2018 IEEE Winter Conference on Applications of Computer Vision

*E*nKCF: Ensemble of Kernelized Correlation Filters for High-Speed Object Tracking

Burak Uzkent
Chester F. Carlson Center for Imaging Science
Rochester Institute of Technology
bxu2522@rit.edu

YoungWoo Seo
Independent Robotics Research (IR2)
youngwoo.blank.seo@gmail.com

Abstract

Computer vision technologies are very attractive for practical applications running on embedded systems. For such applications, it is desirable for the deployed algorithms to run in high-speed and require no offline training. To develop a single-target tracking algorithm with these properties, we propose an ensemble of the kernelized correlation filters (KCF), we call it EnKCF. A committee of KCFs is designed to specifically address the variations in scale and translation of moving objects. To guarantee a high-speed run-time performance, we deploy each of KCFs in turn, instead of applying multiple KCFs to each frame altogether. To reduce any potential drifts between individual KCFs' transition, we developed a particle filter. Experimental results showed that 1) the performance of ours is, on average, 70.10% for precision at 20 pixels, 53.00% for success rate for the OTB100 data, and 54.50% and 40.2% for the UAV123 data, and 2) our method is, on average, better than other high-speed trackers over 5% on precision on 20 pixels and 10-20% on AUC. In addition, our implementation ran at 340 fps for the OTB100 and at 416 fps for the UAV123 dataset that is faster than DCF (333 fps) for the OTB100 and KCF (296 fps) for the UAV123. To increase flexibility of the proposed EnKCF running on various platforms, we also explored different levels of deep convolutional features.

1. Introduction

A recent advancement of air/ground/water unmanned vehicle technologies has increased interests on deploying intelligent algorithms to existing mobile platforms. Among those technologies, computer vision algorithms are getting more attentions primarily because payloads of those mobile platforms are limited to carry any other sensors than a monocular camera. Instead of just manually being flew for video recording, an unmanned air vehicle (UAV) equipped with an object or feature following function would make it more useful for the application of monitoring/surveillance/surveying on private properties/wildlife/crop, video recording on various events, many others. To this end, in this paper, we propose a single-target tracking algorithm that does not require offline training and can run at high-speed. Specifically, we would like to have our algorithm 1) learn the appearance model of a target on the fly and 2) run on a typical desktop as fast as 300-450 fps.

One of the dominant frameworks for online object tracking is the correlation filter that essentially solves a single-target tracking problem as a regression problem in the frequency domain. This method assumes that a target location is given at the beginning like any other online tracking algorithms [22]. Given this positive example for the regression problem, a set of negative examples is collected around the initial target bounding box and represented as a form of the circulant matrix [12]. One can optimally solve this regression problem using a ridge regression in a closed form. However, this solution has to deal with expensive matrix operations $\mathcal{O}(n^3)$. The correlation filter offers a less complex solution, $\mathcal{O}(n \log n)$ by element-wise multiplication in a frequency domain [2, 12]. Thank to this reformulation, an object tracking pipeline based on the correlation filter can run very efficiently and be even easily implemented. In fact, an extension of a linear correlation filter, the kernelized correlation filter with multi-channel features [12] showed impressive object tracking results and outperformed other state-of-the-art, online tracking algorithms in terms of runtime and tracking accuracy. However, a vanilla form of such an online tracker is prone to drift, and fails to track a target over a long period of time [12]. This is primarily due to the dilemma of stability-plasticity in updating appearance model, where the appearance model will be overfitted to only the images used to train, unless a compromise on the frequency of updating the model is carefully implemented [20]. For example, one may handle a target's scale variation by just concatenating multiple correlation filters including KCF and running them on each frame. Alternatively one

978-1-5386-4886-5/18/$31.00 ©2018 IEEE
DOI 10.1109/WACV.2018.00129

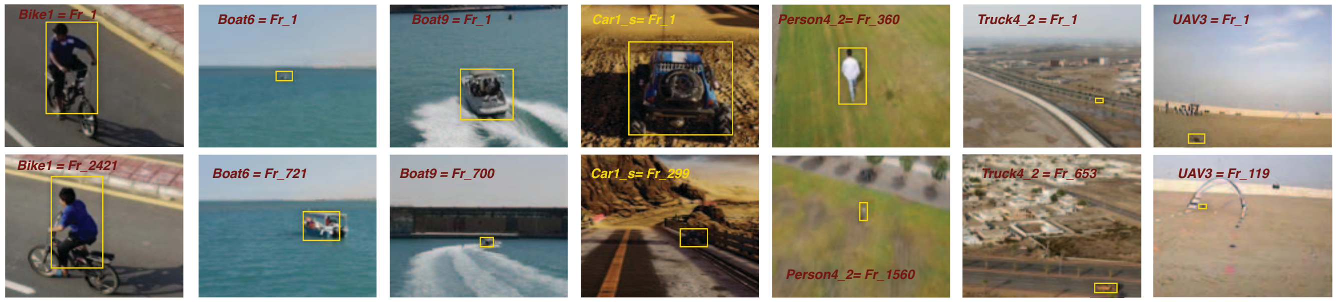


Figure 1: Examples of some tracking results (yellow rectangles) by the proposed method on the "UAV123" dataset. The "UAV123" dataset is challenging for object tracking as the scale and translation of a target can be drastically changed in a few consecutive frames.

