Drone Navigation and Target Interception Using Deep Reinforcement Learning: A Cascade Reward Approach

Ali A. Darwish and Arie Nakhmani, *Senior Member, IEEE*

***Abstract*—This article proposes an architecture for drone navigation and target interception, utilizing a self-supervised, model-free deep reinforcement learning approach. Unlike the traditional methods relying on complex controllers, our approach uses deep reinforcement learning with cascade rewards, enabling a single drone to navigate obstacles and intercept targets using only a forward-facing depth–RGB camera. This research has significant implications for robotics, as it demonstrates how complex tasks can be tackled using deep reinforcement learning. Our work encompasses three key contributions. First, we tackle the challenge of partial observability when employing nonlinear function approximators for learning stochastic policies. Second, we optimize the task of maximizing the overall expected reward. Finally, we develop a software library for training drones to track and intercept targets. Through our experiments, we demonstrated that our approach, incorporating cascade reward, outperforms state-of-the-art deep *Q*-network algorithms in terms of learning policies. By leveraging our methodology, drones can successfully navigate complex indoor and outdoor environments and effectively intercept targets based on visual cues.**

***Index Terms*—Cascading reward function, drone tracking, dueling double deep *Q*-network (DQN), interception, reinforcement learning (RL).**

I. Introduction

DRONE navigation and interception pose considerable challenges due to their inherent complexity. Traditional navigation methods rely on various components, including a drone model, environmental representation, independent control systems, object recognition, occupancy maps, and intricate controllers [1], [2], [3], [4], [5], [6]. However, fine-tuning these elements in real-world scenarios is arduous and often necessitates labeled data and custom engineering from developers. The development of drone tracking and interception systems holds significant practical implications, such as enhancing airport security, enabling effective airspace patrolling, and safeguarding public and personal privacy.

Our proposed approach utilizes reinforcement learning (RL) to optimize all subsystems, aiming to maximize the overall system performance in these applications by representing the problem as a framework for an artificial solver known as an agent. The drone agent within RL learns representations of the environment, including target drones, obstacles, gates, and tracks, and translates this understanding into commands for continuous control. This adaptability allows our approach to be implemented in various environments, both indoors and outdoors.

RL, as a subfield of artificial intelligence (AI), holds significant promise for robotics as it emulates the learning and evolutionary processes observed in intelligent life. Unlike approaches relying on predetermined and preprogrammed instructions, RL enables machine intelligence to increase by not being constrained by such limitations. The RL agent interacts with the environment to acquire meaningful representations and corresponding actions, lacking prior knowledge of the environment. It learns solely through observations, rewards, and punishments.

Actions, observations, and rewards, collectively referred to as experiences, are stored in a table, providing the agent with a basis for learning and generalizing a value function. These experiences are then optimized through cumulative rewards. This learning paradigm, which utilizes rewards and punishments through unsupervised trial-and-error interactions, characterizes RL.

In our case, the drone has six actions: up, down, right turn, left turn, forward, and backward, enabling it to navigate freely within its space. Observations capture the drone's senses, such as RGB-D images, distance information, and geolocation data. Rewards represent the unit of reward or punishment received by the agent after each action in the environment.

RL allows us to witness the evolution of AI in practical applications. For instance, OpenAI's robotic arm evolved using RL to recognize colors and letters and manipulate objects [1]. Such capabilities were once considered science fiction but have become a reality in less than a decade. Although RL principles and algorithms have existed for several decades, the field is still in its early stages but steadily expanding. Only

Manuscript received 14 July 2023; revised 23 October 2023; accepted 14 November 2023. Date of publication 20 November 2023; date of current version 12 December 2023. This work was supported by NVIDIA Corporation. *(Corresponding author: Arie Nakhmani.)*

Ali A. Darwish was with the Department of Electrical and Computer Engineering, University of Alabama at Birmingham, Birmingham, AL 35294 USA, and also with the University of Alabama in Huntsville, Huntsville, AL 35899 USA. He is now with Athenium Analytics, Washington, DC 20001 USA. (e-mail: alex.darwish@athenium.com).

Arie Nakhmani is with the Department of Electrical and Computer Engineering, University of Alabama at Birmingham, Birmingham, AL 35294 USA (e-mail: anry@uab.edu).

Digital Object Identifier 10.1109/JISPIN.2023.3334690

recently RL applications have gained public attention due to advancements in computational power, particularly with the availability of advanced GPUs originally driven by demand from gamers and cryptocurrency miners. These GPUs have made training deep reinforcement learning (DRL) models computationally feasible. In DRL, neural networks approximate and solve the value function, as raw tables and queues become unscalable for optimizers, while neural networks effectively learn from large datasets. DRL shares similarities with RL in terms of how the agent learns through exploration but differs in how the value function is calculated.

Humans, animals, and certain insects demonstrate a remarkable ability to generalize from past experiences by deriving multidimensional representations of the real world from high-dimensional sensory inputs [7]. This high level of metalearning can be achieved through a combination of RL and hierarchical processing, allowing acquired knowledge to be efficiently applied to new situations [8]. Replicating this natural learning approach using sequential programming is complex and challenging. Learning from past experiences and generalizing poses difficulties due to the plasticity-stability dilemma, a concept widely recognized in the field [9]. In the realm of robotics' applications, the traditional approach of developing and tuning models does not scale well in real-life scenarios [10]. However, recent advancements in RL and deep learning (DL) have made it possible to mimic biological learning processes [11]. For instance, Mnih et al. [12] successfully trained an agent to play games by using raw pixels as input and training a deep Q-network (DQN), demonstrating the capability of RL and DL to generalize across a collection of video games. RL has also been applied to solve more complex tasks, such as teaching a robot how to walk [13].

Our approach draws inspiration from the OpenAI Gym game suite [14], [15], specifically Super Mario, Sonic, and Space Invaders. By drawing insights from these games, we aim to develop a drone tracking and interception system that can exhibit similar learning and adaptation capabilities. Our goal is to leverage RL algorithms and apply them to real-world scenarios, allowing the drone agent to learn representations of the environment, make informed decisions, and optimize its performance in tracking and intercepting targets.

The present work introduces a novel approach for drone navigation and target interception in complex environments. A mathematical framework is derived, and an architecture is defined to realize these techniques. Specifically, a model-free architecture is developed for drone interception in a 3-D environment where the states are partially observable. Further details regarding partial observability are provided in the partially observable RL section. The approach combines DRL with a cascading reward function to train a drone to intercept an airborne target. In addition, the drone is trained through RL on depth and RGB images, using a dueling dual DQN as a nonlinear function approximator and leveraging a long short-term memory (LSTM) neural network to address the suboptimality problem [16].

In this article, the architecture is also tested with a drone controller that is not explicitly programmed to fly and detect other unmanned aerial vehicles (drones), as the model of the drone and its environment are not described in the simulation. The drone agent learns to use high-level actions from a discretized action space consisting of six actions. It also learns representations of the environment, such as obstacles, gates, and tracks, and translates them into commands for continuous control [17]. Compared with the explicit decomposition of similar tasks in the traditional programming, which requires example-based supervised learning, path planning, control, and occupancy grids, the proposed approach optimizes all subsystems to maximize the overall system performance. To address the challenges of sparse and delayed rewards, limited state representations, and complex tasks, the objective function is optimized using reward shaping. The cascading reward function, a tiered reward approach, is employed to achieve better learning performance compared with a fixed reward function. We provide three key contributions:

1) tackling partial observability challenges with nonlinear function approximators for learning stochastic policies;
2) optimizing the task of maximizing the overall expected reward;
3) developing a software library for training drones to track and intercept targets.

