SEOUL NATIONAL UNIVERSITY
Department of Naval Architecture and Ocean Engineering

# AUTONOMOUS MOBILE ROBOTICS

# Programming assignment #4

KESSELER CAMILLE 15/12/2022

CONTENTS

## 2D EXTENDED KALMAN FILTER

We want to use the given data to test a 2D Kalman filter, we will briefly repeat the theory of Kalman filter for SLAM and then implement it using the framework given.

### THEORY

Extended Kalman filter can be decompose in the following step at every time:

- Prediction Steps
  - $\bar{\mu}_t = g(u_t, \mu_{t-1})$
  - $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
- Correction Steps
  - $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
  - $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
  - $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

We can define the quantities to use in the Kalman filter:

- State variable contains the robot pose $p_t = [x_t, y_t, \theta_t]^T$ and the landmark in the maps pose $m_t = [m_{ID1}, m_{x1}, m_{y1}, \dots, m_{IDn}, m_{xn}, m_{yn}]$, here we prefixed the map number but in real-case the map doesn't have fixed so the state variable size will change with time when we add more landmark.

$$\mu_t = [x_t, y_t, \theta_t, m_{ID1}, m_{tx1}, m_{ty1}, \dots, m_{IDn} \;\; m_{txn}, m_{tyn}]^T$$

- The covariance of the state variable, which also can be decompose into covariance of robot pose/robot pose, landmark/ landmark and robot pose/landmark.

$$\begin{bmatrix} \Sigma p_t p_t & \Sigma p_t m_{t1} & \cdots & \Sigma p_t m_{tn} \\ \Sigma m_{t1} p_t & \Sigma m_{t1} m_{t1} & \cdots & \Sigma m_{t1} m_{tn} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma m_{tn} p_t & \Sigma m_{tn} m_{t1} & \cdots & \Sigma m_{tn} m_{tn} \end{bmatrix}$$

- For the prediction steps, we need to define the motion model $g$ and its derivative $G$. Theirs forms come from the motion sensor data we have. Here the data odometry are on the form: $u_t = [r_1, d, r_2]^T$.

$$g(u_t, p_{t-1}) = \begin{bmatrix} x_{t-1} + d * \cos(\theta_{t-1} + r_1) \\ y_{t-1} + d * \sin(\theta_{t-1} + r_1) \\ \theta_{t-1} + r_1 + r_2 \end{bmatrix}$$

$$G_t = \frac{\partial g}{\partial p_{t-1}} = \begin{bmatrix} 1 & 0 & -d * \sin(\theta_{t-1} + r_1) \\ 0 & 1 & d * \cos(\theta_{t-1} + r_1) \\ 0 & 0 & 1 \end{bmatrix}$$

We only predict the robot pose, so the landmark position in the state vector is unchanged. So, the total forms are:

$$g(u_t, \mu_{t-1}) = \begin{bmatrix} x_{t-1} + d * \cos(\theta_{t-1} + r_1) \\ y_{t-1} + d * \sin(\theta_{t-1} + r_1) \\ \theta_{t-1} + r_1 + r_2 \\ m_{t-1x1} \\ \dots \\ m_{t-1yn} \end{bmatrix}$$

$$G_t = \frac{\partial g}{\partial \mu_{t-1}} = \begin{bmatrix} 1 & 0 & -d * \sin(\theta_{t-1} + r_1) & 0 \\ 0 & 1 & d * \cos(\theta_{t-1} + r_1) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & I_{2n*2n} \end{bmatrix}$$

- For the correction steps, we need to define the observation model $h$ and its derivative $H$. Theirs forms come from the sensor data for the landmark observations that we have here range and bearing $z_t = [id, r, \alpha]^T$ for each measurement sensor so the total measurement vector and the observation model associate:

$$z_t = \begin{bmatrix} id_1 \\ r_1 \\ \alpha_1 \\ \dots \\ id_M \\ r_M \\ \alpha_M \end{bmatrix}, \quad h(\bar{\mu}_t) = \begin{bmatrix} id_{t1} \\ \sqrt{(\bar{m}_{tx1} - \bar{x}_t)^2 + (\bar{m}_{ty1} - \bar{y}_t)^2} \\ \arctan\left(\frac{\bar{m}_{ty1} - \bar{y}_t}{\bar{m}_{tx1} - \bar{x}_t}\right) - \bar{\theta}_t \\ \dots \\ \sqrt{(\bar{m}_{txM} - \bar{x}_t)^2 + (\bar{m}_{tyM} - \bar{y}_t)^2} \\ \arctan\left(\frac{\bar{m}_{tyM} - \bar{y}_t}{\bar{m}_{txM} - \bar{x}_t}\right) - \bar{\theta}_t \end{bmatrix}$$

