Exercise 2: Mean-Shift tracking

Nikolay Vasilev

I. Introduction

The **Mean-Shift** (MS) algorithm is a popular method in computer vision (CV) for tracking regions (patches) in image sequences. It is non-parametric technique, which is based on estimation of a *probability density function* (PDF) and finding its maximum by iterative shifting a window until it converges to the mode of distribution.

The Mean-Shift tracker [1] uses the MS method to track targets based on their colors. It works by selecting an initial region around the object we would like to track, and iterative shifting this region with the help of the MS method, until we achieve convergence.

In the following paper we will evaluate an implementation of the MS mode seeking and its usage by an MS tracker in Python. We will compute the convergence and computational efficiency of the MS method by using different functions, starting points, kernel sizes/types and termination criterias. The tracker will be evaluated on 5 different sequences from VOT14 with different parameters, which will help us identify the failure cases and improve the tracker.

II. Experiments

We have used Python in order to implement the Mean-Shift algorithm and create a simple tracker.

A. Mean-Shift iterative mode seeking

By running the code in the file *main.py* we can evaluate the Mean-shift algorithm on a simple example. We have implemented a function, which accepts the following six parameters: *a)* function landscape as a NumPy array, which in our case is randomly generated; *b)* x and y coordinates of the center position, from which we start the MS method; *c)* type of the kernel, which help us generate the derivative of the kernel. That means that if we have an Epanechnikov kernel, the MS method will use an Uniform kernel to calculate the mean shift vector, and if we have a Gaussian kernel, Gaussian kernel will be used; *d)* size of the region, which is the same as the size of the kernel; *e)* number of maximum iterations in case there is no convergence; *f)* minimum shifting for convergence.

In Figure 1, we can see the results from a test we have created on two different function landscapes with two different kernel types (Epanechnikov and Gaussian), two different window sizes (7x7 and 15x15) and two different starting (black dot) points (x=30, y=50 and x=55, y=60). After a lot of experimenting with the other attributes, we found out that it would be best if we leave the maximum number of iterations at 200 and the minimum shifting needed for convergence at 0.035. On the first (default) function landscape, we can see that both kernels give pretty similar results (red dots), but we can also see that the bigger window size causes problems. When the window size is 15x15, we can see that both kernels lead to the end point being not really on the maximum value of the function. Of course this may be fixed by increasing the minimum shifting value, which will also increase the speed of the algorithm, but then the smaller window will give us worse values. Anyway, we see that the 7x7 window also works good for the second function landscape - a little bit better for the Epanechnikov kernel. In this case increasing the window size gives us a local maximum, which is not the global one for both kernels. This is totally expected, because the function landscape is not so big, so increasing the window size may make the output of the Mean-Shift method worse. For this two starting positions the window size 7x7 and the Epanechnikov kernel type (Uniform kernel for calculating the Mean-Shift vector), give us the best outputs for both function landscapes.

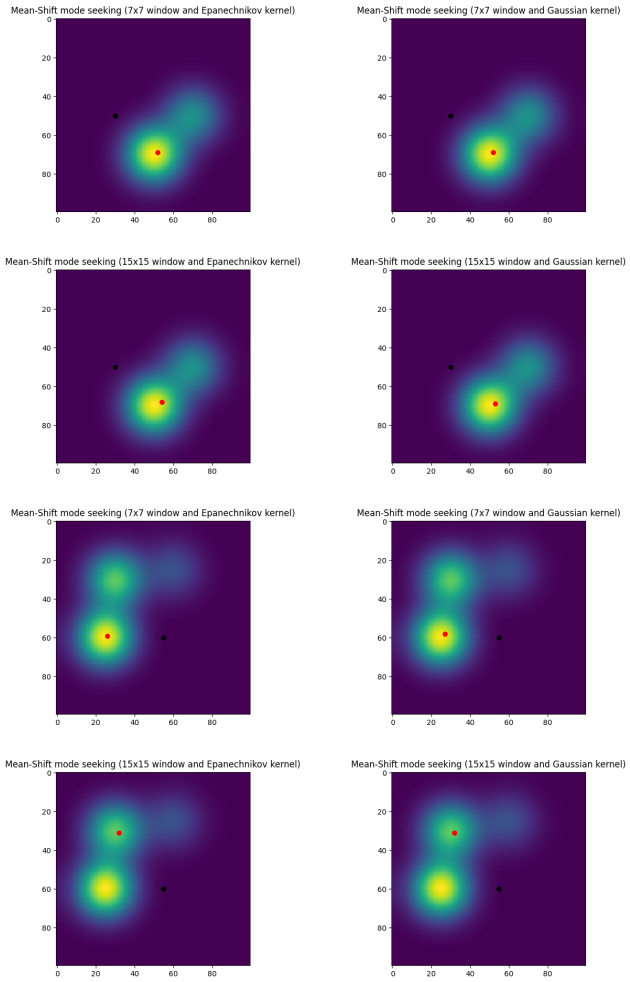

Figure 1. The Mean-Shift iterative mode seeking applied to two different function landscapes with two different kernel types - Epanechnikov (1st column) and Gaussian (2nd column), two different window sizes - 7x7 (1st and 3rd row) and 15x15 (2nd and 4th row), and two different starting points - x=30, y=50 (1st and second row) and x=55, y=60 (3rd and 4th row). The black point is the starting position and the red point is the ending position.

