RTracker: Recoverable Tracking via PN Tree Structured Memory

Yuqing Huang[1,2], Xin Li[2,*], Zikun Zhou[2], Yaowei Wang[2], Zhenyu He[1,*], and Ming-Hsuan Yang[3,4]

[1]Harbin Institute of Technology, Shenzhen   [2]Peng Cheng Laboratory
[3]UC Merced   [4]Yonsei University

{domaingreen2, xinlihitsz, zhouzikunhit, minghsuanyang}@gmail.com,
wangyw@pcl.ac.cn, zhenyuhe@hit.edu.cn

Abstract

Existing tracking methods mainly focus on learning better target representation or developing more robust prediction models to improve tracking performance. While tracking performance has significantly improved, the target loss issue occurs frequently due to tracking failures, complete occlusion, or out-of-view situations. However, considerably less attention is paid to the self-recovery issue of tracking methods, which is crucial for practical applications. To this end, we propose a recoverable tracking framework, RTracker, that uses a tree-structured memory to dynamically associate a tracker and a detector to enable self-recovery ability. Specifically, we propose a Positive-Negative Tree-structured memory to chronologically store and maintain positive and negative target samples. Upon the PN tree memory, we develop corresponding walking rules for determining the state of the target and define a set of control flows to unite the tracker and the detector in different tracking scenarios. Our core idea is to use the support samples of positive and negative target categories to establish a relative distance-based criterion for a reliable assessment of target loss. The favorable performance in comparison against the state-of-the-art methods on numerous challenging benchmarks demonstrates the effectiveness of the proposed algorithm. All the source code and trained models will be released at https://github.com/NorahGreen/RTracker.

1. Introduction

Visual object tracking aims to estimate the location and extent of a target in a video sequence based on the bounding box annotation of the target given in the initial frame, which is a fundamental vision task with a wide range of applications, such as surveillance and autonomous navigation. The challenges in visual object tracking stem from
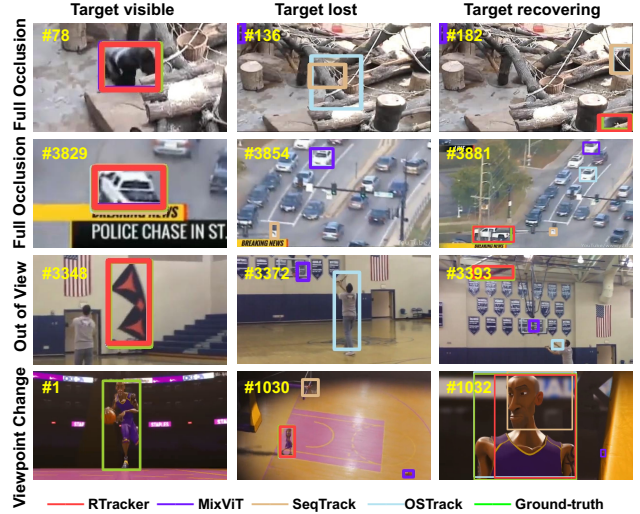

∗ corresponding author

Figure 1. **Performance on challenging sequences involving full occlusion, out-of-view, and viewpoint change.** The proposed RTracker can accurately re-track the targets in these sequences after their reappearance.

the dramatic variations of the target (*e.g.* rotation, deformation, and fast motion) and the various influences from the background (*e.g.* occlusion, illumination variation, and out-of-view) [35]. Existing tracking methods usually focus on learning robust target representation [4, 6, 21–23] or developing robust prediction models [2, 20, 30, 31, 41] to handle these tracking challenges and prevent target loss. However, target loss, which can be caused by full occlusion, out-of-view, or tracking failure, is usually inevitable, especially during long-term tracking in complex real-world application scenarios. Currently, considerably less attention is paid to the issue of how to recover tracking from target loss, which is becoming a bottleneck limiting the practical application of tracking algorithms.

Enabling self-recovery capability in a tracking model is challenging since it first needs to accurately determine

whether a target is present or absent and then provide the correct target information for re-initialization. A few tracking methods [25] assess the presence or absence of a target based on the similarity between its current appearance and that of reference from the initial frame using a fixed threshold. However, these may fail due to significant changes in target appearance over time and do not generalize well to different tracking scenarios. Earlier trackers [17] explore bolstering the self-recovery of trackers by employing online adaptation, which can adjust the tracker in response to any deviation in the appearance of the target given at the beginning of tracking. However, due to the substantial computational requirements and many training samples, performing effective online updates for deep learning tracking methods is impractical.

The key to addressing the above challenges lies in effectively modeling the continuously changing target over time, thus accurately determining the target presence/absence and re-initializing the tracking algorithm. Instead of judging target states based on fixed thresholds, our core idea is to construct a relative measurement grounded in the positive and negative support vector of the target. To enhance the reliability of target state evaluation, we should continuously refine the latest support vector to reflect the appearance changes and identify a closely matched complicated negative support vector for comparison.

To this end, we propose a recoverable tracking framework that dynamically associates detection and tracking via a specifically developed tree-structured memory to achieve tracking with self-recovery capability. Specifically, we construct the Positive-Negative Tree Memory (PN tree), which archives appearance features relevant to the target, maintained according to temporal tracking results, as well as background information as negative samples that closely resemble the appearance of the target. Upon the PN tree memory, we develop a series of walking rules to find the optimal support vector (target/background samples) for the current target and to ascertain its state (present or absent). With the state identified, we apply pre-defined associating processes tailored to various tracking scenarios for achieving self-recovery for the tracker after target loss. We conduct extensive experiments on a variety of challenging benchmarks, including VideoCube [15], LaSOT [10], LaSOT$_{ext}$ [11], TNL2K [32], and GOT-10k [16]. The favorable performance against the state-of-the-art methods on all the benchmarks demonstrates the effectiveness of our proposed recoverable tracking algorithm.

We make the following contributions in this work:

- We propose a novel tracking framework capable of self-recovery for the tracker after target loss. The proposed framework performs recoverable tracking guided by the established distinct control flows that integrate tracking and detection to manage the various states of tracking targets.
- We develop a Positive-Negative Tree Memory that stores and maintains positive and negative target samples over time. Additionally, we design a series of walking rules upon the PN tree to find the 'support vectors' for determining the target present/absent.
- We achieve state-of-the-art performance across numerous tracking benchmarks. Extensive experiments, including ablation studies, are conducted to demonstrate the effectiveness of our proposed method and the effect of each component.

2. Related Work

We discuss the closely related studies, including deep tracking methods, online adaptation trackers, and target search schemes for visual tracking.

Deep tracking methods. Deep tracking methods can be divided into Siamese-based and transformer-based categories based on the used backbone model. Siamese-based trackers [1, 5, 20, 30] first compute the correlated features between the reference image and the test image, then predict the target state upon the correlated features. In contrast, the transformer-based [3, 6, 22, 41] trackers use successive transformer blocks to model the relationship between the reference and test images, thus achieving a more comprehensive correlation between them. Siamese and transformer-based trackers only locate the tracked object in the current frame depending on the similarity between the reference and test images. This may result in tracking failure when the target is fully occluded or out of view. Unlike the above trackers, our approach develops a tree-structured memory to dynamically model the target appearance by maintaining positive and negative target samples.

Online adaptation trackers. The main tracking challenges are caused by the variations of both the target and the background over time. To combat that, several online adaptation techniques are developed, including online learning [18, 24], template updating [28], and memory networks [29, 40]. Online learning tracking methods [2, 9] continuously learn and model the targets throughout the tracking period, effectively adapting to changes in their appearance. However, these methods may cause high computational costs due to the depth of neural networks. Other online adaptation methods [14, 23, 28] mitigate this by periodically re-initializing the tracking template based on a comparison to the reference with a fixed threshold. Besides, several methods [13, 40] utilize dynamic memory networks to merge the initial template with historical tracking information for enhanced adaptability. However, previous online adaptation tracking methods depend on fixed thresholds for updates, which does not generalize well to different sequences. Our proposed method improves the accuracy in

determining the target state by comparing the relative distances between positive and negative samples stored in a PN tree memory structure, which adaptively saves target information and offers a more reliable adaptation.

Tracking aided by detection. To address the problem of target loss, several tracking methods [17, 27, 37] explore a global detector to aid tracking by detecting the target after its reappearance. Kalal *et al*. propose a tracking-learning-detection algorithm (TLD) [17] for handling target loss, which utilizes bidirectional optical flow matching as the local tracker and an online detector based on ensemble learning for global detection. TLD also employs a nearest neighbor classifier to determine the target state and then uses a learning module to associate the tracker and detector based on the target state. Ma *et al*. [25] use a discriminative correlation filter based on the histogram of orientation gradient features for local tracking and implement global detection using an online random fern classifier. Unlike these methods that mainly relied on a classifier to associate trackers and detectors, our proposed method includes constructing relative measurements based on positive and negative target samples to evaluate the target state and associate trackers and detectors more reliably.

3. Proposed Algorithm

The goal of our method is to dynamically associate a tracker with a detector to achieve recoverable tracking based on target states (*i.e.* presence or absence) over time, where the tracker accounts for precise target localization in successive frames and the detector accounts for the global searching. To this end, we propose a Positive-Negative Tree (PN tree) structured memory equipped with a set of walking rules to achieve a reliable determination of the target state. We then associate the tracker and detector using the pre-defined control flows based on the target states to perform tracking. Figure 3 depicts the overall flow of the proposed method, which includes the control flows for the normal case, the target missing case, and the target recovering case.

3.1. PN Tree Structured Memory

The primary issue in self-recovery tracking is to determine whether the target is present or missing in a test frame. Instead of judging target states based on fixed thresholds, our core idea is to construct a relative measurement based on positive and negative target samples to enable a more reliable assessment of the target states. The Positive-Negative Tree is constructed to maintain the 'support vectors' for positive and negative target samples over time, which is akin to the SVM algorithm. In addition, we propose a set of walking rules for exploiting the PN tree memory to determine the current target state. Figure 2 depicts the structure and updating operation of the proposed PN tree.
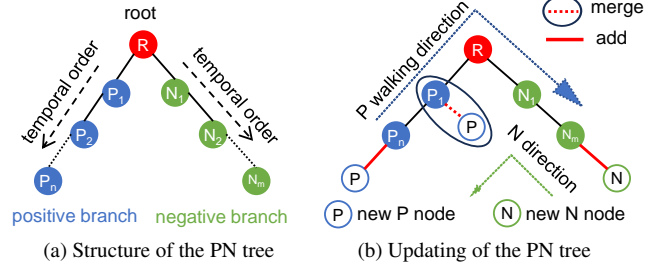
Figure 2. **Definition of the PN tree.** R, P, and N denote the root node, positive nodes, and negative nodes, respectively. New nodes are added to the tree through two ways: addition and merging. The walking paths are used for finding the support samples for target state determination.

