Visual Tracking Lab 2

Master IN COMPUTER VISION AND ROBOTICS

Centre Universitaire Condorcet - UB, Le Creusot

08 November 2017

Submitted to: Prof Desire Sidibe

Submission by: Mohit Kumar Ahuja

1. Introduction

The goal of this Visual Tracking module is to learn about, and, more importantly, to learn how to use basic tracking algorithms and evaluate their performances. After Background Subtraction, we will now use a very effective technique for real-time tracking called Mean-Shift. Mean-Shift is a deterministic algorithm, as opposed to probabilistic ones, which solves the data association problem (matching) in a very effective way.

1.1 Mean-Shift

Mean-Shift (MS) Mean-Shift (MS) is widely known as one of the most basic yet powerful tracking algorithms. Mean- Shift considers feature space as an empirical probability density function (pdf). If the input is a set of points then MS considers them as sampled from the underlying pdf. If dense regions (or clusters) are present in the feature space, then they correspond to the local maxima of the pdf. For each data point, MS associates it with the nearby peak of the pdf

As an example, you can see the car sequence in file "Mean_Shift_Tracking.m". We want to track the car in this sequence. We first needed to define the initial patch of the car in the first frame of the sequence. And then the moving car patch will be estimated by using the Bhattacharya coefficient and the weights corresponding to the neighboring patches. It will be deeply explained in the report.

{Note: This report is the explanation of the code submitted for Lab-2 of Visual Tracking}

Tracker initialization

In this method, we need to initialize the tracker with the position of the target in the first frame. For this task, we can:

- Use a detection method
- Initialize the target manually

I used a manual tracker by defining a patch using the imcrop function in Matlab. As shown down and in Fig(1):

```
% For manual initialization use the function imcrop
[patch, rect] = imcrop(ImSeq(:,:,1)./255);
```

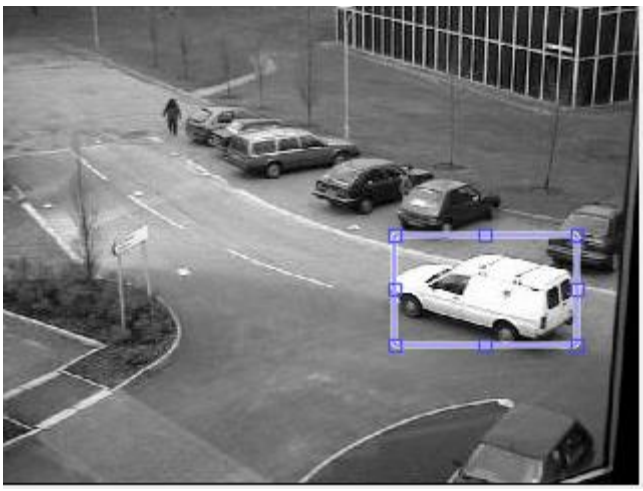

Figure1: Initial Tracker

We can also use automatic tracking using background subtraction and it will automatically detect the car from one frame to another.

The algorithm in detail

1. **Object model**

   The first step of the algorithm is to represent the appearance of the object as a color distribution. We will describe the object's color distribution by a discrete m-bins histogram (try m = 8, m = 16), and we will call this distribution the object's color model q. Let $\{x_1, \ldots\ldots\ldots, x_n\}$ be the set of pixels in the region corresponding to the bounding box of the object, and $b(x_i)$ denotes the color bin at point x_i, i.e. b() is a function which assigns to each pixel one of the m bins of the histogram. A normal histogram h would be defined as:

$$h(u) = C \sum_{i=1}^{n} \Phi(b(x_i) - u), \text{ for } u = 1, \ldots\ldots, m;$$

where C is a normalizer (so that the value in the histogram sum to one) and Φ (.) is the function:

$$\Phi(a) = \begin{cases} 1 & \text{if } a = 0 \\ 0 & 0 \text{ otherwise} \end{cases}$$

But this representation does not take into account the spatial arrangement of pixels that is two different images can have the exact same histogram. Therefore, we introduce in the histogram representation a kernel that weights the contribution of pixels according to their distance from the region's center. So our object's model is a probability distribution q given by:

$$q(u) = C \sum_{i=1}^{n} k(kx_i k_2) k(\|x_i\|^2) \Phi(b(x_i) - u)$$

So, it is being implemented from the Matlab:

```
% Compute distances
for i = 1:size(imagePatch,1)
    for j = 1:size(imagePatch,2)
        dist(i,j) = sqrt((i-c(1))^2 + (j-c(2))^2);
    end
end
```

where $\|x_i\|^2$ is the distance from pixel x_i to the region center (the distances are normalized):

```
dist = dist./max(max(dist));
```

And we use the Epanechnikov profile which is being implemented in Matlab as:

$$k_E(x) = \begin{cases} \frac{1}{2} C_d^{-1} (d+2)(1-x) & \text{if } x < 1 \\ 0 & \text{otherwise} \end{cases}$$

```
% Epachnikov profile
if(dist(i,j)^2 < 1)
    kE = 2/pi*(1-dist(i,j)^2);
else
    kE = 0;
end
```

