Improving Visual Object Tracking using Deep Reinforcement Learning

Dineth Gunawardena
Texas A&M University
934007587

Wahib Kapdi
Texas A&M University
635009343

Github URL: https://github.com/tamu-edu-students/SLTRLProject
Youtube URL: https://youtu.be/75LUAj6q2V8

***Abstract*—Most object tracking works have focused on frame-level training which leads to differences in data distrubtions with training and testing time. This is due to the sequential nature of video sequences that isn't accounted for in the frame-level training and is often reflected in the accuracy limitations. This work explores the sequential nature of object tracking by desining an MDP and using deep reinforcement learning to process the object tracking data and train a model using the sequential nature of the video tracking data. We conduct an experiment on the TransT transformer feature-fusion model with the GOT-10k video tracking dataset. We used supervised learning as a baseline and performed an in-depth analysis to examine the differences after adding reinforcement learning.**

***Index Terms*—visual object tracking, deep reinforcement learning**

I. Introduction

Visual Object Tracking is vital in numerous computer vision applications like autonomous driving, surveillance, and augmented reality. Accurate tracking of an object's position across video frames is essential for interpreting dynamic scenes in real time. Most works have tackled the issue of object tracking using frame-level testing by giving the bounding box information for an object during training time [1], [2]. This leads to better spatial inference on a per frame basis which can show us great results for object detection. However, object tracking is sequential in nature and this isn't accounted for in the previous state-of-the-art works due it being on a per frame basis. If the object becomes occluded and goes behind a pole in the next frame, the model has no context of the temporal information and would usually give a false positive. This leads to lower performance in real testing environments because of the dynamic scene changes which aren't captured in a controlled training set. The mismatch in distributions from the training set and testing set can also come from several other factors in addition to occlusion such as the speed of the object, other objects entering the frame. Supervised models typically use the previous frame's information (bounding box information and image patch prediction) to gain more spatial context of the object in the current frame, but this has not been feasible with occlusion and other dynamic changes in object tracking environments.

Deep reinforcement learning has gained some attraction within object tracking as it utilizes a Markov Decision Process to capture the sequential nature of a system. A Markov Decision Process can be used for an object tracking model by allowing it to explore which actions will lead to better rewards over time. The goal of an MDP framework is to maximize the long-term reward by considering the entire sequence of actions and states rather than just the action/state for a given frame. With the use of deep learning, we can use the computational efficencies from PyTorch to process the data in each from and make a prediction based on the weights updated from reinforcement learning. Previous works have been explored within this domain area [3]–[5] which account for exploring object tracking in a sequential manner but they still come with limitations in regards to real data distributions. The works from [3], [5] uses reinforcement learning to update their action policies but still heavily relies on frame-level dependencies. There have been works which include a history of multiple actions in each training step [4], but these actions are discrete and don't fully account for the stochastic nature of bounding box predictions.

We explore using deep reinforcement learning with guidance from the work, "Towards Sequence-Level Training for Visual Tracking", [6] to enhance the sequential context within object trackers. This work utilizes object-tracking simulations to sample multiple augmented data distributions and account for various scenes within object tracking. They utilize the REINFORCE [7] algorithm as explained in Section IV. We utilize the sequence level training approach from [6] which takes into account all the previous frames in a sequence along with their bounding boxes before making a new prediction on the next frame. The loss function within this paper also takes into account of two separate trackers which perform both exploration and exploitation within a continuous action space.

DISCLAIMER: We do not claim any of the work for sequence level training to be ours. The proposed algorithm and training approach is implemented by Minji Kim and Seungkwan Lee et. al in [6].

For our project, we explore the affects of sequence level training on a model compared to using it with frame-level training. Our project builds on the codebase from "Towards Sequence-Level Training for Visual Tracking," which utilizes

the GOT-10k video traking dataset for robust sequence-level training [6]. We run the Transt transformer model [8] with sequence level training [6] and are able to obtain an average overlap score [9] and success rates which are thresholded at 50% and 75% respectively. These metrics are provided by the GOT-10k evaluation server which requires a specific data format for the resulting tracking data [10]. We further expand on sequence level training through an exploration of the reinforcement algorithm: A2C [11].

In this report, we first go over related work in Section II and background information on reinforcement learning in Section IV. We then describe our methodology and experiment in Sections V and VI respectively. Lastly, we close the report with a conclusion in Section VII and discuss future work in Section VIII.

II. Related Work

Visual object tracking has been a mature problem within computer vision and various techinques have been explored to solve it. Recently, reinforcement learning has been used within several works to improve the accuracy for visual object tracking in deployment

A. Deep Reinforcement Learning for Visual Object Tracking

Recurrent neural networks have been known to account for temporal data within a deep neural network by keeping track of previous data within memory. The work proposed by [3] uses a convolutional recurrent neural network to predict the location of an object, given a sequence of video frames. This is done by first using an observation network, which concatenates both the frame and the location of the bounding box into a feature vector. The bounding box uses the ground truth location for the first frame but otherwise keeps it as 0 for the other frames. This feature vector is then passed into the recurrent neural network which updates its own internal hidden state h_t in memory based on the previous hidden state h_{t-1} and the observation vector. The agent then outputs the location which can be extracted from the hidden state.

