Abstract

In recent years, correlation filter based trackers (CF trackers) have received much attention because of their top performance. Most CF trackers, however, suffer from low frame-per-second (fps) in pursuit of higher localization accuracy by relaxing the boundary effect or exploiting the high-dimensional deep features. In order to achieve real-time tracking speed while maintaining high localization accuracy, in this paper, we propose a novel CF tracker, fd-KCF*, which casts aside the popular acceleration tool, i.e., fast Fourier transform, employed by all existing CF trackers, and exploits the inherent high-overlap among real (i.e., noncyclic) and dense samples to efficiently construct the kernel matrix. Our fdKCF* enjoys the following three advantages. (i) It is efficiently trained in kernel space and spatial domain without the boundary effect. (ii) Its fps is almost independent of the number of feature channels. Therefore, it is almost real-time, i.e., 24 fps on OTB-2015, even though the high-dimensional deep features are employed. (iii) Its localization accuracy is state-of-the-art. Extensive experiments on four public benchmarks, OTB-2013, OTB-2015, VOT2016, and VOT2017, show that the proposed fdKCF* achieves the state-of-the-art localization performance with remarkably faster speed than C-COT and ECO.

1. Introduction

Visual object tracking is one of the fundamental problems in computer vision with many applications. In the model free tracking problem, the goal is to estimate the states (e.g., position and size) of the target in a whole image sequence only with the initial frame [45, 46]. Model free tracking is very challenging because the tracker has to learn the robust appearance model from a very limited training samples to resist extremely challenging interference, such as occlusions, large appearance changes, illumination variation, fast motion, and background clutters. In general, the key problem of model free tracking is how to construct a tracker which can not only tolerate appearance variation of target, but also exclude background interference, while maintaining the processing speed as fast as possible.

In recent years, correlation filter based trackers (CF trackers) have received much attention because of their top performances. Since MOSSE [3], almost all CF trackers [22, 39, 18, 28, 10, 1, 12, 13, 14, 24, 8, 31] have been relying on fast Fourier transform (FFT) to accelerate their computations. Unfortunately, while modern CF trackers’ localization accuracies continue to improve, their fps’ become lower and lower. We believe that the following two reasons cause this phenomenon. On the one hand, to improve the robustness of tracking algorithm itself, some representative CF trackers [12, 24] exploit techniques to relax the boundary effect which is introduced by FFT [3]. These techniques, however, inevitably destroy entire cyclicity of the training samples, resulting in much slower training speed. On the other hand, to improve the robustness of features to
appearance variations of targets, deep features which are always high-dimensional are employed by modern CF trackers [30, 14, 8, 11, 37, 5]. Although these trackers benefit from deep features remarkably, their computational costs increase significantly. Particularly, C-COT [14] which not only relaxes the boundary effect but also employs deep features can run at only 0.3 fps on GPU. Further, despite lots of techniques have been employed to accelerate its computation, ECO [8] can run at only 6 fps. Naturally, it is asked whether or not we can design a CF tracker which is able to efficiently relax or even avoid the boundary effect inherently, i.e., does not employ FFT to accelerate its training, and efficiently exploit deep features at the same time?

To solve the above problem, in this paper, we propose a novel CF tracker, fdKCF* which not only has not the boundary effect inherently, but also can run in real-time even though deep features are employed. First, we introduce the real and dense sampling method to avoid the boundary effect radically. As shown in Fig. 1, this sampling method is based on the traditional sliding window where all training samples are real, and it is different from the cyclic shift based sampling method used in existing CF trackers, such as KCF [22] and BACF [24], where training set contains virtual samples, resulting in the negative boundary effect. Second, we design a novel algorithm, fCKM, to construct the kernel matrix in spatial domain efficiently even if the high-dimensional deep features are employed after investigating the inherent high-overlap among real and dense samples. Finally, a Gauss-Seidel based iterative method is employed to efficiently optimize in dual space.

