Kalman Filters for Sensor fusion and Pose Estimation 5

(Max score: 40)

16-311: Introduction to Robotics (Fall 2017)

OUT: November 11, 2017, at 6:00pm

DUE: November 20, 2017 at 9:00am - Available late days: 1

Instructions

Homework Policy

Homework is due on autolab by the posted deadline. As a general rule, you have a total of 8 late days. For this homework you cannot use more than 1 late day. No credit will be given for homework submitted after the late day. After your 8 late days have been used you will receive 20% off for each additional day late.

You can discuss the exercises with your classmates, but you should write up your own solutions. If you find a solution in any source other than the material provided in the course, you must mention the source.

Submission

Create a tar archive of the folder with your ROS packages/nodes and submit it to Homework 5 on autolab. You should also have one PDF file in your archive, with an explaination of your results regarding the experiments with the robot, and the instructions for running the nodes, if any.

Contents

1 Data filtering and fusion: laser scanner data (14 points)

The laser scanner data on our TurtleBot (that are obtained from the depth camera) can be used to detect obstacles to safely navigate in the environment, create an occupancy map, and extract geometric features such as lines. Unfortunately, measures are subject to noise and errors, that make them fluctuating over time.

Let's focus on the use of the laser scan data for measuring the distance to nearby objects, and let's assume that all sensing uncertainties can be modeled in terms of Gaussian white noise. Therefore, we can use a *Kalman Filter to filter data streams* and obtain more reliable estimates for object distances in the sensor space.

For the laser scan sensor, the `ranges[]` array returns a depth value for the presence (or not) of an object along $n = 640$ radial directions, centered in the `camera_link` coordinate frame. If an object is detected at a distance d along the i-th radial direction, the distance value is reported in `ranges[i]`. If nothing is detected (up to the maximum range), a `NaN` is reported (or any arbitrary value over the declared ranges).

1. (7 points) *Consider the case with the robot that stands still.* The goal is to filter the data stream for each one of the n radial measurements by using a KF. Consider a n-dimensional state vector $\boldsymbol{\xi}$ with components that are all statistically independent from each other.

   Describe the KF model that you have adopted, report the equations and the parameter, the choices for the error models, and implement the code in ROS/Gazebo. For effective matrix manipulation, a suggestion is to use the `SciPy` library.

For performing the experiments, in Gazebo, create a scene (or, better, multiple scenes) with a few complex objects in front of the robot, similar to the one shown in the figure.

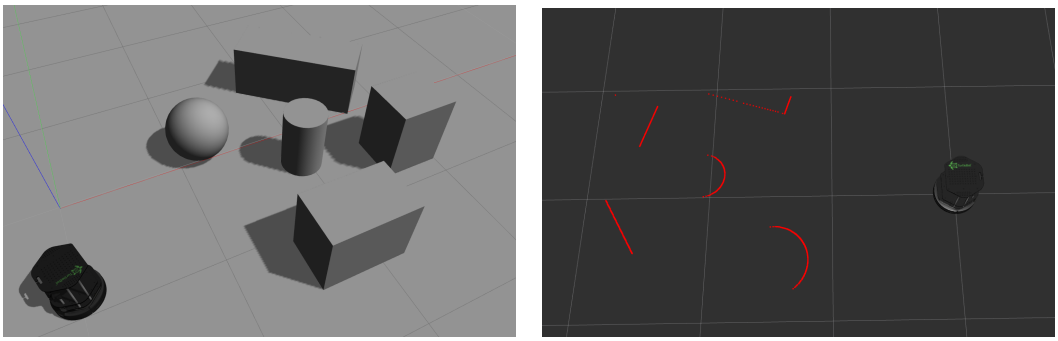

Figure 1: Complex scene. To the right, the view from the LaserScan on rviz.

Check the effect of the filter by observing the filtered scan image in rviz (we will discuss in the next class how to do it). Show the behavior of the covariance estimates over time.

Set up a complex scene in the *real world, and perform similar experiments*. Your filter will be evaluated on real-world scenes.

2. (7 points) Consider the same scenario as before, but now *aggregate the measurement data in sectors*, considering 10 sectors of 64 contiguous data points each. Design and implement a KF as before for each one of the 10 aggregated measurements. In this case, the filter will perform both *data filtering and fusion* (of the 64 measurements).

   Describe the filter and proceed as before for performing the experiments.

2 EKF for robot localization and navigation with a landmark map (26 points)

The robot moves in a squared environment E of dimension $20\times$ 20 m^2. In the environment, $n = 10$ *landmarks* are spread randomly. Each landmark is detectable by the robot using one of its on-board sensors. In particular, the sensor returns the *range* ρ and *bearing* β of a landmark, in the sensor's coordinate frame. A landmark can be reliably sensed only when it is at a distance of *less than 2.5 meters* from the robot. For sake of simplicity, let's assume that the sensor also returns the *identity* i of the landmark.

