Mean-Shift Tracker

Intelligent Multimedia Systems

University of Amsterdam

2012-2013

Avgerinidis Liveris 10318828

Zervos Ioannis 10333630

Introduction

Object tracking is a challenging task of computer vision. It has applications in many domains such as robot vision, surveillance, driver assistance and face tracking. Real time object tracking in video is actually the representation and the localization of a target object in the following frames given the location of the object in the initial frame. The main difficulties of object tracking are that the target may change its shape through time while moving or its color because of shading or illumination variations, and can even be occluded by other objects.

In our approach, we implemented the mean-shift tracker as described in [1]. We have made two basic assumptions; a) we assume that the color of the object remains stable for the whole sequence of frames, and b) the position of the object in the next frame will be close to its position in the current frame. To have a measure of reference, we extract the *color histogram* of the target object in the initial frame and we augment it with the *epanechnikov kernel* to accentuate the pixels of the center of the object and ignore the pixels of the periphery, which may be part of the background. To measure the similarity of the target with a candidate, we use the *Bhattacharyya distance*. For our tracker, we use the *mean-shift algorithm* to give us the new position of the object in the next frame, given its position in the current frame. Finally to make it partially *scale invariant*, we check in the next frame for an object slightly larger and slightly smaller than the original target window and choose the one that has the smallest distance, to cope with size variations in case the object is approaching or moving away from the camera.

1 Related work

In the literature a lot of different approaches for target representation and localization have been proposed. Tracking methods [2] can roughly be divided into three categories 1) Point tracking [3,4], 2) Kernel tracking [1,5] and 3) Silhouette tracking [6]. In the first approach objects detected in consecutive frames are represented by points and then deterministic and statistical methods are used to match the above points. In the second category, a kernel is used to represent the appearance and shape of the object and the motion of the kernel is calculated to track the object. In the last category, a contour of the object is derived and the tracking consists of shape matching or contour evolution.

2 Description of our approach

2.1 Target representation

It has been shown [1] that the color distribution of an image has discriminative power in object tracking. Based on the above, we use a 3 dimension RGB color space to represent our target model (the object we want to track). We use a 3D color histogram to keep the relation and dependence between the color channels instead of using 3 separate histograms of each color channel.

- target model $\hat{q} = \left\{ \hat{q}_u \right\}_{u=1\ldots m} \qquad \sum_{u=1}^{m} \hat{q}_u = 1$
- candidate model $\hat{p} = \left\{ \hat{p}_u \right\}_{u=1\ldots m} \qquad \sum_{u=1}^{m} \hat{p}_u = 1$

To reduce the computational cost, we quantize the 255 different values of RGB color space to 16 bins. This is a compromise we have to make to balance between efficiency and discriminative power. We also use the normalized color space rgb which is insensitive to shadow and shading. Then we compare the target color histogram with the equivalent of the candidate model.

2.2 Similarity function

In order to measure the correlation between the color histogram of the target and the candidate model we use the Bhattacharyya distance as a similarity measure [1].

$$d(y) = \sqrt{1 - \rho[\hat{p}(y), \hat{q}]}, \quad (1)$$

where we choose

$$\hat{\rho}(y) \equiv \rho[\hat{p}(y), \hat{q}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u(y), \hat{q}_u} \quad (2)$$

the sample estimate of the Bhattacharya coefficient between p and q.

Firstly, we reshape the 3 dimension color histograms into 1 dimension to reduce the computational cost and then we measure their similarity. The higher the value of the coefficient, the more similar the histograms are.

2.3 Kernel smoothing

Any isotropic kernel can be used in order to smooth the target and candidate models. As suggested in [1] we use the epanechnikov kernel which gives more weight to pixels closer to the center. This is useful because pixels far from the center are more likely to be occluded or cluttered. An epanechnikov profile is given below

$$k(x) = \begin{cases} \frac{1}{2} c_d^{-1}(d+2)(1-x), & if \quad x \le 1 \\ 0, & otherwise \end{cases} \qquad (3)$$

and the shape of the kernel is shown in the figure 2.1.

