ACVM - Mean Shift Tracking

Sebastijan Trojer

I. Introduction

In this project we implemented a simple tracker using the mean shift algorithm for deterministically finding the region with the best similarity to the stored visual model, which in our case was a color histogram. First we implemented the mean shift algorithm to find maxima of a function, and then we implemented the tracker as proposed in [1]. The tracker was evaluated on the VOT2014 dataset samples, which contains videos annotated with bounding boxes.

II. Experiments

A. Mean shift algorithm

We implemented the mean shift algorithm as presented at the lectures, using epanechnikov kernel, which simplified the algorithm, since its derivative is simply an uniform kernel. We evaluated the approach on different inputs and parameter sets, namely we tested how the algorithm performs with different window sizes and starting points. Besides the given input, we also composed a function with several local maxima. The results of both figures are shown in Figure 1.

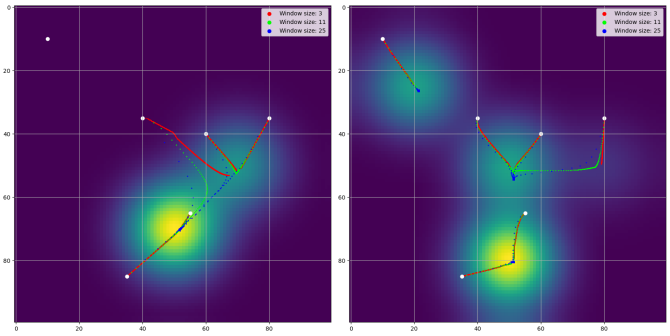

Figure 1. Mean shift on two input functions. The white circles indicate the starting points and colors correspond to different window sizes.

Examining the samples, notice that the starting point seems to have the biggest impact on whether the algorithm will converge - if the starting point is too far from an extrema, it will not move, as it is surrounded by flat level. Similarly, if the starting point is too close to a local extrema, it will converge to that, rather than to a global extrema. Another scenario can be seen in the second image, where the algorithm got stuck at a midpoint between two maximums, where the incline in both directions was the same. The window size also appears to have a big effect on the convergence - a bigger window size allows the method to pick up on more distant maximums and is less likely to converge in a local maximum. A good example of that can be seen on the right side of the second image, where the red path stops at the local maximum on the right, but the green and blue paths are able to find the more remote (better) maximum in the center. We also set a convergence criteria, where we stop the algorithm if the shift is too small. In our case we required a change of minimum 0.0001 pixel. That allowed the algorithm to stop as fast as in 50 iterations, with maximum of 500 allowed. Another consideration about window size is that if the window is too large, the algorithm will never actually converge but jump between two nearby points, while still moving enough to not be stopped, making the process slower or requiring more robust stopping criteria design. A way to address this is to have multiple window sizes defined and use a smaller one, the closer we are to stopping (shift size). Otherwise, the number of iterations was generally larger for the small windows, than for the large ones, which is also reflected by less dots on the images, which makes sense, since a small window can only move in smaller steps.

B. Mean shift tracking

In the second part we implemented a simple mean shift tracking algorithm. Again we used the epanechinkov kernel. The method works by computing the template histogram and searching for the most similar one within a region. This method has a larger set of parameters than the mean shift algorithm, namely convergence criteria, epsilon (to avoid division by 0), bounding box (region) size, number of bins per color in histogram, visual model update parameter (α) and σ for the kernel. We modified the approach a bit by resizing the patch to 50x50 pixels to ensure a consistent size, allowing us to precompute the image offsets. We tested our implementation on several videos and report the performance on the following: ball, fish1, fish2, woman, basketball. The samples were picked because they highlight the scenarios where the tracker has difficulties and where it works well. The number of failures and average tracking FPS are shown in Table I. The parameters used for these results were the following: convergence criteria (minimum shift per iteration in pixels): 1, number of bins per color: 16, model update coefficient: 0.01, kernel sigma: 2.

Video	Failures	Detection FPS
ball	0	806
fish1	7	928
fish2	4	812
woman	1	666
basketball	3	516

Table I
Videos and corresponding number of failures and tracking time.

