SSY191 - Sensor Fusion and Nonlinear Filtering
Implementation of Home Assignment 01

Lucas Rath

Listings


```
function [ xy ] = sigmaEllipse2D( mu, Sigma, level, npoints )
    %SIGMAELLIPSE2D generates x,y-points which lie on the ellipse describing
    % a sigma level in the Gaussian density defined by mean and covariance.
    %
    %Input:
    %   MU          [2 x 1] Mean of the Gaussian density
    %   SIGMA       [2 x 2] Covariance matrix of the Gaussian density
    %   LEVEL       Which sigma level curve to plot. Can take any positive value,
    %               but common choices are 1, 2 or 3. Default = 3.
    %   NPOINTS     Number of points on the ellipse to generate. Default = 32.
    %
    %Output:
    %   XY          [2 x npoints] matrix. First row holds x-coordinates, second
    %               row holds the y-coordinates. First and last columns should
    %               be the same point, to create a closed curve.


    %Setting default values, in case only mu and Sigma are specified.
    if nargin < 3
        level = 3;
    end
    if nargin < 4
        npoints = 32;
    end

    % Procedure:
    % - A 3 sigma level curve is given by {x} such that (x-mux)'*Q^-1*(x-mux) = 3^2
    %       or in scalar form: (x-mux) = sqrt(Q)*3
    % - replacing z= sqrtm(Q^-1)*(x-mux), such that we have now z'*z = 3^2
    %       which is now a circle with radius equal 3.
    % - Sampling in z, we have z = 3*[cos(theta); sin(theta)]', for theta=1:2*pi
    % - Back to x we get:  x = mux  + 3* sqrtm(Q)*[cos(theta); sin(theta)]'

    xy = [];
    for ang = linspace(0,2*pi,npoints)
        xy(:,end+1) = mu + level * sqrtm(Sigma) * [cos(ang) sin(ang)]';
    end
end
```

```
function [mu_y, Sigma_y] = affineGaussianTransform(mu_x, Sigma_x, A, b)
    %affineTransformGauss calculates the mean and covariance of y, the
    %transformed variable, exactly when the function, f, is defined as
    %y = f(x) = Ax + b, where A is a matrix, b is a vector of the same
    %dimensions as y, and x is a Gaussian random variable.
    %
    %Input
    %   MU_X        [n x 1] Expected value of x.
    %   SIGMA_X     [n x n] Covariance of x.
    %   A           [m x n] Linear transform matrix.
    %   B           [m x 1] Constant part of the affine transformation.
    %
    %Output
    %   MU_Y        [m x 1] Expected value of y.
    %   SIGMA_Y     [m x m] Covariance of y.


    % E[A*x + b] = A*E[x] + b = A*mu_x + b
    mu_y = A * mu_x + b;
    % Cov[A*x + b] = A*Cov[x]*A^T
    Sigma_y = A * Sigma_x * A';
end
```

```
function [mu_y, Sigma_y, y_s] = approxGaussianTransform(mu_x, Sigma_x, f, N)
    %approxGaussianTransform takes a Gaussian density and a transformation
    %function and calculates the mean and covariance of the transformed density.
    %
    %Inputs
    %   MU_X        [m x 1] Expected value of x.
    %   SIGMA_X     [m x m] Covariance of x.
    %   F           [Function handle] Function which maps a [m x 1] dimensional
    %               vector into another vector of size [n x 1].
    %   N           Number of samples to draw. Default = 5000.
    %
    %Output
    %   MU_Y        [n x 1] Approximated mean of y.
    %   SIGMA_Y     [n x n] Approximated covariance of y.