The library called TrackGym is made publicly available on GitHub [18].

The rest of this article is organized as follows. Section II discusses current approaches in object tracking, while Section III presents the proposed approach by defining the problem, presenting the system architecture, and detailing the agent–environment unsupervised learning. The cascading reward function and its mathematical framework are explained, and the TrackGym is introduced as a testing environment for the approach. In Section IV, the results are discussed. Finally, Section V concludes this article.

II. Previous Work

A. Drone Tracking

Conventional vision-based object tracking in drones consists of the supervised extraction of features from the target using either computer vision techniques [19], [20], [21], [22], [23], [24], [25] or deep learning [6], [17], [22], [26], [27], [28]. These features are passed to the system for creating an occupancy grid or rich representation of the environment. In the computer vision approach, two representation schemes determine if the target object is described as generative or discriminative [29]. Regardless of the approach, the tracked object is first located using the computer vision system, and the target's coordinates are passed to the control system responsible for moving the drone.

Recent systems that use vision techniques were demonstrated by a variety of researchers. Qu et al. [19] presented an algorithm for intercepting moving targets with wheeled mobile robots in dynamic environments. The algorithm predicts target positions, generates an interception trajectory with path and speed separation, and utilizes Hybrid A$*$ search and gradient descent for path planning. It introduces an spatio-temporal (ST) graph for speed planning, represented by piecewise Bézier curves. Fu et al.

[2] used global feature matching and the iterative Lucas–Kanade optical flow for local feature tracking of moving objects. A local geometric filter handled outlier feature correspondences based on forward–backward pairwise dissimilarity measures. As the output, the local geometric features were represented as a binary classification problem for outlier feature detection. The system successfully tested the real-time tracking of moving objects at a speed of 20 frames/s in the Unreal engine simulator. Another onboard computer vision system enabled for RGB and a depth camera was proposed by Mueller et al. [3] for the long-term tracking of moving objects. Support vector machines (SVM) and structured output tracking with kernels were used for the object identification subsystem. The authors achieved good results in tracking fast-moving and occluded objects.

Zhu et al. [4] proposed a human-in-the-loop visual tracking method for drones with human corrections integrated into the system to compensate for drifts in tracking caused by occlusion and lighting conditions. In this approach, the tracking-learning-detection (TLD) algorithm was used for real-time object tracking. TLD is a widely used algorithm in drone-tracking tasks [30], [31], [32], [33], [34] and achieves very good results in short-term object tracking [30]. However, TLD includes several limitations when implemented in real-world applications, particularly with long-term tracking [34]. Optical flow misalignments limit the effectiveness of TLD due to light conditions and orientation shifts, so the authors proposed a human-in-the-loop approach to compensate. They also used the Kalman filter for movement prediction and proposed an adaptive strategy based on clustering algorithms for correcting bounding box shifts. Pestana et al. [5] proposed using TLD for real-time visual servoing and tracking. This system performed well in real-world testing, even with deformed shapes and occluded objects. Our research also tested this approach and attained similar results to those reported, but our system failed to track fast-moving and distant objects. Also, the performance severely degraded during a long-term tracking course.

Chakrabarty et al. [6] proposed and simulated a search and track system for small drones and used the histogram of oriented gradients' features and linear SVM for object classification. They reported good tracking performance using only a few scenarios. The authors successfully presented a complete system built using the robot operating system formulated with a finite state machine. De Smedt et al. [35] proposed a system for pedestrian tracking using aggregated channel features as well as a particle filter. This technique enabled the authors to test a light system by limiting the area of interest with geometrical constraints, such as the height of a pedestrian. While the authors reported improved results of the tracker compared with a conventional particle filter, the system was limited by requiring preprogrammed object properties, such as height, pose, and drone-tracking angle. Opromolla et al. [30] presented a template matching and morphological filtering approach for visual drone tracking to solve the limitations of trackers susceptible to visual issues, such as illumination variation, target scale, and background variation. Another advantage of this approach is the enabling of detection at varying distances. The system was intended to supplement the global navigation satellite system in challenging environments and areas with low signal coverage. The results showed that it is possible to reduce the probability of false alarms and missed detections using the template matching approach with pixel-level normalized cross correlation.

Although the idea of using DRL is not new in the field of robotics, not until recently could we implement and test the technique because of computing power and implementable algorithms. The testing of these algorithms was limited because an agent–environment interaction is required. To the best of our knowledge, DRL has not been used for drone object tracking and interception. However, there have been attempts to use RL to track objects visually in [29] and [36]. Few research attempts exist using DRL for drone navigation, obstacle avoidance, and landing. Fan et al. [37] proposed a multiagent interception approach for scenarios where the network topology can change due to communication restrictions or attacks. To overcome this challenge, the authors introduced a multiagent-level fusion actor–critic approach with a direction-assisted actor, dimensional pyramid fusion critic, and an experience adviser function. Additionally, they proposed a reward factor to balance individual and shared rewards.

Polvara et al. [38] proposed an autonomous landing system for quadrotor drones via DRL, where drones learn how to land on a designated pad from images acquired by a down-looking camera. The authors used this type of camera in a classification subsystem for landmark detection, while another subsystem handled vertical descent. Each subsystem had a specialized DQN, and they were all connected with a double DQN to reduce overestimation problems during interaction with the environment. The authors tested their approach in several simulated environments and reported performance that surpassed the tracking algorithm and human pilots. They also noted that the trained network could generalize to real environments despite being trained in the Gazebo simulation.

Deng et al. [22] tested a missile terminal guidance approach using the deep deterministic policy gradient (DDPG) algorithm for intercepting maneuvering targets with infrared decoys. The method involves creating a virtual target, improving the DDPG algorithm for better training efficiency, and designing a heuristic reward function. Monte Carlo tests confirm the effectiveness and robustness of this approach, showcasing its performance compared with the traditional methods.

Tran et al. [39] proposed a dataset aggregation approach for obstacle avoidance and control for small drones in a clustered environment. The drone achieved good performance and traveled into areas not accessible to a human pilot. However, the authors' approach was based on transfer learning and not entirely DRL-based. In transfer learning, a human operator (or a trained machine) navigates the drone and stores the learned features and control data. These stored weight files initialize the network for the DRL agent to provide a head start over pure learning from exploration. The authors extracted features from the images using a set of 7×7 sliding windows and calculated the eigenvalues for each neighborhood pixel value. They used the Radon transform for computing line integrals to project 2-D images onto 1-D lines. While this approach performed well as reported by the authors, it is unclear if it generalizes well to other environments and tasks or if it is practical to use

human operators to train and apply corrections for each task. We propose a different solution for tracking and interception as we consider localization and control simultaneously, and our goal is to propose, develop, and test a general RL approach for solving multiple problems in drone navigation.