1) Failure case analysis: Taking a closer look at the scenarios where the tracker failed, we noticed that they all shared a common property - the background colors were similar to the object color and often the object deformed and took less space on the frame, which we do not address here. For example, in the basketball video, the subject's teammates wear the same color t-shirt as the subject, so when the two overlap the tracker shifts to the wrong subject. Another difficult example are the fish videos, where the fish both rotate and move in the environment, which has similar color composition as the fish itself. For comparison, Figure 2 shows 2 frames from the fish2 video. The first video represents a frame where the tracker was unable to consistently track the fish, because it focused on the background instead of the fish. Additionally the fish is also partially occluded in this specific frame, which could make the tracker drift to the other fish if it hadn't failed beforehand. Surprisingly, on the right image, where the fish is barely visible, the tracker works, mainly because the background colors are

more distinct. The tracker also underperformed on videos with rapid target movement, where the object moves substantially between the frames, usually accompanied by illumination and scale change. It was, however, able to deal with partial occlusion present in the woman video.

We also improved the tracker by modeling background distribution. For this, we took a rectangle around the target and modeled the colors, with excluded foreground. That improved the performance in some harder situations, and in our case we got 2 fails less on the fish1 video. However the downside to this is that the average FPS dropped from 660 to 230.

The tracker performed best when the target was of distinct color from the background (e.g. ball video), with semi-consistent movement, scale and illumination.

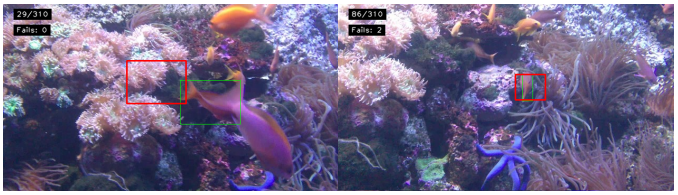

Figure 2. Two frames from the fish2 video. Notice that it is a difficult tracking scenario, due to the colorful background and fish turning away from the camera, occupying less space.

C. Parameters

We also analyzed the performance with different sets of parameters. Specifically we wanted to evaluate the effect of the number of bins per color in the histograms. The histogram bincount determines the amount of colors that we can distinguish, therefore having too few will cause the visual model to pool together more colors, making it harder to distinguish the target from the background where there are similar colors in the background and foreground. However setting the number of bins too high will also have a negative effect, because there will be too many colors, so for example multiple shades of the same color will be put in different bins, making the visual model more sensitive to illumination changes. Therefore it is important to find a balanced value for the parameter. We rerun the experiment with 4, 8, 16 and 32 bins per color and the results are reported in Table II.

Video	4 bins	8 bins	16 bins	32 bins
ball	2	0	0	2
fish1	6	5	7	6
fish2	4	4	4	4
woman	5	2	1	3
basketball	9	5	3	6
Σ	25	20	15	21

Table II
VIDEO FAILURES WITH RESPECT TO HISTOGRAM BINS.

If we look at the results more closely, we notice that the effect of the bins is not always consistent - sometimes a certain amount works better and sometimes worse as some other. Therefore there is no single amount that would be the best for all scenarios, but overall the 16 bin configuration works best. We also tried out few α parameters for model updates and decided to set it to 0.01. If we set it too high, the mean shift tracker would drift away from the target, since it would be updated too much and it would practically track the background, and setting it to 0 keeps the visual model the same throughout the entire video, which can be bad in case the target changes substantially and doesn't resemble the original color composition anymore.

There are several ways the tracker could be improved. For example, instead of fixed window size, we could use a dynamic window size, or as discussed at lectures, each odd iteration do a search in the scale space (e.g. 3 different window sizes) and each even iteration a search in the xy-plane of the image. That would allow us to more accurately determine the position of the maxiumum similarity. Another option would be to replace or combine histograms with more robust features, for example SIFT or HoG. In case we were designing a tracker for a specific application, we could also use our domain knowledge and preprocess the input data to simplify tracking.

III. Conclusion

In this project we implemented the mean-shift mode seek and mean shift tracking algorithms. For each, we considered the effects of different parameters, strengths and weaknesses. Especially for the mean shift tracking algorithm, we provided examples where the approach will inherently have trouble, since, in our case, it relies on color composition of the target, and therefore assumes that the color of the target will differ from the background enough. However, the tracker works very well when the assumption holds and has many practical uses due to its simplicity and speed.

References

[1] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, 2003.