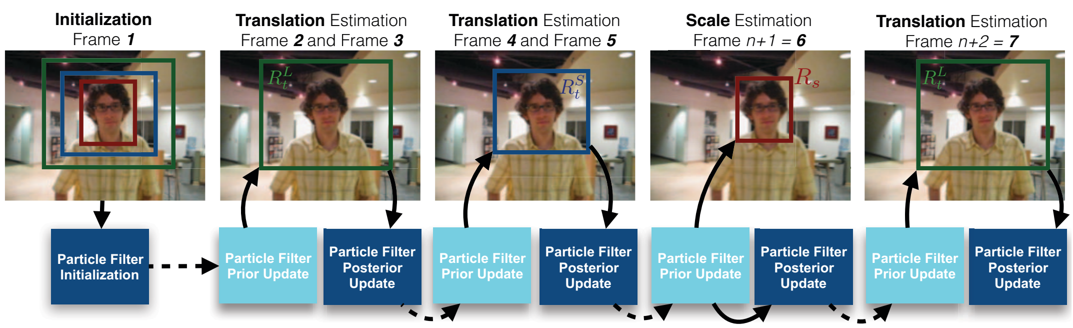


Figure 2: The workflow of the *E*nKCF. The first frame is used to initialize the tracking algorithm and a particle filter. For the next six frames, each of three KCF is deployed in turn to estimate the translation and scale change of a target. Afterward, the order of deploying three KCFs is repeating.

could think of scanning the region of interest (ROI) with a list of templates in predefined scale ratios to find the target in appropriate scale [1, 12, 16, 18, 23]. However, these approaches would drastically reduce run-time performance because multiple KCFs run on each frame.

Another way of handling the scale change for the correlation filter based approach is to estimate a correct scale at a location where a target highly likely appears [26] – estimate a target's translation first and then estimate correct scale. For example, Zhang and his colleagues used the MOSSE tracker [2] to estimate the target's translation. And then their method updated the scale of the target by further analyzing image sub-regions in high confidence. Their method is based on an assumption that the scale of a target would not change drastically over two consecutive frames. Similarly, Ma and his colleagues used two KCFs to learn the translation and scale of a target separately [18]. In particular, a KCF is used to learn the translation of the target and its background. Given this, another KCF is used to learn the target area to estimate the new scale of the target. However, because of running more than a KCF on each frame, this method degrades its run-time performance (i.e., $\leq 50fps$). Our method is motivated by this idea – the idea of deploying multiple KCFs to address the issues of single target tracking the drastic variations in scale and translation, but in a more efficient way. To maximize run-time performance and accuracy, instead of running them all together on every frame, we deploy three KCFs in turn: *target+small background* translation filter (R_t^S), *target-only* scale filter (R_s) and *target+large background* translation filter (R_t^L). By doing so, our method aims at effectively addressing scale change and estimating target's motion while maintaining run-time performance at high-speed. Figure 2 illustrates the workflow of the *E*nKCF.

The contribution of this paper is a novel, single-target tracking algorithm running at a very high-speed ($\geq$ 300 fps). In particular, to effectively address the changes in scale and translation of a target, we extended the KCF and deployed each of three KCFs in turn. Because of such a deployment strategy, the run-time performance of the proposed algorithm maintains high without sacrificing the accuracy of tracking. To reduce any potential drifts while switching KCFs, we developed a particle filter. Additionally, to increase the flexibility of the proposed algorithm's usage, we explore deep convolutional features with varying levels of abstraction.

2. *E*nKCF: Ensemble of Kernelized Correlation Filters

We propose a novel way of utilizing an ensemble of the KCFs [12] to effectively handle scale variations and dynamic maneuvers of a target. To improve the run-time performance and maintain the small-footprint of the original KCF (e.g., $\geq$ 300), we deploy three KCFs in turn, instead of applying them altogether on the same image frame. Each of these KCFs is designed to address the challenges of single-target tracking – variance in scale and translation.

The proposed algorithm, *E*nKCF, learns three KCFs in turn: The first filter, R_t^S, focuses on learning the target area and its background for addressing a marginal translation of a target, the second filter, R_s, focuses entirely on learning the target's scale, and the last filter, R_t^L, focuses on the target area and its background bigger than that of the first filter, R_t^S. By designing *E*nKCF as a composite of three KCFs, a transition filter with larger padding size, R_t^L, will enable *E*nKCF to recover from potential drifts after a scale filter, R_s, is applied to the input images. Another translation filter with smaller padding size, R_t^S, will help *E*nKCF to better localize the position of the target after the large ROI translation filter is applied. Our approach is similar to that of [18] which sequentially applied more than one KCF to every frame, but different in that we operate multiple KCFs in an alternating manner. It is intuitive to alternate multiple KCFs with different goals over a small temporal window of the input image sequences, because the appearance of a target does not change drastically over consecutive images. In other words, for most cases, learning a correlation filter over consecutive frames would not be substantially different from the one updated with smaller frequency. Figure 3 shows examples supporting this observation.

The algorithm 1 shows the pseudo-code of *E*nKCF. The order of running these three KCFs is important because each of these filters aims at addressing different challenges of single-target tracking. A translation filter, R_t^L, is applied to the ith and i+1th image frames, another translation filter, R_t^S, is applied to the i+2th and i+3th image frames, and then the scale filter, R_s, is applied to the i+4th image. This order repeats until the last image of an input video is presented. Note that the translation filter, R_t^L, is intended to run right after the scale filter, R_s, runs which is applied at every other i+4 frames. We run these filters in this way because we want to minimize any drifts that are likely to happen running only R_s. In addition, the filter, R_t^S, is applied to every other two frames before R_s and right after two consecutive frames running R_t^L. By repeating this order of learning three KCFs, we can integrate more discriminating shape features that cannot be learned just by R_t^L. The filter, R_t^L, uses shape and color information together to recover from any potential drifts – drifts could happen due to only scale filter operation in certain frames.

