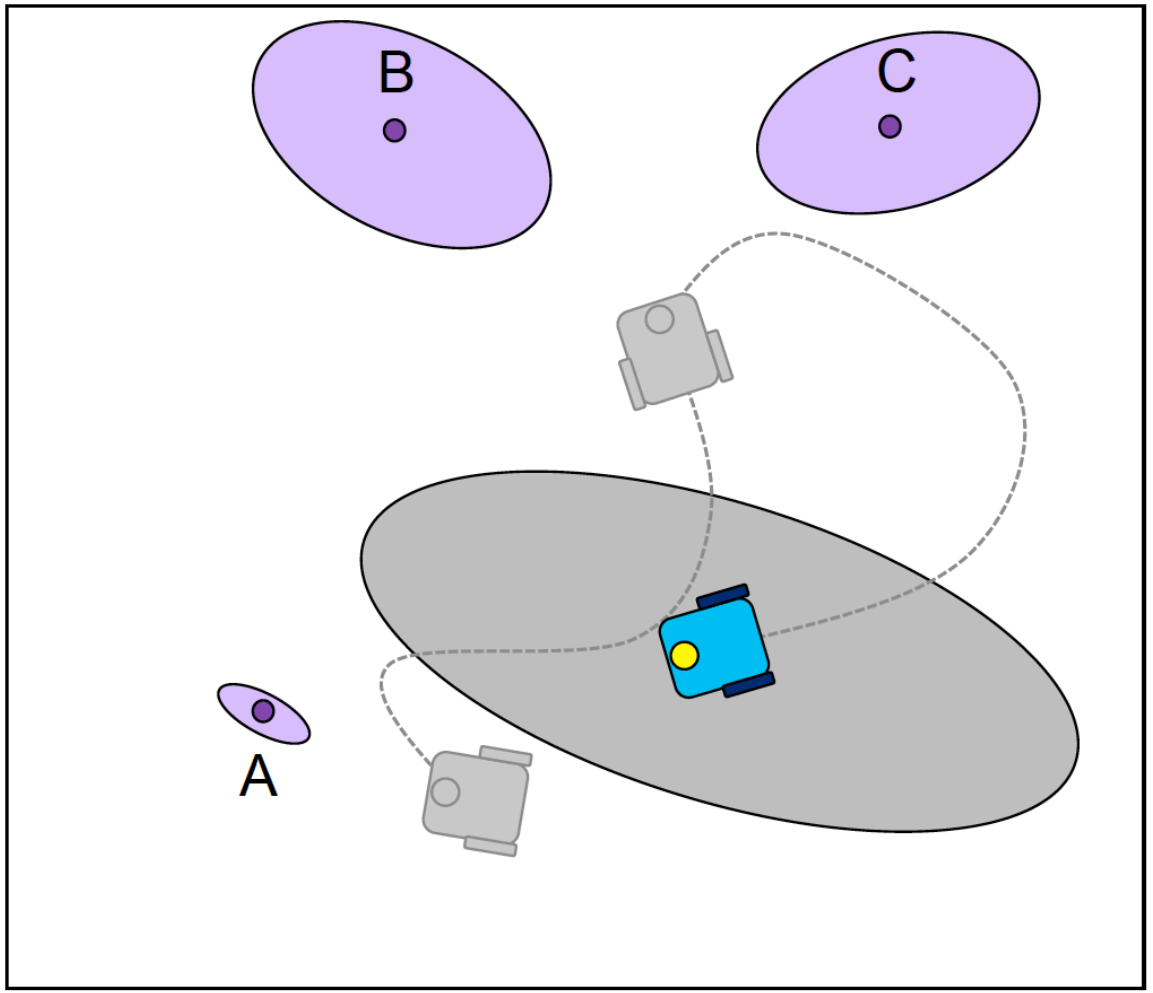
Agent makes first measurement of B & C

Gaussian filtering

- Agent moves again, and its uncertainty increases (motion model)

For every frame:

- **Predict** how body has moved
- **Measure**
- **Update** internal representation



