Parallel Tracking and Verifying

Heng Fan and Haibin Ling

***Abstract*—Visual object tracking has played a crucial role in computer vision with many applications. Being intensively studied in recent decades, visual tracking has witnessed great advances in either speed (e.g., with correlation filters) or accuracy (e.g., with deep features). Real-time and high accuracy tracking algorithms, nevertheless, remain scarce. In this paper we study the problem from a new perspective and present a novel *parallel tracking and verifying* (PTAV) framework, by taking advantage of the ubiquity of multi-thread techniques and borrowing ideas from the success of *parallel tracking and mapping* in visual SLAM. The proposed PTAV framework typically consists of two components, a (base) tracker $\mathcal{T}$ and a verifier $\mathcal{V}$, working in parallel on two separate threads. The tracker $\mathcal{T}$ aims at providing a super real-time tracking inference and is expected to perform well most of the time; by contrast, the verifier $\mathcal{V}$ validates the tracking results and corrects $\mathcal{T}$ when needed. The key innovation is that, $\mathcal{V}$ does not work on every frame but only upon the requests from $\mathcal{T}$; on the other end, $\mathcal{T}$ may adjust the tracking according to the feedback from $\mathcal{V}$. With such collaboration, PTAV enjoys both high efficiency provided by $\mathcal{T}$ and strong discriminative power by $\mathcal{V}$. Meanwhile, in order to adapt $\mathcal{V}$ to object appearance changes, we maintain a dynamic target template pool for adaptive verification, resulting in further improvement. In our extensive experiments on OTB2015, TC128, UAV20L and VOT2016, PTAV achieves top tracking accuracy among all real-time trackers, and in fact even outperforms many deep learning based algorithms. Moreover, as a general framework, PTAV is very flexible with great potentials for future improvement and generalization.**

***Index Terms*—Visual tracking, deep learning, correlation filter, verification, multi-thread, parallel tracking and verifying.**

I. Introduction

As one of the most important components in computer vision, visual object tracking has a long list of applications including robotics, intelligent vehicles, visual surveillance, human-computer interaction and so forth [1]–[4]. Given an initial state (usually a bounding box) of a tracking target in the first frame, visual tracking aims at estimating the unknown states (e.g., position and scale) of the target object in subsequent consecutive frames. Although significant advances have been made in recent decades, robust visual object tracking still remains challenging because of large appearance variations caused by many factors such as object occlusion, deformation, illumination variations, scale changes, motion blur and so on.

Recently, inspired by the successes in image classification and recognition (e.g., [5]), deep convolutional neural networks (CNNs) have been leveraged for visual tracking owing to their power in feature representation, and achieved state-of-the-art performance (e.g., [6]–[16]). Despite significant improvements in accuracy, these algorithms often suffer from high computational burden due to either extracting expensive deep features (e.g., [6], [7], [12]–[15]) or online network fine-tuning (e.g., [8]–[11]), and hardly meet the real-time requirement (see Figure 1 for illustration).

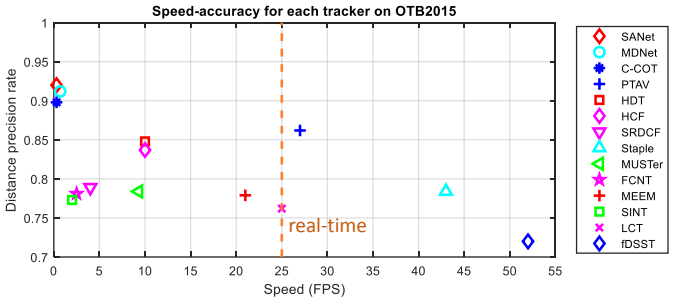


Fig. 1. Speed-accuracy plot of state-of-the-art trackers on OTB2015 [26]. For better illustration, only those trackers with accuracy higher than 0.7 are reported. Compared with high precision deep learning-based trackers (e.g., MDNet, SANet and C-COT) whose speeds are around 1 *fps*, our PTAV runs in real-time without serious accuracy degradation. On the other hand, compared with other real-time trackers (e.g., Staple, LCT and fDSST), PTAV achieves a much higher accuracy. Moreover, PTAV even outperforms some deep learning-based trackers (e.g., HCF, HDT, SINT and FCNT).

In order to develop real-time trackers, researchers have been focusing on the correlation filter for visual tracking (e.g., [17]–[22]). Owing to the highly computational efficiency of correlation filters in the Fourier domain, these trackers can easily achieve super real-time tracking inference using simple hand-crafted features such as raw pixel, HoG [23] and color names [24]. While running efficiently, these trackers usually perform less robustly compared to deep learning-based approaches because of the inferior feature representation (see again Figure 1).

Despite aforementioned progresses in either speed or accuracy, real-time and high accuracy tracking algorithms remain scarce. A natural way is to seek a trade-off between speed and accuracy (e.g., [20], [25]). In this paper we work toward this goal, but from a novel perspective described as the following.

Motivation. Our key idea is to decompose the original tracking task into two parallel but collaborative ones, one for fast tracking and the other for accurate verification. We are mainly inspired by the following observations or related works:

Motivation 1: When tracking a target from visual input, most of the time the target object moves smoothly and its appearance changes slowly or remains the same. Simple but efficient algorithms usually work fine for such easy cases. By contrast, hard cases (e.g., drastic object appearance variations) happen only occasionally, though they can cause serious consequences if not addressed properly. These hard cases usually require computationally expensive processes or analysis, such as the verification in our approach. Intuitively, verifications are needed only occasionally instead of for every frame. Figure 2

H. Fan and H. Ling are with the Department of Computer & Information Science, Temple University, Philadelphia, USA (e-mail: hengfan@temple.edu; hbling@temple.edu).

Fig. 2. Illustration of verification scores on a typical sequence. Verifier validates tracking results every 10 frames. Most of the time the tracking results are reliable (showing in blue). Occasionally, e.g., frame #080, the verifier finds the original tracking result (showing in blue) unreliable and the tracker is corrected and resumes tracking based on detection result provided by verifier (showing in red).

shows a typical example with both cases. In the example video, a simple tracker can efficiently locate the target object most of the time until its appearance drastically changes in frame #080. For this situation, the computationally expensive verifier is employed to detect the tracking failure and return a corrected result. The tracker is then adjusted and resumes tracking with the feedback provided by verifier.

Motivation 2: The ubiquity of multi-thread computing has already benefited computer vision systems, with a notable example as visual SLAM (*simultaneous localization and mapping*). By splitting tracking and mapping into two parallel threads, PTAM (*parallel tracking and mapping*) [27] provides one of the most popular SLAM frameworks with many important extensions (e.g., ORB-SLAM [28]). A key observation in PTAM is that mapping is not needed for every frame because of extensive redundancies existing in a video. Nor does verifying in our tracking task since the target appearance changes slowly most of the time in a video.

Motivation 3: Last but not least, recent advances in either fast or accurate tracking algorithms provide promising building blocks and highly encourage us to seek a practical system for real-time high accuracy visual tracking.

Contribution. With the motivations listed above, we propose a novel framework towards real-time tracking with high accuracy, named *parallel tracking and verifying* (PTAV). PTAV typically consists of a fast tracker[1] denoted by $\mathcal{T}$ and an accurate verifier denoted by $\mathcal{V}$. The two components work in parallel on two separate threads while collaborating with each other. The tracker $\mathcal{T}$ aims at providing a super real-time tracking inference and is expected to perform well most of the time, e.g., most frames in Figure 2. By contrast, the verifier $\mathcal{V}$ validates tracking results and corrects $\mathcal{T}$ when needed, e.g., at frame #080 in Figure 2. By running $\mathcal{T}$ and $\mathcal{V}$ in parallel, PTAV inherits both the high efficiency of $\mathcal{T}$ and the strong discriminative power of $\mathcal{V}$. Figure 3 illustrates the framework of PTAV. With this framework, we implement a tracking solution by combining correlation filter-based tracking (the Staple algorithm [20]) and deep learning-based verification (the Siamese network [29]). Extensive experiments on four large-scale benchmarks, including OTB2015 [26], TC128 [30], UAV20L [31] and VOT2016 [32], demonstrate that the proposed PTAV algorithm achieves promising performance among all real-time trackers, and in fact even outperforms many deep learning-based solutions.

[1]For conciseness, in the rest of this paper, we refer the *fast tracker* as a *tracker*, whenever no confusion caused.

This paper is an extended version of a preliminary conference publication [33]. The main new contributions or differences include: (1) a more robust base tracker (i.e., Staple [20]) for $\mathcal{T}$ in implementing PTAV, which brings clear performance improvement, (2) a dynamic target template pool for adaptive verification against target appearance variations, (3) various ablation studies on $\mathcal{V}$ and $\mathcal{T}$ to analyze PTAV, including two base verifiers (VGGNet [34] and AlexNet [5]) for $\mathcal{V}$ and three base trackers (KCF [17], fDSST [18] and Staple [20]) for $\mathcal{T}$, and (4) more thorough experimental validation and analysis involving more trackers and benchmarks.

II. Related Work

Visual tracking algorithms. Visual tracking has been extensively studied and it is beyond our scope to review all previous studies. Instead, we only sample some representative works and discuss those closely related to ours. Some comprehensive reviews on visual tracking can be found in [1]–[4].

This paper focuses on *model-free* single object tracking, for which existing algorithms are often categorized as either discriminative or generative. Discriminative algorithms usually treat tracking as a classification problem that distinguishes the target from changing background. Various approaches are proposed to learn the classifier such as multiple instance learning (MIL) [35], compressive sensing [36], semi-supervised boosting [37], kernelized structured output support vector machine (SVM) [38] and so forth. By contrast, generative algorithms usually formulate tracking as searching for regions most similar to the target. To this end, numerous object appearance modeling approaches have been proposed, including incremental subspace learning [39], sparse principal component analysis (SPCA) [40] and sparse representation [41]–[44].

Deep learning-based tracking. Motivated by the power of deep features in visual recognition (e.g., [5], [34]), some trackers utilize deep features for object appearance modeling, and achieve excellent performance, though typically at the cost of low running speed. Wang *et al.* [10] introduce a stacked denoising autoencoder to learn generic image features for visual tracking. In [14], Wang *et al.* present a fully convolutional neural network tracking (FCNT) algorithm by transferring pre-trained CNN features to improve tracking accuracy. Ma *et al.* [6] replace HoG [23] with discriminative convolutional features for correlation filter tracking, resulting in remarkable performance gains. A similar idea is presented by Qi *et al.* [7] by merging convolutional features from different layers. In order to address the problem of lack of training samples,

Wang *et al.* [9] employ intermediate features of networks to learn a robust ensemble tracker. In [8], Nam *et al.* propose to impose multiple domain branches on a light architecture of CNNs to learn generic feature for tracking target, and then introduce an online tracking algorithm by updating network weights. In [11], Fan *et al.* introduce the recurrent neural networks (RNNs) to capture internal structure of a tracking target and the generated tracking algorithm achieves promising results on several tracking benchmarks. Though these methods have achieved very impressive results, the heavy computation burden severely limits their practical applications.

