High-Speed Tracking with Kernelized Correlation Filters

Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista

Ngoc Phuc Nguyen
Electrical and Computer Engineering
Concordia University
Montreal, QC, Canada, June 22, 2020
philngtn@gmail.com

***Abstract*—The paper pinpoints the problem of modern tracking is under-sampling of negative samples, and then introduces the analytic model to overcome that constraint. By figuring out the properties of circulant matrices, which can be diagonalizable with Discrete Fourier Transform, the computation and storage of classifier can be reduced by several orders of magnitude. Moreover, authors also derive a new kernelized correlation filter (KCF) and a linear multi-channel filter (DCF), which has matched the performance of non-linear filter but still maintain a very low computational complexity. Both filters outperform top-ranking tracker such as Struck or TLD, while comfortably running at hundreds frames-per-second.**

Keywords—Visual tracking, circulant matrices, discrete Fourier transform, kernel methods, ridge regression, correlation filters

Table of Contents

I. INTRODUCTION

One of the huge leap of recent visual tracking was the use of discriminative learning methods. Online learning can be seen as a problem of tracking since it requires heavy computation. Given an initial image patch that contain the target, the goal is to discriminate its appearance and that of the environment. This classifier can be evaluated exhaustively at many locations, in order to detect target in subsequent frames. Of course, each new detection provides a new image patch that can be used to update the model.

Focusing on the finding of target (the positive sample) is a right way to start, however, the core tenet of discriminative method is fetching more negative samples or the relevant environment to the classifier. The most commonly used negative samples are images patches from different locations and scales, reflecting the prior knowledge of classifier.

Due to the time-sensitive nature of tracking, the modern trackers walk a fine line between incorporating as many samples as possible and keeping computational demand low. It is common practice to randomly choose only a few samples each frame. [3], [4], [5], [6], [7].

It is argued that the main factor of inhibiting performance of tracking is undersampling negative samples. In the paper, authors have developed tools to analytically incorporate thousands of samples at different translations, without iterating them explicitly. This is made possible by the discovery that, in the Fourier domain, some learning algorithms actually become easier as we add more samples, if we use a specific model for translations.

These analytical tools, namely circulant matrices, provide a useful bridge between popular learning algorithms and classical signal processing. Thank to that, authors can be able to propose the tracker based on kernel ridge regression that does not suffer from the "curse of kernelization", which is its larger asymptotic complexity. In another word, it can be seen as a kernelized version of a linear-correlation filter, which forms the basis for the fastest trackers available [9], [10]. The authors also leverage the powerful kernel trick at the same computational complexity as linear correlation filters. The framework easily incorporates multiple feature channels, and by using a linear kernel we show a fast extension of linear correlation filters to the multi-channel case.

II. RELATED WORK

A. Single channel kernelized correlation filter

The initial work of this paper [29] just proposes the kernelized correlation filter with the limitation of one single channel. The old paper also shown the connection between ridge regression with cyclically shifted samples and classical correlation filter, this allow the classifier to learn faster with $O(n\log n)$ perform a costly inverse matrix in the close solution form of the ridge regression.

In this paper, authors are using the simpler diagonalization technique and pushing the classifier to work with multiple channels, which can use the state-of-the-art features such as HOG to significantly improve the

performance of the system. The paper additionally presents a linear multi-channel filter with very low in computation complexity that almost match the performance of non-linear kernels.

III. Theory of Implemented approach

A. Circulant Matrices

Consider a $n \times 1$ vector representing a patch with the object of interest (target), denoted **x**. We can see this is a base sample or the positive sample. The goal is to train the classifier with both a positive sample and negative samples, which can be obtained by translating the base sample.

One way to do one-dimensional translation of a vector is using cyclic shift operator, which means multiply the vector with the permutation matrix **P** in the equation (1). The product $\mathbf{Px} = [x_n, x_1, x_2, \dots, x_{n-1}]^T$ shifts **x** by one element, modeling

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 1 & \cdots & 0 \end{bmatrix} \quad (1)$$

a small translation. We can achieve a larger translation by using the matrix power $\mathbf{P}^u\mathbf{x}$. An example of 2D image translated vertically is shown in Fig 1 with different values of u.

Figure 1 Examples of vertical cyclic shifts of a base sample (the positive sample).

Due to the cyclic property we can get the same signal **x** after every *n* shifts. That means the full set of shifted signals can be obatained as:

$$\{\boldsymbol{P}^u\boldsymbol{x} \mid u = 0,1, \dots, n-1\}$$

or represented as a row of a data matrix **X**:

$$\boldsymbol{X} = C(\boldsymbol{x}) = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \\ x_n & x_1 & x_2 & \cdots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \cdots & x_{n-2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{Px} \\ \boldsymbol{P}^2\boldsymbol{x} \\ \vdots \\ \boldsymbol{P}^{n-1}\boldsymbol{x} \end{bmatrix} \quad (2)$$

what we have found is a circulant matrix which has some interesting properties.

