MŰEGYETEM 1782

Budapest University of Technology and Economics
Dep. of Control for Transportation and Vehicle Systems

Domonkos Csuzdi

Students' Scientific Conference Report

Daum–Huang Filter for LiDAR-based Mobile Robot Localization

Consultant:
Olivér Törő
PhD candidate

Budapest, 2021

Acknowledgement

This work has been supported by the Hungarian Government and co-financed by the European Social Fund. (Contract Identifier: EFOP-3.6.3-VEKOP-16-2017-00001. Talent management in autonomous vehicle control technologies.) Their support is gratefully acknowledged.

1 Mobile Robot Localization

1.1 The Localization Task

During localization the main goal is to determine the coordinate transform between the local coordinate system of the robot, and a given global frame. For most of the problems the Global Navigation Satellite System (GNSS), like GPS provides this information. In an ideal world the GPS data is precise and reliant, making localization algorithms unnecessary. However, it is well known that global positioning by satellites cannot be performed in shielded environments (for example indoors), and even outdoors, the provided precision is often not sufficient.

To overcome these deficiencies, different state estimation algorithms are used to obtain the pose of the robot in the global frame. Localization and mapping often goes hand in hand: localization without a map (or some kind of a representation of the environment on which the global frame is defined), and map creation without the information about the pose is hardly possible. If one of them is assumed to be known, the task is much easier: localization, or mapping. If both are sought after, the Simultaneous Localization and Mapping (SLAM) problem arises, which is significantly harder the any of them separately. In the scope of this report, only the localization task is addressed on a given (ground truth) map.

One possible way to achieve localization is the introduction of pose hypotheses. By this the robot's belief of its pose is a probability distribution, instead of a crisp value [1]. Knowing the pose exactly is not feasible in a noisy real world environment. For a localization task, two main hardware components are used: an effector which is responsible for moving the robot, and an exteroceptive sensor, which is responsible to obtain information about the surrounding environment (like a LiDAR, a sonar, or often a camera). Both introduce errors and noise to the system which could be dealt with by the application of the probabilistic approach. In the following, this modelling method is detailed.

Almost every state estimation (e.g. localization) algorithm is based on Bayesian filtering. It servers as a foundation for these methods, and could not be implemented on its own. A Bayes filter conducts of two main parts which are iterated over time: prediction and update (see more in [1]). These have the following forms (respectively):

$$\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) bel(\mathbf{x}_{t-1}) \mathrm{d}\mathbf{x}_{t-1}, \tag{1.1}$$

$$bel(\mathbf{x}_t) = \eta p(\mathbf{z}_t|\mathbf{x}_t)\overline{bel}(\mathbf{x}_t), \tag{1.2}$$

where

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_{1:t}, \mathbf{u}_{1:t}), \tag{1.3}$$

$$\overline{bel}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}). \tag{1.4}$$

The notations are the following: $\mathbf{x}_t$ is the pose (position in 2D, and heading direction) at time t, $\mathbf{u}_t$ is the input vector at t, η is a normalization constant from Bayes' theorem,

$\mathbf{z}_t$ is the measurement at t, $(.)_{1:t}$ denotes values from time $t = 1$ to t, $bel(\mathbf{x}_t)$ is the belief (also called as posterior), and $\overline{bel}(\mathbf{x}_t)$ is the predicted belief.

First, $\overline{bel}(\mathbf{x}_t)$ is calculated from the prior $bel(\mathbf{x}_{t-1})$, using the motion model (first part in the integral). This is a prediction, because only the kinematics are incorporated, not the measurements. Then in the update part the measurement model is considered. This corrigates (updates) the prediction by incorporating the observations.

However, these probability distributions and integrals cannot be calculated on their own. Each distinct filter realization addresses the solution of the Bayesian recursion differently: the Kalman Filters use Gaussian distributions and their parametric description to estimate the pose hypotheses, while particle filters produce a more general numerical solution by describing an arbitrary distribution via particles. The Daum–Huang filters also use particles, but describes their movement with the help of the Fokker–Planck equation. These realizations are explained in greater detail in Section ****.

1.2 The Mobile Robot

An important preliminary of localization is the introduction of the utilized hardware, the operating environment, and their models. In this subsection, the mobile robot itself is discussed, along with its relevant sensors, followed by the environment representation where the robot has to be localized.

As an agent, the simulated version of ROBOTIS' TurtleBot3 is used via Robot Operating System (ROS) and Gazebo. This two-wheeled platform is widely used for educational and prototyping purposes due to its easy handling and well developed simulational counterpart. Although it has many useful components, here only the LiDAR and the differential drive are discussed due to their relevancy in the localization task. Those and the robot itself can be seen in Figure 1. The mounted 2D LiDAR provides range and angle meas-

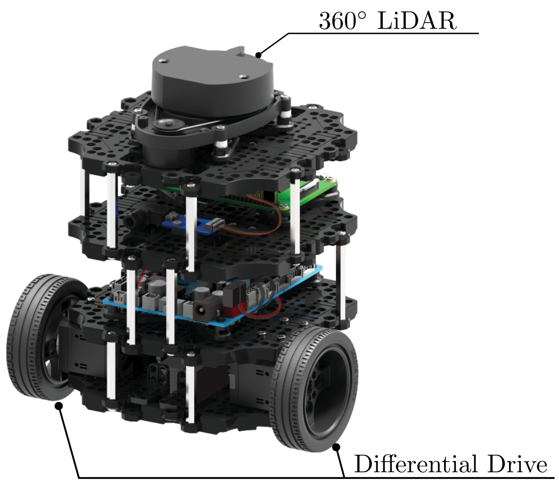


Fig. 1: ROBOTIS' TurtleBot3 Burger platform, with a mounted 360° LiDAR on top, and a differential drive (image source: www.robosklep.com).

urements from the environment with 360° field of view. This particular model (LDS-01) has an angular resolution of 1°, detection range of $0.12 - 3.5$ m-s, and accuracy (3σ) of ± 15 mm-s (actually the precision is distance dependent, but this effect is not considered). Invalid readings indicate out of range measurements. One full measurement is shown in Figure 2.

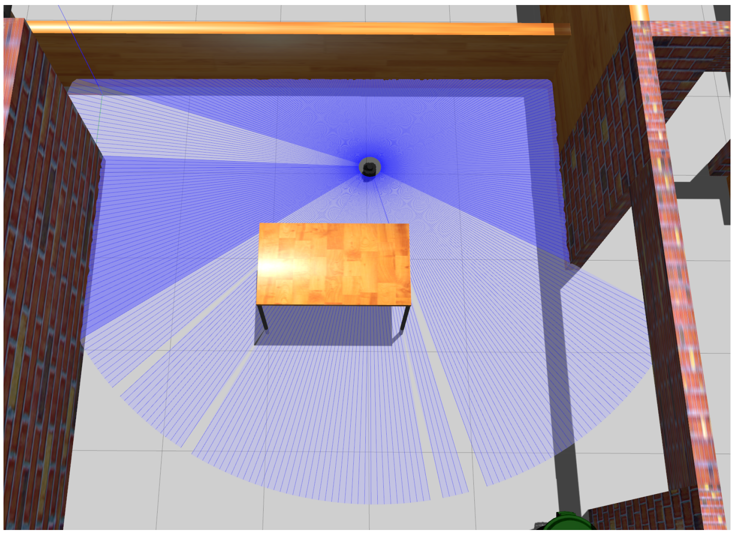


Fig. 2: Visualized 2D LiDAR measurement of the TurtleBot3 in a house environment, using the Gazebo simulator.