B. Partially Observable RL

Learning how to play ATARI games using DRL has seen considerable success [14]. The 2-D environments are ideal for training agents because states of the game within the ATARI setting are fully observable by agents that have access to the entire environment images. Many challenging problems that require decision making were once considered virtually impossible to solve, until recent successful approaches with DRL, such as in the game of Go [40] and Poker [41]. Scaling DRL methods from ATARI-style environments to 3-D is difficult. On the one hand, an agent sees a first-person view of the environment, making the returned states partially observable. On the other hand, rewards in a higher dimensional environment can be sparse. A partially observable state from the perspective of an agent is when the world can be partially seen, with many of the events and goals being hidden. Lample and Chaplot [42] addressed these issues by augmenting an RL algorithm to infer high-level information from the environment. They trained a game-based agent in the video game Doom to fight enemies and collect rewards. The authors trained two networks, one for navigation and a second for fighting, and then cotrained a DQN with game features. This approach is interesting and potentially applicable to our drone-tracking problem. However, an agent's movement in the Doom environment is bound to the x and y axes, so this is limited compared with a natural environment in which drones can operate [43].

Kersandt [44] proposed using a DQN-based algorithm for training an agent on navigation tasks with depth 2-D images within a photorealistic 3-D environment. The author used OpenAI, Airsim simulator, and Unreal for creating an environment and high-level interactions with the agent. Our approach for the environment is inspired by Kersandt, but we suggest that solving real-life tasks in a partially observable environment is limited because they trained a drone on how to navigate only in the x and y dimensions. While this method simplifies the learning by eliminating the third z-dimension, it cannot work with drone interception tasks because the tracking drone must move within a 3-D space freely.

III. Proposed Approach

In this section, we focus on defining the problem of tracking and outlining our technical solution, which leverages RL techniques. In the subsequent sections, we will delve deeper into the technical details of our proposed approach, including the mathematical framework, the system architecture, and the agent–environment interactions.

A. Architecture

Fig. 1 illustrates the proposed architecture for our drone tracking and interception system, adapted from the AirSim/Unreal system [45]. It consists of two main modules: AirSim and Unreal Engine for simulation, and OpenAI Gym for training and control.

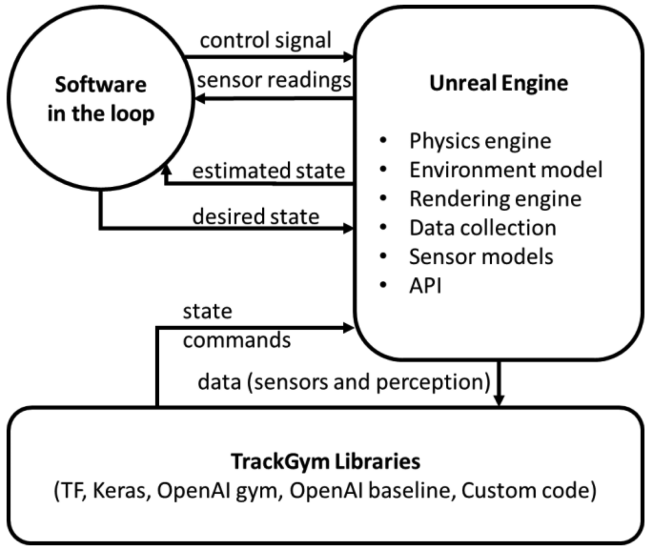

Fig. 1. Proposed system architecture.

In the proposed architecture, we utilize the AirSim libraries to interact with the Unreal API, and this is enabling us to create a realistic simulation environment. The software-in-the-loop feature within AirSim allows us to simulate the flight controller, incorporating sensor data, such as accelerometer and gyroscope readings, to estimate actuator control signals. The Unreal engine serves as the game engine, providing photorealism features and open-source game environments. It generates realistic conditions within the simulated environment, including lighting effects and wind dynamics. We can import predefined vehicle models and customize the environment as needed. The authors of the AirSim library provide a comparison of two simulation scenarios that were tested in the real world with the Px4 controller [45]. This demonstration showcases the accuracy of the simulation versus the real controllers. The sensor data obtained from these real-world tests closely match the data generated in the simulated environment.

Within the architecture, the flight controller of AirSim translates the states of the drone, which consist of RGB-D images and position information, into signals that drive the actuators of the simulated drone. These states, along with the target drone's motion and interaction definition, are passed to the TrackGym libraries. TrackGym incorporates various libraries used to train the neural network with RGB-D and position inputs and outputs the commands for controlling the agent within the environment. It integrates with OpenAI Gym through its API, allowing for training and control of the drone agent using RL.

The high-level actions determined within the OpenAI Gym framework are translated into control signals representing movements such as up, down, yaw right, yaw left, forward, and backward. These control signals are then processed by the flight controller to estimate the actual roll, pitch, and yaw of the drone using data from the gyroscope and accelerometer. The

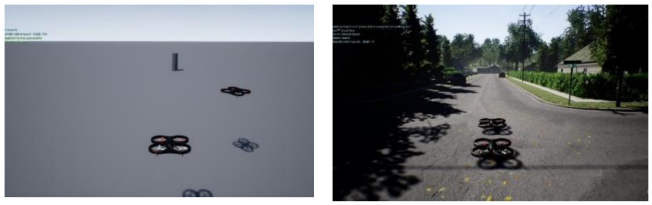

Fig. 2. Unreal environment that was used for training. The left image (a) shows the simple environment used to train the network to identify the target, and the right image (b) illustrates the complex, partially observable environment.

flight controller generates motor signals that drive the propellers, resulting in thrust, torques, and other forces calculated by the physics engine in Unreal. The simulator in Unreal computes the kinematic state of the drone based on the inputs received from the flight controller, considering factors, such as drag, friction, gravity, and motor behavior. Overall, this architecture provides a comprehensive framework for simulating and training the drone agent for tracking and interception tasks. It integrates the necessary components, including simulation, control, and training libraries, to create an effective learning environment for the agent.

B. Environment

Fig. 2 illustrates the simulated environment within the Unreal engine that was used for training the drone-tracking agent. In addition to the landscape, the environment has a tracking drone with a forward-facing depth camera installed and a target drone. For training purposes, two different environments were utilized. The first environment is a simple one, designed to teach the agent the representations of the target using a convolutional neural network (CNN) function approximator. This environment allows the agent to learn the visual features and characteristics of the target drone.

The second environment is more complex than the first one, which is used for transfer learning. Here, the knowledge gained from the first environment in Fig. 2(a) is transferred to learn how to avoid obstacles effectively in the environment, as shown in Fig. 2(b). Transfer learning is a technique where knowledge acquired from one task is applied to another related task, improving learning efficiency and performance. Moreover, the transfer learning approach allows the agent to be trained and tested in both indoor and outdoor environments. The skills and representations learned in the first environment can be effectively transferred and applied in different settings, enabling the agent to generalize its knowledge and adapt to various scenarios.

The training approach and details of the agent's learning process in both environments are explained in Section IV of this article. This section provides a comprehensive overview of the training methodologies, including the use of CNN for target representations and transfer learning for obstacle avoidance, enabling the agent to acquire the necessary skills for successful drone tracking and interception.

We used the OpenAI gym environment [46] for defining the agent drone in the AirSim simulation and providing an abstraction layer for the environment. We also defined the interaction between the agent, the environment, and the reward function with the five methods of step, reset, render, action, and reward. At each time step, the step method returns an observation, a reward, the status of the episode, and logging information.