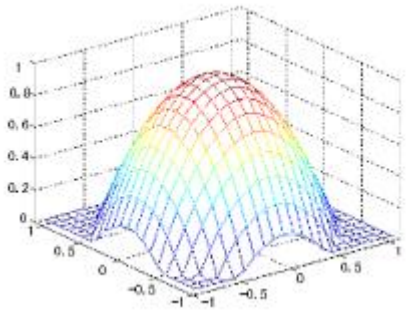

Figure 2.1: An epanechnikov kernel profile

2.4 Target localization

To find the location corresponding to the target in the current frame, the distance d(y) (1) should be minimized, which is equivalent to maximizing the Bhattacharya coefficient ρ(y) (2). We take the gradient and the Taylor expansion [1] and we derive the weights of each pixel bin around the corresponding location

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}} \delta[b(x_i) - u] \qquad (4)$$

where $\hat{y}_0$ is the location of the target in the previous frame and x_i the location of the pixel in the image.

Then the new estimated location is computed by

$$\hat{y}_1 = \frac{\sum_{i=1}^{n_h} x_i w_i}{\sum_{i=1}^{n_h} w_i} \qquad (5)$$

where n_h is the size of the kernel.

It worth's mentioning that we reach the above simple weighted average because the derivative of the epanechnikov kernel is constant. For more details about the above steps the reader could address the [1].

2.5 Mean-shift tracking

The main assumption made is that the tracking object doesn't change drastically in appearance or location between consecutive frames. The main feature that we use to track the object is the color histogram. The intuition behind the mean shift tracker is that, given the two color histograms of the target and the candidate, we shift through the center mass of the selected window until a convergence criterion is satisfied (in our case a similarity measure), as shown in the figure 2.2. Every pixel inside the window is quantized to a bin and each bin has its own corresponding weight. In this way we calculate a gradient vector that points into the spatial location where the two distributions have the largest similarity value (local maxima).

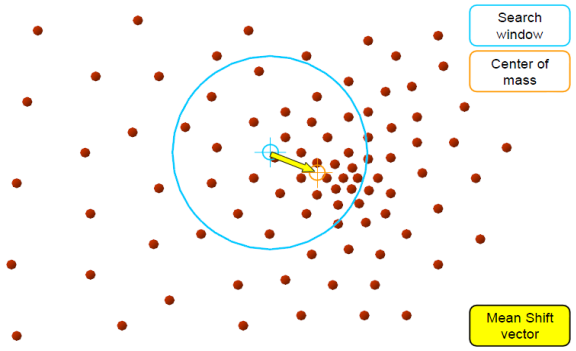

Figure 2.2: Mean shift procedure

Having described the above steps, an informal outline of the mean shift algorithm is presented below [1].

Given the target model $\hat{q}$ and its location $\hat{y}_0$ in the previous frame:

1. Compute the color distribution around the location $\hat{y}_0$ in the current frame and evaluate the Bhattacharya coefficient.

2. Derive the weights of each pixel in the location according to (4).

3. Estimate the new location $\hat{y}_1$ of the object according to (5).

4. Evaluate the similarity between the target $\hat{q}$ and the candidate $\hat{p}$ at location $\hat{y}_1$.

5. Iteratively, we check if the position halfway between the estimated new position and the original position is a better match (Bhattacharyya). If it is, we take this position as new position and repeat.

6. If the step calculated is too big (above distance ε), set $\hat{y}_0 \leftarrow \hat{y}_1$ and go to step 2. Otherwise return the new estimated center of the object.

3 Implementation of the algorithm

Our implementation is mainly based to the steps described above. The main code of our tracker can be found in the file tracker.m.

The implementation begins by asking the user to select an object to be tracked by drawing a bounding box in the initial frame. Then an epanechnikov kernel is created with the corresponding width and height of the drawn box.

Once we create the epanechnikov kernel, we can directly add the kernel weight to each pixel and assign it to the corresponding bin. In this way we reduce the computational cost, as we only have to create once the kernel mask and then we can look up for the corresponding values of the pixels in this kernel matrix.

In the next step we compute the color histogram of the selected bounding box. To avoid division by zero, when we convert the RGB into normalized rgb, we use the matlab built in value eps if the sum of the RGB colors is zero. When we use the normalized rgb color space there is no need to have a 3 dimensions color histogram as the 3rd dimension is dependent on the others two. Hence, we reduce the computational cost by having to calculate only a 2 dimensions histogram.

After constructing the histogram of the region we reshape it into 1 dimension histogram to facilitate further computations. As mentioned in the previous section we quantize the 255 values into 16 bins. Therefore, we construct in the beginning a 16x16x16 3d histogram or a 16x16 if we use the normalized color space and we end up with a 1 dimension histogram of 4096 or 256 bins in total equivalently.