The platform has 2 independently-driven wheels, and one free turning wheel, which makes it eligible to be modelled by differential drive kinematics. The odometry is conducted by (simulated) rotary encoder readings from the two wheels separately. Originally the out-of-the-box ROS-Gazebo model by ROBOTIS did not considered odometry noise, which had to be manually added in order to efficiently model real world conditions. Precisely modelling odometry error is a difficult task, and even nowadays is an actively researched topic [2]. However, this degree of precision is not required here. Instead, both encoder readings are simply corrupted by a random variable each (ξ_L for the left wheel, ξ_R for the right), obtained as

$$\xi_L \sim \mathcal{N}(0, \alpha v_L^2), \tag{1.5}$$

$$\xi_R \sim \mathcal{N}(0, \alpha v_R^2), \tag{1.6}$$

where $\mathcal{N}(\mu, \sigma^2)$ stands for a normal distribution with mean μ and variance σ^2, α is a scaling parameter, and v_L, v_R are the corresponding velocities. Applying this small modification, ROS provides noisy odometry data at each tilmestep, which then can be used to establish the motion model of the robot (see Subsection 1.4).

1.3 The Map

The environment of the agent is represented by a 2D Occupancy Grid Map (OGM) [3]. This model describes the environment by dividing it to finitely many grid cells, where each grid cell is a random binary variable, representing that whether it's occupied, or not. Upon creating the OGM, the occupancy value of each cell (the probability, that its occupied) is iteratively updated. To obtain a final map, these values are thresholded, producing values of 1 if the cell is *mostly* occupied, or 0 is its *mostly not*. The top-down view of the TurtleBot3 House by ROBOTIS and the corresponding OGM can be seen in Figure 3, which was obtained by the SLAM GMapping algorithm [4]. For a pure localization task, the map is considered as ground truth, therefore the previously introduced odometry noise was omitted during the mapping process.

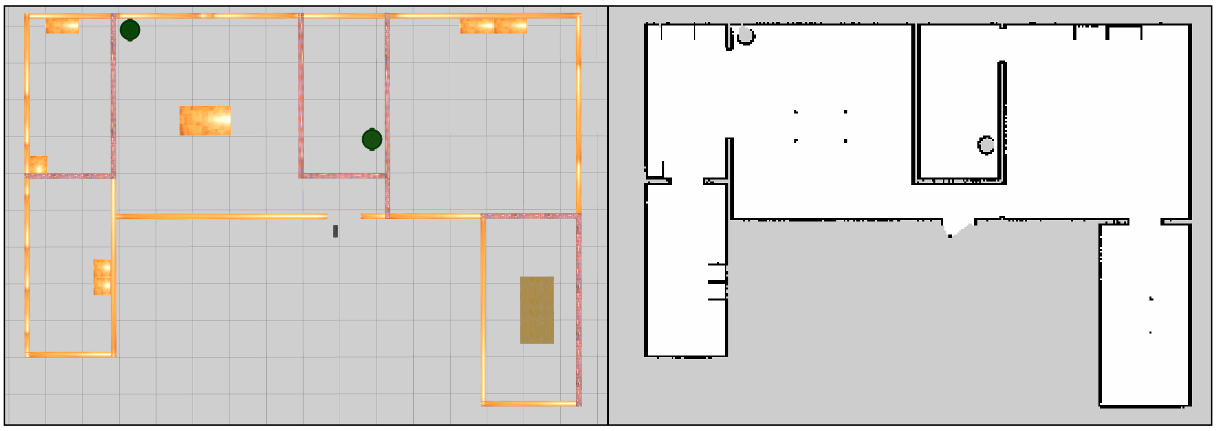


Fig. 3: The top-down view of the TurtleBot3 House in Gazebo, and its OGM representation (mind the open doors in the building). Grey pixels represent unknown area, black pixels are occupied, white pixels are free cells. 1 pixel (cell) in the OGM has a size of 0.05×0.05 m.

1.4 The Motion Model

Now that the hardware and the underlying localization task is introduced, the two main parts of the Bayesian recursion is going to be described in the following subsections: the motion model, and the measurement model.

The motion model is used to describe the probability distribution of $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$. Without any external information, the pose of the robot at time t can be estimated based on the previous pose at time $t-1$, and the control input at time t. Naturally, this estimation will be disrupted by tire slippage and drift. Due to the incremental nature, these errors are integrated over time, thus making the estimation more and more uncertain.

The motion model describes this transition using the kinematic model of the agent. Based on the inputs, two distinct probabilistic models can be established, introduced in [1]: the velocity motion model, and the odometry motion model. Here, only the latter is detailed.

If the odometry of the robot is available (i.e. by integrating wheel encoder measurements), they can be treated as a control input:

$$\mathbf{u}_t = (\overline{\mathbf{x}}_t \;\; \overline{\mathbf{x}}_{t-1})^\top = \left(\overline{x}_t \;\; \overline{y}_t \;\; \overline{\theta}_t \;\; \overline{x}_{t-1} \;\; \overline{y}_{t-1} \;\; \overline{\theta}_{t-1}\right)^\top . \tag{1.7}$$

The key is the fact that the relative difference between two consecutive odometry data is a good estimation of the relative difference between the two consecutive true poses, if the timestep is sufficiently small.

The transition between the state $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ is simplified as a sequence of a rotation, a translation, and another rotation. These are indicated in Figure 4. with δ_{rot1}, δ_{trans}, and δ_{rot2} respectively. It's important to mention that this separation to rotational and translational components is arbitrary (introduced by Thrun et al. in [1]); for another approach, see [5].

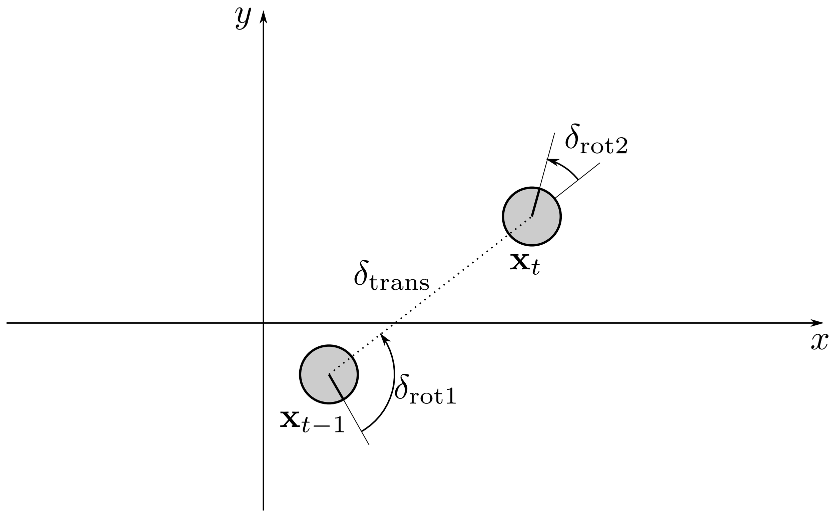


Fig. 4: The rotation-translation-rotation transition sequence from state $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$.

