# Exercise 3: Correlation filter tracking

Advanced Computer Vision Methods

2020/2021

This exercise will address some more complex tracking approaches. You will implement a correlation filter tracker. In the second part of the project you will integrate the implemented tracker to the VOT evaluation framework (or a lite version of the toolkit) and perform an evaluation on the VOT 2013/14 dataset.

## Submission instructions

The exercise should be submitted on-line on the course website. The submission should contain a report and the source code. Do not submit the data which was given as a part of the instructions, except you add some data that you collect from other sources. The report is the most important part of the submission, so make sure that you spend enough time on it, after you are done with coding and experiments. Note that a strict page limit of the report is **two pages maximum**. Detailed description of grading can be found at the end of the document.

The submissions should be done by the deadline. Late submissions are possible, however, a strict deadline is one week after the first one. A baseline for the late submission is 70%. After that you cannot submit the exercise anymore. The assistant will review the submissions and provide a feedback within a week after the deadline (unless stated otherwise). Each exercise must be done individually and all submissions will be checked for plagiarism. A student will be notified about the grade in the submission feedback. Passing all five exercises is required to pass this part of the course.

## Assignment 1: Correlation filters

First you will implement simplified version of the MOSSE [1] correlation filter that you have heard about at the lectures. The main idea behind correlation filters is to learn the filter so that it has high correlation response on the object and low response on the background. In the first frame $t = 1$ construct filter $\mathbf{H}$ using equation:

$$\hat{\mathbf{H}}^{\dagger} = \frac{\hat{\mathbf{G}} \odot \hat{\mathbf{F}}^{\dagger}}{\hat{\mathbf{F}} \odot \hat{\mathbf{F}}^{\dagger} + \lambda}, \tag{1}$$

where $\mathbf{G}$ is a 2-dimensional Gaussian function and $\mathbf{F}$ is feature patch (i.e., grayscale image patch centered at object location). Operation $\odot$ is a point-wise product, division is also calculated element-wise and the $\dagger$ denotes complex-conjugate operator. Note that the $\hat{}$ represents variable in Fourier domain i.e., $\hat{a} = \mathcal{F}(a)$. Fourier transform must be performed in **2-dimensions** e.g., `numpy.fft.fft2`.

After the filter has been constructed it can be used to localize the target (i.e., $t = 2, 3, 4, ...$). Implement the localization step using equation

$$\mathbf{R} = \mathcal{F}^{-1}(\hat{\mathbf{H}}^{\dagger} \odot \hat{\mathbf{F}}), \tag{2}$$

where $\mathbf{R}$ represents 2-dimensional correlation response and new target location is defined as position of the maximum peak in the response and $\mathcal{F}^{-1}$ is the inverse Fourier transform.

Using constant filter $\mathbf{H}$ does not model the target well, especially when it is changing its appearance. That is the reason for online update of the filter and it is typically realized as exponential forgetting:

$$\hat{\mathbf{H}}_t^{\dagger} = (1 - \alpha)\hat{\mathbf{H}}_{t-1}^{\dagger} + \alpha\hat{\tilde{\mathbf{H}}}^{\dagger}. \tag{3}$$

The updated filter at frame $t$ is denoted as $\hat{\mathbf{H}}_t^{\dagger}$ and the filter from previous frame is denoted as $\hat{\mathbf{H}}_{t-1}^{\dagger}$. Filter at the current frame, obtained with the Equation (1), is denoted as $\hat{\tilde{\mathbf{H}}}^{\dagger}$. An important parameter here is the update speed $\alpha$ (typically a low number i.e., $0.02, 0.1, ...$). Observe what is the impact of this parameter to tracking performance. What is the optimal value for $\sigma$ parameter in Gaussian function $\mathbf{G}$? In which situations the correlation filter perform the worst? For additional implementation tips see Exercise 3 slides.