One of the most intriguing property is that **all** circulant matrices are made diagonal by discrete Fourier transform, regardless of the generating vector **x**. The matrix **X** can be rewritten as

$$\boldsymbol{X} = \boldsymbol{F}diag(\widehat{\boldsymbol{x}})\boldsymbol{F}^H \quad (3)$$

where **F** and $\mathbf{F}^{\mathrm{H}}$ are constant matrix that does not depend on **x**, and $\widehat{\boldsymbol{x}}$ denotes the DFT of the vector **x**, $\widehat{\boldsymbol{x}} = \mathcal{F}(\boldsymbol{x})$. From now on, we will always use a hat, such as $\widehat{\boldsymbol{x}}$, as shorthand for the DFT of a vector. The constant matrix **F** and is known as the DFT matrix that computes the DFT of any input vector, as $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{Fx}$. $\mathbf{F}^{\mathrm{H}}$ is the Hermitian transpose, i.e., $\mathbf{F}^{\mathrm{H}} = (\mathbf{F}^*)^T$, and $\mathbf{F}^*$ is the complex-conjugate of **F**.

During this subsection, I have learned about one of the most interesting property of circulant matrices, which is the Eigen-decomposition of the matrices. By doing that, we do not need to iterating all the sample explicitly, and in the Fourier domain, it is easier to add more samples if we use the circulant translations. This paves the way to simplify the computation of the classification to several magnitudes, which enable to make the classifier working in real-time settings.

B. Kernelized Ridge Regression

The goal of training is to find a function $f(\boldsymbol{z}) = \boldsymbol{w}^T\boldsymbol{z}$ that minimizes the square error over sample $\boldsymbol{x}_i$ and the target y_i,

$$\min_{\boldsymbol{w}} \sum_i (f(\boldsymbol{x}_i) - y_i)^2 + \lambda\|\boldsymbol{w}\|^2 \quad (4)$$

The λ is the regularization term that control the overfiting of training data. According to the kernel trick, the solution **w** can be express as a linear combination of the samples:

$$\boldsymbol{w} = \sum_i \alpha_i \varphi(\boldsymbol{x}_i) \quad (5)$$

where $\varphi(\boldsymbol{x}_i)$ is the function that map the sample $\boldsymbol{x}_i$ to non-linear feature-space.

We can drive the function $f(\boldsymbol{z}) = \boldsymbol{w}^T\boldsymbol{z}$ as the linear combination of the samples as

$$f(\boldsymbol{z}) = \boldsymbol{w}^T\boldsymbol{z} = \sum_{i=1}^{n} \alpha_i \kappa(\boldsymbol{x}_i, \boldsymbol{z}) \quad (6)$$

The close form solution of kernelized version of (4) is given by [8]

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} \quad (7)$$

where $\boldsymbol{\alpha}$ is a vector of coefficient α_i, that represent the solution in the dual space (Representer Theorem [23, p. 89]). The dot-product dot product of all pairs of samples are usually stored in a $n \times n$ kernel matrix **K**, with elements

$$K_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad (8)$$

where κ is the kernel function (e.g., Gaussian or Polynomial). Now, if we can prove **K** is circulant matrix, we can diagonalize (6) and obtain a fast solution as for the linear case. According to the paper, some most useful kernels preserve the circulant structure by treating all dimensions of the data equally.

Since the matrix **K** in equation (6) is circulant, hence, we can diagonalize to the linear case, this is the training algorithm for a signal.

$$\boldsymbol{\alpha} = \frac{\widehat{\boldsymbol{y}}}{\widehat{\boldsymbol{k}}^{\boldsymbol{xx}} + \lambda} \quad (9)$$

where $\boldsymbol{k}^{\boldsymbol{xx}}$ is the first row of the kernel matrix $\mathbf{K} = \mathrm{C}(\boldsymbol{k}^{\boldsymbol{xx}})$. We can see the kernel correlation of two vector, **x** and **x**, is the vector $\boldsymbol{k}^{\boldsymbol{xx}}$ with elements

$$k_i^{\boldsymbol{xx}} = \kappa(\boldsymbol{x}, \boldsymbol{P}^{i-1}\boldsymbol{x}) \quad (10)$$

this can be seen as the *kernel auto-correlation*.

