Parallel Tracking and Verifying: A Framework for Real-Time and High Accuracy Visual Tracking

Heng Fan   Haibin Ling[*]
Meitu HiScene Lab, HiScene Information Technologies, Shanghai, China
Department of Computer and Information Sciences, Temple University, Philadelphia, PA USA
{hengfan,hbling}@temple.edu

Abstract

*Being intensively studied, visual tracking has seen great recent advances in either speed (*e.g.*, with correlation filters) or accuracy (*e.g.*, with deep features). Real-time and high accuracy tracking algorithms, however, 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 from the success of* parallel tracking and mapping *in visual SLAM. Our PTAV framework typically consists of two components, a tracker $\mathcal{T}$ and a verifier $\mathcal{V}$, working in parallel on two separate threads. The tracker $\mathcal{T}$ aims to provide a super real-time tracking inference and is expected to perform well most of the time; by contrast, the verifier $\mathcal{V}$ checks 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 the high efficiency provided by $\mathcal{T}$ and the strong discriminative power by $\mathcal{V}$. In our extensive experiments on popular benchmarks including OTB2013, OTB2015, TC128 and UAV20L, PTAV achieves the best tracking accuracy among all real-time trackers, and in fact performs even better than many deep learning based solutions. Moreover, as a general framework, PTAV is very flexible and has great rooms for improvement and generalization.*

1. Introduction

1.1. Background

Visual object tracking plays a crucial role in computer vision and has a variety of applications such as robotics, visual surveillance, human-computer interaction and so forth [36, 41]. Despite great successes in recent decades, robust visual tracking still remains challenging due to many factors including object deformation, occlusion, rotation, illumination change, pose variation, etc. An emerging trend toward improving tracking accuracy is to use deep learning-based techniques (e.g., [9, 28, 33, 38]) for their strong discriminative power, as shown in [33]. Such algorithms, unfortunately, often suffer from high computational burden and hardly run in real-time (see Fig. 1).

[*]Corresponding author.

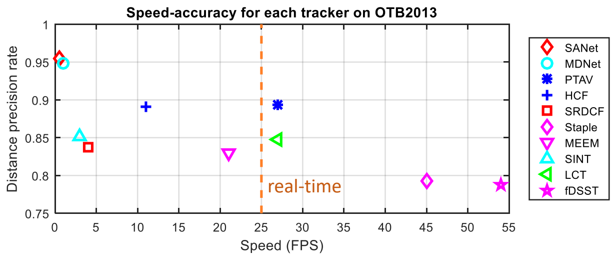


Figure 1. Speed and accuracy plot of state-of-the-art visual trackers on OTB2013 [39]. For better illustration, only those trackers with accuracy higher than 0.75 are reported. The proposed PTAV algorithm achieves the best accuracy among all real-time trackers.

Along a somewhat orthogonal direction, researchers have been proposing efficient visual trackers (e.g., [5, 7, 16, 43]), represented by the series of trackers based on correlation filters. While easily running at real-time, these trackers are usually less robust than deep learning-based approaches.

Despite the above mentioned progresses in either speed or accuracy, real-time high quality tracking algorithms remain scarce. A natural way is to seek a trade-off between speed and accuracy, *e.g.*, [3, 29]. In this paper we work toward this goal, but from a novel perspective as following.

1.2. 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. Our work is inspired by the following observations or related works:

Motivation 1. When tracking a target from visual input,

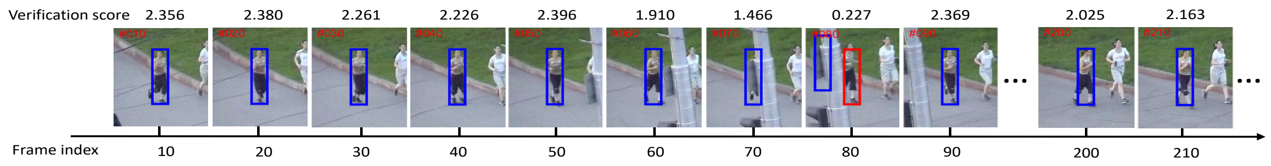


Figure 2. Verifying tracking results on a typical sequence. Verifier validates tracking results every 10 frames. Most of the time the tracking results are reliable (showing in blue). Occasionally, *e.g.* frame #080, the verifier finds the original tracking result (showing in blue) unreliable and the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

most of the time the target moves smoothly and its appearance changes little or slowly. Simple but efficient algorithms usually work fine for such easy cases. By contrast, hard cases appear only occasionally, though may cause serious consequences if not addressed properly. These hard cases typically require to be handled by computationally expensive operations, which are called verifiers in our study. These verifiers, intuitively, are needed only occasionally instead of for every frame, as shown in Fig. 2.

Motivation 2. The ubiquity of multi-thread computing has already benefited computer vision systems, with notably in visual SLAM (*simultaneous localization and mapping*). By splitting tracking and mapping into two parallel threads, PTAM (*parallel tracking and mapping*) [23] provides one of the most popular SLAM frameworks with many important extensions (*e.g.*, [32]). A key inspiration in PTAM is that mapping is not needed for every frame; nor does verifying in our task.

Motivation 3. Last but not least, recent advances in fast or accurate tracking algorithms provide promising building blocks and encourage us to seek a combined solution.

1.3. Contribution

With the motivations listed above, we propose to build real-time high accuracy trackers in a novel framework named *parallel tracking and verifying* (PTAV). PTAV typically consists of two components: a fast tracker[1] denoted by $\mathcal{T}$ and a verifier denoted by $\mathcal{V}$. The two components work in parallel on two separate threads while collaborate with each other. The tracker $\mathcal{T}$ aims to provide a super real-time tracking inference that is expected to perform well most of the time, *e.g.*, most frames in Fig. 2. By contrast, the verifier $\mathcal{V}$ checks the tracking results and corrects $\mathcal{T}$ when needed, *e.g.*, at frame #080 in Fig. 2.