Another thing to mention is that the location returned by the algorithm is in relative coordinates. That means that we have to add the returned coordinates into the previous location of the center to get the absolute location of the object in the current frame.

As suggested in [1], the practical implementation consists of computing the new weights, deriving the new location and checking the size of the kernel shift. Then we only compute the Bhattacharya coefficient as a similarity measure. To make our implementation scale invariant, we also compute the coefficient three times, $h=h_{prev}$, $h=0.9h_{prev}$ and $h=1.1h_{prev}$, and we keep the one that has the largest value. To avoid over-sensitive scale adaptation we use the formula $h_{new}=0.1h_{opt}+0.9h_{prev}$. In this way we give more weight to the previous scale and less to the best one. We also limit the maximum number of iteration in the mean-shift loop to 20 to achieve real time performance.

4 Experiments

For our experiments we use data from the EC Funded CAVIAR project/IST 2001 37540, found at URL: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/. We use the *TwoEnterShop3cor* subset, which contains 1.149 frames of a security camera targeted to a corridor of a mall. During these frames several people/objects are entering, moving and exiting the scene. We pick certain sequences of frames, where the chosen objects are moving until the moment they exit the scene, because our task is object tracking and not object detection. For these

sequences, we compare the performance of our mean-shift tracker and our brute-force tracker, based on the distance from the groundtruth. A video with the output of our experiments can be found here: http://www.youtube.com/watch?v=-oyAiXIXt4Q

4.1 1st experiment

For our first experiment, we pick a sequence of 260 frames. In that sequence, a woman, marked with the id 6, is moving in the crowded corridor and sometimes gets partially occluded.

Figure 4.1: snapshots 1, 75, 192, 250

In figure 4.1 several snapshots are displayed. With red bounding box is the bounding box of the groundtruth, with blue the suggested bounding box of our mean-shift tracker and with green the suggested bounding box of our brute-force tracker.

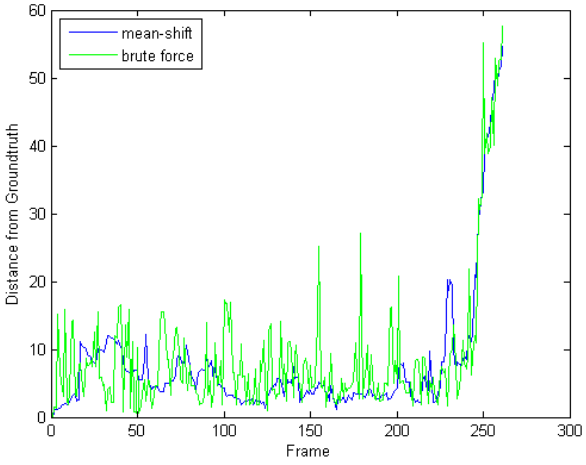

Figure 4.2: Graph of the distance from groundtruth of two approaches

In figure 4.2 we have the graph of the distance of the two suggested centers (mean-shift and brute-force) compared with the groundtruth center. We can see that until the frame 250, both the trackers succeed in tracking the object. The sudden variations in brute force approach can be explained as the brute force searches for the best match in every possible position around the position of the bounding box in the previous frame, whereas mean shift approach gives us only one possible position with high accuracy and thus it has a smoother transition. After frame 250, the object is almost completely occluded and the tracking fails.

For the same experiment, we run the mean-shift tracker and brute-force tracker one more time, but this time we don't normalize the colors. By doing this, our tracker is more sensitive to shadows and intensity variations. Graph in figure 4.3 shows the distance of the center of our bounding boxes from the groundtruth centers.

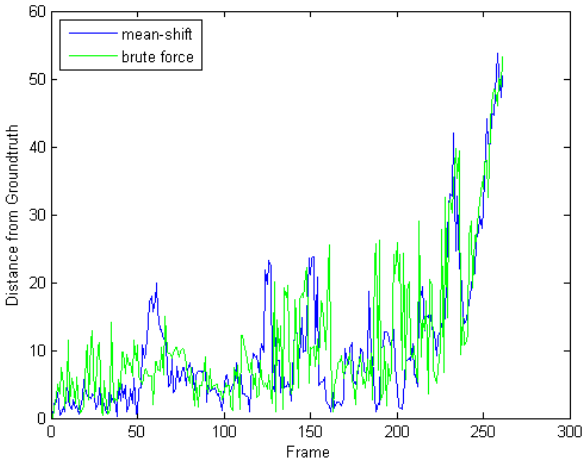

Figure 4.3: Graph of the distance from groundtruth of two approaches, without normalizing the colors

From the graph of the figure 4.3 we can say that the tracker is more sensitive without normalizing the colors of the frames, and so it tends to fail to track the object correctly. Based on this result, we will normalize the colors for the following experiments.

