Assignment 2 report - mean shift tracking

Luka Boljević

I. Introduction

Object tracking is an important task in computer vision that has many practical applications, such as surveillance, autonomous driving, and human-computer interaction. The goal of object tracking is to locate and follow a target object in a sequence of images or video frames, despite variations in its appearance and motion. In this assignment, we will implement and explore the strengths and weaknesses of a rather old, but simple and computationally efficient algorithm based on mean shift mode seeking [1]. We will test the mean shift algorithm itself on artificially generated PDFs (probability density functions), and the tracker on sequences from VOT (Visual Object Tracking) 2014 challenge [2].

II. Experiments

A. Mean shift mode seeking

As stated, we will firstly test the mean shift mode seeking algorithm itself, on two artificially generated PDFs, seen in Figure 1 and 2. Both PDFs are of size 100 by 100, and were initially made up of all 0s. After adding 2 i.e. 3 distinct peaks, at positions written in each Figure's captions, the PDFs were smoothed using a Gaussian kernel with $\sigma = 10$.

Figure 1. First artificial PDF. Before smoothing with a Gaussian kernel with $\sigma = 10$, peaks were placed at $x, y = 50, 70$ with value 1, and at $x, y = 70, 50$ with value 0.5.

Figure 2. Second artificial PDF. Before smoothing with a Gaussian kernel with $\sigma = 10$, peaks were placed at $x, y = 70, 40$ with value 1.2, at $x, y = 50, 65$ with value 0.5, and at $x, y = 40, 35$ with value 0.3.

Mean shift algorithm has 3 hyperparameters: starting position, kernel size, and ϵ, used to stop mean shift iterations when the absolute changes in x and y drop below that value. We used Epanechnikov kernel, i.e. its derivative, for all tests on the two PDFs. We tested how varying the hyperparameters affects the convergence speed (number of iterations needed to finish) and convergence location. A small sample of representative results, for both PDFs, are shown in Tables I and II.

.	Conv. speed	Conv. location
$\epsilon = 0.1$	87	52, 65
$\epsilon = 0.05$	132	51, 68
$\epsilon = 0.01$	229	51, 70
3x3 kernel	481	51, 69
5x5 kernel	229	51, 70
7x7 kernel	116	51, 70
Start @ (75, 75)	232	51, 70
Start @ (20, 50)	199	50, 70
Start @ (40, 60)	169	50, 70

Table I
Comparing how different hyperparameter values affect convergence speed and location for PDF 1. Mode is located at $x, y = 50, 70$. Unless explicitly changed, default values are starting position at $x, y = 50, 50$, kernel size = 5x5, $\epsilon = 0.01$. Convergence location and the different stating locations are given as a pair of x, y coordinates.

.	Conv. speed	Conv. location
$\epsilon = 0.1$	0	50, 50
$\epsilon = 0.05$	172	68, 42
$\epsilon = 0.01$	264	70, 41
3x3 kernel	596	69, 42
5x5 kernel	264	70, 41
7x7 kernel	134	70, 41
Start @ (50, 20)	389	70, 40
Start @ (50, 30)	284	70, 40
Start @ (60, 40)	174	70, 40

Table II
Comparing how different hyperparameter values affect convergence speed and location for PDF 2. Mode is located at $x, y = 70, 40$. Unless explicitly changed, default values are starting position at $x, y = 50, 50$, kernel size = 5x5, $\epsilon = 0.01$. Convergence location and the different stating locations are given as a pair of x, y coordinates.

Again, this is only a small sample of possible results. In any case, we can see that, for a fixed kernel size, we stop sooner and generally farther away from the mode, if we use bigger ϵ. While not explicitly shown in the two Tables, a bigger ϵ also reduces the chance of being stuck in a point where the algorithm can't decide where to go, as the kernel size may be too small to accurately perceive its surroundings. This can be somewhat alleviated by using a bigger kernel size, for example 7x7, but still, a bigger ϵ generally means the algorithm may stop before it reaches the mode. We empirically established that $\epsilon = 0.01$ works really well for the 3 tested kernel sizes and various starting positions, on both PDFs.

Of course, convergence speed also depends on the starting position. We converge faster if we start closer to the mode, even with a small kernel. One problem is, if the neighborhood around the starting position is completely flat (all 0s), the algorithm will not be able to determine in which direction to go, as the gradient in all directions is 0. This can again be alleviated by using a bigger kernel size. Speaking about those, we can see from Tables I and II that we converge in less and less iterations as we use a bigger and bigger kernel size. This makes sense, as with a bigger kernel size, the algorithm "sees better", and is able to more easily and accurately calculate in which direction and how far it can/should go.

