

CMP9135M, Computer Vision
Workshop: Kalman Filter for Visual Tracking

Introduction

You should use MATLAB for the following tasks. If you do not remember how to use it, please refer to the “Getting Started” part in the first workshop of Computer Vision. Alternatively, you can implement the following example using your language and API of choice.

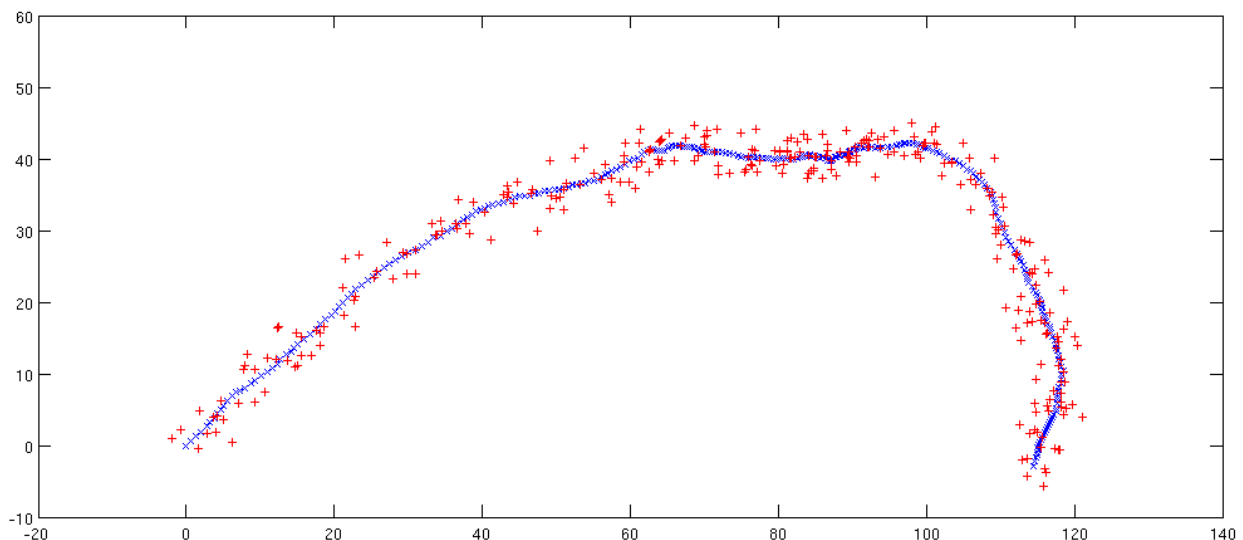
Task 1: Target trajectory

Import all the data files provided with the workshop using the MATLAB command `csvread` (alternatively you can use the import data GUI). The files '`x.csv`' and '`y.csv`' contain the real coordinates (in pixels) of the target trajectory, while '`u.csv`' and '`v.csv`' contain their noisy version. Like in the previous Workshop (Tracking 1), compute the mean and standard deviation of the respective noises ($n_x = u - x$) and ($n_y = v - y$). Plot also their histograms and verify that they are zero-mean Gaussian noises. How would the standard deviations of n_x and n_y be used in a Cartesian observation model representing the above quantities? (i.e. think about the noise matrix R of the Cartesian model seen in the lecture)

NOTE: You can plot the two trajectories, real and noisy one, on the same figure by using the command `hold` after the first plot. For example:

```
plot(x, y, 'xb');  
hold;  
plot(u, v, '+r');
```

You should see something similar to the following graph, where the blue crosses are from the real trajectory, while the red ones are from the noisy observations.



Task 2: Kalman Filter

Copy and paste the following MATLAB function, which implements the prediction step of the Kalman filter, in a file called '`kalmanPredict.m`':

```

function [xp, Pp] = kalmanPredict(x, P, F, Q)
% Prediction step of Kalman filter.
% x: state vector
% P: covariance matrix of x
% F: matrix of motion model
% Q: matrix of motion noise
% Return predicted state vector xp and covariance Pp

xp = F * x;          % predict state
Pp = F * P * F' + Q; % predict state covariance

end

```

The following one instead implements the update step of the Kalman filter, to be saved in a new file 'kalmanUpdate.m':

```

function [xe Pe] = kalmanUpdate(x, P, H, R, z)
% Update step of Kalman filter.
% x: state vector
% P: covariance matrix of x
% H: matrix of observation model
% R: matrix of observation noise
% z: observation vector
% Return estimated state vector xe and covariance Pe

S = H * P * H' + R; % innovation covariance
K = P * H' * inv(S); % Kalman gain
zp = H * x; % predicted observation

%%%%%%%%%% UNCOMMENT FOR VALIDATION GATING %%%%%%%%%%%
%gate = (z - zp)' * inv(S) * (z - zp);
%if gate > 9.21
%    warning('Observation outside validation gate');
%    xe = x;
%    Pe = P;
%    return
%end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xe = x + K * (z - zp); % estimated state
Pe = P - K * S * K'; % estimated covariance

end

```

Finally, the following function implements the whole Kalman filter-based tracker for a target moving according to a Constant Velocity model F and observed by a camera described by a Cartesian observation model H . The input is the observation vector of noisy pixels $z = [u; v]$ given in Task 1. Save this in a new file 'kalmanTracking.m':

```

function [px py] = kalmanTracking(z)
% Track a target with a Kalman filter
% z: observation vector
% Return the estimated state position coordinates (px,py)

```

```

dt = 0.033;      % time interval
N = length(z);   % number of samples

F = [1 dt 0 0; 0 1 0 0; 0 0 1 dt; 0 0 0 1];      % CV motion model
Q = [0.01 0 0 0; 0 1 0 0; 0 0 0.01 0; 0 0 0 1]; % motion noise

H = [1 0 0 0; 0 0 1 0];      % Cartesian observation model
R = [4 0; 0 4];              % observation noise

x = [0 0 0 0]';      % initial state
P = Q;                % initial state covariance

s = zeros(4,N);

for i = 1 : N
    [xp Pp] = kalmanPredict(x, P, F, Q);
    [x P] = kalmanUpdate(xp, Pp, H, R, z(:,i));
    s(:,i) = x; % save current state
end

px = s(1,:); % NOTE: s(2, :) and s(4, :), not considered here,
py = s(3,:); % contain the velocities on x and y respectively

end

```

Run the last function and plot the output trajectory. How close are the new estimated p_x and p_y to the real x and y given in Task 1? How do they compare to the noisy observations u and v ?

Task 3: Evaluation

Besides visual check on the plot, the quality of the tracking can be measured by computing the mean and the standard deviation of the error, defined as follows:

$$e_i = \sqrt{(x_i - px_i)^2 + (y_i - py_i)^2}$$

Also, the Root Mean Squared (RMS) error is a common measure of the tracking accuracy:

$$RMS = \sqrt{\frac{\sum_{i=1}^N e_i^2}{N}}$$

For all these metrics (mean, std and RMS error), the lower the value the better your tracking system is. Verify that the quality of the trajectory estimated with the Kalman filter is indeed better than the one simply observed by a noisy camera (i.e. compare the above errors for u , v and p_x , p_y). Try to modify the model parameters in Task 2 to see if you can achieve even better results.

Task 4: Validation gating

Uncomment the VALIDATION GATING part in `kalmanUpdate` and try the `kalmanTracking` again. Do you obtain the same results? Modify some of the u and v values to be very far from the real target position x and y . Run again the tracker and verify that these new values are indeed excluded from the update process. Try the same for different values of the gating threshold in the `kalmanUpdate` function (current set to 9.21). What does it happen if you set it to 0?