Correlation filter-based tracking. Recently, correlation filter has drawn increasing attention in visual tracking owing partly to its high computation efficiency. Bolme *et al.* [19] propose to use correlation filter for tracking through learning the minimum output sum of squared error (MOSSE). Benefiting from the high computation efficiency of correlation filter, this approach runs amazingly at hundreds of frames per second. Henriques *et al.* [21] incorporate kernel space into correlation filter and propose a circulant structure with kernel (CSK) method for visual object tracking, and later [17] extends CSK to the well-known kernelized correlation filters (KCF) tracker by substituting raw pixel intensities with HoG [23] for appearance representation. To handle the scale issue, [18] and [45] suggest an extra scale filter into correlation filter tracking to adaptively estimate target scale. Later, more efforts have been made to improve correlation filter tracking, such as improving feature representation with color attributes [22] or deep feature [6], [12], [46], adopting part-based strategy [47], [48] or an additional detector [49] to resist occlusion, decreasing the boundary effect of correlation filter [50] and combining correlation filter with a complementary tracker [20].

Verification in tracking. The idea of verification is not new for tracking. A notable example is the tracking-learning-detection (TLD) algorithm [51], in which tracking results are validated *per frame* to decide how learning/detection shall progress. Similar ideas have been explored in other trackers such as [49], [52]. Unlike in previous studies, the verification in PTAV runs only on sampled frames. This mechanism allows PTAV to use strong verification algorithms without worrying much about running time efficiency. In fact, we utilize the Siamese network [29] that is designed for verification tasks.

Interestingly, tracking by itself can be also formulated as a verification problem that finds the best candidate similar to the tracking target [13], [25]. Bertinetto *et al.* [25] propose a fully-convolutional Siamese network for visual tracking by searching the tracking target within a local region. Tao *et al.* [13] formulate tracking problem as a task of object matching in each frame and develop a matching function based on the Siamese network. Despite obtaining excellent performance, the application of such trackers is limited by the heavy computation for extracting deep features in each frame. Compared with these studies, our solution treats verification only as a way to validate and correct the *fast tracker*, and does not run verification per frame.

Ensemble tracking. In order to achieve robustness in tracking, a natural solution is to leverage multiple different components to determine tracking result. For example, Kwon *et al.* [40] combine multiple observation and motion models to handle large appearance changes in tracking. Yoon *et al.* [53] propose an adaptive tracker by selecting reliable ones from multiple trackers. The TLD [51] tracker combines a tracker and a detector to achieve long-term tracking, and a similar idea is adopted in [49]. In [54], Wang *et al.* propose a probabilistic model for tracking by jointly learning the reliability of each tracker. Zhang *et al.* [55] propose to store the intermediate statuses for robust tracking. The method of [56] combines a stable template based model and an adaptive motion based model for tracking. In [20], Bertinetto *et al.* propose a complementary tracking approach based on correlation filters and color histograms. Hong *et al.* [57] present a multi-store tracker (MUSTer) which contains short- and long-term memory stores to process target appearance. The final result is jointly determined by short- and long-term memories. Different from these studies, PTAV consists of two components, which run on two parallel threads asynchronously while collaborating with each other.

Though other ensemble approaches (e.g., [20], [40], [49], [53]–[57]) can be implemented using multiple threads as well, the proposed PTAV fundamentally differs from them. In these existing algorithms, different components are simultaneously used to determine tracking result in each frame, thus their multi-thread implementations are *synchronous*. By contrast, in PTAV, $\mathcal{T}$ and $\mathcal{V}$ function in different ways and run independently except for necessary interactions, and thus the multi-thread implementation is *asynchronous*.

III. Parallel Tracking and Verifying (PTAV)

A. Framework

A typical implementation of PTAV consists of two components: a (fast) tracker $\mathcal{T}$ and a (reliable) verifier $\mathcal{V}$. The two components work together toward real-time and high accuracy tracking.

- **The tracker** $\mathcal{T}$ is responsible of the **real-time** requirement of PTAV, and needs to locate the target in each frame. Meanwhile, $\mathcal{T}$ sends verification request to $\mathcal{V}$ from time to time (though not every frame), and responds to feedback from $\mathcal{V}$ by adjusting tracking or updating models. To avoid heavy computation, $\mathcal{T}$ maintains a buffer of tracking information (e.g., intermediate status) in recent frames to facilitate fast tracing back when needed.
- **The verifier** $\mathcal{V}$ is employed to pursue the **high accuracy** requirement of PTAV. Up on receiving a request from $\mathcal{T}$, $\mathcal{V}$ tries the best to first validate the tracking result (e.g., comparing it with the template), and then provide feedback to $\mathcal{T}$. To adapt $\mathcal{V}$ to object appearance variations over time, the tracking target template is *not* fixed. Instead, $\mathcal{V}$ collects a number of reliable tracking results, and then uses k-means to cluster these results to obtain a target template pool for subsequent verification.

In PTAV, $\mathcal{T}$ and $\mathcal{V}$ run in parallel on two different threads with necessary interactions, as illustrated in Figure 3. The tracker $\mathcal{T}$ and verifier $\mathcal{V}$ are initialized in the first frame. After that, $\mathcal{T}$ starts to process each arriving frame and generates the

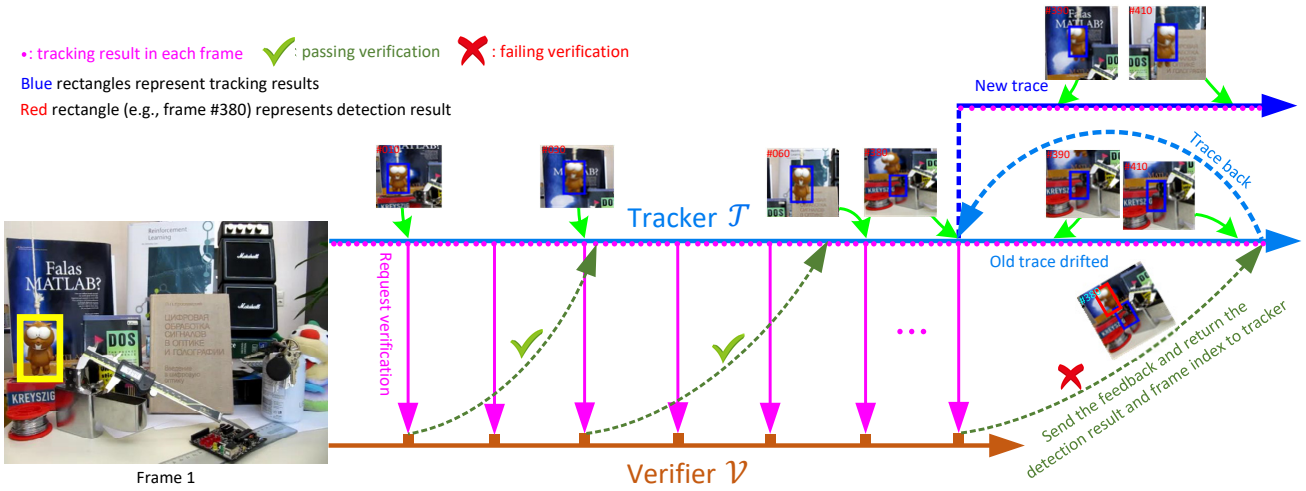


Fig. 3. Illustration of the PTAV framework in which tracking and verifying are processed asynchronously in two parallel threads. On receiving feedback (including detection result and frame index) from the verifier $\mathcal{V}$, the tracker $\mathcal{T}$ will trace back to the frame where failure occurs using frame index and leverage the target detection result to resume tracking at the position where tracker fails.

Algorithm 1: Parallel Tracking and Verifying (PTAV)

Initialize the tracking thread for tracker $\mathcal{T}$;
Initialize the verifying thread for verifier $\mathcal{V}$;
Initialize *current_frame* as the second frame;
Run $\mathcal{T}$ (Alg. 2) and $\mathcal{V}$ (Alg. 3) till the end of tracking;

result (pink solid dot in Figure 3). In the meantime, $\mathcal{V}$ validates the tracking result every several frames. Because tracking is much faster than verifying, $\mathcal{T}$ and $\mathcal{V}$ work asynchronously. Such mechanism allows PTAV to tolerate temporary tracking drift (e.g., at frame 380 in Figure 3), which will be corrected later by $\mathcal{V}$. When $\mathcal{V}$ finds a tracking result unreliable, it searches the correct answer in a local region and sends it to $\mathcal{T}$. Upon the receipt of such feedback, $\mathcal{T}$ stops current tracking job and traces back to resume tracking with the correction provided by $\mathcal{V}$.

Notably, PTAV is a very flexible framework, and some important designing choices are following. (1) The base algorithms for $\mathcal{T}$ and $\mathcal{V}$ may depend on specific applications and available computational resources. In addition, in practice one may use more than one verifiers or even base trackers. (2) The response of $\mathcal{T}$ to the feedback from $\mathcal{V}$, either positive or negative, can be largely designed to adjust to specific requests. (3) The correction of unreliable tracking results can be implemented in various ways, and it can even be conducted purely by $\mathcal{T}$ (i.e., including target detection). (4) $\mathcal{T}$ has numerous methods to use pre-computed and archived information for speeding up. Algorithms 1-3 summarize the general PTAV framework. It is worth noting that the whole system forms a loop during tracking since the tracker $\mathcal{T}$ may trace back and then resume tracking. However, the verifier provides new correct results for the tracker to resume, and therefore the system will not repeat the same loop and typically move forward with better results. Owing to the high efficiency of $\mathcal{T}$, the whole system still runs efficiently. Details about the tracing back process is illustrated in Section III B.

Algorithm 2: Tracking Thread $\mathcal{T}$

while *current_frame is valid* **do**
  if *received a message from* $\mathcal{V}$ **then**
    if *verification passed* **then**
      Update tracking model (optional);
    else
      Correct tracking;
      Trace back and reset *current_frame*;
    end
  end
  Tracking on the *current_frame*;
  if *time for verification according to* N_{int} **then**
    Send the current result to $\mathcal{V}$ to verify;
  end
  current_frame $\leftarrow$ next frame;
end

Algorithm 3: Verifying Thread $\mathcal{V}$

while *not ended* **do**
  if *received request from* $\mathcal{T}$ **then**
    Verifying the tracking result;
    Collect tracking result and perform k-means clustering if needed;
    if *verification failed* **then**
      Provide correction information in s;
      Adjust N_{int} if needed;
    end
    Send verification result s to $\mathcal{T}$;
  end
end

B. PTAV Implementation

1) **Tracking:** We choose the Staple tracker [20] for $\mathcal{T}$ in PTAV. The main idea of Staple is to combine two complementary cues, i.e., template and histogram, for tracking. To such

end, given an image patch $\mathbf{z}$, a linear combination of tracking scores from template and histogram is proposed

$$y(\mathbf{z}) = (1-\alpha)y_{\text{tmpl}}(\mathbf{z}) + \alpha y_{\text{hist}}(\mathbf{z}) \quad (1)$$

where α denotes a trade-off parameter, and $y_{\text{tmpl}}(\mathbf{z})$ and $y_{\text{hist}}(\mathbf{z})$ represent the tracking responses based on template and on histogram information, respectively.

The tracking response on template is derived by learning the optimal correlation filter model $\mathbf{w}$, which is efficiently solved in frequency domain through the fast Fourier transformation (FFT). At time t, the FFT of the filter responses is first calculated using $\mathbf{w}$ and an inverse FFT is then conducted to derive the final response $y_{\text{tmpl}}(\mathbf{z})$. The model $\mathbf{w}$ is online updated in each frame.

The tracking response on histogram is based on a learned color statistic model $\mathbf{h}$, which is robust in resisting deformation. At time t, $\mathbf{h}$ is utilized to calculate $y_{\text{hist}}(\mathbf{z})$, and then dynamically updated. To adapt the tracker to scale changes, a scale filter is adopted to estimate the target scale. More details about the Staple tracker can be found in [20].

