Correlation filter tracking

Nino Brezac

I. Introduction

Tracking using a correlation filter is yet another efficient object tracking solution. The central idea is: the target in the next frame can be found by convolution with an optimal filter (maximum response on target). The new position of the target then updates the optimal filter. This paper provides a short overview of the correlation filter tracker, how it performs and where it can be improved.

II. Experiments

A. Baseline implementation

To start off, we will examine the baseline results of our correlation tracker. Every test run calculates three key criteria: the average overlap between the ground truth and tracker output, total number of failures and average speed of tracking in FPS.

We used VOT 2013, a collection of 16 sample object tracking sequences, as our dataset for testing purposes. VOT Toolkit Lite was used for result evaluation. Our baseline implementation used the following parameter values:

- $\sigma = 1$
- $\alpha = 0.1$

A small scalar value λ is also used, only to offset our calculations and avoid dividing by zero, but its influence was found to be minimal, and as such we used a constant value of $\lambda = 0.001$ for all tests. The results can be seen in table I.

Average overlap	0.46
Total failures	46
Average speed	901.66 FPS

Table I
BASELINE CASE: $\sigma = 1, \alpha = 0.1$

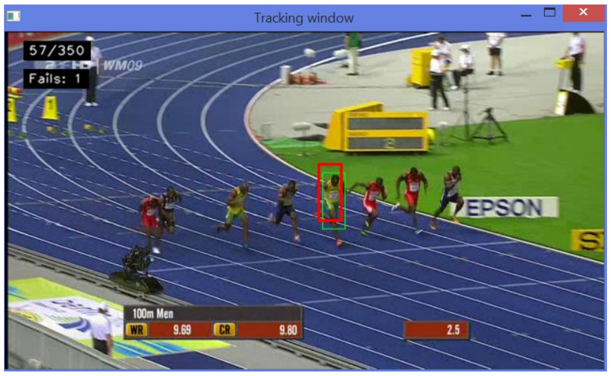

Figure 1. Our tracker in action. The sequence is 'bolt'

B. Parameter influence (α, σ)

As a follow up, we were keen on testing the parameter influence on the performance of the tracker. The first given parameter is σ. Its value is used in creating our Gaussian kernel, and it determines the actual smoothing intensity. The next parameter is *alpha*. It is a standard "forgetting value", commonly seen in various self-adjusting models and systems. It determines the measure of how much our old filter values influence our new measurements.

σ	α	Average overlap	Total failures	Average speed
0.5	0.05	0.44	74	935.23 FPS
0.5	0.1	0.45	53	959.30 FPS
0.5	0.2	0.42	53	**1020.31 FPS**
0.5	0.3	0.43	52	965.97 FPS
1	0.05	0.44	47	920.36 FPS
1	0.1	0.46	**46**	874.41 FPS
1	0.2	0.46	**46**	**1006.68 FPS**
1	0.3	0.45	**42**	993.12 FPS
2	0.05	**0.48**	60	989.3 FPS
2	0.1	**0.47**	53	**1013.26 FPS**
2	0.2	0.44	51	995.89 FPS
2	0.3	0.43	58	1005.40 FPS
3	0.05	**0.49**	69	1006.63 FPS
3	0.1	0.45	59	965.53 FPS
3	0.2	0.45	59	974.13 FPS
3	0.3	0.45	56	905.18 FPS

Table II
PARAMETER STUDY: $\sigma = 0.5, 1, 2, 3;\ \alpha = 0.05, 0.1, 0.2, 0.3$

We decided to test our tracker with given arbitrarily selected parameter values $\sigma = 0.5, 1, 2, 3$, $\alpha = 0.05, 0.1, 0.2, 0.3$. Full results are seen in table II.

The top 3 (best 3) values are bolded for each criteria (overlap, failures, speed). At first glance the general behavior is as such: for a fixed value of σ, larger α values correspond with a smaller overlap but less failures - a trade off is present. It seems that for every positive overlap change of 1% our amount of failures increases by about 10-15.

