Lab4 - EKF Map Based Localization

1 Introduction

The goal of this Lab exercise is to program an EKF Map Based Localization algorithm to work with data taken from a real Turtlebot.

Map based localization is a concept that in real life each one of us uses whenever we go to a new place. Whenever a human being goes to a new environment it tries to know where he is. In order to do that, what he does is to read a map of the location and try to relate key parts of the environment he has seen with features of the map so he can localize himself in the map while he is moving around. This can be applied also to robots using an EKF map based algorithm:

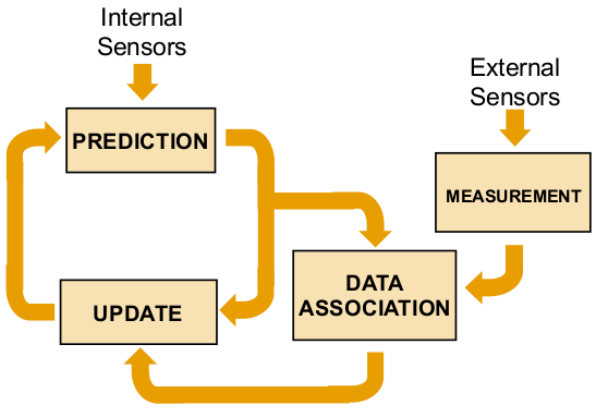

Figure 1: Schematic of EKF map based localization

This EKF map based localization is based in two different sensors, one for the prediction, and one for the update. The prediction is based on the internal odometry of the Turtlebot while the update is based on lines sensed by its Kinect.

2 Pre-lab

Read and understand the guide for this Lab and look about the EKF algorithm in your lecture slides. Find the expressions to compute $x^w_{k|k-1}$ and matrices A_k W_k in order to compute $P^w_{k|k-1}$ from x^w_{k-1} , $u_k^{r_{k-1}}$ and Q_k. You can write them in paper and attach a picture in the online form submission.

3 Lab work

The ROS package named `ekf_localization` contains all necessary code for running this lab and a bagfile obtained from a turtlebot with `nav_msgs/Odometry` and `sensor_msgs/LaserScan` messages. You need to copy your `splitandmerge.py` solution into the `src` folder. Then you can test the code by copying the ROS package into your workspace and running

```
roslaunch ekf_localization ekf_localization.launch
```

As you can see in Rviz that the robot doesn't move at all. You will need to complete the code inside the `ekf_localization.py` file in order to complete properly the EKF Localization algorithm. No changes are needed in the `node.py` nor the `functions.py` files. If you think you need to modify any of these files ask your lab assistant first.

The EKF algorithm is as follows:

▶ ***Pose initialization***

$x_0^w = \hat{x}_0^w$

$P_0^w = \hat{P}_0^w$

for k = 1 **to** *steps* **do:**

$\left[u_k^{R_{k-1}}, Q_k\right] = get\_odometry()$

▶ ***EKF prediction***

$\left[x_{k|k-1}^w, P_{k|k-1}^w\right] = move\_vehicle(x_{k-1}^w, P_{k-1}^w, u_k^{R_{k-1}}, Q_k)$

$[z_k, R_k] = get\_measurements()$

▶ ***Data association***

$\mathcal{H}_k = data\_association(x_{k|k-1}^w, P_{k|k-1}^w, z_k, R_k)$

▶ ***EKF update***

$[x_k^w, P_k^w] = update\_position(x_{k|k-1}^w, P_{k|k-1}^w, z_k, R_k)$

end for

3.1 Pose initialization

The pose initialization is already implemented as:

$$x_0^w = \begin{pmatrix} x_{x_0} \\ x_{y_0} \\ x_{\theta_0} \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.3 \\ -0.03 \end{pmatrix} \tag{1}$$

$$P_0^w = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

3.2 EKF prediction

You will need to complete the function `predict` inside `turtlebot_localization.py`.

This part of the algorithm is where, from the odometry U_k and the previous position the new one is computed.

▶ ***Prediction***

$x_{k|k-1}^w = f(x_{k-1}^w, u_k^{r_{k-1}}, w_k)$

$P_{k|k-1}^w = A_k P_{k-1}^w A_k^T + W_k Q_k W_k^T$

These measurements have an uncertainty associated. This uncertainty has been estimated to be 0.025 m as linear noise and 2 degrees as angular noise, so the Q_k matrix can be estimated for all the measurements as:

$$Q_k = \begin{pmatrix} 0.025 & 0 & 0 \\ 0 & 0.025 & 0 \\ 0 & 0 & 2 \end{pmatrix} \tag{3}$$

Once implemented correctly, when you test the script you should see the robot (blue arrow) moving around the room, and an ellipse representing uncertainty increasing its size when the robot moves.