A *map* of E, with landmarks' positions, is given to the robot in the following format (the text is to explain the numeric fields):

Id	x	y	
1	x_1	y_1	(Coordinates of landmark 1)
2	x_2	y_2	(Coordinates of landmark 2)
...	...	...	
n	x_n	y_n	(Coordinates of landmark n)

The origin $(0, 0)$ of the E map is placed in the center of the squared area. All measures are expressed in meters.

The values ρ^i and β^i correspond to the true range and bearing values *corrupted by a sensing noise* $\boldsymbol{w}_k$, which is modeled as an additive *Gaussian white noise*. The two noise components are assumed to be uncorrelated.

The robot is given a path $\mathcal{P}$ to follow, expressed as a sequence of p points in Cartesian coordinates:

$$\mathcal{P} = (x_0, y_0), (x_1, y_1), \ldots, (x_p, y_p).$$

The robot starts in the initial pose $\boldsymbol{\xi}_0 = (x_0, y_0, 0)$. The initial pose is known to the robot with zero error.

The task of the robot is to exploit the map and use an EKF to localize itself in the environment and navigate accurately following the given path points. At the same time, the robot has to use its range sensors to avoid collisions with the obstacles in E.

1. Design and implement an EKF that makes use of issued velocity controls and knowledge of the robot's kinematic equations to define state (i.e., pose) dynamics, and that exploits landmark observations (when available) to adjust the state estimation.

   At this aim, in Gazebo implement a *virtual landmark sensor*. Given the map and the *ground truth* about robot position (obtained from a service, as we have already done in previous homework), the landmark sensor returns the observation of the closest landmark within the sensing range, if any. The sensor returns a triple (ρ, β, i). In order to simulate perception noise, the returned values must correspond to the true range and bearing values but *corrupted by a Gaussian noise* according to the following covariance matrix (units refer to meters and radians):

$$\boldsymbol{w}_k = \begin{bmatrix} w_k^\rho & w_k^\beta \end{bmatrix}^T \sim N(0, \boldsymbol{W}_k), \quad \boldsymbol{W}_k = \begin{bmatrix} \sigma_{k\rho}^2 & 0 \\ 0 & \sigma_{k\beta}^2 \end{bmatrix} = \begin{bmatrix} 0.025 & 0 \\ 0 & 0.16 \end{bmatrix}$$

   Assume that the landmark sensor can perform observations at a *max frequency of 15Hz*.

2. *Test the EKF and the navigation task in simulation* using the world configuration provided in the given file `ekf-navigation.world`. The world features a number of box obstacles that must be avoided, and a number of path points, that are shown by coke cans (red objects). The world and an example of path are reported in the figure. The path is intended to show the order according to which the path points have to be followed, starting from $(0, 0)$ and moving NE. Note that the grid is 20×20, where each square is a 1×1 m^2. The path points are all located at corner points, such that their coordinates can be easily computed.

