Visual Tracking With No Explicit Detection

A number of hockey players are tracked using the "Estimation from Image Observation" method. The structure of this particle filter is very similar to the MemBer filter.

Contents


```
function TBD_tracker_V01
```

Running custom startup

Here a number of paths are added to the search path. Then the training templates in the form of HSV histograms are loaded into the memory.

```
close all;

disp('running custom startup.m')
toolboxDir = pwd;
fp{1} = toolboxDir;
fp{2} = strcat(toolboxDir, 'external'); % lightspeed, KPMtools
for i=1:length(fp)
```

```
    if(exist(fp{i}))
        addpath(genpathKO(fp{i}));
    end
end
clear fp i;
load HSV_histograms;

disp('done custom startup.m');
```

Initializations

Here some constant parameters in relation to motion dynamics are initialized.

The state is a 4x1 vector including the x and y coordinates of the upper-left corner of the blob and the width (w) and height (h) of the blob.

```
x_dim= 4;   %dimension of state vector
```

Here we set up the state space equation. x= Ax_old + Bv. The settings are equivalent to the following state transition:

$$x(k+1) = x(k) + e_x$$

$$y(k+1) = y(k) + e_y$$

$$w(k+1) = w(k) + e_w$$

$$h(k+1) = h(k) + e_h$$

where the noise term are zero-mean Gaussians with the given standard deviations.

```
model.A=eye(4);
model.sigmax= 10;
model.sigmay= 10;
model.sigmaw= 2;
model.sigmah= 2;
```

Probability of survival is assumed constant.

```
P_S= .9;
```

The constant probabilitiy density for unoccupied pixels in the image:

```
f_unoccupied=0.996;
```

Birth process parameters are set according to assuming 4 Benoulli bith target each with the given number of particles with their states uniformly distributed. The locations of particles areuniformly distributed in each quarter of the image plane, and their widths and heights between 10-18 and 10-28 pixels, respecctively.

```
L_b=4; %no. of Bernoulli birth targets
model.bar_q= zeros(L_b,1);
model.bar_x= cell(L_b,1);
```

Each element of the above cell is a 4x2 matrix. The first row of this matrix includes the min and max of the x coordinates of the particles for the Bernoulli target. The second row includes the min and max for the y coordinates and the last two rows include the min and max of the widths and heights of the particles, respectively.

The first Bernoulli target is in the upper-left quarter

```
model.bar_q(1)=2/10;   %prob of existence for term 1
model.bar_x{1}=[1 150
    1 110
    10 28
    10 18];
```

The second Bernoulli target is in the upper-right quarter

```
model.bar_q(2)=2/10;   %prob of existence for term 2
model.bar_x{2}=[160 310
    1 110
    10 28
    10 18];
```

The third Bernoulli target is in the lower-right quarter

```
model.bar_q(3)=2/10;   %prob of existence for term 3
model.bar_x{3}=[160 310
    120 230
    10 28
    10 18];
```

The fourth Bernoulli target is in the lower-left quarter

```
model.bar_q(4)=2/10;   %prob of existence for term 4
model.bar_x{4}=[1 150
    120 230
    10 28
    10 18];
```

A number of other parameters in relation to limits and merging and pruning are also initialized here.

```
Tmax= 100;          %maximum number of targets
T_threshold= 1e-3;  %threshold to prune targets
Lmax= 400; %maximum number of particles per target
Lmin= 50;  %minimum number of particles per target
```

Data preparation

The "data" subdirectory is added to path, the number of jpeg files (image frames of the video) is counted and recorded.

```
DATA_PATH = sprintf('data/');
imgList = dirKPM(strcat(DATA_PATH, '*.jpg'));
numImages = length(imgList);
INPUT_FILE_FORMAT = 'jpg';
```

The first image frame is read, scaled to 240x320 pixels if required then shown in a figure window.

```
filename = sprintf('%s%s', DATA_PATH, imgList{1});
img = imread(filename, INPUT_FILE_FORMAT);
[imgHeight imgWidth colorDepth] = size(img);

if imgHeight == 240 && imgWidth == 320
    cur_img = img;
else
    cur_img = imresize(img, [240 320]);
end

figure('Name', 'My Trial');
scrsz = get(0,'ScreenSize');
set(gcf,'DoubleBuffer','on');
figposmatrix=[scrsz(3)*0.05 scrsz(4)*0.35 scrsz(3)*0.9 scrsz(4)*0.6];
set(gcf, 'Position',figposmatrix);
uicontrol('String', 'Pause', 'Callback', 'set(gcf, ''UserData'', 1)', ...
```

```
    'Position', [70 10 50 20]);
uicontrol('String', 'Close', 'Callback', 'set(gcf, ''UserData'', 999)', ...
    'Position', [20 10 50 20]);
set(gcf, 'UserData', 0);
```