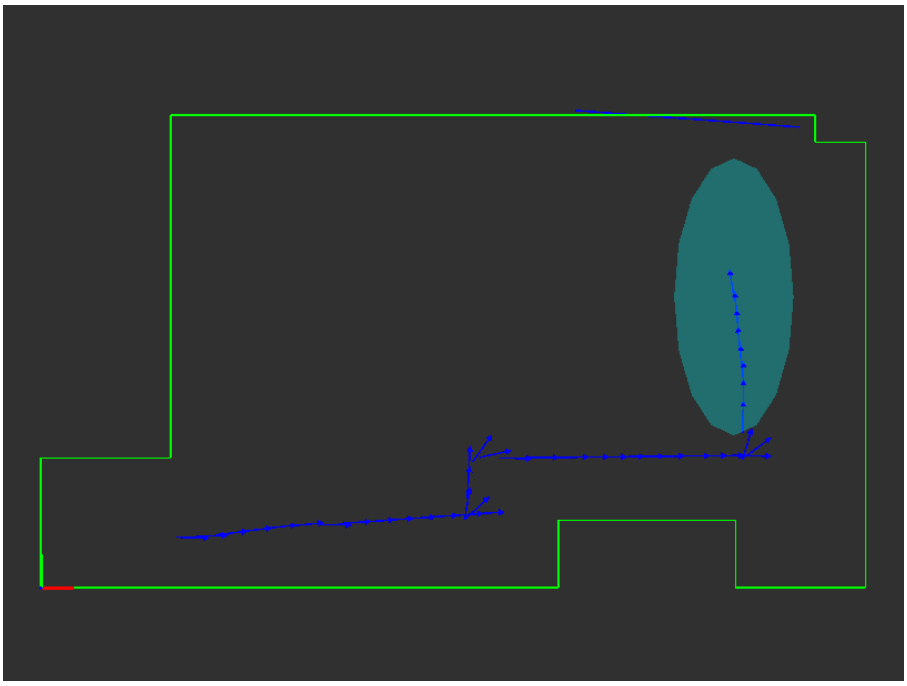

Figure 2: Prediction without updates.

3.3 Data association

Data association is one of the main parts in the algorithm as it is where it is decided how the position of the robot will be updated, and consequently, whether or not the robot will be positioned in its real position. The aim of this function is to give a list containing to which feature each observation corresponds. To decide whether or not the distance between a feature and an observation is close enough to claim they are the same, both uncertainties have to be taken into account. To do so, Mahalanobis distance is used.

After computing the distance, the smallest one is chosen, and if it is smaller than a threshold, the observation and the feature are associated. The square of a Gaussian follows a $\chi^2_{\rho,\varphi}$ distribution, so the threshold will be extracted from there. You will need to choose this threshold value.

► ***Data association***

$$v_{ij} = z_i - h(x^w_{k|k-1})$$
$$S_{ij} = H_j P^w_{k|k-1} H_j^T + R_i$$
$$D^2_{ij} = v^T_{ij} S^{-1}_{ij} v_{ij}$$
Take the smallest D^2_{ij} if $D^2_{ij} < \chi^2_{\rho,\varphi}$

To obtain the measurement function $h(x^w_{k|k-1})$ that transforms a feature in the state vector in the world frame x^w_f to a feature in the robot frame x^r_f you can use Fig. 3.

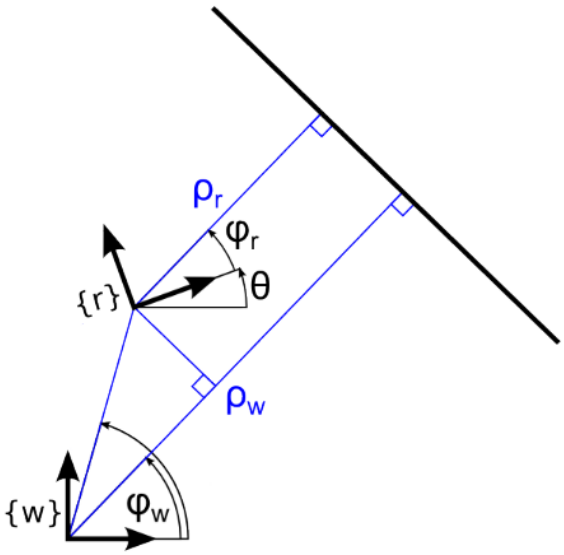

Figure 3: Schematic to deduce the measurement function $h(\hat{x}^w_{k|k-1})$

Regarding R_k, the covariance matrix for the observations, the angle measurement noise has been estimated to 10 degrees and the range measurement to 0.2 m. R_k matrix is estimated as constant.