    %   ys          [n x N] Samples propagated through f


    if nargin < 4
        N = 5000;
    end

    % sample in the original gaussian distribution
    x_s = mvnrnd(mu_x, Sigma_x, N)';
    % apply general non-linear transformation function to samples
    y_s = f(x_s);
    % calculate mean of the transformed samples
    mu_y = mean(y_s,2);
    % calculate estimated unbiased covariance of the transformed samples
    Sigma_y = 1/(N-1) * (y_s - mu_y) * (y_s - mu_y)';
end
```

```
function [mu, Sigma] = jointGaussian(mu_x, sigma2_x, sigma2_r)
    % JointGaussian calculates the joint Gaussian density p([y;x])
    %
    % y = x + r
    %               x ~ N(mu_x,sigma2_x)
    %               r ~ N(0   ,sigma2_r)
    %
    %  =>  p([y;x]) = p(A*[x;r] + b) = p( [1 0; 1 1]*[x;r] + [0;0] )
    %
    % Input
    %   MU_X        Expected value of x
    %   SIGMA2_X    Covariance of x
    %   SIGMA2_R    Covariance of the noise r
    %
    % Output
    %   MU          Mean of joint density
    %   SIGMA       Covariance of joint density

    % define linear transformations matrices [x;y] = A*[x;r] + b according to
    % problem 1.3a
    A_xr2xy  = [1 0; 1 1];
    b_xr2xy  = [0;0];

    % define mean of [x;r] = [E[x] ; E[r]] = [mu_x, mu_r=0]
    mu_xr    = [mu_x; 0];

    % define covariance of [x;r]. Since they are independent,
    % then cov[x;r] = [ cov[x] 0; 0; cov[r] ] = diag(cov[x], cov[r])
    Sigma_xr = blkdiag(sigma2_x,sigma2_r);

    % calculate mean and cov of the new vector [x;y], which is obtained from a
    % linear transformation [x;y] = A*[x;r] + b
    [mu, Sigma] = affineGaussianTransform(mu_xr, Sigma_xr, A_xr2xy, b_xr2xy);
end
```

```
function [mu, sigma2] = posteriorGaussian(mu_x, sigma2_x, y, sigma2_r)
    % Calculates the posterior p(x|y) which is proportional to p(x,y)
    % posteriorGaussian performs a single scalar measurement update with a
    % measurement model which is simply "y = x + noise".
    %
    % Input
    %   MU_X             The mean of the (Gaussian) prior density.
    %   SIGMA2_X         The variance of the (Gaussian) prior density.
    %   SIGMA2_R         The variance of the measurement noise.
    %   Y                The given measurement.
    %
    % Output
    %   MU               The mean of the (Gaussian) posterior distribution
    %   SIGMA2           The variance of the (Gaussian) posterior distribution

    % % Calculate Joint distribution p(x,y) which is proportional to p(x|y)
    % syms mux sx sr x y
    % [mu_xy, Q_xy] = jointGaussian(mux, sx, sr);

    % % Express Gaussian distribution p(x,y)= \propto p(x|y) in terms of x (given y)
    % syms munew snew c
    % eqq1 = ([x;y] - mu_xy ).' * Q_xy^-1  * ([x;y] - mu_xy )     ;
    % eqq2 = (  x    - munew ).  * snew^-1  * (  x    - munew )  + c;
    % sol = solve( coeffs( eqq1 - eqq2, x) , [munew snew c]);

    % % Show results
    % simplify(sol.munew);  % -> (mux*sr + sx*y)/(sr + sx)
    % simplify(sol.snew);   % -> (sr*sx)/(sr + sx)
    % simplify(sol.c);      % -> (mux - y)^2/(sr + sx)

    % apply results from symbolic calculations
    mu  = (mu_x*sigma2_r + sigma2_x*y)/(sigma2_r + sigma2_x);
    sigma2 = (1/sigma2_r + 1/sigma2_x)^-1;
end
```

```
function [ xHat ] = gaussMixMMSEEst( w, mu, sigma2 )
    %GAUSSMIXMMSEEST calculates the MMSE estimate from a Gaussian mixture
    %density with multiple components.
    %
    %Input
    %   W            Vector of all the weights
    %   MU           Vector containing the means of all components
    %   SIGMA2       Vector containing the variances of all components
    %
    %Output
    %   xHat         MMSE estimate

    % The MMSE estimator of x, given some observation y, is equal to E[x|y]
    % where E[x|y] is the mean of mix of multiple Gaussian densities p(x|y):
    %
    % p(x|y) = w1 * N1(x;mu,var) +...+ wn * Nn(x;mu,var),   |w|_1=1, 0<wi<1
    %
    % In case of a mixture of one-dimensional Gaussian distributions
    % weighted by wi, with means mui and variances si2, the MMSE will be:
    %
    % x_{MMSE} = E[x|y] = sum_i { wi *   i } = w' *   i

    xHat = w(:)'*mu(:);
end
```