In summary, the scale filter, R_s, is designed to directly handle the scale change of a target and provides the translation filters, R_t^L and R_t^S, with more accurate ROIs. On the other hand, a translation filter, R_t^L, is devised to look into a larger search area to estimate the target's translation and recover from any potential drifts. Another translation filter, R_t^S, is designed to address the drawback of R_t^L that may learn noisy shape features due to its relatively larger search area. In what follows, we will brief the main idea behind the KCF.

Kernelized Correlation Filter The Kernelized Correlation Filter is a well-known single target tracking algorithm. As its workflow has been detailed in other papers [11, 12], in this section we briefly go over the parts of the KCF relevant to this study. Its computational efficiency is derived from the correlation filter framework representing training examples as a circulant matrix. The fact that a circulant matrix can be diagonalized by Discrete Fourier transform is the key to reduce the complexity of any tracking method based on correlation filter. The off-diagonal elements become zero whereas the diagonal elements represent the eigenvalues of the circulant matrix. The eigenvalues are equal to the DFT transformation of the base sample ($\boldsymbol{x}$) elements. The Kernelized Correlation Filter, in particular, applies a kernel to $\boldsymbol{x}$ to transform into a more discriminating domain. The circulant matrix is then formed by applying cyclic shifts on the kernelized $\boldsymbol{x}$. Such kernelization operation maintains $\mathcal{O}(nlog(n))$ complexity unlike other kernel algorithms leading to $\mathcal{O}(n^2)$ or even higher complexities.

The KCF solves essentially the problem of a regression in the form of the regularization (ridge regression):

$$E_h = \min_{\boldsymbol{h}} \frac{1}{2}||\boldsymbol{y} - \sum_{c=1}^{C} \boldsymbol{h_c} * \boldsymbol{x_c}||^2 + \frac{\lambda}{2}\sum_{c=1}^{C}||\boldsymbol{h_c}||^2 \quad (1)$$

where we seek for $\boldsymbol{h}$ that minimizes E given the desired continuous response, $\boldsymbol{y}$, and the training template $\boldsymbol{x}$. The parameter c enables one to integrate features in the form of multiple channels, such as HoG and color, in this setup [9, 12]. To simplify the closed-form solution for Equation 1, an element-wise multiplication in frequency domain was proposed to learn the frequency domain correspondence of $\boldsymbol{h}$, $\hat{\boldsymbol{w}}$:

$$\hat{\boldsymbol{w}} = \hat{\boldsymbol{x}}^{*} * \hat{\boldsymbol{y}}(\hat{\boldsymbol{x}}^{*} * \hat{\boldsymbol{x}} + \lambda)^{-1}. \quad (2)$$

Where $*$ is an element-wise multiplication. A non-linear version of $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{\alpha}}$ are proposed to increase robustness to any geometric and photometric variations [12]. In particular, the diagonalized non-linear Fourier domain dual form solution is expressed as

$$\hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{y}}(\hat{\boldsymbol{k}}^{\boldsymbol{xx}'} + \lambda)^{-1} \quad (3)$$

Algorithm 1: *E*nKCF Tracking Algorithm

Input: Initial bounding box (x_0, y_0, s_0), frame counter fc, complete cycle of scale filter $n = 5$,

Output: **if** $fc \,\%\, n = 0$ ***(Condition 1)*** **then**

  Estimated Target State (x_t, y_t, s_t), Scale filter (*target-only*) model R_s

else if $fc \,\%\, n > 0 \;\; and \;\; fc \,\%\, n \leq n/2$ ***(Condition 2)*** **then**

  Estimated Target State $(x_t, y_t, s_t = s_{t-1})$, Large Area Translation Filter model R_t^L

else ***Condition 3***

  Estimated Target State $(x_t, y_t, s_t = s_{t-1})$, Small Area Translation Filter model R_t^S

function track$(x_{t-1}, y_{t-1}, s_{t-1})$

  // Translation Estimation - Particle Filter

  Transit Particle Filter to the frame t and compute the mean of prior pdf (x_t, y_t, s_{t-1})

  // Translation Estimation - Correlation Filter

  Crop the ROI for the R_t^L (Condition 2), or R_t^S (Condition 3) given (x_t, y_t) and estimate new position as (x_t, y_t) = *max*$(\mathbf{y}_{\mathbf{R_t}})$

  // Scale Estimation - Correlation Filter

  Scale pool for R_s : $\mathbf{S} = \{1.05, 1.0, 1/1.05\}$,

  Crop the ROI for the R_s^i (Condition 1) and estimate scale factor, $\alpha = \underset{i \in S}{argmax}(PSR(\boldsymbol{y}_{\boldsymbol{R_s^i}}))$, and new scale $s_t = \alpha * s_{t-1}$,

  // Update Translation - Particle Filter

  Do Importance Re-sampling (if necessary) and compute the mean of posterior pdf (x_t, y_t)

  // Model Update

  Update R_t^S (Condition 3),

  Update R_t^L (Condition 2),

  if $PSR(\boldsymbol{y}_{\boldsymbol{R_s}}) \geq T_{R_s}$ **then**

    Update R_s (Condition 1)

  return (x_t, y_t, s_t)

