Activity 4

Line-based Extended Kalman Filter for robot localization

1. Introduction

As pointed out in the previous activity, knowledge of the location of a platform is essential for a lot of robotics applications, and we motivated this with an autonomous vehicle hauling goods inside a warehouse from one place to another. Within this scenario, Activity 3 demonstrated how to acquire a more abstract representation of linear structures perceived with a scanning range finder. This activity will show that —given a map of the linear features in this representation— the robot can localize itself based on the linear structures it perceives.

2. Kalman Filter Localization

The Extended Kalman Filter used for localization in this assignment can be structured into a prediction step and an update step. In the following, we will look at the two steps separately.

2.1. State Prediction

Exercise 2 discussed the motion model for a differential drive robot. Given knowledge of the state $\boldsymbol{x}_{t-1} = [x_{t-1}, y_{t-1}, \theta_{t-1}]^T$ at the previous time step and of the wheel displacements $\boldsymbol{u}_t = [\Delta s_l , \Delta s_r]^T$, the motion model can be employed to obtain an *a priori* estimate of the current state.

$$\hat{\boldsymbol{x}}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) = \boldsymbol{x}_{t-1} + \begin{bmatrix} (\Delta s_l + \Delta s_r)/2.\cos(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2l) \\ (\Delta s_l + \Delta s_r)/2.\sin(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2l) \\ (\Delta s_l - \Delta s_r)/l \end{bmatrix} \quad (1)$$

Where l is the inter wheels distance, erroneously called b in [1] in the Kalman filter based localization chapter.

It is reasonable to assume that the motion is subject to noise, which we choose to model as additive Gaussian noise $v \sim N(0, \mathbf{Q})$ applied to the control inputs. As proposed in [1, p. 272], the noise on the control inputs can be modeled as independent for each wheel with a covariance proportional to the absolute value of the travelled distance, where a constant factor k is used to account for any non-deterministic effects.

$$\boldsymbol{Q} = \begin{bmatrix} k|\Delta s_l| & 0 \\ 0 & k|\Delta s_l| \end{bmatrix} \quad (2)$$

Hence, an *a priori* estimate of the covariance of the state can be computed as

$$\widehat{\boldsymbol{P}}_{\boldsymbol{t}} = \boldsymbol{F}_x.\boldsymbol{P}_{t-1}.\boldsymbol{F}_x{}^T + \boldsymbol{F}_u.\boldsymbol{Q}_t.\boldsymbol{F}_u{}^T \quad (3)$$

where $\mathbf{F}_x$ and $\mathbf{F}_u$ denote the Jacobians of the motion model $f\,(\mathbf{x}_{t-1}, \mathbf{u}_t)$ with respect to the state estimate and the control inputs respectively. See also [1, p. 270-272, 337].

Task: Write a Matlab function $[\widehat{\boldsymbol{x}}_t, \widehat{\boldsymbol{F}}_x, \widehat{\boldsymbol{F}}_u]$= *transitionFunction*($\mathbf{x}_{t-1}$, $\mathbf{u}_t$, l) that accepts the previous state $\boldsymbol{x}_{t-1}$ of a differential drive robot as well as control inputs $\mathbf{u}_t$ and the inter wheel distance l as arguments and computes an estimate of the current state $\boldsymbol{x}_t$ as well as the Jacobians of the state transition function with respect to the state $\widehat{\boldsymbol{F}}_x$ and the control inputs $\widehat{\boldsymbol{F}}_u$ respectively.

Validation: Run the function *validateTransitionFunction()*. This function uses a sequence of control inputs to propagate the state with the supplied motion model. The function reports on the correctness of your implementation. If your implementation is correct, the function plots a ground truth path as well as the forward integration of noisy control inputs using your motion model. You should observe that the ground truth path and your estimate diverge increasingly over the course of the experiment. This illustrates that for many real-world applications where perturbations occur, relying solely on interoceptive information is insufficient.

2.2. State Update

As illustrated in the previous experiment, perturbations in the control inputs will result in an increasingly inaccurate estimate of the state of the robot. Hence, exteroceptive location cues are commonly employed in robotics application. In this exercise, the robot is capable of sensing linear structures and possesses a map $\mathbf{M}$, which contains all linear structures in its operating environment, expressed in a coordinate frame that will be referred to as *world coordinate frame*.