The key idea is, while $\mathcal{T}$ needs to run on every frame, $\mathcal{V}$ does not. As a general framework, PTAV allows the coordination between the tracker and the verifier: $\mathcal{V}$ checks the tracking results provided by $\mathcal{T}$ and sends feedback to $\mathcal{V}$; and $\mathcal{V}$ adjusts itself according to the feedback when necessary. By running $\mathcal{T}$ and $\mathcal{V}$ in parallel, PTAV inherits both the high efficiency of $\mathcal{T}$ and the strong discriminative power of $\mathcal{V}$.

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

Implementing[2] a PTAV algorithm needs three parts: a base tracker for $\mathcal{T}$, a base verifier for $\mathcal{V}$, and the coordination between them. For $\mathcal{T}$, we choose the *fast discriminative scale space tracking* (fDSST) algorithm [7], which is correlation filter-based and runs efficiently by itself. For $\mathcal{V}$, we choose the Siamese networks [6] for verification similar to [37]. For coordination, $\mathcal{T}$ sends results to $\mathcal{V}$ at a certain frequency that allows enough time for verification. On the verifier side, when an unreliable result is found, $\mathcal{V}$ performs detection and sends the detected result to $\mathcal{T}$ for correction. The framework is illustrated in Fig. 3 and detailed in Sec. 3.

The proposed PTAV algorithm is evaluated thoroughly on several benchmarks including OTB2013 [39], OTB2015 [40], TC128 [26] and UAV20L [31]. In these experiments, PTAV achieves the best tracking accuracy among all real-time trackers, and in fact even performs even better than many deep learning based solutions.

In summary, our first main contribution is the novel parallel tracking and verifying framework (*i.e.* PTAV). With the framework, we made a second contribution to implement a tracking solution that combines correlation kernel-based tracking and deep learning-based verification. Then, our solution shows very promising results on thorough experiments in comparison with state-of-the-arts. Moreover, it is worth noting that PTAV is a very flexible framework and our implementation is far from optimal. We believe there are great rooms for future improvement and generalization.

2. Related Work

Visual tracking has been extensively studied with a huge amount of literature. In the following we discuss the most related work and refer readers to [36,41] for recent surveys.

Related tracking algorithms. Existing model free visual tracking algorithms are often categorized as either discriminative or generative. Discriminative algorithms usually treat tracking as a classification problem that distinguishes the target from ever-changing background. The classifiers in these methods are learned by, *e.g.*, multiple instance learning (MIL) [1], compressive sensing [43], P-N learning [21], structured output SVMs [14], on-line boosting [13] and so on. By contrast, generative trackers usually formulate track-

[2]The source code of our implementation is shared at `http://www.dabi.temple.edu/~hbling/code/PTAV/ptav.htm`.

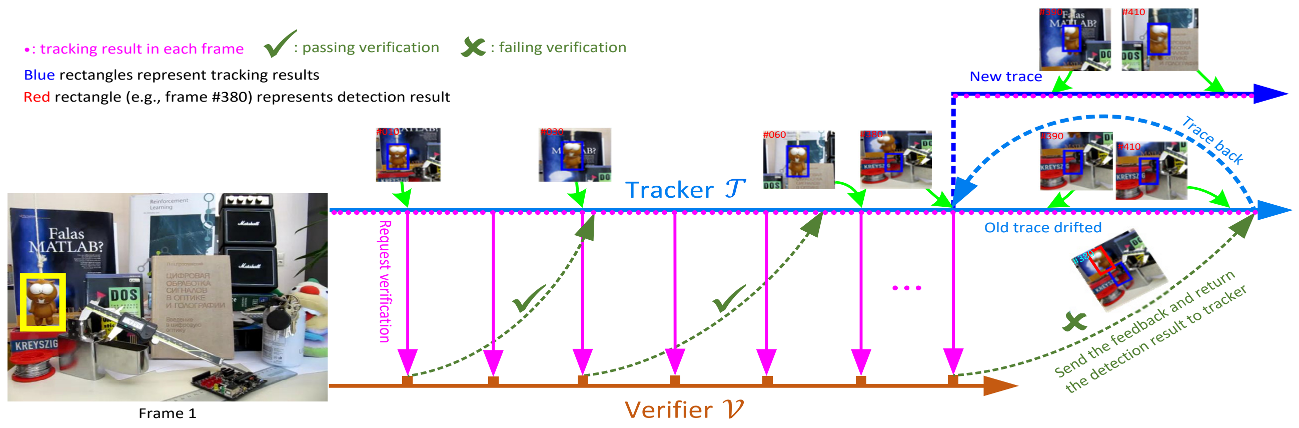

•: tracking result in each frame
✓: passing verification
✗: failing verification
Blue rectangles represent tracking results
Red rectangle (e.g., frame #380) represents detection result
Frame 1
#010
#030
#060
#380
#390
#410
Tracker $\mathcal{T}$
Request verification
New trace
Trace back
Old trace drifted
Send the feedback and return the detection result to tracker
Verifier $\mathcal{V}$

Figure 3. Illustration of the PTAV framework in which tracking and verifying are processed in two parallel asynchronous threads.

ing as searching for regions most similar to the target. To this end, various object appearance modeling approaches have been proposed such as incremental subspace learning [34] and sparse representation [2,10,30]. Inspired by the powerfulness of deep features in visual recognition [24,35], some trackers [9, 28, 33, 38] utilize the deep features for robust object appearance modeling.

