SSY191 - Sensor Fusion and Nonlinear Filtering
Implementation of Home Assignment 02

Lucas Rath

Listings


```
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
    %

    n = length(x_0);
    X = zeros(n,N);

    % sample initial state from the prior distribution x0~N(x_0,P_0)
    X(:,1) = mvnrnd(x_0, P_0)';
    % iterate to generate N samples
    for i=1:N
        % Motion model: X{k} = A*X{k-1} + q{k-1},   where q{k-1} ~ N(0,Q)
        X(:,i+1) = A * X(:,i) + mvnrnd(zeros(n,1), Q)';
    end

end
```

```
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
    %

    m = size(H,1);
    % state sequence includes x0, which does not generate an observation
    N = size(X,2) -1;
    Y = zeros(m,N);

    % iterate to generate N samples
    for i=1:N
        % Measurement model: Y{k} = H*X{k} + r{k},   where r{k} ~ N(0,R)
        Y(:,i) = H * X(:,i+1) + mvnrnd(zeros(m,1), R)';
    end

end
```

```
function [x, P] = linearPrediction(x, P, A, Q)
    % 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
    %   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
    %

    % Use motion model for prediction
    % \hat{x}_{k|k-1} = A_{k-1} * \hat{x}_{k-1|k-1}
    x = A * x;
    % Compute covariance of new prediction:
    % Cov(x_{k|k-1}) = Cov(A_{k-1} * x_{k-1|k-1} + q) = A*P*A' + Q
    P = A * P * A' + Q;

end
```

```matlab
function [x, P] = linearUpdate(x, P, y, H, R)
    % LINEARUPDATE 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
    %

    % inovation mean
    v = y - H * x;
    % inovation covariance
    S = H * P * H' + R;
    % kalman gain
    K = P * H' / S;

    % updated state mean
    x = x + K * v;
    % updated state covariance
    P = P - K * S * K';

end
```

```matlab
function [X, P] = 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
    %

    %% Parameters
    N = size(Y,2);

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

    %% Data allocation
    x = zeros(n,   N +1);
    P = zeros(n,n, N +1);

    %% filter
    x(:,1)   = x_0;
    P(:,:,1) = P_0;

    for i=1:N
        % prediction step: compute p(x_k | y_1:k-1) from p(x_k-1 | y_1:k-1)
        [x(:,i+1), P(:,:,i+1)] = linearPrediction(x(:,i), P(:,:,i), A, Q);
        % update p(x_k | y_1:k-1) from p(x_k | y_1:k-1)
        [x(:,i+1), P(:,:,i+1)] = linearUpdate(x(:,i+1), P(:,:,i+1), Y(:,i), H, R);
    end

    % exclude prior x0~N(x_0, P_0) from posterior
    X = x(:,2:end);
    P = P(:,:,2:end);

end
```