This subsection shows that we just need to compute the auto-correlation of $n \times 1$ vector, that will be scale linearly with the number of sample. With the conventional wisdom of kernel method, the calculation of $n \times n$ kernel matrix **K** will scale quadratically with the increasing of samples.

In order to track the image, we usually evaluate the function f(**z**) on several image locations or several candidate patches, these patches also can be model as the cyclic shifted.

Let the $\boldsymbol{K^z}$ is the asymmetric kernel matrix between all training samples and all candidate patches. Since the samples and patches have different base of cyclic shifts, denote as n and m, each element of kernel matrix $\boldsymbol{K^z}$ is given by:

$$k^{xz}_{m,n} = \kappa(\boldsymbol{P}^{n-1}\boldsymbol{x}, \boldsymbol{P}^{m-1}\boldsymbol{z}) \quad (11)$$

It can be proved that the $\boldsymbol{K^z}$ is circulant for appropriate kernels, hence we just need the first row of the matrix:

$$\boldsymbol{K^z} = C(\boldsymbol{k^{xz}}) \quad (12)$$

where $\boldsymbol{k^{xz}}$ is the kernel correlation of **x** and **z**. The equation (6) could be rewritten as

$$\boldsymbol{f(z)} = (\boldsymbol{K^z})^T\boldsymbol{\alpha} \quad (13)$$

to compute the equation (13) effectively we just diagonalize it to get

$$\hat{\boldsymbol{f}}(\boldsymbol{z}) = \hat{\boldsymbol{k}}^{\boldsymbol{xz}} \odot \hat{\boldsymbol{\alpha}} \quad (14)$$

notice that the **f(z)** is a vector, containing the output for all cyclic shifts of **z** or the full detection response.

In this section, we can see that the authors introduced a fast way to do detect the target in many locations of candidate patches, weighted by the learned coefficient.

C. Fast Kernel Correlation

To compute kernel correlation $\boldsymbol{k^{xx}}$ or $\boldsymbol{k^{xz}}$ effectively, we need to find the way to calculate the kernel for all relative shifts of two input vectors. Typically, the dot product of kernel have a form

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = g(\boldsymbol{x}^T\boldsymbol{x'}) \quad (15)$$

for some function g, then the $\boldsymbol{k^{xx'}}$ has elements

$$k^{\boldsymbol{xx'}}_i = \kappa(\boldsymbol{x}, \boldsymbol{P}^{i-1}\boldsymbol{x'}) = g(\boldsymbol{x'}^T\boldsymbol{P}^{i-1}\boldsymbol{x}) \quad (16)$$

let g can work with element wise on any input vector, we can rewrite the equation (16) in a vector form

$$\boldsymbol{k^{xx'}} = g(C(\boldsymbol{x})\boldsymbol{x'}) = g(\boldsymbol{X}\boldsymbol{x'})$$

when we diagonalize the matrix **X**, the above equation can be rewrite as:

$$\boldsymbol{k^{xx'}} = g(\boldsymbol{\mathcal{F}}^{-1}(\hat{\boldsymbol{x}}^* \odot \hat{\boldsymbol{x}}'))$$

where $\boldsymbol{\mathcal{F}}^{-1}$ means inverse DFT.

In the paper, authors choose to use the polynomial kernel, Radial Basic Function and Gaussian kernel to form the general case for the kernel correlation. The reason for this is that these kernels preserve the circulant structure of a signal.

For Gaussian kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = exp\left(-\frac{1}{\sigma^2}\|\boldsymbol{x} - \boldsymbol{x'}\|^2\right) \quad (17)$$

then we get

$$\boldsymbol{k^{xx'}} = exp\left(-\frac{1}{\sigma^2}(\|\boldsymbol{x}\|^2 + \|\boldsymbol{x'}\|^2 - 2\boldsymbol{\mathcal{F}}^{-1}(\hat{\boldsymbol{x}}^* \odot \hat{\boldsymbol{x}}'))\right) \quad (18)$$

D. Multiple Channels

The dot-product of multiple channels could be calculated simply by summing all the dot-product at each channel, thank to the linearity of DFT, we could sum the result of each channel in the Fourier domain. Thus, we can apply this reasoning to the Gaussian kernel to get the multi-channels kernel correlation function. We can assume that the vector **x** concatenates the individual vectors for i channels, for example 31 gradient orientation bins for a HOG, as $\boldsymbol{x} = [\boldsymbol{x_1}, \dots, \boldsymbol{x_i}]$

$$\boldsymbol{k^{xx'}} = exp\left(-\frac{1}{\sigma^2}(\|\boldsymbol{x}\|^2 + \|\boldsymbol{x'}\|^2 - 2\boldsymbol{\mathcal{F}}^{-1}(\sum_i \hat{\boldsymbol{x}}^*{}_i \odot \hat{\boldsymbol{x}}'_i))\right) \quad (19)$$

IV. Implementation and Results

The paper's algorithm is implemented on Python, using Gaussian kernel, all the testing sequences are working with HOG feature extraction and raw pixel. The test sequences are Dog1, FaceOcc2, FaceOcc1, Girl and Liquor, which are downloaded at http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html.

