# Mean-Shift tracking

Urša Zrimšek

## I. Introduction

In this exercise we implemented a simple tracker that utilizes Mean-shift algorithm for finding the region with the best similarity between the region and a stored visual model of the target object. The Mean-shift provides a robust deterministic way of finding a mode of a probability density function

## II. Experiments

### A. Mode seeking

We first tested our implementation of mean-shift algorithm on the functions, shown on Figure 1. On them we ran the algorithm using different parameters (kernel size, starting points, stopping criteria), and compared the performance.

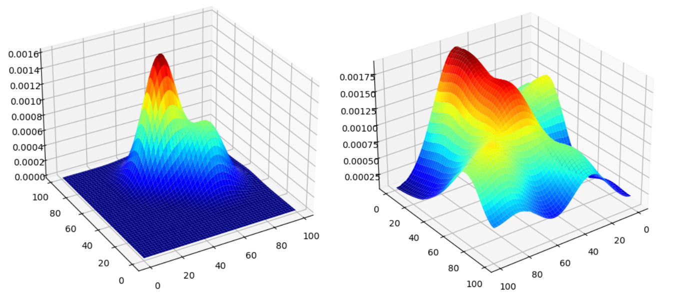

Figure 1. **Functions used to test the validity and performance of mean-shift algorithm**. The left one has global maximum on (50, 70) and a local one on (70, 50). The right one has global maximum on (55, 11) and local maximums on (52, 45), (0, 19), (79, 71).

Analysis on the left function shows us the importance of starting position and kernel size. If we start somewhere on the 'ground', where the function value is 0, with kernel size small enough, that all the values inside the patch are equal to 0, the algorithm gets stuck in the first step, since the function doesn't rise in any direction. Here bigger kernel size can help, because then more starting points are valid.

Apart from not starting on flat ground, different choices of starting points are still very important for the correct result of our algorithm. We can see that if we start on the left function on position (40, 40), we reach (with appropriate choice of kernel and stopping criteria) global maximum, and if we start on (80, 20) we reach the local one.

Now let's look at the convergence and its speed with respect to kernel size and termination criteria. We stopped the algorithm if the distance between two consecutive points was smaller than the number specified. We will analyze this two parameters together, since they are connected. With different kernel sizes, the length of our step changes (shorter step for smaller kernel), and we need to adjust the stopping criteria. We will analyze convergence on the left function in starting point (40, 40). Lets start with kernel of size $3 \times 3$ (the smallest possible, since we need it to be odd). If we set the stopping criteria to 0.1, our algorithm stops after 111 steps, in point (48, 54) - far from maximum. If we set it to 0.01, we arrive close to maximum - (51.4, 69.5) in 516 steps. We can't reach distance smaller than 0.001 in less than 100000 steps. If we look at the points, we can see that they start oscillating around (51.5, 70.5), but each step is still not smaller than the criteria. We should also notice that if we change the starting point to (39, 39), all values inside the patch are equal to 0 and we are stuck, which doesn't happen with kernel of size $5 \times 5$. On starting point (40, 40), this size of kernel reaches (51, 66) in 101 steps with termination criteria 0.1, with 0.01 we reach (51.5, 70.5) in 240 steps and again we have similar results as before for 0.001. With bigger kernel sizes the initial results are even better: 0.1 brings us to (51.5, 69.6) in 55 steps with kernel of size $9 \times 9$, in 38 steps with $11 \times 11$ and to (51.5, 70.5) with $15 \times 15$, but they all can't reach a smaller step than even 0.01. With size $19 \times 19$ we already can't be more precise than 0.1, as will also happen with bigger kernels. So here we will conclude, that the best choice would be kernel of size $15 \times 15$, with stopping criteria 0.1. But this depends a lot of the curvature of our function and what matters more - speed or precision.

With this parameters used on the function on the right of figure 1 and starting point (90, 10), we converge to (57, 13) in 48 steps. Here different starting points have even bigger impact, since we have more local minima, but we won't further analyze that, since it is expected. We can see visual representation of this convergence on Figure 2. The reason for the slight difference from the actual maximum could be the shape of the function - the maximum lies on a long saddle, which means there is not much difference in the mean value along it.

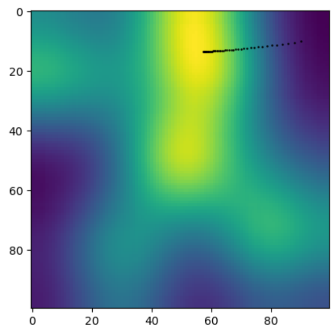

Figure 2. **Convergence of mean-shift algorithm on the right function from Figure 1**.

### B. Mean-Shift tracking

Our next goal was to use the mean-shift algorithm for tracking. We tested it on Vot14 sequences, except for *ball*, which we excluded because of problems that weren't connected to our algorithm.

First let's set the parameters. For number of bins of the histogram, the recommended value is 16, and we also found that this is the best. For 10, it was much faster, but also a lot less precise. For bigger values, it improves, but already for 20 bins, we go under 30 FPS for some sequences, which is too slow for real time use.

Here we can see the number of fails on all the sequences for parameter values (altogether this is 39 fails): $\sigma = 1, \alpha = 0, nbins = 16, \epsilon = 0.1, max_steps = 20$:

- basketball: 1
- bicycle: 1
- bolt: 0
- car: 1
- david: 2
- diving: 1
- drunk: 0
- fernando: 2
- fish1: 2
- fish2: 1
- gymnastics: 1
- hand1: 3
- hand2: 5
- jogging: 1
- motocross: 3
- polarbear: 0
- skating: 4
- sphere: 0
- sunshade: 6
- surfing: 0
- torus: 1
- trellis: 1
- tunnel: 2
- woman: 1

We can see that there are some sequences in which our algorithm works without failure, and on some it has major problems. It turns out that it fails, when there is some quick movement in the sequence - if we observe the one with the most failures, *sunshade*, we can see that we fail when the person moves so quick that we don't have it in our observed patch anymore. On the other hand, if we observe sequence *polarbear*, where we don't fail at all, we see that the bear moves slowly, so we don't have any problem following it - it always stays inside the patch. Here we could improve the algorithm with search with mean shift on a bigger patch than only inside the target position in the previous frame. That could help us where the target is as quick that it moves outside of it. We could also choose the size of the observed patch based on the length of previous move - the faster the target moves, the bigger neighborhood we should observe. But when we tried setting the enlargement factor to 1.2, it didn't improve our algorithm, instead we got 56 fails altogether.

For other parameters, we took only four sequences, two easy and two hard ones - surfing, polarbear, sunshade and hand2, and compared them on these.

For this sequences, we reached 11 fails and 1000 FPS on average with parameters mentioned above. When we compared values for $\sigma$, we got only 7 fails with value 2, 6 with value 3 and 5 with value 5, even 4 with value 20. It didn't affect speed much. For $\epsilon$ we decided the best choice to be 0.01. Smaller slowed down the algorithm, and bigger made it less precise. For $\alpha$, the choice 1 means that we update the template on every step, and it leads to worse results. We came to best results with 0, which is expected since we set the template to ground truth at the beginning.

The maximum number of steps affects some sequences positively, and some negatively. Those that we chose, have the best results with previously decided number 20, because larger ones slow down the algorithm too much. We could also limit the iteration by the length of our move, but that proved to make it worse.

## III. Conclusion

The simple implementation of the mean shift tracker proved promising, since it showed good performance on this sequences.