Recently, correlation filters have drawn increasing attention in visual tracking. Bolme *et al.* [5] propose a correlation filter tracker via learning the minimum output sum of squared error (MOSSE). Benefitting from the high computational efficiency of correlation filters, MOSSE achieves an amazing speed at hundreds of *fps*. Henriques *et al.* [15] introduce kernel space into correlation filter and propose a circulant structure with kernel (CSK) method for tracking. CSK is then extended in [16] for further improvement and result in the well-known kernelized correlation filters (KCF) tracker. Later, [7] and [25] propose to improve KCF by imbedding scale estimations into correlation filters. More efforts on improving KCF can be found in [27, 29], etc.

Verification in tracking. The idea of verification is not new in tracking. A notable example is the TLD tracker [21], in which tracking results are validated *per frame* to decide how learning and/or detection shall progress. Similar ideas have been used in other trackers such as [18, 19]. Unlike in previous studies, the verification in PTAV runs only on sampled frames. This allows PTAV to use strong verification algorithms (Siamese networks [6] in our implementation) without worrying about running time efficiency.

Interestingly, tracking by itself can be also formulated as a verification problem that finds the best candidate similar to the target [4, 37]. Bertinetto *et al.* [4] propose a fully-convolutional Siamese networks for visual tracking. In [37], Tao *et al.* formulate tracking as object matching in each frame by Siamese networks. Despite obtaining excellent performance, application of such trackers is severely restricted by the heavy computation for extracting deep features in each frame. By contrast, our solution only treats verification as a way to check and correct the *fast tracker*, and does not run verification per frame.

3. Parallel Tracking and Verifying (PTAV)

3.1. Framework

A typical 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 initialization), and then provide feedback to $\mathcal{T}$.

In PTAV, $\mathcal{T}$ and $\mathcal{V}$ run in parallel on two different threads with necessary interactions, as illustrated in Fig. 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 result (pink solid dot in Figure 3). In the meantime, $\mathcal{V}$ validates the tracking result every several frames. Because tracking is much faster verifying, $\mathcal{T}$ and $\mathcal{V}$ work asynchronously. Such mechanism allows PTAV to tolerate temporal tracking drift (*e.g.*, at frame 380 in Figure 3), which will be corrected later by $\mathcal{V}$. When finding the tracking result unreliable, $\mathcal{V}$ searches the correct answer from a local region and sends it to $\mathcal{T}$. Upon the receipt of such

Algorithm 1: Parallel Tracking and Verifying (PTAV)

```
1 Initialize the tracking thread for tracker $\mathcal{T}$;
2 Initialize the verifying thread for verifier $\mathcal{V}$;
3 Run $\mathcal{T}$ (Alg. 2) and $\mathcal{V}$ (Alg. 3) till the end of tracking;
```

feedback, $\mathcal{T}$ stops current tracking job and traces back to resume tracking with the correction provided by $\mathcal{V}$.

It is worth noting that PTAV is a very flexible framework, and some important designing choices are following. (1) The choices of base algorithms for $\mathcal{T}$ and $\mathcal{V}$ may depend on 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, either positive or negative, from $\mathcal{V}$ can be largely designed. (3) The correction of unreliable tracking results can be implemented in many ways, and the correction can even be conducted purely by $\mathcal{T}$ (*i.e.*, including target detection). (4) $\mathcal{T}$ has various ways to use pre-computed archived information for speeding up. Algorithm 1 summarizes the general PTAV framework.

3.2. PTAV Implementation

3.2.1 Tracking

We choose the fDSST tracker [7] as the base of the tracker $\mathcal{T}$ in PTAV. As a discriminative correlation filter-based tracker, fDSST learns a model on an image patch f with a d-dimension feature vector. Let f^l denote the feature along the l-th dimension, $l \in 1, 2, \cdots, d$. The objective is to learn the optimal correlation filter h, consisting of one filter h^l per feature dimension, by minimizing the cost function

$$\epsilon(f) = \left\| \sum\nolimits_{l=1}^{d} h^l * f^l - g \right\|^2 + \lambda \sum\nolimits_{l=1}^{d} \| h^l \|^2 \qquad (1)$$

where g represents the desired correlation output associated with the training patch f, λ ($\lambda \geqslant 0$) denotes a tradeoff parameter, and $*$ is circular convolution operation.

Using the fast Fourier transformation (FFT), the solution of Eq. (1) can be efficiently obtained with

$$H^l = \frac{\bar{G} F^l}{\sum_{k=1}^{d} \bar{F}^k F^k + \lambda}, \qquad l = 1, 2, \cdots, d \qquad (2)$$

where the capital letters in Eq. (2) represent the discrete Fourier transform (DFT) of the corresponding quantities, and the bar $(\bar{\cdot})$ indicates complex conjugation.

An optimal filter can be derived by minimizing the output error over all training samples [22]. Nevertheless, this requires solving a $d \times d$ linear system of equations per pixel, leading to high computation. For efficiency, a simple linear update strategy is applied to the numerator A_t^l and denominator B_t of H_t^l by

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

```
1  while Current frame is valid do
2      if received a message s from $\mathcal{V}$ then
3          if verification passed then
4              Update tracking model (optional);
5          else
6              Correct tracking;
7              Trace back and reset current frame;
8              Resume tracking;
9          end
10     else
11         Tracking on the current frame;
12         if time for verification then
13             Send the current result to $\mathcal{V}$ to verify;
14         end
15     end
16     Current frame $\leftarrow$ next frame;
17 end
```

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

```
1 while not ended do
2     if received request from $\mathcal{T}$ then
3         Verifying the tracking result;
4         if verification failed then
5             Provide correction information;
6         end
7         Send verification result $s$ to $\mathcal{T}$;
8     end
9 end
```

$$\begin{array}{rcl} A_t^l & = & (1-\eta) A_{t-1}^l + \eta \bar{G}_t F_t^l \\ B_t & = & (1-\eta) B_{t-1} + \eta \sum_{k=1}^{d} \bar{F}_t^k F_t^k \end{array} \qquad (3)$$

where η is the learning rate. The responding scores y for a new image patch z can be computed by

$$y = \mathcal{F}^{-1} \left\{ \frac{\sum_{l=1}^{d} \bar{A}^l Z^l}{B + \lambda} \right\} \qquad (4)$$

The position of target object is determined by the location of maximal value of y.