Compared to the work we are following in [6], they both use the same algorithm, REINFORCE, and take into account of sequential data. However they have several differences where [6] uses a more structured evaluation with sampled and argmax trackers to learn from both deterministic and stochastic predictions whereas [3] uses predictions across multiple episodes for its reward function to update the model parameters and takes more of an exploratory approach. Both approaches are trained offline, however [3] evaluates rewards at a frame level where [6] evaluates its reward function at a sequence level using average overlap.

B. Tracker as Reinforcement Learning Agent

There have been a few works where the tracker itself is a reinforcement learning agent. There is one that uses an action decision network (ADNet) that defines their states in the MDP as the target within a bounding box and the history of previous actions [4]. The policy network in ADNet takes as input an image patch cropped from the location of the previous state and produces an output of action probability distributions (includes both translation and scale changes). To select an action the agent utilizes reinforcment learning with tracking simulation and policy gradient updates. During the simulation, the agent generates a series of states $\{s_{t,l}\}$, actions $\{a_{t,l}\}$, and rewards $\{r(s_{t,l})\}$ over a sequence of frames. Through reinforcement learning with the tracking simulation, the agent can learn the action dynamics of the tracking target. The action space comprises 11 types of actions, including directional moves and scale changes, encoded in a one-hot vector. The next state is determined by updating the bounding box based on defined movement rules. The action dynamics in ADNet d_t are updated in each iteration by accumulating the most recent k actions and applying a first-in-first-out (FIFO) strategy. The main differences between this work and [6] are from the way the MDP is defined and how tracking is conducted. In [4], they use the states to include action history with a discrete action space as well as ground truth labels for each frame. The work in [6] only requires one ground truth label and includes the sequences of frames with the bounding box as the state space. They also take IOU [12] as the reward from the terminal state and require pre-defined actions for box adjustment which makes it difficult to scale for generic tracking. Within [6], the solution is approached with more generality by specifying a stochastic action space, using cumulative performance with average overlap for the rewards, and takes in the whole video sequence in the state space. For the similarities, both [4] and [6] use simulated training sequences to help update the model parameters which can improve the sequence level performance.

C. Iterative Shift for Visual Tracking

Rather than classifying many candidate locations, some trackers use reinforcement learning to predict iterative shifts of the bounding box, which is efficient and robust to large deformations. In the work by [13], they use an actor-critic network with iterative shift adjustments of the bounding box to mitigate errors that come from using the whole sequence in object tracking. More specifically they use three sub-networks in their iterative shift approach: 1) the actor network, 2) the prediction network, and 3) the critic network. They begin by extracting the deep feature $f \in \mathbf{R}^{1\times 512}$ from the fc4 layer, using the initial bounding box of the target. Next, they concatenate the feature of the candidate box f with the current target feature $f^* \in \mathbf{R}^{1\times 512}$. The prediction network ψ generates a shift δ, which is utilized by the actor network θ. When they choose the action *continue*, they modify the bounding box of the target based on the output δ from ψ. In the case of the action *stop and update*, they halt the iteration and update both the appearance features of the target and the parameters of ψ, while for the action *stop and ignore*, they refrain from any updates. If they select the action *restart*, it may indicate that the target has been lost, prompting them to re-sample the initial bounding box. During the training phase, they employ a deep critic network to assess the Q-

value of the current actions associated with δ, allowing them to fine-tune both the prediction network ψ and the actor network θ. The main difference between this framework and the one proposed from [6], is that one is using an actor-critic network to adjust the bounding box locations whereas the other is utilizing the performance of two trackers to determine the sequence level performance with policy gradient updates. Sequence level training [6] explains a training strategy where a video is sampled randomly, and the performance of the tracking methods is evaluated in terms of their ability to localize a target during occlusion. Iterative shift prediction [13] introduces specific neural network components like the prediction network (ψ) and the actor-network (θ), along with the critic network for estimating Q-values, providing a more detailed look at the architecture and functions.

D. Actor Critic for Visual Object Tracking

Actor-critic networks have also been explored within deep learning for visual object tracking to increase real-time performance. The first known work for this modeled an actor-critic network using an actor, which directly makes the tracker move the bounding box to the object's location in the current frame and a critic, which outputs a Q-value to guide the learning process of both 'Actor' and 'Critic' deep networks [5]. The 'Actor' component is designed to produce a single continuous action that directs the tracker to adjust the bounding box to the object's position in the current frame. Throughout the tracking procedure, the 'Critic' evaluates the action taken by the 'Actor' to assess its quality, thereby enhancing the overall tracking performance. The main differences from this work and [6], are the framework, training method and decision making function.