PN tree definition. We define the PN tree as a specialized variant of the binary tree, characterized by a root node and two distinct branches: the positive and the negative. The root node stores the features of the target template given in the initial frame, and its two branches maintain the target-relevant information as the positive branch and the target-irrelevant samples as the negative one. Each node in the tree structure stores the features of a representative target sample at a specific stage extracted by a similarity perception model [12].

Operations of the PN tree. To meet the target modeling requirements of visual tracking, we define the initialization, update, and deletion operations of the PN tree. **Initialization.** The root and the first positive child node of the PN tree are initialized using the features of the target template given in the initial frame. Simultaneously, the initial negative node is constructed with target-irrelevant information (*i.e.* features of a background sample). **Update.** For an efficient PN tree memory, we adopt two different sample update strategies based on whether the target sample contains a new target appearance. For the updated sample without a new target appearance, we merge the new node with the existing child node (excluding the root) via Equation 1. After merging, we move the node to the deepest node in the positive branch. For the updated sample with a new target appearance, we directly append the new node as the deepest positive node in the PN tree. The merging process is formalized as follows:

$$F_{new} = (F_x + F_{old} \times N)/(N + 1), \qquad (1)$$

where F_{new} and F_{old} are the features of the node before and after the merge operation, N denotes the number of updates applied to this node, and F_x is the feature of the new sample to be updated. For the negative branch, we add the new negative sample as the deepest negative node. Hence, within the PN tree, the depth of a node serves as an indicator of its temporal updated order, where nodes situated at greater depths correspond to more recent updating. **Dele-**

tion. When a branch exceeds N nodes, the earliest node is deleted. This study sets N to 10, balancing memory efficiency and tracking accuracy.

Walking rules of the PN tree. Our key idea is to identify the type (positive or negative) of the test node based on its similarity to the nodes in the PN tree. To this end, we define the walking rules with three elements: walking operation, stopping condition, and walking path. **Walking operation.** For each node walked by, we compute its similarity against the test node using the cosine similarity defined as:

$$S = cos(F_x, F_{node}), \qquad (2)$$

where F_x and F_{node} denote the test node and the candidate node, respectively. **Walking path.** We define a positive path to identify whether the node transitions from a positive label to a negative label and a negative path to identify transitions from a negative to a positive label. The positive path goes from the leaf nodes of the positive branch up to the root node and then descends from the root node to the leaf nodes of the negative branch. The negative path goes in reverse. **Stopping conditions.** For the positive cases, the walking process stops when a node more similar to the test node than the root is encountered after walking through both positive and negative branches. The label of the test node is then determined based on the label of the branch where the walking stops. If such a node is not encountered, the label of the test node is determined as positive. For the negative path, the walking process continues until the entire route is completed, and the label of the test node is predicted as the label of the node with the highest similarity encountered along the entire path. The walking rules guarantee a reliable classification of nodes by employing relative similarity measurements between positive and negative nodes.

3.2. Associating Tracking and Detection

In this section, we use the proposed PN tree to predict the state of the target and dynamically associate the tracker and detector based on the target states. According to different tracking scenarios, we define three processes, including normal case flow, target missing flow, and target recovery flow to control the operation of the tracker and the detector, which is shown in Figure 3.

Target State Prediction. We use the PN tree, incorporating defined operations and walking rules, to model the target appearance and identify the target state. The PN tree models the tracking target by forming a node of the target using the extracted features based on the tracking or detection results. We then identify the type of the newly formed node and update it into the PN tree using the walking rules and update operation, respectively. A positive node signifies successful tracking (normal case), while a negative node indicates target loss (missing target case). We then handle these cases using the following processes.

Normal case. As shown in the top line of Figure 3, we only exploit the tracker to follow the target object. Figure 3 shows the process of the normal case in the top-left part. For each frame, we use the tracker to predict the location of the target object and use the PN tree to identify the state of the target. If the target is present, we continue tracking and update the PN tree memory with the new tracking results.

Missing target case. The tracker stops if the state prediction infers the target is missing within the search region, as shown in the top-right of Figure 3. We then activate the detector for a global search at this point. Concurrently, the latest tracking result, classified as a negative sample, is added to the PN tree memory as a new negative node. This step enriches the PN tree memory with the most recent information and improves state prediction by incorporating examples of failed tracking.

Recovering target case. For the target recovering flow, shown in the bottom part of Figure 3, the detection task continues until the target is detected in a frame. When the target reappears, the labels of the nodes corresponding to the successive frames will change from negative to positive. Therefore, we select the negative path to traverse the PN tree and determine the state of the target. Upon confirming the recovery of the target, we add the new target into the PN tree memory. Simultaneously, we deactivate the detector, activate the tracker, and provide the tracker with the location of the current target for the following tracking.

3.3. Recoverable Tracking

We illustrate the proposed recoverable tracking pipeline in Figure 3 and provide the pseudo-code in Algorithm 1. In the initial frame, we initialize the tracker, detector, and PN tree memory using the given target exemplar. For every following frame, we perform target state prediction and then handle different cases using the corresponding processes defined in Section 3.2 to perform tracking, detection, and updating dynamically.

4. Experiments

In this section, we present the experimental results of our proposed RTracker. We first compare the overall performance on five large-scale challenging tracking benchmarks against the state-of-the-art trackers. We then conduct a comprehensive ablation study to analyze the contribution of each component. A recovery ability evaluation is performed to demonstrate the effectiveness of our tracking method in successfully recovering the tracking target once it has been lost. Finally, the visualized results on several challenging sequences are provided to present an exhaustive qualitative analysis. More detailed results and experimental settings can be found in the supplemental materials.
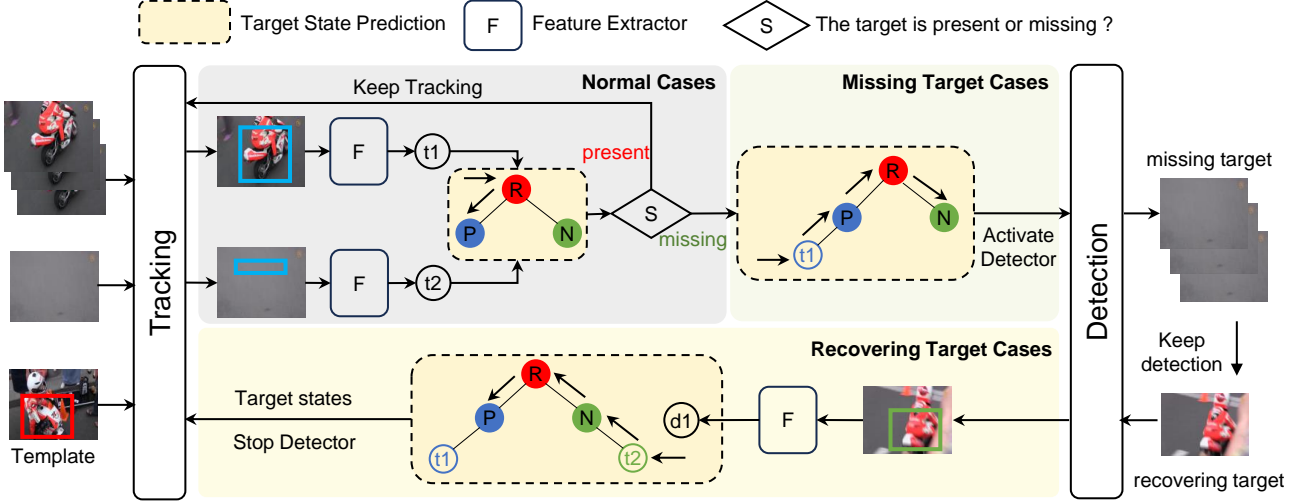
Figure 3. **Overall flow of the proposed algorithm.** Our proposed tracking method dynamically associates the tracker and detector based on the target state. It contains three associating processes: 1) normal case flow, which validates the target state normal, utilizing only the tracker for tracking; 2) target missing flow, which confirms the target as lost, activating the detector for global searching; 3) target recovering flow, which detecting until the lost target recovering, the detector stopped and reactivating the tracker.

Algorithm 1 Recovering Tracking Algorithm

Require: Video $V = (I_0, I_1, \ldots, I_t)$, Tracker T, Detector D, Reference R
1: $M \leftarrow \text{INITMEM}(R)$ ▷ Init PN tree M with R
2: $S \leftarrow 1$ % Init target State S as present
3: Initialize T and D with R
4: **while** frame I_i in V **do**
5:    $Results \leftarrow T(R, I_i)$
6:    $S \leftarrow \text{WALKING}(M, Results)$
7:    **if** $S = 1$ **then**
8:        $M \leftarrow \text{UPDATE}(M, Results)$
9:        **continue**
10:   **else**
11:       $M \leftarrow \text{UPDATE}(M, Results)$
12:       **while** $S = 0$ **do** % Target is missing
13:           $Results \leftarrow D(I_i)$
14:           $S \leftarrow \text{WALKING}(M, Results)$
15:           **if** $S = 0$ **then**
16:               $i \leftarrow i + 1$
17:           **else**
18:               $M \leftarrow \text{UPDATE}(M, Results)$
19:               $T \leftarrow \text{REINIT}(Results)$ ▷ Reinit tracker
20:           **end if**
21:       **end while**
22:   **end if**
23:   $i \leftarrow i + 1$
24: **end while**

4.1. Implementation Details

Our experiments are conducted using 4 NVIDIA Tesla V100 GPUs. We employ the MixViT-L (ConvMAE) [8] as the base tracker in conjunction with the MITS [36] model as the based detector. For the tracking model, we crop the search image that is 4.5 times the area of the target box from the test frame and resize it to a resolution of 384×384 pixels. The template is cropped as twice that of the target box

and has a resolution of 192×192. We train a similarity perception model following the setting of dreamsim [12] on the LaSOT and NIGHTS [12] datasets as the feature extractor in the PN tree.