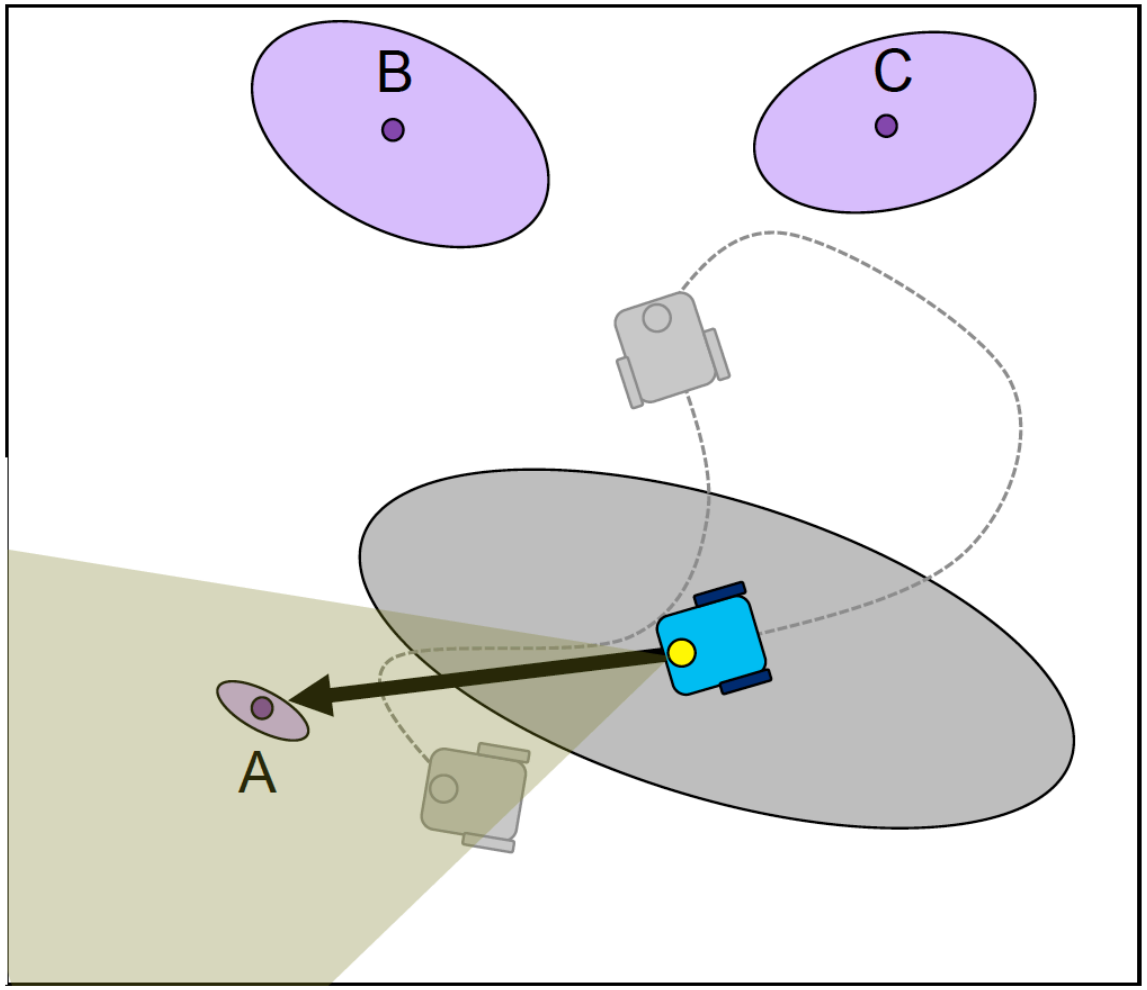
Agent moves again: uncertainty grows more

Gaussian filtering

- Agent re-observes an old feature
  → **Loop closure** detection

For every frame:

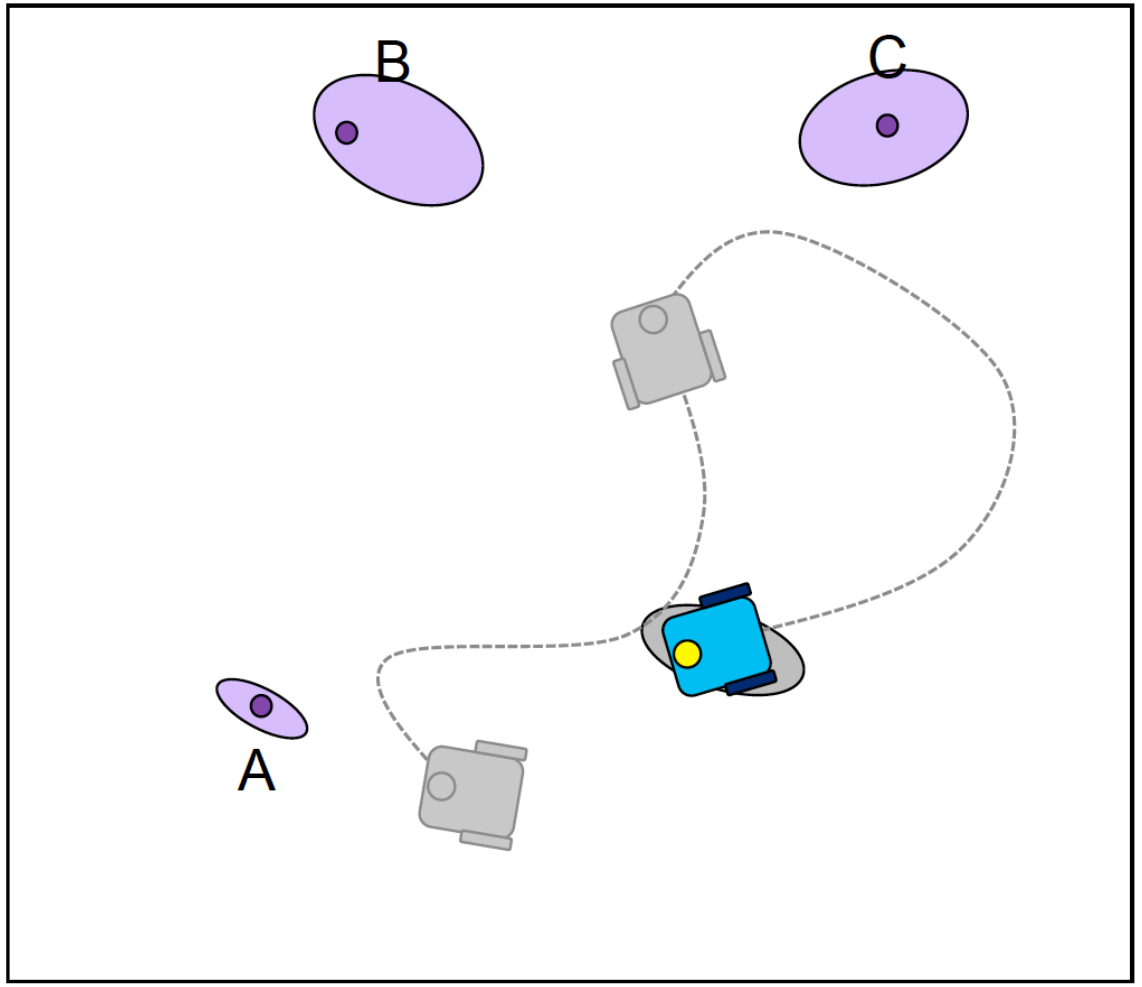
- **Predict** how body has moved
- **Measure**
- **Update** internal representation