It is observed that there exist vast redundant computations in the construction of kernel matrix by using the brute-force approach because of the high-overlap among real and dense samples. Take the linear kernel as an example. Given the $H \times W \times C$ feature maps of learning region and detection region where $C$ is the number of channels, we have to calculate the dot product of any two $C$-dimensional feature vectors, which come from the two feature maps respectively. $K$ times where $K$ is the number of pairs of samples which contain the above two feature vectors respectively. Indeed, this dot product need to be calculated only once. Fig. 2 shows an example. Inspired by this observation, we propose a novel algorithm, fCKM, which can construct the linear kernel matrix efficiently by eliminating redundant calculations, i.e., the dot product of any two $C$-dimensional feature vectors is only calculated once, instead of $K$ times. fCKM conducts the following two steps to construct the matrix of linear kernel: (i) building the dot product value table of any two $C$-dimensional feature vectors which come from the input two feature maps respectively; (ii) obtaining each element in the linear kernel matrix by looking up the table with summing. This way, the redundant calculations of dot product of any two $C$-dimensional feature vectors can be replaced with looking up table with time complexity $O(1)$ rather than $O(C)$ in brute-force approach. Consequently, fCKM enjoys the following two advantages: (i) It is performed in spatial domain without the boundary effect. (ii) Its running speed is fast and insensitive to the number of feature channels. In our experiments, only a few milliseconds are taken to construct the matrix of linear kernel even though the number of feature channels up to 1024. Additionally, it can also be employed to construct many matrices of typical non-linear kernels with very little increase in time-consuming by modifying the first step and adding non-linear mapping after the second step.

Experiments are performed on four public benchmarks: OTB-2013, OTB-2015, VOT2016, and VOT2017. Our fdKCF* achieves the state-of-the-art localization performance, while running at 24 fps. As a fair comparison, when C-COT [14] and fdKCF* employ deep features of the same dimensional, run on same GPU, and do not use any other acceleration techniques, the localization accuracy of fdKCF* is higher than that of C-COT, while the mean fps of fdKCF* is about 80 times that of C-COT. To the best of our knowledge, our fdKCF* is the first CF tracker which achieves both high localization accuracy and real-time speed.

2. Related Work

For the first time, KCF [21, 22] establishes the relationship between correlation filters and ridge regression. Compared to MOSSE, KCF is modeled in kernel or dual space and it can make use of multi-channel features without an increase in parameters. Besides, another important contribution of KCF is the fast calculation of kernel matrix in frequency domain. In order to improve the localization performance of KCF, HCF [30] introduces the higher dimensional deep features into KCF; however, the boundary effect become a bottleneck for its localization performance. Our
fdKCF* is also modeled in kernel or dual space. It is, however, mainly different from KCF and HCF in two aspects. First, samples of our fdKCF* are real (i.e., noncyclic) and dense ones, rather than the cyclic shifts of a real base sample as in KCF and HCF (See Fig. 1). In other words, there is no boundary effect inherently in our fdKCF*, whereas KCF and HCF suffer from it. Therefore, the localization performance of our fdKCF* exceeds HCF with a large margin (about 10 percent on OTB-2013 and OTB-2015). Second, our fdKCF* is accelerated in the spatial domain by exploiting the inherent high-overlap among real and dense samples, rather than in frequency domain as done in KCF and HCF. It is worth noting that compared to HCF, the tracking speed of our fdKCF* is much faster than its (24fps vs. 11fps on GPU) even though the search region of fdKCF* is larger than HCF (4 vs. 1.8 times target size) and fdKCF* does not exploit the cyclic samples structure to accelerate.

By exploiting the real and dense samples, LSART [37] solves for the dual variables of the KCF with linear kernel through propagating messages forward and backward in a network. It does not construct the kernel matrix explicitly. Different from LSART, our fdKCF* constructs the kernel matrix first, and then solve for the dual variables by an iterative method. The efficiency of our fdKCF* is remarkably higher than that of LSART for the following two reasons. First, in order to solve for the dual variables, every update for them requires to propagate messages forward and backward in the network of LSART, and this is time-consuming. Whereas in our fdKCF*, we only construct the kernel matrix once by our fCKM, then solve for the dual variables by an iterative method. Both steps are efficient. Second, only the first-order convergence method like SGD can be employed in LSART, while more efficient ones such as Gaussian-Seidel can be used in our fdKCF*. It is worth noting that LSART can only employ the linear kernel, whereas nonlinear kernels can also be employed in our fdKCF*.

In addition, Siamese networks based trackers [2, 27, 20] achieved state-of-the-art performance in recent years. They treat tracking as a similarity learning problem and train their models by vast offline data. For completeness, we also compare our fdKCF* with typical ones in the experiment.

3. KCF without Boundary Effect

We will start with the kernel ridge regression problem, and suggest readers referring to [22, 44] for the relation between the ridge regression and the kernel ridge regression.

