Siamese Adaptive Transformer Network for Real-Time Aerial Tracking

Daitao Xing\textsuperscript{1}, Athanasios Tsoukalas\textsuperscript{2}, Nikolaos Evangelou\textsuperscript{2}, Nikolaos Giakoumidis\textsuperscript{3} and Anthony Tzes\textsuperscript{2,4}

Abstract—Recent visual object trackers provide strong discriminability towards accurate tracking under challenging scenarios while neglecting the inference efficiency. Those methods handle all inputs with identical computation and fail to reduce intrinsic computational redundancy, which constrains their deployment on Unmanned Aerial Vehicles (UAVs). In this work, we propose a dynamic tracker which selectively activates the individual model components and allocates computation resources on demand during the inference, which allows deep network inference on onboard-CPU at real-time speed. The tracking pipeline is divided into several stages, where each stage consists of a transformer-based encoder that generates a robust target representation by learning pixels interdependence. An adaptive network selection module controls the propagation routing path determining the optimal computational graph according to confidence-based criteria. We further propose a spatial adaptive attention network to avoid computational overhead in the transformer encoder, where the self-attention only aggregates the dependencies information among selected points. Our model achieves a harmonious proportion between accuracy and efficiency for dealing with varying scenarios, leading to notable advantages over static models with a fixed computational cost. Comprehensive experiments on aerial and prevalent tracking benchmarks achieve competitive results while operating at high speed, demonstrating its suitability on UAV-platforms which do not carry a dedicated GPU.

I. INTRODUCTION

Recent developments on visual object tracking techniques have facilitated their applications in a variety of fields such as path planning \cite{1}, visual surveillance \cite{2}, and border security \cite{3}. Among those, Unmanned Aerial Vehicle (UAV) tracking has drawn increasing attention for its flexibility and adaptability under intricate circumstances. Recent methods have focused on building more profound and accurate models to deal with challenging scenarios like fast motion, low-resolution, frequent occlusion, etc. However, most of those methods neglect to fulfill the real-time requirement on embedded platforms with limited computational resources.

In the visual tracking community, Deep Siamese networks play an important role and achieve remarkable progress on accurate tracking. Recent years have witnessed many successful Siamese models \cite{4}–\cite{9}. The recent research \cite{10}–\cite{12} on attention and transformer mechanisms further speeds up the process of building more powerful trackers. However, most prevalent deep Siamese trackers perform inference with a cumbersome computational graph or fixed network parameters, which constrains their deployment on mobile devices with limited computational power.

In this work, we aims to achieve a desired trade-off between tracking efficiency and accuracy for dealing with varying computational demand. We introduce a dynamic tracking architecture consisting of two adaptive components, i.e., adaptive attention and adaptive network selection, which adapts the network structures and parameters to the input during inference. As shown in Figure 1, tracking scenarios could be categorized into various levels according to their complexity and can be tracked with the corresponding inference stage. For canonical ("easy") samples where no background noise appears, a shallow network can correctly track the objects. With the occurrence of distractors or other challenging scenarios like fast motion and occlusion caused by the UAV and object relative motion, more computational resources will be allocated for more sophisticated models to recognize the objects. The selection is determined dynamically based on the prediction states from the previous stage without any prior knowledge. To selectively activate the model components, we estimate the Kullback-Leibler divergence between the predicted localization distribution and the expected one. This provides a criterion to determine whether to employ the next inference stage or directly output the results.

To increase the capability of discriminating between the
Object Tracking. The tracking methods can be divided into: a) Discriminative Correlation Filter (DCF) based trackers and b) deep learning-based trackers. DCF based trackers [13]–[16] could run with real-time speed on CPU, but their performance is constrained by the feature representation ability of handcrafted features. Other DCF tracker [17] introduce deep features to improve on accuracy but suffer in terms of computational speed. In contrast, deep learning based trackers, like the Siamese-based trackers, including anchor based trackers [7], [11] and anchor-free trackers [4], [6], [18], achieve remarkable enhancements in both accuracy and speed by utilizing a high-end GPU device. SiamAPN [8], SiamBAN [9] improves the performance by learning dynamic anchors or adaptive filters. Instead, our tracker adjusts the network architecture to reduce redundant computations.