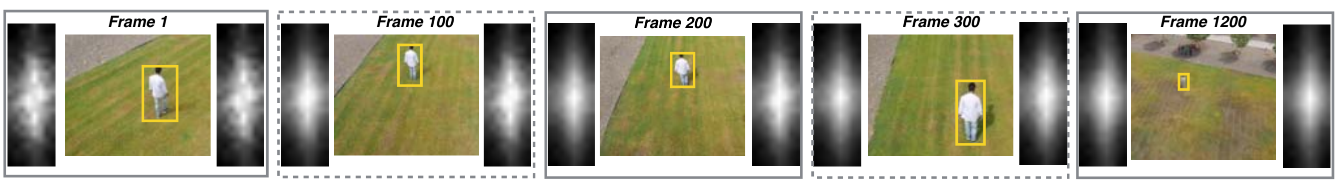


Figure 3: These examples show that there is a marginal difference between the scale filters learned at every frame and the one learned at every 5 frames. In each figure, the leftmost, sub-figures show the scale filters trained at every frame and those at the rightmost show the scale filters trained at every 5 frames.

where λ represents the regularization weight whereas $\hat{\boldsymbol{k}}^{\boldsymbol{x}\boldsymbol{x}'}$ denotes the first row of the kernel matrix $\boldsymbol{K}$ known as *gram matrix* and is expressed as

$$\boldsymbol{k}^{\boldsymbol{x}\boldsymbol{x}'} = exp(-\frac{1}{\alpha^2}(||\boldsymbol{x}||^2 + ||\boldsymbol{x}'||^2 - 2F^{-1}(\sum_{c}^{C} \hat{\boldsymbol{x}}_{\boldsymbol{c}}^{*} \odot \hat{\boldsymbol{x}}_{\boldsymbol{c}}'))). \tag{4}$$

An earlier version based on this formulation used grayscale feature ($C = 1$) to learn the solution vector $\boldsymbol{w}$ and integrating multi-channel features such as HoG and Color showed improved accuracy [1, 9, 12, 18, 23]. In the detection step, the learned correlation filter is correlated with the first row of the gram matrix, $\boldsymbol{k}^{\boldsymbol{x}\boldsymbol{z}'}$, which contains the similarity values between the learned feature template $\boldsymbol{x}$ and the new test template $\boldsymbol{z}$. This can be formulated as

$$\boldsymbol{r}(\boldsymbol{z}) = F^{-1}(\hat{\boldsymbol{k}}^{\boldsymbol{x}\boldsymbol{z}'} \odot \hat{\boldsymbol{\alpha}}) \tag{5}$$

where $\boldsymbol{r}$ denotes the correlation response at all cyclic shifts of the first row of the kernel matrix.

To integrate further temporal information into tracker, we update the correlation filter and the appearance model as below:

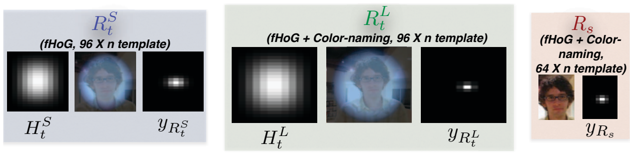
$$\hat{\boldsymbol{\alpha}}_{\boldsymbol{t}} = (1 - \beta)\hat{\boldsymbol{\alpha}}_{\boldsymbol{t-1}} + \beta\hat{\boldsymbol{\alpha}}_{\boldsymbol{t}} \tag{6}$$

$$\hat{\boldsymbol{x}}_{\boldsymbol{t}} = (1 - \beta)\hat{\boldsymbol{x}}_{\boldsymbol{t-1}} + \beta\hat{\boldsymbol{x}}_{\boldsymbol{t}} \tag{7}$$

where β represents the learning rate tuned to a small value in practice.

In the following sections, we will detail two different types of features used in *E*nKCF: hand-crafted features (fHoG + color-naming) and deep convolutional features.

***E*nKCF with Hand-Crafted Features** We, first, use two conventional hand-crafted features, fHoG (shape) [8] and color-naming (color) [16]. Figure 4 shows examples of these features. This setup of the *E*nKCF is designed to develop an online tracking algorithm running at high-speed ($\geq$ 300 fps) on a typical desktop (and eventually running ≥ 30 fps on a low-end embedded system without GPUs).



(a) Small area translation filter (b) Large area translation filter (c) Scale filter

Figure 4: Examples of three filters with hand-crafted features. The responses by the large and small area translation filters, and scale filter represented by $y_{R_t^L}$, $y_{R_t^S}$ and y_{R_s}.

We use both fHoG [8] and color-naming [24] for the large area translation filter, R_t^L. This is because the fHoG tends to be noisier and less discriminating, when it is applied to relatively larger area, and adding color information makes the feature vector for the translation filter more discriminating. Figure 4 (b) shows an example of this translation filter. By contrast, the R_t^S only uses the fHoG features because it covers a relatively smaller area. Figure 4 (a) shows an example of this translation filter. Lastly, we use both fHoG and color-naming features again for the scale filter, R_s. By assigning fHoG and color-naming features, we ensure that the likelihood of inaccurate scale estimation is decreased in comparison to the scale filter with only fHoG features. This is more important in our case as the scale filter is operated in every 5 frames. Also, the scale filter, explained earlier in the algorithm 1, estimates the scale of a target by correlating it with three candidate ROIs. This search may increase the run-time complexity of the scale filter from $\mathcal{O}(n \log n)$ to $\mathcal{O}(3(n \log n))$. To ensure practically feasible operations, we use a smaller template size (i.e., $64 \times n$) for the scale filter, R_s. The smaller template size does not degrade the tracking performance because of 1) zero padding around the target (smaller ROI) and 2) usage of color-naming features.