Let \( \mathbf{X} \in \mathbb{R}^{H \times W \times C} \) and \( \mathbf{Z} \in \mathbb{R}^{H \times W \times C} \) be the feature maps of learning region and search region, respectively, where \( H \) and \( W \) are the height and width of the feature maps, respectively, \( C \) is the number of channels. All training samples \( \{ \mathbf{x}_i \}_{i=1}^N \) where \( \mathbf{x}_i \in \mathbb{R}^{H \times W \times C} \) are sampled from \( \mathbf{X} \), as shown in Fig.3, and all test samples \( \{ \mathbf{z}_i \}_{i=1}^N \) where \( \mathbf{z}_i \in \mathbb{R}^{H \times W \times C} \) are sampled from \( \mathbf{Z} \) in the

![Figure 3: Sampling in feature map \( \mathbf{X} \) (green cuboid). Samples \( \{ \mathbf{x}_i \mid i = 1, \ldots, N \} \) are obtained by using real and dense sampling method (see Fig. 1) where \( \hat{W} = W - w + 1 \) and \( N = (H - h + 1) \times (W - w + 1) \). See Sec.3 for details.](image)

It is clear that in order to calculate \( \mathbf{y} = \mathbf{K}_{\mathbf{XX}} \alpha + \lambda \mathbf{I} \mathbf{y} \), the process of detection in \( \mathbf{Z} \) can be expressed as

\[
\mathbf{y} = \mathbf{K}_{\mathbf{XX}} \alpha + \lambda \mathbf{I} \mathbf{y}
\]

Further, given \( \mathbf{X} \) and \( \alpha^* \), the process of detection in \( \mathbf{Z} \) can be expressed as

\[
\mathbf{f} (\mathbf{Z}) = \mathbf{K}_{\mathbf{ZX}} \alpha^*.
\]

It is clear that in order to calculate \( \alpha^* \) in Eq.(3), we have to construct \( \mathbf{K}_{\mathbf{XX}} \) with \( \{ \mathbf{x}_i \}_{i=1}^N \) first. Constructing \( \mathbf{K}_{\mathbf{XX}} \), however, is extremely time-consuming when dense samples and deep features are employed where \( N \) and \( C \) are generally large. For example, when \( \mathbf{K}_{\mathbf{XX}} \) is constructed, its each element \( \kappa (\mathbf{x}_i, \mathbf{x}_j) \) has to be calculated by the formula

\[
\kappa (\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} \sum_{d=0}^{C-1} \mathbf{X}_{[i/\hat{W}]+m,(i \mod \hat{W})+n,d} \cdot \mathbf{X}_{[j/\hat{W}]+m,(j \mod \hat{W})+n,d}
\]

4022
where \( \tilde{W} = W - w + 1 \) and \( X_{p,q,d} \) is the element of \( X \) at \( p \)-th row, \( q \)-th column and \( d \)-th channel. Therefore, the time complexity of constructing \( K_{\psi,\phi} \) with Eq.(5) is \( O(N^3ChW) \), and so is \( K_{ZX} \). Suppose \( H = \beta h, H = W, h = w, \) and replace \( N \) with \( (H-h+1) \times (W-w+1) \). Then, the above complexity can be simplified to \( O(C\beta^4h^6) \). It is noted that this complexity is extremely high because \( h \) and \( C \) often belong to 10 and \( 10^5 \) orders of magnitude, respectively, when deep features are employed.

4. Fast Calculation of Kernel Matrix (fCKM)

In this section, we first introduce our novel algorithm fCKM for efficient construction of \( K_{ZX} \), then show how it works with linear kernel and Gaussian kernel as two special cases. Finally, the analysis of complexities of fCKM and its comparison with the brute-force approach is presented.

4.1. fCKM for General Kernels

Our fCKM can construct the kernel matrix efficiently where kernel \( \kappa(\cdot, \cdot) \) can be expressed as

\[
\kappa(z, x) = \psi\left( \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} \sum_{d=0}^{C-1} \phi(z_{m,n,d}, x_{m,n,d}) \right),
\]

where \( \psi(\cdot) \) and \( \phi(\cdot, \cdot) \) are two functions with time complexities \( O(\gamma) \) and \( O(\eta) \), respectively, \( z \in R^{h \times w \times C} \) and \( z_{m,n,d} \) is the element of \( z \) at \( m \)-th row, \( n \)-th column and \( d \)-th channel. We call such \( \kappa(\cdot, \cdot) \) as \( (\psi, \phi) \) kernel.

