SSY191 - Sensor Fusion and Nonlinear Filtering
Implementation of Home Assignment 03

Lucas Rath

Listings


```
%% Question 1 - Approximations of mean and covariance

close all; clear all; clc;
cp = fp.getColor(1:10);

% number of samples to estimate mean and covariance of the transformed gaussian
N=10000;
% type of filter update
type = 'CKF';    % {'EKF', 'UKF', 'CKF'}
% choose between two prior distributions
distribution = 2; % 1 or 2


% Set distributions
if distribution == 1
    x_mu = [120 120]';
    x_sigma  = diag([5^2 10^2]);
else
    x_mu = [120 -20]';
    x_sigma  = diag([5^2 10^2]);
end

% Sensor positions
s1 = [0, 100]';
s2 = [100, 0]';

% Noise covariance
R = diag([0.1*pi/180 0.1*pi/180].^2);
% Measurement model y = h(x) + R
h = @(x) dualBearingMeasurement(x,s1,s2);



% approximate the transformed gaussian distribution as a gaussian
[y_mu, y_sigma, y, x] = approxGaussianTransform( x_mu, x_sigma, @(x)genNonLinearMeasurementSequence(

% estimate transformed mean and covariance
if strcmp(type,'UKF') || strcmp(type,'CKF')
    [SP1,W1] = sigmaPoints(x_mu,x_sigma,type)
    hSP1 = h(SP1)
    [ye_mu, ye_sigma]  = estimateSP(h, R, SP1, W1)
elseif strcmp(type,'EKF')
    [hx, dhx] = h(x_mu);
    ye_mu = hx;
    ye_sigma = dhx * x_sigma * dhx' + R;
```

```
end


figure('Color','white','Position',[651  364  588  441]);
grid on; hold on %, axis equal
sc1 = scatter(y(1,:), y(2,:), 20, 'filled', 'MarkerFaceColor', cp(1,:), 'MarkerFaceAlpha',0.1, 'Disp

[ xy ] = sigmaEllipse2D( y_mu, y_sigma, 3, 100 );
p1 = plot(xy(1,:),xy(2,:), 'Color', cp(2,:), 'LineWidth',2, 'DisplayName','Sample 3-sigma ellipse');
sc2 = scatter(y_mu(1), y_mu(2), 100, 'o', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', cp(2,:), 'Marker
% sc2 = scatter(y_mu(1), y_mu(2), 100, 'h','filled', 'MarkerFaceAlpha',0.5, 'MarkerFaceColor', cp(2,

if strcmp(type,'UKF') || strcmp(type,'CKF')
    sc3 = scatter(hSP1(1,:), hSP1(2,:), 100, 'h','filled', 'MarkerFaceAlpha',1, 'MarkerFaceColor', c
end

sc4 = scatter(ye_mu(1,:), ye_mu(2,:), 100, 'o','filled', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', c
% sc4 = scatter(ye_mu(1,:), ye_mu(2,:), 100, 'h','filled', 'MarkerFaceAlpha',0.5, 'MarkerFaceColor',
[ xy ] = sigmaEllipse2D( ye_mu, ye_sigma, 3, 100 );
p2 = plot(xy(1,:),xy(2,:), '--', 'Color', cp(4,:), 'LineWidth',3,'DisplayName','Approximated 3-sigma

xlabel 'y[1] - \phi_1', ylabel 'y[2] - \phi_2'
title(sprintf('Filter type: %s, x~p%d(x)',type,distribution))
legend('Location','southeast')

fp.savefig(sprintf('q1_t_%s_d_%d',type,distribution));


%% Question 2 - A) Non-linear Kalman filtering

close all; clear all; clc;
cp = fp.getColor(1:10);

% select case
icase = 2;  % {1,2}
% generated sequence length
N=100;


% Prior information
x_0 = [0 0 14 0 0]';
P_0 = diag([10 10 2 pi/180 5*pi/180].^2);
% Sampling time
T = 1;
% Sensor positions
s1 = [-200 100]';
s2 = [-200 -100]';


sigma_v = 1;
sigma_w = pi/180;
if icase==1
    sigma_phi1 = 10*pi/180;
else
    sigma_phi1 = 0.5*pi/180;
end
sigma_phi2 = 0.5*pi/180;

Q = diag([0 0 T*sigma_v^2 0 T*sigma_w^2]);
R = diag([sigma_phi1^2 sigma_phi2^2]);



% generate state sequence
```

```
f = @(x) coordinatedTurnMotion(x,T);
X = genNonLinearStateSequence(x_0,P_0,f,Q,N);
% generate measurement sequence
h = @(x) dualBearingMeasurement(x,s1,s2);
Y = genNonLinearMeasurementSequence(X,h,R);

% calcualte unfiltered position from sensors given angles
Xm(1,:) = ( s2(2)-s1(2) + tan(Y(1,:))*s1(1) - tan(Y(2,:))*s2(1) ) ./ ( tan(Y(1,:)) - tan(Y(2,:)) );
