CS 8790: Report for assignment 7

Chanmann Lim

November 10, 2014

Report:

The state of a target moving with constant velocity in 2D coordinate is observed. A sequence of the 2D measurements come with the corresponding *timestamp* at which each observation was taken, the mean (x, y) and the upper triangular elements of the square root of the covariance $sqrtm(R)$ thus observation is now in the form of (t_z, z, R).

One way to start is to initialize our filter with time zero, zero mean and velocity and infinite covariance.

$$t = 0, x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, P = \begin{bmatrix} 100000000 & 0 & 0 & 0 \\ 0 & 100000000 & 0 & 0 \\ 0 & 0 & 100000000 & 0 \\ 0 & 0 & 0 & 100000000 \end{bmatrix}$$

Yet, combining the estimate and the observation is not feasible unless both measurements are valid at the same *timestamp* which lead us to make prediction of the state of the estimate into the same timestamp as the observation using constant velocity model F.

$$F \quad (x, P) \rightarrow (Fx, FPF')$$

, where

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

And since the observation comes in lower dimension (without velocity) than the state estimate we have to use transformation matrix H to project down the state estimate before combining them using innovation fusion equation.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

1.1 The state of the target at the time of the final observation is :

$$\text{mean with standard deviation} = \begin{pmatrix} 245.277904 & , & 0.002747 \\ 510.449761 & , & 0.001239 \\ 0.280005 & , & 0.000007 \\ 0.600000 & , & 0.000002 \end{pmatrix}$$

$$x\_final = \begin{bmatrix} 245.277904 \\ 510.449761 \\ 0.280005 \\ 0.600000 \end{bmatrix}$$

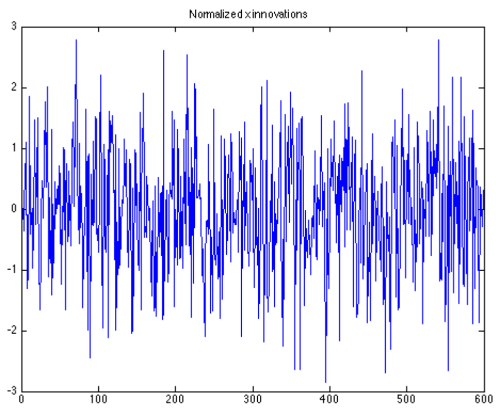
$$P\_final = \begin{bmatrix} 0.00000754 & -0.00000074 & 0.00000002 & -0.00000000 \\ -0.00000074 & 0.00000153 & -0.00000000 & 0.00000000 \\ 0.00000002 & -0.00000000 & 0.00000000 & -0.00000000 \\ -0.00000000 & 0.00000000 & -0.00000000 & 0.00000000 \end{bmatrix}$$

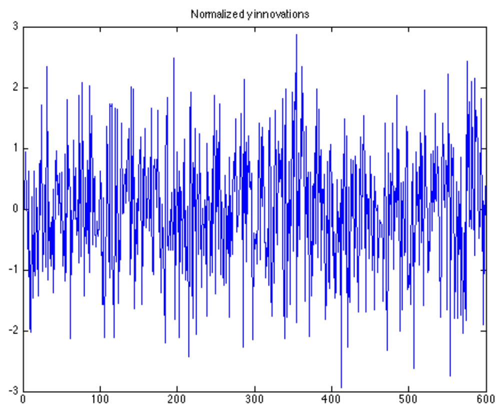
1.2 The prediction of the state of the target one hour after the time of the final observation ($t = 927.969649836$) :

$$x(4527.969650) = \begin{bmatrix} 1253.296210 \\ 2670.451308 \\ 0.280005 \\ 0.600000 \end{bmatrix}$$

$$P(4527.969650) = \begin{bmatrix} 0.00082509 & -0.00013653 & 0.00000021 & -0.00000003 \\ -0.00013653 & 0.00008861 & -0.00000004 & 0.00000002 \\ 0.00000021 & -0.00000004 & 0.00000000 & -0.00000000 \\ -0.00000003 & 0.00000002 & -0.00000000 & 0.00000000 \end{bmatrix}$$