To adapt the tracker to scale variation, a scale filter is adopted to estimate the scale of target. In addition, to further decrease computation, principal component analysis (PCA) is utilized to reduce the dimension of feature vector. For more details, readers are referred to [7].

Unlike in [7], in our implementation the tracker $\mathcal{T}$ stores all intermediate results (*e.g.* H_t^l in each frame t) since sending out last verification request to ensure fast tracing back. To validate the tracking result, $\mathcal{T}$ sends the verification results every V frames, where V is the dynamically adjustable verification interval as described later.

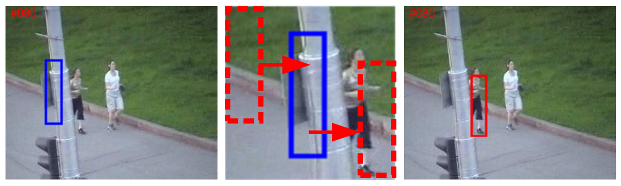


Figure 4. Detection based on verification. When finding an unreliable tracking result (showing in blue in (a)), the verifier $\mathcal{V}$ detects the target in a local region (shown in (b)). The dashed red rectangles in (b) represent object candidates generated by sliding window. The red rectangle in (c) is the detection result.

3.2.2 Verifying

We adopt the siamese networks [6] to develop the verifier $\mathcal{V}$. The siamese networks[3] contain two branches of CNNs, and process two inputs separately. In this work, we borrow the architecture from VGGNet [35] for the CNNs, but with an additional region pooling layer [12]. This is because, for detection, $\mathcal{V}$ needs to process multiple regions in an image, from which the candidate most similar to the target is selected to be result. As a result, region pooling enables us to simultaneously process a set of regions in an image.

Given the tracking result from $\mathcal{T}$, if its verification score is lower than a threshold τ_1, $\mathcal{V}$ will treat it as a tracking failure. In this case, $\mathcal{V}$ needs detect target, again using the siamese networks. Unlike for verification, detection requires to verify multiple image patches in a local region[4] and finds the best one. Thanks to the region pooling layer, these candidates can be simultaneously processed in only one pass, resulting in significant reduction in computation. Let $\{c_i\}_{i=1}^{N}$ denote the candidate set generated by sliding window, and the detection result $\widehat{c}$ is determined by

$$\widehat{c} = \underset{c_i}{\operatorname{argmax}}\, \nu(x_{obj}, c_i), \quad i = 1, 2, \cdots, N \tag{5}$$

where $\nu(x_{obj}, c_i)$ returns the verification score between the tracking target x_{obj} and the candidate c_i.

After obtaining detection result, we determine whether or not take it to be an alternative for tracking result based on its verification score. If detection result is unreliable (e.g., the verification score for detection result is less than a threshold τ_2), we do not change the tracking result. Instead, we decrease the verifying interval V, and increase the size of local region to search for the target. The process repeats until we find a reliable detection result. Then we restore verification interval and the size of the searching region. Figure 4 shows the detection process.

[3]Due to page limitation, we refer readers to the supplementary material for detailed architecture of the siamese networks and its training process.

[4]The local region is a square of size $\beta(\mathrm{w}^2 + \mathrm{h}^2)^{\frac{1}{2}}$ centered at the location of the tracking result in this validation frame, where w and h are the width and height of the tracking result, and β controls the scale.

4. Experiments

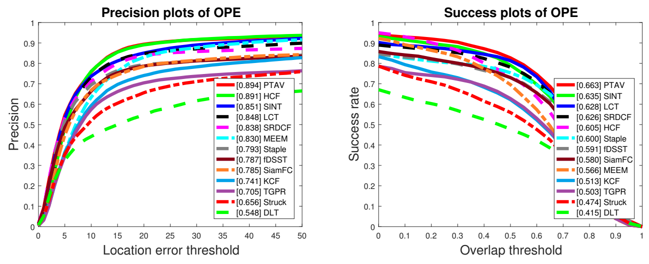
4.1. Implementation details

Our PTAV is implemented in C++ and its verifier uses Caffe [20] on a single NVIDIA GTX TITAN Z GPU with 6GB memory. The regularization term λ in Eq. (1) is set to 0.01, and the learning rate in Eq. (3) to 0.025. Other parameters for tracking remain the same as in [7]. The siamese networks for verification are initialized with VGGNet [35] and trained based on the approach in [37]. The verification interval V is initially set to 10. The validation and detection thresholds τ_1 and τ_2 are set to 1.0 and 1.6, respectively. The parameter β is initialized to 1.5, and is adaptively adjusted based on the detection result. If the detection result with $\beta = 1.5$ is not reliable, the verifier will increase β for a larger searching region. When the new detection becomes faithful, β is restored to 1.5.