To enable agent–environment interactions through OpenAI's abstraction, we developed two classes in the TrackGym called myTrackGymClient and TrackGym. The first class implements the methods related to AirGym, e.g., obtain an image, takeoff, and move. In the latter class, the OpenAI gym methods are implemented with calls to the myTrackGymClient method. The published GitHub repository further describes all classes and methods available in TrackGym [18].

C. Reward Shaping

Crafting a reward function that effectively establishes a goal-reward relationship presents a significant challenge. In RL, agents receive rewards or penalties for their actions without explicit instructions, necessitating their ability to determine whether their actions were rewarded. This structure, known as the credit assignment system [47], enables agents to learn how to reach the target through actions that may be delayed or not related to the goal. Rewarding an agent in an environment with numerous variables poses another obstacle. For instance, when a target is occluded, the agent may need to execute maneuvers that do not align with optimal Q-values to reach the goal. Similarly, achieving the goal may require the agent to maintain specific conditions, such as being on the same z-plane and within a certain angle of attack. Introducing multiple objectives further complicates the reward function. Ideally, the reward function should be continuous, differentiable, and capable of accommodating the various scenarios agents may encounter in dynamic, real-world environments. Since RL agents primarily accomplish tasks for which they are rewarded, designing the reward function becomes crucial in ensuring the agent's desired behavior.

To guide the learning process effectively, three types of termination conditions are defined for each training episode: time-limit, positive, and negative terminations. These conditions determine when to reset the environment, terminating undesired exploration and focusing on critical states for the given task. Time-limit terminations occur when the agent becomes stuck or when the simulation fails. Positive terminations arise when the agent accomplishes the task and receives the maximum possible reward, prompting a restart of the environment to initiate learning of a new policy. Negative terminations occur when the agent fails, such as colliding with an obstacle or reaching a geographic limit. Under these conditions, the agent receives the maximum negative reward, and the training process is restarted. Negative terminations play a crucial role in collision avoidance during the training of a drone to track and intercept the target. They facilitate the learning of Q-values associated with actions that lead to collisions. Negative terminations are also utilized to restrict undesired exploration far from the target by employing geographic coordinates. When the agent reaches

these predefined coordinates, it incurs a penalty, prompting a restart of the training process.

Positive rewards may yield counter-intuitive results, as rewarding the drone for proximity to the target might lead it to fly in circles to accumulate maximum rewards rather than intercepting the target to obtain a smaller, one-time reward. Increasing the positive termination reward could potentially address this issue, but it may also diminish the drone's inclination to reach the target promptly. Negative rewards prove useful in incentivizing the agent to complete the training episode quickly by reaching the goal target. In this scenario, the agent aims to be rewarded for reaching the goal while minimizing penalties associated with negative distances from the target and excessive time steps taken to achieve the goal.

1) Cascade Reward: We design the reward function to incorporate temporal and spatial considerations, aiming to guide the agent's behavior systematically during the process of approaching and intercepting the target. The objective is to establish an ordered approach strategy that the agent can follow.

To successfully intercept the target, the agent is incentivized to prioritize quick and efficient movement on the same z-plane as the target or is already oriented toward it. This emphasis on speed encourages the agent to expedite its approach toward the target when favorable conditions are present. However, it is important to acknowledge that the agent may encounter various obstacles along its path to the target. Therefore, the reward function also incorporates the ability of the agent to adapt its heading and altitude and to navigate around these obstacles. By allowing the agent to modify its trajectory, the reward function enables the agent to effectively address obstacles and continue progressing toward the target.

By structuring the reward function in this manner, we create a framework that motivates the agent to follow a specific order in approaching and intercepting the target. It considers both the temporal aspect of prompt action and the spatial aspect of maneuvering to overcome obstacles, thus providing a comprehensive approach to guide the agent's behavior in the pursuit of successful target interception. Equation (1) represents the reward function for this stage as

$$r_t = -2 - \left(\frac{d_t^{\varphi}}{\max(d_t, 0.1)}\right) \tag{1}$$

where d_t is the distance to target at time step t, r_t is the reward after time step t, and φ is a parameter used in shaping the reward function. We empirically find that assigning $\varphi = 0.4$ shapes the curve toward the goal of directing the agent to minimize the negative penalties and reach the target quickly. When the agent approaches the target from a distant position with a full view, orienting the head toward the target becomes important. We want the agent to maintain the target within its line of sight from this point until interception, so we introduce an additional heading reward. The threshold for entering this stage varies depending on the target. In the Unreal simulation, we determined a threshold at a distance of 10 NED. The NED is the coordinate system used by the AirSim and represents $+X$ as North, $+Y$ as East, and $+Z$ as Down. A higher reward value is provided for smaller heading angles h between the agent and the target according to

$$r_t = 1 - (d_t^{\varphi}) + \left(\frac{1}{\max(h_t, 0.1)^{\varphi}}\right). \tag{2}$$

When the tracking drone is within a predefined distance from the target, for example, $d = 6$ NED, and before intercepting the target, we want the z-coordinate of the agent to be positioned on the same z-plane as the target, so these coordinates are incorporated into the following equation:

$$r_t = 1 - (d_t^{\varphi}) + \left(\frac{1}{\max(h_t, 0.1)^{\varphi}}\right) + \left(\frac{1}{\max(z_t, 0.1)^{\varphi}}\right). \tag{3}$$

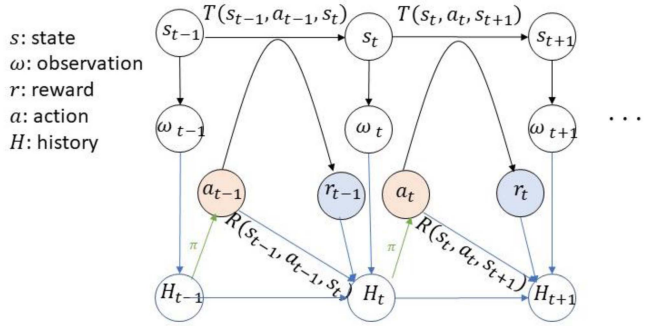

Fig. 3. POMDP state transition. The black lines depict the dynamics, and the blue lines depict the information in the history H_t used by the agent (adapted from [49]).

D. Learning

Within the environment where we train the drone agent, the decision dynamics are also determined by a Markov decision process (MDP), but the agent does not directly observe the states. The agent only receives observations that appear unrelated to future states. The partially observable Markov decision process (POMDP) can be used to model these types of decision processes that include uncertainty [48].

The POMDP is a seven-tuple discrete-time stochastic process given as $(\mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \Omega, \mathrm{O}, \gamma)$. Similar to the MDP, $\mathcal{S}$ is the state space $\{1, \ldots, N_{\mathcal{S}}\}$, $\mathcal{A}$ is the action space $\{1, \ldots, N_{\mathcal{A}}\}$, T: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability between states, R: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function where $\mathcal{R}$ is a continuous reward in $R_{\max} \in \mathbb{R}^+$, γ is the discount factor used with Bellman's equation in the range $[0, 1)$, and O: $\mathcal{S} \times \Omega \to [0, 1]$ is the conditional observation probability set, where Ω is the set of observations $\{1, \ldots, N_{\Omega}\}$. Fig. 3 illustrates the POMDP where at a state s_t the agent receives an observation ω from the set of observations Ω with a probability $O(s_t, s_\omega)$ for each time step t. Here, $\omega_t \in \Omega$ is equal to s_t when the environment is Markovian.

