Correlation filters

Urša Zrimšek

I. Introduction

In this exercise we implemented more complex tracking approach - correlation filter tracker. The idea behind this approach is to find the filter that has a high response on the object we are tracking, and low response on the background. Too speed up the tracking, we are calculating the response in the Fourier domain, where correlation can be calculated as a point wise product.

II. Experiments

We were evaluating our tracker on *VOT2014* sequence, using Pytracking toolkit. With it we could compare average overlap, number of failures and average speed of tracker with different parameters - α, σ, λ and enlargement factor, to determine when it works best.

First let's look at how the performance changes when changing parameter σ, the width parameter of Gaussian function. The results represented in Table I were gathered at $\alpha = 0.1, \lambda = 0.1$ and with enlargement factor 1.5. We can see that the best overlap is achieved at 1.5, but minimal number of failures is at 2.5. We chose 2.5 as the best value, because bigger overlap can also be the reason of more failures and consequently more initializations to the ground truth position. Parameter σ tells us the shape of wanted response. With the width too small, we can see that the tracker is jumping a lot, and with wider function, we have more steady tracking. But the results then (after 2.5) begin to be worse. That could happen because the function is not precise enough.

Table I
Tracker performance with different σ.

σ	0.1	1	1.5	2	**2.5**	3
Overlap	0.43	0.47	0.50	0.49	**0.47**	0.46
Failures	120	81	74	70	**62**	74
Speed	510	513	471	500	**453**	494

Next let's look at parameter α, which defines the update speed of the filter. The results represented in Table II were gathered at $\sigma = 2.5, \lambda = 0.1$ and with enlargement factor 1.5. At $\alpha = 0$ we are not updating the filter at all, and we can see that it brings us to the worst result. Even a small update speed greatly improves performance, which is expected, since it is usual that the object we are tracking changes it's appearance during time. The update speed doesn't affect tracking speed, as there are no additional calculations.

Table II
Tracker performance with different update speeds.

α	0	0.05	**0.1**	0.5	1
Overlap	0.48	0.47	**0.47**	0.46	0.44
Failures	185	73	**62**	86	85
Speed	555	500	**488**	530	507

Next parameter we were optimizing is enlargement factor. It determines how big (depending on the object we are tracking) is the extracted patch, on which we are searching for the new position of the target. The results represented in Table III were gathered at $\sigma = 2, \alpha = 0.1, \lambda = 0.1$. We can see that we reach the best performance when we are looking for a new position on a patch 1.5× bigger then the target size. This is expected to be better as factor 1, since then there is a bigger chance that our target will still be somewhere in the observed area. When increasing this parameter, we were also slowing down the algorithm, because we were doing calculations and looking for maximum on bigger patches.

Table III
Tracker performance with different enlargement factors.

Enlarge factor	1	1.2	**1.5**	2	3
Overlap	0.45	0.44	**0.49**	0.49	0.46
Failures	73	74	**70**	80	105
Speed	935	759	**500**	366	172

Parameter λ didn't affect the results much. Not according to performance, not even on speed, which is expected because the number of calculations stays the same.

To sum up, the best parameters chosen for this tracker are $\alpha = 0.1, \sigma = 2.5, \lambda = 0.1$ and with enlargement factor 1.5. With them, it reaches results bolded in Tables II and I. In the next list we have the results on each of the sequences (sequence: number of fails, tracking speed):

- basketball: 3, 255 FPS
- bicycle: 0, 1395 FPS
- bolt: 2, 1021 FPS
- car: 0, 895 FPS
- david: 0, 485 FPS
- diving: 2, 425 FPS
- drunk: 0, 255 FPS
- fernando: 3, 97 FPS
- fish1: 11, 1157 FPS
- fish2: 6, 789 FPS
- gymnastics: 4, 496 FPS
- hand1: 6, 555 FPS
- hand2: 14, 790 FPS
- jogging: 0, 766 FPS
- motocross: 1, 101 FPS
- polarbear: 0, 440 FPS
- skating: 0, 1182 FPS
- sphere: 0, 346 FPS
- sunshade: 1, 775 FPS
- surfing: 0, 1254 FPS
- torus: 5, 1043 FPS
- trellis: 0, 472 FPS
- tunnel: 0, 379 FPS
- woman: 2, 650 FPS

Let's analyze what is the reason of behind the big number of fails and slow tracking (the average is 633 FPS) on some sequences. We can see that again our tracker fails on sequences with fast movement, like hand1, hand2 and fish1 and fish2. Torus could have additional problem besides being fast - it has a hole in the middle of our tracking surface, so the area that we're giving the most value to is actually the background. If we look at the speed of tracking, we can see that it depends most on the size of the tracking object. When we have a bigger tracking window, we do more calculations at each step and so the tracking is slower. For example, we can compare sequences fernando and motocross to fish1 and surfing to see the difference.

Further we looked at the time needed to initialize the frames and to track it. We saw that we need approximately the same amount of time for both, on average slightly more for tracking. That is expected, since we calculate the filter twice (once for updating it). But in initialization, we need to calculate Gaussian response and cosine window, and that is the reason that it is not twice faster.

III. Conclusion

We can again see that this tracker fails on sequences that have quick movements. Otherwise it works well, and we see that again we can reach good performance with simple implementation. The speed depends very much on the size of the tracked object, which we should take into consideration when using it.