To efficiently leverage Staple as $\mathcal{T}$ in PTAV, in addition to the tracking results of the original Staple algorithm, $\mathcal{T}$ stores all intermediate results (e.g., $\mathbf{w}$ and $\mathbf{h}$) for each frame after sending out the last verification request. Let $\mathcal{W} = \{\mathbf{w}_{\xi-\triangle}, \cdots, \mathbf{w}_{\xi}\}$ and $\mathcal{H} = \{\mathbf{h}_{\xi-\triangle}, \cdots, \mathbf{h}_{\xi}\}$ represent the collections of $\mathbf{w}$ and $\mathbf{h}$, where ξ is the index of the last frame processed by $\mathcal{T}$, and $\triangle$ denotes a fixed size for temporal sliding window to store tracking models. These intermediate results in $\mathcal{W}$ and $\mathcal{H}$ allow $\mathcal{T}$ for fast tracing back.

In particular, when $\mathcal{V}$ detects unreliable tracking result in frame k while $\mathcal{T}$ starts working on frame j ($j > k$), a feedback consisting of correct target position and frame index information is sent to $\mathcal{T}$. Once receiving this feedback from $\mathcal{V}$, $\mathcal{T}$ stops processing frame j and then utilizes the archived target position and tracking model (i.e., $\mathbf{w}_{k-1}$ and $\mathbf{h}_{k-1}$) retrieved from $\mathcal{W}$ and $\mathcal{H}$ to resume subsequent tracking from frame k. Meanwhile, useless intermediate results in $\mathcal{W}$ (i.e., $\mathbf{w}_k$ to $\mathbf{w}_{j-1}$) and $\mathcal{H}$ (i.e., $\mathbf{h}_k$ to $\mathbf{h}_{j-1}$) are discarded.

Note that we do not assume the correctness of $\mathbf{w}_{k-1}$ and $\mathbf{h}_{k-1}$ in the above strategy. In fact, one way is to trace backward from $k-1$ to locate a reliable frame to resume tracking, at additional expense of more verification operations. In practice, however, we find that $\mathbf{w}_{k-1}$ and $\mathbf{h}_{k-1}$ typically provide sufficiently good initial guess for frame k and rely on the detection part to correct the incorrect tracking result. More details are given the following sections on verifying and detection.

To validate the tracking result, $\mathcal{T}$ sends the verification requests every N_{int} frames, where N_{int} denotes the verification interval as described later.

2) **Verifying:** The goal of verifying is to measure the similarity between a given sample and the target object. Inspired by [29], we use the Siamese network to develop the verifier $\mathcal{V}$ (similar to [13]) in PTAV, as depicted in Figure 4. The Siamese network contains two branches of CNNs, and processes two inputs separately. In this work, we borrow the architecture from VGGNet [34] for CNNs, but with an additional region of interest (RoI) pooling layer [58]. This is

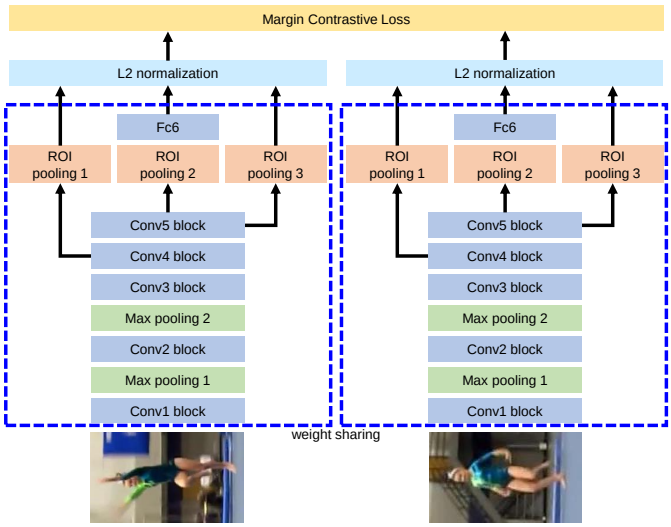


Fig. 4. Illustration of the architecture of the Siamese network for verifier.

because, for detection, $\mathcal{V}$ needs to process multiple regions in an image, from them the candidate most similar to the target object is selected as the final result. For efficiency, RoI pooling is used for simultaneously processing a set of regions.

In the Siamese network, the two CNN branches are connected with a single contrastive loss layer

$$\mathcal{L}(\mathbf{x}_i, \mathbf{x}_j, r_{ij}) = \frac{1}{2} r_{ij} D^2 + \frac{1}{2}(1-r_{ij})\max(0, \varepsilon - D^2) \quad (2)$$

where $D = \|\psi(\mathbf{x}_i) - \psi(\mathbf{x}_j)\|_2$ is the Euclidean metric in which $\psi(\cdot)$ represents feature transformation via the Siamese network, $r_{ij} \in \{0, 1\}$ indicates whether $\mathbf{x}_i$ and $\mathbf{x}_j$ are the same object or not, and ε represents the minimum distance margin.

Once training is finished[2], one can use the learned verifying function ν to compute verification score for each tracking result $\mathbf{x}'$ via

$$\nu(\mathbf{x}_{\text{obj}}, \mathbf{x}') = \psi(\mathbf{x}_{\text{obj}})^T \psi(\mathbf{x}') \quad (3)$$

where $\mathbf{x}_{\text{obj}}$ represents a fixed target template in the first frame. This strategy, as used in our preliminary work [33], may meet problems when the target object undergoes large appearance variations or deformations. As a result, using a fixed target template for verification may be unreliable for distant subsequent tracking results.

To alleviate this issue, we propose to employ a dynamic target template set $\mathcal{S}$ for adaptive verification using k-means clustering. More specifically, $\mathcal{S}$ is comprised of two components $\mathcal{S}_f$ and $\mathcal{S}_d$. The $\mathcal{S}_f = \{\mathbf{x}_{\text{obj}}\}$ contains only the target template $\mathbf{x}_{\text{obj}}$ in the first frame and is fixed during tracking. The set $\mathcal{S}_d$ is initially empty. During tracking, it is dynamically updated by collecting tracking results with high verification scores, as described later.

With the dynamic set $\mathcal{S}$, we can compute the verification score for each tracking result $\mathbf{x}'$ as follows

$$\nu(\mathcal{S}, \mathbf{x}') = \omega_o \psi(\mathbf{x}_{\text{obj}})^T \psi(\mathbf{x}') + \omega_c \sum_{i=1}^{N_C} \sum_{\mathbf{x}_j \in C_i} \psi(\mathbf{x}_j)^T \psi(\mathbf{x}') \quad (4)$$

[2]In this work we adopt the same strategy as in [13] to train the verifier. We refer readers to [13] for detailed training process.

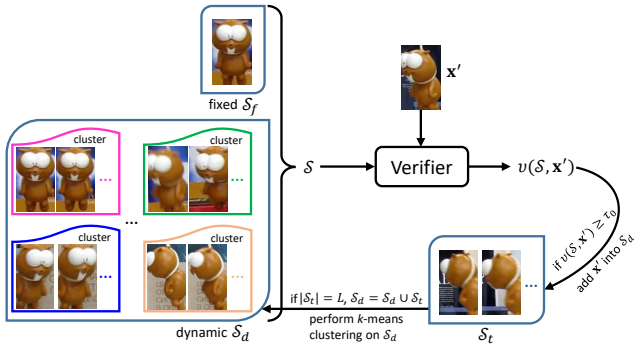


Fig. 5. Illustration of verification using a dynamic target template set.

where ω_o denotes the weight for $\mathcal{S}_f$, ω_c represents the weight for each cluster C_i obtained by performing k-means clustering on $\mathcal{S}_d$, and $N_C = |\mathcal{S}_d|/L$ is the number of clusters (L is roughly a pre-defined size of each cluster, and $|\mathcal{S}_d|$ denotes the size of $\mathcal{S}_d$). The weights ω_o and ω_c are calculated as

$$\omega_o = \frac{\exp(0.5)}{\exp(0.5) + N_C \times \exp(0.5/N_C)} \quad (5)$$

$$\omega_c = \frac{1}{N_C}(1-\omega_o) \quad (6)$$

The set $\mathcal{S}_d$ is updated as follows. For each tracking result $\mathbf{x}'$, we use Equ. 4 to calculate its verification score $\nu(\mathcal{S},\mathbf{x}')$. If $\nu(\mathcal{S},\mathbf{x}')$ is greater than a threshold τ_0, we treat $\mathbf{x}'$ as a reliable template and add it into a temporal set $\mathcal{S}_t$. This process is repeated until the number of elements in $\mathcal{S}_t$ is equal to L. We then move all elements in $\mathcal{S}_t$ to $\mathcal{S}_d$ and thus leave $\mathcal{S}_t$ empty. If the number of elements in $\mathcal{S}_d$ is greater than $L \times N_{C_{\max}}$, where $N_{C_{\max}}$ denotes the maximum number of clusters, the oldest L elements will be removed from $\mathcal{S}_d$. Afterwards, k-means clustering [59] is applied on $\mathcal{S}_d$ to obtain new clusters

$$\{C_i\}_{i=1}^{N_C} = k\text{-means}(\mathcal{S}_d, N_C) \quad (7)$$

Note that when performing k-means clustering, the elements in $\mathcal{S}_d$ are represented with HoG features [23] for the sake of efficiency. After obtaining new clusters, we employ Equ. 5 and 6 to calculate weights ω_o and ω_c. Figure 5 illustrates the process of adaptive verification.

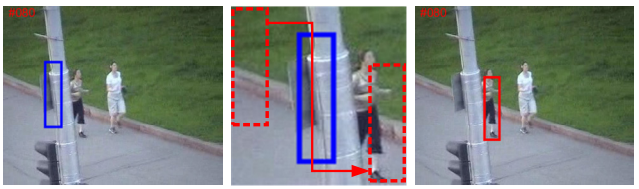
With the dynamic target template set $\mathcal{S}$, $\mathcal{V}$ can make smarter decisions than when only a fixed template is used (i.e., Equ. 3), and hence reduces the number of unnecessary verifications to speed up the entire system. Besides, now that verification is more precise, the verification-based detection (see Section III-B3) is improved as well.

3) **Verification-based detection:** Given a tracking result from $\mathcal{T}$, we use Equ. 4 to compute its verification score. If the verification score is lower than a predefined threshold τ_1, $\mathcal{V}$ will treat it as a tracking failure. In this case, $\mathcal{V}$ needs to detect the target, again using the Siamese network. Unlike for verification, detection requires to verify multiple image patches from a local region[3] and finds the best one. Thanks to the RoI pooling layer, these candidates can be simultaneously processed in just one pass, resulting in significant reduction in computation. Let $\{\mathbf{c}_i\}_{i=1}^N$ denote the candidate set generated by sliding window, and the detection result $\widehat{\mathbf{c}}$ is determined by

$$\widehat{\mathbf{c}} = \underset{\mathbf{c}_i}{\operatorname{argmax}}\ \nu(\mathcal{S},\mathbf{c}_i), \quad i = 1, 2, \cdots, N \quad (8)$$

where $\nu(\mathcal{S},\mathbf{c}_i)$ returns the verification score between the target template set $\mathcal{S}$ and candidate $\mathbf{c}_i$.