The control input $\mathbf{u}_t$ then transformed to the three transition components as:

$$\delta_{\text{rot1}} = \text{arctan2}\left(\overline{y}_t - \overline{y}_{t-1}, \overline{x}_t - \overline{x}_{t-1}\right) - \overline{\theta}_{t-1}, \tag{1.8}$$

$$\delta_{\text{trans}} = \sqrt{(\overline{x}_{t-1} - \overline{x}_t)^2 + \left(\overline{y}_{t-1} - \overline{y}_t\right)^2}, \tag{1.9}$$

$$\delta_{\text{rot2}} = \overline{\theta}_t - \overline{\theta}_{t-1} - \delta_{\text{rot1}}. \tag{1.10}$$

To model odometry noise, the inputs are treated as random variables, formulated by

$$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} + \xi_{\text{rot1}}, \qquad \xi_{\text{rot1}} \sim \mathcal{N}(0, \alpha_1\delta^2_{\text{rot1}} + \alpha_2\delta^2_{\text{trans}}), \tag{1.11}$$

$$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} + \xi_{\text{trans}}, \qquad \xi_{\text{trans}} \sim \mathcal{N}(0, \alpha_3\delta^2_{\text{trans}} + \alpha_4(\delta^2_{\text{rot1}} + \delta^2_{\text{rot2}})), \tag{1.12}$$

$$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} + \xi_{\text{rot2}}, \qquad \xi_{\text{rot2}} \sim \mathcal{N}(0, \alpha_1\delta^2_{\text{rot2}} + \alpha_2\delta^2_{\text{trans}}), \tag{1.13}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are error parameters.

Then, using the control inputs and the previous state, samples from $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$ are obtained by,

$$x_t = x_{t-1} + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}}), \tag{1.14}$$

$$y_t = y_{t-1} + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}}), \tag{1.15}$$

$$\theta_t = \theta_{t-1} + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}, \tag{1.16}$$

$$\mathbf{x}_t = (x_t \;\; y_t \;\; \theta_t)^\top . \tag{1.17}$$

2 Measurement Model

2.1 Introduction

As the form of the motion model depends on many factors, for example the kinematics of the robot (differential drive, car like structure, etc.), and the used control input interpretation (odometry-based, or velocity-based), measurement models are no different. However, the used proprioceptive sensor (LiDAR), and the map representation (OGM) in this task narrow the possibilities.

2.2 Standard Measurement Models on Different Map Realizations

Wide-spread algorithms are often established for topological (feature-based) maps. Here, the environment is described by detected shapes (features), rather than using a location based representation like the OGMs do, as they assign an occupancy value for every location on the map.

For a topological map, the probability distribution of $p(\mathbf{z}_t|\mathbf{x}_t)$, enhanced with the known map $\mathbf{m}$ (resulting in $p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})$) is described a Gaussian distribution:

$$h(\mathbf{x}_t, \mathbf{m}) = \mathbf{z}_t^* + \xi, \quad \xi \sim \mathcal{N}(0, \mathbf{R}), \tag{2.1}$$

$$\Rightarrow p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}) \sim \mathcal{N}(\mathbf{z}_t^*, \mathbf{R}). \tag{2.2}$$

where $\mathbf{R}$ is the covariance matrix of the measurement. This closed-form description is required for Kalman Filters.

However, for location based (metric) maps, the form of $\mathbf{m}$ is different, therefore to make the above model work, features have to be extracted [6], effectively making the map feature-based.

For purely metric maps (without feature extraction), Thrun et al. proposed the *beam range finder model* in [1], especially fitted for range finders, like LiDARs, or sonars. This model describes $p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})$ as a composition of probability densities, resulting in a distribution indicated in Figure 5. Here, the value $\mathbf{z}_t^*$ is calculated by ray tracing: at a given robot pose, the expected measurement can be calculated, by tracing a ray along the map, starting form the robot pose, and ending in the nearest obstacle. Then the travelled distance is recorded, which together with the angle of the beam, serves as $\mathbf{z}_t^*$.

The resulting distribution combines different measurement error possibilities: random measurements (uniform distribution), local measurement noise (Gaussian distr.), failures ("very narrow" uniform distribution, modelling a Dirac delta) and unexpected objects (exponential distribution).

However, the resulting probability distribution does not have a closed form description, which is not a problem for particle filters, but it is for Kalman filters.

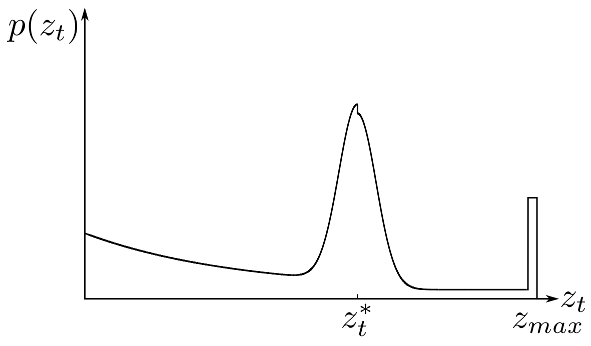


Fig. 5: The probability distribution of $p(z_t|\mathbf{x}_t, \mathbf{m})$. The figure is based on [1].

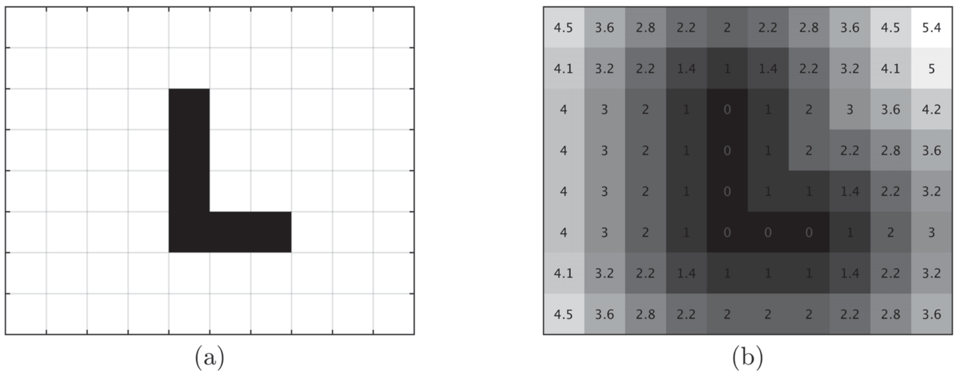
2.3 The Distance Function based Measurement Model

The method, which enables the use of Kalman filters with laser range finder measurements on an OGM is proposed by Dantanarayana et al. in [7, 8]. In the following, this method is going to be detailed, based on [9].

The key idea is the utilization of the distance transform (DT), which is widely used in image processing. For a given binary image (or in this case, an OGM), the distance transform produces an image/map (with the same size), where the value of each pixel/cell is determined by the closest distance to any occupied pixel/cell. If V is the set of the occupied cells, and $\mathbf{x}$ is an arbitrary point on the map, then

$$d_{DF} = DF(\mathbf{x}) = \min_{\mathbf{v_j} \in V} \|\mathbf{x} - \mathbf{v_j}\|\,, \tag{2.3}$$

in which an Euclidean norm is usually applied. The distance transform then could be pre-calculated for a given map. Figure 6 shows the DT in a simplified environment, while the DT of one part of the OGM from Figure 7. can be seen on Figure



4.5	3.6	2.8	2.2	2	2.2	2.8	3.6	4.5	5.4
4.1	3.2	2.2	1.4	1	1.4	2.2	3.2	4.1	5
4	3	2	1	0	1	2	3	3.6	4.2
4	3	2	1	0	1	2	2.2	2.8	3.6
4	3	2	1	0	1	1	1.4	2.2	3.2
4	3	2	1	0	0	0	1	2	3
4.1	3.2	2.2	1.4	1	1	1	1.4	2.2	3.2
4.5	3.6	2.8	2.2	2	2	2	2.2	2.8	3.6

Fig. 6: A simple shape and its distance transform. Figure source: [9].