Xm(2,:) = s1(2) + tan(Y(1,:)) .* ( Xm(1,:) - s1(1) );



for type = {'EKF','UKF','CKF'}

    % filter
    [xf, Pf, xp, Pp] = nonLinearKalmanFilter(Y, x_0, P_0, f, Q, h, R, type{1});

    figure('Color','white','Position',[837  424  603  429]);
    grid on; hold on %, axis equal

    for i=1:5:length(xf)
        ell_xy = sigmaEllipse2D(xf(1:2,i),Pf(1:2,1:2,i),3,50);
    %     fill(ell_xy(1,:),ell_xy(2,:), '--', 'Color',cp(5,:), 'DisplayName','3-sigma level');
        p4 = fill(ell_xy(1,:),ell_xy(2,:), cp(5,:),'facealpha',.2, 'DisplayName','3-sigma level');
%, 'edgecolor','none'
    end

    p1 = plot(X(1,:),X(2,:), 'Color', cp(1,:), 'LineWidth',2, 'DisplayName','True position sequence'
    p2 = plot(xf(1,:),xf(2,:), 'Color', cp(2,:), 'LineWidth',2, 'DisplayName','Sensor position');

    sc1 = scatter(s1(1), s1(2), 100, 'o', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', cp(4,:), 'Marker
    sc2 = scatter(s2(1), s2(2), 200, 'h', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', cp(4,:), 'Marker

    axis manual
    p3 = plot(Xm(1,:),Xm(2,:), 'Color', [cp(3,:) 0.8], 'LineWidth',1, 'DisplayName','Measured positi

    xlabel 'pos x', ylabel 'pos y'
    title(sprintf('Case %d, filter type: %s',icase,type{1}))
    legend([p1 p2 p3 p4 sc1 sc2], 'Location','southwest')

%     fp.savefig(sprintf('q2_t_%s_c_%d','CKF', icase))
%     fp.savefig(sprintf('q2_t_%s_c_%d',type{1}, icase))
end


%% Question 2 - C)


% perform Monte-Carlo simulation
MC = 100;
type = {'EKF','UKF','CKF'};

est_err = cell(2,3);

for imc = 1:MC
    for icase = 1:2

        sigma_v = 1;
        sigma_w = pi/180;
        if icase==1
            sigma_phi1 = 10*pi/180;
        else
            sigma_phi1 = 0.5*pi/180;
        end
```

```
        sigma_phi2 = 0.5*pi/180;

        Q = diag([0 0 T*sigma_v^2 0 T*sigma_w^2]);
        R = diag([sigma_phi1^2 sigma_phi2^2]);

        % Simulate state sequence
        X = genNonLinearStateSequence(x_0, P_0, f, Q, N);
        % Simulate measurements
        Y = genNonLinearMeasurementSequence(X, h, R);

        for itype = 1:numel(type)
            % Run Kalman filter (you need to run all three, for comparison)
            [xf,Pf,xp,Pp] = nonLinearKalmanFilter(Y,x_0,P_0,f,Q,h,R,type{itype});
            % Save the estimation errors and the prediction errors
            est_err{icase,itype}(1:2,end+1:end+length(xf)) = X(1:2,2:end) - xf(1:2,:);
        end
    end
    imc
end



%% plot histogram
close all;

bins = 100;
close all;
pos = {'x','y'};
for icase = 1:2
    figure('Color','white','Position',[381   314   1012   537]);
    sgtitle(sprintf('Normalized histogram of the position error, case: %d',icase))
    for itype = 1:numel(type)
        for ipos = 1:numel(pos)
            subplot(2,3, itype + (ipos-1)*numel(type) );
            hold on;

            idata = est_err{icase,itype}(ipos,:);
            mu   = mean(idata);
            stddev = std(idata);

            % remove outliers
            idx = abs(idata-mu) < stddev*3;
            idata = idata(idx);

            histogram( idata, bins ,'Normalization','pdf','DisplayName','histogram MSE of position e

            [x,y] = normpdf2(mu, stddev^2, 3, 100);
            plot(x,y, 'LineWidth',2, 'DisplayName', sprintf('gaussian N(x; 0, $P_{N|N})$') );

            xlims = max(abs(idata));
            xlim([-xlims xlims]);

            xlabel(sprintf('pos-%s error',pos{ipos}))
