SSY345 – Sensor Fusion and Non-Linear Filtering
Home Assignment 4 - Analysis

Basic information

This home assignment is related to the material in lecture 7 and 8. A large part of the assignment focuses on understanding of the basic concepts that we will rely on later in the course.

In the analysis part we want you to use the toolbox that you have developed and apply it to a practical scenario. Associated with each scenario is a set of tasks that we would like you to perform.

It is important that you comment your code such that it is easy to follow by us and your fellow students. Poorly commented code will result in deduction of POE. Your code should be exported as a txt and uploaded as part of your submission before the deadline. The purpose is to make feedback from your peers possible and to enable plagiarism analysis (Urkund) of all your submissions.

The result of the tasks should in general be visualised and compiled together in a report (pdf-file). A template for the report can also be found on course homepage. Note that, it is sufficient to write short concise answers to the questions but they should be clearly motivate by what can be seen in the figures. Only properly referenced or captioned figures such that it is understandable what will result in POE. Also, all the technical terms and central concepts to explain an answer should be used without altering their actual meaning. The report should be uploaded on the course homepage before the deadline.

1 Smoothing

In this task, you will investigate the results from an RTS smoother.

We build on task 3 of HA3 by performing smoothing on a measurement sequence, and then compare with the filter outputs from the same data.

Table 1: Generate true track

```
%% True track
% Sampling period
T = 0.1;
% Length of time sequence
K = 600;
% Allocate memory
omega = zeros(1,K+1);
% Turn rate
omega(150:450) = -pi/301/T;
% Initial state
x0 = [0 0 20 0 omega(1)]';
% Allocate memory
X = zeros(length(x0),K+1);
X(:,1) = x0;
% Create true track
for i=2:K+1
    % Simulate
    X(:,i) = coordinatedTurnMotion(X(:,i-1), T);
    % Set turn-rate
    X(5,i) = omega(i);
end
```

Generate a true state sequence

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{600} \tag{1}$$

using the code in Table 1. If you plot the positions, you will see that the sequence consists of a straight line, followed by a turn, followed by another straight line.

We proceed to use the dual bearing measurement model and the true sequence to generate measurement sequences

$$\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{600}. \tag{2}$$

The initial prior is

$$\mathbf{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \tag{3}$$

$$\mathbf{P}_0 = \mathrm{diag}\left(\begin{bmatrix} 10^2 & 10^2 & 10^2 & \left(\frac{5\pi}{180}\right)^2 & \left(\frac{1\pi}{180}\right)^2 \end{bmatrix} \right). \tag{4}$$

Let the sensor position be stationary,

$$\mathbf{s}_1 = [300\ ,\ -100]^T, \quad \mathbf{s}_2 = [300\ ,\ -300]^T. \tag{5}$$

The measurement noise standard deviations are known for both sensors,

$$\sigma_\varphi^{(1)} = \pi/180, \quad \sigma_\varphi^{(2)} = \pi/180. \tag{6}$$

The process noise covariance $\mathbf{Q}$ should be set to the well-tuned value in HA3 resulted in. (You may of course change the value if your tuning in HA3 did not give a satisfactory result.)

Task: Filter the measurement sequence using your favorite nonlinear Kalman filter using the coordinated turn motion model, and run the corresponding smoother of the filter on the measurement sequence and the filtering densities.

a Plot the sensor positions, the positions corresponding to the measurements, the true position sequence, the filtered and smoothed position trajectories as well as their corresponding 3σ-covariance contours for 5^{th} of the filter and smoothing estimates.

- Compare the shapes of the filtered and smoothed trajectories and comment on your observations.
- What are the differences between filtering and smoothing error covariances in the same time instance? Comment on your observations.

b Modify a measurement at one time instance $k = 300$ to manually introduce an outlier. Run your smoother on the modified measurement sequence and illustrate the resulting trajectories along with the measurement sequence (in Cartesian coordinate) in the areas where the outlier has an impact. How well do the filter and the smoother cope with the introduced outlier data? Explain the differences. *Hint: to introduce the outlier, a simple way is to add some random noise to the measurement you would like to modify. Note that the bearing measurement is represented as radius so you may need to repeat this random process a few times and observe its position in Cartesian coordinate to make sure that the "noise" is not too small.*

2 Particle filters for linear/Gaussian systems

In this task, you will study the basic properties of particle filter in a linear and Gaussian setting.

Consider the following linear and Gaussian state space model:

$$x_k = x_{k-1} + q_{k-1} \tag{7}$$

$$y_k = x_k + r_k, \tag{8}$$

where the motion and measurement noises, q_{k-1} and r_k, are zero mean Gaussian random variables with variances $Q = 1.5$ and $R = 3$, respectively, and the initial prior is $p(x_0) = \mathcal{N}(x_0; 2, 8)$.

a Generate trajectory and measurement sequence for 30 time steps and run a KF, a PF without resampling and a PF with resampling, to recursively compute $p(x_k|y_{1:k})$, $\mathbb{E}[x_k|y_{1:k}]$ and $\mathrm{Cov}[\mathbf{x}_k|\mathbf{y}_{1:k}]$. Note that for the bootstrap particle filter with resampling, the resampling is usually performed at every time step.