After obtaining the detection result $\widehat{\mathbf{c}}$, we determine whether or not to take it to be an alternative for tracking result according to its verification score. If $\nu(\mathcal{S},\widehat{\mathbf{c}})$ is less than a predefined threshold τ_2, $\widehat{\mathbf{c}}$ is considered to be unreliable, and we do not replace tracking result with $\widehat{\mathbf{c}}$. Instead, we decrease the verifying interval N_{int} to 1, and enlarge the local searching region for target detection. Until detection result $\widehat{\mathbf{c}}$ passes verification (i.e., $\nu(\mathcal{S},\widehat{\mathbf{c}}) \geq \tau_2$), we then restore N_{int} and the size of local searching region to initial settings. Figure 6 describes the detection process.



(a) (b) (c)

Fig. 6. Verification-based detection. When an unreliable tracking result is found (showing in blue in (a)), the verifier $\mathcal{V}$ searches/detects the target in a local region (shown in (b)). The dashed red rectangles in (b) represent object candidates generated by sliding window. The red rectangle in (c) is the detection result.

C. Implementation Details

Our PTAV is implemented in C++ and its verifier uses Caffe [60] on a single NVIDIA GTX TITAN Z GPU with 6GB memory. The merging factor α in Eq. (1) is set to 0.3. Other parameters for tracking remain the same as in [20]. The Siamese network for verification is initialized with the VGGNet [34] and trained based on the approach in [13]. The clustering interval L is empirically set to 5 and the maximum number of clusters $N_{C_{\max}}$ to 10. The verification interval N_{int} is initially set to 10. The thresholds τ_0, τ_1 and τ_2 are set to 0.6, 0.33 and 0.53, respectively. The parameter γ is initialized to 1.5, and is adaptively adjusted based on the detection result. If the detection result with $\gamma = 1.5$ is not reliable, the verifier will increase γ for a larger searching region. Meanwhile, the verification interval N_{int} is decreased to 1. When the new detection result becomes faithful, γ and N_{int} are then restored to 1.5 and 10. The source code and tracking results are available at http://www.dabi.temple.edu/~hbling/code/PTAV/ptav.htm.

IV. Experiments

A. Experiment on OTB2015

Dataset and evaluation metric. The OTB2015 [26] contains 100 fully annotated challenging video sequences. These sequences are labeled based on 11 attributes, including deformation (DEF), occlusion (OCC), scale variation (SV), illumination variations (IV), motion blur (MB), fast motion (FM),

[3]The local region is a square of size $\gamma(w^2+h^2)^{\frac{1}{2}}$ centered at the location of the tracking result in this validation frame, where w and h denote the width and height of the tracking result, and γ controls the scale and is dynamically adjusted based on detection result.

background clutter (BC), out-of-view (OV), low resolution (LR), in-plane rotation (IPR) and out-of-plane rotation (OPR).

Following [26], we use three metrics, *distance precision rate* (DPR), *overlap success rate* (OSR) and *center location error* (CLE), for tracker evaluation. DPR demonstrates the percentage of frames whose estimated average center location errors are within the given threshold distance (e.g., 20 pixels) to groundtruth. OSR shows the percentage of successful frames at the threshold ranging from 0 to 1, and can be defined as the overlap score larger than a fixed value (e.g., 0.5), where the overlap ratio is defined as score=(area($R_{GT} \cap R_T$)/area($R_{GT} \cup R_T$)) with the groundtruth R_{GT} and the tracking result R_T. CLE is the Euclid distance between centers of R_T and R_{GT}.

Overall performance. We evaluate PTAV on OTB2015 [26] and compare it with thirteen trackers from three typical categories: (i) deep feature-based tracking algorithms, including SINT [13], HCF [6], SiamFC [25], HDT [7] and CFNet [46]; (ii) correlation filter based trackers, including fDSST [18], LCT [49], KCF [17], SCT [61] and Staple [20]; and (iii) other representative tracking methods, including TLD [51], MEEM [55] and Struck [38]. We also note that there are other state-of-the-art trackers such as MDNet [8], SANet [11] and C-COT [12] (see Figure 1). However, the speeds of these trackers are around 1 frames per second (fps). Since this work is focused on real-time object tracking, we compare PTAV with trackers whose speeds are no less than 10 fps, except for SINT [13] since it can be viewed as the baseline for tracking by verification. In particular, for SINT [13], we use its tracking results without optical flow because no optical flow part is provided from the released source code. Another baseline for PTAV is the Staple tracker [20], which provides the (fast) tracking part of PTAV. Note that other tracking algorithms may also be used for the tracking part in PTAV.

We report the results in one-pass evaluation (OPE) using DPR and OSR as shown in Figure 7. Overall, PTAV performs favorably against other tracking algorithms. In addition, we present quantitative comparison of DPR at 20 pixels, OSR at 0.5, center location error (CLE) in pixels and tracking speed (fps) in Table I. It demonstrates that PTAV outperforms other trackers in all three metrics. Among the trackers under comparison, HCF [6] uses deep features to represent object appearance and obtains the DPR of 83.7% and OSR of 65.6%. Likewise, HDT [7] exploits all layers in VGGNet [34] for tracking and achieves the DPR of 84.8% and OSR of 64.8%. Compared to these two deep feature-based approaches, PTAV achieves better performance with DPR of 86.2% and OSR of 77.9%. Besides, owing to the adoption of parallel framework, PTAV (27 fps) is more than twice faster than the HCF [6] (10 fps) and HDT [7] (10 fps). Compared with SINT [13], which uses similar Siamese network for tracking, PTAV improves DPR from 77.3% to 86.2% and OSR from 70.3% to 77.6%. In addition, PTAV runs at real-time while SINT [13] needs large improvement in speed. Compared to the baseline Staple [20], PTAV achieves significant improvements on DPR (by 7.8%) and OSR (by 7.0%). Compared to representative MEEM [55] with DPR of 78.1% and OSR of 62.2%, PTAV obtains performance gains by DPR of 8.1% and OSR of 15.7%.

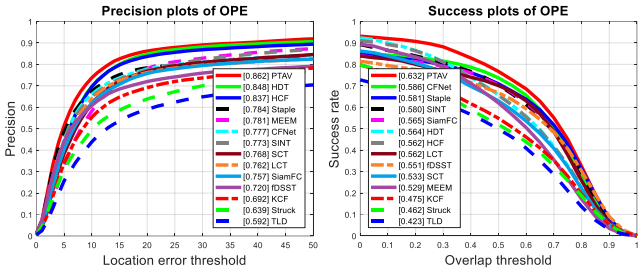


Fig. 7. Comparison with pseudo real-time trackers on OTB2015 [26] using distance precision rate (DPR) and overlap success rate (OSR).

TABLE I
COMPARISONS WITH PSEUDO REAL-TIME TRACKING METHODS ON OTB2015 [26] IN DPR (%) AT A THRESHOLD OF 20 PIXELS, OSR (%) AT AN OVERLAP THRESHOLD OF 0.5, CLE IN PIXELS AND SPEED (FPS). THE BEST TWO RESULTS ARE SHOWN IN RED AND BLUE FONTS, RESPECTIVELY.

	Algorithms	DPR	OSR	CLE	Speed
	PTAV (Ours)	86.2	77.9	18.9	27
(i)	HCF [6]	83.7	65.6	22.8	10
	HDT [7]	84.8	64.9	20.6	10
	SINT [13]	77.3	70.3	26.3	2
	SiamFC [25]	75.7	70.9	37.1	58
	CFNet [46]	77.7	73.2	35.2	43
(ii)	Staple [20]	78.4	70.9	31.9	43
	LCT [49]	76.2	70.1	67.1	25
	SCT [61]	76.8	62.1	38.0	40
	fDSST [18]	72.0	67.6	51.1	51
	KCF [17]	69.2	54.8	45.0	243
(iii)	MEEM [55]	78.1	62.2	27.7	21
	TLD [51]	59.2	48.3	35.0	20
	Struck [38]	63.9	51.6	47.1	10

Attribute-based evaluation. We further analyze PTAV under the eleven different attributes on OTB2015 in terms of DPR and OSR, as summarized in Tables II and III.

For DPR, PTAV achieves the best results under 7 out of 11 attributes including IV (84.7%), OPR (83.5%), SV (82.5%), OCC (81.4%), MB (80.5%), OV (79.0%) and BC (87.6%). For sequences with deformation, HDT [7] performs the best with average DPR of 82.1% owing to the use of richer deep features. Our tracker uses Staple [20] as the tracking part, which leverages color information to handle object deformation. Accompanied by an accurate verifier, PTAV achieves competitive performance and ranks the second in the case of deformation with average DPR of 81.2%. Furthermore, compared to the baseline Staple [20], we obtain large gain on DPR by 6.4% under deformation. For low resolution sequences, CFNet [46] obtains the best result with average DPR of 86.1% by taking advantage of deep features. PTAV ranks the second with competitive average DPR of 84.0%.

For sequences with fast motion and in-plane rotation, HDT [7] and HCF [6] perform better than ours, because in these two situations deep features are more efficacious than hand-crafted features to represent object appearance. In PTAV, we utilize simple HoG [23] features and RGB histograms to model object appearance in tracking, which are sensitive to in-plane rotation and fast motion (we can see that from the performance of Staple [20]). Nevertheless, with the help of useful feedbacks from a robust and accurate verifier, PTAV still achieves competitive performance with average DPRs of 78.1% and 82.8% under these two challenges, respectively.

TABLE II
AVERAGE DPR (%) OF INDIVIDUAL ATTRIBUTES ON OTB2015 [26]. THE BEST TWO RESULTS ARE SHOWN IN RED AND BLUE FONTS, RESPECTIVELY.

Att.	PTAV	HDT [7]	HCF [6]	Staple [20]	MEEM [55]	CFNet [46]	SINT [13]	SiamFC [25]	LCT [49]	fDSST [18]	KCF [17]	Struck [38]	TLD [51]	SCT [61]
IV	84.7	82.0	81.7	79.1	74.0	75.7	80.9	73.5	74.6	72.8	70.8	54.9	55.9	76.0
OPR	83.5	80.8	81.0	74.2	79.8	75.3	79.4	74.5	75.0	66.4	67.5	59.9	57.1	75.1
SV	82.5	81.1	80.2	73.1	74.0	74.8	74.2	74.3	68.6	66.9	63.9	60.4	56.4	69.3
OCC	81.4	77.4	76.7	72.6	74.1	71.3	73.1	69.6	68.2	62.6	62.2	53.3	52.4	70.7
DEF	81.2	82.1	79.1	74.8	75.4	66.9	75.0	67.6	68.9	59.9	61.7	52.7	48.4	72.3
MB	80.5	79.4	79.7	72.6	72.1	76.1	72.8	69.8	67.3	68.4	61.7	59.4	53.6	70.0
FM	78.1	80.6	79.7	70.3	73.4	74.1	72.5	73.0	67.5	69.3	62.8	62.0	54.8	71.9
IPR	82.8	84.4	85.4	77.0	79.3	80.3	81.1	74.8	78.2	72.5	69.3	63.4	60.3	76.8
OV	79.0	66.3	67.7	66.1	68.3	65.0	72.5	67.8	59.2	57.7	49.8	49.1	45.2	53.8
BC	87.6	84.7	84.7	77.0	75.1	73.7	75.1	69.4	74.0	78.4	71.6	57.3	46.1	77.8
LR	84.0	76.6	78.7	60.9	60.5	86.1	78.8	83.4	49.0	61.7	54.5	62.8	55.2	55.4
Avg.	86.2	84.8	83.7	78.4	78.1	77.7	77.3	75.7	76.2	72.0	69.2	63.9	59.2	76.8

TABLE III
AVERAGE OSR (%) OF INDIVIDUAL ATTRIBUTES ON OTB2015 [26]. THE BEST TWO RESULTS ARE SHOWN IN RED AND BLUE FONTS, RESPECTIVELY.