Actual filtering

```
time=0;
while time<numImages,
```

Initial preparation

The current frame is fed in and scaled if required

```
    clear q_predict N_predict w_predict X_predict;
    hat_X=[];
    time=time+1; pause(0.01);
    filename = sprintf('%s%s', DATA_PATH, imgList{time});
    img = imread(filename, INPUT_FILE_FORMAT);
    [imgHeight imgWidth colorDepth] = size(img);

    if imgHeight == 240 && imgWidth == 320
        cur_img = img;
    else
        cur_img = imresize(img, [240 320]);
    end
```

Particle generation and prediction

At the time=1, only birth particles are gnerated. The `Constraned` function checks for the out-of-scene particles and the particles which are partially within the scene (partial particles), then removes all the out-of-scene and the partial particles which have a small part of them within the scene. Thus, the actual number of particles is decreased. That is why after calling this function, `N_predict` needs to be calculated again.

```
    if time==1,

        L_predict= L_b;
        q_predict= model.bar_q;
        for j=1:L_b
            N_predict_unconstrained= max(round(model.bar_q(j)*Lmax),Lmin);
            [X_predict{j},unused]= Constrained(gen_birthstate_density(model,j,N_predict_unco
            N_predict(j)= size(X_predict{j},2);
            w_predict{j}= ones(N_predict(j),1)/N_predict(j);
        end
```

Otherwise the existing particles for each existing target are propagated, and a birth targets are also generated in a fashion similar to above.

```
else
    L_predict= L_b+ L_update;
    q_predict= model.bar_q;             %birth
    for j=1:L_b
        N_predict_unconstrained= max(round(model.bar_q(j)*Lmax),Lmin);
        [X_predict{j},unused]= Constrained(gen_birthstate_density(model,j,N_predict_unco
        N_predict(j)= size(X_predict{j},2);
        w_predict{j}= ones(N_predict(j),1)/N_predict(j);
    end
    count= L_b;

    for j=1:L_update                    %existing
        count= count+ 1;
        q_predict(count)= q_update(j)* P_S;
        [X_predict{count},indices]= Constrained(gen_newstate(model,X_update{j}));
        N_predict(count)= size(X_predict{count},2);
        w_predict{count}= w_update{j}(indices);
        w_predict{count}=w_predict{count}/sum(w_predict{count});
    end;
end
```

Weight update

The particles weights for the predicted Bernoulli targets are updated according to the fromulae derived in the Fusion paper.

```
clear q_update N_update w_update X_update;
L_update= L_predict;
X_update= X_predict;
N_update= N_predict;
for j=1:L_predict
    g_fun =compute_likelihood(f_unoccupied,upper_hists,lower_hists,X_predict{j},cur_img)
    w_temp= w_predict{j}.*g_fun;
    sum_w_temp = sum(w_temp)
    q_update(j)= min(1/(1 +  abs(1-q_predict(j))/(q_predict(j)*sum_w_temp)),0.999)
    w_update{j}= w_predict{j}.*g_fun/(sum_w_temp);
end;
```

Resampling

The particles for each updated Bernoulli target are resampled so their numbers are limited and their weights are all the same.

```
for j=1:L_update
    N_update(j)= max(round(q_update(j)*Lmax),Lmin);
    idxs= resample(w_update{j},N_update(j));
    w_update{j}= ones(N_update(j),1)/N_update(j);
    X_update{j}= X_update{j}(:,idxs);
end;
```

Pruning

All the Bernoulli targets whose probabilities of existence are smaller than a threshold are removed.

```
idx= find( q_update > T_threshold );
N_update2= N_update;
q_update2= q_update;
w_update2= w_update;
X_update2= X_update;
N_update= zeros(length(idx),1);
q_update= zeros(length(idx),1);
w_update= cell(length(idx),1);
X_update= cell(length(idx),1);
w_update= cell(length(idx),1);
for i=1:length(idx)
    N_update(i)= N_update2(idx(i));
    q_update(i)= q_update2(idx(i));
    w_update{i}= w_update2{idx(i)};
    w_update{i}=w_update{i}/sum(w_update{i});
    X_update{i}= X_update2{idx(i)};
end;
L_update= length(idx);
```