4.2 2nd experiment

For our second experiment we choose a sequence of 110 frames. In that sequence, two men are crossing vertically the corridor. Until frame 80 both our approaches succeed in tracking the object with id 0. After frame 80, that object passes in front of another object with similar colors and both our approaches get confused.

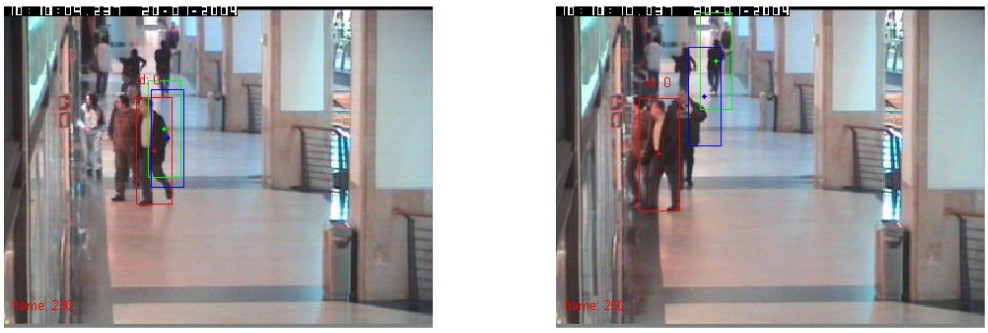

Figure 4.4: snapshots 80, 100

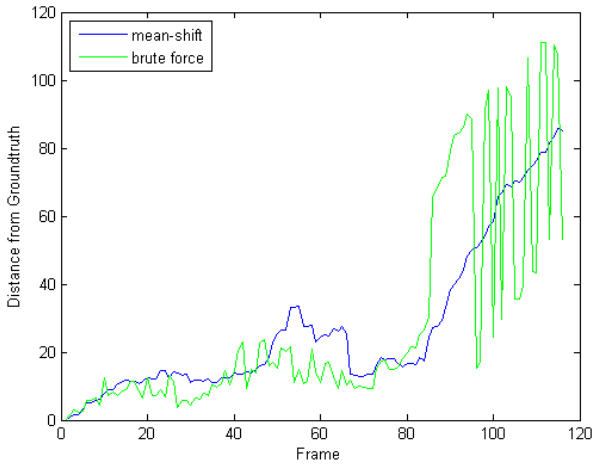

Figure 4.5: Graph of the distance from groundtruth of two approaches

In figure 4.5 we can see the graph of the distance of the center of our approaches from the center of the groundtruth. Although for 80 frames the performance of both approaches is average, after that point, the tracking is affected by the objects of the background and the performance is getting worse.

4.3 More videos

Here we present the output when we apply our tracker to other videos. Given 200 frames of a football match, we run multiple times our tracker, and we try to track different player each time. One advantage of this scenario is that the players have different colors from the

background and that makes the tracking easier. Another positive thing for our tracker is that the objects maintain their width-height ratio and so the bounding box of the initial target succeeds in tracking other instances of the object while moving.

In figure 4.6a, we can see the first frame of the sequence and in 4.6b the target extracted from it. In figure 4.6c we have the 150th frame, where the tracked object is almost completely occluded, but our mean-shift tracker successfully follows it as we can see in the later 195th frame (figure 4.6d).