$$H = \frac{\partial h}{\partial \bar{\mu}_t}$$

$$= \sum_{i,m} F_m \begin{bmatrix} 0 & 0 & \frac{-(\bar{m}_{txi} - \bar{x}_t)}{\sqrt{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2}} & \frac{-(\bar{m}_{tyi} - \bar{y}_t)}{\sqrt{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2}} & 0 & 1 & \frac{(\bar{m}_{txi} - \bar{x}_t)}{\sqrt{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2}} & \frac{(\bar{m}_{tyi} - \bar{y}_t)}{\sqrt{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2}} \\ \frac{\bar{m}_{tyi} - \bar{y}_t}{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2} & \frac{-(\bar{m}_{txi} - \bar{x}_t)}{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2} & 0 & 0 & -1 & 0 & \frac{-(\bar{m}_{tyi} - \bar{y}_t)}{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2} & \frac{(\bar{m}_{txi} - \bar{x}_t)}{(\bar{m}_{txi} - \bar{x}_t)^2 + (\bar{m}_{tyi} - \bar{y}_t)^2} \end{bmatrix} P_i$$

With $F_m = \begin{bmatrix} 0 & 0 \\ \dots & \dots \\ 1 & 0 \\ 0 & 1 \\ \dots & \dots \\ 0 & 0 \end{bmatrix}$, where the 1 are at position $2m, 2m+1$ and $P_i = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & \dots & 0 \end{bmatrix}$, where the 1 on the 4th and 5th line are at position $2 * i, 2 * i + 1$.

## IMPLEMENTATION

I used the template code given by the TA. I modified the following part outside of the asked part:

- Map class: add members function to the class to add Mappoint, get mappoint with ID and check if present mappoint.
- Plot directory addition with python function for the image.

### EKF CLASS

The whole process is store in an ekf class, the header file contains the following member function and variable:

- State state: State of the EKF at the actual time
- Map map: Landmark position of the EKF at the actual time
- Vector<int> Landmark_detection: contain the number of time the landmark was detected
- Vector<Pose> Trajectory: Memory of all the pose computed during the EKF.
- Vector<double> RMSE: memory of the RMSE compute at each time
- Int Save_instant(int step): save instant pose in a dat file
- Int RMSE_land(Map map): compute the RMSE at this time
- Int SaveTrajectory(string file): save the trajectory in a dat file
- Int SaveMapPoints(string file): save the map point in a dat file
- Int SaveRMSE(string file): save the RMSE in a dat file
- Int UpdateTraj(): copy the data from state to map and trajectory
- Int correct(vector<Landmark> lm, Eigen::Matrix3f Q): correction step of the EKF
- Int predict(Odometry Movement, eigen::Matrix3f R): prediction step of the EKF

## INITIALIZATION

For the initialization, we used a robot pose and uncertainty of zero as we are sure of the position 0 of the robot. We assume that we know the number of landmark beforehand but not information about their localization so initialized to 0 the position too. For the landmark/robot covariance we set it to 0 matrix too. And also we used a map where we supposed the map point are ordered by ID. (i.e map.data[3] is the ID 3 landmark).

We re-initialized the landmark position during the ekf-process. When doing the correction step we first check if the landmark has not been seen before. If this is the first time seeing we change its position with the measurement, this implies that the first time we see a landmark we do not correct our position with it as the error will be 0.

For real application maybe we can start with a dimension 3 state and add landmark as we go on and change the size of all the matrix during the computation.

## ANGLE ROBUSTNESS

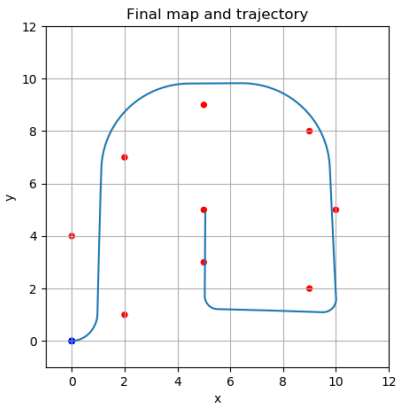
One problem that arises usually is when computing the error on the angle for the landmark measurement. This comes from the fact that we use atan2 and sum the number to the robot angle. When summing this can go out of the boundary [-PI,PI], we need to check this when computing the observation model. So we add the condition:

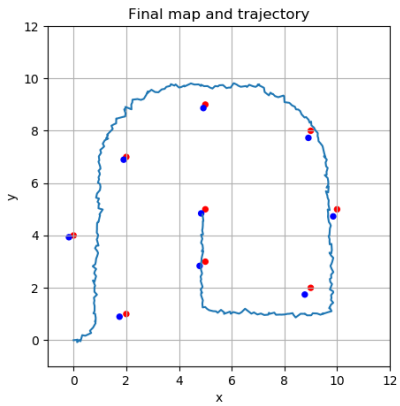
- While $(\alpha_{compute} < -\Pi)$: $\alpha_{compute} \mathrel{+}= 2 * PI$
- While $(\alpha_{compute} > \Pi)$: $\alpha_{compute} \mathrel{-}= 2 * PI$

# RESULTS COMPARISON

## INITIAL DATA AND GROUND TRUTH

We use the data given in entry for the testing of the EKF, we only used odometry.dat for ekf computation. We can plot the ground truth landmark, the full compute trajectory after the ekf and final landmark to look at it.