Transformer. Transformer was first proposed for machine translation in [19] and shows great potential in many sequential tasks. Recent works [10] employ the attention mechanism for more robust tracking. Aerial trackers, SiamAPN++ [11] and HiFT [12], make use of transformer to fuse feature maps for better discrimination and achieve SOTA performance. While those trackers concentrate on building a more robust representation with transformer, our tracker focuses on reducing the intrinsic computational redundancy.

III. SIAMATN FRAMEWORK

In this section, we describe the proposed SiamATN framework. As shown in Figs. 2, SiamATN consists of 4 parts: (1) a Siamese network backbone for feature extraction, (2) multiple spatial adaptive attention sub-networks for further feature encoding, (3) the corresponding prediction head for classification and regression and (4) the adaptive network selection module for selectively activating model components.

A. Siamese Networks for Visual Tracking

The Siamese network consists of two parameter sharing branches, i.e., the template branch, which takes the initial cropped frame (denoted as \( z \in \mathbb{R}^{H_t \times W_t \times 3} \)) as reference image and the search branch, which processes the current frame (denoted as \( x \in \mathbb{R}^{H_t \times W_t \times 3} \)) for tracking. The Siamese backbone (denoted as \( \phi (\cdot) \)) performs the same projection on the input \( z \) and \( x \) and outputs a common embedding feature space \( \phi (z) \in \mathbb{R}^{C \times H_x \times W_x} \) and \( \phi (x) \in \mathbb{R}^{C \times H_x \times W_x} \) for subsequent tasks, where \( r \) is the downsample ratio. Specifically, we employ ShuffleNet [20] as the backbone, considering its computational efficiency on mobile devices and extract the feature maps from conv4 with a spatial downsample ratio \( r = \frac{1}{16} \). The lightweight backbones are insufficient for extracting robust discriminative features, which is vital for the tracking performance, especially under uncertainty scenarios. To alleviate this problem, we apply additional encoding blocks to reinforce the feature representation. For an input sample \( x \) (or \( z \)), the forward propagation of an L-stages encoding network (Sec. III-B) can be formulated as:

\[
M^L_x = \mathcal{F}^L \circ \mathcal{F}^{L-1} \circ \cdots \circ \mathcal{F}^1 (\phi (x))
\]

where \( \mathcal{F}^\ell \) is the encoding network at stage \( \ell, 1 \leq \ell \leq L \). Considering the diverse computational demands for different tracking scenarios, we may terminate the inference procedure at an intermediate stage. Specifically, the correlation features on stage \( \ell \) can be represented as:

\[
M^\ell_x = \mathcal{F}^\ell \circ \mathcal{F}^{\ell-1} \circ \cdots \circ \mathcal{F}^1 (\phi (x)), 1 \leq \ell \leq L
\]

where the \( \ell \) is determined base on the adaptive router network (Sec. III-C) and \( M^\ell_x = \phi (x) \). Afterwards, a depth-wise cross-correlation is performed between \( M^\ell_x \) and \( M^\ell_t \) as \( R^\ell = M^\ell_x \ast M^\ell_t \), where \( \ast \) denotes the depth-wise correlation operation and the \( R^\ell \) is a multi-channel response map which is adopted as the input of the prediction head. Inspired from [18], the correlation response map \( R^\ell \) is fed into two parallel branches: one for object classification and centerness prediction and another for bounding box regression. Each branch consists of 3 stacked convolution layers to generate the final results \( A^\ell_{cls}, A^\ell_{cen} \) and \( A^\ell_{reg} \), where \( A^\ell_{cls} \) represents the foreground and background probability score map, \( A^\ell_{cen} \) denotes the predicted centerness scores and \( A^\ell_{reg} \) predicts the distances from each feature point to the four sides of the bounding box.
B. Spatial Adaptive Attention Network