It can be observed from Fig. 2 and Fig. 3 that most elements of any sample are also elements of its spatially adjacent ones for real and dense samples. Such large shared elements will lead to large redundant computations in constructing \( K_{ZX} \) of \( (\psi, \phi) \) kernel with brute-force approach, i.e., calculating \( \kappa(z_i, x_j) \)'s with Eq.(6). This is because there are \( K \) pairs of samples, \((z, x)\)'s, which contain \((z_{m,n,s}, X_{i,j,s})\) and the relative position of \( z_{m,n,s} \) in \( z \) is the same as that of \( X_{i,j,s} \) in \( x \), where \( K \in [1, h \times w] \) and \( z_{m,n,s} \) is the \( C \)-dimensional feature vector at the \( m \)-th row and \( n \)-th column of \( Z \), leading to \( \sum_{d=0}^{C-1} \phi(z_{m,n,d}, X_{i,j,d}) \) has to calculate \( K \) times. In order to reduce these redundant computations, we design a novel algorithm fCKM to construct \( K_{ZX} \) of \( (\psi, \phi) \) kernel efficiently. Our fCKM consists of the following three steps.

(1) Building Base Table. The base table \( T \in R^{HW \times HW} \) of \( (\psi, \phi) \) kernel is constructed with expression:

\[
T(i,j) = \sum_{d=0}^{C-1} \phi(z_{|i/W|, j \mod W, d}, X_{i/[W], i \mod W, d}).
\]

Consequently, \( T \) contains \( \sum_{d=0}^{C-1} \phi(z_{m,n,d}, X_{i,j,d}) \) for all \((m, n, i, j)\), and any one is calculated only once. The time complexity of this step is \( O(\eta C \beta^4 h^4 + C \beta^4 h^4) \).

(2) Constructing Summation Matrix. The summation matrix \( S \in R^{N \times N} \) is constructed through looking up the base table \( T \) and summing. Specifically,

\[
S(i,j) = \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} T(p, q),
\]

where

\[
p = (|i/\tilde{W}| + m) \times W + (i \mod \tilde{W}) + n,
\]

\[
q = (|j/\tilde{W}| + m) \times W + (j \mod \tilde{W}) + n,
\]

\[
\tilde{W} = W - w + 1.
\]

Consequently, \( S(i,j) = \psi^{-1}(\kappa(z_i, x_j)) \) for all \((i, j)\). The time complexity of this step is \( O(\beta^4 h^6) \).

(3) Mapping. \( K_{ZX} \in R^{N \times N} \) of \( (\psi, \phi) \) kernel can be obtained by mapping \( S \) with function \( \psi(\cdot) \). Specifically,

\[
K_{ZX}(i,j) = \psi(S(i,j)).
\]

Consequently, \( K_{ZX}(i,j) = \kappa(z_i, x_j) \) for all \((i, j)\). The time complexity of this step is \( O(\beta^4 h^4) \).

According to above steps, \( \sum_{d=0}^{C-1} \phi(z_{m,n,d}, X_{i,j,d}) \) for all \((m, n, i, j)\) is calculated only once, rather than \( K \) times, in constructing \( K_{ZX} \) of \( (\psi, \phi) \) kernel, resulting in the high efficiency of our fCKM.

Last, we would like to discuss the key difference between KII [23] and our fCKM. In short, KII needs to validate whether the contribution function satisfies the necessary and sufficient condition, and it can only accelerate the filtering of a single filter, rather than a group of highly overlapping filters. Whereas, our fCKM focus on accelerating the filtering of such a group of filters.

4.2. fCKM for Linear Kernel Matrix

\( (\psi, \phi) \) kernel is linear if \( \psi(x) = x \) and \( \phi(x, y) = xy \). Therefore, the linear kernel matrix \( K^L_{ZX} \) can be constructed efficiently with iCKM. Specifically, according to Sec.4.1, \( K^L_{ZX} \) can be constructed through the following two steps.

(1) Building the base table \( T^L \) as follows.

\[
T^L(i,j) = \sum_{d=0}^{C-1} Z_{[i/W], j \mod W, d} \times X_{i/[W], i \mod W, d}.
\]

(2) Constructing the summation matrix \( S^L \) with Eq.(8), where \( T \) is replaced with \( T^L \) in Eq. 11. Finally, \( K^L_{ZX} = S^L \) because \( \psi(x) = x \).

4.3. fCKM for Gaussian Kernel Matrix

Not only the linear kernel, but also many commonly used non-linear kernels, such as Gaussian, multi-quadric, and sigmoid ones, are \( (\psi, \phi) \) kernel. As the most commonly used one in CF trackers, Gaussian kernel is used to show how fCKM works to construct the Gaussian kernel matrix.

