Applied Signal Processing
Hand-In Problem 3
Object Tracking Using the Kalman Filter

October 23, 2022

This hand-in problem concerns Kalman filtering. The project is individual. You should hand in solutions of the theoretical tasks as well as plots from the Matlab tasks. The code produced in Task 3 should be verified according to the instructions below. The secret key produced by the verification and and your `student_id` should be included in your report. Upload your report to Canvas.

The Kalman filter can be used for many estimation tasks. A classical "Wiener-filter type" example is to recover a distorted and/or disturbed random signal. One of the differences to the Wiener filter is that the Kalman filter is time-varying. This means that it can take initial uncertainty into account in an optimal way, and also that the dynamics describing the signal do not have to be stationary. Like the Wiener filter, the Kalman filter gives an LMMSE (Linear Minimum Mean Square Error) estimate of a signal. In some cases there are better non-linear filters, but not if the signal and noise are jointly Gaussian distributed. The Kalman filter has therefore found a wide use in applications such as adaptive filtering, calibration and object tracking. In this problem you will apply the Kalman filter to track a moving object using 2D camera measurements.

The object is assumed to move freely in the xy-plane, and the position at time t is described by the variables $x(t)$, $y(t)$. We assume that the object is nominally moving at constant speed along a straight line, which means that $\ddot{x}(t) = \ddot{y}(t) = 0$ (zero acceleration). However, to allow for changes in the movement, we will allow for changes in the speed for when deriving the discrete time model later.

The state in the motion model is given by:

$$s(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \\ s_4(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ \dot{x}(t) \\ y(t) \\ \dot{y}(t) \end{bmatrix}$$

(the name $s(t)$ for the state vector is unusual, but is used because $x(t)$ denotes the x-coordinate). The state equations are then

$$\begin{aligned} \dot{s}_1(t) &= s_2(t) \\ \dot{s}_2(t) &= 0 \\ \dot{s}_3(t) &= s_4(t) \\ \dot{s}_4(t) &= 0\,. \end{aligned}$$

Now, measurements usually come in discrete-time. It is then more practical to use a discrete-time model of the motion of the object.

Task 1 (Theory): Apply the finite-difference approximation

$$\dot{x}(t)|_{t=kT} \approx \frac{x(kT+T) - x(kT)}{T}\,,$$

where T is the sampling time, to the continuous-time state equations. Thus, derive a discrete-time state-space model of the form

$$s(k+1) = As(k) + w(k)\,,$$

where A is a 4×4 matrix (from now on we use $s(k)$ to denote discrete time, corresponding to $t = kT$). The symbol $w(k)$ in the equation above denote a zero mean discrete random noise vector, i.e. the discrete time process noise. The covariance matrix of the noise vector should be selected to model possible speed changes between the sampling instances but not influence the position states.

Assume that a camera-based sensor measures the xy-position of the object at discrete time k, with additive noise $v_x(k)$ and $v_y(k)$ respectively. Derive the measurement equation

$$z(k) = Cs(k) + v(k)\,,$$

where $z(k)$ is a 2×1 vector and C is 2×4 matrix.

A real-world object will of course not behave exactly according to our model. The usefulness of the Kalman filter stems from that it can give useful results even when the model is not perfect.

Task 2 (Matlab): Generate x and y coordinates (in discrete time) as follows:

```
x = 0:0.01:9.99; y = sin(0.5*x);
Y = [x;y]; Z = Y + 0.1*randn(size(Y));
```

Now you have 1000 samples of true positions and measurements. Plot the data and the measurements in xy-plots:

```
plot(Y(1,:),Y(2,:),'x'); plot(Z(1,:),Z(2,:),'x')
```

respectively. The measurements are very crude to illustrate the potential of the Kalman filter. In practice, the measurement noise would be much smaller.

The next thing to do is to implement the Kalman filter, as described in the Kalman filter chapter in the lecture notes.

Task 3 (Matlab): Complete the MATLAB-function described in Table 1 (on the next page). It should take input data:

- Y: Measured signal - matrix of size $p \times N$ where N is the number of samples and p the number of outputs
- A: System matrix, $n \times n$
- C: Measurement matrix of size $p \times n$
- Q: Covariance matrix of the process noise, $n \times n$
- R: Covariance matrix of the measurement noise, $p \times p$
- x0: Estimate of $x(0)$, $n \times 1$ (default all zeros)
- P0: Error covariance for $x(0)$, $n \times n$ (default eye(n))

and deliver as output data:

- Xfilt: $[\hat{x}^+(0), \hat{x}^+(1), \hat{x}^+(2), \ldots, \hat{x}^+(N-1)]$, Kalman-filtered estimate of the states
- Pplus: Covariance matrix of $\hat{x}^+(N-1)$, size $n \times n$ (only the last value needs to be saved)

For implementation and validation of your code follow the instructions in the `hip3.m` and `student_sols.m` files provided.

```
function [Xfilt,Pplus] = kalmfilt(Y,A,C,Q,R,x0,P0)

% [Xfilt,Pplus] = kalmfilt(Y,A,C,Q,R,x0,P0)
% Matlab function for Kalman filtering

[p,N] = size(Y); % N = number of samples, p = number of "sensors"
n=length(A);     % n = system order
Xpred = zeros(n,N+1); % Kalman predicted states
Xfilt = zeros(n,N); % Kalman filtered states (after using the measurement)

if nargin < 7
  P0=eye(n); % Default initial covariance
end;
if nargin < 6
  x0=zeros(n,1); % Default initial states
end;

% Filter initialization:
Xpred(:,1) = x0;  % Index 1 means time 0
P = P0;           % Initial covariance matrix (uncertainty)

% Kalman filter iterations:
for t=1:N
  Xfilt(:,t) = Xpred(:,t) + ... % Filter update based on measurement Y(:,t)
  Pplus = P - ...               % Covariance after measurement update
  Xpred(:,t+1) = ...            % Prediction
  P = ...                       % Covariance increase after prediction
end
```

Table 1: Kalman filter MATLAB code.

Task 4: (Matlab) Apply your MATLAB-implementation of the Kalman filter to the object tracking data. Set the sampling time $T = 0.01$s. The input to the Kalman filter is the noisy measurements `Z`. Try to figure out suitable values for Q and R from how the true data are generated. Use the zero vector as initial state vector, and $P_0 = 10^6 \times \mathbf{I}$ as initial state covariance matrix. Investigate how the algorithm behavior changes if you increase/decrease R by a factor of 10. Try to fine-tune the performance of the algorithm by testing different scalings of R (you can verify that increasing R by a factor of 10 has the same effect as decreasing Q and P_0 by the same factor). Plot the estimated object trajectory: `plot(Xfilt(1,:),Xfilt(3,:),'x')`. Also plot the estimated speeds in the x and y directions (versus time). Imagine how a direct finite-difference approximation of the measurements would have looked like and be amazed by the power of the Kalman filter!