Now using the LiDAR measurements, which in this case come in the form of angle-range pairs, each detection can be transformed to map coordinates, and assigned a distance transform value. Theoretically, if the estimated pose of the robot, and the true pose of the robot aligns, and the sensor is noiseless, each reading should indicate an occupied cell on the OGM. Therefore, all the projected detections should have a DT of zero. It other

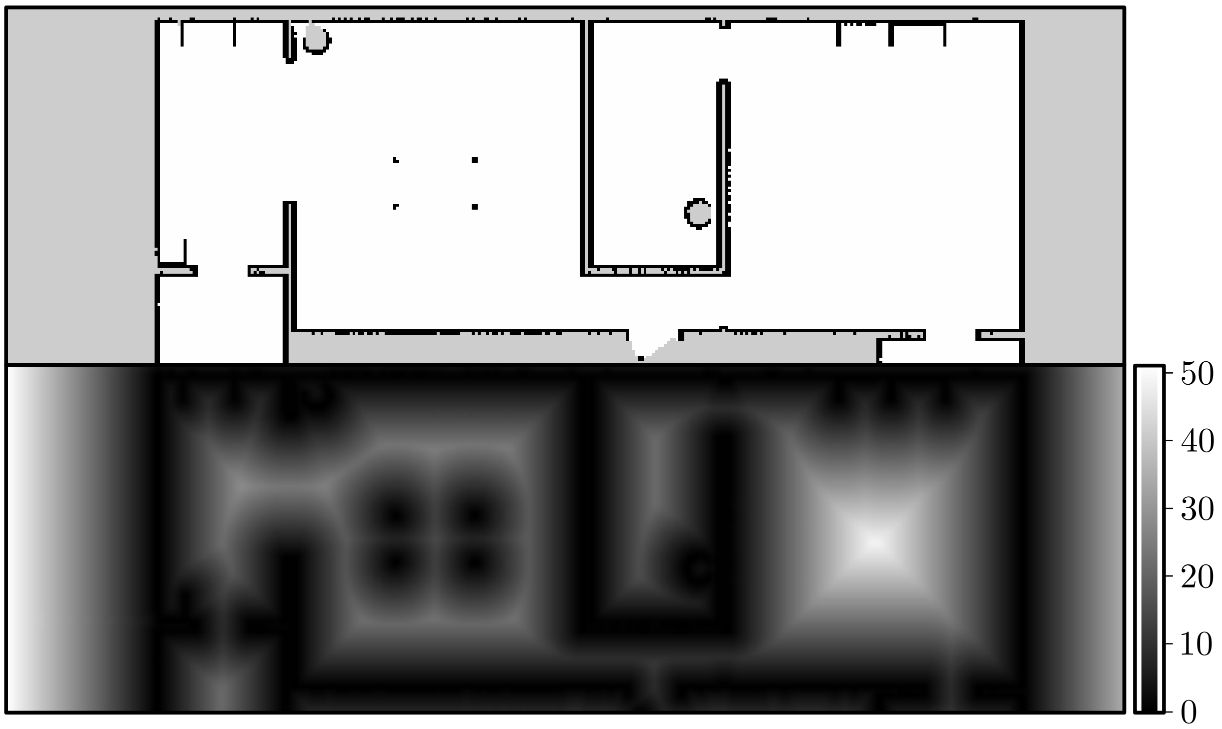


Fig. 7: Part of the OGM of the TurtleBot3 House, and its distance transform.

cases, the ray endpoints could end up on free cells, thus having a DT value larger, than zero.

The method of transforming (projecting) the laser scans to the map, based on the estimated location of the robot is shown in Figure 8. Mathematically, these projections are obtained as

$$\mathbf{x}_{oi} = \begin{bmatrix} x_{oi} \\ y_{oi} \end{bmatrix} = \begin{bmatrix} x + r_i \cos(\varphi_i + \theta) \\ y + r_i \sin(\varphi_i + \theta) \end{bmatrix}, \tag{2.4}$$

where the variables are according to Figure 8. (the time dependency is omitted).

Based on a set of measurements (for a whole revolution of the LiDAR), a vector can be constructed as follows:

$$\mathbf{d}_{\mathrm{DF}} = \begin{bmatrix} DF(\mathbf{x}_{o1}) \\ DF(\mathbf{x}_{o2}) \\ \vdots \\ DF(\mathbf{x}_{on}) \end{bmatrix}, \tag{2.5}$$

where n is the number of valid LiDAR readings for a whole revolution (a valid reading occurs when the laser ray actually collided with an object within range, and this was detected by the sensor). To obtain a single scalar value, which indicates the disparity between the projected measurements, and the map, the Chamfer distance (CD) is formulated. For a set of measurements,

$$d_{\mathrm{CD}} = \frac{1}{n} \sum_{i=1}^{n} DF(\mathbf{x}_{oi}). \tag{2.6}$$

The CD is a good indicator for the estimation of the robot's pose: $CD = 0$ means, that

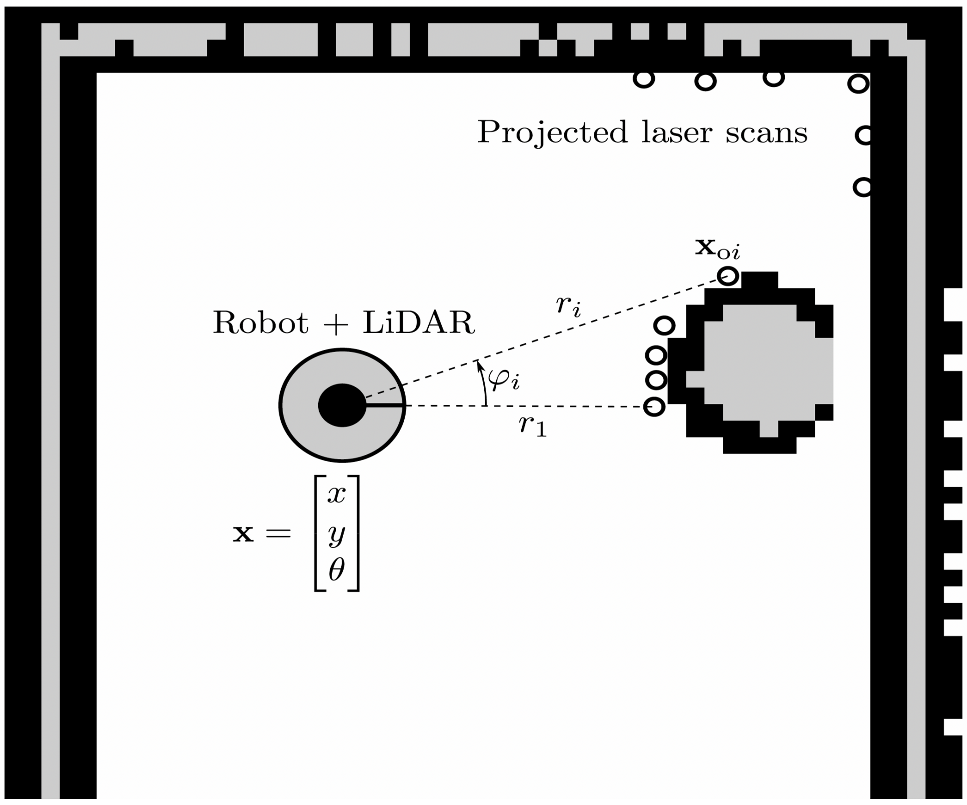


Fig. 8: Projection of the LiDAR scans to the OGM, based on the pose of the robot.

every ray endpoint for a set of measurement at time t is at the right place, therefore the initial robot pose estimation is good. Although, it is important to mention that this assumption does not hold in every scenario. Consider an extreme case: for a whole revolution, the LiDAR only records one measurement (one ray), and the OGM only contains many occupied cell. Then the CD (effectively the DT) would equals to zero at a plethora of robot poses, thus not indicating the sought true pose. For a detailed map and sufficiently many LiDAR reading per revolution, this deficiency is minimized.