A. Liquor

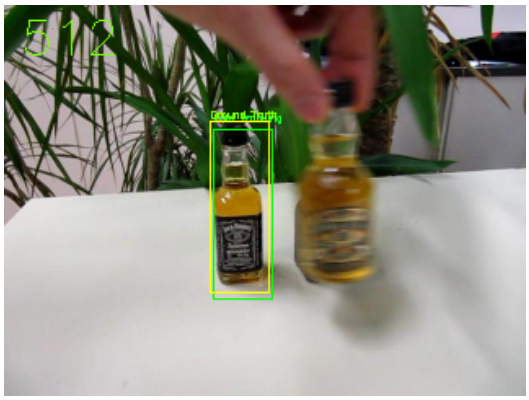

Figure 2 Object Tracking with on Liquor sequence

- Tracking object with HOG feature extractions:

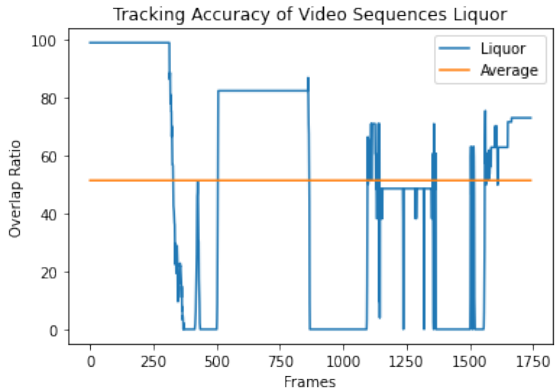

Figure 3 Tracking accuracy of Liquor sequence with HOG feature extraction

The computation time is 397.26 seconds, then the FPS = 4.38.

- Tracking object with raw pixels:

The computation time is 195.3, then the FPS = 8.91

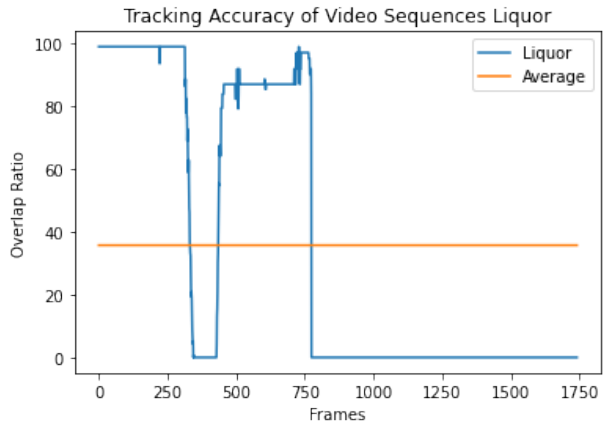

Figure 4 Tracking accuracy of Liquor sequence with raw pixels

B. Dog1 sequence

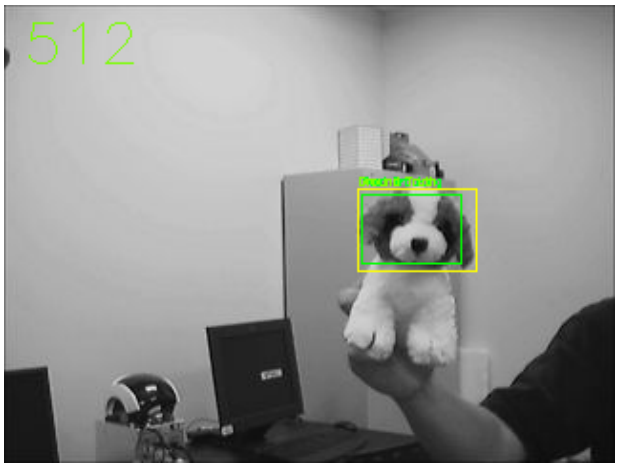

Figure 5 Object Tracking with on Dog1 sequence

- Tracking object with HOG feature extractions:

