

## Sheet 7: Extended Kalman Filter

June 29, 2023

### General Notice

In this exercise, you will implement an extended Kalman filter (EKF). A code skeleton with the EKF work flow is provided for you. A visualization of the EKF state is also provided by the framework.

The following folders are contained in the `kf_framework.tar.gz` tarball:

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

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

You can run the EKF in the terminal: `python kalman_filter.py`. It will only work properly once you filled in the blanks in the code.

Some implementation tips:

- To read in the sensor and landmark data, we have used dictionaries. Dictionaries provide an easier way to access data structs based on single or multiple keys. The functions `read_sensor_data` and `read_world_data` in the `read_data.py` file read in the data from the files and build a dictionary for each of them with time stamps as the primary keys.

To access the sensor data from the `sensor_readings` dictionary, you can use

```
sensor_readings[timestamp, 'sensor']['id']  
sensor_readings[timestamp, 'sensor']['range']  
sensor_readings[timestamp, 'sensor']['bearing']
```

and for odometry you can access the dictionary as

```
sensor_readings[timestamp, 'odometry']['r1']  
sensor_readings[timestamp, 'odometry']['t']  
sensor_readings[timestamp, 'odometry']['r2']
```

To access the positions of the landmarks from `landmarks` dictionary, you can use

```
position_x = landmarks[id][0]  
position_y = landmarks[id][1]
```

### Exercise 1: Theoretical Considerations

The EKF is an implementation of the Bayes Filter.

- (a) The Bayes filter processes three probability density functions, i. e.,  $p(x_t | u_t, x_{t-1})$ ,  $p(z_t | x_t)$ , 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.,  $\mu_t$ ,  $\Sigma_t$ ,  $g$ ,  $G_t$ ,  $h$ ,  $H_t$ ,  $Q_t$ ,  $R_t$ ,  $K_t$  and why they are needed. What are the differences and similarities between the KF and the EKF?

### 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  $g$ . Do not use Python.
- (b) Implement the prediction step of the EKF in the function `prediction_step` using your Jacobian  $G_t$ . For the noise in the motion model assume

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

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

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

Once you have successfully implemented all the functions, after running the filter script you should see the state of the robot being plotted incrementally with each time stamp.