***E*nKCF with Deep Convolutional Features** In addition to hand-crafted features, we explore deep convolutional features to extend the applicability of the *E*nKCF and boost tracking performance.

There has been a large volume of studies on utilizing a pre-trained CNN features for tracking-by-detection algorithms. For example, [5] used the activation of the first and second convolutional layer of the VGGNet [21]. They reported that the first convolutional layer features lead to slightly better precision and success rates than the second layer due to increased invariance for translation in this layer. [17] proposed a KCF tracker integrating features with higher abstraction capacity. Their method, firstly used the third convolutional layer features to estimate the response map. In the next step, the KCF, concentrating on the second convolutional layer features, ran to update transition. The third KCF then worked on the transition given by the previous KCF, and learned the first convolutional layer features to update the transition. This coarse-to-fine translation estimation accommodates different levels of abstractions of the object. However, their method runs multiple KCFs in a sequential fashion, increasing the run-time complexity. For this reason, we follow an approach similar to [5], in order to embed deep features in *E*nKCF. The *E*nKCF algorithm enables us to exploit different level of feature encodings with different KCFs. The translation filters, R_t^L and R_t^S, consider at least twice bigger area than the scale filter, R_s. Given this, we can assign deeper feature encodings to R_s as it learns less spatial information than R_t^L and R_t^S. Thus we assign the activation of the fourth convolutional layer features ($26 \times 26 \times 128$) to R_s whereas R_t^L and R_t^S are assigned the activation of the second layer features ($109 \times 109 \times 64$). Figure 5 illustrates this feature assignment. Additionally, we assign the second convolutional layer features to R_s as in R_t^L and R_t^S and compare this setting (*conv222-VGG*) to *conv224-VGG*, where the *conv222* setting represents the assignment of the activation of 2^{th} convolutional layer features of VGGNet to R_t^L, R_t^S and R_s. We use the VGGNet since it provides higher spatial resolution at the first several layers than AlexNet [15].

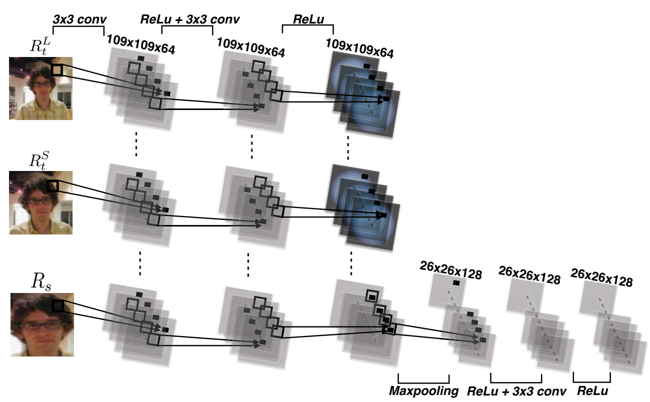


Figure 5: The proposed conv224-VGGNet feature extraction in Deep*E*nKCF. R_t^L and R_t^s used an hanning window to avoid distortion caused by FFT operation whereas R_s does not, in order to avoid target boundary information loss.

Particle Filter for Smoothing Transition among KCFs As explained earlier, the *E*nKCF updates the target's scale at every other k frames. Although the strategy of updating every other kth frames will result in an optimal run-time performance, this may lead drifts at the later frames. To prevent such potential drifts, we developed a Bayes filter that incorporates a target's motion to smooth any intermediate outputs from individual KCFs. In particular, we use a particle filter to estimate the target's image coordinate based on the *E*nKCF's outputs. The state in this paper, $\boldsymbol{X_t}$, represents the target's pixel coordinates and its velocity, $\boldsymbol{X_t} = \{x, y, v_x, v_y\}$, where x and y are the pixel coordinates of the target's centroid, v_x and v_y are the velocities estimated along the x-axis and y-axis. The particle filter predicts, using a motion model with constant acceleration assumption, the target's next state by generating a predefined number of particles. Then it uses the confidence maps of *E*nKCF as observation to update its state. The particle weight is computed as, $w_{p_t}(x_t, y_t) = \sum_{i=1}^{N}\sum_{j=1}^{M} \boldsymbol{y_R}(x_t - i, y_t - j)$, where w_p is the weight of the particle p at time t, $\boldsymbol{y_R}$ is the response map from one of the KCFs, and N and M denote the number of rows and columns of the confidence map.

3. Experiments

So far we explained how the proposed algorithm could efficiently perform a single-target tracking task while effectively addressing variations in scale and translation of moving objects. To accurately evaluate the performance of the proposed algorithm from the perspective of this paper's goal, we need data with challenging variations in scale and translation. To this end, we chose the UAV123 data[1][19] that contains 123 video sequences of objects captured from low-altitude UAVs. Because of the nature of the video acquisition – the targets and the camera are moving together, this data pose a great deal of challenges as the scale and position of the targets change drastically in most of the sequences. To verify that our algorithm is useful not only for the image data with such extreme challenges but also for the one with nominal difficulties in single-object tracking, we also used the OTB100 data[2] [25] that contains the videos recorded from smart phones and typical cameras in perspective view. In addition, we use the temporarily down-sampled UAV123 dataset to check how robust our algorithm is to drastic motions of camera and moving targets. Lastly, we evaluate the proposed algorithm with deep convolutional features on the UAV123 dataset to investigate how much performance gain we can achieve in using deep features over the conventional features like fHoG and color-naming.

