Exercise 8 - Condensation Tracker

Computer Vision

Alberto Montes (malberto@student.ethz.ch) November 28, 2016

Implementation

The task consist into implement the color histogram computation. The function needs to take into account the bounding box that overlap with the frame, compute the histogram, and then normalize it to sum up 1. The implementation is on Listing 1.

```
function [ hist ] = color_histogram( xMin, yMin, xMax, yMax, frame,
    hist_bin )
%COLOR_HISTOGRAM Perform a color histogram over the image given.
%   The image is given as a frame and the coordinates (x, y) of the left
%   bottom corner of the bounding box as well as its width and height

yMax = min(round(yMax), size(frame, 1));
yMin = max(round(yMin), 1);
xMax = min(round(xMax), size(frame, 2));
xMin = max(round(xMin), 1);

img = frame(yMin:yMax,xMin:xMax,:);
hist = zeros(hist_bin, size(frame, 3));
for i = 1:size(frame, 3)
    [hist(:,i), ~] = histcounts(img(:,:,i), hist_bin, 'BinLimits', [0,
    255],...
        'Normalization', 'probability');
end
hist = reshape(hist, [1, hist_bin * size(frame, 3)]);
```

Listing 1: color_histogram.m

Then to compute the propagation of particles, there are two ways of model it: no movement and constant velocity. For the first one, the particles are represented as x, y coordinates and the matrix A is the following:

$$\begin{bmatrix} x \\ y \end{bmatrix}_t = A \cdot \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + w_{t-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} \tag{1}$$

In the case the model take into account constant movement, the particles are represented as coordinates and velocities for each coordinates, and the model matrix A is:

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_t = A \cdot \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t-1} + w_{t-1} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}_{t-1} \tag{2}$$

This computation implemented on the propagate function is on Listing 2 where the propagation is computed and then a random noise is added.

```
function [ particles ] = propagate( particles, sizeFrame, params )
%PROPAGATE Summary of this function goes here
%   Detailed explanation goes here

nParticles = size(particles, 1);
```

```
if params.model == 0
    A = eye(2);
    noise = normrnd(zeros(nParticles, 2), params.sigma_position);
elseif params.model == 1
    A = eye(4);
    A(3) = 1; A(8) = 1;
    noise = [normrnd(zeros(nParticles, 2), params.sigma_position),...
            normrnd(zeros(nParticles, 2), params.sigma_velocity)];
end

particles = particles * A + noise;

% Correct the particles going out of the image frame
particles = max(particles, 0);
particles(:,1) = min(particles(:,1), sizeFrame(2));
particles(:,2) = min(particles(:,2), sizeFrame(1));
```

Listing 2: propagate.m

Then, the particles needs to be observed, where for each particle, the surrounding bounding box is extracted and its color histogram is computed. Then for each particle, a weight is computed following a Gaussian and the chi squared distance of the original histogram and the one computed for each particle. The implementation is on Listing 3.

```
function [ particles_w ] = observe( particles, frame, H, W, hist_bin,...
    hist_target, sigma_observe )
%OBSERVE Tihs function make observations
%   Compute for all particles its color histogram describing the
   bounding
%   box defined by the center of the particle and W and H.

numParticles = size(particles, 1);
particles_w = zeros(numParticles, 1);
for i = 1:numParticles
    xy = particles(i, 1:2);
    xMin = xy(1) - W/2;
    xMax = xy(1) + W/2;
    yMin = xy(2) - H/2;
    yMax = xy(2) + H/2;

    max(frame);
    min(frame);
    hist = color_histogram(xMin, yMin, xMax, yMax, frame, hist_bin);

    particles_w(i) = 1 / (sqrt(2*pi) * sigma_observe) * ...
        exp( -(chi2_cost(hist, hist_target)) / (2 * sigma_observe^2));

end

% Normalize particle weights
particles_w = particles_w / sum(particles_w);
```