2.2.1. Measurement Function

As introduced in Exercise 3, lines can be parametrized as $\mathbf{m}^i = [\, a^i \,, r^i\,]^{\mathrm{T}}$. This parametrization will be applied to both, the output of our perception system z_t as well as the entries of the map $\mathbf{M}$. Note however, that while the parametrization is identical, the coordinate frames in which the measurements and the map are represented differ. While lines in the map are represented in the world coordinate frame, the robot senses lines in its *body coordinate frame* relative to its own, varying pose. Hence, the measurement can be modeled by transforming the lines in the map from the world coordinate frame into the body coordinate frame. A more detailed description of this transformation is given in [1, p. 338-340]. For the remainder of this exercise, a map $\mathbf{M}$ with K entries is represented by a 2 x K matrix by horizontally concatenating individual $\mathbf{m}^i$.

Task: Write a Matlab function $[\hat{\boldsymbol{z}}_t, \widehat{\boldsymbol{H}}_x]$= *measurementFunction*($\widehat{\boldsymbol{x}}_t$, $\mathbf{m}^i$) that accepts an *a priori* estimate of the state $\widehat{\boldsymbol{x}}_t$and a map entry $\mathbf{m}^i$ and that models a measurement $\hat{\boldsymbol{z}}_t$ as it would be perceived by a robot with state $\widehat{\boldsymbol{x}}_t$, which constitutes the transformation from a line expressed in the world coordinate frame into the body coordinate frame of the robot. Additionally, compute the Jacobian of the measurement model $\widehat{\boldsymbol{H}}_x$ with respect to the state.

Validation: Run the function *validateMeasurementFunction()*. The function reports on the correctness of your implementation.

2.2.2. Measurement Association

In order to apply the Kalman filter update correctly, associations between observations and map entries need to be established. To this end we employ the Mahalanobis distance between a predicted measurement $\hat{\boldsymbol{z}}_t^i$ and an observation $\boldsymbol{z}^j$. With the *innovation* $\boldsymbol{v}_t^{ij}$ as a measure of the difference between a predicted and observed measurement

$$\boldsymbol{v}_t^{ij} = \boldsymbol{z}_t^j - \hat{\boldsymbol{z}}_t^i \tag{4}$$

And the innovation covariance $\boldsymbol{\Sigma}_{IN_t}^{ij}$

$$\boldsymbol{\Sigma}_{IN_t}^{ij} = \widehat{\boldsymbol{H}}_t^i . \widehat{\boldsymbol{P}}_t . \widehat{\boldsymbol{H}}_t^{iT} + \boldsymbol{R}_t^j \tag{5}$$

The Mahalanobis distance is calculated as

$$\mathrm{d}_t^{ij} = \boldsymbol{v}_t^{ijT} . \left(\boldsymbol{\Sigma}_{IN_t}^{ij}\right)^{-1} . \boldsymbol{v}_t^{ij} \tag{6}$$

In real-world robotics application there will always be corrupting measurements that do not correspond to entries in the map. In the example presented in the introduction, it could be a previously closed door that was opened or furniture that got moved around. Hence, we introduce a *validation gate* g and only consider associations that fall into this gate $\mathrm{d}_t^{ij} < g^2$. When multiple map entries fall into the validation gate of a single measurement, the measurement is associated with the entry with the smallest Mahalanobis distance. On the other hand, multiple measurements may be associated with a single map entry. Please find additional information in [1, p. 334-335, 340-342].