Merging

The centers of Bernoulli targets are computed by weighted averaging the particle states. Those targets which are too close to each other are then merged. Note that the targets which are less than 11 pixels away from each other (horizontally or vertically) are merged.

```
[L_update,N_update,q_update,X_update,w_update]= merge_targets(q_update,X_update,w_update
```

Truncation

If after pruning and merging, the number of updated Bernoulli targets is still larger than the threshold `Tmax`, only the first `Tmax` targets with the largest probabilities of existence are kept and the rest are truncated.

```
if L_update > Tmax,
    [notused,idx]= sort(-q_update);
    N_update2= N_update;
    q_update2= q_update;
    w_update2= w_update;
    X_update2= X_update;
    N_update= zeros(Tmax,1);
    q_update= zeros(Tmax,1);
    w_update= cell(Tmax,1);
    X_update= cell(Tmax,1);
    for i=1:Tmax
        N_update(i)= N_update2(idx(i));
        q_update(i)= q_update2(idx(i));
        w_update{i}= w_update2{idx(i)};
        X_update{i}= X_update2{idx(i)};
    end;
    L_update= Tmax;
end;
```

State estimation from particles

The cardinality of the multi-Bernoulli set of targets (the actual number of targets to be estimated for their states) is given by the following MAP estimate.

```
if length(q_update)==0,
    no_of_targets=0;
else
    cdn_update= prod(1-q_update)*abs((poly(q_update./(1-q_update)))');
    [notused,mode]= max(cdn_update);
    no_of_targets= mode-1;
end
```

The state estimates are the center of particle clusters for the first `no_of_targets` Bernoulli targets with the largest probabilities of exitence.

```
[notused,idx]= sort(-q_update);
for j=1:no_of_targets
    targest= sum(X_update{idx(j)}.*repmat(w_update{idx(j)}',[x_dim 1]),2);
    hat_X= [hat_X targest];
end;
```

Showing the results ...

```
imagesc(cur_img); axis image;
title(['\bf\fontsize{14} Tracking Results']);
```

```
    mainTitle = sprintf('[ %d/%d ] \n', time, numImages);
    suptitle(mainTitle);

    NO_PEOPLE=size(hat_X,2);

    if NO_PEOPLE>0,
        for PNO=1:NO_PEOPLE,
            rectangle('Position', ...
                [round(hat_X(1,PNO)) round(hat_X(2,PNO)) ...
                round(hat_X(3,PNO)) round(hat_X(4,PNO))],...
                'EdgeColor','r','LineWidth',2);
        end
    end
```

Checking the buttons if clicked ...

```
    usr_cmd = get(gcf, 'UserData');
    switch usr_cmd
        case 999
            close(gcf);
        case 1
            waitforbuttonpress;
            set(gcf, 'UserData', 0);
    end

end
waitforbuttonpress;
close(gcf);

end
```

Functions

The auxiliary functions called within the main are listed.

Birth state generator

this function generates uniformly distrbuted targets in the image plane note that 320-24=296 and 240-24=216

```
function X= gen_birthstate_density(model,birthid,num_par)

j= birthid;
barX = model.bar_x{j};
xmin=barX(1,1); xmax=barX(1,2);
```

```
ymin=barX(2,1); ymax=barX(2,2);
wmin=barX(3,1); wmax=barX(3,2);
hmin=barX(4,1); hmax=barX(4,2);
X1=round(rand(1,num_par)*(xmax-xmin)+xmin);
X2=round(rand(1,num_par)*(ymax-ymin)+ymin);
X3=round(rand(1,num_par)*(wmax-wmin)+wmin);
X4=round(rand(1,num_par)*(hmax-hmin)+hmin);
X=[X1 ; X2 ; X3 ; X4];
end
```

Particle progation

This new state generation function follows the linear state spc eqn. x= Ax_old + Bv

```
function X= gen_newstate(model,X_old)

if isempty(X_old),
    X= [];
else
    X= model.A*X_old+ diag([model.sigmax model.sigmay model.sigmaw ...
        model.sigmah])*randn(size(model.A,1),size(X_old,2));
end
end
```