Gaussian filtering

- Agent updates its position: resulting pose estimate becomes **correlated** with the feature location estimates
- Agent's uncertainty shrinks and so does the uncertainty in the rest of the map (due to correlation)

For every frame:
- **Predict** how body has moved
- **Measure**
- **Update** internal representation



Agent "re-measures" A: loop closure
→ Uncertainty shrinks

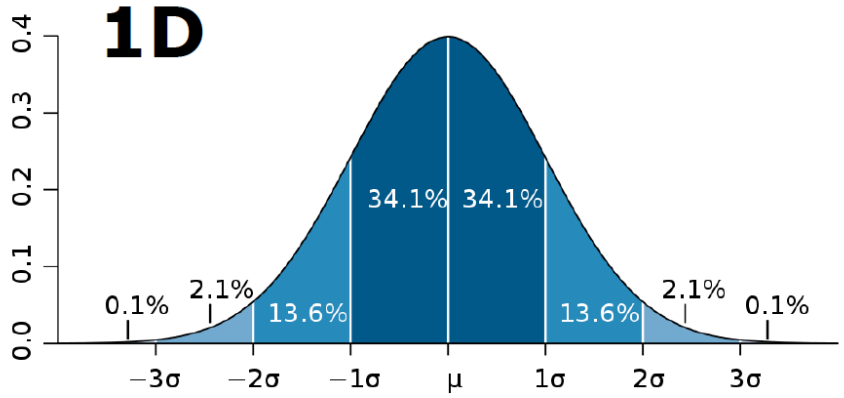
Kalman filter

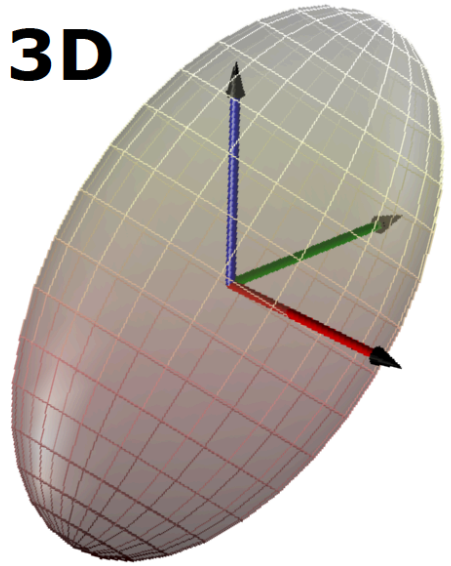
- It is a Bayes filter!
- Estimator for the **linear Gaussian** case
- Optimal solution for linear models and Gaussian distributions

Kalman filter distributions

- Everything is Gaussian!

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \right)$$





Linear Model

- The Kalman filter assumes
  - a **linear transition and observation model**
  - zero-mean Gaussian noise

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

$$z_t = C_t x_t + \delta_t$$

Components of a Kalman filter

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

B_t Matrix $(n \times l)$ that describes how the control u_t changes the state from $t-1$ to t.

C_t Matrix $(k \times n)$ that describes how to map the state x_t to an observation z_t.

ϵ_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.

δ_t

Linear motion model

- Motion under Gaussian noise leads to

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

Linear motion model

- Motion under Gaussian noise leads to

$$p(x_t \mid u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}}$$
$$\exp\left(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right)$$

  - R_t describes the noise in the motion

Linear observation model

- Measurement under Gaussian noise leads to

$$p(z_t \mid x_t) = ?$$

Linear observation model

- Measurement under Gaussian noise leads to

$$p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right)$$

  - Q_t describes the measurement noise

Everything stays Gaussian

- Given an initial Gaussian belief, the belief is always Gaussian

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

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

- Proof is non-trivial
  (see “Probabilistic robotics, Sec. 3.2.4.)

Kalman filter algorithm

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

2: $\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

3: $\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

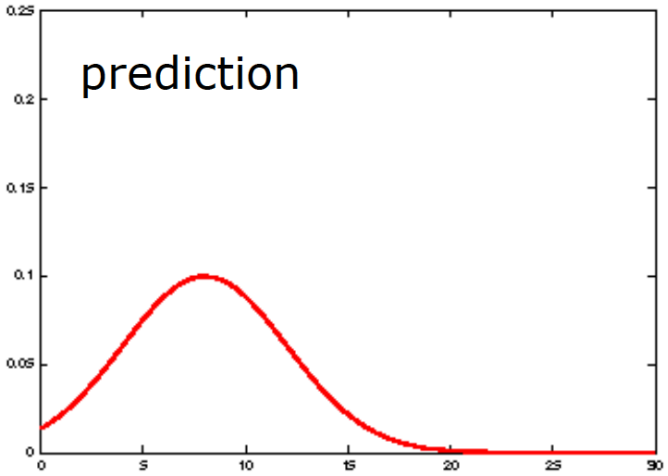
4: $K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

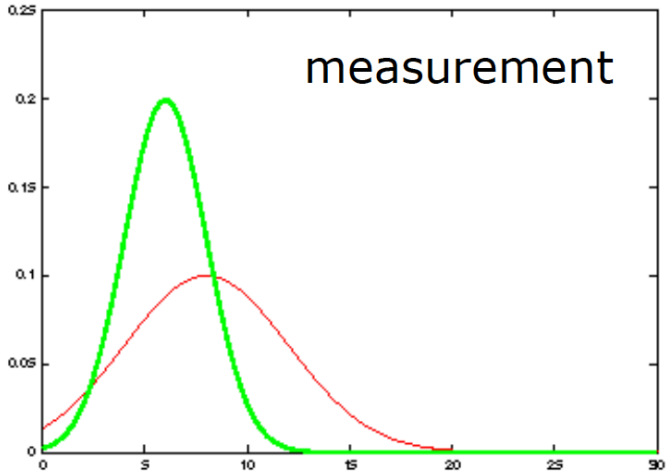
5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