4.2. State-of-the-Art Comparison

We compare our tracker with the state-of-the-art tracking algorithms on five challenging tracking benchmarks, including VideoCube, LaSOT, LaSOT$_{ext}$, TNL2K, and GOT-10k. Table 1 presents all the tracking results.

VideoCube [15]. VideoCube is a comprehensive and challenging long-term visual tracking benchmark designed to reflect the complexities of the real world, such as object occlusion and disappearances. It comprises 500 video segments, each containing at least 4,008 frames, averaging about 14,920. This benchmark introduces a novel global instance tracking task, where the tracker should locate a specified instance in a video without assuming consistent camera or motion patterns. Our tracker is evaluated solely on its test set, comprising 100 sequences. As shown in Table 1, our approach achieves the best performance compared to the state-of-the-art methods. Compared to the second-best tracker MixViT [8], our method achieves a 2.8% increase in normalized precision (NP) and a 2.4% gain in success rate (SUC). The positive performance highlights the promising potential of our tracker to effectively address challenges related to camera switching and occlusion, which benefits from the association between the tracker and detector through accurate target state prediction.

LaSOT [10]. LaSOT is a high-quality, large-scale benchmark for long-term single-object tracking, featuring an av-

Table 1. **State-of-the-art comparisons on the datasets of VideoCube, TNL2K, LaSOT, LaSOT$_{ext}$, and GOT-10k.** The best two results are shown in red and blue color. Our approach performs favorably against the state-of-the-art methods on all datasets.

Method	VideoCube [15]			LaSOT [10]		LaSOT$_{ext}$ [11]			TNL2K [32]		GOT-10k [16]		
	P	NP	SUC	AUC	NP	AUC	NP	P	P	SUC	AO	SR$_{0.75}$	SR$_{0.5}$
SiamFC [1]	3.1	12.9	6.1	33.6	42.0	23.0	31.1	26.9	28.6	29.5	34.8	39.8	35.3
RPN++ [19]	-	-	-	49.6	56.9	34.0	41.6	39.6	41.2	41.3	51.7	32.5	61.6
Ocean [42]	18.3	51.9	32.5	56.0	65.1	-	-	-	37.7	38.4	61.1	47.3	72.1
TransT [6]	-	-	-	64.9	73.8	-	-	-	51.7	50.7	67.1	60.9	76.8
KeepTrack [26]	35.9	68.7	50.6	67.1	77.2	48.2	-	-	-	-	-	-	-
GlobalTrack [26]	29.3	63.1	44.8	51.7	59.7	35.6	43.6	41.1	38.6	40.5	-	-	-
Stark [38]	-	-	-	67.1	77.0	-	-	-	-	-	68.8	64.1	78.1
OSTrack [41]	-	-	-	71.1	81.1	50.5	61.3	57.6	-	55.9	73.7	70.8	83.2
CiteTracker [22]	-	-	-	69.7	78.6	-	-	-	59.6	57.7	74.7	73.0	84.3
DropTrack [34]	-	-	-	71.8	81.8	52.7	63.9	60.2	57.9	56.9	75.9	72.0	86.8
MITS [36]	36.9	66.7	46.6	72.1	80.1	50.3	60.1	58.6	58.5	55.5	80.4	75.9	89.7
SwinTrack [23]	-	-	-	71.3	-	49.1	-	55.6	55.7	55.6	72.4	67.8	80.5
SeqTrack-L [7]	60.9	77.4	66.1	72.5	81.5	50.7	61.6	57.5	-	57.8	74.8	72.2	81.9
MixViT-L [8]	59.9	78.7	67.2	73.3	82.8	50.9	61.0	57.9	61.7	59.0	75.7	75.1	85.3
ARTrack-L [33]	31.3	53.2	39.5	73.1	82.2	52.8	62.9	59.7	-	60.3	78.5	77.8	87.4
UNINEXT-H [39]	-	-	-	72.2	80.8	56.2	63.8	63.8	62.8	59.3	-	-	-
RTracker-L	63.2	81.5	69.6	74.7	84.5	54.9	65.5	62.7	63.7	60.6	77.9	76.9	87.0

erage video length exceeding 2,500 frames. It offers diverse real-world challenges, including scenarios where target objects can intermittently disappear and reappear in view. As shown in Table 1, our tracker improves all metrics, *e.g.* 1.4% in Area Under the Curve (AUC) compared with MixViT and ARTrack. The encouraging performance demonstrates that our tracker can tackle the object disappearing and reappearing situation, which shows the self-recovery ability.

LaSOT$_{ext}$ [11]. LaSOT$_{ext}$ extends LaSOT with 150 additional videos, introducing the tracking challenges due to similar distractors. UNINEXT employs a more robust backbone, ViT-Huge, and diverse training datasets for feature extraction, achieving a promising 56.2% AUC score. In comparison, our approach only uses the ViT-Large as the backbone to save computation costs while delivering a comparable performance with a 54.9% AUC score. This benefits from the PN tree memory, adeptly storing target-relevant and irrelevant samples like similar distractors in the background, to reliably ascertain the target states for associating the tracker and detector.

TNL2K [32]. TNL2K is a challenging tracking benchmark that employs template patches and language descriptions to locate target objects in video sequences, facilitating the connection between local and global searches. However, we only use the bounding box for evaluation. Compared to UNINEXT with a more robust backbone, our approach still demonstrates improvements with a 0.9% increase in precision and a 1.3% increase in SUC, attributed to the RTracking framework that facilitates the connection between local

Table 2. **Ablation study of the proposed algorithm on the VideoCube, LaSOT$_{ext}$, and TNL2K datasets.** The best results are marked in **bold**.

Method	VideoCube		LaSOT_ext		TNL2K	
	AUC	NP	AUC	NP	AUC	NP
Base T	67.2	78.7	50.7	61.6	59.0	75.5
Base D	46.6	66.7	50.3	60.1	55.5	69.7
Fixed THR	60.9	72.1	50.7	59.9	56.5	71.5
W/O WR	62.6	73.6	49.0	58.0	57.4	73.3
RTracker	**69.6**	**81.5**	**54.9**	**65.5**	**60.6**	**76.9**

and global search.

GOT-10k [16]. GOT-10k is a large-scale tracking benchmark covering most categories for over 560 real-world moving objects. The ground truths for the test set are withheld, and we assess our approach using the evaluation platform provided by the authors. We follow the one-shot rule with zero overlapping in object classes between the training and test sets and use the tracker and the detector trained only on the GOT-10k training set. Our RTracker also achieves competitive results, comparable to the most recent trackers MITS [36] and ARTrack [33]. The good performance shows that our tracker has a good generalization ability to the tracking scenarios involving class-agnostic targets.

4.3. Ablation Study

To evaluate the effect of each individual component of our tracker, we carry out ablation studies on five different variants of the RTracker:

RTracker, our intact model dynamically associates a
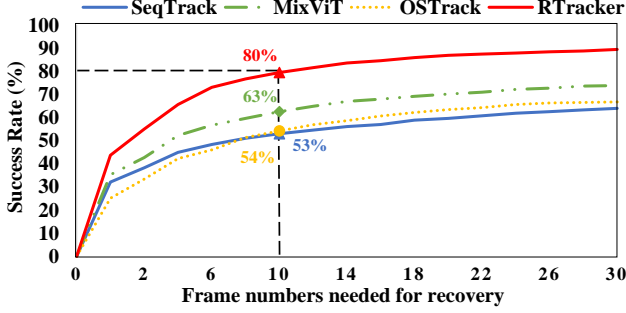
Figure 4. **Evaluation of the recovery ability on LaSOT.** The success rate is the percentage at which a tracker successfully recovers disappeared targets within specific frame numbers.

tracker and a detector that offers global search capabilities to achieve recoverable tracking based on the changing target states predicted by relative measurements utilizing the PN tree memory.

Base Tracker (T), which employs only the tracker in RTracker to track targets. In this work, we use the MixViT-L [8] as our base tracker.

Base Detector (D), which employs only the detector in RTracker to track targets. In this work, we use the MITS [36] as our base detector.

Fixed Threshold (THR), which predicts the current state of the target by comparing the tracking confidence score with a pre-set fixed threshold in place of the state prediction by the PN tree.

W/O Walking Rules (WR), which determines the current state of the target based on the relative distances to temporally adjacent positive and negative samples instead of the state prediction through the walking rules.

Table 2 presents the experimental results of these variants on the VideoCube [15], LaSOT$_{ext}$ [11], and TNL2K [32] datasets.

Effect of associating tracking and detection. By comparing our RTrack with the base tracker, it is clear that the proposed tracking algorithm improves tracking performance by 2.8%, 3.9%, and 1.4% in terms of NP on VideoCube, LaSOT$_{ext}$ and TNL2K, respectively. While compared with the base detector, our tracker achieves performance gain of 23%, 4.6%, and 5.1% in terms of AUC on VideoCube, LaSOT$_{ext}$ and TNL2K, respectively. The gap performance between the base tracker, the base detector, and our proposed method demonstrates the effectiveness of our approach in associating the tracker and detector.

Effect of the PN tree. With the relative measurement between the positive and negative features stored in the PN tree, RTrack achieves performance gains of 8.7%, 4.2%, and 4.1% in AUC on VideoCube, LaSOT$_{ext}$, and TNL2K, while 9.4%, 5.6% and 5.4% in NP, respectively. The en-

hancements observed validate the benefits of relative measurements. Unlike a fixed threshold for ascertaining the target state, relative measurements provide favorable adaptability to target changes during tracking, enhancing the robustness of target state predictions.

Effect of using the walking rules. Without the walking rules, RTracker decrease by 7%, 5.9% and 3.2% in precision on VideoCube, LaSOT$_{ext}$ and TNL2K. The results confirm the effectiveness of our method in employing walking rules to continually update memory, capture appearance changes, and accurately determine target states.

