SSY345 Model Predictive Control

HA1 Implementation

Isak Åslund (isakas)

March 29, 2020

Matlab code

sigmaEllipse2D.m

```
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

    %Your code here

    %Evenly spaced points
    theta = linspace(0, 2*pi, npoints);

    %Level curve
    xy = mu + level*sqrtm(Sigma) * [cos(theta); sin(theta)];

end
```

affineGaussianTransform.m

```
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.

    %Your code here

    % Mean - Expected value of E[y]=E[Ax+b] = AE[x] + b
    mu_y = A*mu_x + b;

    % Variance - Expected value of E[(y-E[y])(y-E[y])']
    Sigma_y = A*Sigma_x*A';
end
```

approxGaussianTransform.m

```
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

    %Your code here

    % 1. Draw N samples from the gaussian density, given input and call them x_s
    x_s = mvnrnd(mu_x, Sigma_x, N);

    % 2. For each vector in x_s, apply y=f(x). Concat the result y into y_s
    for i = 1:N
       y_s(i,:) = f(x_s(i,:)');
    end

    % 3. Calculate mean and covar of y_s'
    mu_y = mean(y_s)';
    Sigma_y = cov(y_s);
    y_s = y_s';

end
```

jointGaussian.m

```
function [mu, Sigma] = jointGaussian(mu_x, sigma2_x, sigma2_r)
    %jointGaussian calculates the joint Gaussian density as defined
    %in problem 1.3a.
    %
    %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


    %Your code here
    [mu, Sigma] = affineGaussianTransform([mu_x; 0], [sigma2_x 0; 0 sigma2_r], [1 0;

end
```

posteriorGaussian.m

```
function [mu, sigma2] = posteriorGaussian(mu_x, sigma2_x, y, sigma2_r)
    %posteriorGaussian performs a single scalar measurement update with a
    %measurement model which is simply "y = x + noise".
    %
    %Input
    %   MU_P            The mean of the (Gaussian) prior density.
    %   SIGMA2_P        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 variance by expanding the exponentials and grouping
    % terms of x together.
    % Then match these against our expected dist., which is a
    % normal dist. And we get the following:
    sigma2 = sigma2_x * sigma2_r / (sigma2_x + sigma2_r);
    mu = sigma2_x/(sigma2_x + sigma2_r) * y + sigma2_r/(sigma2_x + sigma2_r) * mu_x;

end
```

gaussMixMMSEEst.m

```
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
    xHat = sum(w.*mu);
end
```