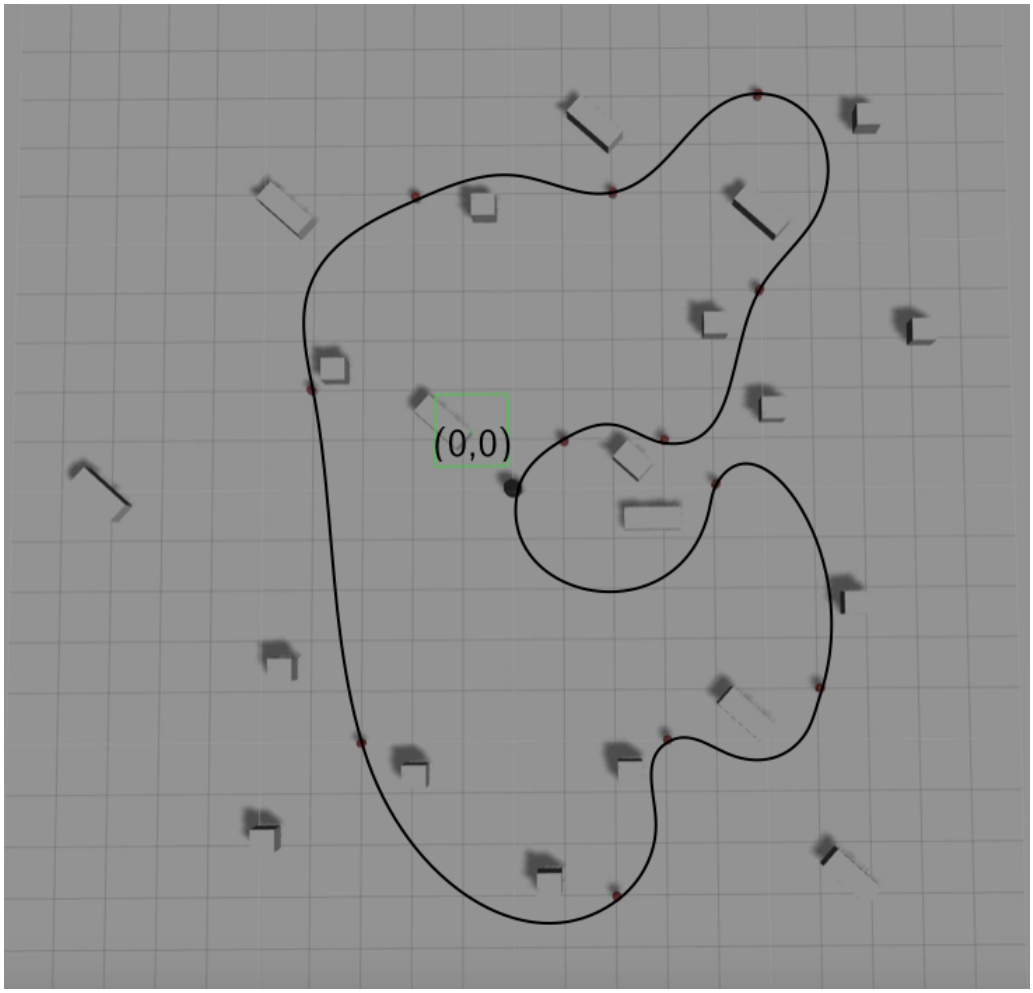

Figure 2: The world to navigate.

   You are free to place the n landmarks at randomly selected locations.

3. In order to navigate between path points, you have to use a *feedback-based* controller, similar to the one of the previous homework. In this case, the pose is provided by the current estimate obtained by the

EKF.[1] Note that in this case the environment features the presence of obstacles. For navigation between path points, Let's follow the approach suggested by DWA, but in a simplified way. Define two different feedback-based controllers, one, *Go-to-Goal*, that brings the robot to the next point in the path, and one, *Avoid-Obstacle* that makes the robot moving away from an obstacle, as detected using the laser scan sensor. Then define a *blending* rule that each time moves the robot according to both controllers by assigned different weights:

$$\boldsymbol{u} = \alpha(d)\boldsymbol{u}_{GtG} + (1 - \alpha(d))\boldsymbol{u}_{AO}$$

where $\alpha(d) = 1 - e^{-\delta d}$, $\alpha(d) \in [0, 1]$, d is the distance to the current closest obstacle, and δ is a constant. In practice, δ defines at which distance the weight of the obstacle avoidance controller becomes active and how it grows. The controls $\boldsymbol{u}$ refer to the linear and angular velocities.

4. (8 points) *Test EKF + Navigation in the real-world*, creating a somehow equivalent scenario. In the class it will be shown how to observe landmarks using the camera sensor. Landmarks will be made by using AR visual tags similar to the ones shown in the figure.

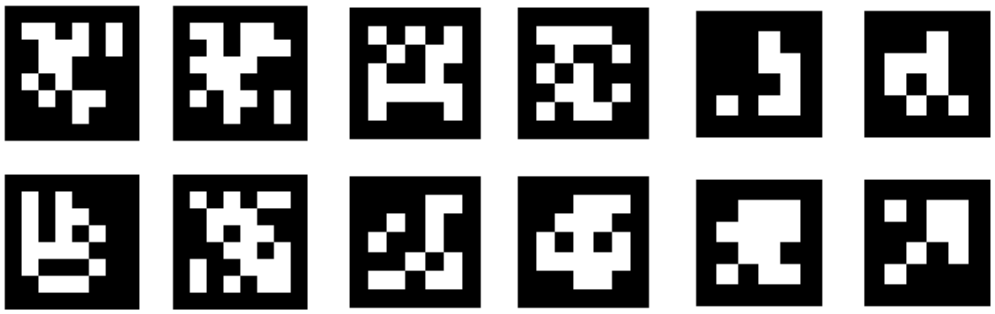

Figure 3: Examples of visual tags.

In the report, provides the details about the design of the EKF, as well as of the feedback-based controllers. Show the evolution of the error in pose estimation and try to quantify the error following the path points.

[1] Use the pose estimates from the `/odom` topic to check whether your EKF estimate is good or not (Odometry is precisely based on an EKF using data from wheel encoders and IMUs).