4.4. Recovery Ability Evaluation

We assess the recovery ability of the proposed method on the LaSOT dataset by measuring the number of frames required to relocate a lost target. A recovery is deemed successful if the overlap between the predicted bounding box and its ground truth exceeds 0.5. As illustrated in Figure 4, RTracker can recover more lost targets within the same time as other trackers. Notably, RTracker achieves a high success rate, successfully recovering targets in 80% of the test cases where they were lost. In comparison, MixViT has a 63% success rate in recovering targets, while SeqTrack and OSTrack exhibit average performance, managing successful recovery in about half of the target loss cases. The comparison shows that RTracker, combining tracking and detection with PN tree memory, has enhanced target recovery ability in challenging situations and is more effective than other methods reliant solely on tracking.

4.5. Qualitative Study

To demonstrate the self-recovery capability of our proposed method, we visualize the tracking results of several challenging sequences with MixViT, SeqTrack, and OSTrack. In the $CartoonSlamDUNK$ sequence, RTracker can immediately relocate to the position of the target after it gets lost at the 104th frame and recovers at the 106th frame, whereas other trackers can only relocate the target after the 130th frame. In the $CrashCar$ series, our tracker accurately tracks the target despite shifts in the tracking viewpoint, whereas other trackers may not. The visualized results demonstrate that RTracker can recover from the missing or out-of-view targets due to the association of the tracker and detector via the target states. Moreover, our RTracker can accurately locate the target even if the target undergoes drastic appearance changes or is blurred. In the $paddle$ sequence, the color of the target changes from red to black by the 29th frame. Our tracker still tracks the target correctly, whereas other trackers might locate the area more similar to the first frame. Despite the fast motion and full occlusion in the $jianzi$ sequence, our tracking method still performs well.

CartoonSlamDUNK_video_05_done Out of View, Viewpoint Change, Full Occlusion

CrashCar_video_24-Done Out of View, Viewpoint Change, Drastic Appearance Change

paddle-3 Out of View, Full Occlusion, Fast Motion, Motion Blur
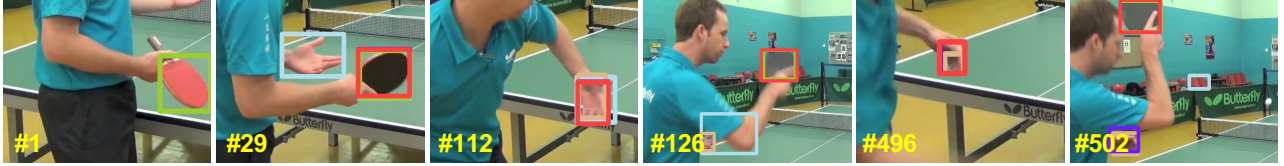
jianzi-9 Full Occlusion, Fast Motion, Out of View, Motion Blur

——RTracker ——MixViT ——SeqTrack ——OSTrack ——Ground-truth

Figure 5. **Visualized results of the proposed algorithm, MixViT, OSTrack, and SeqTrack method on four challenging sequences with drastic changes.** This indicates that our RTracker performs well with the support of detection through PN tree memory, whereas the other method that relies only on the tracker faces difficulties with these sequences.
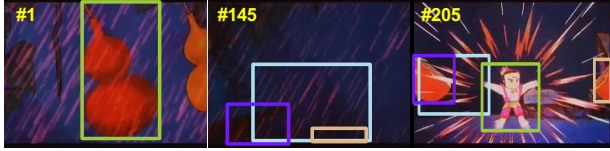
Figure 6. **Failure case of the proposed method.** The target disappears at frame 145 and recovers in a completely different appearance at frame 205, with similar distractors in the background.

4.6. Limitations

Due to the additional computational loads for evaluating the present state of the target and facilitating integration with the detector, RTracker runs at a speed of 0.75 times that of the baseline method. We intend to reduce the effects of this extra cost by using lightweight techniques. In addition, RTracker may not be able to recover tracking correctly and timely in a few extreme scenarios where the target reappears with a totally different appearance, and there are also similar background distractors simultaneously. Figure 7 shows an example where the target disappears at frame 145 and reappears at frame 205, where all trackers fail to track accurately. To address this issue, our future work will incorporate semantic descriptors to guide the tracker in effectively adapting to targets with entirely different appearances.

5. Conclusion

In this work, we present a recoverable tracking framework, RTracker, that dynamically associates a tracker and a detector for self-recovery. Specifically, we construct a Positive-Negative Tree Structured memory, which maintains the samples relevant and irrelevant to the tracking target chronologically. In addition, we formulate a set of walking rules for the PN tree memory, enabling the reliable determination of the target state by assessing the relative distances between positive and negative samples. Upon this target state, we define three control flows to associate the tracker and the detector adaptively for robust tracking. Both qualitative and quantitative assessments demonstrate that our approach performs favorably against state-of-the-art methods, highlighting the effectiveness of dynamically integrating trackers and detectors with PN tree memory for improving tracking performance.

6. Acknowledgments

References

[1] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *ECCVW*, 2016. 2, 6

[2] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *ICCV*, 2019. 1, 2

[3] Philippe Blatter, Menelaos Kanakis, Martin Danelljan, and Luc Van Gool. Efficient visual tracking with exemplar transformers. In *WACV*, pages 1571–1581, 2023. 2

[4] Boyu Chen, Peixia Li, Lei Bai, Lei Qiao, Qiuhong Shen, Bo Li, Weihao Gan, Wei Wu, and Wanli Ouyang. Backbone is all your need: a simplified architecture for visual object tracking. In *ECCV*. Springer, 2022. 1

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020. 2

[6] Xin Chen, Bin Yan, Jiawen Zhu, Dong Wang, Xiaoyun Yang, and Huchuan Lu. Transformer tracking. In *CVPR*, 2021. 1, 2, 6

[7] Xin Chen, Houwen Peng, Dong Wang, Huchuan Lu, and Han Hu. Seqtrack: Sequence to sequence learning for visual object tracking. In *CVPR*, 2023. 6, 1

[8] Yutao Cui, Cheng Jiang, Gangshan Wu, and Limin Wang. Mixformer: End-to-end tracking with iterative mixed attention, 2023. 5, 6, 7, 1

[9] Martin Danelljan, Luc Van Gool, and Radu Timofte. Probabilistic regression for visual tracking. In *CVPR*, 2020. 2

[10] Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling. Lasot: A high-quality benchmark for large-scale single object tracking. In *CVPR*, 2019. 2, 5, 6, 1

[11] Heng Fan, Hexin Bai, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Harshit, Mingzhen Huang, Juehuan Liu, Yong Xu, Chunyuan Liao, and Haibin Ling. Lasot: A high-quality large-scale single object tracking benchmark. *IJCV*, 2021. 2, 6, 7, 1

[12] Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data, 2023. 3, 5, 1

[13] Zhihong Fu, Qingjie Liu, Zehua Fu, and Yunhong Wang. Stmtrack: Template-free visual tracking with space-time memory networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13774–13783, 2021. 2

[14] Kaijie He, Canlong Zhang, Sheng Xie, Zhixin Li, and Zhiwen Wang. Target-aware tracking with long-term context attention. *arXiv preprint arXiv:2302.13840*, 2023. 2

[15] Shiyu Hu, Xin Zhao, Lianghua Huang, and Kaiqi Huang. Global instance tracking: Locating target more like humans. *IEEE TPAMI*, 2022. 2, 5, 6, 7

[16] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE TPAMI*, 2019. 2, 6

[17] Zdenek Kalal, Krystian Mikolajczyk, and Matas Jiri. Tracking-learning-detection. *IEEE TPAMI*, 2012. 2, 3

[18] Junseok Kwon and Kyoung Mu Lee. Tracking by sampling trackers. In *ICCV*, 2011. 2

[19] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. In *CVPR*, 2018. 6

[20] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with siamese region proposal network. In *CVPR*, 2018. 1, 2

[21] Xin Li, Chao Ma, Baoyuan Wu, Zhenyu He, and Ming-Hsuan Yang. Target-aware deep tracking. In *CVPR*, 2019. 1

[22] Xin Li, Yuqing Huang, Zhenyu He, Yaowei Wang, Huchuan Lu, and Ming-Hsuan Yang. Citetracker: Correlating image and text for visual tracking. In *ICCV*, 2023. 2, 6

[23] Liting Lin, Heng Fan, Zhipeng Zhang, Yong Xu, and Haibin Ling. Swintrack: A simple and strong baseline for transformer tracking. In *NeurIPS*, 2022. 1, 2, 6

[24] Baiyang Liu, Junzhou Huang, Casimir Kulikowski, and Lin Yang. Robust visual tracking using local sparse appearance model and k-selection. In *PAMI*, 2012. 2

[25] Chao Ma, Xiaokang Yang, Chongyang Zhang, and Ming-Hsuan Yang. Long-term correlation tracking. In *ICCV*, 2015. 2, 3

[26] Christoph Mayer, Martin Danelljan, Danda Pani Paudel, and Luc Van Gool. Learning target candidate association to keep track of what not to track. In *ICCV*, 2021. 6

[27] Georg Nebehay and Roman Pflugfelder. Clustering of static-adaptive correspondences for deformable object tracking. In *CVPR*, 2015. 3

[28] Yibing Song, Chao Ma, Lijun Gong, Jiawei Zhang, Rynson WH Lau, and Ming-Hsuan Yang. Crest: Convolutional residual learning for visual tracking. In *ICCV*, pages 2574–2583, 2017. 2

[29] Mingjie Sun, Jimin Xiao, Eng Gee Lim, Bingfeng Zhang, and Yao Zhao. Fast template matching and update for video object tracking and segmentation. In *CVPR*, 2020. 2

[30] Paul Voigtlaender, Jonathon Luiten, Philip H.S. Torr, and Bastian Leibe. Siam r-cnn: Visual tracking by re-detection. In *CVPR*, 2020. 1, 2

[31] Ning Wang, Wengang Zhou, Jie Wang, and Houqiang Li. Transformer meets tracker: Exploiting temporal context for robust visual tracking. In *CVPR*, 2021. 1

[32] Xiao Wang, Xiujun Shu, Zhipeng Zhang, Bo Jiang, Yaowei Wang, Yonghong Tian, and Feng Wu. Towards more flexible and accurate object tracking with natural language: Algorithms and benchmark. In *CVPR*, 2021. 2, 6, 7

[33] Xing Wei, Yifan Bai, Yongchao Zheng, Dahu Shi, and Yihong Gong. Autoregressive visual tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9697–9706, 2023. 6, 1