Given the embedding feature \( \phi(x) \), the encoder network \( \mathcal{F} \) is designed to learn a robust appearance model. However, the Convolution layers are insufficient to learn global dependencies which may degrade the model capacity for localizing the target objects under complex scenarios. Accordingly, we explore the multi-head self-attention mechanism to capture the spatial relationship between feature points and generate discriminative features for target localization. Specifically, let \( x_p^\ell \in \mathbb{R}^d \) denotes an element point of \( M_p^\ell \), where \( p \) is the spatial position. The attention function is performed on query vector \( q_p^\ell \), key vector \( k_p^\ell \) and value vector \( v_p^\ell \) which are learned from \( x_p^\ell \) as:

\[
q_p^\ell = W_q^\ell x_p^\ell, \quad k_p^\ell = W_k^\ell x_p^\ell, \quad v_p^\ell = W_v^\ell x_p^\ell
\]  

where \( W_q^\ell, W_k^\ell, W_v^\ell \in \mathbb{R}^{d \times d} \) are learnable weights. By dividing the embedded vector into \( N \) parts, each part represents an attention head. And the final attention feature \( \hat{x}_p^\ell \) is calculated by:

\[
\hat{x}_p^\ell = \text{Concat}([\sum_{p=1}^{N} \sigma \left( \frac{\langle q_p^\ell(i) \rangle^T k_p^\ell(i) \rangle}{\sqrt{d_{\text{head}}}} \right] \cdot v_p^\ell(i)]^N) W_p^\ell
\]  

where \( W_p^\ell \in \mathbb{R}^{d \times d} \) is the learnable projection matrix, \( d_{\text{head}} \) is the dimension of each head, equal to \( \frac{d}{N} \) by default. \( \mathcal{F}^\ell \) is the sample space on \( \ell \) stage which provides reference points as keys and values and is set to \( M_p^\ell \) in full attention scheme. \( \sigma \) is the softmax function and Concat refers to the concatenation operation. The query \( q_p^\ell \) is compared with every key point \( k_{p'}^\ell \) on \( \chi^\ell \) and extract useful information from the corresponding value \( v_{p'}^\ell \). The final output of the multi-head attention layer is given after a fully connected feed forward network as:

\[
\mathcal{F}^\ell(x_p^\ell) = x_p^\ell + \hat{x}_p^\ell + \rho(W_2^\ell \rho(W_1^\ell(x_p^\ell + \hat{x}_p^\ell)))
\]  

where \( W_1^\ell \) and \( W_2^\ell \) are learnable linear transform weights and \( \rho \) is activation function. The same network and parameters are applied on \( M_p^\ell \) to generate the correlation filter. And each \( \mathcal{F}^\ell \) consists of \( D = 2 \) multi-head attention layers.

Despite the strong discriminability provided by the self-attention module, its computational and memory costs are relatively high when traversing the whole feature map. Additionally, most reference points are irrelevant to the query point but only deliver noisy background information. To address this problem, we introduce the spatial attention module, which adaptively selects the reference points based on the response map from previous layer. By ranking \( A_{cen}^\ell \) according to the response score, only the pixels which share a similarity with the target object are taken as the reference features:

\[
\chi^\ell = [M_p^\ell(i, j, \cdot)]_{A_{cen}^\ell(i, j) > \tau}
\]  

Only the points with scores higher than the threshold \( \tau \) during tracking are selected as the reference points in \( \chi^\ell \). For batch processing efficiency during training, we replace the above criterion with top-k similarities, i.e., \( (i, j) \in \text{TOP}(A_{cen}^\ell, K) \). The selected points serve as key and value features. Then the attention is operated between the query vectors and the reference points and its computational complexity is linear to \( K \). Thus, we model the pixels interdependencies in a computational efficient way.