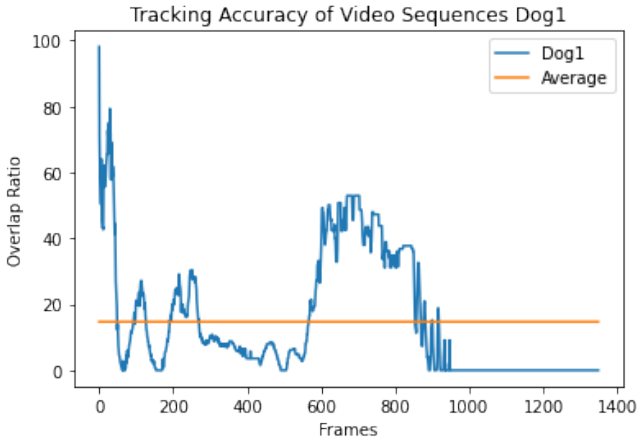

Figure 6 Object Tracking with HOG feature extraction on Dog1 sequence

The computation time is 74.9 seconds, then the FPS = 18.02.

- Tracking object with raw pixels:

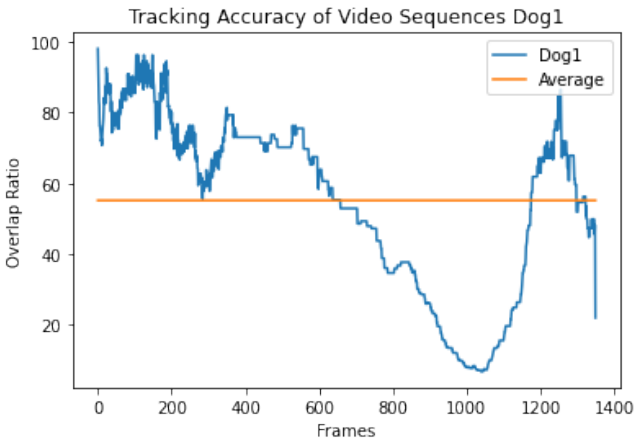

Figure 7 Tracking accuracy of Dog1 sequence with raw pixels

The computation time is 72.9 seconds, then the FPS = 18.57

C. FaceOcc2 sequence

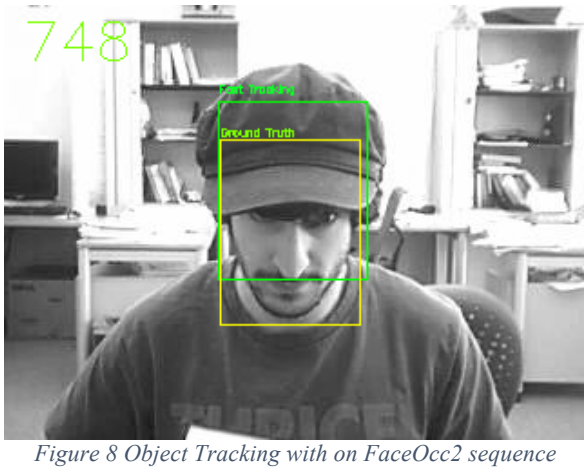

Figure 8 Object Tracking with on FaceOcc2 sequence

- Tracking object with HOG feature extractions:

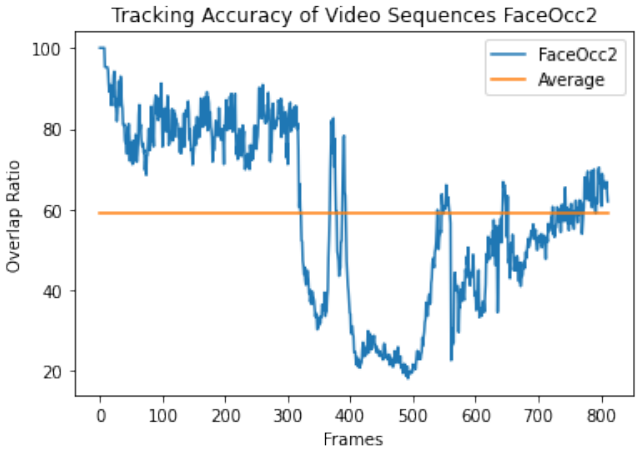

Figure 9 Object Tracking with HOG feature extraction on FaceOcc2 sequence

The computation time is 335.9 seconds, then the FPS = 2.4.