The trajectory when only using the prediction step is way smoother but show some drift as we expected. The trajectory with the correction step is less smooth we can see that the correction add more small oscillation of the trajectory but we have more precise position.

## RMSE COMPUTATION

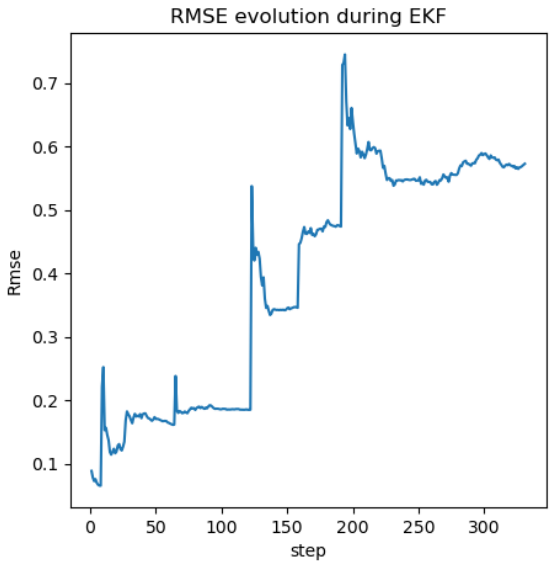
To compare the quality of the trajectory and the map, we compute the RMSE of the map point at every time steps. The formula used is:

$$RMSE = \sum_{landmark \in map} (x_{landmark} - x_{GT-landmark})^2 + (y_{landmark} - y_{GT-landmark})^2$$

As with time, we add landmark to the map the compute RMSE do not always contains the same level of information, so we will give along the list of landmarks used at the time t.

## RESULTS

We can plot the RMSE in function of the steps as the following picture shows:




We can see that every time we add a new landmark to the map the RMSE is getting bigger then with time it goes down. This make sense as the first time we add the landmark we do not correct any position using it, but then we correct its position along the robot position using the other landmark.

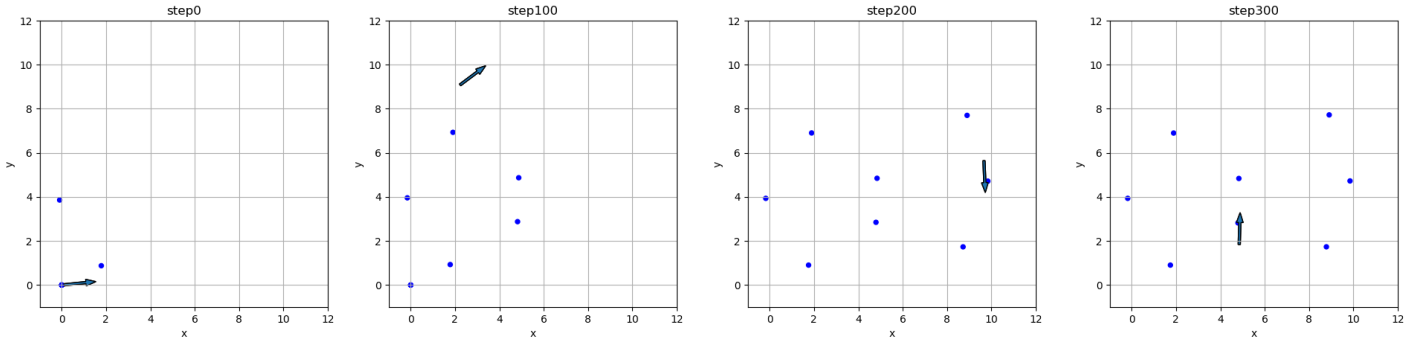
# VISUALIZATION

## HOW TO

In order to visualize the map and robot position at time t. I choose to use python as it offers simple and quick plot option for 2D visualization. The python script is inside the plot folder. The data are saved in dat file during the ekf cpp computation and then we open then, read data, plot them and finally save the image in the plot folder.

## IMAGE

We plot for every step the position of the robot and the landmark, the saved pictured can be found in the plot/Instant_image . I just show some iterations here.



## REFERENCE

Course pdf : LN_12D_EKF_Claus_Brenner_2022_11_29.pdf

[SLAM] Extended Kalman Filter (EKF) SLAM blog by Jinyong:
http://jinyongjeong.github.io/2017/02/16/lec05_EKF_SLAM/

EKF-SLAM (Cyrill Stachniss) youtube video: https://www.youtube.com/watch?v=X30sEgIws0g&t=3435s