Solutions for examination in Sensor Fusion, 2022-08-17

1. (a) **In the lab gravity is used to indicate down, during acceleration the accelerometer measures both acceleration and gravity, hence "down" is shifted.**
   A suitable outlier detector is give by $\boldsymbol{T = \big|\|y_a\| - \|g\|\big| > T}$ for a suitable threshold $\boldsymbol{T_{\text{th}}}$.

   (b)
$$\boldsymbol{T = 2\log\frac{\mathcal{N}(y;\mu,R)}{\mathcal{N}(0;\mu,R)} = \mu^T R^{-1}\mu - 2y^T R^{-1}\mu \gtrless^{\mathcal{H}_1}_{\mathcal{H}_0} T_{\text{th}}}$$

   The test statistic T follows from the Neyman-Pearson theorem, and T_{th} is a chosen threshold. (The test statistic can also be formulated without the log, only resulting in different threshold values.)

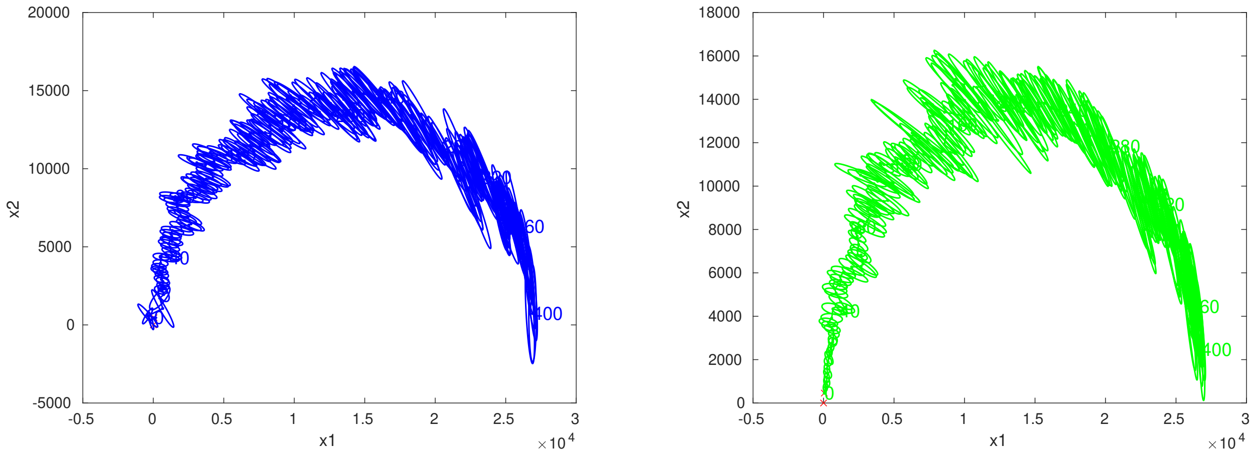
   (c) **Increase $\boldsymbol{Q}$.**
   The estimate follows the measurements (which matches a reasonable intersection) too slowly, resulting in an overshoot. A more responsive filter is obtained using higher process noise.

   (d) $\boldsymbol{\mathcal{N}\left(\begin{pmatrix}1.25\\2.4\end{pmatrix}, \begin{pmatrix}0.125 & 0\\0 & 0.199\end{pmatrix}\right)}$
   The estimates are independent, hence the fusion formula can be used.

   (e) **(i), (iv), (v)**

2.
```
%% Exercise 2
load('data20221020.mat')
Y = sig(ex2_y', .5);

%% 2a
sm = exsensor('radar');  % Construct a measurement model
sm.th = [0;0];  % Radar location
sm.pe = diag([100, 0.1].^2);  % Measurement noise

%% 2b
% To decide on a motion model, look at the trajectory from the measurements
xmeas = [ex2_y(:, 1).*cos(ex2_y(:, 2)), ex2_y(:, 1).*sin(ex2_y(:, 2))];
figure(1); clf;  % Plot the measurements as red crosses
plot(xmeas(:, 1), xmeas(:, 2), 'rx');
hold all

% The measurements seem to describe a circle, use a CT model
mm = exmotion('ctpv2d', .5);  % Set sampling frequency of 2Hz

%% 2c
% Obtain the initial position from the first measurement
mm.x0 = [xmeas(1, :), 0, 0, 0];
% Set the uncertainty of the initial state with the following reasoning:
% - the target is quite far away but the initialization is quite ok
% - the target's initial velocity is much more uncertain
% - the target's initial heading is in radians and very uncertain
% - the target's initial angular velocity is in rad/s and very uncertain
mms.px0 = blkdiag(10*eye(2), 100*eye(1), 3, 3);
% Set the process noise covariance with the following reasoning:
% - Look at the structure of the matrix given with the original motion model
% - The change in velocity needs to be allowed to be fairly large
% - The change in angular velocity needs to be much smaller since the state
%   is in rad/s
mm.pv = diag([0, 0, 500, 0, 0.1].^2);

% Combine the motion model and the measurement model
mms = addsensor(mm, sm);

% Do UKF and plot results with confidence intervals
xukf = ukf(mms, Y);
xplot2(xukf, 'conf', 90);
print(1, '-depsc', fullfile('fig', 'ex2c'));

%% 2d
% Run PF with 1000 particles and plot results with confidence intervals
xpf = pf(mms, Y, 'Np', 1000);
```

```
figure(2), clf,
plot(xmeas(:, 1), xmeas(:, 2), 'rx');
hold all
xplot2([], xpf, 'conf', 90);
print(2, '-depsc', fullfile('fig', 'ex2d'));
```