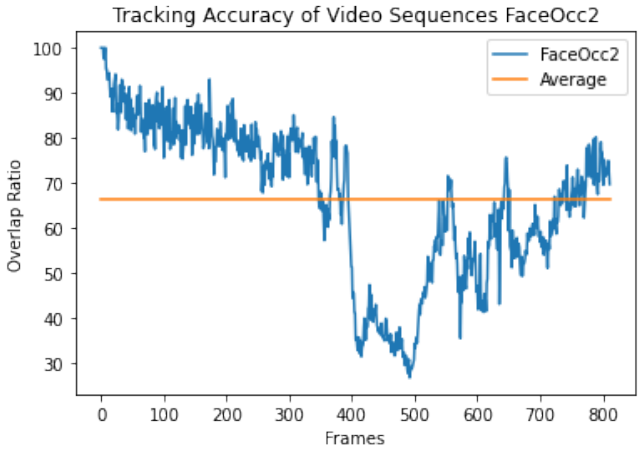

Figure 10 Tracking accuracy of FaceOcc2 sequence with raw pixels

The computation time is 195.9 seconds, then the FPS = 4.14.

D. FaceOcc1 sequence

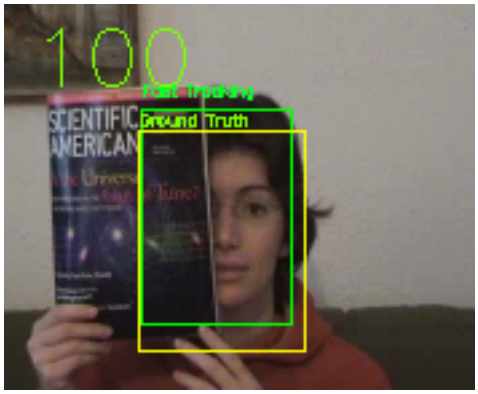

Figure 11 Object Tracking with on FaceOcc1 sequence

- Tracking object with HOG feature extractions:

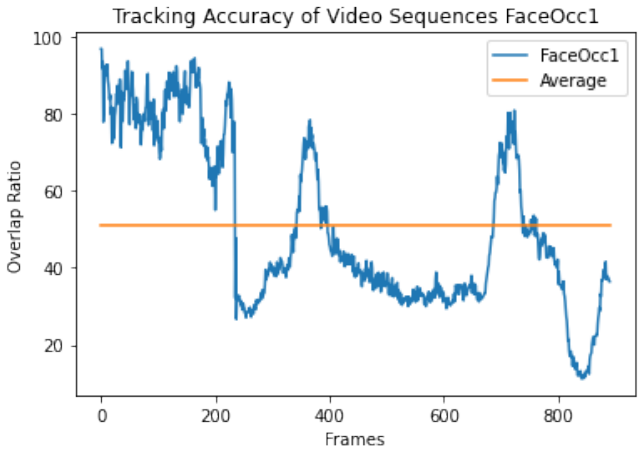

Figure 12 Object Tracking with HOG feature extraction on FaceOcc1 sequence

The computation time is 234.9 seconds, then the FPS = 3.79.

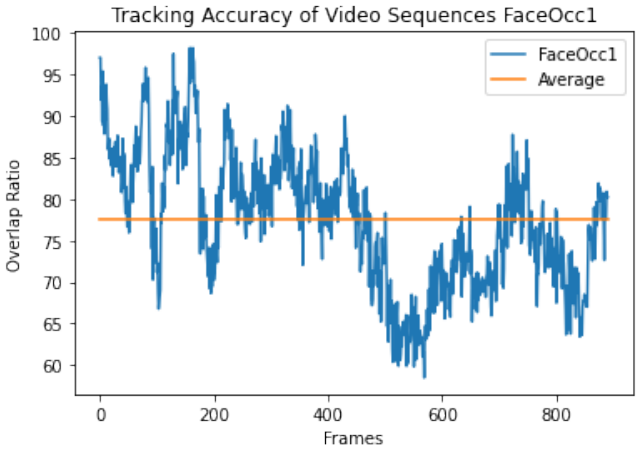

Figure 13 Tracking accuracy of FaceOcc1 sequence with raw pixels

The computation time is 143.03 seconds, then the FPS = 6.23.

E. Girl sequence

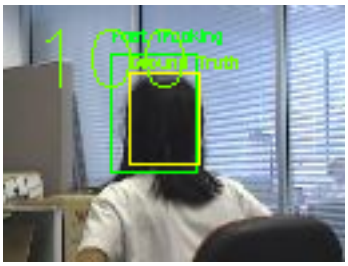

Figure 14 Object Tracking with on Girl sequence

- Tracking object with HOG feature extractions:

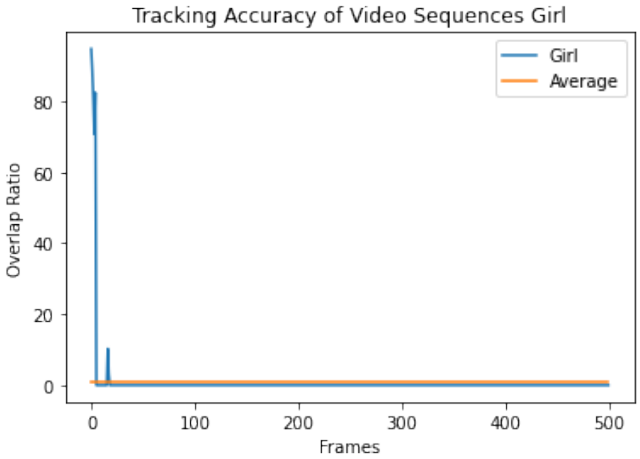

Figure 15 Object Tracking with HOG feature extraction on Girl sequence

The computation time is 43.8 seconds, then the FPS = 11.42.