The actor-critic framework [5] outputs a continous action and adjusts the bounding box based on the current state. The sequence level training framework [6] takes into account the whole sequence and maximizes a performance metric over a sequence of predicted bounding boxes. For the training method, [5] includes an offline training phase before deploying the tracker online, integrating a double bounding box scheme to improve accuracy. Sequence level training [6] uses the REINFORCE algorithm which has policy gradient updates for sequence level training. Lastly, the decision making function from [5] has continuous action spaces and emphasizes real-time adjustments, directly linking the action to the observed state of the bounding box. In [6], Self-Critical Sequence Training (SCST) is utilized, where two trackers (sampling and argmax) are used to generate rewards that inform the learning process for the next action.

E. Deep Reinforcement Learning with Expert Demonstrator

There have been works which explore using an expert tracker to guide the RL tracking agents [12]. In [12], they use a real-time CNN-based tracker called A3CT, which is trained through an end-to-end reinforcement learning approach that leverages demonstrations from a leading tracker. They also use a real-time CNN-based tracker referred to as A3CTD, which enhances its performance during the tracking phase by utilizing the reinforcement learning functions acquired during training, in conjunction with expert guidance. Their tracking problem is framed as a Markov Decision Process (MDP), which includes states, actions, transitions, and rewards. In this framework, states are defined as a pair cropped image patches from consecutive frames, derived from bounding box predictions. Actions are represented as vectors that specify adjustments to these bounding boxes. To evaluate the performance of the tracker, rewards are calculated using the Intersection-over-Union (IoU) metric, which measures the overlap between predicted and ground-truth bounding boxes, thus encouraging accurate localization. Furthermore, the method incorporates expert demonstrations to guide the learning process, providing a foundation for effective training. The tracking agent itself is built upon a deep neural network (DNN) architecture, which is designed to learn policies and value functions based on these state representations. They establish an A3C framework in which the first half of the learning agents engages in conventional A3C training, while the second half is trained to replicate the actions of the expert tracker demonstrator through a supervised approach. This has some differences with [6] as they define the state space to include a pair of frames rather than all the previous ones we've seen in the sequence. Furthermore, [12] seems to use a semi-supervised learning manner with ground truth bounding boxes in their reward function for each state whereas [6] uses reinforcement learning after given an intital frame ground truth bounding box. Lastly, [12] uses an expert demonstator with A3C training to update the shared weights through guidance of an expert tracker. The work from [6] has some similarity in the sense they use a guiding argmax tracker for the sampled tracker with shared model weights.

F. Hierarchical Deep Reinforcement Learning

One work explores using Actor-Critic Networks and proposed a Hierarchical Deep Reinforcement Learning (HDRL) framework for visual object tracking called PACNet [14]. PACNet has three main components a Siamese-based observation network, a policy network for deciding tracking mode, and an actor-critic network for bounding box regression. PACNet [14] and Sequence-Level Training [6] both seek to enhance visual tracking but differ fundamentally in approach, training methods, and model modifications. PACNet introduces a hierarchical deep reinforcement learning (HDRL) framework, implementing separate policy and actor-critic networks to improve the decision-making process in tracking. It is trained offline using A3C and policy gradient algorithms and achieves performance on datasets like OTB-100 [15], UAV-123, LaSOT [16], VOT-2019, and GOT-10k [10]. In contrast, Sequence-Level Training aims to bridge the gap between frame-level training and sequence-level testing without altering model architectures. Instead, it proposes sequence-level training strategies that focus on data sampling, learning objectives, and data augmentation to improve existing models, such as SiamRPN++, SiamAttn, TransT, and TrDiMP. Sequence-level training has shown improvements on datasets including

LaSOT, TrackingNet, and GOT-10k. In essence, while PACNet introduces a novel framework to redefine tracking, Sequence-Level Training enhances existing architectures through targeted training strategies. Both approaches aim to improve visual tracking performance, but [14] focuses on a new HDRL framework, while the [6] improves existing models through training strategies.

G. Tracker as DQN Agent

Apart from A3C another strong candidate for training visual object tracking agents is Deep Q Learning as proposed by [17]. Specifically, the authors propose a Deep Q-Network (DQN) [17] agent integrated with the AirSim simulation platform to enhance object tracking capabilities. The model leverages a recurrent neural network (RNN) architecture to extract sequential features from video frames, enabling accurate prediction of object location over time. The DQN training utilizes a replay memory unit to store and sample transitions, which aids in refining the model's performance. Tracking accuracy is evaluated by calculating the intersection over union (IoU) [12] of bounding boxes predicted by the agent's actions. The simulation setup employs a Markov Decision Process (MDP), with states representing the drone's environmental observations, actions as potential movements or camera adjustments, rewards driven by successful object tracking (measured through IoU metrics), and transitions governed by the virtual environment's dynamics. Notably, the paper highlights that the model achieved superior speed and accuracy over conventional methods when evaluated on the VisDrone2019 [18] and OTB-100 [15] datasets. However, [6] does show some shortcomings of using frame-level tracking and we will not be using this in our project.