[34] Qiangqiang Wu, Tianyu Yang, Ziquan Liu, Baoyuan Wu, Ying Shan, and Antoni B. Chan. Dropmae: Masked autoencoders with spatial-attention dropout for tracking tasks. In *CVPR*, 2023. 6

[35] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *CVPR*, 2013. 1

[36] Yuanyou Xu, Zongxin Yang, and Yi Yang. Integrating boxes and masks: A multi-object framework for unified visual tracking and segmentation. *arXiv preprint arXiv:2308.13266*, 2023. 5, 6, 7, 1

[37] Bin Yan, Haojie Zhao, Dong Wang, Huchuan Lu, and Xiaoyun Yang. 'skimming-perusal' tracking: A framework for real-time and robust long-term tracking. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 3

[38] Bin Yan, Houwen Peng, Jianlong Fu, Dong Wang, and Huchuan Lu. Learning spatio-temporal transformer for visual tracking. In *ICCV*, 2021. 6

[39] Bin Yan, Yi Jiang, Jiannan Wu, Dong Wang, Zehuan Yuan, Ping Luo, and Huchuan Lu. Universal instance perception as object discovery and retrieval. In *CVPR*, 2023. 6

[40] Tianyu Yang and Antoni B. Chan. Learning dynamic memory networks for object tracking. In *ECCV*, 2018. 2

[41] Botao Ye, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. Joint feature learning and relation modeling for tracking: A one-stream framework. In *ECCV*. Springer, 2022. 1, 2, 6

[42] Zhipeng Zhang, Houwen Peng, Jianlong Fu, Bing Li, and Weiming Hu. Ocean: Object-aware anchor-free tracking. In *ECCV*, 2020. 6

RTracker: Recoverable Tracking via PN Tree Structured Memory

Supplementary Material

This document provides additional information on the experimental implementation and results.

A. Implementation Details

A.1. Feature Extractor in the PN tree

We use the features extracted by a similarity perception model [12] to describe the state of the tracking targets as they are more robust to changes in the target.

Datasets. To adapt the similarity perception model for tracking tasks, we train it on two datasets: Novel Image Generations with Human-Tested Similarity (NIGHTS) [12] dataset and LaSOT [10] training set. NIGHTS comprises human similarity judgments for image triplets, each consisting of a reference image and two altered versions, along with human assessments about which version is most similar to the reference. For LaSOT, we employ the target found in the initial frame of each sequence as the template. Subsequently, we crop the ground truth area of the tracking target from the following frames as A, ensuring that it pertains to the same object as the template. We also extract targets from other sequences of the same category as B, which is similar to the template. In total, we exploit 20,000 triplets from NIGHTS and generate an additional 4,000 triplets from LaSOT as our training datasets.

Training Settings. We denote the distance between two samples as D computed as follows:

$$D_1 = 1 - cos(f_\theta(Template), f_\theta(A)), \quad (3)$$
$$D_2 = 1 - cos(f_\theta(Template), f_\theta(B)), \quad (4)$$

where f_θ represents the feature extractor, D_1 is the distance between the template and A samples while D_2 is the distance between the template and B samples. To improve target state assessment, especially in scenarios with similar noise to tracking targets, we aim to maximize the difference between the distances, D_1 and D_2, while simultaneously minimizing D_1 for the tracking target from the same sequence within the triplet $(Template, A, B)$. Therefore, we follow the training settings of dreamsim [12] and use a hinge training loss can be formulated as:

$$\mathcal{L} = max(0, m - \Delta D \cdot \bar{y}), \Delta D = D_0 - D_1,$$
$$\bar{y} = \begin{cases} 1 = D_1 < D_0 \\ 0 = D_0 < D_1, \end{cases}$$

where $\bar{y}$ is the relative distance judgment between the template and A/B and m equals to 0.05.

___
* corresponding author
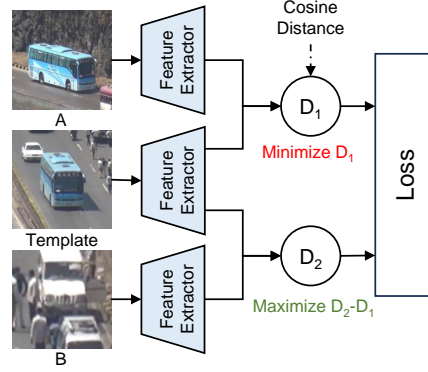
Figure 7. **Training method.** Given a triplet, we have a template, A and B samples. we compute the cosine distances, D_1 and D_2, between the features of the Template and A/B. Our training objective is to minimize D_1 and maximize the gap between D_1 and D_2.

B. More Detailed Experimental Results

B.1. Detailed results on VideoCube

Table 1 shows the evaluation results on the VideoCube benchmark. We conduct tests on the VideoCube dataset using the provided checkpoints from SeqTrack-Large [7], MixViT-Large [8], and MITS [36]. However, for AR-Track [33], since the authors did not release the checkpoint of ARTrack-Large, we independently train ARTrack-Large on our own machine to conduct the testing.

B.2. Detailed results on LaSOT

The AUC score under different attributes of LaSOT [10] test set is shown in Figure 8 and Figure 9. Compared to the second-best tracker MixViT, our proposed method achieves an improvement of 3.2% in out-of-view cases and 2.6% AUC scores in full-occlusion cases on the LaSOT test set. This demonstrates that our tracker possesses a robust self-recovery capability, attributed to our approach of combining detection with tracking based on the target states, enabling the tracker to handle situations where the target is missing and swiftly relocate the target.

B.3. Detailed results on LaSOT$_{ext}$

The AUC score under different attributes of LaSOT$_{ext}$ [11] test set is shown in Figure 10 and Figure 11. It is worth noting that compared to the second-best tracker, our proposed method achieves an improvement of 5.1% and 4.4% AUC scores in fast-motion and low-resolution cases on the LaSOT$_{ext}$ test set, which benefits from the effective aid of the detection.

1

B.4. Detailed results on TNL2K

The AUC score under different attributes of TNL2K [32] test set is shown in Figure 12 and Figure 13. TNL2K has introduced a novel challenge involving adversarial samples. Compared to other trackers, our proposed method shows a 1.3% increase in AUC scores, highlighting the effectiveness of our target state prediction in distinguishing the target from similar objects.

Success plots of OPE - Fast Motion (53)

- [0.649] Ours
- [0.624] MixViT
- [0.591] OSTrack
- [0.582] SeqTrack
- [0.456] DiMP
- [0.435] LTMU
- [0.414] ATOM
- [0.396] GlobalTrack
- [0.316] SiamRPN++
- [0.312] SPLT
- [0.290] C-RPN
- [0.233] ECO
- [0.195] SiamFC
- [0.163] MEEM
- [0.161] LCT
- [0.156] CFNet
- [0.146] HCFT
- [0.143] TLD
- [0.140] CSRDCF
- [0.125] SAMF
- [0.118] fDSST
- [0.106] SCT4
- [0.104] ASLA
- [0.089] KCF
- [0.086] CN
- [0.066] CSK
- [0.055] MIL

Success plots of OPE - Background Clutter (100)

- [0.684] Ours
- [0.665] MixViT
- [0.659] SeqTrack
- [0.627] OSTrack
- [0.497] DiMP
- [0.466] LTMU
- [0.449] SiamRPN++
- [0.438] ATOM
- [0.434] GlobalTrack
- [0.409] C-RPN
- [0.351] SPLT
- [0.319] ECO
- [0.308] SiamFC
- [0.271] CFNet
- [0.254] SAMF
- [0.246] HCFT
- [0.238] MEEM
- [0.233] CSRDCF
- [0.231] LCT
- [0.222] fDSST
- [0.213] TLD
- [0.212] SCT4
- [0.198] KCF
- [0.197] ASLA
- [0.185] CN
- [0.170] CSK
- [0.138] MIL

Success plots of OPE - Full Occlusion (118)

- [0.683] Ours
- [0.651] MixViT
- [0.630] SeqTrack
- [0.628] OSTrack
- [0.475] DiMP
- [0.461] LTMU
- [0.438] GlobalTrack
- [0.415] ATOM
- [0.366] SiamRPN++
- [0.349] SPLT
- [0.348] C-RPN
- [0.254] ECO
- [0.245] SiamFC
- [0.215] MEEM
- [0.199] CFNet
- [0.189] HCFT
- [0.188] SAMF
- [0.182] TLD
- [0.167] CSRDCF
- [0.166] LCT
- [0.158] SCT4
- [0.142] fDSST
- [0.140] KCF
- [0.138] ASLA
- [0.127] CN
- [0.105] CSK
- [0.094] MIL

Success plots of OPE - Low Resolution (141)

- [0.693] Ours
- [0.667] MixViT
- [0.652] SeqTrack
- [0.650] OSTrack
- [0.491] DiMP
- [0.483] LTMU
- [0.452] GlobalTrack
- [0.445] ATOM
- [0.385] SiamRPN++
- [0.358] SPLT
- [0.355] C-RPN
- [0.267] ECO
- [0.252] SiamFC
- [0.195] CFNet
- [0.194] MEEM
- [0.185] HCFT
- [0.180] TLD
- [0.177] CSRDCF
- [0.170] SAMF
- [0.170] LCT
- [0.147] fDSST
- [0.143] SCT4
- [0.137] ASLA
- [0.126] KCF
- [0.117] CN
- [0.095] CSK
- [0.084] MIL

Success plots of OPE - Aspect Ration Change (249)

- [0.734] Ours
- [0.716] MixViT
- [0.709] SeqTrack
- [0.696] OSTrack
- [0.545] DiMP
- [0.515] LTMU
- [0.507] GlobalTrack
- [0.479] ATOM
- [0.472] SiamRPN++
- [0.435] C-RPN
- [0.411] SPLT
- [0.308] SiamFC
- [0.288] ECO
- [0.243] MEEM
- [0.243] CFNet
- [0.234] HCFT
- [0.215] CSRDCF
- [0.211] SAMF
- [0.198] LCT
- [0.192] TLD
- [0.171] SCT4
- [0.168] fDSST
- [0.163] ASLA
- [0.158] KCF
- [0.149] CN
- [0.128] MIL
- [0.127] CSK

Success plots of OPE - Partial Occlusion (187)