Listing 3: observe.m

With all the particle and its respective weights, it must be estimated the position of the object to track computing a weighted mean along all the particles coordinates and velocities to obtain an estimation for each time step.

```
function [ meanState ] = estimate( particles, particles_w )
%ESTIMATE Estimate the mean state of the given particles and their
   weights.
```

```
%   Detailed explanation goes here
meanState = sum(particles .* repmat(particles_w, 1, size(particles, 2)),
    1);
```

Listing 4: estimate.m

Finally a resample is performed to keep the particles with higher weight and remove the ones less probable. The algorithm of resampling is the same used on Exercise 3 about the robot localization using particle filters too. The implementation of the resampling function is on Listing 5

```
function [ particles, particles_w ] = resample( particles, particles_w )
%RESAMPLE Resample the particles based on their weights
%   Return these new particles along with their corresponding weights
nParticles = size(particles, 1);
index = floor(random('unif', 0, 1) * nParticles) + 1;
beta = 0;
mw = max(particles_w);
updatedParticles = zeros(nParticles, size(particles, 2));
updatedParticles_w = zeros(nParticles, 1);
for i = 1:nParticles
    beta = beta + random('unif', 0, 1) * 2 * mw;
    while beta > particles_w(index)
        beta = beta - particles_w(index);
        index = mod(index, nParticles) + 1;
    end
    updatedParticles(i,:) = particles(index,:);
    updatedParticles_w(i) = particles_w(index);
end

particles = updatedParticles;
particles_w = updatedParticles_w;
particles_w = particles_w / (sum(particles_w) + eps);
```

Listing 5: resample.m

Experiments

Once the implementation is finished, is time to experiment with different videos, scenarios and parameters to see and understand the behavior of the Condensation Tracker.

video1.wmv

With the first video, where the background is uniform and there is a moving hand, it can be seen how the condensation tracker is capable of tracking the hand until this disappear (Figure 1).

The tracker is capable to follow the hand but with some limitations on the trajectory predictions. As it is initialize with only some fingers on the view, then the observations tent to give a high score at any part of the hand or arm so the trajectory predicted is not smooth. Also remark that when the hand completely disappear the frame, the position prediction tent to go to the bottom left corner, where the histogram is more similar to the initial one.

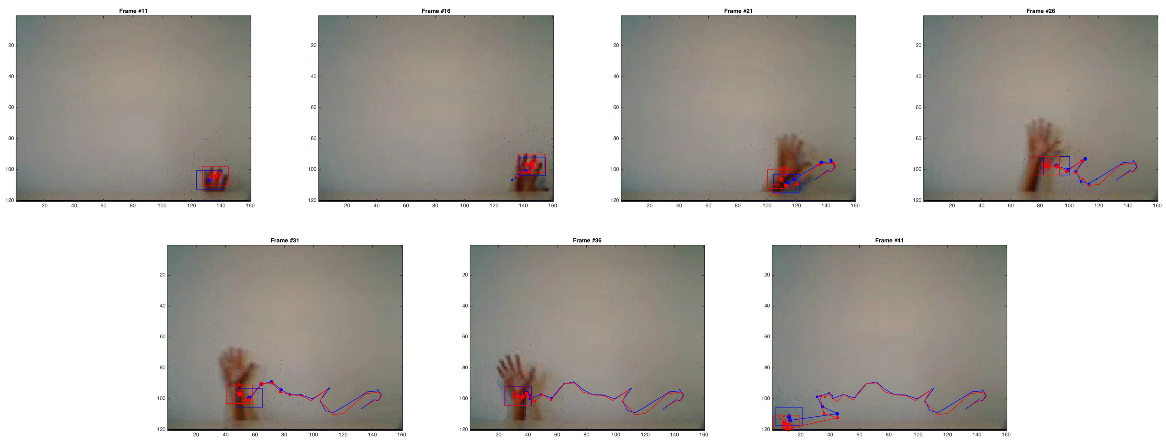
Figure 1: Tracking of the hand at the first video using the implemented Condensation Tracker

`video2.wmv`

First, with the default parameters, on Figure 1 there is the tracking for `vide2.wmv`. It can be seen how the tracker has been able to follow the object even whit an occlusion.