Using the CD, the measurement model is formulated as

$$p(\mathbf{z}_t|\mathbf{x}_t, DF_t) \sim \mathcal{F}(d_{\mathrm{CD}t}, \mathbf{\Sigma}_{\mathrm{CDt}}). \tag{2.7}$$

, where $\mathcal{F}$ denotes the folded Gaussian distribution. The covariance is calculated as

$$\mathbf{\Sigma}_{\mathrm{CD}} = \mathbf{J}_{\mathrm{CD}}\mathbf{R}\mathbf{J}_{\mathrm{CD}}^{\top}, \tag{2.8}$$

where $\mathbf{R}$ is the measurement covariance matrix, and $\mathbf{J}_{CD}$ is the Jacobian of the Chamfer distance with respect to $\mathbf{z}_i = \begin{bmatrix} \varphi_i & r_i \end{bmatrix}^{\top}$, evaluated at $\mathbf{x}$. However, due to the assumption, that the measurement only has error in r_i, the differentiation with regarding to φ_i is omitted.

Due to the fact, that the expected CD is zero at the true pose, the measurement equation has the close form of

$$d_{CD} := h(\mathbf{x}_t, \mathbf{z}_t) = 0, \tag{2.9}$$

forming an implicit equation for $\mathbf{x}_t$ and $\mathbf{z}_t$, which is applicable for Kalman filters.

2.4 The Kalman Filters

2.4.1 Kalman Filter

The Kalman filter (KF) assumes that "everything" *is* linear and Gaussian, and under these assumptions it can be proved that "everything" *stays* linear and Gaussian during the Bayes filter updates. A Gaussian distribution can be fully characterized by its first two moments, therefore the Kalman filter offers a recursion to calculate the mean $\boldsymbol{\mu}_t$ and the covariance $\boldsymbol{\Sigma}_t$ of the belief $bel(\mathbf{x}_t)$. In this context only the discrete-time KF is going to be discussed.

Consider the discrete-time (linear) state space model of a system:

$$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \mathbf{G}_t\mathbf{v}_t, \tag{2.10}$$

$$\mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{w}_t, \tag{2.11}$$

where $\mathbf{A}_t$ is the state transition matrix, $\mathbf{B}_t$ is the input gain, $\mathbf{G}_t$ is the noise gain, $\mathbf{C}_t$ is the measurement matrix. To incorporate uncertainties and inaccuracies into the system, the process noise $\mathbf{v}_t$ and the measurement noise $\mathbf{w}_t$ is introduced. Both are Gaussian random variables: $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, with $\mathbf{Q}$ as process noise covariance, and $\mathbf{R}$ as measurement noise covariance.

Using the linear nature of the model, and assuming, that the probability distributions in the Bayes filter are Gaussian distributions, the motion model has the following form:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) = \det(2\pi\tilde{\mathbf{Q}}_t)^{-\frac{1}{2}}\exp\{-\frac{1}{2}(\mathbf{x}_t - \mathbf{A}_t\mathbf{x}_{t-1} - \mathbf{B}_t\mathbf{u}_t)^\top\tilde{\mathbf{Q}}_t^{-1}(\mathbf{x}_t - \mathbf{A}_t\mathbf{x}_{t-1} - \mathbf{B}_t\mathbf{u}_t)\}, \tag{2.12}$$

where $\tilde{\mathbf{Q}}_t = \mathbf{G}_t\mathbf{Q}_t\mathbf{G}_t^\top$. The measurement model can be calculated similarly.

The derivation of the Kalman filter algorithm is not trivial, and does not fall within the scope of this document. The algorithm can be seen in Algorithm 1., where $\mathbf{I}$ denotes the identity matrix.

Algorithm 1 Kalman filter($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$)

```
1: prediction:
2:     $\overline{\boldsymbol{\mu}}_t = \mathbf{A}_t\boldsymbol{\mu}_{t-1} + \mathbf{B}_t\mathbf{u}_t$
3:     $\overline{\boldsymbol{\Sigma}}_t = \mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^\top + \tilde{\mathbf{Q}}_t$
4: update/correction:
5:     $\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t\mathbf{C}_t^\top\left(\mathbf{C}_t\overline{\boldsymbol{\Sigma}}_t\mathbf{C}_t^\top + \mathbf{R}_t\right)^{-1}$
6:     $\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}_t\overline{\boldsymbol{\mu}}_t)$
7:     $\boldsymbol{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{C}_t)\overline{\boldsymbol{\Sigma}}_t$
8: return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$
```

2.4.2 Extended Kalman Filter

However, assuming a linear state space model is a rather strong assumption. For a nonlinear model the general state space equations are (not considering nonlinear noise gain):

$$\mathbf{x}_t = \mathbf{a}(\mathbf{u}_t, \mathbf{x}_{t-1}) + \mathbf{G}_t \mathbf{v}_t, \tag{2.13}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \mathbf{w}_t. \tag{2.14}$$

The state propagation function $\mathbf{a}(.)$ can be locally linearized around the previous state $\mathbf{x}_{t-1} = \boldsymbol{\mu}_{t-1}$, resulting $\tilde{\mathbf{A}}_t$. The measurement function $\mathbf{h}(.)$ is linearized around the current predicted state $\overline{\boldsymbol{\mu}}_t$, resulting $\mathbf{H}_t$.

The EKF algorithm is the extension of the KF by handling nonlinear state equations through their linearized forms. Algorithm 2. contains the EKF algorithm.

Algorithm 2 Extended Kalman filter($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$)

```
1: prediction:
2:     $\overline{\boldsymbol{\mu}}_t = a(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})$
3:     $\overline{\boldsymbol{\Sigma}}_t = \tilde{\mathbf{A}}_t \boldsymbol{\Sigma}_{t-1} \tilde{\mathbf{A}}_t^\top + \tilde{\mathbf{Q}}_t$
4: update/correction:
5:     $\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\top \left(\mathbf{H}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\top + \mathbf{R}_t\right)^{-1}$
6:     $\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t \left(\mathbf{z}_t - h(\overline{\boldsymbol{\mu}}_t)\right)$
7:     $\boldsymbol{\Sigma}_t = \left(\mathbf{I} - \mathbf{K}_t \mathbf{H}_t\right) \overline{\boldsymbol{\Sigma}}_t$
8: return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$
```

2.4.3 Advantages and Disadvantages of Kalman Filters

Under linear and Gaussian assumptions the Kalman filter yields the optimal estimation (by calculated mean square error). However, these criteria are rarely satisfied, and the optimal estimation property does not hold any more. Another major problem comes from the nature of Gaussian distributions: they are unimodal, therefore the Kalman Filter cannot track multiple high-probability hypotheses.

EKF offers one type of solution to the nonlinearity: linearization by Taylor expansion. This however, introduces linearization error into the system. For highly nonlinear equations not only the first order approximation is the problem, but its center point as well. As were mention before, the state equation is linearized around the previous state; but now this state also an approximation. This overall effect can cause the algorithm to diverge, or to yield poor results. In addition, the initial conditions also have to be carefully chosen in order to obtain desired results.

Despite these, the EKF is still one of the most popular state estimation algorithm in robotics, and often used in Global Navigation Satellite Systems (GNSS) as well, thanks to its computational simplicity and easy implementation. Other Kalman filter based realizations are including but not limited to the unscented Kalman filter (UKF), which uses a different type of linearization instead of the Taylor expansion based one, and the ensemble Kalman Filter (EnKF), which is suited for high dimensional estimation problems.

