applied sciences

MDPI

Article

Deep Reinforcement Learning-Based DQN Agent Algorithm for Visual Object Tracking in a Virtual Environmental Simulation

Jin-Hyeok Park [1], Khurshedjon Farkhodov [2], Suk-Hwan Lee [3] and Ki-Ryong Kwon [1,*]

[1] Department of IT Convergence and Application Engineering, Pukyong National University, Busan 48513, Korea; mons88@pukyong.ac.kr
[2] Department of Artificial Intelligence Convergence, Pukyong National University, Busan 48513, Korea; farkhodovxf@pukyong.ac.kr
[3] Department of Computer Engineering, Dong-A University, Busan 49315, Korea; skylee@dau.ac.kr
* Correspondence: krkwon@pknu.ac.kr; Tel.: +82-51-629-6257

Abstract: The complexity of object tracking models among hardware applications has become a more in-demand task to accomplish with multifunctional algorithm skills in various indeterminable environment tracking conditions. Experimenting with the virtual realistic simulator brings new dependencies and requirements, which may cause problems while experimenting with runtime processing. The goal of this paper is to present an object tracking framework that differs from the most advanced tracking models by experimenting with virtual environment simulation (Aerial Informatics and Robotics Simulation—AirSim, City Environ) using one of the Deep Reinforcement Learning Models named as Deep Q-Learning algorithms. Our proposed network examines the environment using a deep reinforcement learning model to regulate activities in the virtual simulation environment and utilizes sequential pictures from the realistic VCE (Virtual City Environ) model as inputs. Subsequently, the deep reinforcement network model was pretrained using multiple sequential training image sets and fine-tuned for adaptability during runtime tracking. The experimental results were outstanding in terms of speed and accuracy. Moreover, we were unable to identify any results that could be compared to the state-of-the-art methods that use deep network-based trackers in runtime simulation platforms, since this testing experiment was conducted on the two public datasets VisDrone2019 and OTB-100, and achieved better performance among compared conventional methods.

Keywords: object tracking; object detection; deep learning; deep reinforcement learning; deep Q-network; DQN; AirSim; virtual simulation environment; virtual drone

Citation: Park, J.-H.; Farkhodov, K.; Lee, S.-H.; Kwon, K.-R. Deep Reinforcement Learning-Based DQN Agent Algorithm for Visual Object Tracking in a Virtual Environmental Simulation. *Appl. Sci.* **2022**, *12*, 3220. https://doi.org/10.3390/app12073220

Academic Editor: José Sánchez Moreno

Received: 17 January 2022
Accepted: 15 March 2022
Published: 22 March 2022

1. Introduction

Visual object tracking is a classic computer vision problem that entails detecting an object in a scene and distinguishing it from other objects in each frame. In sequential frames, it might be static or dynamic. To our knowledge, there are a slew of fundamental challenges to visual object tracking, including occlusion, motion blur, background clutter, ambient illumination fluctuations, etc. To address these issues, the most common tracking approaches [1–4] monitor specific object classes utilizing a variety of feature learning algorithms. However, several other tracking techniques offer great efficiency and competitive outcomes when compared to the state-of-the-art object tracking models. Nevertheless, they still have constraints that need to be addressed in order to achieve high accuracy and quicker tracking performance under difficult environmental circumstances.

There are certain tracking filters [5,6] and object-detection-based models [7,8] that may be competitive alternatives to classic methods, but they still have drawbacks when compared to deep network-based tracking strategies.

Despite the relative success of conventional tracking approaches [9–11], deep convolutional neural network (Deep CNN)-based visual object tracking models [12–14] have gained

popularity in recent years. The popularity and advantages of CNN-based trackers may be explained in two ways: firstly, by their tracking robustness, and secondly, by the highly efficient feature representations located inside their detection units. In the majority of target tracking situations, pretrained CNN classifiers were employed [11,15,16] for finding objects and classifying them, or cropping and regressing methods [9,10,17]. However, the disparity between the CNN-based feature representation for classification and the tracking algorithm has an impact on the output of the final tracking results. Furthermore, the pretrained CNN classifier fails to function well in the challenging environment of the tracking process, where the captured spatial features are not explored thoroughly during training.

Nonetheless, tracking by detection method has been proven to be effective in numerous difficult tracking circumstances [11,18,19] where CNN was used in training and superior tracking results were finally obtained compared to standard target trackers. Unfortunately, in the event of a visually crowded scene with numerous occluded frames and a small distance of correlation, targeted items may be missed. The basic strategy and goal of CNN-based models relies on a target classification approach. In addition, an object classification model will be used to address a backdrop cluttering problem. Consequently, there is a necessity to investigate the ability of deep learning models to automatically learn effective features in a virtual environment that uses drone agents with spatial and temporal constraints. In particular, it should be considered to make a long–short term feature learning strategy and classify objects into identical target classes with the proposed end-to-end model.

The motivation for this study was to develop an object tracking algorithm capable of learning and tracking the target using deep reinforcement learning and an artificial intelligence network model in a wide variety of skills and abilities that can compete with other models in complicated tasks. In this study, we created a novel tracker that was based on a sequential recurrent neural network [20,21] prediction and tracking architecture. The deep reinforcement learning-based Q-network agent tracker was integrated with the high-dimensional cross-platform virtual simulator for drones called AirSim.