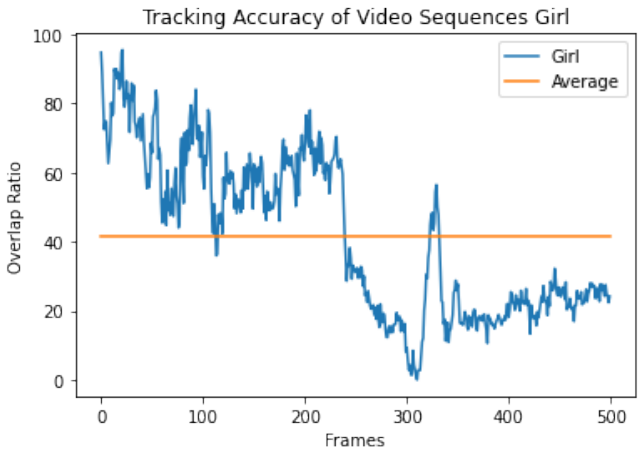

Figure 16 Tracking accuracy of Girl sequence with raw pixels

The computation time is 9.75 seconds, then the FPS = 51.2.

The computational time is extremely low compared to the one the states in the paper. The reason for this is that Python is a slow programming language but it is a dynamic one. If we want to speed up the computational time, other programming language should be used such as C++, the result of tracker when using the HOG feature is not good as we are expected. In the paper, authors used the advanced variation of HOF called FHOG, which calculate 31 gradient orientation bins.

The results for raw pixels present a very good tracker however, more works should be done on the feature extraction. I have learned a lot from this paper. In the first part, I have researched the dual space from Linear Programming to understand how a solution in primal space can be mapped to a dual space, which is given the lower boundary of the optimal problems in this case is ridge regression. Then, beside computing on the raw pixels, I have chance to research that the feature extraction, such as HOG, can improve the classifier significantly.

V. Conclusion and Future Work

The paper has introduced a new approach to train the classifier with multiple images to provide the tracker information about the environment around the target. The diagonalization by DFT of circulant matrices give a foundation to deal with image translations. This blueprint is applied on linear and kernel ridge regression which allow the tracker could run very fast since the computational cost will linearly increase proportionally to the amount of samples. In the future work, I will try to rewrite the code on other programming language such as C++ to speed up the FPS. Moreover, I will develop the FHOG library for python to enhance the accuracy of the tracker.

References

[1] A. Smeulders, D. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, “Visual tracking: An experimental survey,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 7, pp. 1442–1468, Jul. 2014.

[2] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, “Recent advances and trends in visual tracking: A review,” Neurocomput., vol. 74, no. 18, pp. 3823–3831, Nov. 2011.

[3] K. Zhang, L. Zhang, and M.-H. Yang, “Real-time compressive tracking,” in Proc. Eur. Conf. Comput. Vis., 2012, pp. 864–877.

[4] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learningdetection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 34, no. 7, pp. 1409–1422, Jul. 2012.

[5] B. Babenko, M. Yang, and S. Belongie, “Robust object tracking with online multiple instance learning,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 8, pp. 1619–1632, Aug. 2011.

[6] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, “Online random forests,” in Proc. 3rd IEEE Int. Conf. Comput. Vis. Workshop On-line Comput. Vis., 2009, pp. 1393–1400.

[7] S. Hare, A. Saffari, and P. Torr, “Struck: Structured output tracking with kernels,” in Proc. Int. Conf. Comput. Vis., 2011, pp. 263–270.

[8] R. Rifkin, G. Yeo, and T. Poggio, “Regularized least-squares classification,” Nato Sci. Ser. Sub Ser. III, vol. 190, pp. 131–154, 2003.

[9] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” in Proc. Comput. Vis. Pattern Recognit., 2010, pp. 2544–2550.

[10] D. S. Bolme, B. A. Draper, and J. R. Beveridge, “Average of synthetic exact filters,” in Proc. Comput. Vis. Pattern Recognit., 2009, pp. 2105–2112.