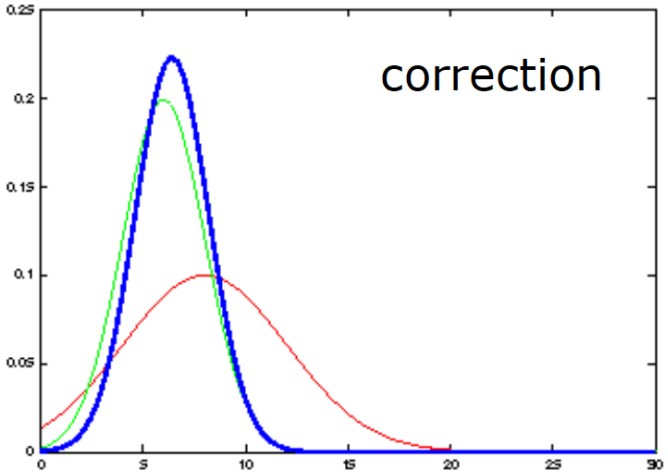
6: $\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

7: *return* μ_t, Σ_t

1D Kalman filter example

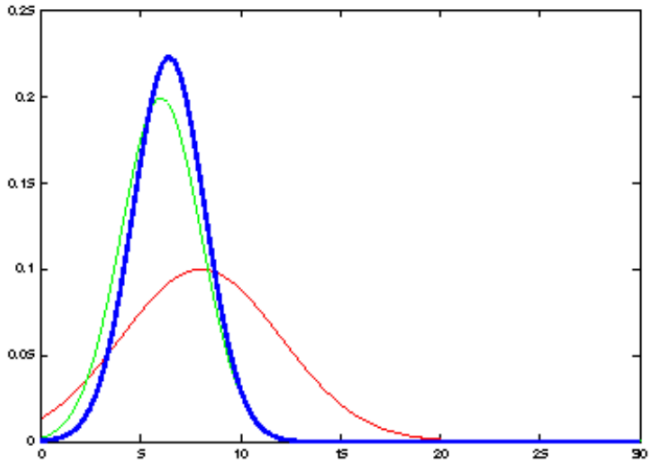


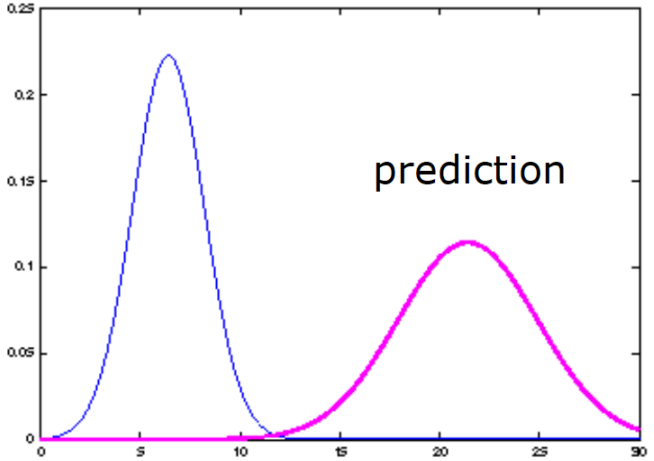


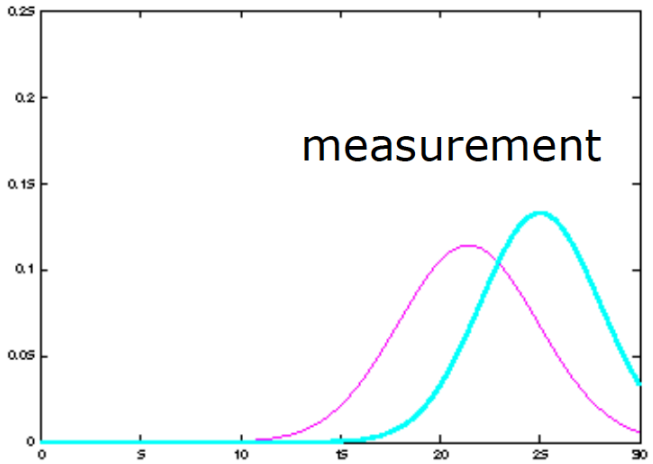


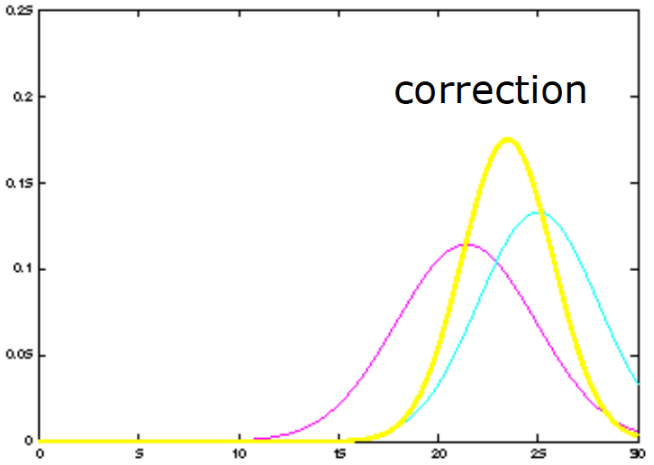
It's a weighted mean!

