Exercise 5
EKF Simultaneous Localization And Mapping (SLAM)

1 Introduction

This exercise extends and builds on the results of Exercise 4. It is therefore important to finish that exercise first. With landmark-based localization from Exercise 4 relying on an a priori known map, this exercise focuses on autonomous map building, while exploring the environment. This is described in detail in [1] in chapter 5.8.

2 EKF SLAM

Review Kalman Filter Localization and Line Parametrization from Exercise 4.

The idea behind EKF SLAM is to treat the map in a probabilistic way, similar to treating the pose (x, y, θ) of the robot. Therefore, we add the line parameters α^i, r^i of the landmark m^i to the EKF state:

$$\boldsymbol{x} = \begin{bmatrix} x & y & \theta & \alpha^1 & r^1 & \dots & \alpha^i & r^i \end{bmatrix}^T \tag{1}$$

We assume that the landmarks do not have any dynamics, i.e. they are static in the world frame. By properly including the landmarks in the state vector, we can now obtain the uncertainty of the landmarks as well as the correlations between them. For the implementation of the SLAM-filter, we assume that we know the number of landmarks and have a (very rough) estimate of them at startup. This simplifies the implementational details significantly, as no landmark book-keeping is necessary.

2.1 Time Update (propagation)

Exercise 3 dealt with the state propagation model of the robot when not including the landmarks in the state. As these landmarks are now part of the state vector, we have to adjust the state propagation model. Since the landmarks are considered to be static, we model them using zero velocity dynamics:

$$\dot{\alpha} = 0 \tag{2}$$

$$\dot{r} = 0 \tag{3}$$

Task: Derive and implement the state prediction and covariance propagation for the state defined above, by re-using your results from Exercise 4. To this end, adapt the state transition function in `transitionFunction.m`. The state transition function maps the state at time $\boldsymbol{x}_{t-1}$ to

the state $\hat{\boldsymbol{x}}_t$ given the wheel odometry $\boldsymbol{u}_t$

$$\hat{\boldsymbol{x}}_t = f\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t\right); \tag{4}$$

Additionally, derive the Jacobians $\boldsymbol{F}_x$ and $\boldsymbol{F}_u$ for this formulation and implement your result in the file `transitionFunction.m`.

$$\boldsymbol{F}_x = \frac{\partial f\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t\right)}{\partial \boldsymbol{x}}; \quad \boldsymbol{F}_u = \frac{\partial f\left(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t\right)}{\partial \boldsymbol{u}} \tag{5}$$

(Hint: What has changed since Exercise 4?)

Validation: Validate your implementation using the validation script `test/validateTransitionFunction.m`. If your implementation of the state transition and covariance propagation is correct, the message you should see is: `State transition function appears to be correct!`.

2.2 Measurement Update

The main work of EKF SLAM is done in the measurement update. We assume that all landmarks are visible, and that the number of landmarks does not change. This is not a limitation, but it simplifies implementation and debugging effort at this stage. Measurement association and update work similarly to Exercise 4, except that the measurement Jacobians $\boldsymbol{H}^i$ need special attention.

Like in Exercise 4the relative position and heading of the robot with respect to the landmarks can be estimated. However, in this exercise the ground truth positions of the landmarks are not known. Therefore, in a global fixed reference frame we can not know whether the estimated position of the robot or of the landmarks is wrong and should be adjusted. This can be intuitively explained by the fact that if both robot and landmarks are moved or rotated by the same amount simultaneously, we cannot detect this motion using only relative distance or bearing measurements of the robot with respect to the walls or wheel encoder information. To eliminate these degrees of freedom, we fix the first two landmarks in the state vector (assuming that both walls are not parallel), in order to fully constrain the unobservable states. In other words, we arbitrarily choose a global fixed frame of reference in the filter state, defined by these two lines.

Task: Based on Exercise 4, implement a measurement function h that returns the predicted measurement given the state vector (which now includes both robot state and landmarks) and the landmark index as defined in Eq. (6). To this end, update the script `measurementFunction.m`. (Hint: What has actually changed since the previous exercise?).

$$\boldsymbol{m} = h(\boldsymbol{x}_t) \tag{6}$$

Additionally, provide the corresponding Jacobian of the measurement with respect to the state vector. Compared to Exercise 4, the landmark position is now part of the state vector. This has to be reflected in the computation of the Jacobian.

$$\boldsymbol{H}_{x,i} = \frac{\partial h_i\left(\boldsymbol{x}_t\right)}{\partial \boldsymbol{x}}; \tag{7}$$

Validation: Validate your implementation using the validation script `test/validateMeasurement-Function.m`. If your implementation of the measurement update is correct, the message is: `measurement function appears to be correct!`.

2.3 Validation of Complete EKF SLAM System

With the state propagation and measurement update working, it is now time to validate the complete EKF SLAM setup. First, run the script `test/validateFilter.m`. This will evaluate whether your filter will provide the same output as the baseline implementation. To evaluate your filter using randomly generated input values, you can run the script `test/validateCompleteEKFLoop.m`. The output of your filter should now approximate the ground truth trajectory.

3 CoppeliaSim Experiment

Task: So far, EKF localization and mapping have been implemented and verified in Matlab. In this exercise, we evaluate and test the complete functionality in the simulation environment CoppeliaSim.

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 `vrep/vrepSimulation.m`. 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. If everything is implemented properly, you should see the yellow ghost following the pose of the robot.

References

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