Finding Optimal Hyper-parameters As each of three KCFs in *E*nKCF is designed to address specific challenges in single target tracking problem, the optimal parameters for individual KCFs should be different. We set the learning rates (β) of individual filters, R_t^L, R_t^S, and R_s as 0.020, 0.020 and 0.010. For the kernel of the Gaussian neighboring function, we empirically found the optimal values of α as 0.7, 0.6, and 0.9 for R_t^L, R_t^S, and R_s. We set the scale filter update threshold, T_{R_s}, to 4, peak-to-sidelobe (PSR) ratio. The padding size for the correlation filters is tuned to 2 for R_t^L, 1.50 for R_t^S, and 0 for R_s. For our particle filter implementation, we empirically found 1,000 for the number of particles as an optimal one for balancing the run-time performance and accuracy. To keep the level of variance among the particles reasonable, we performed the re-sampling only when the efficient number of samples, $\hat{N}_{eff} \approx (\sum_{p=1}^{P} \boldsymbol{w}_p^2)^{-1}$, is lower than a pre-defined threshold.

Performance on UAV123 Dataset We used the precision and success rates to compare the performance of the proposed algorithm with those of the state-of-the-art tracking algorithms. For the precision, we rank the trackers based on the precision numbers at 20 pixels whereas in the success rate plots, they are ranked by the area under curve (AUC) scores. The tracking algorithms under the comparison include ones at high-speed ($\geq$300 fps): KCF [12], CSK [11], DCF [12], MOSSE [2], and STC [26] and ones at relatively lower-speed ($\leq$50): ECO [4], CCOT [7], SCT [3], SAMF [16], DSST [6], Struck [10], MUSTER [13], TLD [14], and OAB [27]. The ECO and CCOT trackers originally use deep convolutional features, however, to perform fair comparison, we use fHoG and color-naming features similar to our tracker. Figure 6 shows the results on the UAV123 dataset. The *E*nKCF outperformed other high-speed trackers by 3%-15% at 20 pixels precision. In particular, three algorithms, SAMF, DSST, and Struck did about 5% better than ours in accuracy, but 10-20 times slower. The more recent trackers, ECO and CCOT, on the other hand, outperforms *E*nKCF by 15% while running at 10-15 times slower. For the scale adaptation, *E*nKCF did fourth best in terms of AUC for the success rate plot. It outperformed other high-speed trackers by about 20%-25% in AUC. In addition, it performed even better than some of the lower-speed trackers. For example, for AUC, *E*nKCF outperformed Struck and DSST by 5% and 10% while running at more than 10 and 30 times faster. For the DSST, we believe our algorithm outperformed it because, first of all, the DSST uses a 1-D scale filter and searches the target's scale over 33 candidates in a scale pool whereas our algorithm uses a 2-D scale filter with 3 candidate scales. Learning a 2-D filter resulted in learning more spatial information. Second of all, the DSST uses only fHoG whereas our algorithm uses both fHoG and color-naming features. Learning complementary features such as color and shape provided a better understanding of the ROI.

Finding Optimal Combination and Deployment Order of Multiple KCFs In addition to performance comparison with other trackers, we also conducted an experiment to see what combination and deployment order of the KCF works best. Table 1 presents results of this experiment. To achieve fair comparison, we did not use the particle filter. The combination presented at the 2^{nd} column switched the

[1] https://ivul.kaust.edu.sa/Pages/Dataset-UAV123.aspx

[2] http://cvlab.hanyang.ac.kr/tracker_benchmark/benchmark_v10.html

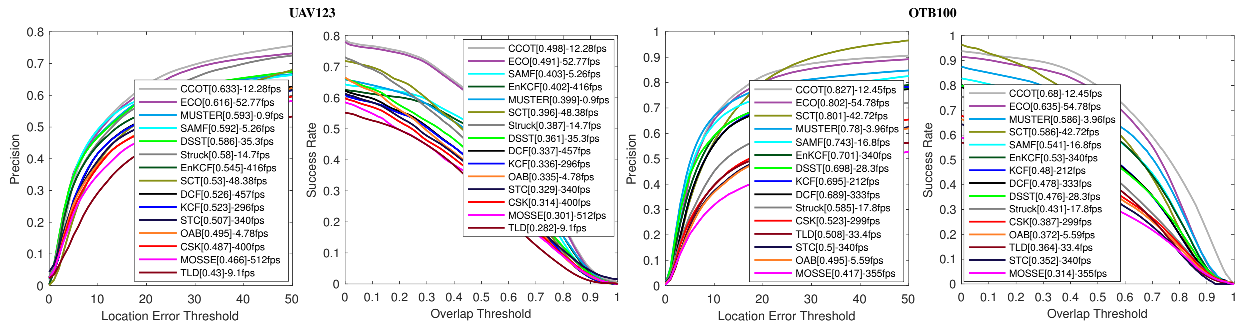


Figure 6: Comparison of *E*nKCF's performance with other trackers on the UAV123 and OTB100 datasets.

