Exercise 3
Extended Kalman Filter

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 attached:

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]

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 $bel(x_t)$. 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. What are the differences and similarities between the KF and the EKF?

2: EKF Prediction Step

Assume a differential drive robot operating on a 2-dimensional plane,

Let us refer to $s_t = \langle x_t, y_t, \theta_t \rangle$ as the state variables in order to avoid ambiguities with the x coordinate of the robot. If we denote the control vector u_t by $\langle \delta_{r_1}, \delta_t, \delta_{r_2} \rangle$, this particular motion model is defined as:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = g(s_{t-1}, u_t) = \begin{bmatrix} x_{t-1} + \delta_t \cos(\theta_{t-1} + \delta_{r_1}) \\ y_{t-1} + \delta_t \sin(\theta_{t-1} + \delta_{r_1}) \\ \theta_{t-1} + \delta_{r_1} + \delta_{r_2} \end{bmatrix}$$

Where: δ_t δ_{r1}, and δ_{r2} are translation , initial and final rotation, respectively.
(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

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

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.

Let $l = \langle l_x, l_y \rangle$ denote the landmark expressed w.r.t. the world reference frame, whose distance we are measuring. Then the measurement function h for a single landmark is scalar (since it returns a single range value) and is as follows:

$$z_t = h(s_t, l) = \sqrt{(x_{t-1} - l_x)^2 + (y_{t-1} - l_y)^2}$$

Where we added an l argument to h in order to make it parametric with respect to the landmark that we are measuring.

Again, we compute the Jacobian H_t by differentiating w.r.t. the state variables and evaluating it w.r.t. the current estimate of the state:

$$H_t = \begin{bmatrix} \frac{\bar{\mu}_{x,t} - l_x}{h(\bar{\mu}_t, l)} & \frac{\bar{\mu}_{y,t} - l_y}{h(\bar{\mu}_t, l)} & 0 \end{bmatrix}$$

(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 *Rt* 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.