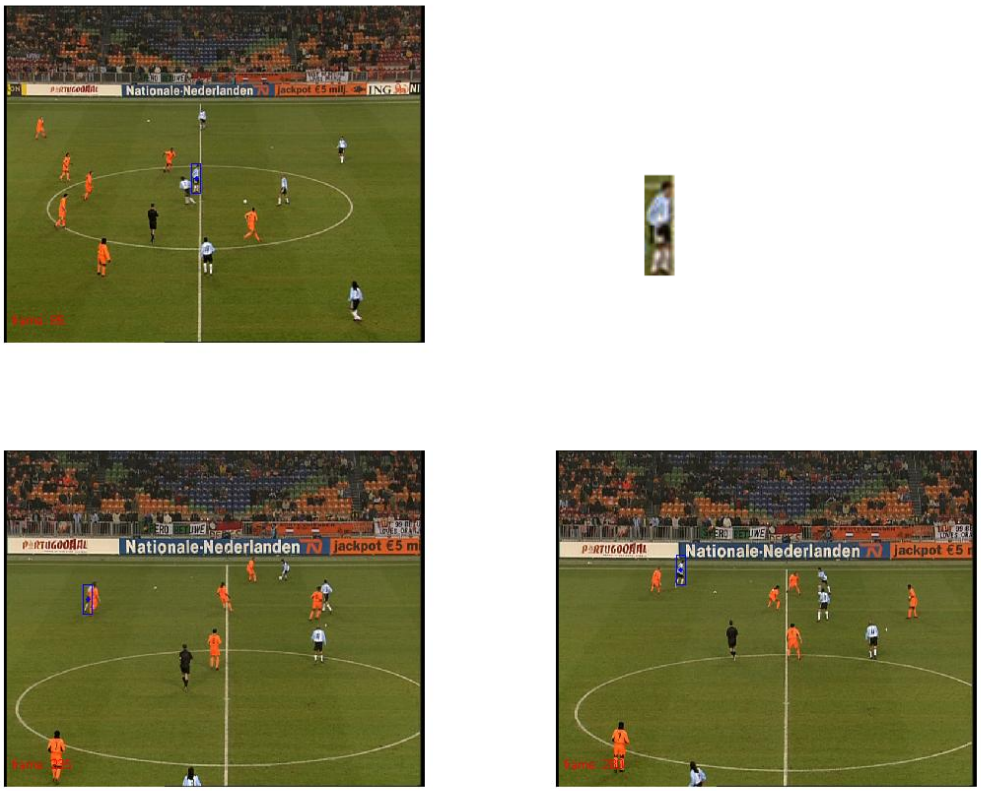

Figure 4.6: 1st frame, target from the 1st frame, 150th frame, 195th frame

5 Discussion

In general, our mean-shift algorithm implementation gives really good tracking results. In most cases outperforms the brute force algorithm which was used as a baseline for our experiments, although it cannot be used for real-time tracking as it is computationally expensive.

As we can see in section 4.3, it can successfully track the football player when he rotates his body and he moves into the opposite direction and even after he is occluded in high degree by another one for a short sequence of frames. However, if the football player is occluded by another one of the same team, then our tracker can't distinguish them because they almost share the same color information.

Another key point of the mean shift algorithm is the update process of the initial target. In our implementation we keep the color histogram and the size of the initial chosen bounding box. In this way, if the object changes drastically in its scale it is difficult to track it. However, for small scale changes we can successfully track it as we have implemented the adaptive scaling as suggested in [1].

As future work, it would be useful to update the target using information of every different frame and giving more weight to the latest one. Another improvement it would be to extract only the foreground where the object is located. This can really enhance the tracking procedure because the background can be misleading leading to tracking failure as we see in section 4.2.

In the ideal case, it would be nice for our algorithm to automatically detect the most suitable color space for the specific image. We would also like to try the opponent color space but in the implementation we should take care of the negative values. In our experiments, we observe that the normalized color space improves the performance compared to the RGB, despite the fact that it becomes more unstable for small RGB values.

6 References

[1] D. Comaniciu , V. Ramesh and P. Meer "Kernel-based object tracking", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp.564 -577 2003.

[2] O. Javed, Journal: ACM Computing Surveys - CSUR , vol. 38, no. 4, pp. 13-es, 2006.

[3] Veenman, C.J., Reinders, M.J.T., Backer, E., *Resolving Motion Correspondence for Densely Moving Points*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 1, pp. 54-72, Januari 2001.

[4] TJ Broida, and R. Chellappa, "Estimation of object motion parameters from noisy images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 1, pp. 90-99, Jan. 1986.

[5] M. J. Black and A. D. Jepson "Eigentracking: Robust matching and tracking of articulated objects using a view-based representation", *Proc. ECCV*, pp.329 -342 1996.

[6] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation, " in Proc. ACM SIGGRAPH 2004, Aug. 2004, pp. 600-608.