2.5 The Particle Filters

2.5.1 Main concepts

Particle filters (PF) are used to solve nonlinear filtering problems. Unlike EKF and other nonlinear extensions of the Kalman filter, this method uses a nonparametric representation of the estimated posterior. This nonparametric nature enables the estimation of various distributions, including multimodal and/or non-Gaussian.

The main concept in PF is to represent arbitrary probability distributions by cumulative probability mass of particles. When for example a Gaussian distribution is sampled multiple times, more samples come from regions with high probability density. This (trivial) effect can also be used in reverse: if samples are given, what is the probability distribution from which they were sampled? If the number of particles is sufficiently large (ideally infinity), they can offer a good approximation of the probability distribution.

The Bayesian filtering framework is implemented in as follows . First, draw N samples (called as particles) from a prior distribution. Then migrate each of these particles according to the motion model. Now the particle set will approximately describe the predicted belief (1.4). (In the KF this predicted belief was described by its mean and covariance, as it was assumed to be a Gaussian distribution. Now, it is described by particles, and the Gaussian nature is not required.)

The KF used the $\mathbf{K}_t$ Kalman gain as a weight to calculate the posterior from the predicted belief. Particle filters use *importance sampling* for this task: each sample is given an importance weight which describes its relevance. If this relevance is high, it means that the specific particle is a good approximation of the true (sought) state. Choosing a proper weighting method is a difficult task. Generally, weights are calculated from a target and a proposal distribution. The target distribution is the posterior, the proposal distribution however, can be arbitrary (under some criteria). Now using the particles and their weights, the posterior distribution is approximated.

The next step of the filter is optional, although essential: resampling. Here, a new (same sized) particle set is constructed by drawing particles from the old set with replacement, according to their weights. This means that some particles can be chosen multiple times (more likely high-weighted ones), thus certain particles can be eliminated. Finally, their weights are equalized. With the resulting particle set (either the new, redrawn one, or the old, weighted one), the algorithm is repeated in the new time step.

Generally, the most demanding task when designing PFs is the proper selection of the proposal density and the resampling frequency (i.e. when to resample). Choosing the right proposal distribution and the right time to resample is crucial in the performance of the PF. In the next subsection, the bootstrap particle filter method with adaptive resampling is introduced in detail.

2.5.2 The Bootstrap Particle Filter

Denote the particles and their set at time t as:

$$\mathcal{X}_t = \{\mathbf{x}_t^{[1]}, \mathbf{x}_t^{[2]}, \dots \mathbf{x}_t^{[N]}\}. \tag{2.15}$$

The bootstrap particle filter uses the predicted belief as the proposal distribution, which results in the following weight assignment:

$$w_t^{[n]} = p(\mathbf{z}_t | \mathbf{x}_t^{[n]}). \tag{2.16}$$

This means, that if the proposal distribution is the predicted belief, the weight is calculated according to the measurement model (the mathematical derivation is omitted). In the original algorithm [**?**], resampling is performed in every step. However, this might not be beneficial, therefore an effective sample size based adaptive resampling is applied [**?**]. One major problem with particle filters is when many particles are in irrelevant locations, therefore only a few of them represent the high probability region of the posterior properly. This called particle degeneracy. However, the value of the weights can indicate this ill-favored situation. The effective sample size (ESS or N_{eff}) can be approximated as the following:

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (w_t^{[i]})^2}, \tag{2.17}$$

if the sum of weights are 1 (normalized). Consider the situation where all the weights are equal, thus $w_t^{[1]} = w_t^{[2]} = \cdots = w_t^{[N]} = 1/N$. This results in $N_{\text{eff}} = N$, which is beneficial. The worst case scenario is when one weight is 1, and all the others are 0, which results in $N_{\text{eff}} = 1$. It is common to define a threshold N_T, where if $N_{\text{eff}} < N_T$, a resampling is performed.

The pseudocode of the bootstrap particle filter with adaptive resampling is presented in Algorithm 3. There are several ways to implement the resampling in Line 14: the most basic method is the ”roulette wheel” sampling, but in this scenario, the stratified sampling [**?**] is used.

2.5.3 Advantages and Disadvantages of Particle Filters

Particle filters are well suited for nonlinear and non-Gaussian state estimation. They are especially versatile: many PF variations can be constructed for a plethora of applications.

They are easy to implement, and demand low computational power. Their main usages are most importantly robot localization [?] and SLAM [?].

However, its numeric solution method (approximate probability distributions by samples) leaves a lot of room for errors. Most importantly, the particle filter can suffer from the so called *curse of dimensionality*, which is described in [?] as "One must expect N (the number of particles) to rise rapidly with the dimension of the space (...) It is most difficult to make any precise provable statement on the crucial question of how many samples are required to give a satisfactory representation of the densities for filter operation". With a good proposal distribution, this unwanted property can be avoided. Daum and Huang in [?] showed that the BPF suffers from this phenomena, while for example the unscented particle filter does not.

The curse of dimensionality problem is connected to the so called particle degeneracy phenomenon. It means that only a low amount of samples estimates the target posterior efficiently; others are in regions with very low probability. This occurs when the proposal distribution and the target distribution are not matched well. Figure 9. shows a setup which demonstrates this effect: the prior distribution (the predicted belief in the case of the BPF) is colored in blue, and represented by its particles, shown as black dots. The weighting of the particles is done according to the likelihood (red curve), which is the

Algorithm 3 Bootstrap particle filter $(\mathcal{X}_{t-1}, \mathbf{u}_t, \mathbf{z}_t)$

```
 1: $\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
 2: for $n = 1$ to $N$ do
 3:     sample $\mathbf{x}_t^{[n]} \sim p(\mathbf{x}_t | \mathbf{x}_{t-1}^{[n]}, \mathbf{u}_t)$
 4:     $w_t^{[n]} = p(\mathbf{z}_t | \mathbf{x}_t^{[n]}) w_{t-1}^{[n]}$
 5: end for
 6: calculate weight sum $\hat{w}_t = \sum_{i=1}^{N} w_t^{[i]}$
 7: for $n = 1$ to $N$ do
 8:     normalize $w_t^{[n]} = w_t^{[n]} \hat{w}_t^{-1}$
 9:     $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle \mathbf{x}_t^{[n]}, w_t^{[n]} \rangle$
10: end for
11: calculate $N_{\text{eff}}$ using (2.17)
12: if $N_{\text{eff}} < N_T$ then
13:     for $n = 1$ to $N$ do
14:         draw $i$ with probability $\propto w_t^{[i]}$
15:         $\mathcal{X}_t = \mathcal{X}_t + \langle \mathbf{x}_t^{[i]}, N^{-1} \rangle$
16:     end for
17:     return $\mathcal{X}_t$
18: else
19:     $\mathcal{X}_t = \overline{\mathcal{X}}_t$
20:     return $\mathcal{X}_t$
21: end if
```

measurement model in the BPF. As a result, only 3 particles are going to have weights other, than 0. Upon resampling, only these 3 particles will represent the target posterior; the others are eliminated. (Of course the size of the particle set will not change, only that they are distributed across 3 distinct positions). By common sense, a probability distribution is hardly represented properly by 3 particles. Counter-intuitively, having a more precise sensor (i.e. having a ”thinner” likelihood function) only makes this worse.