Of course, the computational efficiency is also important, which is why we compared the time needed for all these 8 cases to go through the MS mode seeking. As we can see in Table I, in most cases we can increase the speed of our algorithm by just increasing the window size. Of course the speed may be increased also by increasing the minimum value

Table I
PERFORMANCE TIME OF MS MODE SEEKING CASE

Case	Iterations needed	Speed
Epanechnikov 7x7 (x=30, y=50)	90 / 200	5.2ms.
Epanechnikov 15x15 (x=30, y=50)	40 / 200	2.4ms.
Gaussian 7x7 (x=30, y=50)	107 / 200	6.1ms.
Gaussian 15x15 (x=30, y=50)	28 / 200	1.2ms.
Epanechnikov 7x7 (x=55, y=60)	136 / 200	7.3ms.
Epanechnikov 15x15 (x=55, y=60)	43 / 200	2.6ms.
Gaussian 7x7 (x=55, y=60)	104 / 200	5.8ms.
Gaussian 15x15 (x=55, y=60)	41 / 200	2.7ms.

for convergence, but in both cases the end point calculated by the MS method may not be as good.

B. Mean-Shift tracker

We used the MS mode seeking method in order to create a MS tracker class in Python, which has two main functionalities – initialization and tracking.

1) Initialization: To initialize our tracker, we created a function, which generates a presentation (extracts histogram q) of the target appearance. The function accepts the following four parameters: *a)* the initialization image of a sequence; *b)* chosen region of an object, which will be tracked; *c)* the kernel type, which can be Epanechnikov or Gaussian and is not the same as the one in the MS method; *d)* Number of bins in the histogram.

This functions only extracts the histogram of the target region and sets other variables, which will be needed in the tracking. After testing on multiple sequences, we found out that it is the best to generate histograms with 12 bins.

2) Tracking: This function starts from the estimated target position in the previous frame and uses the MS method to estimates the new position. The function accepts the following six parameters: *a)* the current image from the sequence; *b)* number of maximum iterations in case there is no convergence; *c)* minimum shifting for convergence; *d)* a constant α, which is needed to update the model q; *e)* a constant used for the adaptive scale and tell us by how much we try to increase and decrease the region; *f)* a constant γ, which we use to calculate the new scale.

We can divide that function in the following three main parts:

- **Adaptive scale** – We use the parameter, which is calculated as a multiplication of the scale constant and the previous image target's scale $\nabla s = scale * s_{prev}$. Then we calculate three possible regions with three different scales - one has the previous image target's scale s_{prev}, the other has increased scale $s_{prev} + \nabla s$ and the third has decreased scale $s_{prev} - \nabla s$. We calculate the Bhattacharyya measures for each of the regions and find the one with biggest value. This is also the optimal scale, which with the help of the parameter γ help us calculate the new size of the region the following way:

  $$s_{new} = \gamma s_{opt} + (1 - \gamma) s_{prev}$$

  We found out that the best values here are $\gamma = 0.1$ and $scale = 1e - 5$.
- **Target localization** – This is the step, where we use the MS method. We iterative extract the current histogram p, calculate the weights and the backproject, and compute the Mean-shift vector and the new position. We repeat that until maximum 20 iterations or until we have a shifting smaller than 1.
- Model update – In the end, when we have calculated the new position of the target we find the histogram q_calc of the new position and set it as a template for the next image. To calculate the new value, we again apply moving average similarly as on the Adaptive scale step, but here we have the constant $\alpha = 1e - 8$:

  $$q_{new} = \alpha q_{calc} + (1 - \alpha) q_{prev}$$

  The results from the evaluation of the Mean-Shift tracker on 5 different image sequences from VOT14 is visible in Table II. We have done the evaluation using both kernels – Epanechnikov and Gaussian – as well as the parameters that we already mentioned to fit the tracker best. From the results we can see that the tracker fails mostly for image sequences with collision, such as the sequences "jogging" and "skating". Another interesting thing is that the Gaussian kernel has less fails in those same sequences, while in the others it has worse speed and more fails.

Table II
PERFORMANCE OF MS TRACKER ON 5 DIFFERENT IMAGE SEQUENCES FROM VOT14 USING THE EPANECHNIKOV AND GAUSSIAN KERNELS

Sequence	Kernel type	Speed	Fails
bolt	Epanechnikov	205.8 FPS	3 times
bolt	Gaussian	233.4 FPS	8 times
car	Epanechnikov	254.7 FPS	1 time
car	Gaussian	290.3 FPS	1 time
jogging	Epanechnikov	248.0 FPS	7 times
jogging	Gaussian	240.2 FPS	5 times
skating	Epanechnikov	124.5 FPS	15 times
skating	Gaussian	113.0 FPS	10 times
hand1	Epanechnikov	388.4 FPS	4 times
hand1	Gaussian	366.6 FPS	5 times

III. CONCLUSION

Even though, as we proved, the Mean-Shift is sensitive to collisions and handle them poorly, we can conclude that the usage of the Mean-Shift algorithm in computer vision proves to be highly efficient and effective, if the right parameters are used. Both the MS method and tracker are valuable tools, which may be easily implemented and used in wide range of applications.

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.