ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab

Exercise 4
Line-based Extended Kalman Filter for robot localization

1 Introduction

As pointed out in the previous exercise, 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, Exercise 3 demonstrated how to acquire a more abstract representation of linear structures perceived with a scanning range finder. This exercise 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.

This exercise closely follows the example given in [1, p. 331-342]. We **strongly** advice you to read the respective pages prior to attending the exercise session and to turn to this document for advice throughout the exercise. Please note that there are different approaches to implementing Jacobians. While deriving the functions encountered in this exercise is straightfoward, those of you familiar with Matlab's Symbolic Math Toolbox/MuPAD may find that a combination of Matlab's anonymous functions with calls to `sym`, `jacobian` and `matlabFunction` can increase efficiency in solving this problem set. Please also consider turning to [1] for help with the Jacobians.

2 Kalman Filter Localization

The Extended Kalman Filter used for localization in this exercise 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 $\mathbf{x}_{t-1} = [x_{t-1}, y_{t-1}, \theta_{t-1}]^T$ at the previous time step and of the wheel displacements $\mathbf{u}_t = [\Delta s_l, \Delta s_r]^T$, the motion model can be employed to obtain an *a priori* estimate of the current state. Here, we follow the order of $\mathbf{u}_t$ established on [1, p. 337], which conflicts with the order stated on [1, p. 272]. Please consider any implications of this reversed order on the implementation of the function and its Jacobians.

$$\hat{\mathbf{x}}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) = \mathbf{x}_{t-1} + \begin{bmatrix} (\Delta s_l + \Delta s_r)/2 \cdot cos\,(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2b) \\ (\Delta s_l + \Delta s_r)/2 \cdot sin\,(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2b) \\ (\Delta s_r - \Delta s_l)/b \end{bmatrix} \tag{1}$$

It is reasonable to assume that the motion is subject to noise, which we choose to model as additive Gaussian noise $\nu \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ applied to the control inputs. As proposed in [1, p. 272],

the noise on the control inputs can be modelled 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.

$$\mathbf{Q} = \begin{bmatrix} k|\Delta s_l| & 0 \\ 0 & k|\Delta s_r| \end{bmatrix} \tag{2}$$

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

$$\hat{\mathbf{P}}_t = \mathbf{F_x} \cdot \mathbf{P}_{t-1} \cdot \mathbf{F_x}^T + \mathbf{F_u} \cdot \mathbf{Q}_t \cdot \mathbf{F_u}^T \quad , \tag{3}$$

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: Derive the Jacobians $\hat{\mathbf{F}}_\mathbf{x}$ and $\hat{\mathbf{F}}_\mathbf{u}$ of the state transition function with respect to the state and the control inputs. To validate your solution, implement it in the Matlab/Octave function $[\hat{\mathbf{x}}_t, \hat{\mathbf{F}}_\mathbf{x}, \hat{\mathbf{F}}_\mathbf{u}] = transitionFunction(\mathbf{x}_{t-1}, \mathbf{u}_t, b)$ that accepts the previous state $\mathbf{x}_{t-1}$ of a differential drive robot as well as control inputs $\mathbf{u}_t$ and the inter wheel distance b as arguments.

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 $\mathbf{m}^i = \left[\alpha^i, r^i\right]^T$. 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 modelled 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 $2 \times K$ matrix by horizontally concatenating individual $\mathbf{m}^i$.

Task: Derive the measurement model $\hat{\mathbf{z}}_t = h(\hat{\mathbf{x}}_t, \mathbf{m}^{(i)})$ that describes a line $\mathbf{m}^{(i)}$ as perceived in the *body coordinate frame*. Derive the Jacobian of the measurement model $\hat{\mathbf{H}}_\mathbf{x}$ with respect to the state. To check your solution, implement it in the Matlab/Octave function $[\hat{\mathbf{z}}_t, \hat{\mathbf{H}}_\mathbf{x}] = measurementFunction(\hat{\mathbf{x}}_t, \mathbf{m}^i)$ that accepts an *a priori* estimate of the state $\hat{\mathbf{x}}_t$ and a map entry $\mathbf{m}^i$.

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 $\hat{\mathbf{z}}_t^i$ and an observation $\mathbf{z}^j$. With the *innovation* $\mathbf{v}_t^{ij}$ as a measure of the difference between a predicted and observed measurement

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

and the *innovation covariance* $\mathbf{\Sigma}_{IN_t}^{ij}$

$$\mathbf{\Sigma}_{IN_t}^{ij} = \hat{\mathbf{H}}_t^i \cdot \hat{\mathbf{P}}_t \cdot \hat{\mathbf{H}}_t^{iT} + \mathbf{R}_t^j \tag{5}$$

the Mahalanobis distance is calculated as

$$d_t^{ij} = \mathbf{v}_t^{ijT} \cdot \left(\mathbf{\Sigma}_{IN_t}^{ij}\right)^{-1} \cdot \mathbf{v}_t^{ij} \quad . \tag{6}$$

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 below this threshold $d_t^{ij} < g^2$, i.e. the distance between the observed line and the line in the map is not too big. 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: Write a Matlab/Octave function $[\hat{\mathbf{v}}_t, \hat{\mathbf{H}}_t, \mathbf{R}_t] = associateMeasurements(\hat{\mathbf{x}}_t, \hat{\mathbf{P}}_t, \mathbf{Z}_t, \mathbf{R}_t, \mathbf{M}, g)$ that accepts the *a priori* state estimate $\hat{\mathbf{x}}_t$ and its covariance $\hat{\mathbf{P}}_t$, N measurements $\mathbf{Z}_t$ expressed as a $2 \times N$ matrix and their covariances $\mathbf{R}_t$ expressed as $2 \times 2 \times N$ tensor, as well as the map $\mathbf{M}$, and a scalar validation gate g. The function returns a $2 \times K$ matrix $\hat{\mathbf{v}}_t$, of innovations $\mathbf{v}_t^{ij}$ where K denotes the number of successfully matched line features, as well as a $2 \times 3 \times K$ tensor $\hat{\mathbf{H}}_t$ of the Jacobians of the measurement function in the same order and a $2 \times 2 \times K$ tensor $\mathbf{R}_t$ of the corresponding measurement uncertainties. Although the focus of this exercise 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*(). The function reports on the correctness of your implementation.

2.3 Updating 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 $[\mathbf{x}_t, \mathbf{P}_t] = filterStep(\mathbf{x}_{t-1}, \mathbf{P}_{t-1}, \mathbf{u}_t, \mathbf{Z}_t, \mathbf{R}_t, \mathbf{M}, g, b)$ that accepts the previously introduced quantities and that performs a single, complete filter step, consisting of state propagation, measurement association and a subsequent state update. 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 noise 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 deviations of the two solutions.

3 CoppeliaSim 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 CoppeliaSim.

Task: Write a Matlab/Octave function $[\mathbf{x}_t, \mathbf{P}_t] = incrementalLocalization(\mathbf{x}_{t-1}, \mathbf{P}_{t-1}, \mathbf{u}_t, \mathbf{S}_t, \mathbf{M}, params, k, b, g)$ 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.

Validation: Start CoppeliaSim, load scene **`scene/mooc_exercises.ttt`** and start the simulation. You should see a circular robot, a set of walls and the visualization of laser measurements. Now run the script `vrepSimulation`. The robotic platform 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.

References

[1] Roland Siegwart, Illah Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2nd edition, 2011.

[2] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.