H. Visual Tracking as a Sequence Generation Problem

Another interesting technique to solve Visual Object Tracking is mentioned by [19]. The author uses an encoder-decoder transformer architecture called SeqTrack [19]. The encoder, based on a modified Vision Transformer (ViT), extracts visual features from video frames, while the decoder, a causal transformer with masked self-attention, autoregressively generates bounding box coordinates. This process converts bounding box values into discrete tokens and utilizes a straightforward cross-entropy loss function. A key distinction from previous methods is the elimination of complex head networks, such as classification and regression heads, simplifying the tracking framework. This method also employs a single loss function, avoiding the need for multiple task-specific losses. In terms of architecture, causal masking ensures autoregressive generation and cross-attention layers enable the effective incorporation of visual features. The model achieves state-of-the-art results on several tracking benchmarks, like the LaSOT and GOT-10K datasets, offering a strong balance between speed and accuracy compared to existing approaches. Additional features include online template updating and window penalty during inference, with a consistent image size for template and search images, unlike previous models. The approach presents four model variants, differing in encoder architectures and input resolutions, providing flexibility for varied tracking needs.

III. Application of Visual Object Tracking with RL

A. Visual Tracking with Camera Movement

The majority of the work in our project includes object tracking with video input and generating a bounding box. It is worth mentioning that similar techniques extend to using camera movement for object tracking as shown in [20]. The paper introduces an end-to-end active object tracking system based on deep reinforcement learning, leveraging a ConvNet-LSTM architecture to translate raw video frames into camera control actions directly. The model is trained in virtual environments, specifically ViZDoom and Unreal Engine, using the A3C reinforcement learning algorithm and employs environment augmentation techniques to improve generalization. This setup enables the model to perform effectively on unseen scenarios, with promising implications for real-world tracking applications. The system is structured as a Markov Decision Process (MDP) where the state consists of raw RGB video frames from the agent's first-person view, and the action space is a discrete set of six actions: turning left/right, moving forward, turning left/right while moving forward, and no operation. The reward function incentivizes keeping the target object centered and at an optimal distance, defined by

$$r = A - \left(\frac{\sqrt{x^2 + (y-d)^2}}{c} + \lambda |\omega| \right),$$

where (x, y) represents the object's position in the agent's local coordinates, ω is the object's orientation, and A, c, d, λ are tuning parameters. Ultimately, the tracker generalizes well across varying object movements, appearances, and backgrounds, surpassing traditional passive trackers with hand-tuned camera control, and demonstrates the potential for real-world application, validated through VOT dataset videos and deployment on a real robot. Another difference between our project and this one is that we are trying to use SLT [6] and apply it using different RL algorithms from [21] whereas [20] uses A3C for environment augmentation.

B. Visual Object Tracking for Drones

Much like camera movement visual object tracking using reinforcement learning can be used for applications as shown by [22]. This paper introduces a deep reinforcement learning (RL) based approach for visual object tracking in drone images. The authors propose four models (Model-A, B, C, and D) that build upon the ADNet baseline, introducing modifications to improve tracking performance on drone datasets. Key contributions include a novel reward function, new action types specific to drone imagery, and testing on both VisDrone2019 and OTB-100 datasets. The models use a Markov Decision Process strategy and are trained in three stages: supervised learning, reinforcement learning, and online

adaptation. Technical details include state and action definitions, reward functions, and network architectures. Model-C introduces a hybrid reward function and updates network parameters after each video clip, while Model-D uses ADNet's original reward function. Results show improvements over the baseline, with Model-D demonstrating the best overall performance, achieving up to 3.87% improvement in precision on VisDrone2019 [18] and 3.15% improvement on OTB-100 [15]. The paper highlights the challenges of tracking in drone imagery and the importance of adapting existing trackers for this domain. The main differences between this work and [6] come from the fact that sequence level training is centered on optimizing overall tracking performance through sequential decision-making, while the action-sequence-based tracker emphasizes specific model architectures in context of action selection and feature extraction. The sequence level training framework primarily leverages sequence-level rewards, while the action-sequence tracker introduces varying reward functions and updates based on different model configurations.

IV. Background

Within this section, we discuss two main reinforcement learning algorithms explored within the report. The first is REINFORCE [7] which is used by [6] to incorporate the policy gradient theorem with Monte-Carlo. The second reinforcement learning algorithm is A2C [11] which adds a critic network that evaluates the actions taken by the actor in each time step. This incorporates boot-strapping into the policy update by evaluating the actor and using the critic value function as policy improvement. Both approaches have their own uses which are further discussed in this section. All equations from Section IV are taken from "Reinforcement Learning" by Richard Sutton and Andrew Barto [23].

