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

Basic information

Please submit all problems in MATLAB Grader, upload your short reports to Canvas, and answer all quiz questions in Canvas no later than **18 May 2020 18:00**.

You are encouraged to discuss the assignments with classmates, but you must write up your own solutions (including Matlab implementations). Also, when writing up the solutions, you should write the names of people with whom you have discussed the assignments. It is important that you make an effort to present/illustrate your solutions nicely and reflect on your results. This means that you should describe the results and explain the mechanisms producing them, that is; **why** do you get the results!

This home assignment is related to the material in lecture 7 and 8. The focus is on the implementation and evaluation of non-linear RTS smoothers, and on importance sampling methods for filtering, i.e. particle filters. To pass the home assignment you need to get a pass on all implementation assignments. To be able to get a higher grade in the course, you need to collect POEs by completing the analysis part and answering the accompanying quiz questions on Canvas.

Now we want you to use the toolbox that you have developed and apply it to practical scenarios. Associated with each scenario is a set of tasks that we would like you to perform, and a set of questions on Ping-Pong that you should answer.

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 clear captions to the figures but they should clearly **explain** what is seen in the figure and answer any related questions in the task. The "report" should also be uploaded to ping-pong before the deadline. Only properly referenced or captioned figures will result in POE.

1 Smoothing

In this scenario we investigate the results from an RTS smoother. To this end 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.

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. The sampling time is $T = 0.1s$. We are going to use the range/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

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

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

Let the sensor position be stationary,

$$\mathbf{s}_k = [280\ ,\ -140]^T, \forall k. \tag{5}$$

The measurement noise standard deviations are known,

$$\sigma_r = 15, \quad \sigma_\varphi = 2\pi/180. \tag{6}$$

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

Task: From the three alternatives, select your favourite non-linear Kalman smoother. Use the CT motion model. From the following 2 subtasks, we expect 2 figures to be included in the report.

a Run the selected smoother on the measurement sequence. Plot the state and measurement sequences in a new figure. Add a plot of the filtered and smoothed position trajectories. Also plot corresponding 3σ-covariance contours for 1/5th of the filter and smoothing estimates.

- Compare the shape of the trajectories and explain why there are differences and/or similarities.
- What are the differences between filter and smoothing error covariances in the same time instance? Explain the reason for the differences.

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(200:400) = -pi/201/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
```

b Modify a measurement at one time instance $k = 150$, such that its corresponding position in the state-space differs significantly from the true state position, say by 100 meters. Run your smoother on the modified measurement sequence and produce a figure illustrating the resulting trajectories in the areas where the outlier has an impact. How well do the filter and smoother cope with the introduced outlier data? Explain the differences.

2 Particle filters for linear and Gaussian systems

Let us study a setting very similar to problem 2 from HA2 such that we can compare a particle filter (PF) with a Kalman filter (KF) 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 $R = 2.5$, respectively, and the initial prior is $p(x_0) = \mathcal{N}(x_0; 2, 6)$.

a) Generate data 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 $\mathrm{Cov}[\mathbf{x}_k|\mathbf{y}_{1:k}]$. (You may already have code that can do this.)
Compare the performance of the PFs with a KF in terms of mean square error (MSE) and by illustrating the approximations to the posterior densities that the three filters produce (the two PF versions and the KF).
How many particles do you need in order for your PFs to perform approximately as well as the KF? (Note that these numbers are much larger in higher dimensions.)

   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 select the width of the kernel to obtain a reasonable illustration.

b) Illustrate the particle trajectories for a PF without resampling along with the true trajectory in the same figure (use a trajectory length of least 20 time steps). 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.

c) Illustrate the particle trajectories for a PF *with* resampling along with the true trajectory in the same figure. Describe and discuss the difference to the results obtained in *b*).

3 Bicycle tracking in a village

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. We know that the bicycle is 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:

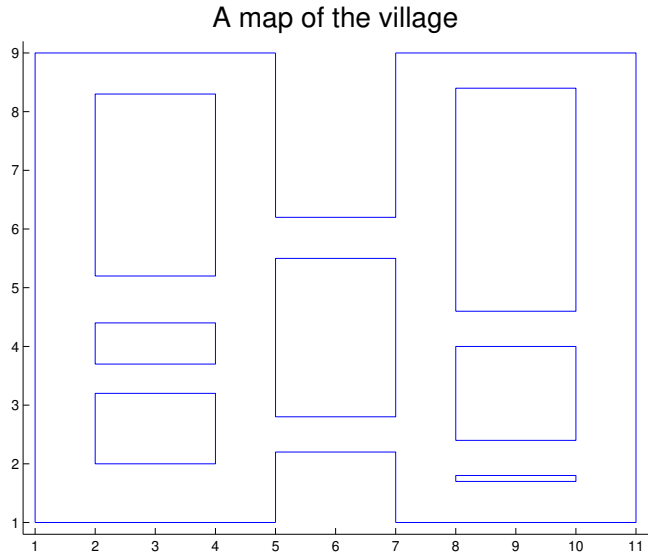

Figure 1: This figure illustrates a map of a village in a cartesian coordinate system.

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.

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 $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \sigma_r^2 \mathbf{I}_{2x2})$.

c) You can assume that the bicycle does not enter any buildings. Discuss how the information from the map can be used in a filter. Is it easier to view the map as a measurement or as a component in the motion model? Try to motivate.

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, which information is it that enables it 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 2-dimensional constant velocity equation as motion model and (9) as measurement model (four state variables). How well that model fits the data depends on the trajectories that you create (to some extent it is up to you!).
- You have access to a Matlab-function *isOnRoad.m* that checks which points are on a road in the village.