4023
(ψ, φ) kernel is Gaussian if $\psi(x) = \exp\left(-\frac{x^2}{\sigma^2}\right)$ and $\phi(x, y) = (x - y)^2$. Therefore, the Gaussian kernel matrix $K_{ZX}^G$ can be constructed efficiently with fCKM. Specifically, according to Sec.4.1, $K_{ZX}^G$ can be constructed through the following three steps.

1. Building the base table $T^G$ as follows.

$$T^G(i, j) = \sum_{d=0}^{C-1} (Z_{[j/W], j \text{ mod } d} - X_{[i/W], i \text{ mod } d})^2.$$  

2. Constructing the summation matrix $S^G$ with Eq.(8), where $T$ is replaced with $T^G$ in Eq.12.

3. Mapping $S^G$ to $K_{ZX}^G$ with $\psi(x) = \exp\left(-\sqrt{\frac{S^G(i, j)}{\sigma^2}}\right)$, i.e.,

$$K_{ZX}^G(i, j) = \exp\left(-\sqrt{\frac{S^G(i, j)}{\sigma^2}}\right).$$  

### 4.4. Complexity Analysis

In this section, we analyze and compare the time and space complexities of our fCKM against those of the brute-force approach in detail when the linear kernel is employed.

According to Sec. 4.1, the time complexities of the brute-force approach with Eq.(6) and our fCKM in constructing the kernel matrix $K_{ZX}$ of (ψ, φ) kernel are $O(\eta C^3 h^6 + C^3 h^6 + \gamma^4 h^2)$ and $O(\eta C^3 h^4 + C^4 h^4 + \beta^4 h^6 + \gamma^4 h^4)$, respectively. Therefore, $K_{ZX}^G$ can be constructed in the time complexities $O(C^3 h^6)$ and $O(C^4 h^4 + \beta^4 h^6)$ by the brute-force approach with Eq.(5) and our fCKM, respectively, because the time complexity of $\phi(\cdot, \cdot)$ and $\psi(\cdot)$ are $O(1)$ and $O(h)$, respectively, i.e., $\eta = 1$ and $\gamma = 0$. Further, their proportional relation is

$$\frac{C^3 h^6}{C^4 h^4 + \beta^4 h^6} = \frac{Ch^2}{C + h^2} = \frac{h^2}{1 + h^2/C}$$

In practice, when the high-dimensional deep features are employed, $C > h^2 \gg h \gg 1$. Therefore, the time complexities of constructing $K_{ZX}^G$ by brute-force approach is about $h^2$ times that of our fCKM.

Fig. 4 shows the effect of our fCKM on the reduction of computational costs compared to the brute-force approach with common $H = W = 60$ and $h = w = 15$ which are the case if the cell size (stride) of features is $4$ and $4$. It can be concluded from the figure that, when the number of channels increases, the increment of FLOPs with fCKM is much slower than that with the brute-force approach, and the more the number of channels is, the more the acceleration of fCKM is. Therefore, our fCKM can construct $K_{ZX}^G$ efficiently even though the high-dimensional deep features are exploited. For example, on a TITAN X GPU, when typically $H = W = 60$ and $h = w = 15$, the execution time of the Step (1) only increases $8\mu$s with $C$ increasing 1, and it always takes $4\mu$s to perform Step (2) whatever $C$ is. Therefore, when $H = W = 60$, $h = w = 15$, and $C = 600$, the time-consuming of our fCKM is about $8.8\mu$s, whereas that of brute-force approach is about $600\mu$s.

On the other hand, fCKM has $O(H^2 W^2)$ higher space complexity than the brute-force approach. This requirement of extra space is used to store the base table $T$. However, it is negligible on the current GPU and RAM. For example, fCKM only requires $50$MB more than the brute-force approach does under $H = W = 60$ and float data type.

### 5. Fast-deepKCF without Boundary Effect

#### 5.1. Fast Training

In Sec. 3, we have shown how to efficiently construct $K_{X_i}^t$ by our fCKM, where $X_i$ is the $X$ in frame $t$. Achieving optimal $\alpha^*_t$ by directly using Eq.(3), however, is time-consuming, because the time complexity of matrix inversion is $O(N^3)$. Even though $\alpha^*_t$ is achieved through solving a system of linear equations with Gauss elimination method, the complexity is still $O(\frac{1}{2} N^3)$. In order to achieve $\alpha^*_t$ more efficiently, we adopt the iterative approach [12] based on the Gauss-Seidel. Specifically, we decompose $K_{X_iX_i}^t + \lambda I$ into a lower triangular $L_t$ and a strictly upper triangular $U_t$, i.e., $K_{X_iX_i}^t + \lambda I = L_t + U_t$. Then, $\alpha^*_t$ can be efficiently solved by the following iterative expressions:

$$\alpha^*_t(j) \leftarrow \alpha^*_t(j-1)$$

$$\alpha^*_t(j) \leftarrow L_t \backslash \left( y - U_t \alpha^*_t(j-1) \right)$$

where $j$ indicates the number of iterations, and $\alpha^*_t$ is the optimal solution at frame $t-1$. In practice, 5 iterations are enough for the satisfactory solution $\alpha^*_t$. Note that this iterative method is efficient because Eq. (15b) can be solved efficiently with forward substitution, and the time complexity of each iterations is $O(N^2)$.