A. REINFORCE

The algorithm used in [6] is the REINFORCE method which uses the policy gradient theorem to update the policy of continuous action spaces. The policy gradient theorem starts with a performance metric known as $J(\theta)$, which is a scalar measure with respect to the policy parameter. Performance is defined through the following formula:

$$J(\theta) = v_\theta^\pi(s_0) \tag{1}$$

where v_θ^π is the true value function for the policy [23]. However the performance of a continuous and differentiable policy would depend on both the action selection and the state distribution of those selections. The policy gradient defined through [23] demonstrates how to establish this relationships with the policy gradient theorem:

$$\nabla J(\theta) = \sum_s \eta(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) \tag{2}$$

where $\eta(s)$ is the number of times s was visited in a given episode. This equation is further normalized with a probability distribution as $\eta(s)$ depends on the episode length:

$$\eta(s) \propto \frac{\eta(s)}{\sum_{s'} \eta(s')} \approx \mu(s) \tag{3}$$

$$\nabla J(\theta) \propto \sum_s \left[\mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) \right] \tag{4}$$

To obtain the monte-carlo policy gradient we can use the same approach as stochastic gradient ascent where the expectation of the gradient is proportional to the actual gradient:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s) = \mathbb{E}_\pi \left[\sum_a q_\pi(s_t,a) \nabla_\theta \pi(a|s_t) \right] \tag{5}$$

$$\theta_{t+1} = \theta_t + \alpha \sum_a \hat{q_\pi}(s_t,a) \nabla_\theta \pi(a|s_t;\theta) \tag{6}$$

However, Equation 6 uses $\hat{q_\pi}$ which is an approximation to q_π. This approximation can be taken out by multiplying and diving the equation by $\pi(a|s_t,\theta)$ which eventually results in the following parameter update:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t)}{\pi(A_t|S_t)} \tag{7}$$

.

G_t from here is the full rewards from time t until the end of the episodes. This is sampled on each step to help update the policy parameters which are used to improve the policy after each episode.

The REINFORCE algorithm can also be updated with the use of a baseline $b(s)$ to further normalize the returns. Because the baseline can be any function it will be valid as the action probabilities all sum up to 0:

$$\theta_{t+1} = \theta_t + \alpha \left(G_t - b(S_t)\right) \frac{\nabla_\theta \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{8}$$

The algorithm from [23] is shown in Algorithm 1 using REINFORCE with baselines. θ and w are parameters for the policy and value functions respectively.

B. A2C

Actor-critic methods take advantage of a concept in reinforcement learning known as bootstrapping where the state value estimates are updated based on the previous time step state value estimates. This is different from using a baseline in REINFORCE which uses an unbiased baseline value and tends to learn slowly [23]. The one-step actor critic method shows how to transform REINFORCE using baseline value to a temporal difference update. We can take out the returns from the whole episode and instead use an advantage function that measures the benefit of taking an action a at state s while following a policy π:

$$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{9}$$

This advantage function can be included into the policy parameter update as follows:

Algorithm 1 REINFORCE with Baseline (episodic), for estimating π^θ

Input: A differentiable policy parameterization $\pi(a|s,\theta)$
Input: A differentiable state-value function parameterization $\hat{v}(s,w)$
Algorithm parameters: Step sizes $\alpha_\theta > 0$, $\alpha_w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d_0}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)
while true **do** ▷ For each episode
  Generate an episode $S_0, A_0, R_1, \ldots, S_T, A_T, R_T$, following $\pi(\cdot|\cdot,\theta)$
  for each step of the episode $t = 0, 1, \ldots, T-1$ **do** ▷ For each time step
    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ ▷ Return starting from time step t
    $G_t \leftarrow G_t - \hat{v}(S_t, w)$ ▷ Subtract the current state-value estimate
    $w \leftarrow w + \alpha_w \nabla_w \hat{v}(S_t, w)$
    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \ln \pi(A_t|S_t,\theta) G_t$
  end for
end while

$$\theta_{t+1} = \theta_t + \alpha \left(r_{t+1} + \gamma \hat{v}(s_{t+1}; w) - \hat{v}(s_t; w)\right) \nabla_\theta \ln \pi(a_t|s_t;\theta) \tag{10}$$

where $r_{t+1} + \gamma \hat{v}(s_{t+1}; w) - \hat{v}(s_t; w)$ is the temporal difference error used in both the value function parameter and policy parameter update. The algorithm for A2C is shown below from [23] in Algorithm 2 where w and θ are parameter updates for the critic and actor respectively.