The probability of transitioning to time step $t + 1$ for the state $s_{t+1} \in \mathcal{S}$ is given by the state transition function $\mathrm{T}(\mathrm{s_t}, \mathrm{a_t}, \mathrm{s_{t+1}})$, and the reward it receives is $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$. The agent's actions in a state do not directly link to previous or future states in a partially observable environment. Instead, a history of the previous observations is used to better estimate the state

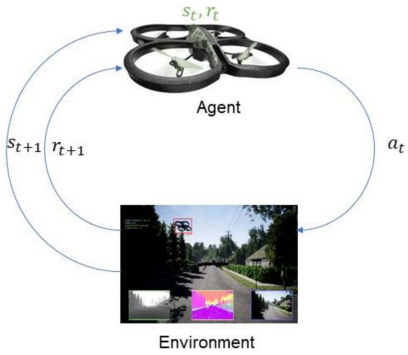

Fig. 4. Environment interaction framework in the RL model.

dynamics of T [49]. The history of observations up to time t is denoted by $H_t \in \mathcal{H}_t = \Omega \times (\mathcal{A} \times \mathcal{R} \times \Omega)^t$ and all observed history is denoted by $\mathcal{H} = \bigcup_{t=0}^{\infty} \mathcal{H}_t$. To avoid suboptimality in the partially observable environment, other methods achieve a stochastic policy, such as policy gradient [49]. For example, Mnih et al. [12] used CNNs on the POMDP domain with a large set of states, actions, and rewards to solve the problem of playing ATARI with RL.

The agent receives the state s_t from the environment at each step t and performs an action a_t from an action space $\mathcal{A}$. For each a_t, there is a reward r_t representing the reward function $R(s_t, a_t)$. Fig. 4 illustrates the general principle of agent–environment interaction. The state transition model at time step $t+1$ becomes $T(s_{t+1}|s_t, a_t)$. During training, we maximize the agent's reward using Bellman's equation for a discounted cumulative reward of

$$R = \sum_{k=0}^{\infty} (\gamma^k (r_{t+1+k})) \tag{4}$$

where γ is the discount factor within [0, 1]. The closer the factor is to 0, the more it prioritizes instant rewards, and the closer to 1, the more it prioritizes future discounts. This deep Q-learning approach [50] is known as an off-policy, where the optimal policy value is learned independently of the agent's actions. The policy π_t is a direct map from each state s to the best action a corresponding to the state $\pi(s) = a$ such that

$$\pi_t(a|s) = P(A_t = a | S_t = s). \tag{5}$$

As described in Section III-E, we use CNN for the Q-function value approximation. The $Q(s, a)$ can be decomposed into the sum of $V(s)$, representing the value of being in a state, and the advantage $A(s, a)$ of taking action in this state. Hence

$$Q(s, a) = A(s, a) + V(s). \tag{6}$$

We use a dueling DQN [50] that separates the estimator of the state value $V(s)$ and the advantage $A(s, a)$, and uses aggregation to obtain an estimated $Q(s, a)$. Because dueling DQN produces more accurate Q-value estimates compared with DQN and double DQN [50], it is useful for training a drone to intercept other drones by learning which states are valuable without having to learn the effect of the selected actions. This approach is achievable by subtracting the average advantage of all actions possible in the state as calculated in the following equation:

$$Q\left(s, a; \theta^{(1)}, \theta^{(2)}, \theta^{(3)}\right) = V\left(s; \theta^{(1)}, \theta^{(3)}\right) + \left(A\left(s, a; \theta^{(1)}, \theta^{(2)}\right) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A\left(s, a'; \theta^{(1)}, \theta^{(2)}\right)\right) \tag{7}$$

where parametrized Q-value function $Q(s, a; \theta)$ is introduced with the parameter θ that defines the Q-value. θ is a parameter used to parametrize the network. In the second part of (7), we could subtract $\max_{\bar{a} \in |\mathcal{A}|} A(s, a'; \theta^{(1)}, \theta^{(2)})$ from the advantage instead of taking the mean, but the max value decreases the stability of the optimization [50]. Using the mean, the advantages only need to change as fast as the mean changes instead of compensating for changes that optimize the action's advantage [50].

The experiences are collected in a preliminary phase and stored in a d-queue memory [12], and then are randomly sampled regardless of the relevance of the experience to the task. The intrinsic structure of the tracking problem requires that we change the sampling distribution and prioritize experiences that are closely related to successful drone interception actions. Therefore, we propose using the prioritized experience replay (PER) [51] to sample experiences that are important for the training but occur less frequently. In PER, the criterion for which the importance of each transition is measured is the amount of DRL the agent learns from a transition. The magnitude of the temporal difference, i.e., a one-step look-ahead error δ, estimates the goodness of the transition $p_i = |\delta_i| + \varepsilon$, where ε is a small constant to guarantee that no experience has zero probability. The probability of prioritizing states leads to tracking and hitting a target

$$p_{(i)} = \frac{p_i^{\xi}}{\sum_{\rho} p_{\rho}^{\xi}} \tag{8}$$

where $p_i > 0$ is the priority of the transition i, and ξ determines the prioritization for introducing randomness into the experience selection such that $\xi = 0$ is uniform sampling, and $\xi = 1$ selects the highest priority experience. The term $\sum_{\rho} p_{\rho}^{\xi}$ includes all priority values in the replay buffer.

Changing the experience replay distribution with the PER introduces a bias that affects the convergence of the estimation, so the importance of sampling balances this effect [51] with

$$w_i = \left(\frac{1}{\rho} \cdot \frac{1}{p(i)}\right)^{\sigma} \tag{9}$$

where ρ is the replay buffer size, w_i is the importance sampling for transition i, and σ is a hyperparameter to control how much importance sampling is used and annealed through the training within (0, 1].

E. Function Approximator

We propose a CNN with LSTM for the Q predictions and values for V, as shown in Fig. 5. The network input is a sequence of four 90×256 grayscale depth and RGB images normalized to map the range of pixel values of 0–255 and processed with a

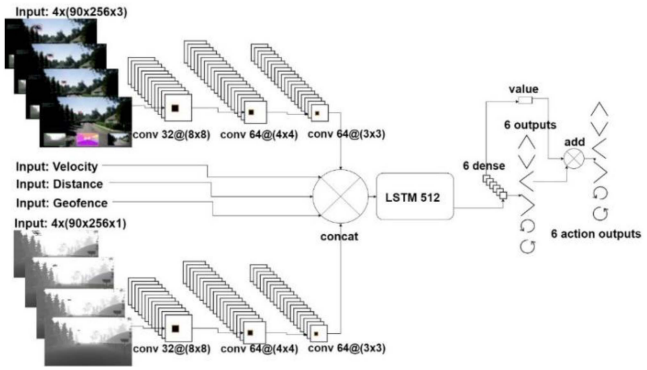

Fig. 5. CNN with LSTM and multiple inputs used for learning the *Q*-values.