- [0.725] Ours
- [0.704] MixViT
- [0.697] SeqTrack
- [0.685] OSTrack
- [0.521] DiMP
- [0.504] LTMU
- [0.491] GlobalTrack
- [0.466] SiamRPN++
- [0.459] ATOM
- [0.432] C-RPN
- [0.397] SPLT
- [0.306] SiamFC
- [0.290] ECO
- [0.246] CFNet
- [0.244] MEEM
- [0.233] HCFT
- [0.212] SAMF
- [0.211] CSRDCF
- [0.198] LCT
- [0.189] TLD
- [0.177] SCT4
- [0.175] fDSST
- [0.174] ASLA
- [0.161] KCF
- [0.149] CN
- [0.131] CSK
- [0.129] MIL

Figure 8. **Success plots of different attributes on LaSOT.**

3

Success plots of OPE - Deformation (142)

- [0.751] Ours
- [0.749] SeqTrack
- [0.743] MixViT
- [0.717] OSTrack
- [0.566] DiMP
- [0.531] GlobalTrack
- [0.529] LTMU
- [0.528] SiamRPN++
- [0.511] ATOM
- [0.479] C-RPN
- [0.447] SPLT
- [0.351] SiamFC
- [0.279] ECO
- [0.271] CFNet
- [0.257] HCFT
- [0.256] MEEM
- [0.235] CSRDCF
- [0.229] SAMF
- [0.209] LCT
- [0.180] ASLA
- [0.178] SCT4
- [0.174] KCF
- [0.173] TLD
- [0.163] CN
- [0.162] fDSST
- [0.149] MIL
- [0.148] CSK

Success plots of OPE - Motion Blur (89)

- [0.736] Ours
- [0.717] MixViT
- [0.705] SeqTrack
- [0.690] OSTrack
- [0.546] DiMP
- [0.529] LTMU
- [0.497] GlobalTrack
- [0.493] ATOM
- [0.442] SiamRPN++
- [0.413] C-RPN
- [0.413] SPLT
- [0.308] SiamFC
- [0.305] ECO
- [0.251] MEEM
- [0.238] CFNet
- [0.226] CSRDCF
- [0.222] HCFT
- [0.220] SAMF
- [0.217] LCT
- [0.202] TLD
- [0.193] fDSST
- [0.173] SCT4
- [0.161] KCF
- [0.150] CN
- [0.150] ASLA
- [0.128] CSK
- [0.114] MIL

Success plots of OPE - Out-of-View (104)

- [0.704] Ours
- [0.678] MixViT
- [0.672] SeqTrack
- [0.664] OSTrack
- [0.499] GlobalTrack
- [0.499] LTMU
- [0.495] DiMP
- [0.416] ATOM
- [0.416] SiamRPN++
- [0.393] SPLT
- [0.365] C-RPN
- [0.256] SiamFC
- [0.239] ECO
- [0.209] MEEM
- [0.189] HCFT
- [0.183] CFNet
- [0.175] CSRDCF
- [0.173] LCT
- [0.167] TLD
- [0.159] SAMF
- [0.139] SCT4
- [0.137] fDSST
- [0.130] KCF
- [0.115] ASLA
- [0.113] CN
- [0.105] MIL
- [0.092] CSK

Success plots of OPE - Rotation (175)

- [0.734] Ours
- [0.724] SeqTrack
- [0.717] MixViT
- [0.700] OSTrack
- [0.545] DiMP
- [0.513] LTMU
- [0.510] GlobalTrack
- [0.485] SiamRPN++
- [0.468] ATOM
- [0.438] C-RPN
- [0.424] SPLT
- [0.310] SiamFC
- [0.285] ECO
- [0.247] CFNet
- [0.245] MEEM
- [0.239] HCFT
- [0.221] CSRDCF
- [0.212] SAMF
- [0.196] LCT
- [0.189] TLD
- [0.174] ASLA
- [0.172] SCT4
- [0.172] fDSST
- [0.158] KCF
- [0.156] CN
- [0.135] CSK
- [0.131] MIL

Success plots of OPE - Scale Variation (273)

- [0.744] Ours
- [0.727] MixViT
- [0.724] SeqTrack
- [0.708] OSTrack
- [0.558] DiMP
- [0.535] LTMU
- [0.516] GlobalTrack
- [0.494] ATOM
- [0.494] SiamRPN++
- [0.452] C-RPN
- [0.428] SPLT
- [0.332] SiamFC
- [0.318] ECO
- [0.267] CFNet
- [0.250] MEEM
- [0.244] HCFT
- [0.239] CSRDCF
- [0.226] SAMF
- [0.215] LCT
- [0.204] TLD
- [0.195] fDSST
- [0.187] ASLA
- [0.184] SCT4
- [0.171] KCF
- [0.163] CN
- [0.141] CSK
- [0.136] MIL

Success plots of OPE - Camera Motion (86)

- [0.784] Ours
- [0.775] MixViT
- [0.746] SeqTrack
- [0.740] OSTrack
- [0.593] DiMP
- [0.581] LTMU
- [0.547] ATOM
- [0.543] GlobalTrack
- [0.513] SiamRPN++
- [0.483] SPLT
- [0.482] C-RPN
- [0.358] ECO
- [0.333] SiamFC
- [0.289] CFNet
- [0.266] MEEM
- [0.261] HCFT
- [0.251] CSRDCF
- [0.248] SAMF
- [0.234] LCT
- [0.223] TLD
- [0.206] SCT4
- [0.202] fDSST
- [0.180] KCF
- [0.170] ASLA
- [0.159] CN
- [0.140] CSK
- [0.139] MIL

Figure 9. **Success plots of different attributes on LaSOT.**

4

Figure 10. **Success plots of different attributes on LaSOT$_{ext}$.**

Success plots of OPE - Viewpoint Change (59)

[0.589] Ours	
[0.548] SeqTrack	
[0.537] MixViT	
[0.537] OSTrack	
[0.456] LTMU	
[0.424] DiMP	
[0.383] ATOM	
[0.382] GlobalTrack	
[0.342] SiamRPN++	
[0.282] SPLT	
[0.276] C-RPN	
[0.219] SiamFC	
[0.171] ECO	
[0.163] CFNet	
[0.155] MEEM	
[0.142] CSRDCF	
[0.137] HCFT	
[0.124] SAMF	
[0.105] TLD	
[0.094] LCT	
[0.089] fDSST	
[0.081] CN	
[0.080] SCT4	
[0.080] KCF	
[0.079] ASLA	
[0.073] CSK	
[0.063] MIL	

Success plots of OPE - Motion Blur (67)

[0.360] Ours	
[0.339] MixViT	
[0.333] OSTrack	
[0.327] SeqTrack	
[0.249] DiMP	
[0.242] LTMU	
[0.231] GlobalTrack	
[0.219] ATOM	
[0.196] SiamRPN++	
[0.146] C-RPN	
[0.132] SPLT	
[0.118] SiamFC	
[0.114] MEEM	
[0.096] ECO	
[0.086] HCFT	
[0.083] CFNet	
[0.074] TLD	
[0.071] LCT	
[0.069] SAMF	
[0.068] CSRDCF	
[0.059] SCT4	
[0.057] KCF	
[0.055] CN	
[0.053] fDSST	
[0.049] ASLA	
[0.048] CSK	
[0.041] MIL	

Success plots of OPE - Out-of-View (32)

[0.436] Ours	
[0.393] MixViT	
[0.379] OSTrack	
[0.347] SeqTrack	
[0.250] GlobalTrack	
[0.248] LTMU	
[0.219] DiMP	
[0.197] ATOM	
[0.165] SiamRPN++	
[0.133] SPLT	
[0.117] C-RPN	
[0.087] MEEM	
[0.083] SiamFC	
[0.074] ECO	
[0.065] CFNet	
[0.064] TLD	
[0.057] HCFT	
[0.050] SCT4	
[0.050] CSRDCF	
[0.049] LCT	
[0.048] KCF	
[0.045] SAMF	
[0.042] fDSST	
[0.037] CN	
[0.035] ASLA	
[0.029] CSK	
[0.025] MIL	

Success plots of OPE - Rotation (39)

[0.586] Ours	
[0.572] SeqTrack	
[0.551] MixViT	
[0.543] OSTrack	
[0.502] LTMU	
[0.437] GlobalTrack	
[0.426] DiMP	
[0.413] SiamRPN++	
[0.412] ATOM	
[0.320] C-RPN	
[0.285] SPLT	
[0.280] SiamFC	
[0.249] ECO	
[0.244] MEEM	
[0.228] CFNet	
[0.192] HCFT	
[0.175] CSRDCF	
[0.172] LCT	
[0.168] TLD	
[0.161] SAMF	
[0.147] ASLA	
[0.143] CN	
[0.135] fDSST	
[0.133] KCF	
[0.130] SCT4	
[0.129] CSK	
[0.108] MIL	

Success plots of OPE - Scale Variation (141)

[0.528] Ours	
[0.486] MixViT	
[0.484] SeqTrack	
[0.483] OSTrack	
[0.404] LTMU	
[0.376] DiMP	
[0.363] ATOM	
[0.342] GlobalTrack	
[0.323] SiamRPN++	
[0.255] C-RPN	
[0.254] SPLT	
[0.211] SiamFC	
[0.203] ECO	
[0.175] MEEM	
[0.169] CFNet	
[0.141] HCFT	
[0.138] CSRDCF	
[0.133] SAMF	
[0.122] LCT	
[0.121] TLD	
[0.113] fDSST	
[0.107] SCT4	
[0.105] KCF	
[0.104] ASLA	
[0.099] CN	
[0.088] CSK	
[0.077] MIL	

Success plots of OPE - Camera Motion (18)