Task: Understand the implementation (*no coding to do in this task!!!*, just understanding the provided implementation) of the function $[\widehat{\boldsymbol{v}}_t, \widehat{\mathbf{H}}_t, \mathbf{R}_t]$ = *associateMeasurements*$(\widehat{\boldsymbol{x}}_t, \widehat{\boldsymbol{P}}_t, \boldsymbol{Z}_t, \boldsymbol{R}_t, \mathbf{M}, g)$ that accepts the *a priori* state estimate $\widehat{\boldsymbol{x}}_t$ and its covariance $\widehat{\boldsymbol{P}}_t$, N measurements $\boldsymbol{Z}_t$ expressed as a 2 x N matrix and their covariances $\boldsymbol{R}_t$ expressed as 2 x 2 x N tensor, as well as the map $\mathbf{M}$, and a scalar validation gate g. The function returns a 2 x K matrix $\widehat{\boldsymbol{v}}_t$, of innovations $\boldsymbol{v}_t^{ij}$ where K denotes the number of successfully matched line features, as well as a 2 x 3 x K tensor $\widehat{\mathbf{H}}_t$ of the Jacobians of the measurement function in the same order and a 2 x 2 x K tensor $\mathbf{R}_t$ of the corresponding measurement uncertainties. Although the focus of this function is on the correct association of the perceived lines with the map, the outputs are defined to facilitate the EKF implementation in the next task and in turn to avoid duplications of computations.

Validation: Run the function *validateAssociations*(). NB: this will not work if the previous validation tests in 2.1 and 2.2.1 did not succeed.

2.3. Update the Estimate

The previous tasks provided the essential building blocks for implementing the Extended Kalman Filter updates according to the well established equations. Information on the Extended Kalman Filter can either be found in [1, p. 335-336] or in [2].

Task: Write a Matlab/Octave function $[\boldsymbol{x}_t, \boldsymbol{P}_t]$ = *filterStep*$(\boldsymbol{x}_{t-1}, \mathbf{P}_{t-1}, \mathbf{u}_t, \mathbf{Z}_t, \mathbf{R}_t, \mathbf{M}, g, l)$ that accepts the previously introduced quantities and that performs a single, complete filter step. Consider using the function *blockDiagonal*$(\mathbf{R})$ provided with this exercise to reshape the measurement covariances appropriately, as well as the Matlab/Octave function *reshape* and *permute* to reshape the outputs of the previously implemented functions to match the EKF equations.

Validation: Run the function *validateFilter*(). The function will load a set of corrupted measurements and iterate your filter over a sequence of perturbed control inputs. For feedback, ground truth motion, the output of a baseline implementation, the output of your filter and forward integration of control inputs are displayed. As already observed in Section 2.1, the solution that does not take exterioceptive information into account diverges quickly from the true path. Your filter solution however will follow the true path more accurately, as it corrects its state estimation with information from an additional sensor and an accurate map. As identical input data and assumptions about the statistical properties of the perturbations are employed, your results should in theory align perfectly with those of the baseline implementation. However, numerical differences might introduce small derivations of the two solutions.

3. V-REP Experiment

So far, line extraction and EKF localization have been implemented and verified separately. In this exercise, we will combine them to implement the complete functionality and test it in the simulation environment V-REP.
Task: Write a Matlab/Octave function $[\boldsymbol{x}_t, \mathbf{P}_t]=$ *incrementalLocalization* $(\boldsymbol{x}_{t-1}, \mathbf{P}_{t-1}, \mathbf{u}_t, \mathbf{S}_t, \mathbf{M}, params, k, l, g)$ that takes the previous pose, control inputs and the laser scan data $\mathbf{S}$ as arguments and returns an *a posterori* estimate the pose of the robots and its covariance. NB: $\mathbf{S}$ is a 2 x n matrix, where n is the number of points in the laser scan and where the 1st line of the matrix a list of the n laser angles and the 2nd the list of the n detected distances.

Validation: Start V-REP, load scene scene/Exercise4.ttt and start the simulation. You should see the p3-dx robot in the lab's experimentation environment and the visualization of laser measurements. Now run the script *vrepSimulation*. The robot should start moving on a circular path. Close to the actual robot you should see a yellow 'ghost', which visualizes the pose as estimated by your localization. Just like in real robotics applications, you might find that the localization does not work flawlessly. As both faces of each wall are entries to the map and as line features are solely associated by Mahalanobis distance, the measurements may be incorrectly associated with the opposite face of the wall, visible as a bias in the localization. You may also move the starting position in the simulation environment and re-run the simulation. Depending on the number and constellation of observed walls, the observability of the global pose may be affected, resulting in impaired state estimates.