Att.	PTAV	HDT [7]	HCF [6]	Staple [20]	MEEM [55]	CFNet [46]	SINT [13]	SiamFC [25]	LCT [49]	fDSST [18]	KCF [17]	Struck [38]	TLD [51]	SCT [61]
IV	64.2	53.5	54.0	59.8	51.7	57.4	61.8	54.9	56.6	55.6	47.4	42.0	41.4	52.3
OPR	60.4	53.6	53.7	53.8	52.8	55.3	58.6	54.4	54.1	50.1	45.4	42.7	39.0	51.6
SV	59.1	48.9	48.8	52.9	47.3	55.5	55.8	55.5	49.2	51.0	39.9	40.7	38.8	44.2
OCC	60.6	52.8	52.5	54.8	50.3	53.6	55.8	52.3	50.7	47.8	43.8	39.3	36.3	50.2
DEF	59.9	54.3	53.0	55.4	48.9	49.2	55.5	49.0	49.9	46.1	43.6	38.3	34.1	50.6
MB	61.2	56.3	57.3	55.8	54.3	59.3	57.4	55.5	53.2	54.8	45.6	46.1	42.6	52.1
FM	58.3	55.4	55.5	54.1	52.8	57.0	55.7	56.4	52.7	55.4	45.5	46.1	41.8	52.5
IPR	59.0	55.5	55.9	55.2	52.8	59.0	58.5	55.7	55.7	54.5	46.5	45.2	42.5	52.2
OV	56.9	47.2	47.4	48.1	48.4	48.0	55.9	51.1	45.2	45.7	39.3	37.8	33.5	43.4
BC	64.1	58.0	58.7	57.4	52.1	54.5	56.7	50.4	55.3	58.5	49.8	44.2	35.2	55.2
LR	54.6	42.0	42.4	41.1	35.5	61.9	53.9	60.4	33.0	44.6	30.6	34.7	37.2	31.0
Avg.	63.2	56.4	56.2	58.1	52.9	58.6	58.0	56.5	56.2	55.1	47.5	46.2	42.3	53.3

For OSR, on the other hand, PTAV achieves the best results under 10 of 11 attributes including IV (64.2%), OPR (60.4%), SV (59.1%), OCC (60.6%), DEF (59.9%), MB (61.2%), FM (58.3%), IPR (59.0%), OV (56.9%) and BC (64.1%). Low resolution (LR) is the only attribute for which PTAV does not rank the best, while CFNet [46] and SiamFC [25] obtain better results. Specifically, PTAV achieves an OSR of 54.6%, higher than all other trackers including its two baselines.

Qualitative evaluation. To further analyze and demonstrate the performance of PTAV, we conduct rich qualitative evaluation described as following.

Occlusions: Figure 8(a) shows tracking results on sequences *Box*, *Lemming*, *Girl2* and *Jogging-1*, all involving heavy target occlusions. From Figure 8(a) we can see that PTAV handles well the occlusion in these sequences. Though the tracking part may lose the target temporally due to occlusions, it can quickly be corrected by the verifier and resumes tracking. Compared to deep trackers (HCF [6], HDT [7] and CFNet [46]), which lose the tracking target when occlusions happen (e.g., #342 in *Girl2* and #489 in *Lemming*), PTAV performs more robustly. Since correlation filter is sensitive to occlusion and no re-detection module is adopted, KCF [17], fDSST [18] and Staple [20] lose the tracking target in all sequences. LCT [49] uses an additional detector to re-localize the target after occlusion. However, it still fails when occlusions are accompanied with background clutters (e.g., #476 in *Box*). MEEM [55] stores multiple statuses during tracking, and re-detects the target using old memories (e.g., *Lemming*, *Girl2* and *Jogging-1*). However, it meets problems in presence of background clutters (e.g., #476 in *Box*). SiamFC [25] and SINT [13] deal with occlusion by searching in a local region when the target recovers from occlusions. Other trackers such as TLD [51], SCT [61] and Struck [38] do not well handle the occlusion and drift to background (e.g., #342 in *Girl2*).

Background clutters: Figure 8(b) displays the results on sequences *Coke*, *Deer*, *Football* and *Bolt2* with background clutter. We observe that SiamFC [25], SINT [13], KCF [17], Struck [38] and Staple [20] drift to background in *Football* (e.g., #360). CFNet [46] and fDSST [18] localize well the target in *Football*, but still fail in *Coke* (e.g., #291) where background clutters are accompanied with occlusions and illumination changes. MEEM [55] and SCT [61] handle nicely occlusion and motion blur, but fail in presence of deformations in *Bolt2* (e.g., #40). LCT [49] and TLD [51] are robust against background clutters since they use an extra detector to seek for the target after it moves away from the cluttering region. However, they lose the target when heavy deformation happens in *Bolt2* (e.g., #271). HCF [6] and HDT [7] perform robustly in these sequences because of the powerful deep features. Likewise, PTAV is able to handle these cases owing to the verifier. Besides, the cooperation between tracking and verifying allows PTAV to run in real-time.

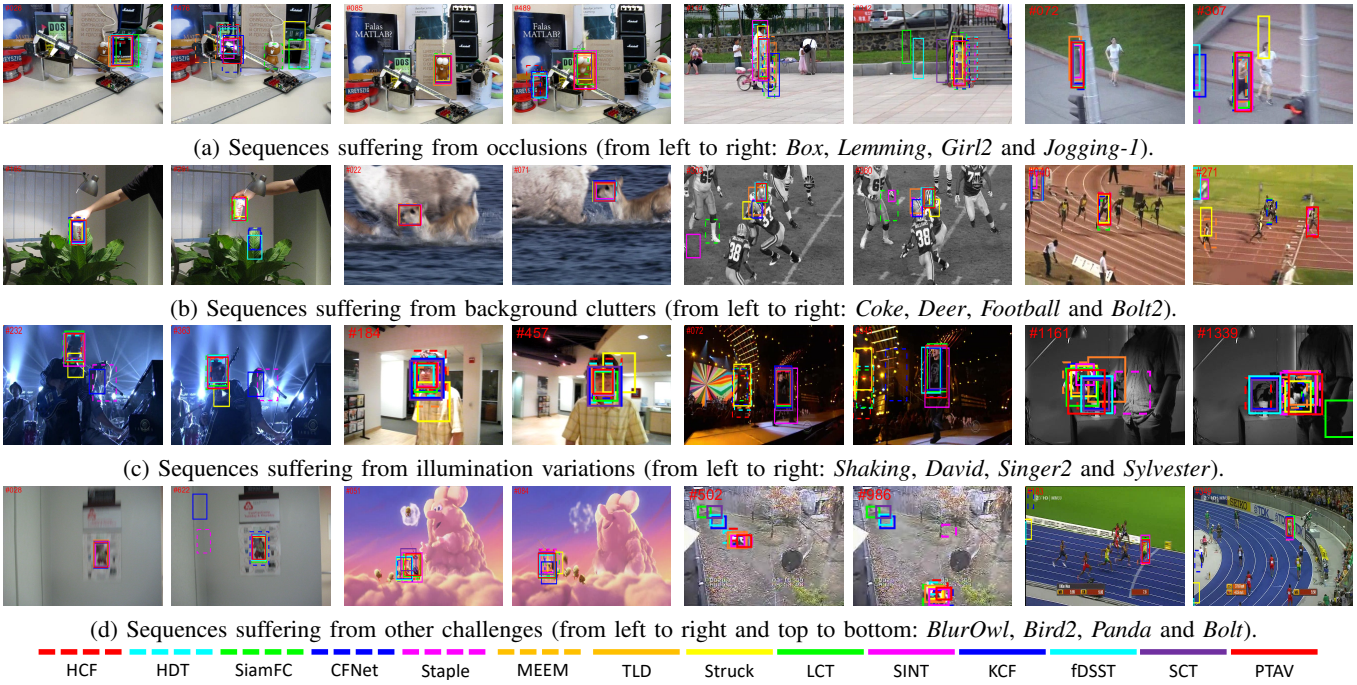
Illumination variations: Figure 8(c) shows sampled results of sequences *Shaking*, *David*, *Singer2* and *Sylvester*. We can see that deep feature-based trackers SiamFC [25], CFNet [46], HCF [6] and HDT [7] lose the target in *Singer2* (e.g., #345) where background clutters happen. MEEM [55] can handle rotations but still fails in *Singer2* (e.g., #345). Staple [20] and KCF [17] are sensitive to rotation and fail in *Shaking* (e.g., #363) and *Sylvester* (e.g., #1339). LCT [49] and SCT [61] work well in *Shaking*, *David* and *Singer2*, yet have problems when rotation occurs in *Sylvester* (e.g., #1339).PTAV performs well on these sequences. Though its tracking part may drift to background due to rotation (e.g., #1161 in *Sylvester*), this situation is found and immediately corrected by its verifier (e.g., #1339 in *Sylvester*).

Other challenges: Figure 8(d) demonstrates the results of sequences *BlurOwl*, *Bird2*, *Panda* and *Bolt2* with other challenges including motion blur, rotation, scale change, deformation and so on. In *BlurOwl*, the camera moves quickly, causing serious motion blur (e.g., #622). KCF [17] and Staple [20] lose the target, while PTAV well localizes the object thanks to its verifier which corrects tracker. In *Bolt2*, TLD [51], Struck [38], CFNet [46] and fDSST [18] drift to background due to deformation. On *Panda*, LCT [49], fDSST [18], Staple [20], SCT [61] and KCF [17] lose tracking target owing to scale change and rotation. By contrast, MEEM [55], HCF [6], HDT [7], SiamFC [25], SINT [13] and our PTAV perform favorably owing to powerful feature representation.

B. Experiment on TC128

The TC128 benchmark [30] consists of 128 fully annotated color sequences. On TC128 [30], PTAV runs at 24 *fps* and is compared with eleven state-of-the-art trackers including MEEM [55], HCF [6], HDT [7], Staple [20], SiamFC [25], SRDCF [50], DeepSRDCF [62], fDSST [18], KCF [17], LCT [49] and Struck [38]. Following [30], we report evaluation results in OSR and DPR as shown in Figure 9.

Among the eleven compared trackers, DeepSRDCF [62] extends SRDCF [50] by replacing hand-crafted features with



(a) Sequences suffering from occlusions (from left to right: *Box*, *Lemming*, *Girl2* and *Jogging-1*).

(b) Sequences suffering from background clutters (from left to right: *Coke*, *Deer*, *Football* and *Bolt2*).

(c) Sequences suffering from illumination variations (from left to right: *Shaking*, *David*, *Singer2* and *Sylvester*).

(d) Sequences suffering from other challenges (from left to right and top to bottom: *BlurOwl*, *Bird2*, *Panda* and *Bolt*).

Fig. 8. Qualitative evaluation of the proposed algorithm and other thirteen state-of-the-art trackers on sixteen challenging sequences.

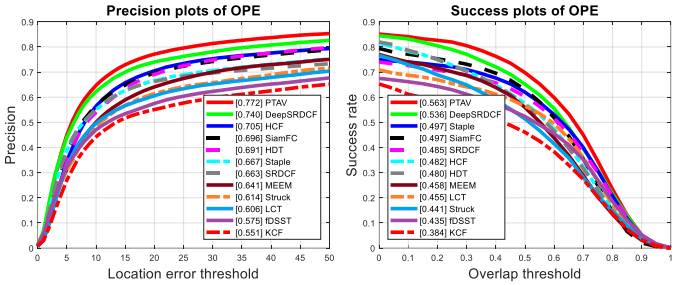


Fig. 9. Comparison with eleven state-of-the-art trackers on TC128 [30] using distance precision rate and overlap success rate.