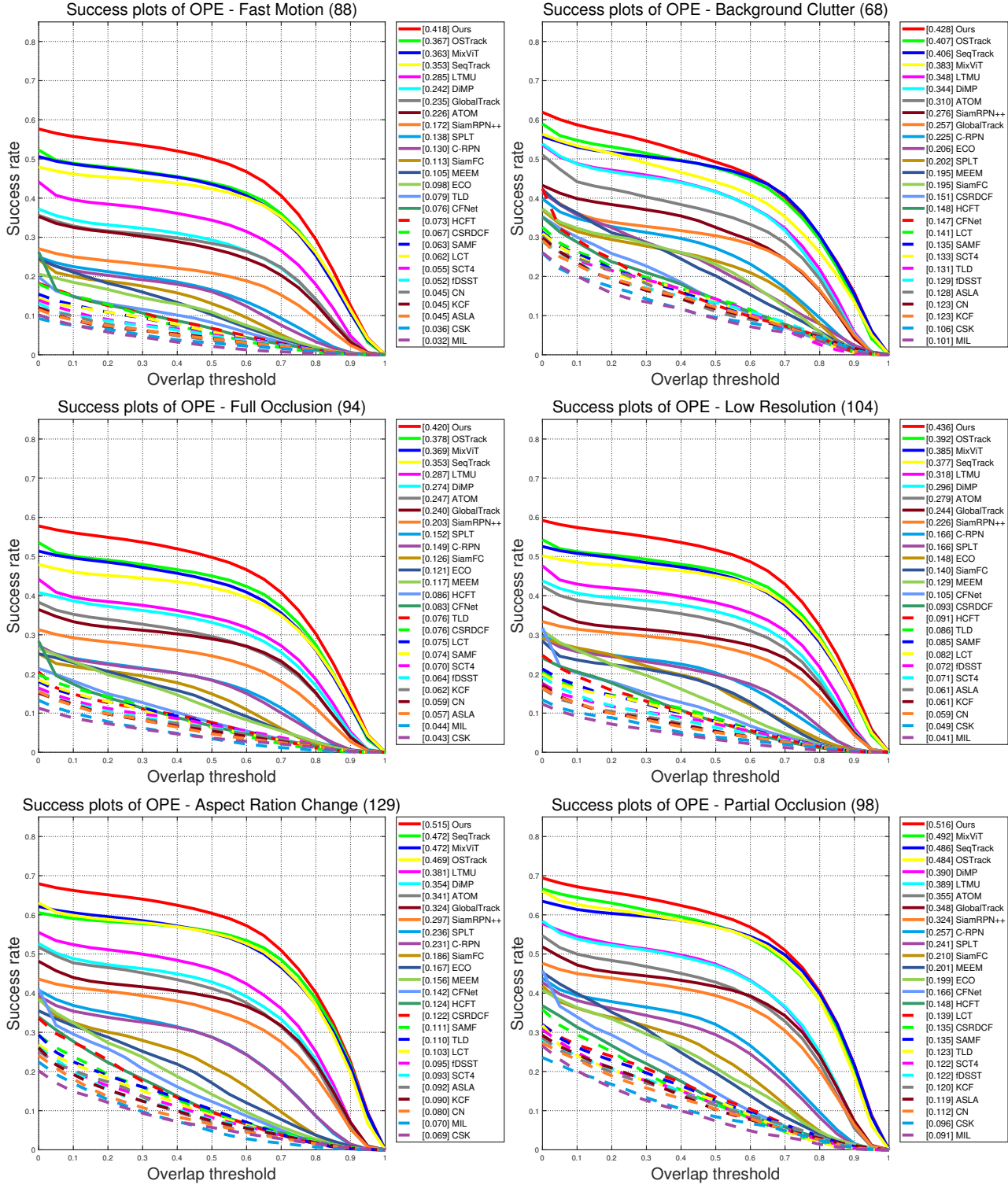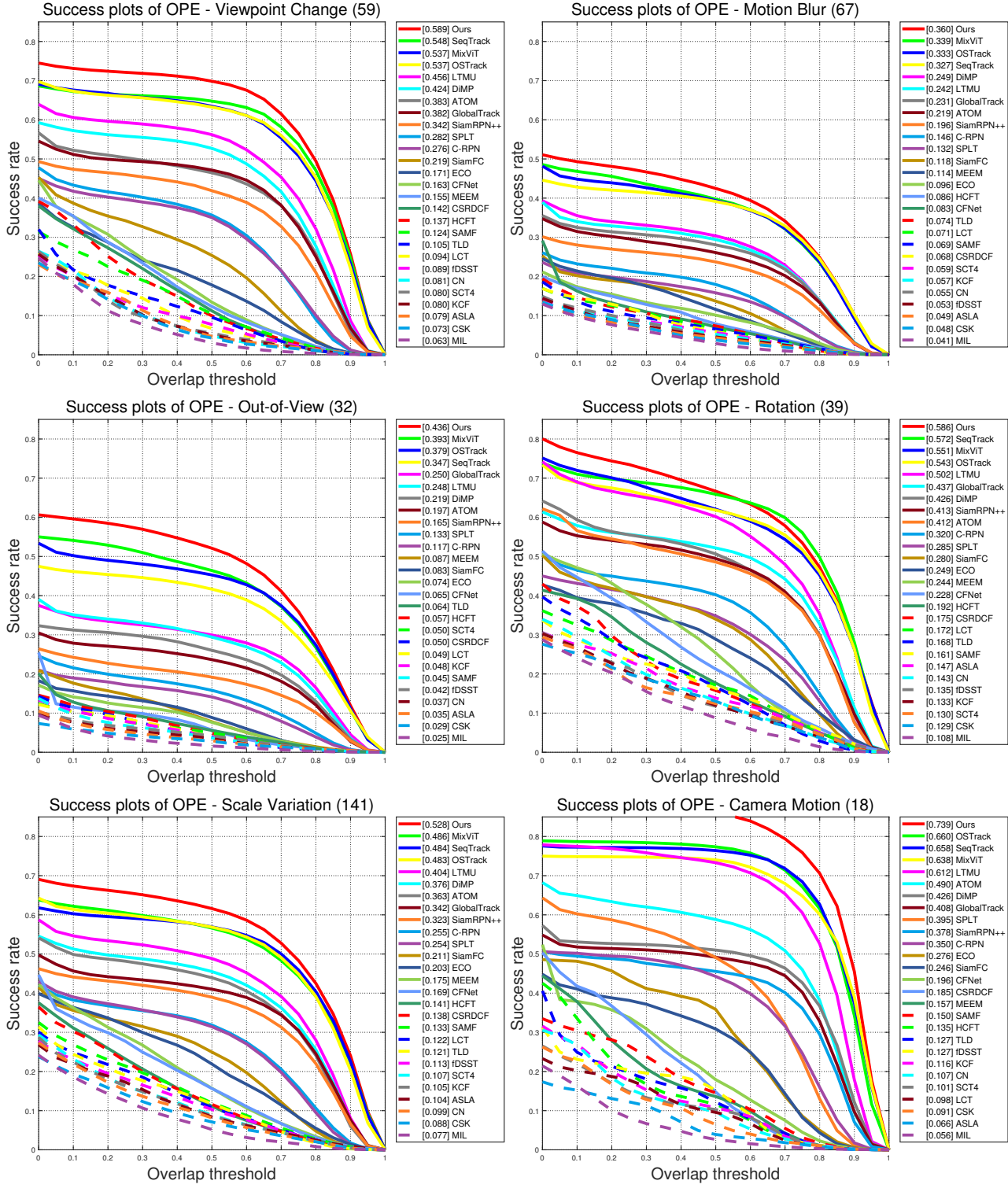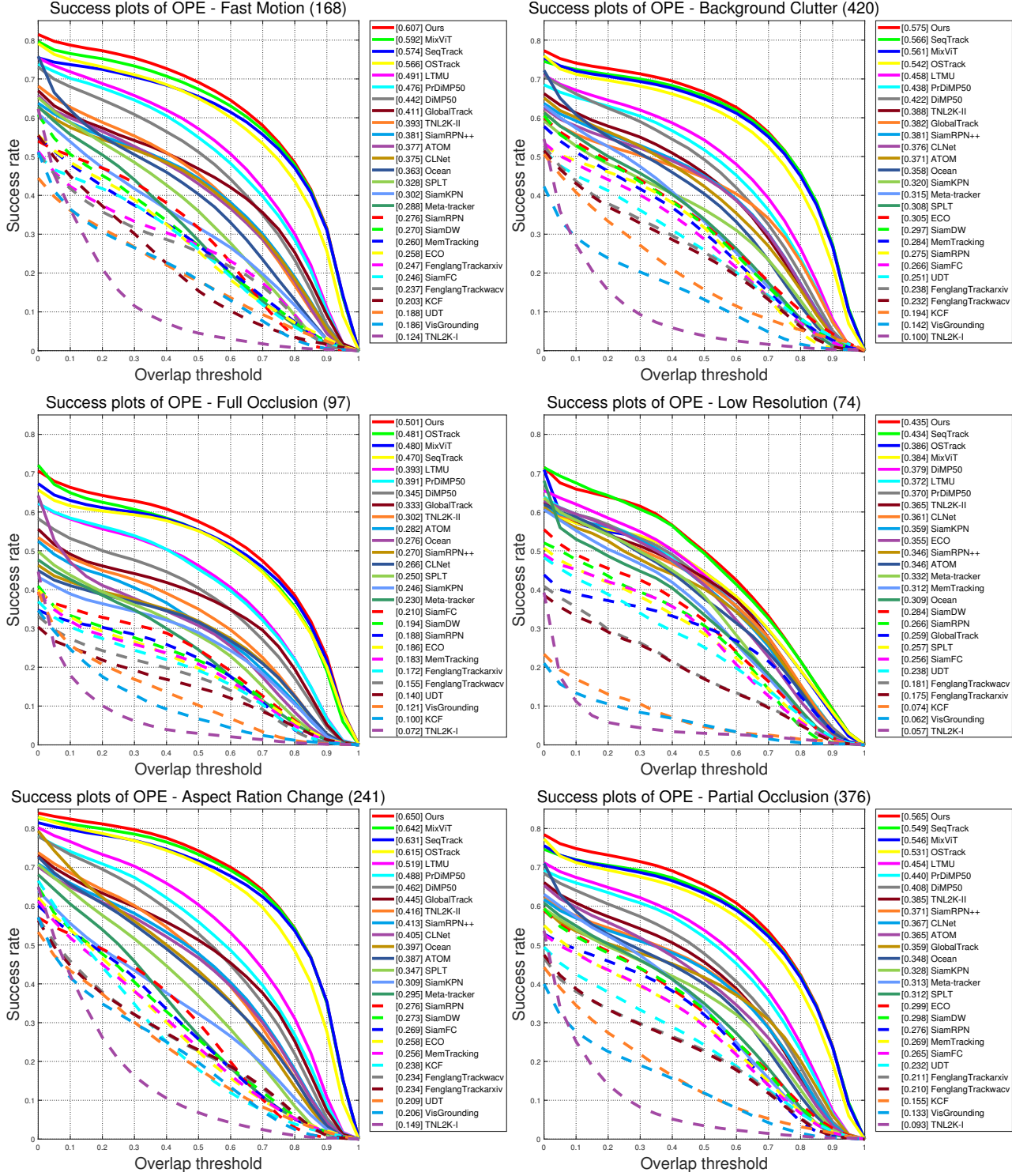|	
[0.739] Ours	
[0.660] OSTrack	
[0.658] SeqTrack	
[0.638] MixViT	
[0.612] LTMU	
[0.490] ATOM	
[0.426] DiMP	
[0.408] GlobalTrack	
[0.395] SPLT	
[0.378] SiamRPN++	
[0.350] C-RPN	
[0.276] ECO	
[0.246] SiamFC	
[0.196] CFNet	
[0.185] CSRDCF	
[0.157] MEEM	
[0.150] SAMF	
[0.135] HCFT	
[0.127] TLD	
[0.127] fDSST	
[0.116] KCF	
[0.107] CN	
[0.101] SCT4	
[0.098] LCT	
[0.091] CSK	
[0.066] ASLA	
[0.056] MIL	