As a comparison to KCF, it is easy to know that the time complexities of constructing kernel matrix and solving linear system are $O(C^3 h^6 \log(\beta h))$ and $O(N)$, respectively, in KCF, being significantly lower than those of our
fdKCF\*. This is because KCF exploits circulant samples which cause the negative boundary effect.

5.2. Update

To robustly locating the target object, updating the appearance model of a tracker is often necessary. Similar to other CF trackers [18, 38, 22], we update $X_t$ in Sec. 5.1 by means of the following linear weighting approach, i.e.,

$$
X_1 = \tilde{X}_1,
$$

$$
X_t = (1 - \delta) X_{t-1} + \delta \tilde{X}_t, \quad t > 1,
$$

where $\tilde{X}_t$ is the actual sampled feature map of learning region in frame $t$, and $\delta$ is the learning rate.

5.3. Fast Multi-scale Detection

Scale-pyramid based method [28] is employed to locate the target object and estimate its proper scale simultaneously. Specifically, given $X$, $\alpha^*$, and the scale-pyramid $\{Z^i\}_{i=1}^S$ of feature maps of detection regions, where $S$ is the level of the scale-pyramid, the fast detection of target object on each scale can be expressed as

$$
f(Z^i) = K_{Z^i} X \alpha^*, \quad \forall i.
$$

Note that any $(\psi, \phi)$ kernel can be used in Eq.(17).

Particularly, if $(\psi, \phi)$ kernel is linear, the optimal scale as well as the target location can be achieved with a more efficient approach. That is,

$$
w = \sum_{i=1}^N \alpha^*(i)x_i, \quad (18a)
$$

$$
\hat{f}(Z^i) = \hat{Z}^i \odot \hat{w}^*, \quad \forall i, \quad (18b)
$$

where $\alpha^*(i)$ is the $i$-th element of $\alpha^*$ and $\hat{\bullet}$ denotes the DFT of $\bullet$ in $H \times W$ dimension. Before conducting DFT for $\bullet$, 0’s are padded to $\bullet$’s bottom right to make its dimensionality be $H \times W$ if the dimensionality of $\bullet$ is less than $H \times W$. Eq.(18) is more efficient than Eq.(17) in the case of multi-scale detection with linear $(\psi, \phi)$ kernel because Eq.(18a) is executed only once.

6. Experiments

We evaluate our fdKCF\* on four public benchmarks, OTB-2013 [45], OTB-2015 [46], VOT2016 [26] and VOT2017 [25], and compare its performance with the state-of-the-art and representative trackers. All parameters of fdKCF\* are kept consistent in all experimental comparisons.

6.1. Implementation Details

Platform. Our fCKM is implemented in C++, and the rest of our fdKCF\* is implemented in PyTorch [33]. Experiments are performed on Linux with single TITAN X GPU.

Features. Similar to C-COT [14], our fdKCF\* only employs deep features to show performances of the algorithm itself. Specifically, we adopt the VGG-M-BN [4] trained on ImageNet [16] for feature extraction. We first change original strides of Conv-2 and Conv-5 from 2 to 1 to improve the localization accuracy. Then, the output maps of Conv-1 followed by an average pooling layer with kernel size $2 \times 2$ and stride $2 \times 2$ are employed as shallow level features (96 channels), and the output maps of Conv-5 followed by a $2 \times 2$ bilinear interpolation layer are employed as deep level features (512 channels). As a result, both shallow level features and deep level features are of $4 \times 4$ cell size (stride).

Parameters. We set different learning rates (Sec. 5.2) for shallow level features and deep level features. Specifically, $\delta = \delta_s = 0.01$ for shallow ones and $\delta = \delta_d = 0.005$ for deep ones. The regularization parameter $\lambda$ in Eq. 2 is set to 0.01. The maximum number of iterations $j$ in Eq. 15 is set to 5. Similarly to SRDCF [12], we set the image area of the square sampling region to $4^2$ times the target area, and it is re-scaled to the area of $200^2$ if its area is less than $200^2$, and to the area of $240^2$ if its area is greater than $240^2$.