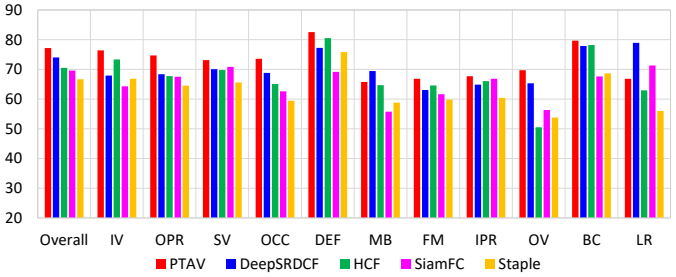


Fig. 10. Average DPR (%) in term of individual attributes on TC128 [30].

convolutional features and obtains the best performance with DPR of 74.0% and OSR of 53.6%. By contrast, PTAV improves the state-of-the-art methods on DPR to 77.2% and OSR to 56.3%, obtaining the gains of 3.2% and 2.7%, respectively. In comparison with SiamFC [25] with DPR of 69.6% and OSR of 49.7%, PTAV achieves improvements of 7.2% and 6.6% on DPR and OSR, respectively. Compared with the other baseline, Staple [20], which obtains a DPR of 66.7% and an OSR of 49.7%, PTAV achieves significant improvements as well, showing clearly the benefits of introducing a verifier. For more detailed analysis, we show the average DPR for five trackers on different attributes in Figure 10. PTAV can well handle various challenging factors and outperform the other four trackers in nine out of eleven attributes.

C. Experiment on UAV20L

The recent UAV20L dataset [31] contains 20 fully annotated sequences, with length ranging from 1,717 to 5,527 frames. PTAV runs at 30 *fps* and is compared with ten trackers including SiamFC [25], MUSTer [57], SRDCF [50], HCF [6], MEEM [55], SAMF [45], Struck [38], fDSST [18], LCT [49] and KCF [17].

Following [31], we report evaluation results in Figure 11. PTAV achieves the best performance in both DPR (73.2%) and OSR (50.4%), outperforming other trackers by large margins (6.2% and 10.1% compared to the second best in DPR and OSR, respectively). Furthermore, we also analyze PTAV on twelve individual attributes provided with UAV20L [31], icnluding scale variation (SV), aspect ratio change (ARC), low resolution (LR), fast motion (FM), full occlusion (FOC), partial occlusion (POC), out-of-view (OV), background clutter (BC), illumination variation (IV), viewpoint change (VC), camera motion (CM) and similar object (SOB). Figure 12 displays the average DPR for five trackers on different attributes, and PTAV achieves the best results on each attribute.

D. Experiment on VOT2016

Finally, we test PTAV on VOT2016 [32] with 60 challenging sequences. VOT2016 aims at evaluating short-term tracking performance and thus a tracker is re-initialized whenever failure happens. In other words, a tracker is reset if its tracking

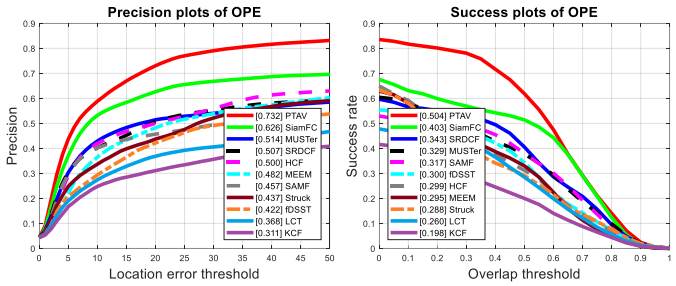


Fig. 11. Comparison with ten state-of-the-art trackers on UAV20L [31] using distance precision rate and overlap success rate.

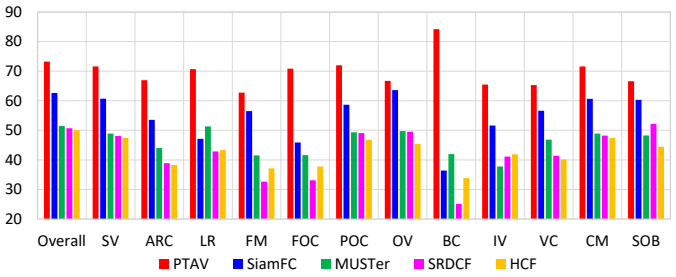


Fig. 12. Average DPR (%) in term of attributes on UAV20L [31].

results are found unreliable. Unfortunately, this protocol is not directly applicable to our tracker since PTAV automatically detects failures by itself and rolls back to resume tracking.

To follow the above evaluation protocol, we modify PTAV by running it multiple rounds with different starting frames. In particular, in each round, we run PTAV at current starting frame without resetting. For the first round, the first frame in the input sequence is used as the start frame. Afterward, we compare the tracking results with groundtruth to find the first failure using the VOT2016 protocol, and then we re-initialize PTAV from the failure frame for the next round. We repeat this process until no failure is detected. On VOT2016, PTAV runs at 25 *fps*.

PTAV is compared with top ten trackers in the VOT2016 challenge, including C-COT [12], TCNN [63], SSAT [32], MLDF [32], Staple [20], DDC [32], EBT [64], SRBT [32], Staple+ [32] and DNT [65]. Table IV demonstrates comparison results in VOT2016. It shows that C-COT [12] and TCNN [63] achieve the best results with EAOs of 33.1% and 32.5%, respectively. C-COT [12] utilizes deep features to model object appearance and TCNN [63] proposes tree-structured CNNs for tracking with online update. Despite obtaining superior performances, their speeds are around 1 and 2 *fps*. By contrast, PTAV achieves competitive result (EAO of 31.2%), while running in real-time.

E. Ablation Study

1) **Different trackers for $\mathcal{T}$:** In PTAV, $\mathcal{T}$ is required to be efficient and accurate most of the time. To show the effects of different $\mathcal{T}$, we compare three base trackers including Staple [20] (the choice in this paper), fDSST [18] and KCF [17]. Among these trackers, KCF [17] runs the most efficiently while least accurately in short time. Compared to KCF [17] and fDSST [18], Staple [20] performs more robustly since it utilizes color information for tracking, which results in its relative inefficiency. The comparison results on OTB2015 [26], TC128 [30] and UAV20L [31] are shown in Table V.

TABLE IV
COMPARISONS ON VOT2016 [32] IN TERMS OF EXPECTED AVERAGE OVERLAP (EAO%), ACCURACY (%), ROBUSTNESS (%) AND NO-RESET AVERAGE OVERLAP (AO%). THE BEST TWO RESULTS ARE SHOWN IN RED AND BLUE FONTS, RESPECTIVELY.

Algorithms	EAO	Accuracy	Robustness	AO
PTAV (Ours)	31.2	56.1	27.9	43.2
C-COT [12]	33.1	53.9	23.8	46.9
TCNN [63]	32.5	55.4	26.8	48.5
SSAT [32]	32.1	57.7	29.1	51.5
MLDF [32]	31.1	49.0	23.3	42.8
Staple [20]	29.5	54.4	37.8	38.8
DDC [32]	29.3	54.1	34.5	39.1
EBT [64]	29.1	46.5	25.2	37.0
SRBT [32]	29.0	49.6	35.0	33.3
Staple+ [32]	28.6	55.7	36.8	39.2
DNT [65]	27.8	51.4	32.9	42.7

TABLE VIII
COMPARISONS OF DIFFERENT N_{int} ON OTB2015 [26].

	$N_{\text{int}} = 5$	$N_{\text{int}} = 10$	$N_{\text{int}} = 15$
DPR (%)	86.3	86.2	84.6
Speed (fps)	25	27	30

From Table V, we observe that PTAV with Staple as base tracker ($\text{PTAV}_{\text{Staple}}$) performs better than those with fDSST ($\text{PTAV}_{\text{fDSST}}$) and KCF ($\text{PTAV}_{\text{KCF}}$). Though KCF runs the fastest among these trackers, it performs least accurately in short time, resulting in more requests for verifications and detections, and significantly increased computations. As shown in Table V, the speeds of PTAV_{KCF} on OTB2015 [26], TC128 [30] and UAV20L [31] are respectively 24, 19 and 20 *fps*, which are much slower than $\text{PTAV}_{\text{Staple}}$ (27, 23 and 30 *fps*, respectively) and $\text{PTAV}_{\text{fDSST}}$ (27, 24 and 26 *fps*, respectively).

In term of tracking accuracy, on OTB2015 [26], $\text{PTAV}_{\text{fDSST}}$ achieves competitive performance (85.2% of DPR and 77.9 % of OSR) compared to $\text{PTAV}_{\text{Staple}}$ (86.2% of DPR and 77.9% of OSR). However, on the more challenging UAV20L [31], $\text{PTAV}_{\text{Staple}}$ significantly outperforms $\text{PTAV}_{\text{fDSST}}$ in both accuracy and efficiency. Specifically, $\text{PTAV}_{\text{Staple}}$ obtains a DPR of 73.2%, an OSR of 62.4% and speed of 30 *fps* while $\text{PTAV}_{\text{fDSST}}$ with DPR of 63.1%, OSR of 47.8% and speed of 26 fps. The main reason accounting for this is that the baseline Staple leverages color cues for tracking. In UAV20L [31], the tracking target frequently suffers from severe view changes, which are fatal to HoG features. Nevertheless, Staple is able to deal with view changes using color statistics, and thus performs better than fDSST in short periods. As a consequence, $\text{PTAV}_{\text{Staple}}$ performs more favorably than $\text{PTAV}_{\text{fDSST}}$ and requires less verifications and detections, further improving efficiency. Besides, we observe that all the three PTAV versions improve their baseline trackers by large margins.

2) **Different verifiers for $\mathcal{V}$:** The verifier $\mathcal{V}$ plays a crucial role in PTAV by validating tracking results and correcting $\mathcal{T}$ if needed. To study the effects of $\mathcal{V}$, we compare two

TABLE V
COMPARISONS OF DPR (%), OSR (%), CLE IN PIXELS AND SPEED (FPS) AMONG DIFFERENT $\mathcal{T}$ WITH VGGNET [34] BASED $\mathcal{V}$ ON THREE BENCHMARKS.

	OTB2015 [26]				TC128 [30]				UAV20L [31]			
	DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed
$\text{PTAV}_{\text{Staple}}$	86.2	77.9	18.9	27	77.2	70.0	32.1	23	73.2	62.4	56.1	30
$\text{PTAV}_{\text{fDSST}}$	85.2	77.9	19.4	27	75.2	66.1	32.7	24	63.1	47.8	102.7	26
PTAV_{KCF}	74.2	58.6	34.5	24	63.9	54.6	53.6	19	41.6	32.1	195.3	20
Staple [20]	78.4	70.9	31.9	43	66.7	62	57.5	43	48.5	44.3	223.1	49
fDSST [18]	72.0	67.6	51.1	51	57.5	52.4	82.1	51	42.2	34.4	256.8	52
KCF [17]	69.2	54.8	45.0	243	55.1	46.1	77.4	242	31.1	20.0	282.4	245

TABLE VI
COMPARISONS OF DPR (%), OSR (%), CLE IN PIXELS AND SPEED (FPS) BETWEEN DIFFERENT $\mathcal{V}$ WITH SAME TRACKER ON THREE BENCHMARKS.