Resampling

resampling function resample_idx= resample(w,L) w- the weights with sum(w)= 1 L- no. of samples you want to resample resample_idx- indices for the resampled particles

```
function resample_idx= resample(w,L)

resample_idx= [];
[notused,sort_idx]= sort(-w);   %sort in descending order
rv= rand(L,1);
i= 0; threshold= 0;
while ~isempty(rv),
    i= i+1; threshold= threshold+ w(sort_idx(i));
    rv_len= length(rv);
    idx= find(rv>threshold);
    resample_idx= [ resample_idx; sort_idx(i)*ones(rv_len-length(idx),1) ];
    rv= rv(idx);
end;
end
```

Likelihood

compute the likelihood function g_y(x) in the form of a vector for all particles

```
function g= compute_likelihood(p_g,upper_hists,lower_hists,X,img)

nH=3; nS=3; nV=10;
M= size(X,2);
g=zeros(M,1);
P= ceil(X);  %coordinate associated with X
for i=1:M,
    img1=image_grab(img,[P(1,i) P(2,i) P(3,i) ceil(P(4,i)/2)]);
    v1=hist_vector(img1);
    img2=image_grab(img,[P(1,i) P(2,i)+ceil(P(4,i)/2) P(3,i) ...
        ceil(P(4,i)/2)]);
    v2=hist_vector(img2);
    % The histogram likelihoods
    p1=kernel_likelihood(v1,upper_hists);
    p2=kernel_likelihood(v2,lower_hists);
    % The multinomial pmf values

    g(i)=p1*p2/(p_g^(X(3,i)*(X(4,i))));
end
end
```

HSV Histogram

this function returns the k vector of histogram values for all n's as in equation (1) in Okuma's paper

```
function v=hist_vector(img)

img2=rgb2hsv(img);
nH=3; nS=3; nV=10;
bh_matrix=(img2(:,:,1)*nH);
bs_matrix=(img2(:,:,2)*nS);
bv_matrix=(img2(:,:,3)*nV/256);

b_matrix=bh_matrix+bs_matrix*nH+bv_matrix*nH*nS;
[xdim,ydim]=size(b_matrix);
Reshaped_b=reshape(b_matrix,xdim*ydim,1);
[nbins,xbins]=hist(Reshaped_b,0.5:1:(nH*nS*nV-1.5));
v=nbins';
if sum(v)==0,
    disp('all zero bins!!');
```

```
    pause;
else
    v=v/sum(v);
end
end
```

Blob extraction

```
function img2=image_grab(img1,P)

[XMAX,YMAX,unused]=size(img1);

xmin=max(1,P(1));
xmax=min(YMAX,P(1)+P(3));

ymin=max(1,P(2));
ymax=min(XMAX,P(2)+P(4));

img2=img1(ymin:ymax,xmin:xmax,:);
if prod(size(img2))==0,
    disp('null image');
    disp(P);
    disp([ymin ymax]);
    disp([xmin xmax]);
    pause;
end
end
```

Constraining particles

This function constrains the blobs (particles) to be inside the image region.

```
function [Y,indices]=Constrained(X)

M=size(X,2);
XMAX=240; YMAX=320;
X(3,:)=max(ones(1,M),X(3,:));
X(4,:)=max(ones(1,M),X(4,:));
X(1,:)=max(ones(1,M),X(1,:));
X(2,:)=max(ones(1,M),X(2,:));
X(3,:)=min(YMAX*ones(1,M),X(1,:)+X(3,:))-X(1,:);
X(4,:)=min(XMAX*ones(1,M),X(2,:)+X(4,:))-X(2,:);
indices=(X(3,:)>10) & (X(4,:)>10);
Y=X(:,indices);
end
```

Distance

This function computes the squared Bhattacharyya similarity measure between two histogram vectors. V1 and V2 are matrices with the same size and Y is a row function with Y(i) denoting the similarity between the i-th columns of V1 and V2, i.e. V1(:,i) and V2(:,i).

```
%Note: Y includes the SQUARES of the similarity measures

function Y=Hist_Dist(V1,V2)

Y=(1-sum((V1.*V2).^0.5));
%Y=(sum(V1-V2)).^2;
end
```

Kernel estimation

This function computes the kernel density estimate for a vector v1 given a matrix of vectors of the same size, all recorded in V2.

```
function p=kernel_likelihood(v1,V2)

[m,n]=size(V2);
Sigma=0.5;
V1=repmat(v1,1,n);
Y=Hist_Dist(V1,V2);
p=sum(exp(-Y/2/Sigma^2))/(sqrt(2*pi)*Sigma*n);
end
```