- Compare the performance of the two PFs with the KF in terms of mean square error (MSE). How many particles do you need in order for your two PFs to perform approximately as well as the KF? (Note that these numbers are much larger in higher dimensions.) Comment on your results and observations.
- Plot the true trajectory and the measurement sequence as well as $\mathbb{E}[x_k|y_{1:k}]$ and $\mathrm{Cov}[\mathbf{x}_k|\mathbf{y}_{1:k}]$ for the three filters over time.
  Hint: one way to visualize $\mathbb{E}[x_k|y_{1:k}]$ *together with Cov*$[\mathbf{x}_k|\mathbf{y}_{1:k}]$ *is to use* `errorbar(x,y,err)` *in* MATLAB.
- Illustrate the approximations to the posterior densities that the two PFs produce with the posterior density in KF for three time instances that you choose: one time step near the beginning, one time step near the end and one time step somewhere in between. Comment on your results and observations.
  Hint: the particle filter approximates the posterior distribution as a weighted sum of impulse functions. One way to obtain a nice illustration of the posterior approximation of the particle filter is by placing a narrow Gaussian kernel around each particle. The function plotPostPdf.m *can be passed as a function handle into your already implemented PF function to do that, but note that you need to tune the width of the kernel (*`sigma=1` *may be a good start) for the PF without resampling to obtain a reasonable illustration.*

b For this subtask only you study what happens if you use an incorrect prior at time 0. Keep the trajectory and the measurement sequence the same, but this time run the three filters with prior $p(x_0) = \mathcal{N}(x_0; -20, 2)$. Also, for the two PFs, use the same number of particles as in the last subtask.

Observe how fast the estimates produced by the three different filters can approach to the true trajectory and comment on your findings.

c Illustrate the particle trajectories for a PF *without* resampling (with 100 particles) along with the true trajectory and the estimation in the same figure. Motivate the result and discuss the implications it has on the performance of the filter.
Hint: The function plotPartTrajs.m *can be passed as a function handle into your already implemented PF function to draw the trajectories.*

d Illustrate the particle trajectories for a PF *with* resampling (with 100 particles) along with the true trajectory and the estimation in the same figure. Describe and discuss the difference to the results obtained with a PF *without* resampling.

3 Bicycle tracking in a village

In this task, you will use particle filter to solve a more practical problem.

Suppose we are interested in tracking a bicycle in a village and that we have an accurate map of the buildings in the village, see Fig. 1. It is also assumed that the bicycle must be always on the road between the buildings. The tracking is based on (two-dimensional) measurements of the difference in position. The following problems should be considered:

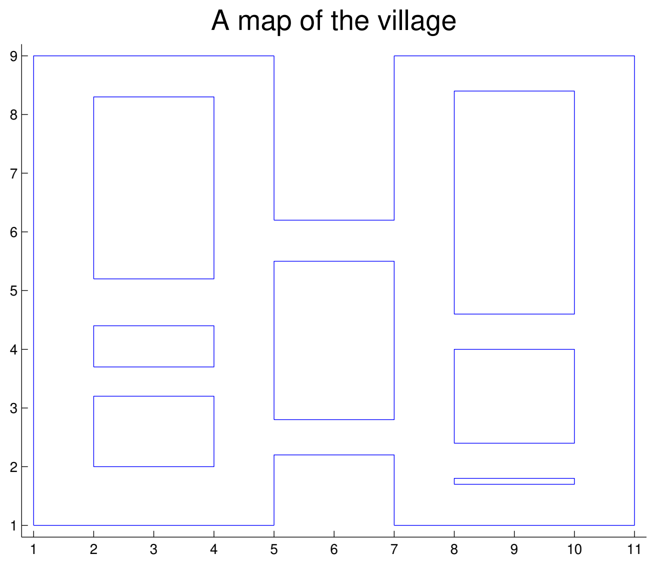

Figure 1: A map of a village in a Cartesian coordinate system.

a Generate a sequence of positions, $\mathbf{x}_k^p$, using *MapProblemGetPoint.m*. Run the file and use the left mouse button to generate a trajectory of the bicycle, press Return to stop collecting measurements. Try to make it resemble the movement of a real bicycle and make sure that the trajectory you generate contains a few turns. Include the figure of this position sequence to your report.

b Generate *velocity* measurements

$$\mathbf{y}_k = \mathbf{x}_k^v + \mathbf{r}_k, \tag{9}$$

where $\mathbf{x}_k^v = \mathbf{x}_k^p - \mathbf{x}_{k-1}^p$ and $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \sigma_r^2 \mathbf{I}_{2x2})$. Include the figure of this velocity measurement sequence to your report.

c Assume that the bicycle is always in the village and must not enter any buildings. Discuss how the information from the map can be used in your particle filter. How do we design the filter if we view the map as a measurement or a component in the motion model? Which way is computationally cheaper? Motivate your answers.

d Construct a PF algorithm with the ability to track both the position and the velocity of the bicycle. Assume that the initial position of the bicycle is known. You need to tune both the measurement and motion noise in order to obtain good performance. It is advisable to illustrate your particles in the village in each recursion in order to visualize the filter performance.

e Generalize the PF such that it can track the bicycle without any knowledge about its initial position. Is your filter able to position the bicycle? If so, how do you initialize the particles and which information is it that enables your filter to do so? Please try to illustrate (or at least point out in words) the events when the filter manages to rule out large chunks of particles.

Hints:

- Ignoring the information from the map, it may be reasonable to use a constant velocity motion model and (9) as measurement model. How well that model fits the data depends on the trajectories that you create (to some extent it is up to you!).
- You can use MATLAB function *isOnRoad.m* to check which points are on a road in the village.