Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lecture: Introduction to Mobile Robotics
Summer term 2011

Prof. Dr. Wolfram Burgard
PD Dr. Cyrill Stachniss
Juniorprof. Dr. Maren Bennewitz
Juniorprof. Dr. Kai Arras

Sheet 8

Topic: Extended Kalman Filter

Submission deadline: July 5, 2011
Submit to: `mobilerobotics@informatik.uni-freiburg.de`

General Notice

In this exercise, you will implement an extended Kalman filter (EKF) similar to the previous exercise sheet. The framework for the implementation can be obtained from the website of the course. The tarball contains the following folders:

data This folder contains files representing the world definition and sensor readings used by the filter.

octave This folder contains the EKF framework with stubs for you to complete.

plots The framework uses this folder to store images generated by the visualization.

samples Sample implementations of the sensor and motion model are provided in this folder.

To run the EKF, change into the directory `octave` and launch the *Octave* program. Inside *Octave*, type `extended_kalman_filter` to start the EKF. Running the EKF may take some time. While the EKF is running, plots visualizing the state of the filter are generated and stored in the `plots` directory.

Note: You first have to complete all the stubs in order to get the filter working correctly.

We use the librobotics library for some of the visualization. All functions defined in the library are available in the framework.

Some implementation tips:

- Turn off the visualization to speed up the computation by commenting out the line `plot_state(...` in the file `extended_kalman_filter.m`.
- While debugging run the filter only for a few steps by replacing the for-loop in `extended_kalman_filter.m` by something along the lines of `for t = 1:50`.

- The command `repmat` allows you to replicate a given matrix in many different ways and is magnitudes faster than using for-loops.
- When converting implementations containing for-loops into a vectorized form it often helps to draw the dimensions of the data involved on a sheet of paper.
- Many of the functions in *Octave* can handle matrices and compute values along the rows or columns of a matrix. Some useful functions that support this are `sum`, `sqrt`, and many others.

Exercise 1: Theoretical Considerations

The EKF is an implementation of the Bayes Filter.

(a) The Bayes filter processes three probability density functions, i. e., $p(x_t \mid u_t, x_{t-1})$, $p(z_t \mid x_t)$, and $\text{bel}(x_t)$. State the normal distributions of the EKF which correspond to these probabilities.

(b) Explain in a few sentences all of the components of the EKF, i. e., μ_t, Σ_t, g, G_t, h, H_t, Q_t, R_t, K_t and why they are needed.

Exercise 2: EKF Prediction Step

We assume a differential drive robot operating on a 2-dimensional plane, i.e., its state is defined by $\langle x, y, \theta \rangle$. Its motion model is defined on slide 10 (Odometry Model) in the chapter Probabilistic Motion Models of the lecture slides.

(a) Derive the Jacobian matrix G_t of the noise-free motion function g. Keep in mind that you implemented a similar function for exercise sheet 4. Do not use Octave.

(b) Implement the prediction step of the EKF in the file `prediction_step.m` using your Jacobian G_t. For the noise in the motion model assume

$$Q_t = \begin{pmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}.$$

(c) Compare how this choice of Q_t models motion noise to the way in which the motion model from the particle filter (exercise sheet 6) models it. *Bonus question*: How could one change Q_t such that it models the motion noise in the same way as the motion model in sheet 6?

Exercise 3: EKF Correction Step

(a) Derive the Jacobian matrix H_t of the noise-free measurement function h of a range-only sensor. Do not use Octave.

(b) Implement the correction step of the EKF in the file `correction_step.m` using your Jacobian H_t. For the noise in the sensor model assume that R_t is the diagonal square matrix

$$R_t = \begin{pmatrix} 0.5 & 0 & 0 & \dots \\ 0 & 0.5 & 0 & \dots \\ 0 & 0 & 0.5 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \in \mathbb{R}^{\text{size}(z_t) \times \text{size}(z_t)}.$$

After having successfully implemented the prediction step and the correction step, you can generate an animation from the saved images using *ffmpeg* or *mencoder*. With *ffmpeg* you can use the following command to generate the animation from inside the `plots` folder:

```
ffmpeg -r 10 -b 500000 -i kf_%03d.png ekf.mp4
```