(a) Exercise 2(b)

(b) Exercise 2(c)

Figure 1: Figures for Exercise 2.

3. (a) To derive the time update, insert the given information into the Bayesian time update equation. Similarly to the point-mass filter, the integrals turn into sums over the discrete state values:

$$p^i_{k|k-1} = p(x_k = s^i|y_{1:k-1}) = \int p(x_k = s^i|x_{k-1})p(x_{k-1}|y_{1:k-1})\, dx_{k-1}$$
$$= \sum_{j=1}^{n} p(x_k = s^i|x_{k-1} = s^j)p(x_{k-1} = s^j|y_{1:k-1}) = \sum_{j=1} \Pi_{ij} p^j_{k-1|k-1}.$$

This can be simplified further if p is a vector comprising the p^i,

$$p_{k|k-1} = \Pi p_{k-1|k-1}.$$

(b) Similarly, insert the given information into the Bayesian measurement update equation (denote $\ell^i_k = p(y_k|x_k = s^i)$, and ℓ_k the matching vector notation):

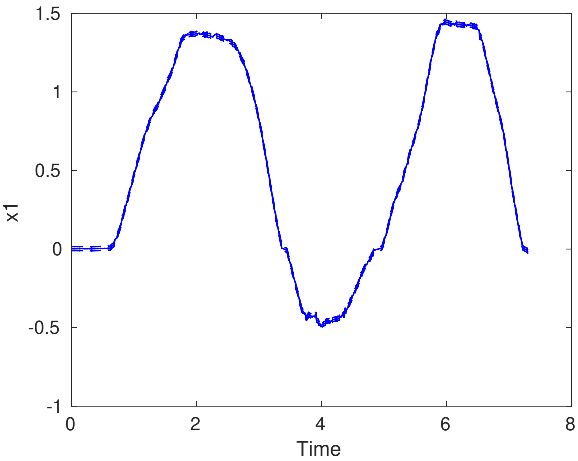
$$p^i_{k|k} = p(x_k = s^i|y_{1:k}) = \frac{p(y_k|x_k = s^i)p(x_k = s^i|y_{1:k-1})}{p(y_k|y_{1:k-1})} = \frac{\ell^i_k p^i_{k|k-1}}{\sum_{j=1}^n \ell^j_k p^j_{k|k-1}}.$$

Using vector notation the expression can be further simplified

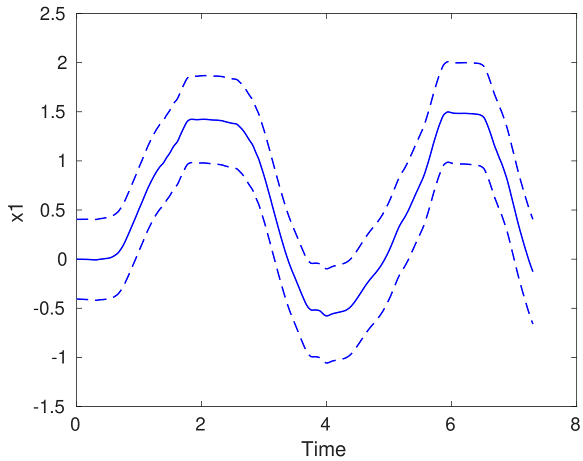
$$p_{k|k} = \frac{\ell_l \odot p_{k|k-1}}{< \ell_l, p_{k|k-1} >},$$

where $\odot$ denotes elementwise multipliction and $< \cdot, \cdot >$ scalar product.