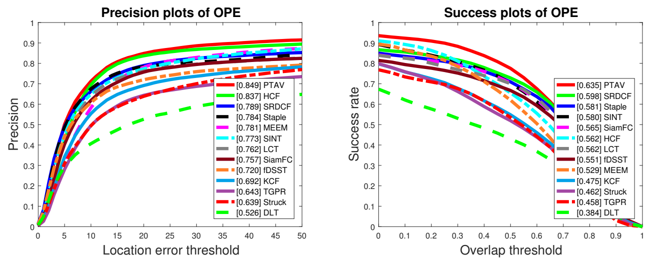
4.2. Experiments on OTB2013 and OTB2015

Overall performance. OTB2013 [39] and OTB2015 [40] are two popular tracking benchmarks, which contain 50 and 100 videos, respectively. We evaluate the PTAV on these benchmarks in comparison with 11 state-of-the-art trackers from three typical categories: (i) deep features-based trackers, including SINT [37], HCF [28], SiamFC [4] and DLT [38]; (ii) correlation filters-based tracking approaches, including fDSST [7], LCT [29], SRDCF [8], KCF [16] and Staple [3]; and (iii) other representative tracking methods, including TGPR [11], MEEM [42] and Struck [14]. For SINT [37], we use its tracking results without optical flow because no optical flow part is provide from the released source code. In PTAV, the fDSST tracker [7] is chosen to be our tracking part, and thus it can be regarded as our baseline. It is worth noting that other tracking algorithms may also be used for tracking part in our PTAV.

Following the protocol in [39, 40], we report the results in *one-pass evaluation* (OPE) using *distance precision rate* (DPR) and *overlap success rate* (OSR) as shown in Fig. 5. Overall, PTAV performs favorably against all other state-of-the-art trackers on both datasets. In addition, we present quantitative comparison of DPR at 20 pixels, OVR at 0.5, and speed in Table 1. It shows that PTAV outperforms other state-of-the-art trackers in both rates. On OTB2015, our tracker achieves a DPR of 84.9% and an OVR of 77.6%. Though the HCF [28] utilizes deep features to represent object appearance, our approach performs better compared with its DPR of 83.7% and OSR of 65.6%. Besides, owing to the adoption of parallel framework, PTAV (27 fps) is more than twice faster than HCF (10 fps). Compared with SINT [37], PTAV improves DPR from 77.3% to 84.9% and OSR from 70.3% to 77.6%. In addition, PTAV runs at real-time while SINT does not. Compared with the baseline fDSST [7], PTAV achieves significant improvements



(a) Comparisons on OTB2013



(b) Comparisons on OTB2015

Figure 5. Comparison on OTB2013 and OTB2015 using distance precision rate (DPR) and overlap success rate (OSR).

Table 1. Comparisons with state-of-the-art tracking methods on OTB2013 [39] and OTB2015 [40]. Our PTAV outperforms existing approaches in distance precision rate (DPR) at a threshold of 20 pixels and overlap success rate (OSR) at an overlap threshold of 0.5.

			Deep trackers				Correlation-filters based trackers					Representative trackers		
		PTAV (Ours)	HCF [28]	SINT [37]	DLT [38]	SiamFC [4]	SRDCF [8]	Staple [3]	LCT [29]	fDSST [7]	KCF [16]	MEEM [42]	TGPR [11]	Struck [14]
OTB2013	DPR (%)	89.4	89.1	85.1	54.8	78.5	83.8	79.3	84.8	78.7	74.1	83	70.5	65.6
	OSR (%)	82.7	74	79.1	47.8	74.0	78.2	75.4	81.2	74.7	62.2	69.6	62.8	55.9
	Speed (*fps*)	27	11	3	9	46	4	45	27	54	245	21	1	10
OTB2015	DPR (%)	84.9	83.7	77.3	52.6	75.7	78.9	78.4	76.2	72	69.2	78.1	64.3	63.9
	OSR (%)	77.6	65.6	70.3	43	70.9	72.9	70.9	70.1	67.6	54.8	62.2	53.5	51.6
	Speed (*fps*)	25	10	2	8	43	4	43	25	51	243	21	1	10

Table 2. Average precision and success scores of PTAV and other five top trackers on different attributes: background cluttered (BC), deformation (DEF), fast motion (FM), in-plane rotation (IPR), illumination variation (IV), low resolution (LR), motion blur (MB), occlusion (OCC), out-of-plane rotation (OPR), out-of-view (OV) and scale variation (SV).

	Distance precision rate (%) on eleven attributes						Overlap success rate (%) on eleven attributes					
Attribute	PTAV	HCF [28]	SRDCF [8]	Staple [3]	MEEM [42]	SINT [37]	PTAV	HCF [28]	SRDCF [8]	Staple [3]	MEEM [42]	SINT [37]
BC	87.9	84.7	77.6	77.0	75.1	75.1	64.9	58.7	58.0	57.4	52.1	56.7
DEF	81.3	79.1	73.4	74.8	75.4	75.0	59.7	53.0	54.4	55.4	48.9	55.5
FM	77.7	79.7	76.8	70.3	73.4	72.5	60.8	55.5	59.9	54.1	52.8	55.7
IPR	83.0	85.4	74.5	77.0	79.3	81.1	60.7	55.9	54.4	55.2	52.8	58.5
IV	86.0	81.7	79.2	79.1	74.0	80.9	64.3	54.0	61.3	59.8	51.7	61.8
LR	78.9	78.7	63.1	60.9	60.5	78.8	56.3	42.4	48.0	41.1	35.5	53.9
MB	81.0	79.7	78.2	72.6	72.1	72.8	62.9	57.3	61.0	55.8	54.3	57.4
OCC	83.2	76.7	73.5	72.6	74.1	73.1	62.3	52.5	55.9	54.8	50.3	55.8
OPR	82.8	81.0	74.6	74.2	79.8	79.4	61.1	53.7	55.3	53.8	52.8	58.6
OV	73.6	67.7	59.7	66.1	68.3	72.5	57.0	47.4	46.0	48.1	48.4	55.9
SV	79.7	80.2	74.9	73.1	74.0	74.2	59.0	48.8	56.5	52.9	47.3	55.8
Overall	84.9	83.7	78.9	78.4	78.1	77.3	63.5	56.2	59.8	58.1	52.9	58.0

on both DPR (12.9%) and OSR (10.0%).