Algorithm 2 One-step Actor–Critic (episodic) for estimating π^θ

Input: A differentiable policy parameterization $\pi(a|s,\theta)$
Input: A differentiable state-value function parameterization $\hat{v}(s,w)$
Parameters: Step sizes $\alpha_\theta > 0$, $\alpha_w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d_0}$
Initialize state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)
while true **do** ▷ For each episode
  Initialize S ▷ First state of the episode
  while S is not terminal **do** ▷ For each time step in the episode
    Sample action $A \sim \pi(\cdot|S,\theta)$
    Take action A, observe S', reward R
    $R \leftarrow R + \hat{v}(S', w) - \hat{v}(S, w)$
    if S' is terminal **then**
      $\hat{v}(S', w) \leftarrow 0$
    end if
    $w \leftarrow w + \alpha_w \nabla_w \hat{v}(S, w)$
    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \ln \pi(A|S,\theta)$
    $S \leftarrow S'$
  end while
end while

V. Methodology

For this project, we use sequential level training as described in [6], which defines an MDP to account for interframe dependencies and a reward function that utilizes both exploration and exploitation. We further build off the Algorithm 3 from [6] and utilize A2C to modify sequence level training.

A. Markov Decision Process

In what sense is our problem sequential?
The problem is considered sequential because visual tracking involves predicting the position of an object across a series of frames in a video. For each frame, it will contain the object and its detected bounding box in the environment at that moment in time. As the object moves, the bounding box will change to match the new object's location. Due to this, the future state only depends on the current state properties. As the object in the frame moves, the states change and we follow a sequential process in predicting the future state.

What is our problem's state space?
For our problem, the state space comprises all the relevant information available at each time step t when the tracker makes its prediction:

1) **Current Frame**: The video frame v_t at time t provides the immediate visual context for tracking the object.
2) **Historical Frames**: The sequence of frames $(v_0, v_1, \ldots, v_t)$ gives the tracker access to previous visual information, which may be critical for understanding the object's movement and appearance over time.
3) **Initial Target Bounding Box**: The ground-truth bounding box g_0 of the target object in the initial frame v_0 serves as a reference point for tracking.
4) **Previous Estimates**: The previously estimated target states $(l_1, l_2, \ldots, l_{t-1})$ are included in the state space. This historical information is essential for understanding the trajectory of the object and informs the current prediction.
5) **Action-Dependent Information**: While not directly part of the state representation, the implications of previous actions (the predicted bounding boxes) are inherently part of the tracker's decision-making process, influencing the next bounding box prediction.

The state space at each time step t can be represented as:

$$o_t = (v_t, v_{t-1}, \ldots, v_0, g_0, l_1, l_2, \ldots, l_{t-1})$$

This state representation allows the tracker to make informed predictions about the current location of the target based on both immediate and historical context, effectively capturing the temporal dependencies crucial for successful tracking.

What is our problem's action space?
The action space in the problem context of sequence-level training for visual object tracking is defined by the set of actions the tracker can perform at each frame to estimate the

bounding box of the target object. Specifically, this involves selecting the coordinates of the bounding box around the target object in each frame, which includes continuous adjustments of size, position, and orientation based on previous frames. The tracker can utilize both deterministic and stochastic policies to decide on bounding box transformations, allowing it to track objects across frames while accounting for factors like occlusions and motion blur.

What is our problem's transition function?
The transition function in the context of sequence-level training for visual object tracking defines how the state of the tracked object changes from one frame to the next based on the action taken by the tracker. Specifically, the transition function determines the updated position, scale, and orientation of the target object's bounding box in the next frame, given the tracker's current estimated bounding box and any changes caused by the tracker's action (such as movement within the search window).

In this framework, the transition function reflects the sequential dependency between frames, where each predicted bounding box in the current frame serves as the basis for searching and localizing the object in the subsequent frame. This dependency is reinforced by the tracker's learned policy, which considers how present actions affect future tracking performance and allows adjustments for realistic scenarios like variations in object appearance, occlusions, or environmental changes.

What reward function will we use?
The reward function used in this sequence-level training for visual object tracking is designed to measure tracking performance across the entire sequence, specifically based on the *average overlap ratio* between the predicted bounding box sequence and the ground truth bounding boxes. This metric encourages the tracker to maintain consistent localization accuracy throughout the video.

In the context of reinforcement learning (RL), the reward function $r(\mathbf{l})$ for a sequence $\mathbf{l} = (l_1, \ldots, l_T)$ of estimated target states is used to directly optimize the sequence-level objective:

$$L(\theta) := -\mathbf{E}_{\mathbf{l}\sim\pi_\theta}\left[r(\mathbf{l})\right]$$

where π_θ represents the tracker's policy parameterized by θ, and $r(\mathbf{l})$ evaluates the sequence's tracking accuracy (e.g., using average overlap).

To optimize the expected reward, the tracker is trained with the *self-critical sequence training* (SCST) [24] approach, which utilizes a form of the REINFORCE algorithm. The SCST method compares the tracker's performance in two modes: 1. **Sampling mode** (taking stochastic actions), 2. **Argmax mode** (taking deterministic actions).