Scaling. In order to balance the localization accuracy and tracking speed, we set 5 levels scale-pyramid (Sec. 5.3).

Kernel. We only employ the linear kernel in fdKCF\* in our current experiments. The reasons are (1) most of state-of-the-art CF trackers, such as BACF [18] and ECO [8], can only employ the linear kernel, (2) fdKCF\* with linear kernel runs slightly faster than that with Gaussian kernel.

6.2. Evaluation on OTB datasets

In our OTB-2013 and OTB-2015 experiments, we compare our fdKCF\* with state-of-the-art CF trackers and non-CF trackers, respectively. When comparing with CF trackers, following the standard benchmark protocols in the OTB-2015 [46], all trackers are quantitatively evaluated by five metrics, namely precision plot, success plot, distance precision (DP), overlap precision (OP), and AUC. In addition, as pointed out in [32], the definition of DP in OTB-2015 is defective because it is sensitive to the size of bounding boxes, and they propose the normalized precision, $P_{norm}$, to measure the localization accuracy. Based on their work, we evaluate all trackers with $P_{norm} \geq 0.2$ which is computed as the percentage of frames in a video where $P_{norm}$ is smaller than 0.2. When comparing with non-CF trackers, all trackers are quantitatively evaluated by AUC metric because they all reported AUCs in their original papers and there is no way to obtain the detailed tracking results of some of them to evaluate them with other metrics.

Comparison with CF trackers. We divide state-of-the-art CF trackers into two groups for a thorough comparison. The first group consists of seven trackers which can run at real-time speed, i.e. beyond 20 fps. These trackers are MKCFup [39], BACF [18], ECO-HC [8], LCT [31], Sta-
Figure 5: The mean precision and success plots of our fdKCF* and (a) seven modern real-time CF trackers, (b) seven modern CF trackers that produce state-of-the-art localization accuracy, on OTB-2013 and OTB-2015, respectively. The mean distance precisions and AUCs are reported in the legends. fdKCF* outperforms all other real-time CF trackers by large margins.

Table 1: The mean OP (mOP) and mean $P_{norm}@0.2$ (mPN) of our fdKCF* and (a) seven modern real-time CF trackers, (b) seven modern CF trackers that produce state-of-the-art localization accuracy, on OTB-2013 and OTB-2015. dec and deep are denoted SRDCFdecon and deepSRDCF, respectively. The best three results on each line are shown in red, blue and magenta, respectively. fdKCF* outperforms all other real-time CF trackers by large margins. In addition, we also report the mean fps’ (mFPS) of trackers in (b) on OTB-2015, and fdKCF* is visibly faster than them.

Figure 6: The mean precision and success plots of our fdKCF* and seven modern CF trackers that produce state-of-the-art localization accuracy on the very hard sequences [19] of OTB-2015. The mean distance precisions and AUCs are reported in the legends. In addition, as shown in Table 1a, our fdKCF* obtains the mean $P_{norm}@0.2$ and mean OP of 84.6% and 88.4% on OTB-2013, as well as 82.0% and 82.8% on OTB-2015, respectively, leading the second best trackers with a significant gain of 3.0%, 4.3%, 4.9% and 4.6%, respectively. These results show that our fdKCF* significantly outperforms all other state-of-the-art real-time CF trackers in localization performance. We believe that this is because our efficient fdKCF* not only avoids the boundary effect inherently, but also utilizes the powerful deep features.
all trackers in the second group (including ECO), although there is not sparse update and feature dimension reduction, which are employed by ECO to accelerate, in fdKCF*

We believe that the following three reasons cause the localization performance of our fdKCF* to be slightly worse than that of ECO. (1) The clustering algorithm GMM employed by ECO improves its robust, whereas there is no similar component in our fdKCF*. (2) The self-adaptive dimensionality reduction and weighting different features are used in ECO, whereas our fdKCF* does not use the similar components. (3) Both deep features and hand-crafted features, i.e., HOG [7] and Color-Names [15], are employed in ECO, whereas our fdKCF* only employs deep features. In fact, hand-crafted features can improve the localization performance of trackers on the sequences with easy challenges. To illustrate this fact, Han, et al. [19] divide the sequences of OTB-2015 into three sets: easy, hard, and very hard, according to the localization performance of most state-of-the-art trackers. It is not hard to find that trackers employed hand-crafted features always achieve high localization performances on the easy sequences. To further illustrate the above point, we show the comparison of our fdKCF* with the second group of CF trackers on the very hard sequences in Fig. 6. It is seen that the localization performance of our fdKCF* is obviously better than that of all other CF trackers (including ECO) on the very hard set. Note that C-COT, which mainly use deep features, also outperforms other trackers except fdKCF*. It is concluded that employing hand-crafted features may not improve but weaken the localization performance of trackers on very hard sequences.