Attribute-based evaluation. We further analyze the performance of PTAV under different attributes in OTB2015 [40]. Table 2 shows the comparison of PTAV with other top five tracking algorithms on these eleven attributes. In terms of distance precision rates (DPR), PTAV achieves the best results under 8 out of 11 attributes. For the rest three (FM, IPR and SV), PTAV obtains competitive performances. Compared with other deep learning-based trackers [28, 37], PTAV can better locate the target object in videos. On the other hand, PTAV achieves the best results of overlap success rates (OVR) under all 11 attributes. Compared with correlation filters-based trackers [3, 8] and MEEM [42], PTAV performs more robust under occlusion, background cluttered and low resolution with the help of cooperation between tracker and verifier.

Qualitative evaluation. Figure 6 summarizes qualitative comparisons of PTAV with seven state-of-the-art trackers (HCF [28], SRDCF [8], Staple [3], MEEM [42], SINT [37], KCF [15] and fDSST [7]) on twelve sequences sampled form OTB2015 [40]. The correlation filters-based trackers (KCF [16], SRDCF [8], fDSST [7] and Staple [3]) perform well in sequences with deformation, illumination variation and partial occlusion (*Basketball*, *Bolt*, *Shaking* and *Panda*). However, when full occlusion happens (*Coke* and *Jogging-1*), they are prone to lose the target. HCF [28] uses deep features to represent object appearance, and can deal with these cases to some degree. Nevertheless, it still fails when

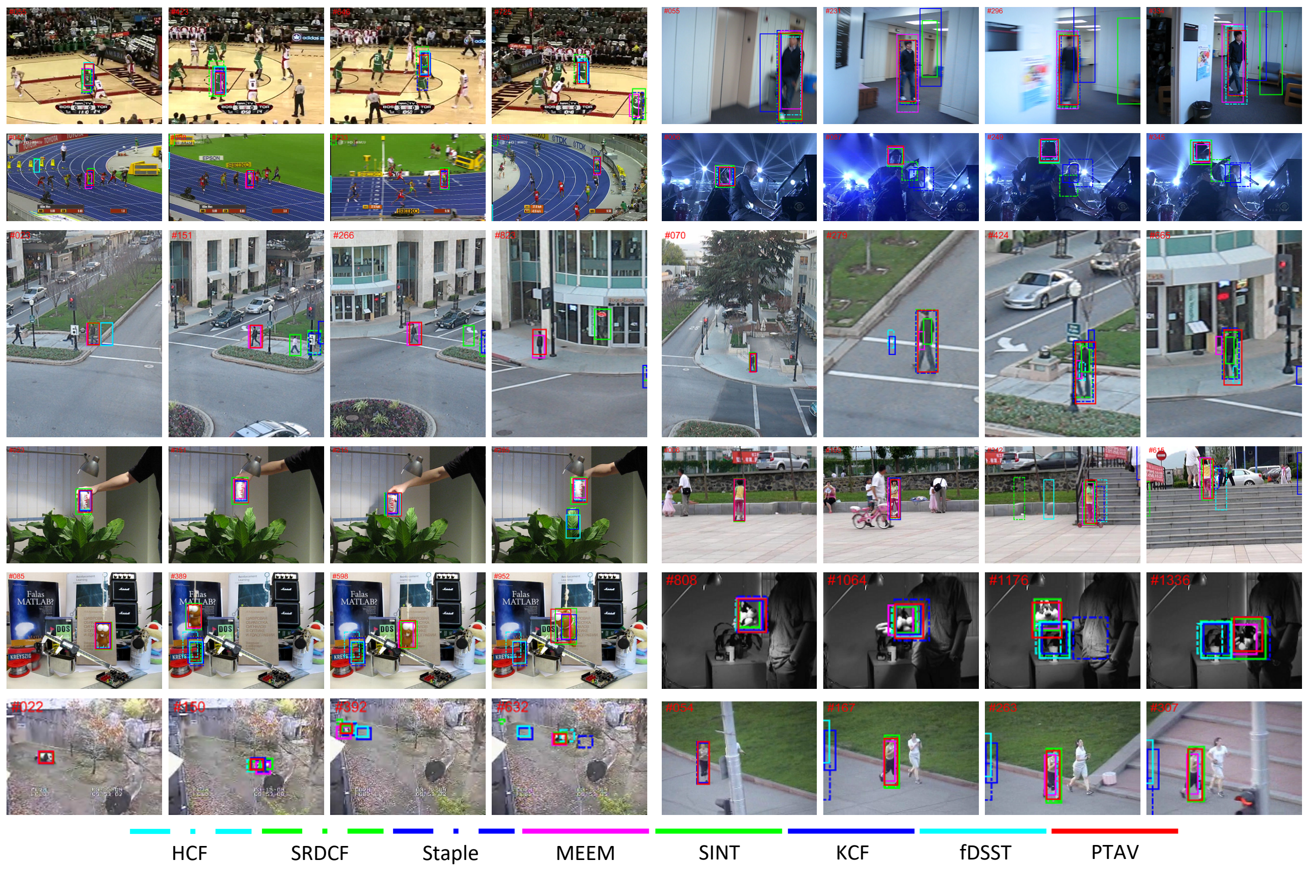


Figure 6. Qualitative evaluation of the proposed algorithm and other seven state-of-the-art trackers on twelve sequences (from left to right and top to bottom: *Basketball*, *BlurBody*, *Bolt*, *Shaking*, *Human3*, *Human6*, *Coke*, *Girl2*, *Lemming*, *Sylvester*, *Panda*, and *Jogging-1*.)

occlusion happens with other situations such as deformation and rotation (*Girl2*, *Human3*, *Human6*, *Sylvester* and *Lemming*).