Merging function

This function computes the weighted average of the particles for each Bernoulli target and extract the locations of the targets. Then, each pair of targets which are closer than a given threshold are merged into one.

```
function [L_new,N_new,q_new,X_new,w_new] = merge_targets(q,X,w,dthresh,qthresh)

x_dim= size(X{1},1);
L= length(q);
stillmerge= 1;
while stillmerge
    Xmean= zeros(x_dim,L);
    for i=1:L
        Xmean(:,i)= sum(repmat(w{i}',[x_dim 1]).*X{i},2);
    end
```

```
distmatrixm= 10*ones(L,L);
distmatrixp= 10*ones(L,L);
pextmatrix= 10*ones(L,L);
for i=1:L
    for j=i+1:L
```

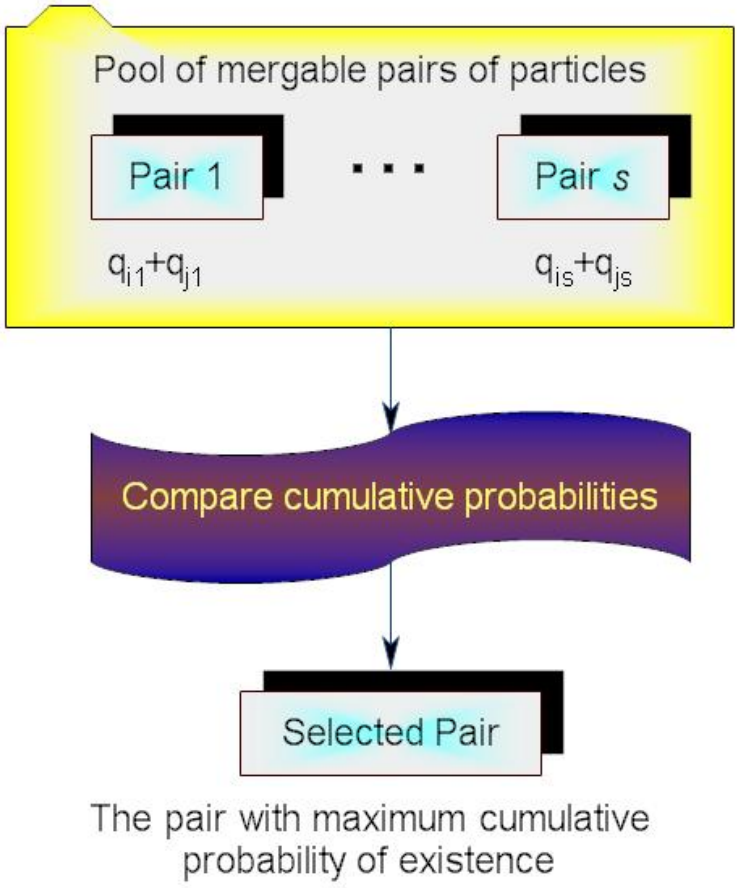
You may chose to use either Eucleadian distance or infinte norm to compare and find close targets.

```
            distmatrixm(i,j)= norm(Xmean([1 2],i)-Xmean([1 2],j),inf);
            %distmatrixm(i,j)= norm(Xmean(:,i)-Xmean(:,j),2);
            pextmatrix(i,j)= q(i)+q(j);

        end
    end
    cm= distmatrixm <= dthresh;
    cq= pextmatrix <= qthresh;
    cj= cm.*cq;
    if any(any(cj))

        ctemp=cj.*pextmatrix;
```

One pair of particles which are too close and need to be merged is chosen here. The pair chosen is the one with maximum cumulative probability of existence.




```
[n,m]= find(ctemp/max(max(ctemp))==1,1);
```

The merged target has all the particles. Their weights are scaled according to the probabilities of existence, so as to sum to 1. The merged probability of existence is the sum of the probabilities for the two targets, thresholded at 0.999.

```
X{n}= [X{n} X{m}];
w{n}= [q(n)*w{n}; q(m)*w{m}]/(q(n)+q(m));
q(n)= min(q(n)+q(m),0.999);
q(m)= [];
X(m)= [];
w(m)= [];
disp(['Two out of ' num2str(L)  ' targets were merged']);
L= L-1;
```

```
        if L==1, stillmerge = 0; end
```

It is important to note that only two targets have been merged so far, and the computation of mutual distances should be repeated then the above process to be looped for the next best pair of targets to be merged.

```
    else
        stillmerge = 0;
    end
end
q_new= q;
X_new= X;
w_new= w;
L_new= L;
for j=1:L
    N_new(j)= length(w_new{j});
end
end
```