		OTB2015 [26]				TC128 [30]				UAV20L [31]			
		DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed
Staple	$\text{PTAV}_{\text{VGGNet}}$	86.2	77.9	18.9	27	77.2	70.0	32.1	23	73.2	62.4	56.1	30
	$\text{PTAV}_{\text{AlexNet}}$	84.0	75.5	20.7	34	75.0	68.9	40.3	31	65.9	58.6	70.7	33
fDSST	$\text{PTAV}_{\text{VGGNet}}$	85.2	77.9	19.4	27	75.2	66.1	32.7	24	63.1	47.8	102.7	26
	$\text{PTAV}_{\text{AlexNet}}$	79.4	74.3	31.2	29	67.5	61.1	43.8	26	54.7	43.2	121.0	31
KCF	$\text{PTAV}_{\text{VGGNet}}$	74.2	58.6	34.5	24	63.9	54.6	53.6	19	41.6	32.1	195.3	20
	$\text{PTAV}_{\text{AlexNet}}$	72.3	58.0	37.1	31	62.4	52.6	54.9	22	39.6	27.6	216.1	27

TABLE VII
COMPARISONS IN TERMS OF DPR (%), OSR (%), CLE IN PIXELS AND SPEED (FPS) BETWEEN FIXED TEMPLATE AND DYNAMIC TEMPLATE SET USING $\mathcal{T}$ BASED ON DIFFERENT TRACKERS AND $\mathcal{V}$ BASED ON VGGNET [34] ON THREE BENCHMARKS.

		OTB2015 [26]				TC128 [30]				UAV20L [31]			
		DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed	DPR	OSR	CLE	Speed
Staple	dynamic templates	86.2	77.9	18.9	27	77.2	70.0	32.1	23	73.2	62.4	56.1	30
	fixed template	85.6	77.1	20.4	25	76.3	68.9	32.3	22	72.6	61.6	62.8	28
fDSST	dynamic templates	85.2	77.9	19.4	27	75.2	66.1	32.7	24	63.1	47.8	102.7	26
	fixed template	84.9	77.6	21.1	25	74.1	64.2	35.7	21	62.4	45.6	113.2	25
KCF	dynamic templates	74.2	58.6	34.5	24	63.9	54.6	53.6	19	41.6	32.1	195.3	20
	fixed template	73.5	57.9	37.2	21	62.8	52.9	56.9	14	40.8	30.9	214.2	18

different alternatives based on VGGNet [34] and the much lighter AlexNet [5][4]. Compared to the VGGNet-based $\mathcal{V}$, the AlexNet-based $\mathcal{V}$ runs more efficiently but less accurately. Specifically, the speed of VGGNet based-$\mathcal{V}$ runs at 6 *fps* while its AlexNet counterpart runs at 17 fps. The comparisons of PTAV with different verifiers are reported in Table VI. From Table VI, we observe that, when using the same base tracker, PTAV with VGGNet-based $\mathcal{V}$ ($\text{PTAV}_{\text{VGGNet}}$) outperforms that with AlexNet-based $\mathcal{V}$ ($\text{PTAV}_{\text{AlexNet}}$) on all three benchmarks. However, owing to heavier computational burden, the efficiency of $\text{PTAV}_{\text{VGGNet}}$ is slightly decreased, but still competitive to that of $\text{PTAV}_{\text{AlexNet}}$, showing the flexibility of our framework.

3) **Fixed template v.s. dynamic template set:** To adapt $\mathcal{V}$ to target appearance variation, we propose the dynamic template set for adaptive verification, which can take advantages of confident tracking results to improve validation quality, leading to reduction of verifications and detections for efficiency. Table VII shows the results of PTAV using dynamic template set versus using a fixed template. From Table VII we observe that using dynamic template set for verification improves PTAV for different trackers in both accuracy and efficiency. In specific, when using the Staple as base tracker, DPRs on three benchmarks are improved from 85.6%, 76.3% and 72.6% to 86.2%, 77.2% and 73.2%, respectively. For fDSST as the base tracker, the DPRs are improved from 84.9%, 74.1%, and 62.4% to 85.2%, 75.2%, and 63.1%, respectively. For tracker with KCF, the DPRs are improved from 73.5%, 62.8%, and 40.8% to 74.2%, 63.9%, and 41.6%, respectively. The consistent performance gains demonstrate the effectiveness of the proposed dynamic templates for verification. In addition, the speeds of PTAV are also consistently boosted owing to the higher validation quality, as shown in Table VII.

[4]For the verifier with AlexNet [5], one just needs to replace and initialize the five convolutional blocks in Figure 4 with AlexNet.

TABLE IX
COMPARISONS OF TRACKING SPEED (FPS) ON THREE BENCHMARKS.

	OTB2015 [26]	TC128 [30]	UAV20L [31]	VOT2016 [32]
Single thread	15	14	17	15
Two threads	27	23	30	25

4) **Different verification interval N_{int}:** In PTAV, different verification interval N_{int} may affect both the accuracy and efficiency. A smaller N_{int} implies that more frequent verification operations are performed during tracking, which results in more computation and thus degrades the efficiency of system, as evidenced by the lower speed (25 *fps*) of $N_{\text{int}} = 5$ compared to that (27 *fps*) of $N_{\text{int}} = 10$ in Table VIII. Besides, the deep feature based verifier in our framework is fixed during tracking for the sake of efficiency. As a consequence, the discriminative power of verifier is limited compared with existing online fine-tuned deep trackers (e.g., [8]). Thus, using a smaller N_{int} in PTAV does not guarantee large improvement in accuracy. As shown in Table VIII, the DRP is only increased by 0.1% from 86.2% to 86.3% when changing N_{int} from 10 to 5. A larger N_{int}, on the contrary, may save some computation but may put

PTAV at the risk when the target appearance changes quickly. If the $\mathcal{T}$ loses the target, it may update vast backgrounds in its appearance model until next verification. Even if the $\mathcal{V}$ re-locates the target and offers a correct detection result, the tracker may still lose it due to heavy changes in the target appearance model. From Table VIII, we observe that when increasing N_{int} from 10 to 15, the speed is slightly improved from 27 *fps* to 30 *fps*. Nevertheless, the DPR is significantly decreased by 1.6% from 86.2% to 84.6%. Taking into account both accuracy and speed, we set N_{int} to 10 in our experiments.

5) **Two threads v.s. one:** In PTAV, the tracker $\mathcal{T}$ does not rely on the verifier $\mathcal{V}$ most of the time, and two separate threads process tracking and verifying in parallel for efficiency. Consequently, $\mathcal{T}$ does not have to wait for the feedback from $\mathcal{V}$ to process next frame, and it traces back and resumes tracking only when receiving a correction feedback from $\mathcal{V}$. Owing to storing intermediate results, $\mathcal{T}$ is able to quickly trace back without much extra computation. Table IX shows the comparison of speed between using two threads and using only a single thread. It shows that using two threads in parallel clearly improves the efficiency of the whole system. It is worth noting that the only difference of PTAV using one single thread and two parallel threads is the efficiency. In term of tracking accuracy, these two different implementations achieve the same performance. For PTAV with one single thread, the DPRs on OTB-2015 [26], TC-128 [30] and UAV20L [31] are 86.2%, 77.2% and 73.2% respectively, which are the same as PTAV with two threads.

F. A Special Design of PTAV

The main contribution of our work is the proposal of a novel tracking framework, PTAV, which aims at facilitating tracking in practical applications. PTAV is flexible and each component of it can be designed based on specific objective. For instance, in order to improve the ability of PTAV in dealing with occlusion, we develop a more accurate verifier in PTAV by incorporating attention mechanism into $\mathcal{V}$, which is able to automatically assign more importance to the visible parts of target for accurate validation when occlusion happens. In specific, the attention model is applied after each RoI pooling layer. Let $\mathbf{f} \in \mathbb{R}^{m\times n\times c}$ denote the feature after the RoI pooling layer. The attention model consists of convolution and softmax operations. In detail, the RoI pooled feature $\mathbf{f}$ is fed to two consecutive convolution layers with kernel sizes 3×3 and 1×1 to obtain $\mathbf{f}_{map} \in \mathbb{R}^{m\times n\times 1}$. Afterwards, we perform softmax operation on spatial locations of $\mathbf{f}_{map}$, and obtain the spatial attentional map $\mathbf{m} \in \mathbb{R}^{m\times n}$. The attentional feature $\mathbf{f}_{att}$ is derived by multiplying each channel of $\mathbf{f}$ with $\mathbf{m}$. These attentional features are then normalized and fed to the loss layer as in Eq. (2) to learn an attentional verifier.

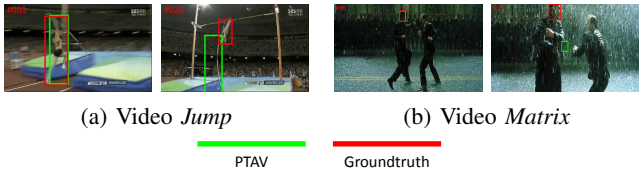
We evaluate this PTAV with the specially designed verifier for handling occlusion (referred to as PTAV$_{occ}$) on OTB2015 [26]. Table X shows comparisons between PTAV and PTAV$_{occ}$ in terms of the overall performance on OTB2015 and evaluation results on 49 sequences with occlusions. From Table X, we observe that the specially designed attention model effectively improves the performance of the system in dealing with occlusion. In detail, PTAV$_{occ}$ improves the DPRs on OTB2015 and its occlusion subset from 86.2% to 86.8% and from 81.4% to 82.6%. Such improvements with simple modification exhibit the flexibility and practicability of our framework.

TABLE X
COMPARISON OF PTAV AND ITS VARIANT PTAV$_{occ}$ ON DPR (%).

Dataset	PTAV	PTAV$_{occ}$
OTB2015 [26]	86.2	86.8
OTB2015-Occlusion [26]	81.4	82.6

G. Failure Cases

With the collaboration between $\mathcal{T}$ and $\mathcal{V}$, PTAV usually performs well in various challenging situations; however, there exist scenarios in which PTAV may fail. As shown in Figure 13(a) on *Jump*, though $\mathcal{V}$ can detect unreliable tracking results of $\mathcal{T}$, it does not provide correct feedbacks for subsequent tracking due to heavy deformation. On *Matrix* shown in Figure 13(b), the target undergoes severe illumination variation, occlusion, rotation and background cluttering, causing difficulties for $\mathcal{T}$ to localize the target. Even though $\mathcal{V}$ corrects $\mathcal{T}$ when validating unreliable tracking results, $\mathcal{T}$ still drifts to background quickly. In certain cases where the target suffers form heavy appearance changes, $\mathcal{V}$ cannot provide effective feedbacks to $\mathcal{T}$, resulting in irrecoverable drift and failures.



(a) Video *Jump* (b) Video *Matrix*

Fig. 13. Failure cases of *Jump* and *Matrix* for PTAV on OTB2015 [26].

H. Discussion about the PTAV Architecture

PTAV provides a flexible framework to integrate a tacker and a verifier for real-time tracking with high accuracy. It is necessary to analyze the architecture of PTAV for accuracy and efficiency.