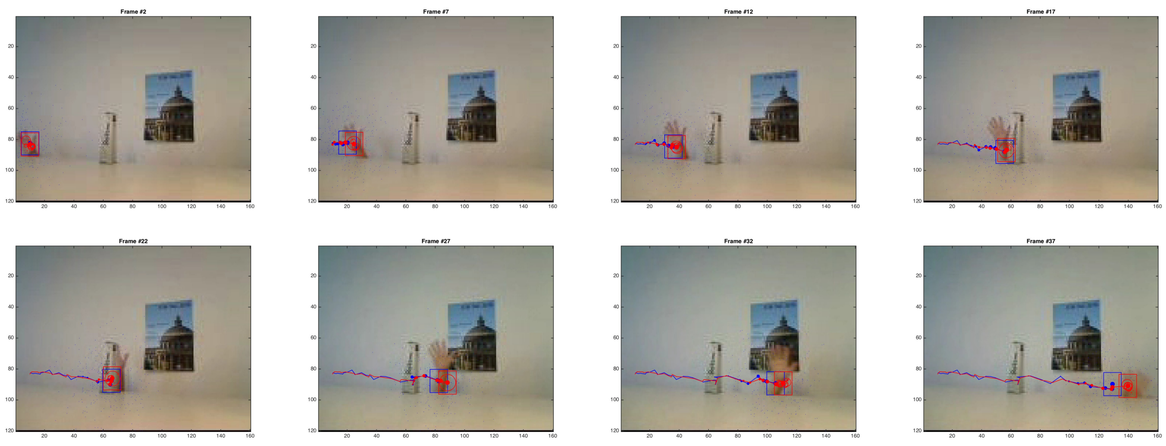
Figure 2: Tracking of the hand at the second video using the implemented Condensation Tracker

Effect of the model

In order to study the performance of the tracker depending on its parameters, first, the constant velocity model is tested. The tracking is on Figure 3 where it can be seen how, as the initial velocity was set wit $v_0 = [1, 10]^T$, the prior probability distribution tent to predict in a lower position in the vertical axis. But thanks to the observations, and the enough dispersion of the particles, the posterior probability perfectly tracks the object even when this is occluded.

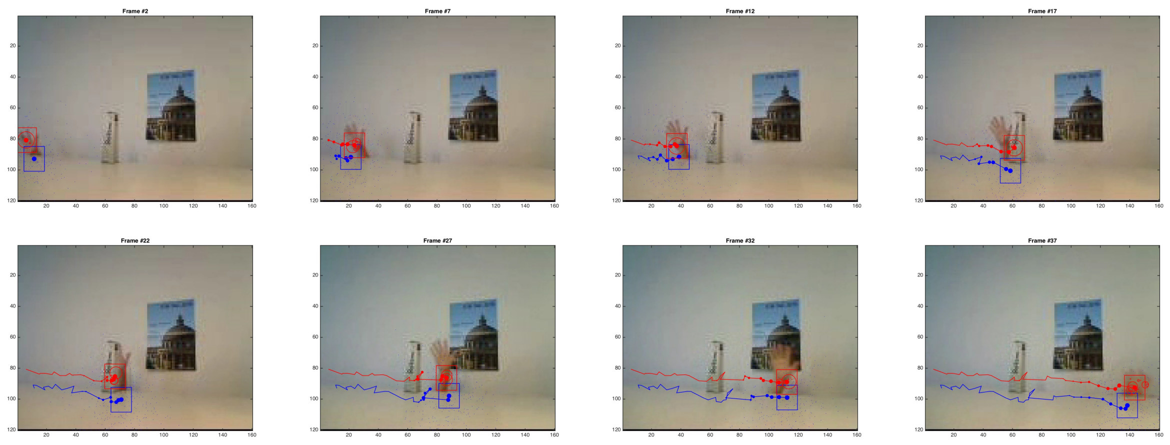
Figure 3: Tracking for the second video with the constant velocity model.

Effect of the system noise

Effect of the measurement noise

`video3.wmv`

Effect of the model

Effect of the system noise

Effect of the measurement noise