1D Kalman filter example



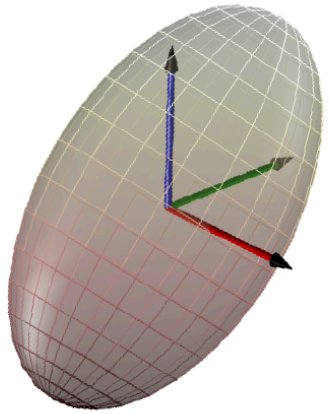






Kalman filter summary

- Gaussian distributions and noise
- Linear motion and observation model



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

$$z_t = C_t x_t + \delta_t$$

- What if this is not the case?

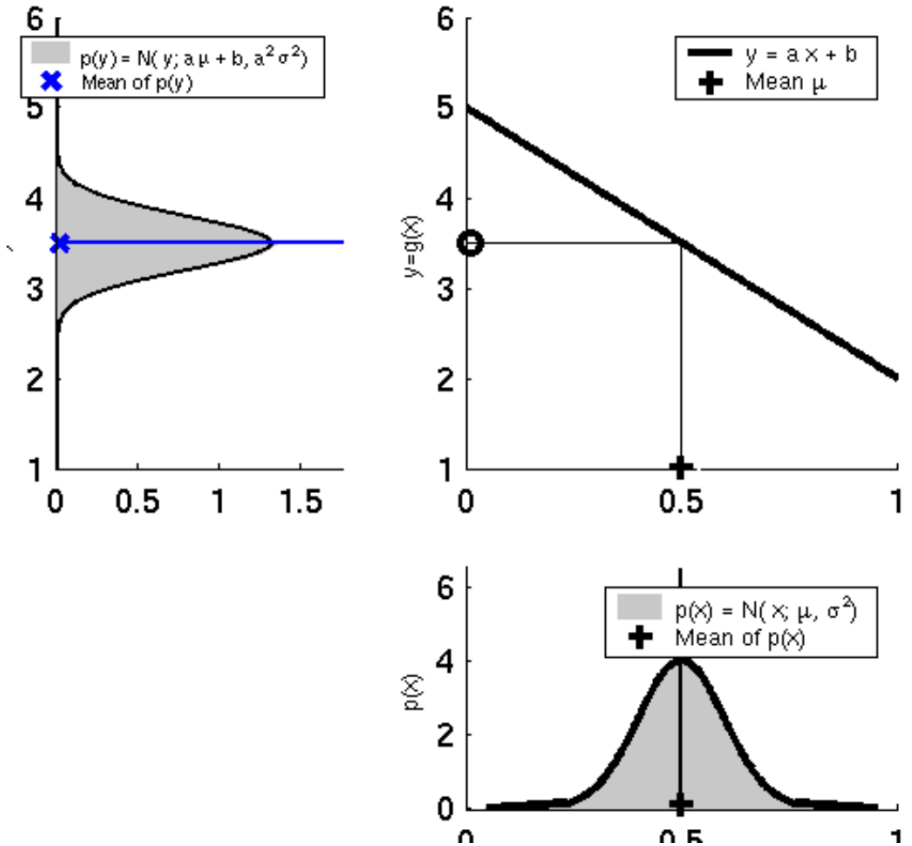
Nonlinear dynamic systems

- Most realistic problems (in computer vision/robotics) involve nonlinear functions

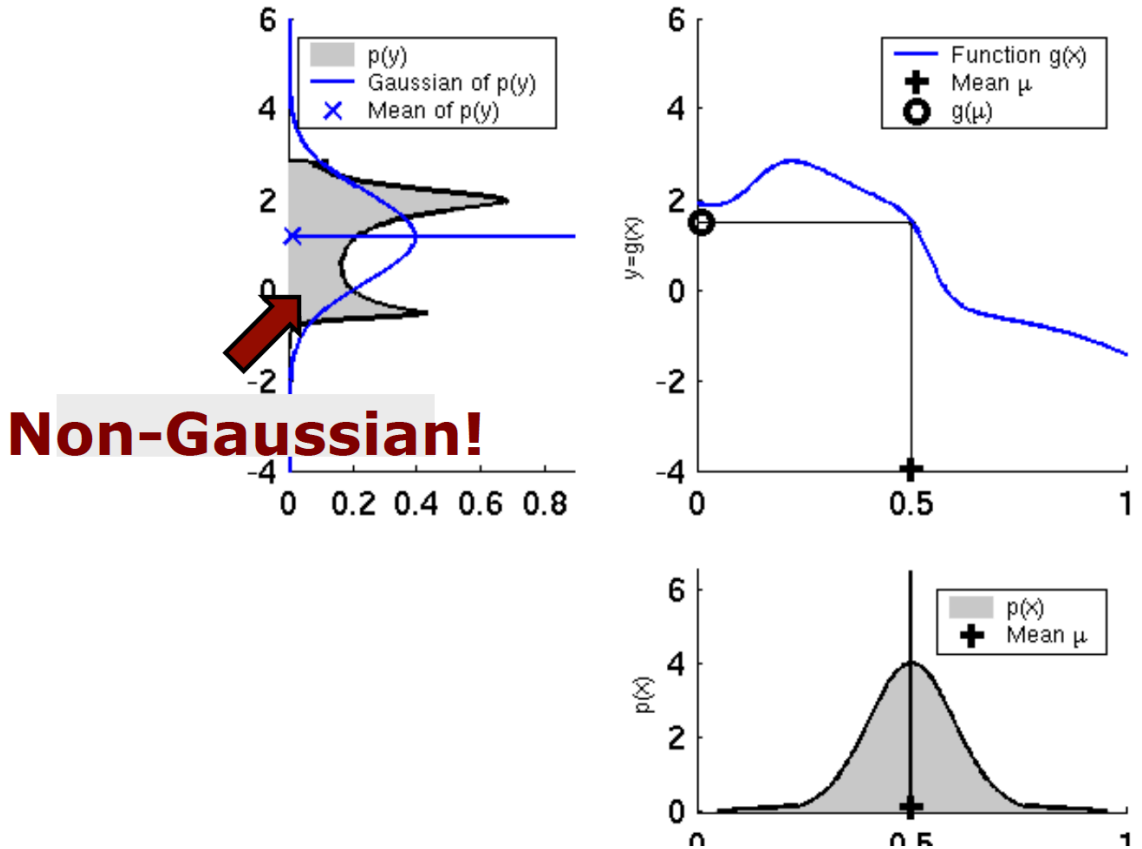
$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \qquad z_t = C_t x_t + \delta_t$$

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \qquad z_t = h(x_t) + \delta_t$$

Linearity assumption revisited



Nonlinear function



Non-Gaussian distributions

- The non-linear functions lead to non-Gaussian distributions
- Kalman filter is not applicable anymore!