Accuracy. PTAV improves tracking accuracy by combining a tracker with a verifier using two parallel threads. In order to demonstrate the effectiveness of this design in term of accuracy, we compare three trackers on OTB2015 [26] including the baseline Staple [20], PTAV implemented on a single thread (referred to as PTAV$_{st}$) and the proposed PTAV. In detail, PTAV$_{st}$ achieves the DPR of 86.2%. Compared to the baseline Staple with DPR of 78.4%, PTAV$_{st}$ obtains 7.8% improvement in DPR, demonstrating a clear benefit of integrating a tracker with a verifier for tracking. Besides, PTAV also achieves the same performance as PTAV$_{st}$ with DPR of 86.2%, showing no accuracy sacrifice in parallelizing tracker and verifier on two threads.

Efficiency. PTAV improves the system efficiency by splitting tracking and verifying on two parallel threads. Due to

the uncertainty in different video sequences, it is difficult to quantitatively analyze the efficiency of PTAV$_{st}$ with one thread and PTAV with two threads. Therefore, we conduct a qualitative analysis on the speed. For PTAV$_{st}$, its speed is jointly determined by the tracker and the verifier. In our implementation, the tracker and verifier run at 43 *fps* and 6 *fps*, respectively. When combining them on one single thread, the upper limit for the system speed with verification interval 10 is $1/(1/43+1/60) \approx 25$ *fps*. However, because the verifier needs perform object detection, the speed of PTAV$_{st}$ is 15 *fps*. By contrast, the speed of the proposed PTAV is determined by the relatively slow component of tracker and verifier. In our solution, the upper limit for speed can be 43 *fps* as fast as the tracker. Nevertheless, due to the interactions between verifier and tracker such as object detection and tracing back, the speed of the system is lower than the upper limit. Since the verification results for most time are positive (i.e., larger than the threshold τ_0), the detections are occasionally performed. As a result, the whole system still runs in real-time.

V. Conclusion

In this paper, we propose a new visual tracking framework, *parallel tracking and verifying* (PTAV), which decomposes object tracking into two sub-tasks, fast tracking and reliable verifying. We show that, by carefully distributing the two tasks into two parallel threads and allowing them to work together, PTAV can achieve the best known tracking accuracy among all real-time tracking algorithms. Furthermore, to adapt the verifier to object appearance variations, we propose using dynamic target templates for adaptive verification, resulting in further improvements in both accuracy and efficiency. The encouraging results are demonstrated in extensive experiments on four popular benchmarks. Moreover, PTAV is a flexible framework with great rooms for improvement and generalization, and thus is expected to inspire the designing of more efficient tracking algorithms in the future.

We summarize the potential directions to improve PTAV and shed light on our future work. First, the performance of PTAV heavily depends on the fast tracking part. In this work, we only apply one base tracker based on correlation filter for target localization, which may be sensitive to large deformation and rotation (see Figure 13). To improve PTAV, a better way for the tracking part in our future work is to simultaneously use multiple efficient trackers for target localization, and each of them is capable of dealing with different challenges. With multi-thread technology, these trackers can be implemented in parallel to save computation. Second, we adopt a simple clustering method to update the target templates for verification. Although this strategy shows advantages in comparison to our previous work [33], we do not have principled interpretations and explanations. Our future work will emphasize directly learning an adaptive deep verifier to avoid manually designing empirical update strategy. Third, the verification interval is fixed and may not be optimal for all sequences. For instance, for videos without drastic appearance changes, the verification interval should be a relatively larger number to maintain the efficiency of the whole system. By contrast, for sequences with heavy appearance variations, the verification interval should be a small to guarantee the accuracy. Therefore, it is of great interest to develop a learning-based solution to estimate a suitable verification interval.

References

[1] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *TPAMI*, vol. 36, no. 7, pp. 1442–1468, 2014.

[2] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Computing Surveys*, vol. 38, no. 4, p. 13, 2006.

[3] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. V. D. Hengel, "A survey of appearance models in visual object tracking," *ACM TIST*, vol. 4, no. 4, pp. 1–58, 2013.

[4] M. Kristan, J. Matas, A. Leonardis, T. Vojívr, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. vCehovin, "A novel performance evaluation methodology for single-target trackers," *TPAMI*, vol. 38, no. 11, pp. 2137–2155, 2016.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[6] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," in *ICCV*, 2015.

[7] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, J. Lim, and M.-H. Yang, "Hedged deep tracking," in *CVPR*, 2016.

[8] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *CVPR*, 2016.

[9] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Stct: Sequentially training convolutional networks for visual tracking," in *CVPR*, 2016.

[10] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," in *NIPS*, 2013.

[11] H. Fan and H. Ling, "Sanet: Structure-aware network for visual tracking," in *CVPR Workshop*, 2017.

[12] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *ECCV*, 2016.

[13] R. Tao, E. Gavves, and A. W. Smeulders, "Siamese instance search for tracking," in *CVPR*, 2016.

[14] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *ICCV*, 2015.

[15] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *ICML*, 2015.

[16] H. Fan and H. Ling, "Siamese cascaded region proposal networks for real-time visual tracking," in *CVPR*, 2019.

[17] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *TPAMI*, vol. 37, no. 3, pp. 583–596, 2015.

[18] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Discriminative scale space tracking," *TPAMI*, vol. 39, no. 8, pp. 1561–1575, 2017.

[19] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *CVPR*, 2010.

[20] L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik, and P. H. Torr, "Staple: Complementary learners for real-time tracking," in *CVPR*, 2016.

[21] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the circulant structure of tracking-by-detection with kernels," in *ECCV*, 2012.

[22] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, "Adaptive color attributes for real-time visual tracking," in *CVPR*, 2014.

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005.

[24] J. Van De Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning color names for real-world applications," *TIP*, vol. 18, no. 7, pp. 1512–1523, 2009.

[25] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *ECCV Workshop*, 2016.

[26] Y. Wu, J. Lim, and M.-H. Yang, "Object tracking benchmark," *TPAMI*, vol. 37, no. 9, pp. 1834–1848, 2015.

[27] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *ISMAR*, 2007.

[28] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[29] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *CVPR*, 2005.
[30] P. Liang, E. Blasch, and H. Ling, "Encoding color information for visual tracking: Algorithms and benchmark," *TIP*, vol. 24, no. 12, pp. 5630–5644, 2015.
[31] M. Mueller, N. Smith, and B. Ghanem, "A benchmark and simulator for uav tracking," in *ECCV*, 2016.
[32] M. Kristan and et al, "The visual object tracking vot2016 challenge results," in *ECCV Workshop*, 2016.
[33] H. Fan and H. Ling, "Parallel tracking and verifying: A framework for real-time and high accuracy visual tracking," in *ICCV*, 2017.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2014.
[35] B. Babenko, M.-H. Yang, and S. Belongie, "Robust object tracking with online multiple instance learning," *TPAMI*, vol. 33, no. 8, pp. 1619–1632, 2011.
[36] K. Zhang, L. Zhang, and M.-H. Yang, "Real-time compressive tracking," in *ECCV*, 2012.
[37] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised on-line boosting for robust tracking," in *ECCV*, 2008.
[38] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr, "Struck: Structured output tracking with kernels," *TPAMI*, vol. 38, no. 10, pp. 2096–2109, 2016.
[39] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *IJCV*, vol. 77, no. 1-3, pp. 125–141, 2008.
[40] J. Kwon and K. M. Lee, "Visual tracking decomposition," in *CVPR*, 2010.
[41] X. Mei and H. Ling, "Robust visual tracking using ℓ_1 minimization," in *ICCV*, 2009.
[42] C. Bao, Y. Wu, H. Ling, and H. Ji, "Real time robust l1 tracker using accelerated proximal gradient approach," in *CVPR*, 2012.
[43] T. Zhang, S. Liu, C. Xu, S. Yan, B. Ghanem, N. Ahuja, and M.-H. Yang, "Structural sparse tracking," in *CVPR*, 2015.
[44] H. Fan and J. Xiang, "Robust visual tracking with multitask joint dictionary learning," *TCSVT*, vol. 27, no. 5, pp. 1018–1030, 2017.
[45] Y. Li and J. Zhu, "A scale adaptive kernel correlation filter tracker with feature integration," in *ECCV Workshop*, 2014.
[46] J. Valmadre, L. Bertinetto, J. F. Henriques, A. Vedaldi, and P. H. Torr, "End-to-end representation learning for correlation filter based tracking," in *CVPR*, 2017.
[47] T. Liu, G. Wang, and Q. Yang, "Real-time part-based visual tracking via adaptive correlation filters," in *CVPR*, 2015.
[48] H. Fan and J. Xiang, "Robust visual tracking via local-global correlation filter," in *AAAI*, 2017.
[49] C. Ma, X. Yang, C. Zhang, and M.-H. Yang, "Long-term correlation tracking," in *CVPR*, 2015.
[50] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, "Learning spatially regularized correlation filters for visual tracking," in *ICCV*, 2015.
[51] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *TPAMI*, vol. 34, no. 7, pp. 1409–1422, 2012.
[52] Y. Hua, K. Alahari, and C. Schmid, "Occlusion and motion reasoning for long-term tracking," in *ECCV*, 2014.
[53] J. H. Yoon, D. Y. Kim, and K.-J. Yoon, "Visual tracking via adaptive tracker selection with multiple features," in *ECCV*, 2012.
[54] N. Wang and D.-Y. Yeung, "Ensemble-based tracking: Aggregating crowdsourced structured time series data," in *ICML*, 2014.
[55] J. Zhang, S. Ma, and S. Sclaroff, "Meem: robust tracking via multiple experts using entropy minimization," in *ECCV*, 2014.
[56] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "Prost: Parallel robust online simple tracking," in *CVPR*, 2010.
[57] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, "Multi-store tracker (MUSTER): A cognitive psychology inspired approach to object tracking," in *CVPR*, 2015.
[58] R. Girshick, "Fast R-CNN," in *ICCV*, 2015.
[59] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *TPAMI*, vol. 24, no. 7, pp. 881–892, 2002.
[60] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM MM*, 2014.
[61] J. Choi, H. Jin Chang, J. Jeong, Y. Demiris, and J. Young Choi, "Visual tracking using attention-modulated disintegration and integration," in *CVPR*, 2016.
[62] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *ICCV Workshops*, 2015.
[63] H. Nam, M. Baek, and B. Han, "Modeling and propagating cnns in a tree structure for visual tracking," *arXiv*, 2016.
[64] G. Zhu, F. Porikli, and H. Li, "Beyond local search: Tracking objects everywhere with instance-specific proposals," in *CVPR*, 2016.
[65] Z. Chi, H. Li, H. Lu, and M.-H. Yang, "Dual deep network for visual tracking," *TIP*, vol. 26, no. 4, pp. 2005–2015, 2017.

Heng Fan received his B.E. degree in College of Science, Huazhong Agricultural University (HZAU), Wuhan, China, in 2013. He is currently a Ph.D. student in the Department of Computer and Information Science, Temple University, Philadelphia, USA. His research interests include computer vision, pattern recognition and machine learning.

Haibin Ling received the BS and MS degrees from Peking University, China, in 1997 and 2000, respectively, and the PhD degree from the University of Maryland College Park in 2006. From 2000 to 2001, he was an assistant researcher at Microsoft Research Asia. From 2006 to 2007, he worked as a postdoctoral scientist at the University of California Los Angeles. After that, he joined Siemens Corporate Research as a research scientist. Since fall 2008, he has been with Temple University where he is now an Associate Professor. He received the Best Student Paper Award at the ACM UIST in 2003, and the NSF CAREER Award in 2014. He serves as associate editors for *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *Pattern Recognition*, and *Computer Vision and Image Understanding*, and has served or will serve as area chairs for CVPR 2014, CVPR 2016 and CVPR 2019.