$$R_k = \begin{pmatrix} 0.2 & 0 \\ 0 & 10 \end{pmatrix} \tag{4}$$

3.4 Update

In this step, the matches obtained on the data association are used to update the position of the robot and its uncertainty. The main algorithm is the following one. Starting by calculating the innovation v_k which is the difference between the observed features z_k and the predicted observation $h(x^w_{k|k-1})$. Then the uncertainty of this innovation is calculated in S_k. Finally. the Kalman gain K_k is computed allowing to update the state vector x^w_k and the covariance matrix P^w_k.

► ***Update***

$$v_k = z_k - h(x^w_{k|k-1})$$
$$S_k = H_k P^w_{k|k-1} H_k^T + R_k$$
$$K_k = P^w_{k|k-1} H_k^T S_k^{-1}$$
$$x^w_k = x^w_{k|k-1} + K_k v_k$$
$$P_k = \left(I - K_k P^w_{k|k-1}\right) P^w_{k|k-1} \left(I - K_k P^w_{k|k-1}\right)^T + K_k R_k K_k^T$$

Note that $h(x^w_{k|k-1})$ and H_k have already been computed for each feature in the data association. In this step the difference is that both can have more than one observation to update the robot position. In order to update with all the information at the same time $h(x^w_{k|k-1})$ and H_j computed for each observation associated to a feature are stacked vertically to obtain the final $h(x^w_{k|k-1})$ and H_k matrix and be able to run the update algorithm.

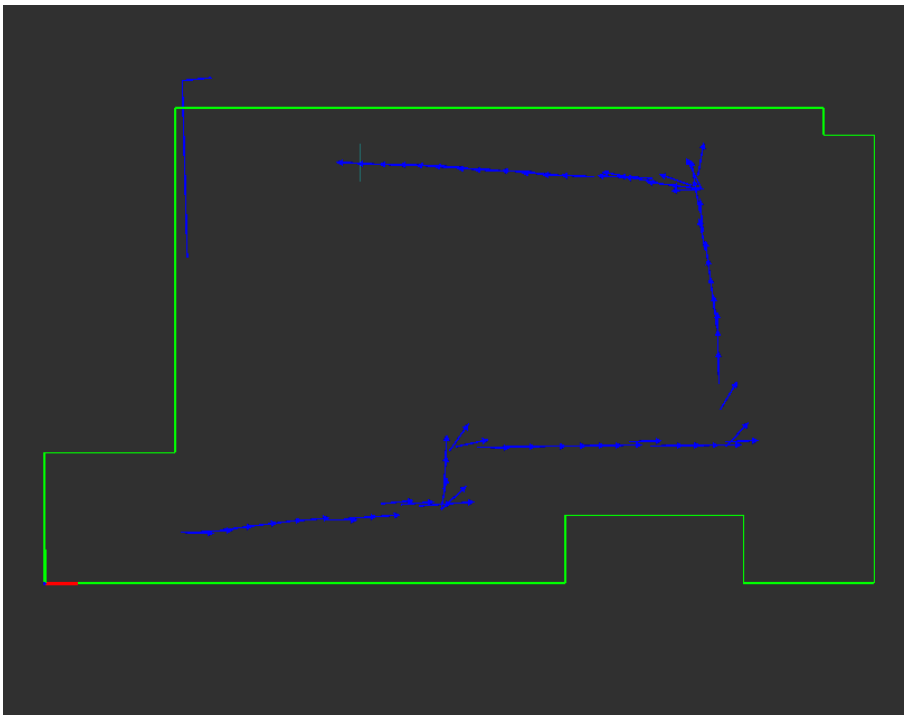

Figure 4: Prediction with updates.

4 Optional

Using polar representation introduces a new challenge. Walls are represented by lines without ends and not by segments. These can introduce confusions in the data association. Try adding new constraints in the data association subsection to avoid matching the sensed lines with "ghost" walls.

5 Lab report

Write a brief report (MAX 4 PAGES) explaining your solution and problems faced. Include the final code.