# Assignment 2: Better filter update (additional)

(*Start this after you are done with Assignment 1 and 3.*) For additional points implement the actual MOSSE tracker. Update (and construct) correlation filter in a different way: store numerator and denominator of the filter separately. Note that the localization step is implemented differently, too. For additional details see paper [1]. What is the difference in performance comparing to the correlation filter from Assignment 1? For additional performance boost you can use PSR measure to improve filter update step by detecting occlusion, like described in [1].

# Assignment 3: Tracking performance evaluation

You have to integrate your tracker with **one** of the evaluation tools described in the following. Evaluate your correlation filter tracker with one of the toolkits and report the performance on one of the VOT datasets (VOT13 [2] or VOT14 [3] are recommended, since older are larger). We recommend to use the lite version of the toolkit since it is simpler to integrate.

## Tracking toolkit (lite)

The toolkit and instructions how to integrate your tracker can be found at the github page: `https://github.com/alanlukezic/pytracking-toolkit-lite`. Include a latex table (terminal output of the `compare_trackers` command) or an AR-plot generated by the toolkit in your report.

### VOT toolkit

VOT toolkit can be set-up using Matlab or Python and a C++ compiler. If you do not have these, we would recommend you to use the lite toolkit described in the previous section.

Checkout the toolkit from the Github. You can use a Matlab version: `https://github.com/votchallenge/vot-toolkit` or the new Python version of the toolkit: `https://github.com/votchallenge/vot-toolkit-python`. For more details how to integrate and evaluate your tracker using the VOT toolkit see VOT webpage: `http://www.votchallenge.net/howto/`. Keep in mind that the evaluation process may take quite some time to complete - up to several days in extreme circumstances if the tracker is very slow. Include a latex table or any of the graphs generated by the toolkit in the report.

## Additional information

You can test your tracker using the same `run_tracker.py` script from the second project. After that, the integration of the tracker into the toolkit should be easier. We also recommend you to use the `get_patch` function from the Project 2. In the `ex3_utils.py` two functions are provided for you to make the implementation easier:

- `create_cos_window`: creates the cosine window (for input arguments see the comments in the source code), you can find information where the cosine window should be used in the lecture and exercise slides.
- `create_gauss_peak`: creates the Gaussian peak (for input arguments see the comments in the source code), you can find information where the Gaussian peak should be used in the lecture and exercise slides.

## Grading

The tasks marked with *Req.* are required to successfully complete the exercise. The number in the brackets represents number of points of other tasks while *Add.* stands for additional tasks which can bring you more than 100 points.

- (Req.) Implement the correlation filter tracker and integrate it with the toolkit (VOT or toolkit-lite). Show tracking performance as the number of failures and average overlap (in the report include the table with performance analysis, outputed by the toolkit).
- (20) Show how does tracking performance change with different parameters $\alpha$ (update speed) and $\sigma$ (parameter of a Gaussian $\mathbf{G}$).
- (20) Construct filter H using larger template F (extract larger region - capture more background). Does the performance improve? How does the performance change when changing the increase factor?
- (10) Report tracking speed (in frames-per-second): measure time needed to process a single frame and average measurements. Observe tracking speed over different sequences and report them. Does the time needed to process initialization frame

and other frames differ significantly? Do not take into account the time needed to read frame from disk.

- (Add) implement the actual MOSSE tracker (which updates numerator and denominator separately). Report how does tracking performance change.

# References


[1] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *Comp. Vis. Patt. Recognition*, pages 2544–2550. IEEE, 2010.

[2] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Čehovin, G. Nebehay, G. Fernandez, and T. e. a. Vojir. The visual object tracking vot2013 challenge results. In *Vis. Obj. Track. Challenge VOT2013, In conjunction with ICCV2013*, pages 98–111, Dec 2013.

[3] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Čehovin, G. Nebehay, T. Vojir, and G. et al. Fernandez. The visual object tracking vot2014 challenge results. In *Proc. European Conf. Computer Vision*, pages 191–217, 2014.