C. Adaptive Network Selection

To adapt the network structure to the input during the inference, we set up the adaptive routing network to select the appropriate depth of encoding layers and the corresponding classifier. We utilize the background appearance information learned from the prediction heads to estimate the tracking complexity. For stage \( \ell \) at frame \( t \), we estimate the divergence
between the predicted target localization distribution $q_t$ and the expected distribution $p_t$ with:

$$KL(p_t || q_t^*) = \int_{\mathcal{X}} p_t(x) \log \frac{p_t(x)}{q_t^*(x)} dx$$  \hspace{1cm} (7)$$

where $KL$ is the Kullback-Leibler divergence and $\mathcal{X}$ is the sample space which equals to the size of the feature maps, as shown in Figure 3. In practice, we choose the predicted and ground truth centerness score map as $q_t^*$ and $p_t$ respectively. Note that $A_{cls}$ and $A_{reg}$ are not calculated at this stage. While $p_t$ can be approximated from $\hat{b}_t$, it may bring target drifts caused by fast motion, camera movement, etc. Therefore, we apply the Kalman filter to correct the observation. Given the estimated state $\hat{b}_t$ from the Kalman filter, the optimal stage $\ell_t^{opt}$ of the encoding network $\mathcal{F}$ is selected by:

$$\ell_t^{opt} = \min \left( \left\{ \ell : \max \left\{ 0, KL(p_t || q_t^*) - \eta \right\} = 0 \right\} \right)$$ \hspace{1cm} (8)

Where $\eta$ is the threshold to control the routing path.

### D. Training Objectives

Our model is trained in an end-to-end way, where the training objective is a weighted average loss for the prediction branches on each stage:

$$\mathcal{L} = \sum_{\ell=1}^{L} \mathcal{L}^\ell$$ \hspace{1cm} (9)

And the loss on stage $\mathcal{L}^\ell$ is:

$$\mathcal{L}^\ell = \lambda_{cls} \mathcal{L}_{cls}^{\ell} + \lambda_{cen} \mathcal{L}_{cen}^{\ell} + \lambda_{iou} \mathcal{L}_{iou}^{\ell} + \lambda_{reg} \mathcal{L}_{reg}^{\ell},$$ \hspace{1cm} (10)

where $\mathcal{L}_{cls}$ is the cross-entropy loss for classification, $\mathcal{L}_{cen}$ is the binary cross entropy loss for the centerness score, $\mathcal{L}_{iou}$ is the GIOU [21] loss between prediction boxes and the ground truth box and $\mathcal{L}_{reg}$ is the $L1$ loss for regression. Constants $\lambda_{cls}$, $\lambda_{cen}$, $\lambda_{reg}$ and $\lambda_{iou}$ weight the losses.

### IV. EXPERIMENTAL AND SIMULATION STUDIES

#### A. Implementation Details

The backbone, i.e., ShuffleNet [20] is pre-trained on Imagenet [22] and the $conv$ 4 of with depth of $C = 232$ is extracted for further encoding. We add a convolution layer to reduce the feature dimension into $d = 192$. The network is trained offline for 100 epochs with 128 image pairs per batch. The patch pairs $z$ and $x$ are cropped from two images of the same video with a maximum gap of 100 frames and are resized into $80 \times 80$ pixels and $320 \times 320$ respectively. The training data consists of the training splits from LASOT [23], TrackingNet [24], Got10K [25] and COCO [26]. We set up spatial adaptive attention stages number $L = 3$ as it provides enough representation power for complex cases. For search branch, We take $K = 100$ all stages. We use the ADAMW [27] optimizer with an initial learning rate of $10^{-5}$ for the backbone parameters and $10^{-4}$ for rest of components. During training, the first layer and all BatchNorm layers from the backbone are frozen. The learning rate drops by a factor 0.1 on 90 epochs. For all stages, the prediction losses are weighted with $\lambda_{cls} = 5$, $\lambda_{iou} = 5$ and $\lambda_{reg} = \lambda_{cen} = 2$ respectively. During tracking, $\eta$ is set to 0.1. All experiments are conducted on a laptop with an Intel i7-9750H CPU and an NVidia 2060 for GPU speed test.

