SSY345 Sensor fusion and non linear filtering

HA2 Implementation

Isak Åslund (isakas)

April 20, 2020

Matlab code

genLinearStateSequence.m

```
function X = genLinearStateSequence(x_0, P_0, A, Q, N)
%GENLINEARSTATESEQUENCE generates an N-long sequence of states using a
%    Gaussian prior and a linear Gaussian process model
%
%Input:
%   x_0         [n x 1] Prior mean
%   P_0         [n x n] Prior covariance
%   A           [n x n] State transition matrix
%   Q           [n x n] Process noise covariance
%   N           [1 x 1] Number of states to generate
%
%Output:
%   X           [n x N+1] State vector sequence
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your code here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate initial state from prior.
X0 = mvnrnd(x_0, P_0, 1)';

% Generate state sequence by letting the initial state propagate
X = zeros(length(x_0), N+1);
X(:,1) = X0;

for k = 2:N+1
    X(:,k) = mvnrnd(A * X(:,k-1), Q, 1);
end
```

genLinearMeasurementSequence.m

```
function Y = genLinearMeasurementSequence(X, H, R)
%GENLINEARMEASUREMENTSEQUENCE generates a sequence of observations of the state
% sequence X using a linear measurement model. Measurement noise is assumed to be
% zero mean and Gaussian.
%
%Input:
%   X           [n x N+1] State vector sequence. The k:th state vector is X(:,k+1)
%   H           [m x n] Measurement matrix
%   R           [m x m] Measurement noise covariance
%
%Output:
%   Y           [m x N] Measurement sequence
%

% your code here
% Create measurement vector
Y = zeros(size(H,1), size(X,2)-1);

% Create measurement data from all states except the first
for k = 1:size(Y,2)
   Y(:,k) = mvnrnd(H*X(:,k+1), R, 1);
end


end
```

linearPrediction.m

```
function [x, P] = linearPrediction(x, P, A, Q)
%LINEARPREDICTION calculates mean and covariance of predicted state
%   density using a liear Gaussian model.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%   A           [n x n] State transition matrix
%   Q           [n x n] Process noise covariance
%
%Output:
%   x           [n x 1] predicted state mean
%   P           [n x n] predicted state covariance
%

% Your code here

% Calculate posterior from the process model
x = A*x;
P = A*P*A' + Q;

end
```

linearUpdate.m

```
function [x, P, V] = linearUpdate(x, P, y, H, R)
%LINEARPREDICTION calculates mean and covariance of predicted state
%   density using a linear Gaussian model.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%   y           [m x 1] Measurement
%   H           [m x n] Measurement model matrix
%   R           [m x m] Measurement noise covariance
%
%Output:
%   x           [n x 1] updated state mean
%   P           [n x n] updated state covariance
%   V           [m x 1] inovation

% Your code here
S = H*P*H' + R;
K = P*H'*inv(S);
V = y - H*x;

x = x + K*V;
P = P - K*S*K';

end
```

kalmanFilter.m

```
function [X, P, V] = kalmanFilter(Y, x_0, P_0, A, Q, H, R)
%KALMANFILTER Filters measurements sequence Y using a Kalman filter.
%
%Input:
%   Y           [m x N] Measurement sequence
%   x_0         [n x 1] Prior mean
%   P_0         [n x n] Prior covariance
%   A           [n x n] State transition matrix
%   Q           [n x n] Process noise covariance
%   H           [m x n] Measurement model matrix
%   R           [m x m] Measurement noise covariance
%
%Output:
%   x           [n x N] Estimated state vector sequence
%   P           [n x n x N] Filter error convariance
%   V           [m x N] inovation

% Parameters
N = size(Y,2);

n = length(x_0);
m = size(Y,1);

% Data allocation
X = zeros(n,N);
P = zeros(n,n,N);
V = zeros(1,N);

% 1. Predict the next state
% 2. Update the prediction with measurement
% 3. Save the update for next iteration

for k = 1:N
    [x_0, P_0] = linearPrediction(x_0, P_0, A, Q);
    [x_0, P_0, V_0] = linearUpdate(x_0, P_0, Y(:,k), H, R);
    V(:, k) = V_0;
    X(:,k) = x_0;
    P(:,:,k) = P_0;
end
end
```