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 8 and 9. The focus is on the implementation and evalutation 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 PingPong.

1 Implementation

To build up your own sensor fusion toolbox, we ask you to implement a number of functions according to specification. You should show (both for yourself and us) that your code is working by copying and pasting your code into corresponding problem in Mathworks Cody Coursework here. There you also find more information about, e.g., dimensionality and descriptions of the variable names.

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 pdf and uploaded as part of your submission to pingpong before the deadline. The purpose is to make feedback from your peers possible and to enable plagiarism analysis (Urkund) of all your submissions.

1.1 Non-linear RTS smoothers

In this assignment we want you to implement one recursion of a backwards RTS smoother. Then you will apply it in a RTS smoother function that filters and then smooths an entire sequence of measurements. You will re-use several already implemented functions from HA3.

1.1.1 Smoothing Update

In this assignment we want you to implement the smoothing updates for ERTS, URTS, and CRTS smoothers, respectively.

Assume that a forward non-linear Kalman filter has previously produced the filtered and predicted means and covariances

$$\hat{\mathbf{x}}_{k+1|k}, \quad \mathbf{P}_{k+1|k}, \tag{1}$$

$$\hat{\mathbf{x}}_{k|k}, \quad \mathbf{P}_{k|k}, \tag{2}$$

given $\mathbf{Y}_{1:k}$. Also available is a smoothing (approximated) posterior from the previous backwards smoothing step at time $k+1$, with mean and covariance

$$\hat{\mathbf{x}}_{k+1|K}, \quad \mathbf{P}_{k+1|K}, \tag{3}$$

given $\mathbf{Y}_{1:K}$.

Task: Design and implement a function that approximates the smoothing mean and covariance at time k, using ERTSS, URTSS or CRTSS equations. Note that the sigma point generator function is inserted in your function as a handle.

```
[xs, Ps] = nonLinRTSSupdate(xs_kplus1, Ps_kplus1, xf_k, Pf_k, ...
   xp_kplus1, Pp_kplus1, f, T, sigmaPoints, type)
```

1.1.2 Non-linear RTS smoother

In this assignment we combine filter components developed in HA3 with the smoothing update step that you implemented in the previous task to produce a function that smooths a discrete measurement sequence, given a non-linear state-space model.

Given a sequence of measurements $\mathbf{Y}_{1:K} = [\mathbf{y}_1, \ldots, \mathbf{y}_K]^T$, a prior on $\mathbf{x}_0$, a process model, and a measurement model, we are going to compute and store the smoothed, filtered, and predicted means and covariances

$$\begin{aligned}
&\hat{\mathbf{x}}_{k|K}, \quad \mathbf{P}_{k|K}, && \text{for } k = 1 \ldots K, \\
&\hat{\mathbf{x}}_{k|k}, \quad \mathbf{P}_{k|k}, && \text{for } k = 1 \ldots K, \\
&\hat{\mathbf{x}}_{k|k-1}, \quad \mathbf{P}_{k|k-1}, && \text{for } k = 1 \ldots K.
\end{aligned}$$

Task: Design and implement a function that takes as input a measurement sequence, a state prior, a motion model and a measurement model, and returns smoothed, filtered and predicted estimates, with corresponding covariances. To solve this, you should use the smoothing update function that you have already implemented, and you should also be able to re-use the code that you wrote for the non-linear Kalman filter. However, note that the sigma point generator function is inserted in your function as a handle, and that sigma-point generator should also be used for filtering. Consequently, smaller modifications of the filtering code from HA3 are necessary, such that the `sigmaPoints` function from HA3 is not called directly.

```
[xs, Ps, xf, Pf, xp, Pp] = nonLinRTSsmoother( ...
   Y, x_0, P_0, f, T, Q, S, h, R, sigmaPoints, type)
```

1.2 Importance sampling

In this assignment you will implement the basic components of a particle filter and then combine them into a complete particle filter function.

1.2.1 Resampling

An important method in particle filtering is resampling of the particles as filtering progresses.

Task: Design and implement a function that takes as input a set of particles and their corresponding weights, and then outputs resampled particles with weights. The function should also output a resampling index vector `j`, such that a resampled particle $x_k^{(i)}$ has the parent particle $x_{k-1}^{(j(i))}$.

```
[Xk, Wk, j] = resampl(Xk, Wk)
```

1.2.2 Particle filter update

In this assignment you will implement the update step of a sequential importance sampling filter.