Compared with these trackers, PTAV locate the target object more reliably. Even when experiencing a short drift, the verifier in PTAV can sense the drift and then detect the correct target for subsequent tracking. SINT [37] deals well with occlusion thanks to its capability in re-locating the target. However, it meets problems when motion blur occurs (*BlurBody*), which causes serious change in it extracted features. Different from SINT, PTAV uses a correlation filters based method for its tracking part, which works well for motion blur. MEEM [42] uses multiple classifier to track the target and works well in most cases. However, it may lose the target in presences of heavy occlusion and scale variations (*e.g.*, *Human6*).

4.3. Experiment on TC128

For experiments on the TC128 [26] dataset containing 128 videos, our PTAV runs at 21 frames per second. The comparison with state-of-the-art trackers (MEEM [42], HCF [28], LCT [29], fDSST [7], Struck [14], SRDCF [8], Staple [3], KCF [16]) is shown in Figure 7. Among the

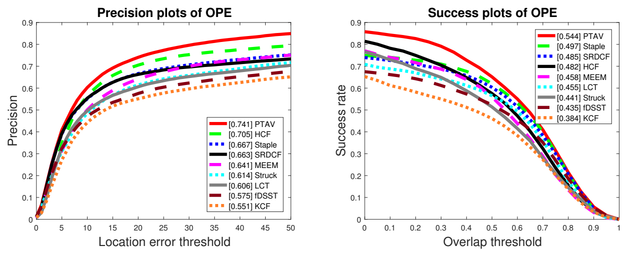


Figure 7. Comparison on TC128 using DPR and OSR.

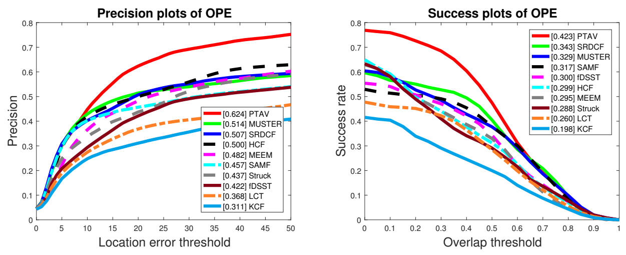


Figure 8. Comparison on UAV20L using DPR and OSR.

eight compared trackers, HCF [28] obtains the best distance precision rate (DPR) of 70.5% and Staple [3] achieves the best overlap success rate (OSR) of 49.7%. By comparison, PTAV improves the state-of-the-art methods on DPR to **74.1%** and OSR to **54.4%**, obtaining the gains of 3.6% and 4.7%, respectively.

Compared with fDSST [7], which obtains a DPR of 57.5% and an OSR of 43.5%, PTAV achieves significant improvements, showing clear the benefits of using a verifier. More results are left in the supplementary material.

4.4. Experiment on UAV20L

We also conduct experiment on the recently proposed UAV20L [31] dataset that contains 20 videos. The shortest video contains 1,717 frames, and the longest one contains 5,527 frames. Our PTAV runs at 25 frames per second. The comparison with state-of-the-art trackers (MUSTer [17], SRDCF [8], HCF [28], MEEM [42], SAMF [25], Struck [14], fDSST [7], LCT [29] and KCF [16]) is shown in Figure 8. PTAV achieves the best performance on both the distance precision rate (62.4%) and the success overlap rate (42.3%), outperforming other approaches by large margins (11% and 8% compared with the second best in the distance precision rate and the success overlap rate, respectively).

4.5. Detailed analysis of PTAV

Different verification interval V. In PTAV, different verification interval V may affect both the accuracy and efficiency. A smaller V means more frequent verification, which requires more computation and thus degrades the efficiency. A larger V, on the contrary, may cost less computation but may put PTAV at the risk when the target appearance change quickly. If the tracker loses the target object, it may update vast backgrounds in its appearance model until next verification. Even if the verifier re-locates the target and offers a correct detection result, the tracker may still lose it due to heavy changes of tracking appearance model. Table 3 shows sampled results with different V on OTB2013. Taking into account both accuracy and speed, we set V to 10 in our experiments.

Two threads v.s. one.[5] In PTAV, the tracker does not rely on verifier in most time. To improve efficiency, we use two separate threads to process tracking and verifying in parallel instead of a single thread to process all tasks in a line. As a consequence, the tracker does not have to wait for the feedback from verifier to process next frame, and it traces back and resumes tracking only when receiving the positive feedback from verifier. Owing to storing all intermediate status, the tracker is able to quickly trace back without any extra computation. Table 4 shows the comparison of speed between using two threads and using only a single thread. From Table 4, we can see that using two threads in parallel clearly improves the efficiency of system.

Table 3. Comparison of different V in DPR and speed.

	$V = 5$	$V = 10$	$V = 15$
DPR (%)	89.7	89.4	87.9
Speed (*fps*)	23	27	29

Table 4. Comparison on the tracking speed (*fps*).

Threads	OTB2013 [39]	OTB2015 [40]	TC128 [26]	UAV20L [31]
One	16	14	11	15
Two	27	25	21	25

Table 5. Comparison of different trackers in PTAV with $V = 10$.

		PTAV with fDSST	PTAV with KCF
OTB2013	DP (%)	89.4	80.4
	OS (%)	82.7	66.3
	Speed (*fps*)	27	24
OTB2015	DP (%)	84.9	73.5
	OS (%)	77.6	57.9
	Speed (*fps*)	25	21

Different tracker $\mathcal{T}$. 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 two different choices including fDSST [7] and KCF [16]. Compared to fDSST, KCF is more efficient while less accurate in short time interval. The comparison on OTB2013 and OTB2015 are provided in Table 5. It shows that PTAV with fDSST performs better and more efficiently than PTAV with KCF. Though KCF runs faster than fDSST, it performs less accurately in short time, which results in more requests for verifications and detections, hence significantly increasing the computation. By contrast, fDSST is more accurate in short time, and finally leads to efficiency in computation.