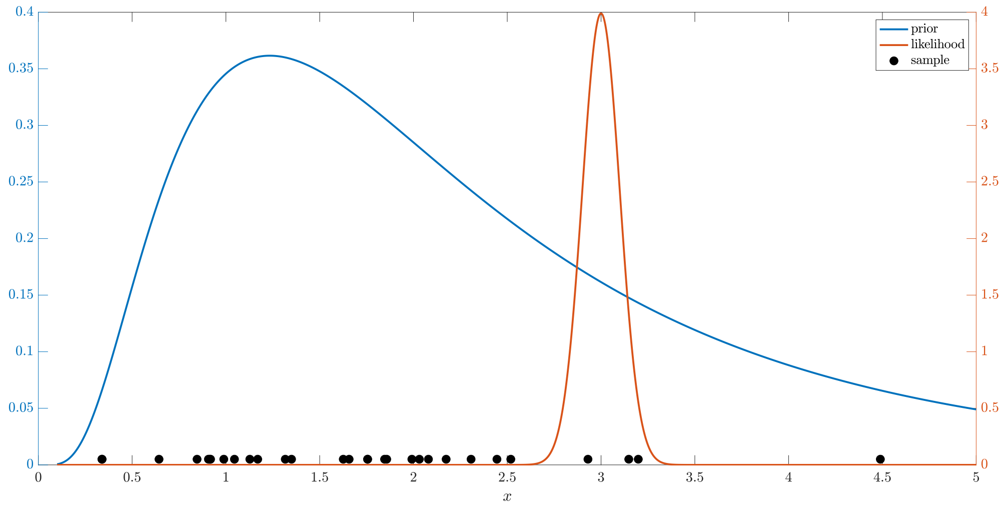


Fig. 9: Particle degeneracy, shown in a one-dimensional example with 30 particles.

There are three possible ways to minimalize particle degeneracy: the most obvious one is to have more particles (populate the state space more densely). However, in higher dimensions this demands exponentially more particles. Another solution is to use a different proposal density, therefore having a different likelihood function. In this way the target distribution could be more effectively represented by particles. The third one is resampling: having more particles on top of each other in the same place is better than having only a few of them, while all the others are spread across the domain with negligible weights.

Although particle filters demand low computational power, this only holds when the algorithm is properly tuned, therefore requires the minimal amount of particles. A poor choice of a particle filter, or a poorly designed particle filter requires a lot of computational capacity to give proper results: the particle requirement can be in the order of millions. As were mentioned in [**?**], “for typical low dimensional tracking problems, the PF requires 2 to 6 orders of magnitude more computer throughput than the extended Kalman filter, to achieve the same accuracy.”

2.6 The Daum–Huang filters

2.6.1 Main concepts

To overcome the undesirable properties of the particle filters (especially the curse of dimensionality), Daum and Huang proposed a new method, where the state estimation is based on log-homotopy based particle flow [?]. This new filtering method is called Daum-Huang filter (DHF) [?].

In particle filters, the problem lies where the Bayes' law is performed to obtain the posterior (this is done though weighting). Upon resampling, the particles "jump" from the prior to the posterior. The main concept behind the DHF is to eliminate this often faulty "jump", and handle it as a continuous movement. Consider a homotopy in λ, based on the logarithmic form of the Bayes' law:

$$\log p(\mathbf{x}, \lambda) = \log g(\mathbf{x}) + \lambda \log h(\mathbf{x}) - \log K(\lambda), \tag{2.18}$$

where $g(.)$ is the prior, $h(.)$ is the likelihood, $K(.)$ is a normalization constant, and λ is the parameter of the homotopy: $\lambda \in [0, 1]$. As λ changes continuously form 0 to 1, $p(.)$ changes from being the prior $g(.)$, to becoming the posterior, as $\lambda = 1$ (this gives back the standard Bayes' rule). This is essentially the time evolution (in this case λ is *pseudotime*) of $p(.)$. Meanwhile the particles that represent the probability distribution also have to move: this movement is described by a stochastic law of motion, see (**??**) with the modification, that $t := \lambda$. The question is that how to move these particles according to the also evolving probability distribution, from which they were sampled.

The evolution of a probability density under a governing SDE is described by the Fokker–Planck equation, if $\mathbf{x} \in \mathbb{R}^d$:

$$\frac{\partial p(\mathbf{x}, \lambda)}{\partial \lambda} = -\sum_{i=1}^{d} \frac{\partial}{\partial x_i} \left[a_i(\mathbf{x}, \lambda) p(\mathbf{x}, \lambda)\right] + \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial^2}{\partial x_i \partial x_j} \left[B_{i,j}(\mathbf{x}, \lambda) p(\mathbf{x}, \lambda)\right], \tag{2.19}$$

where

$$\mathbf{B} = \frac{1}{2} \mathbf{b} \mathbf{b}^\top. \tag{2.20}$$

Now if (2.18) is partially derivated according to λ, it can be substituted into the left hand side of (2.19). After some simplification, the flow equation in its final form:

$$\begin{aligned} \log h(\mathbf{x}) - \frac{\partial \log K(\lambda)}{\partial \lambda} &= -\mathbf{a}^\top(\mathbf{x}, \lambda) \cdot \nabla \log p(\mathbf{x}, \lambda) - \nabla \cdot \mathbf{a}(\mathbf{x}, \lambda) \\ &+ \frac{1}{2p(\mathbf{x}, \lambda)} \left(\nabla^\top \mathbf{B}(\mathbf{x}, \lambda) p(\mathbf{x}, \lambda) \nabla\right). \end{aligned} \tag{2.21}$$

The main difference between different DHF realizations is the method to solve (2.21) for $\mathbf{a}(.)$. This is still open question, and holds many room for innovation even to this day. Here, only the exact flow simplification [?] is discussed, where the diffusion term ($\mathbf{B}$) is neglected.

2.7 The Exact Flow Daum–Huang Filter

Under the criteria, that the process noise $\mathbf{v}$, and the measurement noise $\mathbf{w}$ belong to the exponential family of probability distributions, the solution for (2.21) can be expressed as:

$$\mathbf{a}(\mathbf{x}, \lambda) = \mathbf{C}(\lambda)\mathbf{x} + \mathbf{c}(\lambda), \tag{2.22}$$

where

$$\mathbf{C}(\lambda) = -\frac{1}{2}\overline{\boldsymbol{\Sigma}}\mathbf{H}^\top \left(\lambda \mathbf{H}\overline{\boldsymbol{\Sigma}}\mathbf{H}^\top + \mathbf{R}\right)^{-1} \mathbf{H}, \tag{2.23}$$

$$\mathbf{c}(\lambda) = (\mathbf{I} + 2\lambda\mathbf{C}) \left[(\mathbf{I} + \lambda\mathbf{C})\, \overline{\boldsymbol{\Sigma}}\mathbf{H}^\top\mathbf{R}^{-1}\mathbf{z} + \mathbf{C}\overline{\mathbf{x}}\right]. \tag{2.24}$$

Here, $\overline{\mathbf{x}}$ denotes the state average at $\lambda = 0$ across all the particles. The matrix $\mathbf{H}$ is the first order Taylor expansion of the measurement matrix, performed around the current state average at a given λ. The pseudocode of the exact flow Daum–Huang filter (EDH) can be found in [**?**], Algoritm 1. However, the method is also summarized here.