2. Color density matching

The second step of the algorithm is to find the objet in a new frame of a sequence. To do this, we have to look around the previous position of the object, and each such potential position defined a target candidate. Each candidate is also represented by a color distribution p, and we evaluate the

similarity between the model and candidate distributions, using the Bhattacharyya coefficient ρ:

$$\rho\,[p, q] = \sum_{u=1}^{m} \sqrt{p(u)q(u)}$$

So, we made a function in Matlab named "`compute_bhattacharyya_coefficient`":

```
k = sum(sqrt(p.*q));     % Bhattacharya Coefficient
```

3. Tracking algorithm

We followed the steps to implement the algorithm are:

1. Initialization of the target in first frame as shown in Fig(1).
2. Compute distances, Epanechnikov profile and Bhattacharyya coefficient.
3. Then we computed the new location z in function "`compute_meanshift_vector`" which is given as:

$$z = \frac{\sum_{i=1}^{n_h} y_i w_i g\left(\left\|\frac{y-y_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{y-y_i}{h}\right\|^2\right)}$$

And we implemented it in matlab as:

```
% Compute new center of mass
z = [sum(sum(x.*weights)) sum(sum(y.*weights))]./sum(sum(weights));
```

4. Then we computed the weights which is given by:

$$w_i = \sum_{u=1}^{m} \delta(b(y_i) - u)\sqrt{\frac{q_u}{p_u(y)}}$$

And implemented as a function "`compute_weights`" in Matlab:

```
% Compute the ratio vector btw both color distributions
ratio = sqrt(qTarget./pCurrent);
% Find weights over all the patch, for every pixels
for i = 1:size(imPatch,1)
    for j = 1:size(imPatch,2)
        % Check in which bin is the current pixel value
        val = imPatch(i,j);
        bin = ceil(val*Nbins/255);
        if(bin==0)
            bin = 1;
        end
        % Compute the weight
        weights(i,j) = ratio(bin);
    end
end
```

5. While $\rho[p(z), q] < \rho[p(y), q]$, do $z \leftarrow \frac{1}{2}(y + z)$. Which is implemented in Matlab as:

```
% Check if the bhattacharyya coeff is higher with the new center
% If not, take the half between both centers
new_ImPatch = extract_image_patch_center_size(I, z, ROI_Width,
 ROI_Height);

new_ColorModel = color_distribution(new_ImPatch, Nbins);

new_Rho_0 = compute_bhattacharyya_coefficient(TargetModel,
 new_ColorModel);

while(new_Rho_0 < rho_0)

    z = 0.5*(prev_center + z);

    new_ImPatch = extract_image_patch_center_size(I, z,
    ROI_Width, ROI_Height);

    new_ColorModel = color_distribution(new_ImPatch, Nbins);

    new_Rho_0 = compute_bhattacharyya_coefficient(TargetModel,
    new_ColorModel);
end

display(prev_center);
display(z);
new_center =  z;
```

Results:

Figure 2: Initialization of Tracker

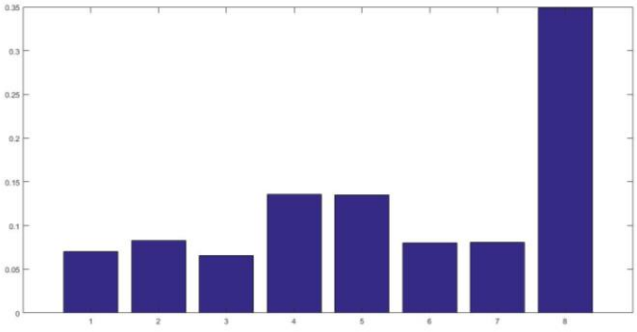

Figure3: Histogram representation

Figure 4: Target tracked in every frame

Figure 5: Target Tracked in Final Frame

Conclusion:

The tracking consists in running for each frame the algorithm describe above. Thus, given the target model, the new location of the target in the current frame is found to minimize the distance between the two distributions p and q in the neighborhood of the previous location.

Questions:

1. **Implement a mean-shift algorithm to track a single object in a sequence. What can be done to deal with the varying size of the object?**

   If the scale of the object starts varying, we can create different windows for different scales. For ex: If we select 5 or 3 windows for different scales of the object, like one is near to previous scale and other is bit larger than the previous scale and the third is the previous scale. So, by doing this the bhattacharyya coefficient for all the 3 or 5 patches will choose the one with the highest score.

2. **How would you modify your code to deal with color images? Apply your tracking algorithm to the given sequences and any sequence of your choice. Comments?**

   If we want to deal with color images, we just need to change our code in "`color_distribution`" and compute weights according to the new patches in "`compute_weights`" function. Now, we have to calculate three different histograms in red, green and blue channels.

3. **What are the limitations of this algorithm? How can we deal with occlusion? Demonstrate it.**

   According to me, Mean-shift algorithm doesn't give better result when illumination changes, also its results are not too good for noisy images and during occlusions in image sequences the target is completely lost.

   To deal with occlusion, we can use sparse representation method in which we use target templates and trivial templates, which is the best solution to use for scale difference as well as to handle occlusions.

{Note: Code is fully commented for better understanding of the reader}