Figure 11. **Success plots of different attributes on LaSOT$_{ext}$.**

Figure 12. **Success plots of different attributes on TNL2K.**

Success plots of OPE - Viewpoint Change (310)

Legend:
[0.580] Ours
[0.570] SeqTrack
[0.569] MixViT
[0.549] OSTrack
[0.468] LTMU
[0.457] PrDiMP50
[0.418] DiMP50
[0.388] GlobalTrack
[0.371] TNL2K-II
[0.367] ATOM
[0.365] SiamRPN++
[0.355] CLNet
[0.348] Ocean
[0.318] SPLT
[0.286] SiamKPN
[0.277] SiamDW
[0.270] Meta-tracker
[0.255] SiamRPN
[0.250] FenglangTrackarxiv
[0.242] FenglangTrackwacv
[0.240] MemTracking
[0.240] SiamFC
[0.239] ECO
[0.196] UDT
[0.189] KCF
[0.176] VisGrounding
[0.129] TNL2K-I

Success plots of OPE - Motion Blur (49)

Legend:
[0.590] Ours
[0.578] MixViT
[0.565] SeqTrack
[0.549] OSTrack
[0.508] LTMU
[0.478] PrDiMP50
[0.452] DiMP50
[0.420] GlobalTrack
[0.407] TNL2K-II
[0.405] SiamRPN++
[0.389] CLNet
[0.378] Ocean
[0.357] ATOM
[0.344] SPLT
[0.316] SiamKPN
[0.299] SiamRPN
[0.291] Meta-tracker
[0.282] MemTracking
[0.280] FenglangTrackarxiv
[0.279] SiamDW
[0.252] ECO
[0.250] FenglangTrackwacv
[0.249] SiamFC
[0.227] KCF
[0.207] UDT
[0.167] VisGrounding
[0.122] TNL2K-I

Success plots of OPE - Out-of-View (234)

Legend:
[0.473] Ours
[0.459] MixViT
[0.437] OSTrack
[0.433] SeqTrack
[0.365] LTMU
[0.350] PrDiMP50
[0.324] DiMP50
[0.316] GlobalTrack
[0.275] TNL2K-II
[0.259] Ocean
[0.256] SiamRPN++
[0.255] ATOM
[0.249] CLNet
[0.237] SPLT
[0.198] Meta-tracker
[0.196] SiamKPN
[0.189] SiamDW
[0.177] SiamFC
[0.171] ECO
[0.169] SiamRPN
[0.165] MemTracking
[0.146] FenglangTrackarxiv
[0.133] FenglangTrackwacv
[0.130] UDT
[0.126] KCF
[0.116] VisGrounding
[0.089] TNL2K-I

Success plots of OPE - Adversarial Samples (100)

Legend:
[0.677] Ours
[0.664] MixViT
[0.664] SeqTrack
[0.651] OSTrack
[0.556] LTMU
[0.545] PrDiMP50
[0.514] DiMP50
[0.488] SiamRPN++
[0.479] TNL2K-II
[0.476] ATOM
[0.472] CLNet
[0.448] GlobalTrack
[0.445] Ocean
[0.426] SiamKPN
[0.410] SPLT
[0.403] SiamDW
[0.403] Meta-tracker
[0.383] MemTracking
[0.378] ECO
[0.367] SiamRPN
[0.362] SiamFC
[0.348] FenglangTrackarxiv
[0.343] FenglangTrackwacv
[0.322] UDT
[0.255] KCF
[0.222] VisGrounding
[0.142] TNL2K-I

Success plots of OPE - Scale Variation (406)

Legend:
[0.582] Ours
[0.568] MixViT
[0.557] SeqTrack
[0.547] OSTrack
[0.469] LTMU
[0.449] PrDiMP50
[0.430] DiMP50
[0.383] TNL2K-II
[0.380] SiamRPN++
[0.379] GlobalTrack
[0.375] ATOM
[0.374] CLNet
[0.358] Ocean
[0.319] SPLT
[0.308] SiamKPN
[0.290] Meta-tracker
[0.288] SiamDW
[0.280] ECO
[0.271] SiamRPN
[0.267] MemTracking
[0.258] SiamFC
[0.236] FenglangTrackarxiv
[0.230] FenglangTrackwacv
[0.224] UDT
[0.166] KCF
[0.156] VisGrounding
[0.113] TNL2K-I

Success plots of OPE - Camera Motion (228)

Legend:
[0.609] Ours
[0.600] SeqTrack
[0.595] MixViT
[0.576] OSTrack
[0.485] LTMU
[0.484] PrDiMP50
[0.445] DiMP50
[0.414] TNL2K-II
[0.403] ATOM
[0.402] GlobalTrack
[0.402] CLNet
[0.399] SiamRPN++
[0.375] Ocean
[0.341] SPLT
[0.338] SiamKPN
[0.316] SiamDW
[0.315] ECO
[0.314] Meta-tracker
[0.297] MemTracking
[0.293] SiamRPN
[0.271] SiamFC
[0.239] UDT
[0.227] FenglangTrackarxiv
[0.223] FenglangTrackwacv
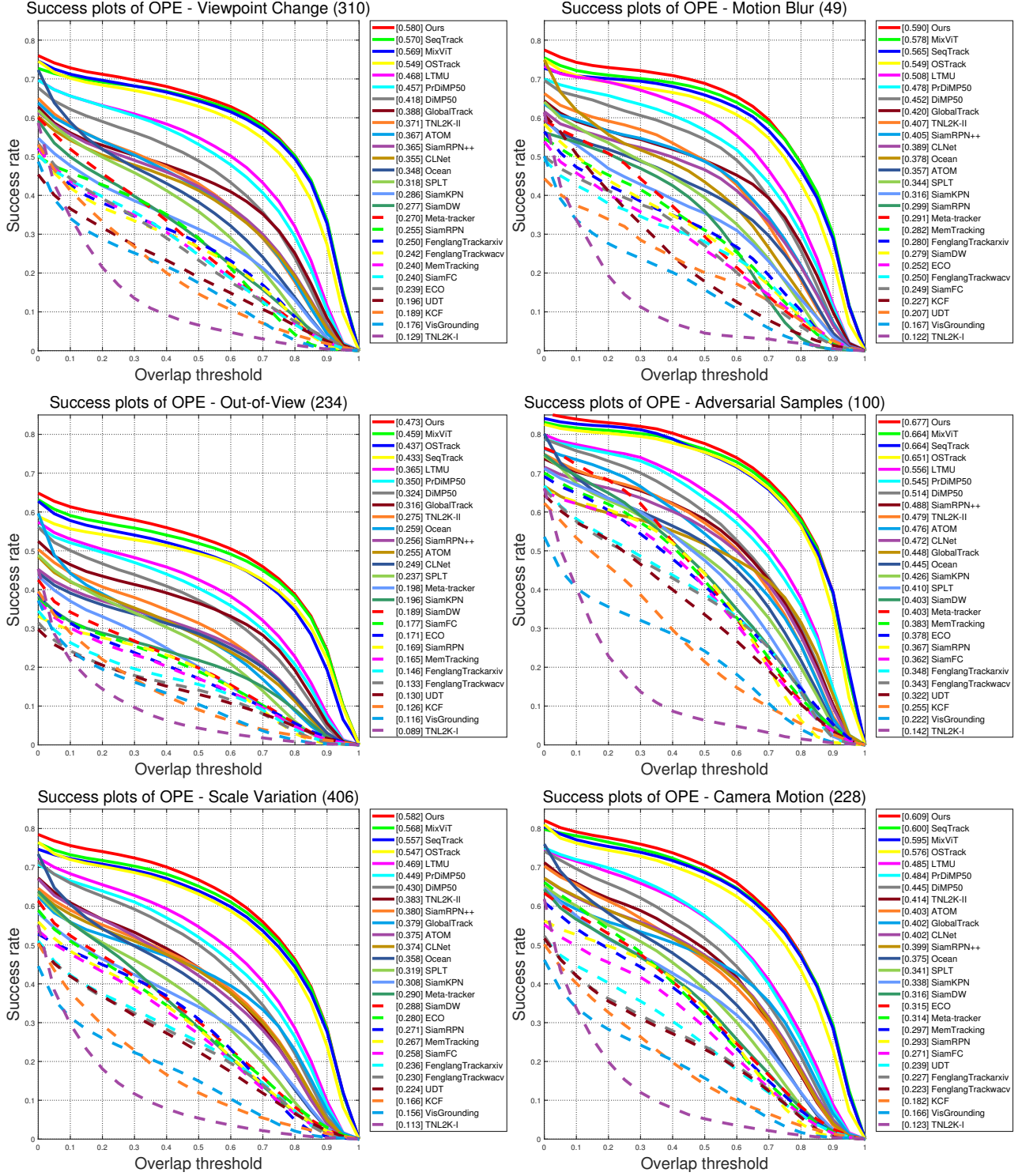[0.182] KCF
[0.166] VisGrounding
[0.123] TNL2K-I

Figure 13. **Success plots of different attributes on TNL2K.**