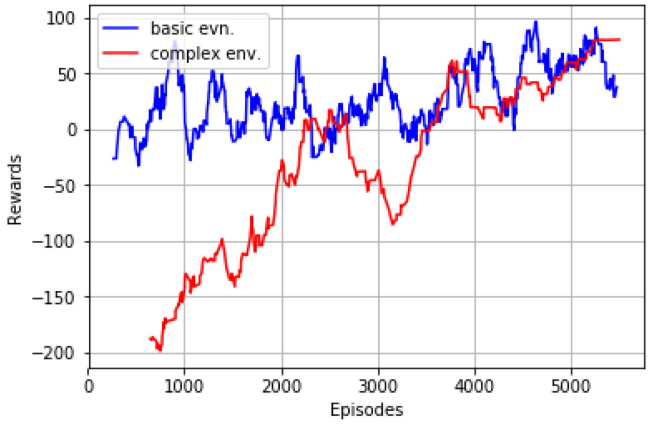

Fig. 6. Comparison between the earned rewards in the basic and advanced environments.

scaling function to reduce the resolution and returns the action probabilities with the action value V. We provided four frames as an input to each branch to capture the motion of the target drone and eventually teach the RL agent to predict the future state of the target with the help of the LSTM. By adding an LSTM to the architecture, we can improve the agent's ability to track moving objects. The RGB frames give the network the ability to extract texture information from the states. Building around our previous experiences with image classification and segmentation, we believe this network can better identify the background from the target using the encoded colors in the pixels.

The velocity of the agent and its distance to the target were incorporated to help learn its relative motion to the target. A vector of the geofence coordinates was also added to keep the agent within a region of interest and to shorten the exploration and training times. The geofence box range values included -100 in the z-direction and -200 to 200 in the x and y directions. When the agent crosses the geofence, it is punished with -100 points, and the simulation is restarted. By passing the geofence coordinates into the network, we create an association between the region of exploration and the states. The action space consists of six discrete commands of {(up), (down), (yaw right $30^\circ/s$), (yaw left $30^\circ/s$), (forward), (backward)}.

IV. Training and Results

Training RL agents in realistic physics' simulations poses a significant challenge due to the time-consuming nature of the process. Unlike the OpenAI game suite, which allows for accelerated training, the physics engine employed in Unreal cannot be accelerated. To mitigate this issue and ensure the algorithms' correctness, we initially tested the RL agents using the ATARI breakout agent, a well-established benchmark, prior to incorporating them into the realistic environments. Then, we developed two distinct environments. The first environment, as shown in Fig. 2(a), aimed to train the agent in recognizing the target. It consisted of a simplified setting comprising an obstacle and a target drone. This environment provided a controlled and straightforward setup for the agent to learn the task of target recognition. Subsequently, we leveraged the concept of transfer learning to enhance the agent's capabilities by training it to both avoid obstacles and locate the target simultaneously. The second environment, as depicted in Fig. 2(b), aimed to replicate real-life scenarios with photorealistic objects. This environment provided a more complex and challenging setting, resembling real-world conditions that the agent would encounter.

By incorporating these two environments into our training pipeline, we were able to progressively train the RL agent to first recognize the target and then transfer that knowledge to navigate through complex environments while avoiding obstacles. This approach allowed us to bridge the gap between simple and realistic scenarios, facilitating the agent's learning process and enabling it to acquire the necessary skills for successful target interception in more challenging and realistic environments.

For each environment, the training consisted of episodes in which the cumulative reward was calculated until the episode was terminated either by a total reward of $+100$ or -300. We had 5500 episodic steps that took approximately two weeks to train for each algorithm and environment on an Nvidia TitanXp GPU. The target drone was programmed to move to different locations as the episodes progressed. After the agent was trained in the simple environment in Fig. 2(a), the saved weights were transferred to the agent before the training was restarted in Fig. 2(b).

Fig. 6 shows the smoothed cumulative reward achieved per episode for t time steps for the basic environment in Fig. 2(a) and the complex environment in Fig. 2(b). The plotted line represents an increase in the accumulated rewards with the progress through the training episodes. Notable dips exist in the plot caused by moving the target within the environment. Overall, the trend line increased and broke the 80-point reward between episodes 5200 and 5500, which indicates that the performance of the interception could be improved if more data were provided to the network. At the beginning of training, we positioned the target closer to the agent, and, as training progressed, the target was moved farther away. The maneuvers required to intercept the target evolved and became more purpose driven instead of exploratory, as represented by the agent circling the target to adjust its controls before approaching for interception. Fig. 6

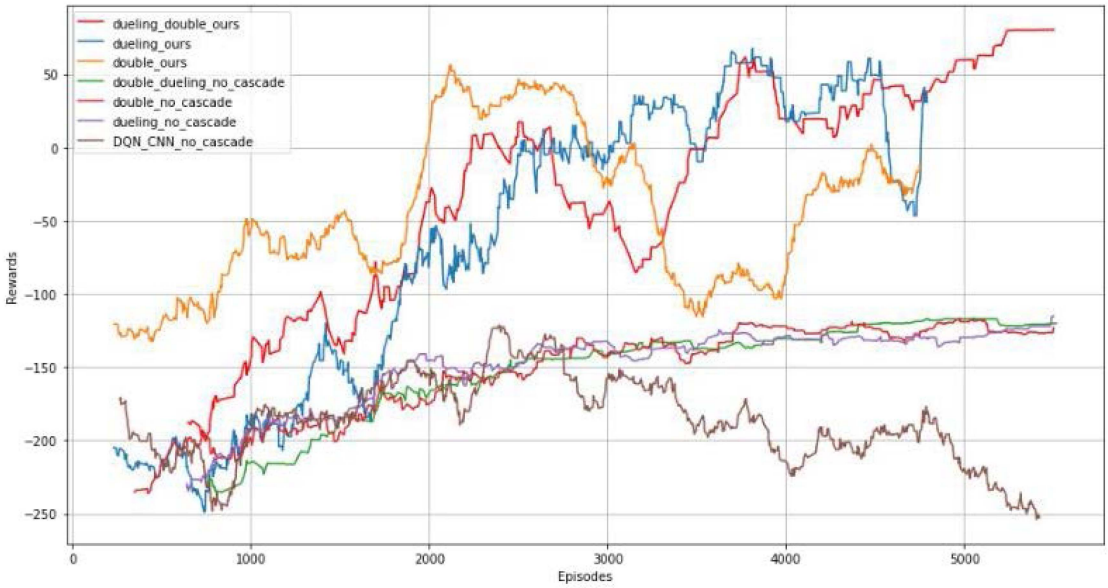

Fig. 7. Agent learning behavior using different algorithms in a complex environment.

also depicts a comparison between the learning behavior of the agent in the complex and basic environments. The agent's performance was notably stable but experienced too many crashes during early training, which is an expected behavior because the agent has not encountered these types of obstacles in the new environment, even though the learning was transferred directly from the basic environment.

The agent appears to have rebuilt its neural network graph, which is attributed to the new features existing in the complex environment and many unrecognized obstacles. The agent initially crashed into many barriers after being penalized by the simulator for flying away from the target. Eventually, the agent learned to avoid these obstacles to find its way to the target, as evident in the episodic rewards, as seen in Fig. 6. We tested three DQN algorithms of double dueling, double, and dueling, all in a complex environment. The first test was performed with the CNN-LSTM network with input consisting of the RGB images, velocity, geofence, and distance to target as well as the depth images but without using the cascade reward.

