EKF Simultaneous Localization and Mapping

1 Introduction

The goal of the program is to familiarize with the Simultaneous Localization and Mapping (SLAM). **This document is just an example of how SLAM algorithm can be implemented using an Extended Kalman Filter (EKF). You can either start with this or create your algorithm of your own.** The instructions for this document are based on data already recorded, and can be a good starting point. For your project and to demonstrate the system works, you will have to actually use data from the Turtlebot in the lab.

SLAM is a concept that in real life is used whenever a new place is visited. What a robot does while executing a SLAM algorithm is composed of two parts: build a map and localize itself into it. A map can contain multiple types of information. However in the implementation presented here only lines sensed will be used as map features.

This EKF-SLAM is based in two different sensors. The first one are the encoders of the wheels of the Turtlebot which will give us information of the movement of the robot (odometry). The second one is the Kineckt sensor which allows the robot to sense the environment (walls, obstacles, doors...) and map the features.

2 Theoretical work

- Read and understand what are characteristics of feature-based SLAM.
- If you are going to implement EKF SLAM:
  - Find the expression of F_k and G_k for the prediction equation as functions of the jacobian of the composition with respect to the state vector (which compared to the previous lab has, a part from the robot position, n number of features) $J_{1\oplus}$ and the odometry readings $J_{2\oplus}$.
  - Is it possible to do the state augmentation with just one equation for n new features? Justify your answer with equations and shapes of matrices.
  - Consider $f({}^{A}x_{B}, {}^{B}z)$ (implemented in `self.tfPolarLine`) the function that converts an observed feature ${}^{B}z$ in the frame B into A using the transformation ${}^{A}x_{B}$ (how the frame B is seen from A), with partial derivatives $F_{1|x,z} = \delta f({}^{A}x_{B}, {}^{B}z)/\delta {}^{A}x_{B}$ and $F_{2|x,z} = \delta f({}^{A}x_{B}, {}^{B}z)/\delta {}^{B}z$. Show the equations to add n feature to the sate vector. Where are the partial derivatives of f going to be located (write them within the equations)?
  - Considering the previous question what are the expressions for $F_{1|x,z} = \delta f({}^{A}x_{B}, {}^{B}z)/\delta {}^{A}x_{B}$ and $F_{2|x,z} = \delta f({}^{A}x_{B}, {}^{B}z)/\delta {}^{B}z$?
  - Considering that we have n features in the state vector, find the size of the matrices H, S, v and R in order to be able to update the filter with m associated features.

3 Implementation work

For this implementation it is possible to use code previously done in the Lab4 - EKF Map Based Localization.

3.1 Running the code

In this section (and thinking of the final goal of the project where you have to test it in a real robot) the launch file (*ekf_slam.launch* in the *project_slam* package) has several arguments in order to work with a gathered dataset or a simulation of the robot. The different arguments and its default values are:

- rviz [true]: if true displays the robot and the map in rviz.
- bagfile [true]: if true uses the data from the bagfile

- frequency [1.0]: sets a different frequency for the bagfile to publish the data (if you don't have a powerful computer you can make it slower).
- pause [false]: if true sets the rosbag play node to pause so the messages can be posted step by step by pressing `s`.
- simulator [false]: if true launches gazebo simulator with a simple room. If this option is used, run the teleoperation node for the turtlebot in a different terminal so you can drive arround the turtlebot.
- dataset1 [false]: if the data is coming from a bagfile, dataset 1 is used when set to true (real dataset with ground truth map).
- dataset2 [false]: if the data is coming from a bagfile, dataset 2 is used when set to true (real dataset without ground truth map).
- dataset3 [true]: if the data is coming from a bagfile, dataset 3 is used when set to true (synthetic dataset with ground truth map).

3.2 Prediction

In the `predict` function implement the equations:

$$\widehat{x}^B_{k|k-1} = f\left(\widehat{x}^B_{k-1}, \widehat{u}^{k-1}_k\right) \tag{1}$$

$$P^B_{k|k-1} = F_k P^B_{k-1} F_k^T + G_k Q_k G_k^T \tag{2}$$

Use the measurement uncertainty of the previous lab (0.025m for the linear movement noise and 2 degrees for the angular). Once implemented please check that the uncertainty grows without boundaries as shown in the `prediciton.mp4` video. Note that F_k ang G_k are the ones asked in the second point of the theoretical work.

3.3 State augmentation

This part is totally new from the previous lab. In this function the state vector grows with the newly observed features in order to build the stochastic map. Use the non associated observations to enlarge the state vector. Use the equations of the 3rd point of the theoretical work in this step. Once this step is implemented but not the following ones, the state vector should add all observed lines as new features, hence, the state vector will keep growing which might make you computer to slow down or crash. You can see an example in `state_augmentation.mp4` video.

3.4 Data association

Compared to the previous lab, now the lines of the map are not saved as initial and ending point, so the equations for computing distance and transform lines between frames have to change.

First, you need to complete the function `tfPolarLine` in order to return the Jacobians asked in the theoretical part. Once this function is implemented use it in `lineDist` to compute the distance between a given observation z and a map feature given by its index in the map. With this two functions implemented compute the data association. Finally you also need to define the chi-square threshold in `slef.chi_thres` in the class constructor. Note that in this case the data association also returns the indexes of the non associated features in order to be able to add them to the map. Note that while the state augmentation is not fulfilled you will not be able to associate any data since the state vector will only contain the robot position.

3.5 Update

With the associations done, and taking into account the theoretical work, update the filter. Note that while the state augmentation is not fulfilled you will not be able to update the robot position since no data will be associated. Once you have this part implemented the algorithm should behave similar to the video `slam.mp4`.

4 Optional

Since some time the line extraction algorithm is not really robust and lines which are not in the environment appear in the scan, trace each line before using it as a map feature in order to make sure this outliers do not take part into the process. Use the variables `featureObservedN` (a vector containing how many times a feature has been observed) and `min_observations` in order to assess a minimum number of observations for a feature before adding it to the map. The video `slam_with_optional_part.mp4` shows how the SLAM algorithm should behave at this point.