In one timestep, first the particles are propagated just like in the case of the PF, then $\overline{\boldsymbol{\Sigma}}$ is calculated using the prediction part of an EKF/UKF. After this, the homotopy is discretized: λ is equally divided into a certain amount of steps, e.g. 10. For each *pseudotime* step, $\mathbf{H}_t$ is calculated by linearization around the current state average of the particles, then using the predicted covariance $\overline{\boldsymbol{\Sigma}}_t$, the measurement $\mathbf{z}_t$, the measurement noise covariance $\mathbf{R}_t$, and the initial ($\lambda = 0$) state average $\overline{\mathbf{x}}_t$, $\mathbf{C}(\lambda)$ and $\mathbf{c}(\lambda)$ is calculated. By this, the drift function $\mathbf{a}(.)$ is obtained. Recall (**??**), where the diffusion term is assumed to be 0. In this case, a simple numerical integration yields the new state of the particles. This is repeated for all λ. It is important to mention that during the homotopy, the time does not change, only the pseudotime. After λ reached 1, the particles are in the position, where they estimate the posterior distribution. Now, the state estimation at t can be calculated by taking the average of the particles. Finally, the EKF/UKF update is performed. This was one time step (and along with it, 10 pseudotime step) of the EDH filter.

Figure 10. illustrates the particle flow of the DHF. The time is fixed, and the homotopy is performed from $\lambda = 0$ to $\lambda = 1$. At $\lambda = 1$, the particles moved from the predicted belief (prior, in this context) to their position where they represent the posterior. The true state is indicated by a red square, while the orange dot is the average of the particles. The red arrows show the value of the drift vector $\mathbf{a}(\mathbf{x}, \lambda)$. As λ evolves, the particle average gets closer and closer to the (unknown) true position.

2.7.1 Other Daum–Huang Filter realizations

Solving (2.21) requires various simplifications. Assuming exact flow, where the diffusion part is neglected (only deterministic flow is considered), the measurement model is linear (or linearized), and the noise acting on the system is a Gaussian random variable (or

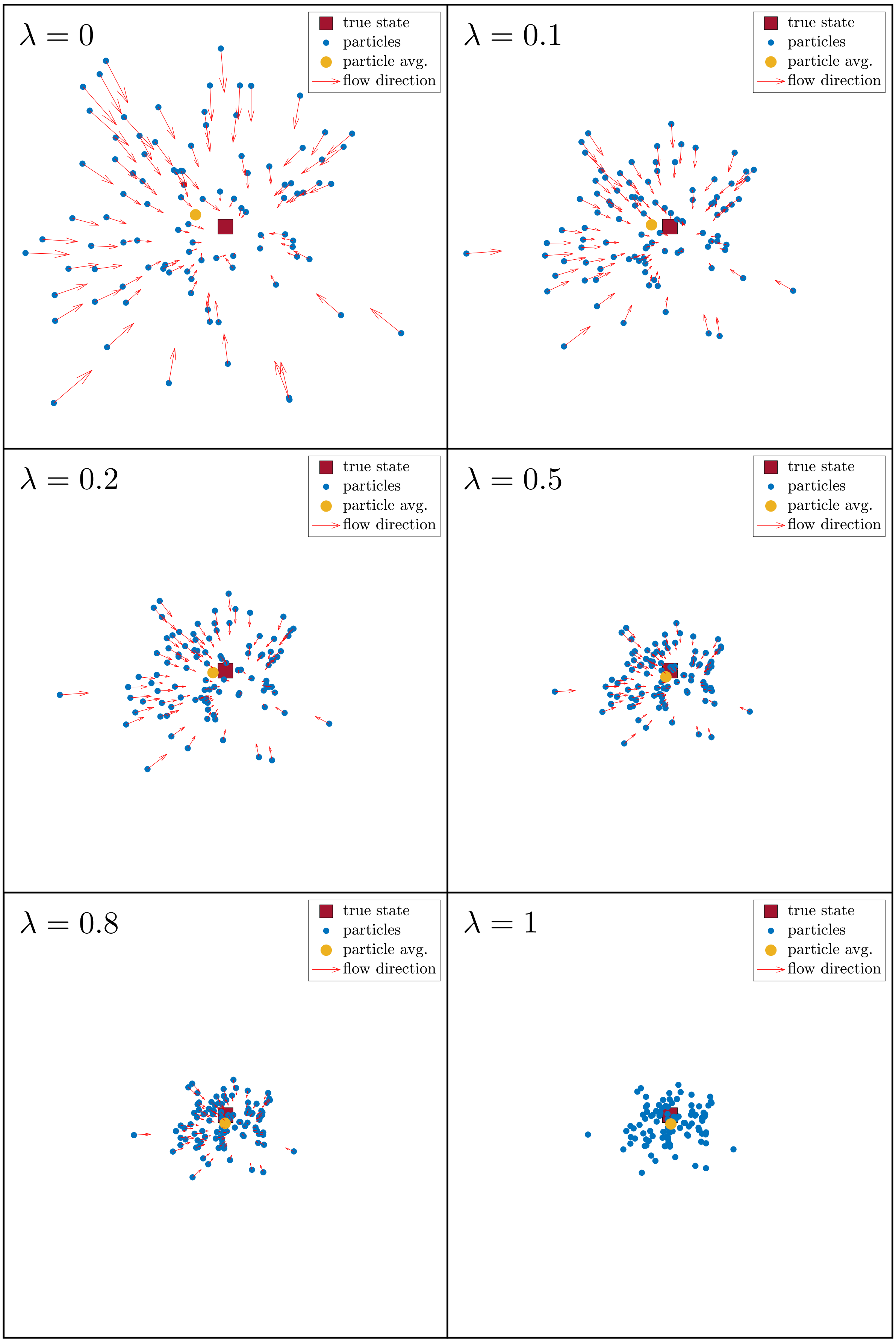


Fig. 10: The particle flow of the DHF (calculated using the exact flow simplification). The time is at a fixed value, while the homotopy is performed from $\lambda = 0$ to $\lambda = 1$.

from the exponential family) is only one variation of the Daum–Huang filter. When the linearization of the measurement function is performed around the location of *each* particle instead of their average, the local exact flow Daum–Huang filter is constructed [**?**]. Particle weighting can be developed for Daum–Huang filters as well, proposed in [**?**].

Another simplification is the assumption of incompressible flow [**?**] (this was the first type of the Daum–Huang filter), the assumption of geodesic flow [**?**], and the latest results are from the consideration of stochastic particle flow (the diffusion term is no longer neglected), using Gromov's method [**?**].

It is difficult to summarize the performance of Daum–Huang filters, so is the case with particle filters. Many realizations exist, each with they own advantages and disadvantages. The main problem is with computational capacity: a method with much more advanced and/or more precise calculations can yield better results at a cost of computational time. The main challenge is to develop algorithms and variations that does not require significantly more computation, yet result in a more stable and precise estimation of the state.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* Cambridge, Mass.: MIT Press, 2005. [Online]. Available: http://www.amazon.de/gp/product/0262201623/102-8479661-9831324?v=glance&n=283155&n=507846&s=books&v=glance

[2] M. Fazekas, P. Gáspár, and B. Németh, "Calibration and improvement of an odometry model with dynamic wheel and lateral dynamics integration," *Sensors (Switzerland)*, vol. 21, pp. 1–29, 1 2021.

[3] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 116–121, 1985.

[4] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, pp. 34–46, 2 2007.

[5] A. I. Eliazar and R. Parr, "Learning probabilistic motion models for mobile robots," *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, pp. 249–256, 2004.

[6] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 376–382, 1991.

[7] L. Dantanarayana, R. Ranasinghe, and G. Dissanayake, "C-log: A chamfer distance based method for localisation in occupancy grid-maps," 2013, pp. 376–381.

[8] L. Dantanarayana, G. Dissanayake, R. Ranasinghe, and T. Furukawa, “An extended kalman filter for localisation in occupancy grid maps.” Institute of Electrical and Electronics Engineers Inc., 2 2016, pp. 419–424.

[9] L. Dantanarayana, “Navigation and control for assistive robotics,” 2016.