The rewards applied during this training are shown in Fig. 7, depicted as the double_dueling_no_cascade, dueling_no_cascade, and dueling_no_cascade curves. In the second test, we also used the CNN-LSTM network with a similar input as the first test and used the cascade reward function. The rewards applied during this training are shown in Fig. 7 as the dueling_double_ours, dueling_ours, and double_ours curves. In the third test, we trained the agent using the DQN network using the CNN, as shown in Fig. 8, with depth images only to represent the states as input and without using the cascade reward. These rewards are depicted in Fig. 7 as the DQN_CNN_no_cascade curve.

During the third training, the agent failed to achieve its goal of intercepting the target using the DQN algorithm and the CNN network without LSTM. This result indicates that the agent

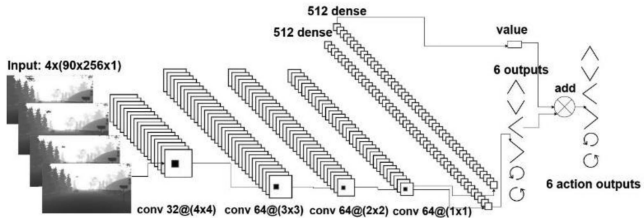

Fig. 8. CNN used for learning the *Q*-values from the depth images.

requires more than depth representations of the environment to recognize the target in the complex environment. The low cumulative reward values achieved also suggest that obstacle avoidance requires a richer representation of the environment. The performance of the agent improved when the CNN-LSTM was incorporated because the states provided richer representations. The CNN-LSTM with the cascading reward function outperformed the other methods with the agent improving its skills to navigate the environment and intercept the target successfully. All variations of the DQN during this training performed well with the double dueling DQN algorithm, performing slightly better than the double or dueling DQN.

Table I presents the testing results of our approach in comparison with other models saved at different episode counts, namely 3500, 4000, and 4500 episodes. The term "succ %" denotes the success rate, while "Interception steps" signifies the average number of steps required for a successful interception, excluding episodes where interception attempts failed from the calculation. As depicted in Table I, during the training phase, the Double_Ours, Double_Dueling, and Double_DQN_CNN models exhibit limited competence in the interception task. In contrast, our dueling double and dueling models achieve the success rates of 15% and 11% after 3500 episodes, with 72 and

TABLE I
COMPARISONS OF DIFFERENT METHODS ON THE SUCCESS RATE, INTERCEPTED STEPS, AND REWARD

Episode	Method	*Succ%*	Interc. Steps	Reward
3500	Dueling_Double_ Ours	15	72	0
	Dueling_Ours	11	89	−9
	Double_Ours	0	-	−150
	Double_Dueling	0	-	−162
	Double	0	-	−148
	DQN_CNN	0	-	−191
4000	Dueling_Double_ Ours	43	51	23
	Dueling_Ours	41	56	22
	Double_Ours	7	87	−91
	Double_Dueling	3	96	−124
	Double	0	-	−129
	DQN_CNN	1	127	−129
4500	Dueling_Double_ Ours	63	32	44
	Dueling_Ours	68	37	53
	Double_Ours	12	81	−7
	Double_Dueling	4	112	−120
	Double	3	93	−131
	DQN_CNN	3	87	−131

89 steps required for interception, respectively. The performance of all algorithms improves as training progresses. However, our proposed methods consistently maintain a higher success rate in interception and require fewer steps to achieve it, as demonstrated in the table.

V. CONCLUSION

We introduced a novel approach for addressing the challenges of object tracking and interception in partially observable indoor and outdoor environments. Our approach leverages the power of DRL in conjunction with cascading reward functions. Furthermore, we have developed and published a platform dedicated to RL-based tracking research, providing a means to train agents in photorealistic simulations with the aim of transferring the learning to real-world scenarios. Our findings demonstrate the successful application of RL-based control for drone navigation and the importance of feature extraction techniques, environmental modeling, and reward-shaping strategies. Achieving robust and stable control necessitates a delicate tradeoff between performance and stability. DRL, by virtue of its employment of nonlinear and adaptable function approximators, holds promising prospects for practical control implementation in real-life applications. However, we encountered challenges in these endeavors, primarily due to the potential occlusion of the small target drone by the background and the high dimensionality of the observations. Moreover, the training time required for effective tracking, particularly in real physics' simulations, added another layer of complexity. Therefore, enhancing the learning speed will be a pivotal factor in achieving success with the RL approach in these advanced tasks.

Our future research directions involve expanding the capabilities of the agent in the context of drone navigation to tackle more complex tasks, such as identifying, tracking, and intercepting fast-moving targets. Recently developed algorithms for multitarget tracking and trajectory prediction based on deep learning [23], [24] could also improve the interception if blended into our framework. In addition, we aim to extend the current single-agent environment to a multiagent learning setup, as this has the potential to expedite the learning process through enhanced interaction and cooperation between multiple agents.

ACKNOWLEDGMENT

The authors would like to thank Nvidia Corporation for their generous donation of the TitanXp GPU that we used for training the agent.

REFERENCES

[1] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014.
[2] C. Fu, R. Duan, D. Kircali, and E. Kayacan, "Onboard robust visual tracking for UAVs using a reliable global-local object model," *Sensors*, vol. 16, no. 9, 2016, Art. no. 1406, doi: 10.3390/s16091406.
[3] M. Mueller, G. Sharma, N. Smith, and B. Ghanem, "Persistent aerial tracking system for UAVs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1562–1569, doi: 10.1109/iros.2016.7759253.
[4] Y. Zhu, C. Wang, Y. Niu, and L. Wu, "hTLD: A human-in-the-loop target detection and tracking method for UAV," in *Proc. IEEE/CSAA Guid., Navig. Control Conf.*, 2018, pp. 1–6.
[5] J. Pestana, J. L. Sanchez-Lopez, P. Campoy, and S. Saripalli, "Vision based GPS-denied object tracking and following for unmanned aerial vehicles," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot.*, 2013, pp. 1–6, doi: 10.1109/ssrr.2013.6719359.
[6] A. Chakrabarty, R. A. Morris, X. Bouyssounouse, and R. Hunt, "An integrated system for autonomous search and track with a small unmanned aerial vehicle," *AIAA Inf. Syst.-AIAA Infotech Aerosp.*, 2017, doi: 10.2514/6.2017-0671.
[7] J.-F. Gariépy et al., "Social learning in humans and other animals," *Front. Neurosci.*, vol. 8, 2014, Art. no. 58, doi: 10.3389/fnins.2014.00058.
[8] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, vol. 2, pp. 994–1000, doi: 10.1109/CVPR.2005.254.
[9] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects," *Front. Psychol.*, vol. 4, 2013, Art. no. 504.
[10] J. Kober, J. Andrew Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.
[11] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, vol. 2, pp. 2094–2100.
[12] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.
[13] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," 2019, *arXiv:1812.11103*.

