Long-Term tracking

Urša Zrimšek

Introduction

Up until now, all the trackers we implemented, needed manual interaction to continue tracking after the target was lost. In this exercise we want to find an approach that will detect when the tracker loses the target, start searching for it, and perform re-detection when it becomes visible again. To do this, we will modify an existing SiamFC tracker into long-term tracker.

Short-term tracker SiamFC

To get an insight of how we are improving the tracker with long term approach, we first used short-term SiamFC on the long term sequences we will also use later. We achieved:

- precision: 0.62,
- recall: 0.32,
- F-score: 0.42.

On figure 1 we can see how tracker looses the target when it is occluded and is unable to re-detect it when it appears again.

Figure 1. **Tracking with SiamFC.** On the left image tracker is successfully tracking the car. In the middle it becomes occluded, but short-term tracker doesn't detect it, so it's still tracking around the position before occlusion. On the right image the car becomes visible again, but we can't re-detect it, because we are not looking close to the new position.

Long-term tracker

The main difference between the short-term and long-term tracking is that the target can disappear for longer periods in long-term. The goal of a long-term tracker is to identify when the target is lost and then perform re-detection when it is visible again.

We approached this by observing the value of maximum response of the correlation that we get from the short-term tracker. When confidence score, defined as the ratio between current response and the response of the first frame of the sequence (where the target was manually selected), falls under a certain threshold, we detect that we are not on the target anymore, and start looking for it. We do that by sampling multiple patches from the image, selecting the one with the biggest max response, and performing another update of short-term tracker there. If confidence score is bigger than the threshold, we found the target, and we continue tracking from there, otherwise we report that the target is not visible, and continue with sampling on the next frame. For motivation, let's first look at figure 2 where we can see how the long-term tracker improved tracking where the short-term tracker failed (figure 1).

Since we are using an already pretrained short-term tracker, we are only optimizing the detection of lost target and its re-detection. We need to optimize the value of the threshold, the method of sampling during target re-detection and the number of sampled patches that we check in each frame where target is lost.

Figure 2. **Tracking with SiamFC modified into long-term tracker.** Before left image tracker is successfully tracking the car exactly the same as on figure 1. On the left it becomes occluded, long-term tracker detects that we can't see it anymore, and begins sampling around this location to find it again. On the right image the car becomes visible again and long-term tracker re-detects it and continues tracking. All samples are represented by green circles, and the best location by rectangle - blue if we're still searching for the target and red if we are tracking.

The threshold defines how similar to the target at initialization we want the patch to be to consider it as the target. If we set it to high, it will stop tracking when the target changes its appearance a bit, even if we are still on it, and if we set it too low, it won't detect that we lost it. Another part of the implementation, where we are using the threshold is when we are checking if we found the target by sampling. Again, if the threshold is too high, we won't detect that we found it, even if we sample from it's area, and if it is too low, we will detect objects that are not our target, report wrong location, and make it even harder to find the real target again. The optimal value of the threshold differs for different kinds of sampling and even more for different sequences. We observed couple of sequences, and set it to 0.45. It works perfect for sequence `car9`, but at `deer` we are detecting the target on many wrong locations. And on `cat1`, we loose it when it turns around, even though we are still on it. But later, we see that we shouldn't set it lower, as then it would detect the fox as the cat. It is not possible to set a predefined threshold optimally for all sequences, so we propose making an adaptive threshold as an improvement.

We tried three different approaches for sampling: uniform sampling over entire image and Gaussian sampling around the last target position with fixed and with growing standard deviation. To see which approach is better, we visualized tracking on different sequences. We can see that it depends on the reason why we lost the target and mostly where the target reappears again.

On figure 3 we show examples where one technique finds the target successfully and other fails (or it takes more time).

If it is occluded for a short time, it makes sense to search for it with Gaussian sampling around previous location, as it will probably appear there. But when it appears on the other side of the frame (the camera angle changes, it is occluded longer, or we detect something that isn't the target and drift away from true position), the Gaussian sampling makes it less possible to re-detect it. We can partially fix that by multiplying standard deviation with some coefficient (we achieved best results by 1.05) to the power of frames processed since we last detected

Figure 3. **Comparison of sampling techniques.** In the upper row we can see visualization of uniform sampling on sequence `car9`. On figure 2 we showed re-detection with Gaussian sampling, where the tracker finds the car immediately after it reappears, but with uniform sampling the samples are spread out over entire image, and it takes longer for one to land close on the car. In the bottom row we show where uniform sampling is better than Gaussian. In sequence `person14` there is another person similar to the target nearby, when the real target disappears. The tracker starts tracking this person and when camera moves, it drifts to the corner of the frame. When it looses the wrong target, uniform sampling quickly finds the real target (left image) and Gaussian sampling looks for the target around the corner, so it is a lot less likely to find it.

the target. That makes our search region bigger the longer we wait for the target to reappear. But if the tracker starts detecting wrong targets, and it drifts to the border, it is still hard to recover to the real location, even with growing standard deviation (specially if it again detects wrong target and shrinks the search region back).

The last parameter we need to set is the number of samples we take from each frame, where the target is not visible. We can see that even with low number of samples, the searching for the target is already a lot slower than the frames where we are only performing short-term tracking. The bigger number of samples makes it faster to detect the target when it is visible (specially when sampling uniformly), as there is more possibility that the sample lands on it, but it makes the tracking much slower. Often it happens that our target is not even on the observed frame, as it leaves or it is occluded, and then more samples can't help, they only slow down tracking. So we set this parameter to 10, but we propose that it is set to the biggest value that the computing power of the machine it runs on can take to still run fast enough according to the needs of use. We also tried 15 samples, which improved uniform sampling, but not Gaussian. We can see best results we got in table I.

Table I
Best performances of long-term trackers using different sampling techniques.

	Precision	Recall	F-score
Uniform	0.61	0.46	0.53
Gaussian	0.62	0.45	0.52
Gaussian + growing σ	0.64	0.43	0.51

In table I we can see a numerical result of the best performance we reached with each technique. This results are not accurate, as there is a lot of uncertainty coming from random sampling of locations. We would need to do a proper evaluation on a lot more sequences and with more repetitions, to get a better estimation of the performance of trackers, as we can see that it depends a lot on the type of sequences and on the time that is needed for a sample to fall onto the right location. Even the best results reached are not quite relevant, since for most of the runs, the Gaussian with growing σ was better than the constant one, but the best result shows differently. So we should take this only to show that the performance is similar, and most importantly better than one from short-term tracker, but we should rely more on visual results shown in figures above, that show that the best version of the tracker depends on the sequence on which it is used.

Conclusion

We can see that for targets that have a possibility to disappear, long-term tracking improves performance, even with little optimization. We could still implement adaptive threshold, and adapt standard deviation of Gaussian sampling to stay bigger if we only detect the target for couple of frames (as it is probably a wrong detection). We could also improve the performance by taking more samples. Nice advantage of this tracker (comparing to the previously implemented) is, that the size of the target does not affect the speed of tracking. So we can set the biggest possible number of samples that our machine can handle and leave it the same for all targets. All our previous trackers had good performance only because it was possible to manually correct them when they lost the target. But this tracker has the possibility to continue without intermission, which is a great advantage.