#### B. Evaluation on Visual Tracking Benchmarks

In this section, we compare our approach with 15 SOTA trackers. There are 3 anchor-based Siamese methods (SiamRPN++ [5], DaSiamRPN [7], SiamAPN [8]), 4 anchor-free Siamese methods (SiamFC [4], SiamFC++ [6], SiamBAN [9] and SiamCar [18]), 5 DCF based methods (ECO [15], fDSST [14], KCF [13], CSRDCF [28], CCOT [17]) and 3 attention based methods (SiamGAT [10], SiamAPN++ [11], HiFT [12]).

**UAV123 [29]** is one of the largest UAV tracking benchmarks, including 123 low altitude aerial videos with more than 112K frames, and adopts success and precision metrics for evaluation. We report the performance diversity when using different stages for tracking. As shown in Figure 4, the combination of a lightweight backbone with one attention layer, i.e., SiamATN-stage1 brings remarkable performance enhancement over trackers based on deep networks (Retinanet50 [30]) or handcraft features (ECO [15]) especially on precision score. With two attention blocks, our tracker achieves comparable results with SOTA free-anchor Siamese tracker SiamBAN [9]. Introducing adaptive network selection won’t effect the tracking performance, but help in achieving more stable results with a precision score of 85.2% and success score of 65.0%. The overall results demonstrate that SiamATN achieves superior performance against other SOTA trackers.

**UAV20L [29]** contains 20 long-term sequences with an average of 3k frames per sequence. As shown in Table I, the classic DCF trackers based on the handcrafted features run at real-time speed on the CPU but have limited accuracy.
In contrast, deep trackers relying on CNNs can achieve high performance but are only applicable on GPU devices. Instead, our SiamATN runs at real-time speed (34Hz) on the CPU while obtaining SOTA results. Specifically, SiamATN gains a precision score of 86.5% and an AUC score of 68.2%, outperforming the recent SOTA Siamese aerial tracker HiFT. Similar to UAV123, we also report the performance and speed under different settings. The tracker without adaptive module gives the best AUC score but costs an extra 2× inference time.

**C. Ablation Analysis**

**Speed, FLOPs and Params.** Table II illustrates the complexity analysis of the proposed tracker. For reference, SiamRPN++ [11] has almost 60G multiply–accumulate operations which is too heavy to run fast on the CPU. The adaptive network selection enables the advantage of the lightweight backbone and spatial adaptive attention, and fulfills the balance between the model complexity and the inference speed without adding extra operations.

**Visualization of the adaptive network selection.** Figure 5 shows a tracking sequence with its KL-divergence score for each frame. Only stage 1 is used for tracking when no distractors appears in the background. Otherwise, stage 2 and stage 3 will be triggered to deal with hard negative examples and keep the focus on the tracking object until the background is clear again. Thus, our model allocates model components on demand at test time, leading into a notable advantage in the computational efficiency while maintaining a high accuracy.

**D. Experimental Field-Test**

The field tests are set to: (1) track a fast-moving drone with ground PTZ camera (first video), (2) track a moving person with flying UAV, and (3) track a drone with PTZ camera which is mounted on a flying drone. Those configurations are quite challenging due to a several factors including camera motion, out-of-view, motion blur, scale variance, partial occlusion and object deformation. Figure 6 shows the precise tracking results obtained in complex environments and constrained power resources in real-time on CPU, exhibiting the robustness and practicability of the tracker in real-world applications; the blue lines illustrate the FPS changes during tracking.

**V. CONCLUSIONS**

In this work, a dynamic tracker is implemented achieving real-time speed on CPU with high accuracy. The network selection module adaptively determines the optimal computational graph based on the tracking complexity. The experiments both on aerial benchmark and real-world tests demonstrate its effectiveness and portability on UAV-tracking.

**REFERENCES**

Fig. 6: Visualization of real-world UAV tracking.