[14] V. Mnih et al., "Playing ATARI with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[15] M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.
[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.
[17] T. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[18] 2023. [Online]. Available: https://github.com/AloshkaD/DroneTracking
[19] C. Qu, J. He, J. Li, C. Fang, and Y. Mo, "Moving target interception considering dynamic environment," in *Proc. IEEE Amer. Control Conf.*, 2022, pp. 1194–1199, doi: 10.23919/ACC53348.2022.9867177.
[20] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, 2006, Art. no. 13–es, doi: 10.1145/1177352.1177355.
[21] S. Lankton, J. Malcolm, A. Nakhmani, and A. Tannenbaum, "Tracking through changes in scale," in *Proc. IEEE 15th Int. Conf. Image Process.*, 2008, pp. 241–244, doi: 10.1109/ICIP.2008.4711736.
[22] T. Deng, H. Huang, Y. Fang, J. Yan, and H. Cheng, "Reinforcement learning-based missile terminal guidance of maneuvering targets with decoys," *Chin. J. Aeronaut.*, to be published, doi: 10.1016/j.cja.2023.05.028.
[23] A. Nakhmani and A. Tannenbaum, "Particle filtering using multiple cross-correlations for tracking occluded objects in cluttered scenes," in *Proc. IEEE 47th Conf. Decis. Control*, 2008, pp. 652–657, doi: 10.1109/CDC.2008.4738656.
[24] A. Nakhmani and A. Tannenbaum, "Scale-invariant visual tracking by particle filtering," *SPIE Image Signal Process. Remote Sens. XIV*, vol. 7109, pp. 183–190, 2008.
[25] A. Nakhmani and A. Tannenbaum, "Particle filtering with region-based matching for tracking of partially occluded and scaled targets," *SIAM J. Imag. Sci.*, vol. 4, no. 1, pp. 220–242, 2011.
[26] P. G. Kandhare, A. Nakhmani, and N. M. Sirakov, "Trajectory type prediction and multi-target tracking," in *Proc. IEEE SoutheastCon*, Huntsville, AL, USA, 2019, pp. 1–6, doi: 10.1109/SoutheastCon42311.2019.9020381.
[27] P. G. Kandhare, A. Nakhmani, and N. M. Sirakov, "Deep learning for location prediction on noisy trajectories," *Pattern Anal. Appl.*, vol. 26, pp. 107–122, 2023, doi: 10.1007/s10044-022-01095-y.
[28] A. Darwish and A. Nakhmani, "Internal covariate shift reduction in encoder-decoder convolutional neural networks," in *Proc. SouthEast Conf.*, 2017, pp. 179–182, doi: 10.1145/3077286.3077320.
[29] S. Yun, J. Choi, Y. Yoo, K. Yun, and J. Y. Choi, "Action-driven visual object tracking with deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2239–2252, Jun. 2018, doi: 10.1109/tnnls.2018.2801826.
[30] R. Opromolla, G. Fasano, and D. Accardo, "A vision-based approach to UAV detection and tracking in cooperative applications," *Sensors*, vol. 18, no. 10, 2018, Art. no. 3391, doi: 10.3390/s18103391.
[31] G. Wenhua, J. Nianping, and L. Zhenxing, "TLD target tracking algorithm based on particle filter," *Electron. Technol.*, vol. 51, pp. 188–195, 2015.
[32] W. Hailong, W. Guangyu, and L. Jianxun, "An improved tracking-learning-detection method," in *Proc. IEEE 34th Chin. Control Conf.*, 2015, pp. 3858–3863, doi: 10.1109/chicc.2015.7260234.
[33] B. Nemade and V. A. Bharadi, "Adaptive automatic tracking, learning and detection of any real time object in the video stream," in *Proc. IEEE 5th Int. Conf. - Confluence Next Gener. Inf. Technol. Summit (Confluence)*, 2014, pp. 569–575, doi: 10.1109/confluence.2014.6949039.
[34] Z. Xin, Q. Qiumeng, Y. Yongqiang, and W. Congqing, "Improved TLD visual target tracking algorithm," *J. Image Graph.*, vol. 18, pp. 1115–1123, 2013.
[35] F. De Smedt, D. Hulens, and T. Goedeme, "On-board real-time tracking of pedestrians on a UAV," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2015, pp. 1–8, doi: 10.1109/cvprw.2015.7301359.
[36] D. Zhang, H. Maei, X. Wang, and Y. Wang, "Deep reinforcement learning for visual object tracking in videos," 2017, *arXiv:1701.08936*.
[37] D. Fan, H. Shen, and L. Dong, "Switching-aware multi-agent deep reinforcement learning for target interception," *Appl. Intell.*, vol. 53, pp. 7876–7891, 2023, doi: 10.1007/s10489-022-03821-9.
[38] R. Polvara et al., "Autonomous quadrotor landing using deep reinforcement learning," 2017, *arXiv:1709.03339*.
[39] L. D. Tran et al., "Reinforcement learning with autonomous small unmanned aerial vehicles in cluttered environments—After all these years among humans, you still haven't learned to smile," in *Proc. 15th AIAA Aviation Technol., Integr., Oper. Conf.*, 2015, doi: 10.2514/6.2015-2899.
[40] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[41] M. Moravčik et al., "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.
[42] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2140–2146.
[43] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "ViZDoom: A doom-based AI research platform for visual reinforcement learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8, doi: 10.1109/cig.2016.7860433.
[44] K. Kersandt, "Deep reinforcement learning as control method for autonomous UAVs," M.S. thesis, Aerospace Sci. Technol., Universitat Politècnica de Catalunya, Barcelona, Spain, 2018.
[45] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*. Cham, Switzerland: Springer, 2017, pp. 621–635, doi: 10.1007/978-3-319-67361-5_40.
[46] G. Brockman et al., "OpenAI gym," 2016, *arXiv:1606.01540*.
[47] B. J. Lansdell, P. R. Prakash, and K. P. Kording, "Learning to solve the credit assignment problem," 2019, *arXiv:1906.00889*.
[48] R. Bellman, "A Markovian decision process," *J. Math. Mech.*, vol. 6, pp. 679–684, 1957.
[49] F.-L. Vincent, H. Peter, I. Riashat, G. B. Marc, and P. Joelle, "An introduction to deep reinforcement learning," *Foundations Trends Mach. Learn.*, vol. 11, no. ¾, pp. 219–354, 2018, doi: 10.1561/2200000071.
[50] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[51] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016, *arXiv:1511.05952*.

Ali A. Darwish received the B.S. and M.S. degrees in software engineering from the National Polytechnic University of Armenia, Yerevan, Armenia, in 2006, the M.S. degree in computer engineering from the American University of Sharjah, Sharjah, UAE, in 2013, the M.S. degree in electrical engineering from the University of Alabama in Huntsville, Huntsville, AL, USA, in 2015, and the Ph.D. degree in computer engineering from the University of Alabama at Birmingham, Birmingham, AL, USA, and the University of Alabama in Huntsville, Huntsville, AL, USA, in 2020.

He currently serves as a Senior AI Engineer with Athenium Analytics, Dover, NH, USA. In his accomplished career, he has secured several patents, all related to the field of AI and ML.

Arie Nakhmani (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Technion – Israel Institute of Technology, Haifa, Israel, in 2004, 2007, and 2011, respectively.

He completed the postdoctoral training with Georgia Tech, Atlanta, GA, USA, and Boston University, Boston, MA, USA. He is currently an Associate Professor of electrical and computer engineering with the University of Alabama at Birmingham, Birmingham, AL, USA. He has authored or coauthored more than 100 peer-reviewed publications and a textbook *Modern Control: State-Space Analysis and Design Methods* (McGraw Hill, 2020). His research interests include visual tracking, biomedical signal analysis, and machine learning.

Dr. Nakhmani is an Associate Scientist with the Comprehensive Cancer Center.