Our preferences might vary depending on the specific tracking scenario. Intuitively, if the the target we are track changes in size and shape very frequently, a higher α value is needed, as we need to update our filter more frequently. The value σ is much more empirical, we argue.

According to our results, it does seem that a larger σ is always better overlap wise, but the lowest amount of errors was seen in examples with $\sigma = 1$, whilst larger and smaller values had more and more failures. Overall, our baseline example ($\sigma = 0.5, \alpha = 0.1$) was actually almost the best case. It provided the least amount of failures (46) with reasonable overlap percentage (0.46), bested only by using a larger value of α.

C. Larger template extraction

A possible improvement on the tracker is larger template extraction. A bigger window size increases region in which we track the object, which can prove to be key in certain cases. Targets can quickly "escape" from small windows, something we can combat with appropriate window scaling - it prevents erroneus window calculation and tracks the target longer, at the cost of having a weaker filter as it includes background noise.

We used three of the best cases from the previous section and compared results using different window sizes (1, 1.5, 2). The cases we used were:

1) $\sigma = 1, \alpha = 0.2$ (overlap: 0.46, failures: 42)
2) $\sigma = 2, \alpha = 0.1$ (overlap: 0.47, failures: 53)
3) $\sigma = 3, \alpha = 0.05$ (overlap: 0.49, failures: 69)

Our results are seen in table III.

σ	α	window scale	Average overlap	Total failures	Average speed
1	0.2	1	0.46	46	1006.68 FPS
1	0.2	1.5	0.31	36	587.37 FPS
1	0.2	2	0.18	37	333.66 FPS
2	0.1	1	0.47	53	1013.26 FPS
2	0.1	1.5	**0.33**	**30**	**606.27 FPS**
2	0.1	2	0.2	31	303.92 FPS
3	0.05	1	0.49	69	1006.63 FPS
3	0.05	1.5	0.33	32	546.53 FPS
3	0.05	2	0.2	24	307.74 FPS

Table III
Larger window size: influence when using scale of 1, 1.5, 2

This time we have a "clear winner" so to speak. $\sigma = 2, \alpha = 0.1$ with its window scaled times 1.5 provided the least amount of failures (30) with biggest overlap out of all scaled test runs (0.33). Only the largest σ and smallest α had less failures (24), albeit with quite poor overlap (0.2).

We can see the general behavior is as expected: larger scales cause less failures, but in addition to smaller overlap and tracking speed. However, a scale of 2 was almost always worse than scale 1.5, which means we should take care not to oversize our window, any scaling over a factor of 2 seems pointless.

However we are not too satisfied with the results. Arguably, the decreases in failure count seem insufficient, as our overlaps and tracking speed dropped more steeply than we wished.

The mechanism seems useful in certain scenarios, where we desire more precision but don't mind the slower tracking speed. For such volatile targets, window scaling could yield better results, a good rule-of-thumb would be to find optimal results pre-scaling and then apply various scaling factors in the range from 1 to 2.

D. *Tracking speed*

We were also interested in the average speed of processing initial frames and "regular" frames. A result set on 5 sequences is seen in table IV. As we can see initialization is normally processed quite quicker than subsequent tracking iterations.

The discrepancy was smaller for, easier tracking examples: such as "cup" or "hand" and more present in harder ones such as "jump". All the runs were performed with our favorite configuration $\sigma = 1$, $\alpha = 0.2$.

sequence	initialization FPS	tracking FPS
bolt	993.67 FPS	765.88 FPS
hand	999.11 FPS	995.344 FPS
cup	1000.07 FPS	981.62 FPS
jump	999.83 FPS	685.83 FPS
car	999.12 FPS	985.27 FPS

Table IV

III. Conclusion

We have evaluated another solid means of object tracking. Our tracker performs mostly well, achieving an overlap percentage of around 45-50%, while keeping its failure count under 10 per sequence most of the time. We found that a somewhat standard values of $\sigma = 1$, $\alpha = 0.2$ worked best, but do not exclude the option of adjusting to the target in question. Experiments in scaling the tracking window were also conducted, with somewhat mediocre results, as it proved useful only in niche cases, where the target is volatile and leaves its tracking window frequently.