Assume that at time instance $k-1$, the density of state vector $\mathbf{x}_{k-1} \in \mathbb{R}^n$, given $\mathbf{Y}_{1:k-1}$, all measurements up to and including time $k-1$, is approximated by particles $\mathbf{x}_{k-1}^{(i)}$ and corresponding weights $w_{k-1}^{(i)}$ for $i = 1 \dots N$. Additionally, suppose that the process model is given by

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1}, \tag{4}$$

where $\mathbf{q}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$, and that the measurement model is given by

$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{r}_k, \tag{5}$$

where $\mathbf{r}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$.

Task: Use the simple choice of proposal density $q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{y}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$, and implement the function

```
[X_k, W_k] = pfFilterStep(X_kmin1, W_kmin1, yk, proc_f, proc_Q, ...
   meas_h, meas_R)
```

`proc_f` is a handle to a function that computes $f(\mathbf{x}_{k-1})$, and `meas_h` is a handle to a function that computes $h(\mathbf{x}_k)$. Both functions should be able to evaluate several samples of $\mathbf{x}$ simultaneously.

1.2.3 Particle filter

In this assignment we use the previously implemented resampling and particle filter update functions to create a complete SIS/SIR particle filter.

Given a sequence of measurements $\mathbf{Y}_{1:N} = [\mathbf{y}_1, \dots, \mathbf{y}_N]^T$, a prior on $\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \mathbf{P}_0)$, a process model (**??**), and a measurement model (**??**), we are going to compute the posterior state densities in time instances $k = 1 \dots K$ as Monte-Carlo approximations from particles $\mathbf{x}_k^{(i)}$ and corresponding weights $w_k^{(i)}$ for $i = 1 \dots N$. Also, we will compute the approximated mean and covariance at each time instance.

Task: Design and implement a function that takes as input a measurement sequence, a state prior, a motion model and a measurement model, and returns filtered posterior densities as particles with corresponding weights, as well as approximations of the means and covariances. The function should optionally use resampling. To solve the task, you should use the resampling and PF update step functions that you have already implemented.

```
[xfp, Pfp, Xp, Wp] = pfFilter(x_0, P_0, Y, proc_f, proc_Q, ...
```

```
meas_h, meas_R, N, bResample, plotFunc)
```

Beside the function handles also used in the particle filter update step function, this function takes input argument `plotFunc` which is a function handle to a plot function that should be called after the PF update step function has been called. The plot function will be used to create illustrations in the analysis part of the assignment.

2 Analysis

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.

2.1 Smoothing

In this scenario we investigate the results from an RTS smoother. To this end we build on task 2.3 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

$$\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{600} \tag{6}$$

using the code in Table **??**. 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

$$\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{600}. \tag{7}$$

Let the sensor position be stationary,

$$\mathbf{s}_k = [280\ ,\ -140]^T, \forall k. \tag{8}$$

The measurement noise standard deviations are known,

$$\sigma_r = 15, \quad \sigma_\varphi = 2\pi/180. \tag{9}$$

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.

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
```

- 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.

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.

2.2 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:

$$x_k = x_{k-1} + q_{k-1} \tag{10}$$
$$y_k = x_k + r_k, \tag{11}$$

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 both a KF 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 PF 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*).

2.3 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. **??**. 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:

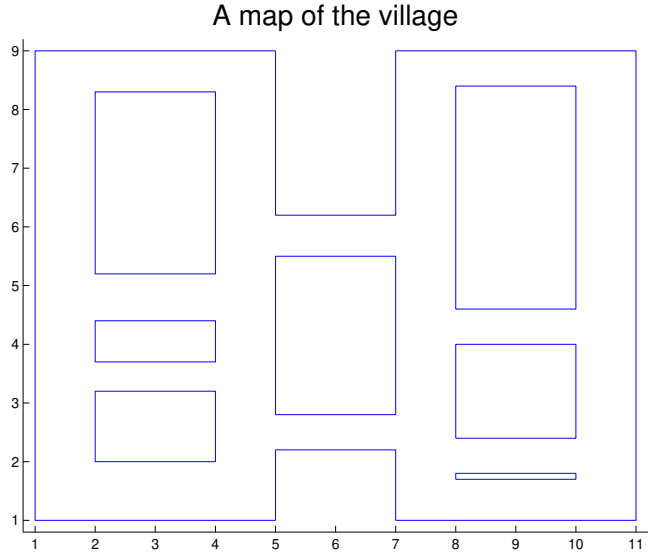

Figure 1: This figure illustrates a map of a village in a cartesian coordinate system.

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

$$\mathbf{y}_k = \mathbf{x}_k^v + \mathbf{r}_k, \tag{12}$$

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 (**??**) 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.