Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Dr. Gian Diego Tipaldi, Dr. Luciano Spinello, Prof. Wolfram Burgard
Lecture: Robot Mapping
Winter term 2014

# Sheet 4

Topic: Extended Kalman Filter SLAM

Submission deadline: Nov. 20

Submit to: `robotmappingtutors@informatik.uni-freiburg.de`

**Exercise: Implement an EKF SLAM System**

Implement an extended Kalman filter SLAM (EKF SLAM) system. To support this task, we provide a small Octave framework (see course website). The framework contains the following folders:

**data** contains files representing the world definition and sensor readings.

**octave** contains the EKF SLAM framework with stubs to complete.

**plots** this folder is used to store images.

The below mentioned tasks should be implemented inside the framework in the directory `octave` by completing the stubs.

After implementing the missing parts, you can run the EKF SLAM system. To do that, change into the directory `octave` and launch *Octave*. Type `ekf_slam` to start the main loop (this may take some time). The program plots the current belief of the robot (pose and landmarks) in the directory `plots`. Figure 1 depicts some example images of the state of the EKF. You can use the images for debugging and to generate an animation. For example, you can use *ffmpeg* from inside the `plots` directory as follows:

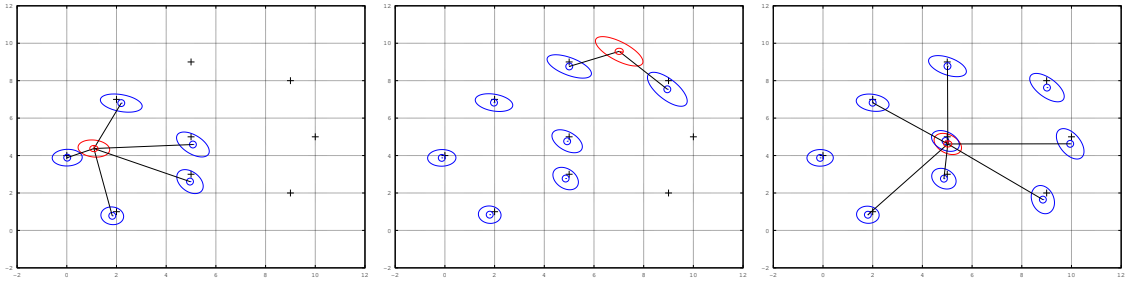
```
ffmpeg -r 10 -b 500000 -i ekf_%03d.png ekf_slam.mp4
```

(a) Implement the prediction step of the EKF SLAM algorithm in the file `prediction_step.m`. Use the odometry motion model:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1}) \\ \delta_{trans} \sin(\theta_{t-1} + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{pmatrix}.$$

Compute its Jacobian $G_t^x$ to construct the full Jacobian matrix $G_t$:

$$G_t^x = I + \begin{pmatrix} 0 & 0 & -\delta_{trans} \sin(\theta_{t-1} + \delta_{rot1}) \\ 0 & 0 & \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1}) \\ 0 & 0 & 0 \end{pmatrix}.$$



$t = 50$ $t = 150$ $t = 330$

Figure 1: Example images of the state of the EKF at certain time indices.

For the noise in the motion model assume

$$R_t^x = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.01 \end{pmatrix}.$$

(b) Implement the correction step in the file `correction_step.m`. The argument $z$ of this function is a struct array containing $m$ landmark observations made at time step $t$. Each observation $z(i)$ has an id $z(i).id$, a range $z(i).range$, and a bearing $z(i).bearing$.

Iterate over all measurements ($i = 1, \ldots, m$) and compute the Jacobian $H_t^i$ (see Slide 05 Page 35ff.). You should compute a block Jacobian matrix $H_t$ by stacking the $H_t^i$ matrices corresponding to the individual measurements. Use it to compute the Kalman gain and update the system mean and covariance *after* the for-loop. For the noise in the sensor model assume that $Q_t$ is a diagonal square matrix as follows

$$Q_t = \begin{pmatrix} 0.01 & 0 & 0 & \ldots \\ 0 & 0.01 & 0 & \ldots \\ 0 & 0 & 0.01 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \in \mathbb{R}^{2m \times 2m}.$$

Some implementation tips:

- While debugging, run the filter only for a few steps by replacing the for-loop in `ekf_slam.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.