            title(sprintf('filter type: %s, pos-%s \n mean: %.2f, std.dev: %.1f',type{itype},pos{ipo
        end
    end
    fp.savefig(sprintf('q2_hist_c_%d',icase))
end
% close all;


%% Question 3
```

```
clear all; close all; clc;
cp = fp.getColor(1:10);

% xf(3,:)

sigma_v = 1         *1e-4;
sigma_w = pi/180 ;

name2save = 'S';
savefig = false;

% True track
% Sampling period
T = 0.1;
% Length of time sequence
K = 600;
% Allocate memory
omega = zeros(1,K+1);
% Turn rate
omega(200:400) = -pi/201/T;
% Initial state
x0 = [0 0 20 0 omega(1)]';
% Allocate memory
X = zeros(length(x0),K+1);
X(:,1) = x0;
% Create true track
for i=2:K+1
    % Simulate
    X(:,i) = coordinatedTurnMotion(X(:,i-1), T);
    % Set turn rate
    X(5,i) = omega(i);
end


% Prior information
x_0 = [0 0 0 0 0]';
P_0 = diag([10 10 10 5*pi/180 pi/180].^2);
% Sensor positions
s1 = [280 -80]';
s2 = [280 -200]';

% measurement noise
R = diag([4*pi/180 4*pi/180].^2);
% generate measurement sequence
h = @(x) dualBearingMeasurement(x,s1,s2);
Y = genNonLinearMeasurementSequence(X,h,R);



% Motion model
f = @(x) coordinatedTurnMotion(x,T);
Q = diag([0 0 T*sigma_v^2 0 T*sigma_w^2]);

[xf, Pf, xp, Pp] = nonLinearKalmanFilter(Y, x_0, P_0, f, Q, h, R, 'CKF');


% calcualte unfiltered position from sensors given angles
Xm(1,:) = ( s2(2)-s1(2) + tan(Y(1,:))*s1(1) - tan(Y(2,:))*s2(1) ) ./ ( tan(Y(1,:)) - tan(Y(2,:)) );
Xm(2,:) = s1(2) + tan(Y(1,:)) .* ( Xm(1,:) - s1(1) );



% figure('Color','white','Position',[758  175  603  429]);
```

```
figure('Color','white','Position',[520  180  654  417]);
grid on; hold on, axis equal;

for i=1:15:length(xf)
    ell_xy = sigmaEllipse2D(xf(1:2,i),Pf(1:2,1:2,i),3,50);
%     fill(ell_xy(1,:),ell_xy(2,:), '--', 'Color',cp(5,:), 'DisplayName','3-sigma level');
    p4 = fill(ell_xy(1,:),ell_xy(2,:), cp(5,:),'facealpha',.1, 'DisplayName','3-sigma level');
%,'edgecolor','none'
end

p1 = plot(X(1,:),X(2,:), 'Color', cp(1,:), 'LineWidth',2, 'DisplayName','True position sequence');
p2 = plot(xf(1,:),xf(2,:), 'Color', cp(2,:), 'LineWidth',2, 'DisplayName','Sensor position');
sc1 = scatter(s1(1), s1(2), 100, 'o', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', cp(4,:), 'MarkerEdge
sc2 = scatter(s2(1), s2(2), 200, 'h', 'MarkerFaceAlpha',0.8, 'MarkerFaceColor', cp(4,:), 'MarkerEdge

axis manual
p3 = plot(Xm(1,:),Xm(2,:), 'Color', [cp(3,:) 0.3], 'LineWidth',1, 'DisplayName','Measured position')
% p3 = scatter(Xm(1,:),Xm(2,:), 40, 'o', 'MarkerFaceAlpha',0, 'MarkerFaceColor', cp(3,:), 'MarkerEdg

xlabel 'pos x', ylabel 'pos y'
% title(sprintf('Case %d, filter type: %s',icase,type{1}))
legend([p1 p2 p3 p4 sc1 sc2], 'Location','west')
% if savefig fp.savefig(sprintf('q3_%s',name2save)); end



% plot position error
figure('Color','white','Position',[428  692  930  207]);
grid on, hold on;
plot( (1:K)*T, vecnorm(xf(1:2,:)-X(1:2,2:end), 2, 1) , 'LineWidth',1)
ylabel('$|p_k - \hat{p_{k|k}}|_2$', 'Interpreter','Latex', 'FontSize',16), xlabel('Time [s]')
title 'Position error'
% if savefig fp.savefig(sprintf('q3_%s_err',name2save)); end



%% Help functions

function [x, P] = estimateSP(f, R, SP, W)
    n = size(SP,1);
    x = zeros(n,1);
    for i=1:numel(W)
        x = x + f(SP(:,i)) * W(i);
    end
    P = R; %zeros(n,n);
    for i=1:numel(W)
        P = P + (f(SP(:,i))-x)*(f(SP(:,i))-x).' * W(i);
    end
end


function [x,y] = normpdf2(mu, sigma2, level, N)
    x = linspace(mu-level*sqrt(sigma2), mu+level*sqrt(sigma2), N);
    y = normpdf(x, mu, sqrt(sigma2));
end
```