- What can be done to resolve this?
  - Local linearization!

- Extended Kalman filter
  - Still a Gaussian filter

EKF Linearization

- First-order Taylor expansion!
- Prediction

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \underbrace{\frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}}_{=: G_t} (x_{t-1} - \mu_{t-1})$$

- Correction

$$h(x_t) \approx h(\bar{\mu}_t) + \underbrace{\frac{\partial h(\bar{\mu}_t)}{\partial x_t}}_{=: H_t} (x_t - \bar{\mu}_t)$$

Jacobian matrices

Reminder Jacobian Matrix

- It is a non-square matrix $m \times n$ in general
- Given a vector-valued function

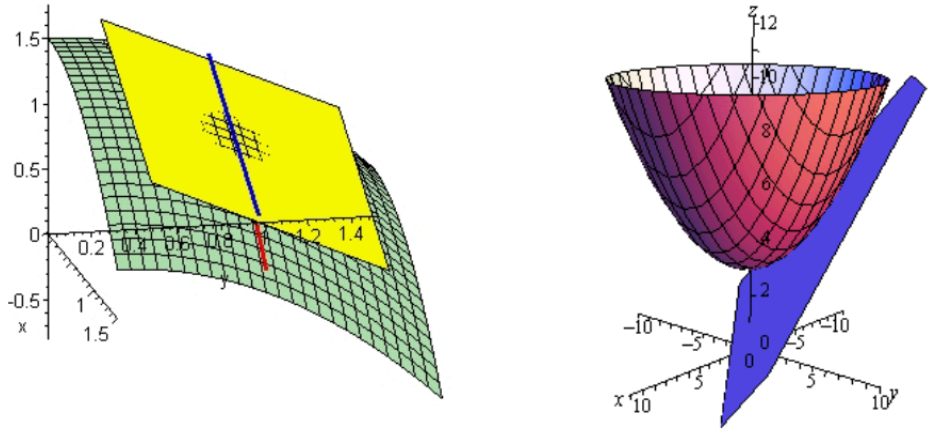
$$g(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{pmatrix}$$

- The Jacobian matrix is defined as

$$G_x = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_n} \end{pmatrix}$$

Reminder: Jacobian Matrix

- It is the orientation of the tangent plane to the vector-valued function at a given point



- Generalizes the gradient of a scalar valued function!

EKF Linearization

- First-order Taylor expansion!
- Prediction

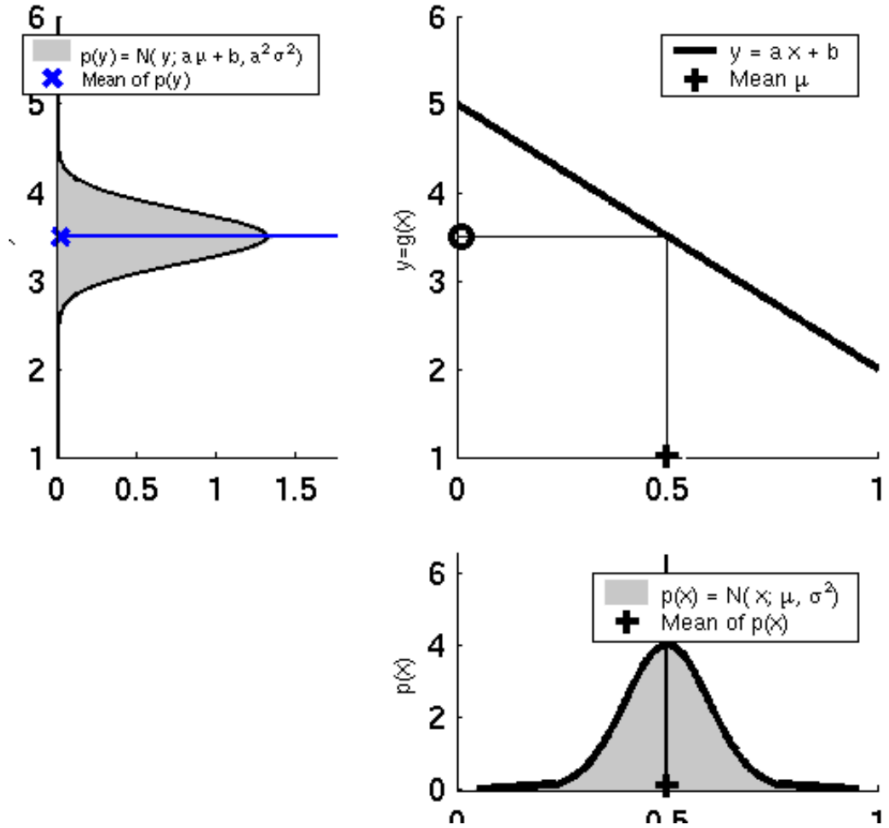
$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + \underbrace{\frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}}_{=: G_t} (x_{t-1} - \mu_{t-1})$$