1.3 The plots of the sequence of normalized x and y innovations separately :

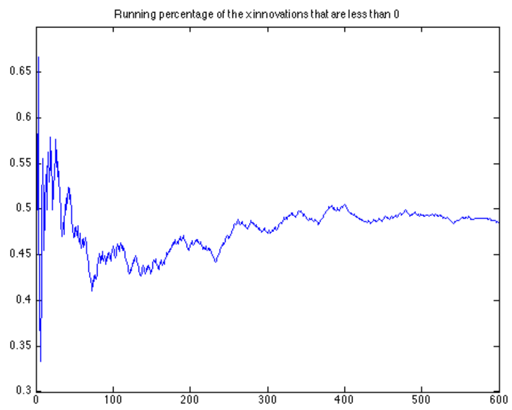


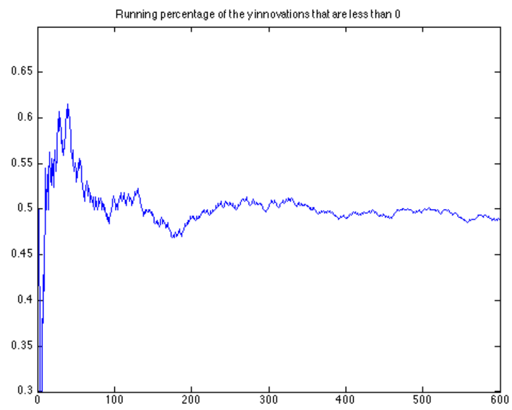


The normalized x innovations is obtained by $(z - Hx)(1)/\sqrt{S(1,1)}$ and y innovations $= (z - Hx)(2)/\sqrt{S(2,2)}$, where S is the innovation covariance.

1.4 The Percentage of x and y innovations that are less than 0 is $[48.50\%, 49.00\%]$

The plot of the running percentage of the x and y innovations that are less than zero :





For the second dataset, `Target2.txt`, the motion of the target deviates slightly from true constant velocity and the uncertainty increases thus in order to account for the deviation we will have to make the covariance bigger by incorporating the process noise covariance matrix Q while predicting the state estimate into the future using the previous kinematic model.

Appendix:

assignment_7.m

```
% Run filter_7 on Target1.txt with q = 0
filter_7 ('Target1.txt', 0);

% Run filter_7 on Target2.txt with q = 0
filter_7 ('Target2.txt', 0);

% Run filter_7 on Target2.txt with q = 0.15
filter_7 ('Target2.txt', 0.15);

% Run filter_7 on Target2.txt with q = 1
filter_7 ('Target2.txt', 1);
```

filter_7.m

```
function filter_7 (data_file, q)
    % initialize filter
    t = 0;
    x = zeros(4,1);
    P = eye(4) * 10^8; % large covariance P
    result = report();

    % open data file
    fid = fopen(data_file);
    while ~feof(fid)
        tline = fgetl(fid);
        data = sscanf(tline, '%f');

        if ~isempty(data)
            [t_new, z, R] = get_observation(data);
            [t, x, P] = predict(t, x, P, t_new, q);
            [x, P, vx] = update(x, P, z, R);
            result = result.add_data(vx);
        end
    end
    % close file
    fclose(fid);

    result = result.update_estimate(t, x, P);

    result.print_final_estimate();
    result.print_prediction(t + 3600, q);
    result.plot();
    result.print_innovations_percentage();
end
```

get_observation.m

```
function [ t_z, z, R ] = get_observation( data )
%GET_OBSERVATION get observation from data
%   data - column vector containing observation data

    t_z = data(1);
    z = data(2:3);
    R = [data(4:5)'; data(5:6)']; % sqrtm(R) not R
    R = R * R;
end
```