— Best — 2^{nd} Best — 3^{th} Best

Method	*E*nKCF	R_t^S+R_t^L+R_s	R_t^L+R_s	R_t^S+R_s	R_t^L*	R_t^S*	R_t^L+R_s*	R_t^S+R_s*	R_t^L + R_t^S+R_s*
Pr. (20px)	53.9	48.93	52.41	48.10	51.88	51.29	55.85	52.14	58.16
SR (50%)	40.2	36.75	38.23	36.04	35.12	34.43	39.89	38.51	41.58
FPS	416	412	370	425	365	384	135	151	99

Table 1: Results of running different combinations of the KCFs for UAV123 dataset. The '$*$' represents a sequential approach where multiple KCFs are, in the given order, applied to every frame like in LCT [18].

order of deploying R_t^L and R_t^S. The one at the 3^{th} column removed R_t^S and replaces with R_t^L whereas the one at the 4^{th} column removes R_t^L. The combinations with the "*" marker ran the filters on every frame in the listed order. The trackers at the 7^{th} and 8^{th} columns ran a translation and a scale filter on every frame which is similar to the LCT tracker [18].[3] This experiment empirically verified that the way *E*nKCF uses multiple KCFs is optimal, in terms of both tracking accuracy and run-time performance. For example, one could achieve 4.26% higher in accuracy by running R_t^L and R_t^S, and R_s at every frame (produced 58.16% for precision at 20 pixels), but this combination decreases run-time performance to 317 fps (=416fps - 99fps).

Evaluation on Particle Filter Contribution For the UAV123 dataset, we also evaluated the performance of the *E*nKCF with and without the particle filter. In particular, we use 50 video sequences with no drastic camera motion. To evaluate the robustness of particle filter, we add noise from a uniform distribution ($[-20, 20]$), to the translation estimations of the KCFs. For this experiment, the *E*nKCF with particle filter achieves 51.98% precision at 20 pixels, outperforming the one without particle filter by 6.32%. This experiment validated that the integration of particle filter into the *E*nKCF can further improve the performance.

Performance on OTB100 Dataset Figure 6 shows the results on the OTB100 dataset. The performance of *E*nKCF on the OTB100 data is similar to that of the UAV123 dataset. Specifically, it performed reasonably well at estimating target scale in that it showed the highest precision and success rates among the other high-speed trackers. Interestingly, *E*nKCF outperformed another correlation filter based tracker, DSST, but performing 5% behind of another low-speed scale adaptive SAMF tracker. Finally, the ECO, and CCOT achieve 15%-20% higher precision and success rates than ours while operating at 10-20 times slower rates.

Performance on UAV123_10fps Dataset We were curious about how the frame rate of a testing video would affect the performance of a tracking algorithm. To this end, we use the temporarily down-sampled version of the UAV123, called UAV123_10fps dataset. The downsampling of a given video sequence would obviously make the original UAV123 data more challenging because the displacements of the moving objects become bigger. To tackle this challenge, we slightly modified the proposed algorithm – ran R_t^L every frame and removed the particle filter due to larger motion. Figure 7 shows the performance of the modified version of *E*nKCF and other tracking methods. The precision rates of the ECO and CCOT dropped about 10% in comparison to the original UAV123 dataset. Our tracker outperforms other high-speed correlation filter based trackers including KCF, DSST by about 15%-20% and showed a precision rate similar to that of SAMF. It ranks as fourth in AUC while running about 10 to 100 times faster than the top three trackers. It was interesting to observe how sensitive the performance of the state-of-the-art tracking algorithms is to the frame rate, and, with a slight modification, the proposed method effectively handled large displacement of objects in successive frames.

[3] To be precise, the LCT tracker comes with a re-detection module for a long-term tracking.

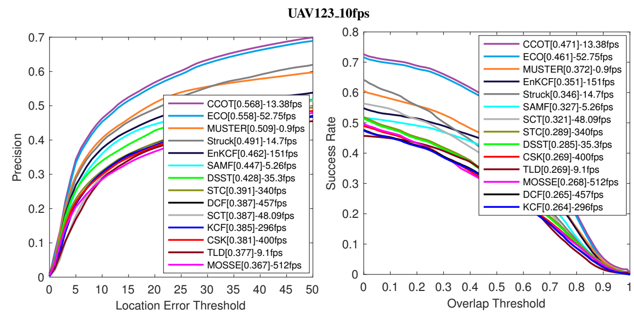


Figure 7: Comparison of a modified *E*nKCF's performance with those of the state-of-the-art tracking algorithms on a lower frame rate data.

Evaluation with Deep Features Our focus so far was to develop a single-target tracker that does not require offline learning to effectively address the variations in scale and translation, and can run at a high-speed, $\geq$ 300fps. We would like to see how much the performance gain we could achieve by replacing the conventional features, fHoG and color-naming with deep convolutional features. We use deep convolutional features similar to [5, 17]. Note that running the *E*nKCF with deep features would obviously increase the computational cost and degrade the run-time performance.

Figure 8 shows the DeepE*n*KCF's performance on UAV123 dataset. Deep*E*nKCF outperformed the *E*nKCF with hand-crafted features by about 3% to 5% in precision (20 px) and 2% in success rates (AUC). The *conv224-VGG* setting performed slightly better than the *conv222-VGG* in precision while achieving similar success rate (AUC). This indicates that higher level feature abstraction works better for the smaller ROIs. By using feature abstractions from the VGGNet pre-trained on millions of images, we can better represent the low level features of the object than hand-crafted features in challenging cases such as low contrast ROIs. However, we believe that the contribution of the deep features is limited by two factors: increased translational invariance in deeper layers and losing targets due to large target and camera motion in the UAV123 dataset.