- Correction

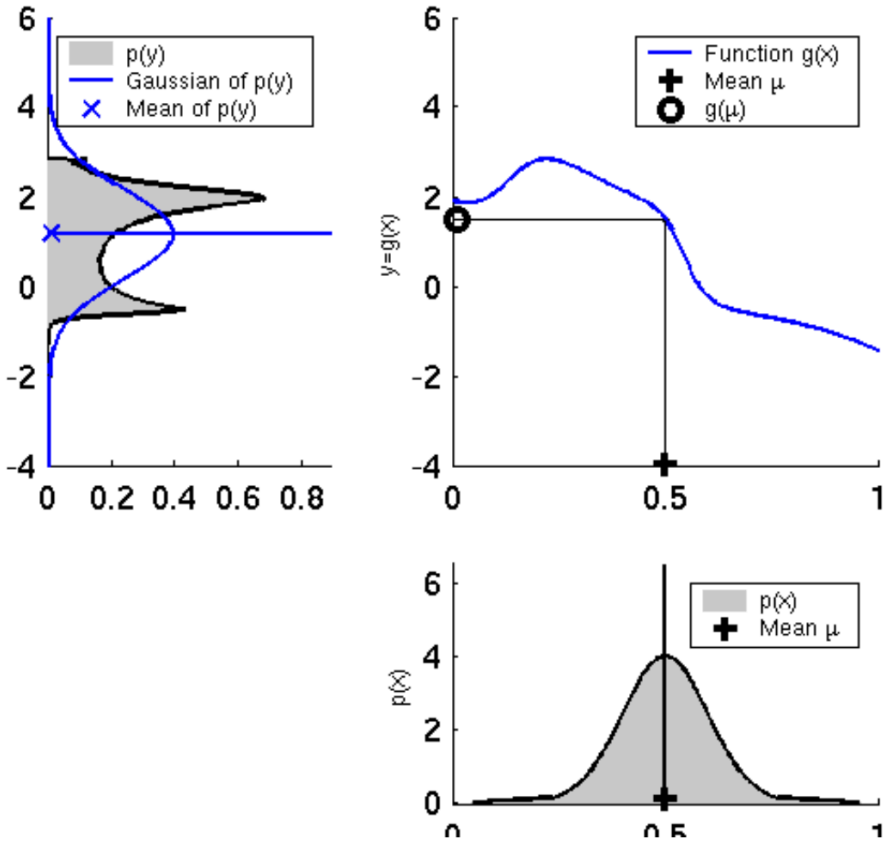
$$h(x_t) \approx h(\bar{\mu}_t) + \underbrace{\frac{\partial h(\bar{\mu}_t)}{\partial x_t}}_{=: H_t} (x_t - \bar{\mu}_t)$$

Linear functions!