For each training episode, the estimated gradients are computed as follows:

$$\nabla_\theta L(\theta) \approx -\left(r(\mathbf{l}) - r(\mathbf{l}')\right)\nabla_\theta \log p_\theta(\mathbf{l})$$

where $r(\mathbf{l})$ is the reward in sampling mode and $r(\mathbf{l}')$ is the reward in argmax mode. If the reward in sampling mode is higher, the resulting positive difference $r(\mathbf{l}) - r(\mathbf{l}')$ reinforces the chosen action, encouraging the tracker to adopt policies that improve sequence-level performance.

This reward function thus guides the tracker in learning an optimal sequence-level tracking policy that aligns closely with the evaluation metric used in testing [6].

B. Psuedocodes and Ideas

[6] uses SCST with REINFORCE to train their networks. Which we have tried to recreate in our experiment initially.

Algorithm 3 SLT as given by Kim et. al [6]

Require: A tracker parameterized by $theta$, and dataset $gamma$
  while not converged **do**
    Sample a video $v = (v_0, \ldots, v_T)$ and ground-truth $g = (g_0, \ldots, g_T)$ from Γ
    Initialize the tracker with $\{v_0, g_0\}$
    $l_0 \leftarrow g_0$ ▷ Initial target location for the sampling tracker
    $l'_0 \leftarrow g_0$ ▷ Initial target location for the argmax tracker
    for $t = 1$ to T **do**
      $l_t \sim p_\theta(l; v_t, l_{t-1})$ ▷ Sample action for the sampling tracker
      $l'_t \leftarrow \arg\max_l p_\theta(l; v_t, l'_{t-1})$ ▷ Action for the argmax tracker
    end for
    Compute rewards r and r' for $\{l_1, \ldots, l_T\}$ and $\{l'_1, \ldots, l'_T\}$
    Compute loss $L \leftarrow -(r - r')\sum_{t=1}^{T} \log p_\theta(l_t; v_t, l_{t-1})$
    Update parameters $\theta \leftarrow \theta - \alpha\nabla_\theta L$
  end while

What we want to do here is try and improve the learning by making the sampled action part more deliberate. We achieve this by replacing the softmax sampling with a Critic Network. We try to convert the above approach to an A2C Modified SLT. We can look at the Algorithm 4 to under stand our proposed algorithm.

VI. Experiments

A. Hardware and Resources

We plan to use the codebase from the following paper: Towards Sequence-Level Training for Visual Tracking [6]. This paper contains a codebase which provides a very good framework to add new trackers to existing SLT implementation by the author. We used the following publicly available datasets for training and learning: GOT-10k [10]. This project will use the following software: CUDA 11.3, Python 3.9, PyTorch 1.10.1, Torchvision 0.11.2. We used 2 Nvidia A6000 GPUs to run the algorithm.

Algorithm 4 Our proposed format with A2C SLT

Require: A tracker parameterized by θ, a critic network parameterized ϕ, and dataset Γ
while not converged **do**
Sample a video $v = (v_0, \ldots, v_T)$ and ground-truth $g = (g_0, \ldots, g_T)$ from Γ
Initialize the tracker with $\{v_0, g_0\}$
$l_0 \leftarrow g_0$ ▷ Initial target location for the sampling tracker
Initialize cumulative reward $R \leftarrow 0$
for $t = 1$ to T **do**
Sample action $l_t \sim p_\theta(l; v_t, l_{t-1})$ ▷ Sample action from the policy
Compute immediate reward r_t for l_t and g_t
$R \leftarrow \gamma R + r_t$ ▷ Accumulate rewards
end for
Compute the advantage for each step: $A_{T+1} = R - V_\phi(v_{T+1})$ ▷ Using the critic V_ϕ
Compute the policy loss: $L_{\text{policy}} \leftarrow -\sum_{t=1}^{T+1} A_t \log p_\theta(l_t; v_t, l_{t-1})$
Compute the value loss: $L_{\text{value}} \leftarrow \frac{1}{T+1}\sum_{t=1}^{T+1}(R - V_\phi(v_t))^2$
Update the actor parameters θ: $\theta \leftarrow \theta - \alpha \nabla_\theta L_{\text{policy}}$
Update the critic parameters ϕ: $\phi \leftarrow \phi - \beta \nabla_\phi L_{\text{value}}$
end while

B. Benchmark

We have come across various papers using different algorithms. We planned on recreating the results achieved by [6]. The stretch goal would be to improve the tracker provided by [6] by trying the mentioned RL algorithms using our A2C SLT Transt algorithm. This performance was tested on the evaluation server provided by the GOT-10K itself (http://got-10k.aitestunion.com/).

C. Results

We ran both the SCST based SLT algorithm and A2C SLT Algorithm for 10 epochs and took roughly 7 hours per tracker. The accuracy for A2C SLT is lower than expected but can be improved further given additional training time. The measures given in Table I are percentage accuracy.

TABLE I
PERFORMANCE RESULTS: BASE VS SLT (PAPER) VS SLT (OURS) VS A2C SLT ON GOT-10K.