4. Conclusion

Running a computer vision algorithm on any existing embedded systems for real-world applications is economically and practically very attractive. Among other practical considerations, it would be desirable if such computer vision algorithms require no offline training and operate at high-speed. To develop a single-target tracking algorithm to meet these properties, we proposed an extension of KCF that applies three KCFs, in turn, to address the variations in scale and translation of moving objects. We also developed a particle filter to smooth the transition between three KCFs, especially the transition between the scale and large ROI translation filter. Through experiments, we found that the way the *E*nKCF deployed three KCFs was optimal, and the particle filter contributed to increase the performance. We used two public datasets to evaluate the performance of the proposed algorithm and other state-of-the-art tracking algorithms, and found that, on average, the performance of the proposed algorithm is better than other high-speed trackers over 5% on precision at 20 pixels and 10-20% on AUC. Our implementation ran at 340 fps for OTB100 and at 416 fps for UAV123 data that is faster than DCF (333 fps) for OTB100 and KCF (296 fps) for UAV123. Finally, we explored the idea of utilizing deep features for the proposed algorithm and found that the deep features helped the proposed algorithm boost the performance by 5%.

Although the proposed algorithm showed a promising result for the challenging data, we believe the corner case analysis has not been extensively done yet. As future work, we would like to thoroughly study under what conditions our algorithm would fail to track objects. In addition, we also would like to investigate, for long-term tracking, a way of re-initializing a target when the target is lost, due to occlusion, illumination change, drastic camera motion, etc.

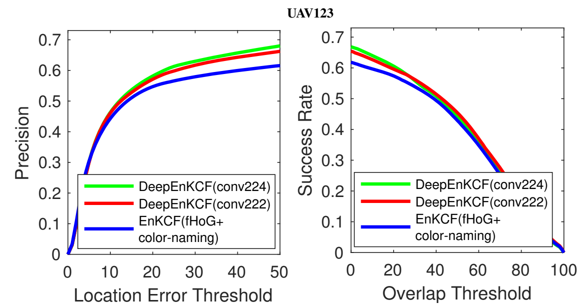


Figure 8: Comparison of *E*nKCF's (*fHoG* + *color-naming*) performance with Deep*E*nKCF on the UAV123 dataset. The frame rate of the Deep*E*nKCF (conv224) was, on average, 30.74 fps and the conv222 setting was 35.23 fps on a CPU.

References

[1] A. Bibi and B. Ghanem. Multi-template scale-adaptive kernelized correlation filters. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 50–57, 2015.

[2] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, 2010.

[3] J. Choi, H. J. Chang, J. Jeong, Y. Demiris, and J. Y. Choi. Visual tracking using attention-modulated disintegration and integration. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 4321–4330, 2016.

[4] M. Danelljan, G. Bhat, F. S. Khan, and M. Felsberg. ECO: Efficient Convolution Operators for Tracking. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2017.

[5] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Convolutional features for correlation filter based visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 58–66, 2015.

[6] M. Danelljan, G. Hger, F. S. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *Proceedings of the British Machine Vision Conference*, 2014.

[7] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg. Beyond correlation filters: learning continuous convolution operators for visual tracking. In *Proceedings of European Conference on Computer Vision*, pages 472–488, 2016.

[8] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[9] H. K. Galoogahi, T. Sim, and S. Lucey. Multi-channel correlation filters. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3072–3079, 2013.

[10] S. Hare, A. Saffari, and P. H. Torr. Efficient online structured output learning for keypoint-based object tracking. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1894–1901, 2012.

[11] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *Proceedings on European Conference on Computer Vision*, pages 702–715, 2012.

[12] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[13] Z. Hong, Z. Che, C. Wang, X. Mei, D. Prokhorov, and D. Tao. Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 749–758, 2015.

[14] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, 2012.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[16] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *Proceedings of European Conference on Computer Vision*, pages 254–265, 2014.

[17] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3074–3082, 2015.

[18] C. Ma, X. Yang, C. Zhang, and M.-H. Yang. Long-term correlation tracking. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 5388–5396, 2015.

[19] M. Mueller, N. Smith, and B. Ghanem. A benchmark and simulator for uav tracking. In *Proceedings of European Conference on Computer Vision*, pages 445–461, 2016.

[20] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. Prost: Parallel robust online simple tracking. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 723–730. IEEE, 2010.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[22] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: an experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, 2014.

[23] M. Tang and J. Feng. Multi-kernel correlation filter for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3038–3046, 2015.

[24] J. van De Weijer, C. Schmid, J. Verbeek, and D. Larlus. Learning color names for real-world applications. *IEEE Transactions on Image Processing*, 18(7):1512–1523, 2009.

[25] Y. Wu, J. Lim, and M.-H. Yang. Online Object Tracking: A Benchmark. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2411–2418, 2013.

[26] K. Zhang, L. Zhang, Q. Liu, D. Zhang, and M.-H. Yang. Fast visual tracking via dense spatio-temporal context learning. In *Proceedings of the European Conference on Computer Vision*, pages 127–141, 2014.

[27] T. Zhang, B. Ghanem, S. Liu, and N. Ahuja. Robust visual tracking via multi-task sparse learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2042–2049, 2012.