5. Conclusion

In this paper, we propose a new general tracking framework, named *parallel tracking and verifying* (PTAV), which decomposes object tracking into two sub-tasks, tracking and 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. The encouraging results are demonstrated in extensive experiments on four popular benchmarks. Since PTAV is a very flexible framework with great rooms for improvement and generalization, we expect this work to stimulate the designing of more efficient tracking algorithms in the future.

Acknowledgement. This work is supported by National Key Research and Development Plan (Grant No. 2016YFB1001200) and National Nature Science Foundation of China (no. 61528204).

[5]Some ensemble tracking methods can be implemented in multi-threads, *e.g.*, TLD [21], MUSTer [17] and LCT [29]). In such cases, different threads function similarly and almost independently. PTAV, by contrast, is fundamentally different. In PTAV, the tracking and verifying threads function very differently, and interact with each other frequently.

References

[1] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE TPAMI*, 33(8):1619–1632, 2011. 2

[2] C. Bao, Y. Wu, H. Ling, and H. Ji. Real time robust l1 tracker using accelerated proximal gradient approach. In *CVPR*, 2012. 3

[3] L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik, and P. H. Torr. Staple: Complementary learners for real-time tracking. In *CVPR*, 2016. 1, 5, 6, 7

[4] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *ECCV Workshop*, 2016. 3, 5, 6

[5] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *CVPR*, 2010. 1, 3

[6] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005. 2, 3, 5

[7] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Discriminative scale space tracking. *IEEE TPAMI*, 2016. 1, 2, 3, 4, 5, 6, 7, 8

[8] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *ICCV*, 2015. 5, 6, 7, 8

[9] H. Fan and H. Ling. SANet: Structure-aware network for visual tracking. In *CVPRW*, 2017. 1, 3

[10] H. Fan and J. Xiang. Robust visual tracking with multitask joint dictionary learning. *IEEE TCSVT*, 27(5):1018–1030, 2017. 3

[11] J. Gao, H. Ling, W. Hu, and J. Xing. Transfer learning based visual tracking with gaussian processes regression. In *ECCV*, 2014. 5, 6

[12] R. Girshick. Fast R-CNN. In *ICCV*, 2015. 5

[13] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *ECCV*, 2008. 2

[14] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr. Struck: Structured output tracking with kernels. *IEEE TPAMI*, 38(10):2096–2109, 2016. 2, 5, 6, 7, 8

[15] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *ECCV*, 2012. 3, 6

[16] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE TPAMI*, 37(3):583–596, 2015. 1, 3, 5, 6, 7, 8

[17] 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. 8

[18] Y. Hua, K. Alahari, and C. Schmid. Occlusion and motion reasoning for long-term tracking. In *ECCV*, 2014. 3

[19] Y. Hua, K. Alahari, and C. Schmid. Online object tracking with proposal selection. In *ICCV*, 2015. 3

[20] 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*. ACM, 2014. 5

[21] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE TPAMI*, 34(7):1409–1422, 2012. 2, 3, 8

[22] H. Kiani Galoogahi, T. Sim, and S. Lucey. Multi-channel correlation filters. In *ICCV*, 2013. 4

[23] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *ISMAR*, 2007. 2

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 3

[25] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *ECCV Workshop*, 2014. 3, 8

[26] P. Liang, E. Blasch, and H. Ling. Encoding color information for visual tracking: Algorithms and benchmark. *IEEE TIP*, 24(12):5630–5644, 2015. 2, 7, 8

[27] T. Liu, G. Wang, and Q. Yang. Real-time part-based visual tracking via adaptive correlation filters. In *CVPR*, 2015. 3

[28] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang. Hierarchical convolutional features for visual tracking. In *ICCV*, 2015. 1, 3, 5, 6, 7, 8

[29] C. Ma, X. Yang, C. Zhang, and M.-H. Yang. Long-term correlation tracking. In *CVPR*, 2015. 1, 3, 5, 6, 7, 8

[30] X. Mei and H. Ling. Robust visual tracking using ℓ_1 minimization. In *ICCV*, 2009. 3

[31] M. Mueller, N. Smith, and B. Ghanem. A benchmark and simulator for uav tracking. In *ECCV*, 2016. 2, 8

[32] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM: a versatile and accurate monocular slam system. *IEEE TR*, 31(5):1147–1163, 2015. 2

[33] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 1, 3

[34] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *IJCV*, 77(1-3), 2008. 3

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014. 3, 5

[36] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE TPAMI*, 36(7):1442–1468, 2014. 1, 2

[37] R. Tao, E. Gavves, and A. W. Smeulders. Siamese instance search for tracking. In *CVPR*, 2016. 2, 3, 5, 6, 7

[38] N. Wang and D.-Y. Yeung. Learning a deep compact image representation for visual tracking. In *NIPS*, 2013. 1, 3, 5, 6

[39] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *CVPR*, 2013. 1, 2, 5, 6, 8

[40] Y. Wu, J. Lim, and M.-H. Yang. Object tracking benchmark. *IEEE TPAMI*, 37(9):1834–1848, 2015. 2, 5, 6, 8

[41] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, 2006. 1, 2

[42] J. Zhang, S. Ma, and S. Sclaroff. Meem: robust tracking via multiple experts using entropy minimization. In *ECCV*, 2014. 5, 6, 7, 8

[43] K. Zhang, L. Zhang, and M.-H. Yang. Real-time compressive tracking. In *ECCV*, 2012. 1, 2