Comparison with non-CF trackers. We compare fdKCF* with state-of-the-art non-CF trackers, including SINT+ [40], SINT++ [43], RASNet [42], SASiam [20], SiamRPN [27], DaSiamRPN [49], StruckSiam [47], CREST [35], DSLT [29], DAT [34], PTAV [17], TRACA [6], FlowTrack [50], LSART [37], and VI-

<table>
<thead>
<tr>
<th></th>
<th>AUC-2013</th>
<th>AUC-2015</th>
<th>Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINT+</td>
<td>0.655</td>
<td>0.571</td>
<td>no</td>
</tr>
<tr>
<td>SINT++</td>
<td>0.624</td>
<td>0.574</td>
<td>no</td>
</tr>
<tr>
<td>RASNet</td>
<td>0.670</td>
<td>0.642</td>
<td>no</td>
</tr>
<tr>
<td>SASiam</td>
<td>0.677</td>
<td>0.657</td>
<td>yes</td>
</tr>
<tr>
<td>SiamRPN</td>
<td>0.658</td>
<td>0.636</td>
<td>yes</td>
</tr>
<tr>
<td>DaSiamRPN</td>
<td>0.655</td>
<td>0.658</td>
<td>yes</td>
</tr>
<tr>
<td>StruckSiam</td>
<td>0.638</td>
<td>0.621</td>
<td>yes</td>
</tr>
<tr>
<td>CREST</td>
<td>0.673</td>
<td>0.623</td>
<td>no</td>
</tr>
<tr>
<td>DSLT</td>
<td>0.683</td>
<td>0.660</td>
<td>no</td>
</tr>
<tr>
<td>DAT</td>
<td>0.704</td>
<td>0.698</td>
<td>no</td>
</tr>
<tr>
<td>PTAV</td>
<td>0.663</td>
<td>0.625</td>
<td>yes</td>
</tr>
<tr>
<td>TRACA</td>
<td>0.652</td>
<td>0.603</td>
<td>yes</td>
</tr>
<tr>
<td>FlowTrack</td>
<td>0.689</td>
<td>0.655</td>
<td>no</td>
</tr>
<tr>
<td>LSART</td>
<td>0.701</td>
<td>0.682</td>
<td>no</td>
</tr>
<tr>
<td>fdKCF*</td>
<td>0.705</td>
<td>0.675</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 2: The AUCs of our fdKCF* and other state-of-the-art non-CF trackers on OTB-2013 and OTB-2015. The best two results are shown in red and blue, respectively.

6.3. Evaluation on VOT datasets

We present the evaluation results on VOT2016 [26] and VOT2017 [25] datasets which contain 60 sequences, respectively. We follow the VOT challenge protocol to compare trackers, where mainly reports the expected average overlap (EAO) and rank trackers based on it.

Fig. 7 shows the EAO ranking plots where we compare our fdKCF* against the top-15 CF trackers on VOT2016 and VOT2017, respectively. The performances of these trackers come from the VOT report. On the whole, the EAO score of our fdKCF* is competitive to that of C-COT, which is the winner of VOT2016 challenge, and slightly worse than that of ECO. However, the tracking speed of our fdKCF* is visibly faster than those of C-COT and ECO. These conclusions are consistent with those obtained on the OTB datasets. Note that CFWCR and CFCF are improved versions based on ECO and C-COT, respectively. On the contrary, our fdKCF* is a novel tracking framework without any further improvements and tricks.

7. Conclusions and Future work

A novel CF tracker, fdKCF*, with the state-of-the-art localization accuracy and real-time speed is proposed in this paper. fdKCF* achieves the state-of-the-art accuracy because there is no boundary effect with it and powerful deep features are also employed. fdKCF* is able to run at real-time speed because a novel acceleration method, fCKM, is developed in spatial domain. Through exploiting the inherent high-overlap among real and dense samples, fCKM is able to construct the kernel matrix efficiently even though the high-dimensional deep features are employed. Future work can be found in the supplementary material.

Acknowledgement. This work was supported by National Natural Science Foundation of China under Grants 61772527, 61806200, and 61702510.
References