B. Mean shift tracker

The mean shift tracker has a few hyperparameters: (1) kernel type (we implemented Epanechnikov and Gaussian), (2) σ (a parameter used for both kernels), (3) number of histogram bins m used for representing the target template/candidate, (4) update speed α, (5) ϵ_v (small value used for numerical stability when calculating weights v during mean shift iterations), and (6) ϵ_{stop} (has the same purpose as ϵ in mode seeking). Default values for the tracker are: Epanechnikov kernel with $\sigma = 1$, $m = 16$, $\alpha = 0$, $\epsilon_v = 1e^{-3}$ and $\epsilon_{stop} = 1$. As mentioned in the introduction, we will test the mean shift tracker on a few sequences from VOT2014 challenge. We evaluated the performance in terms on number of failures (meaning it completely lost the target), and mean and median overlap between ground truth and predicted bounding boxes (to try to capture how well the target was tracked). Results are shown in Table III.

Sequence	# failures	Mean overlap	Median overlap
ball	0	0.6758	0.6925
sunshade	0	0.5736	0.5473
surfing	0	0.5822	0.5606
fernando ($m = 32$)	0	0.4247	0.3438
jogging ($\epsilon_v = 1e^{-5}$)	1	0.6325	0.6817
david ($m = 32$, $\alpha = 0.05$, $\epsilon_v = 1e^{-5}$)	1	0.5122	0.5109
basketball ($m = 32$, $\alpha = 0.001$, $\epsilon_v = 1e^{-4}$)	2	0.6462	0.6853

Table III
TRACKING PERFORMANCE ON DIFFERENT VOT2014 SEQUENCES. UNLESS DIFFERENT VALUES ARE WRITTEN BELOW THE SEQUENCE NAME, DEFAULT HYPERHPARAMETERS WERE USED.

One thing to keep in mind when analyzing results from Table III is that the implemented tracker is not scale-adaptive, meaning that size of the target template does not change, unless tracking completely fails and the tracker is reinitialized. In other words, while mean/median overlap values aren't maybe the most representative, they give a somewhat decent idea of how well the target is tracked. Of course, not all sequences work well with default hyperparameters, so we had to tune them slightly to improve performance.

Increasing m to 32 for some sequences in Table III was necessary so the target could be more easily distinguished from the background/surroundings, as 16 colors per channel just wasn't enough. Of course, we could have increased m higher, but that would drastically slow down under-the-hood computations, and in turn the tracking process.

We aren't really sure how to explain why using different ϵ_v helps, as its primary purpose is numerical stability during mean shift. We can only say that increasing it from $1e^{-3}$ to $1e^{-4}$ or $1e^{-5}$ helped tracking in some sequences.

Using $\alpha > 0$ was useful in sequences where the color distribution of the target changes during the sequence - for example, in sequence "david", the target starts walking from a dark towards a brighter part of a room, so updating the histogram representation of the template is a smart idea. We found that in sequences from VOT2014, $\alpha = 0.05$ is the highest value necessary to achieve good tracking. For most sequences where $\alpha > 0$, $\alpha = 0.001$ or $\alpha = 0.005$ were enough. Too high of a value for α would "over-update" the template, and tracking would slowly start completely failing.

Using $\epsilon_{stop} = 1$ was perfectly fine for all our tests. There was no need to modify it, as shifts smaller than 1 pixel aren't really noticeable. On the other hand, stopping mean shift iterations with a large ϵ_{stop} too soon could degrade tracking performance. We also limited the number of mean shift iterations to 20, as it could happen that the shifts in x and y direction never drop below $\epsilon_{stop} = 1$. Generally though, tracking would converge in 1-10 iterations. Lastly, we experimented with using Gaussian kernel, but it didn't bring any noticeable improvements over Epanechnikov, so we just stuck with Epanechnikov.

One downside of this simple tracker was already mentioned - it is not scale adaptive. A proposed improvement from [1] is to search for the target with 3 different template sizes - current, and $\pm$ 10% of current, and then select the one which achieves maximum similarity to the ground truth bounding box.

Another downside is that for some sequences, the tracker struggled to distinguish the target from the background, no matter which hyperparameters were used. This happens because we essentially memorize and track based on the color distribution of the target, and if it is similar to the background color distribution, the tracker can wander off quite easily. One sequence where this happens is "skating". Tracking in these scenarios could be improved if we also keep track of the background color distribution, and then track using only those colors which distinguish the target from the background the most.

The algorithm also struggles when the target moves too fast between frames. Unless we allow for way more than 20 iterations, which would slow the tracker down drastically, the target will simply be too far away for mean shift to catch up, resulting in failure.

III. Conclusion

In this report, we implemented and tested the mean shift mode seeking algorithm, and a tracker based on ideas of mean shift. We showed convergence speed on two artifical PDFs for different hyperparameter values, and discussed how each affects performance. Furthermore, despite its simplicity, the mean shift tracker is efficient, performant, and capable of real-time tracking. Of course, the tracker isn't without its downsides, but even so, it is more than capable of rather successful tracking.

References

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

[2] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin, "A novel performance evaluation methodology for single-target trackers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2137–2155, Nov 2016.