Model	Data Used	AO	SR50	SR75
TransT (Baseline)** [6]	Multiple	66.2	75.5	58.5
SLT + TransT ([6])	Multiple	72.0	81.6	68.3
SLT + TransT (Ours)	GOT-10K	72.5	82.2	68.8
A2C SLT + TransT	**GOT-10K**	**70.5**	**79.9**	**66.3**

* Multiple Dataset represents a superset of the following publicly available datasets: TrackingNet, GOT-10k, LaSOT, ImageNet-VID, DAVIS, YouTube-VOS, MS-COCO, SBD, LVIS, ECSSD, MSRA10k, and HKU-IS.
** Data that we did not verify, but taken directly from [6].

We can see that SLT improves the performance of the supervised learning model TransT. We believe the performance of A2C SLT should improve given more time to train. [6] suggests 120 epochs even for SCST. A2C SLT is already beating the baseline by a considerable margin, but improving it further could also allow us to beat the normal SLT Tracker.

The reason we believe A2C can improve further with more training is that our Critic Network is currently lagging behind our Actor-network. We need to bootstrap the critic to converge before A2C SLT starts giving expected improvements over the baseline and normal SLT.

VII. CONCLUSION

Using deep reinforcement learning for object tracking has shown to improve the performance of trackers at a sequence level. Frame level training isn't able to account for the issues from real world environments such as occlusion, motion blur and other dynamic changes. The sequence level tracking solution proposed by [6] clearly improves the performance for video-based trackers, by treating it as sequential video tracking rather than just individual frames. Our experiment proves that it works well for the TransT tracker. Our implementation of A2C-based sequence-level tracking did show some improvements in the loss at the beginning but needs more time to train as it uses a critical network and is attempting to bootstrap itself into becoming the ideal critic network.

VIII. FUTURE WORKS

As stated before, the A2C SLT Tracker could be run for more epochs. There are several ways to take the work done here forward. One special case would be [19] where they have implemented something similar and treated the tracker as a Sequence Generation rather than a Tracking problem. With sequence generation the tracker uses smaller incremental changes over time rather than a global change from a sequence of videos.

Many other RL strategies could be used to improve the trackers, specifically trackers that use a policy gradient such as Proximal Policy Optimization [25], which uses a clipped objective function that avoids the unstable policy updates from a monte-carlo approach like REINFORCE. Many other example networks could be picked from [21] to further explore the sequential perfomance on object tracking.

REFERENCES

[1] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[2] Zedu Chen, Bineng Zhong, Guorong Li, Shengping Zhang, and Rongrong Ji. Siamese box adaptive network for visual tracking. *CoRR*, abs/2003.06761, 2020.

[3] Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep reinforcement learning for visual object tracking in videos, 2017.

[4] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-decision networks for visual tracking with deep reinforcement learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1349–1358, 2017.

[5] Boyu Chen, Dong Wang, Peixia Li, Shuang Wang, and Huchuan Lu. Real-time 'actor-critic' tracking. page 328–345, Berlin, Heidelberg, 2018. Springer-Verlag.

[6] Minji Kim, Seungkwan Lee, Jungseul Ok, Bohyung Han, and Minsu Cho. Towards sequence-level training for visual tracking, 2022.

[7] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

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

[9] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild, 10 2018.

[10] Lianghua Huang, Xin Zhao, and Kaiqi Huang. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1562–1577, 2021.

[11] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[12] Matteo Dunnhofer, Niki Martinel, Gian Luca Foresti, and Christian Micheloni. Visual tracking by means of deep reinforcement learning and an expert demonstrator. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, page 2290–2299. IEEE, October 2019.

[13] Liangliang Ren, Xin Yuan, Jiwen Lu, Ming Yang, and Jie Zhou. Deep reinforcement learning with iterative shift for visual tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[14] Dawei Zhang, Zhonglong Zheng, Riheng Jia, and Minglu Li. Visual tracking via hierarchical deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(4):3315–3323, May 2021.

[15] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015.

[16] 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, 2019.

[17] Jin-Hyeok Park, Khurshedjon Farkhodov, Suk-Hwan Lee, and Ki-Ryong Kwon. Deep reinforcement learning-based dqn agent algorithm for visual object tracking in a virtual environmental simulation. *Applied Sciences*, 12(7), 2022.

[18] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Detection and tracking meet drones challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[19] Xin Chen, Houwen Peng, Dong Wang, Huchuan Lu, and Han Hu. Seqtrack: Sequence to sequence learning for visual object tracking. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14572–14581, 2023.

[20] Wenhan Luo, Peng Sun, Fangwei Zhong, Wei Liu, Tong Zhang, and Yizhou Wang. End-to-end active object tracking and its real-world deployment via reinforcement learning, 2019.

[21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[22] Derya G¨ozen and Sedat Ozer. Visual object tracking in drone images with deep reinforcement learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10082–10089, 2021.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[24] Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning, 2017.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.