predict.m

```
function [ t, x, P ] = predict( t, x, P, t_new, q )
%PREDICT - constant velocity precition for (x, P) at time "t_new"
%   F : constant velocity transformation matrix respected to "delta_t"
%   Q : process noice covariance matrix to ensure that P is conservative

    delta_t = (t_new-t);
    F = [1 0 delta_t 0; 0 1 0 delta_t; 0 0 1 0; 0 0 0 1];
    Q = q * [1 0 0 0; 0 1 0 0; 0 0 0 0; 0 0 0 0] * delta_t; % velocity should not be inflated
    % (Fx, FPF' + Q)
    t = t_new;
```

```
    x = F * x;
    P = (F * P * F') + Q;
end
```

update.m

```
function [ x, P, vx ] = update( x, P, z, R )
%UPDATE - update sensor estimate
%   incorporate information from new observation (z, R)
%   returns
%       x  - updated mean x
%       P  - updated covariance P
%       vx - normalized unit innovations


    % Use dimensionality transformation matrix H
    %   to project down the state dimension
    %   since the observation doesn't come with velocity
    H = eye(2, 4);

    S = (H * P * H') + R;
    W = (P * H') / S;
    P = P - (W * S * W');
    innovation = (z - H * x);
    x = x + W * innovation;

    vx = innovation ./ sqrt(S([1;4]));
end
```

report.m

```
classdef report

    properties
        normalized_unit_innovations;
        running_percentages;
        t;
        x;
        P;
    end

    methods
        function obj = report()
            obj.normalized_unit_innovations = [];
            obj.running_percentages = [];
        end

        function obj = add_data(obj, vx)
        % vx - normalized unit innovation vector
        % vs - innovation size
            obj.normalized_unit_innovations(:, end + 1) = vx;
            obj.running_percentages(:, end + 1) = ...
                sum(obj.normalized_unit_innovations < 0, 2) / length(obj.normalized_unit_innovations);
        end

        function obj = update_estimate(obj, t, x, P)
            obj.t = t;
            obj.x = x;
            obj.P = P;
        end

        function print_prediction(obj, t, q)
            [~, x_new, P_new] = predict(obj.t, obj.x, obj.P, t, q);
            fprintf('x(%f) = \n\n', t);
            fprintf('%14f \n', x_new);
            fprintf('\n');

            fprintf('P(%f) = \n\n', t);
            fprintf('%14.8f %14.8f %14.8f %14.8f \n', P_new);
            fprintf('\n');
        end

        function print_final_estimate(obj)
```

```
            % print final position
            standard_deviation = sqrt(diag(obj.P));
            fprintf('The final position of the target with standard deviation =\n');
            fprintf('\n%14f, %14f\n%14f, %14f\n%14f, %14f\n%14f, %14f\n', ...
                obj.x(1), standard_deviation(1), ...
                obj.x(2), standard_deviation(2), ...
                obj.x(3), standard_deviation(3), ...
                obj.x(4), standard_deviation(4));
            fprintf('\n');

            fprintf('x_final = \n\n');
            fprintf('%14f \n', obj.x);
            fprintf('\n');

            fprintf('P_final = \n\n');
            fprintf('%14.8f %14.8f %14.8f %14.8f \n', obj.P);
            fprintf('\n');
        end

        function plot(obj)
            % plot x innovations
            figure;
            plot(obj.normalized_unit_innovations(1,:));
            ylim([-3, 3]);
            title('Normalized x innovations');

            figure;
            plot(obj.running_percentages(1, :));
            ylim([.3, .7]);
            title('Running percentage of the x innovations that are less than or equal to 0');

            % plot y innovations
            figure;
            plot(obj.normalized_unit_innovations(2,:));
            ylim([-3, 3]);
            title('Normalized y innovations');

            figure;
            plot(obj.running_percentages(2, :));
            ylim([.3, .7]);
            title('Running percentage of the y innovations that are less than or equal to 0');
        end

        function print_innovations_percentage(obj)
            template = 'Percentage of the innovations that are less than 0 = [ %.2f%%, %.2f%% ]\n';
            percent = ...
                sum(obj.normalized_unit_innovations < 0, 2) / length(obj.normalized_unit_innovations);
            fprintf(template, percent * 100);
        end
    end
end
```