[11] Y. Wu, J. Lim, and M. H. Yang, “Online object tracking: A benchmark,” in Proc. Comput. Vis. Pattern Recognit., 2013, pp. 2411–2418.

[12] S. Avidan, “Support vector tracking,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 8, pp. 1064–1072, Aug. 2004.

[13] H. Grabner, C. Leistner, and H. Bischof, “Semi-supervised on-line boosting for robust tracking,” in Proc. Eur. Conf. Comput. Vis., 2008, pp. 234–247.

[14] Y. Wu, B. Shen, and H. Ling, “Online robust image alignment via iterative convex optimization,” in Proc. Comput. Vis. Pattern Recognit., 2012, pp. 1808–1814.

[15] L. Sevilla-Lara and E. Learned-Miller, “Distribution fields for tracking,” in Proc. Comput. Vis. Pattern Recognit., 2012, pp. 1910–1917.

[16] C. Lampert, M. Blaschko, and T. Hofmann, “Beyond sliding windows: Object localization by efficient subwindow search,” in Proc. Comput. Vis. Pattern Recognit., 2008, pp. 1–8.

[17] B. Alexe, V. Petrescu, and V. Ferrari, “Exploiting spatial overlap to efficiently compute appearance distances between image windows,” in Proc. Adv. Neural Inf. Processing Syst., 2011, pp. 2735–2743.

[18] H. Harzallah, F. Jurie, and C. Schmid, “Combining efficient object localization and image classification,” in Proc. Int. Conf. Comput. Vis., 2009, pp. 237–244.

[19] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, “Multiple kernels for object detection,” in Proc. Int. Conf. Comput. Vis., 2009, pp. 606–613.

[20] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[21] R. C. Gonzalez and R. E. Woods, Digital Image Processing. Upper Saddle River, NJ, USA: Prentice Hall, 2008.

[22] P. Dollar, R. Appel, S. Belongie, and P. Perona, “Fast feature pyramids for object detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[23] B. Scholkopf and A. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA, USA: MIT Press, 2002.

[24] D. Casasent and R. Patnaik, “Analysis of kernel distortion-invariant filters,” Proc. SPIE, vol. 6764, p. 1, 2007.

[25] R. Patnaik and D. Casasent, “Fast FFT-based distortion-invariant kernel filters for general object recognition,” Proc. SPIE, vol. 7252, 2009, p. 1.

[26] K.-H. Jeong, P. P. Pokharel, J.-W. Xu, S. Han, and J. Principe, “Kernel based synthetic discriminant function for object recognition,” in Proc. IEEE Int. Conf. Acoust. Speech Signal Process., 2006, pp. 1–5.

[27] C. Xie, M. Savvides, and B. Vijaya-Kumar, “Kernel correlation filter based redundant class-dependence feature analysis (KCFA) on FRGC2.0 data,” in Proc. 2nd Int. Conf. Anal. Model. Faces Gestures, 2005, pp. 32–43.

[28] A. Mahalanobis, B. Kumar, and D. Casasent, “Minimum average correlation energy filters,” Appl. Optics, vol. 26, pp. 3633–3640, 1987.

[29] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista,“Exploiting the circulant structure of tracking-by-detection with kernels,” in Proc. Eur. Conf. Comput. Vis., vol. 7575, pp. 702–715, 2012.

[30] J. Revaud, M. Douze, S. Cordelia, and H. Jegou, “Event retrieval in large video collections with circulant temporal encoding,” in Proc. Comput. Vis. Pattern Recognit., 2013, pp. 2459–2466. [31] J. F. Henriques, J. Carreira, R. Caseiro, and J. Batista, “Beyond hard negative mining: Efficient detector learning via blockcirculant decomposition,” in Proc. Int. Conf. Comput. Vis., 2013, pp. 2760–2767.

[32] H. K. Galoogahi, T. Sim, and S. Lucey, “Multi-channel correlation filters,” in Proc. Int. Conf. Comput. Vis., 2013, pp. 3072–3079.

[33] V. N. Boddeti, T. Kanade, and B. V. Kumar, “Correlation filters for object alignment,” in Proc. Comput. Vis. Pattern Recognit., 2013, pp. 2291–2298.

[34] R. M. Gray, Toeplitz and Circulant Matrices: A Review. Boston, MA, USA: Now Publishers, 2006.

[35] P. J. Davis, Circulant Matrices. Providence, RI, USA: Amer. Math. Society, 1994.

[36] A. Vedaldi and A. Zisserman, “Efficient additive kernels via explicit feature maps,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 34, no. 3, pp. 480–492, Mar. 2011