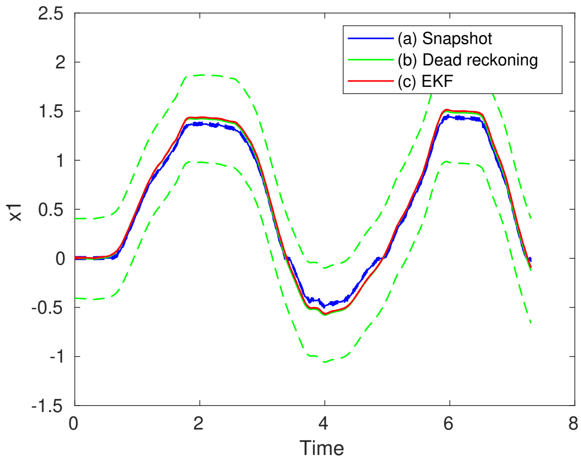
4.
```
%% Exercise 2
load('data20221020.mat')

%% Exercise 4a
% The accelerometer acts as an inclinometer, to estimate the angle x.  In
% the starting position x = 0 (see the figure).  Given the application,
% angles between roughly 0 and pi/4 makes sense.  Initially, the IMU
% measures [-9.82; 0; 0].  The measurement equation is:
%       [ cos(x_k) sin(x_k) 0][-9.82]             [-cos(x_k)]
% y_k = [-sin(x_k) cos(x_k) 0][  0  ] + e_k = 9.82[ sin(x_k)] + e_k,
%       [    0        0     1][  0  ]             [    0    ]
% where y_k is the accelerometer measurement.
% Note:  (1) As the rotation is purely around z, the z component is
% uninteresting (=0).  Hence, use only the xy components as measurements.
% (2) The magnitute is irrelevant, hence its possible (but not required) to
% normalize the measurements, to obtain the measurement equation:
%               [-cos(x_k)]
% y_k/||y_k|| = [ sin(x_k)] + e_k,
% assuming e_k to be Gaussian.

% Normalize the acc measurements
acc = ex4_acc./repmat(sum(ex4_acc.^2,2),[1 3]);
N = size(acc, 1);
T = 1/100;  % Measurement frequency

% Measurement function
h = @(t, x, u, th) [-cos(x) ; sin(x)];
sm = sensormod(h, [1 0 2 0]); % Sensor model
R = 0.01^2 * eye(2); % Set covariance to something reasonable and diagonal
sm.pe = R;
Ya = sig(acc(:, 1:2), 100, [], zeros(N, 0));

xa = zeros(N, 1);  % Collect angle estimates
Pa = zeros(N, 1, 1);  % Collect covaraiances
for i = 1:N
  xnls = estimate(sm, Ya(i));
  xa(i, :) = xnls.x0;
  Pa(i, :, :) = var(xnls.px0);
end
xhata = sig(Ya.y, 100, [], xa, [], Pa);
% Plot results
figure(1); clf;
xplot(xhata, 'conf', 90);
print(1, '-depsc', fullfile('fig', 'ex4a'))

%% Exercise 4b
% Using the gyroscope as input yields standard dead reckoning (where only
% the z axis must be considered.
% x_k = x_k-1 + T u_k-1 + w_k-1,
% where u_k is the z coment of the gyro measurements, T the sample time,
% and w_k-1 is process noise, which is approximated with 0 when
% integrating, but can be used to compute the uncertainty in the estimate.

xb = zeros(N,1);  % Vector to collect all angle estimates
x0 = 0;
P0 = 0.1;
Q = T*.1^2;  % The process noise is a tuning variable, but should be reasonable
xb(1) = x0;  % Define initial angle
% Dead reckoning the angle (keeping track of the covariance is strictly not
% part of the exercise but gives a bit more understanding for the solution.
Pb = zeros(N, 1, 1);
Pb(1, :, :) = P0;
for i = 2:N
    xb(i) = xb(i-1) + T * ex4_gyr(i,3);
    Pb(i, :, :) = Pb(i-1, :, :) + Q;
end
xhatb = sig(zeros(N, 0), 100, [], xb, [], Pb);

% Plot results
figure(2); clf
xplot(xhatb, 'conf', 90)
print(2, '-depsc', fullfile('fig', 'ex4b'))

%% Exercise 4c
% Now simply combine the results in a and b.
```



(a) Estimated knee angle using the measurement model for the accelerometer measurements.



(b) Estimated knee angle using the motion model with the gyroscope measurements as an input.



(c) Combined estimates.

Figure 2: Figures for exercise 4.

```
% Make motion model
f = @(t, x, u, th) x + T*u;
% Combine models into an NL object
mms = nl(f, h, [1 1 2 0], 100);
% Make a sig object with the acc measurements and gyro inputs
Yc = sig(acc(:, 1:2), 100, ex4_gyr(:,3));

% Specify initial state
mms.x0 = x0;

% Specify uncertainties measurements and initial state
mms.px0 = P0;
mms.pe = R;
mms.pv = Q;

% Run EKF
xhatc = ekf(mms, Yc);

% Plot results and compare to previous results
figure(3); clf
xplot(xhata, xhatb, xhatc, 'conf', 90,...
  'legend', {'(a) Snapshot', '(b) Dead reckoning', '(c) EKF'});
print(3, '-depsc', fullfile('fig', 'ex4c'))
```