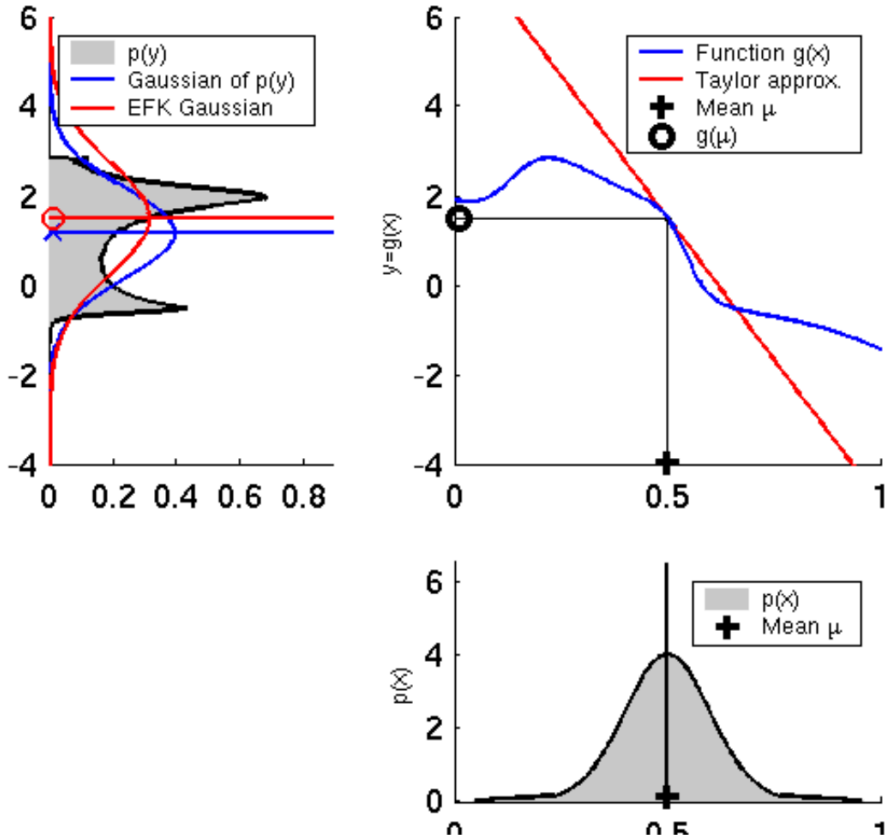
Linearity assumption revisited



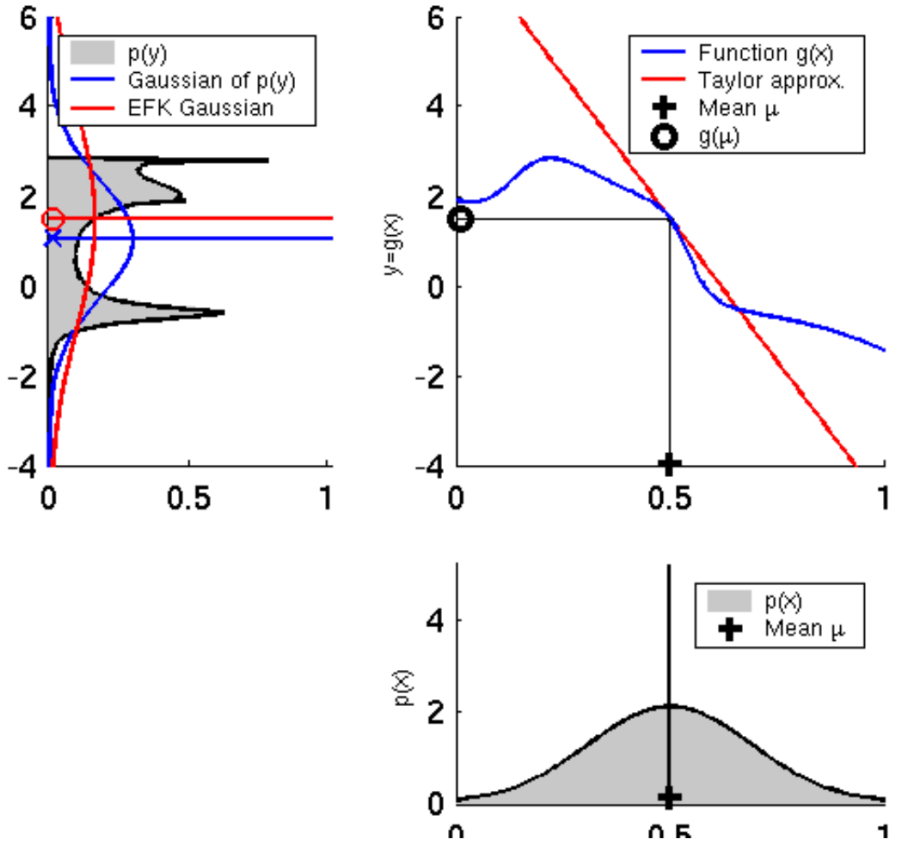
Non-linear function



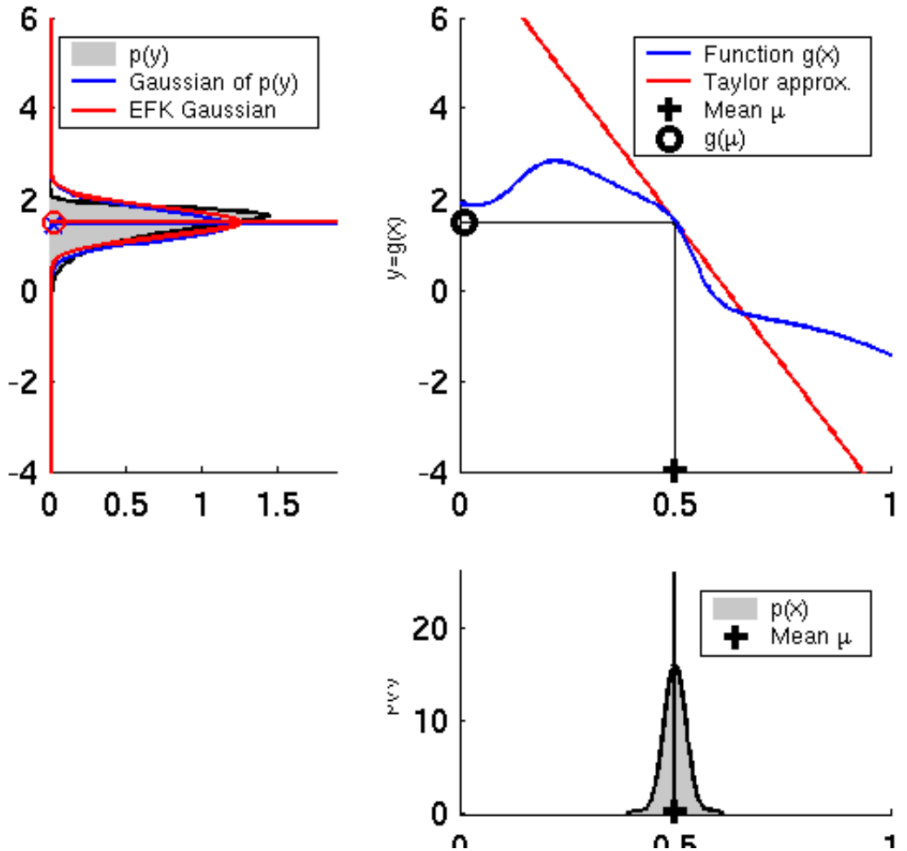
EKF linearization (1)



EKF linearization (2)



EKF linearization (3)



Linearized motion model

- The linearized model leads to

$$p(x_t \mid u_t, x_{t-1}) \approx \det(2\pi R_t)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1}))^T R_t^{-1}(x_t - \underbrace{g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})}_{\text{linearized model}})\Big)$$

  - R_t describes the noise in the motion

Linearized observation model

- The linearized model leads to

$$p(z_t \mid x_t) = \det\left(2\pi Q_t\right)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}\left(z_t - h(\bar{\mu}_t) - H_t\left(x_t - \bar{\mu}_t\right)\right)^T Q_t^{-1}\left(z_t - \underbrace{h(\bar{\mu}_t) - H_t\left(x_t - \bar{\mu}_t\right)}_{\text{linearized model}}\right)\Big)$$

  - Q_t describes the measurement noise

Extended Kalman filter algorithm

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

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: *return* μ_t, Σ_t

$A_t \leftrightarrow G_t$

$C_t \leftrightarrow H_t$

KF vs. EKF

EKF summary

- Extension of the Kalman filter
- One way to handle the non-linearities
- Performs local linearizations
- Works well in practice for moderate non-linearities
- Large uncertainty leads to increased approximation error error