For testing reasons, we built our unique tracking model using the AirSim [22] (Aerial Informatics and Robotics Simulation) simulator platform. The platform enables the algorithm to be evaluated in a realistic virtual environment that includes elements such as pedestrians, automobiles, trees, street signs, buildings, and weather conditions. Figure 1 shows the virtual CityEnviron model scenario with a virtual AirSim drone from the Microsoft AirSim v1.2 version. The inspiration and idea for our proposed model came from deep reinforcement learning for human-level control [23], which uses a specific architecture in conjunction with a deep convolutional network [24] with hierarchical layers of tiled convolutional filters integrated alongside artificial neural networks in order to learn concepts such as object categories directly from raw sensory data.

Our proposed deep reinforcement learning and tracking technique is thought of as a sequential feature learning/prediction and decision-making technique for drone agents to monitor actions' rewards through environment sequence and to acquire a tracking output of the recommended methodology. Traditionally, we estimated the adaptive action-value function using a deep sequential neural network [25] such as

$$Q^{\pi}(s,a) \doteq \mathbb{E}_{\pi}[G_t|s,a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s,a\right] \quad (1)$$

where the equation estimates the expected total returns, also known as the sum of rewards G_t beginning from a fixed location state s and performing an action a in accordance with some policy π.

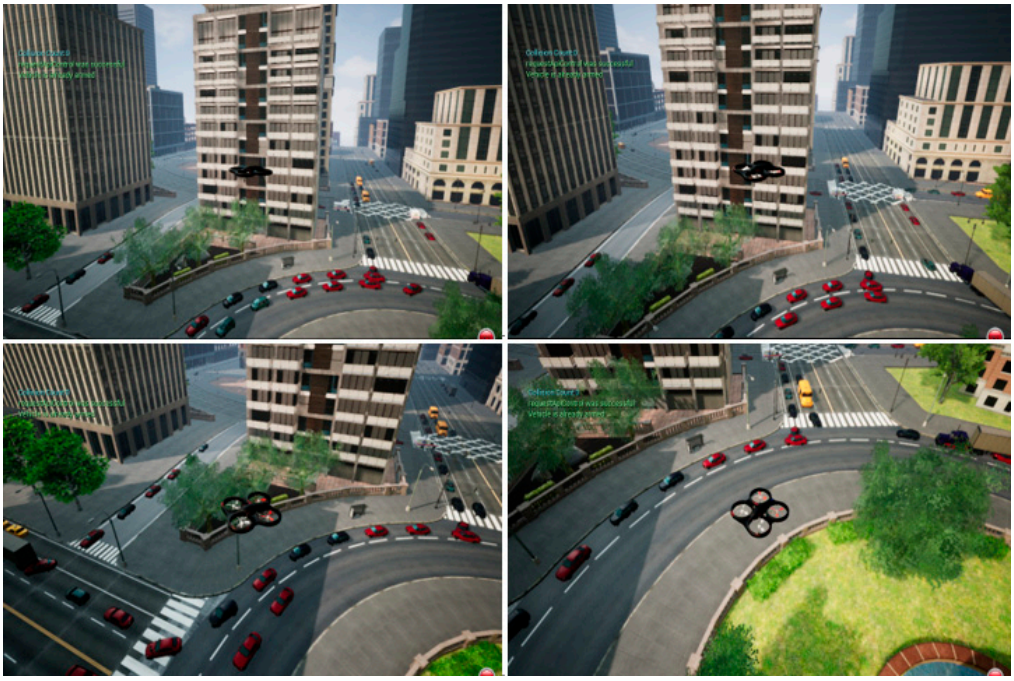

Figure 1. Microsoft AirSim v1.2 is a realistic virtual CityEnviron environment scenario with an AirSim drone simulator from various angles on a city map.

Reinforcement Learning (RL) algorithms are mostly based on the above-mentioned Equation (1), which estimates the best value function from sufficient experience to obtain the excellent value estimation. Our proposal utilizes a Q-learning algorithm that uses an action-value function with particular parameters to learn and obtain the best action-value by repeating the learning process. Basically, the reinforcement learning method is known to be unstable as a nonlinear function approximator in a neural network to represent the action-value (known as Q) function given above. The $Q^{\pi}(s, a)$ equation function above estimates the expected total returns by taking some policy π. $\pi(a|s)$ is the policy that maps from state observation s to action a. Typically, stochastic policies are used; however, deterministic policies can also be specified. Most of the useful RL algorithms heavily rely on the above value function and will be obtained from sufficient experience, the optimal policy π^* that can be found in every state, and taking the greedy action that leads to the state with the highest value estimations. The process will be applied continuously for every state environment as an iterative update.

The rest of this article is organized as follows. Section 2 will give an overview of deep network-based object tracking descriptions and related work. In Section 3, we describe the deep reinforcement learning-based DQN agent drone tracking algorithm integrated with the Virtual CityEnviron (VCE) simulation platform. Experimental simulation results for verifying our algorithm are presented in Section 4 and our conclusion follows in Section 5.

2. Related Work

2.1. Visual Object Tracking

Visual object tracking has gained more attention in computer vision in the last decade than ever before. There have been numerous successful studies on various tracking benchmarks [18,19,26]. Classification-based trackers have also been proposed, which may be referred to as tracking-by-detection or tracking-by-classification [27–31]. These techniques primarily focus on separating targets from the scene by collecting target locations and detecting them using trained classifier models. To be more specific, a tracker captures foreground patches close to the target position and background patches from a distance, which are then trained as a foreground–background classifier to score the current or next frame's target's location in order to recognize it.

Robust MIL-based feature learning [12] and tracking-learning-detection adopted for unsupervised learning [14] techniques were presented to improve the resilience of tracking models in noisy environments. In general, the classification model is trained offline using manually labeled pictures before being utilized for online or real-time tracking operations. Numerous neural network-based trackers utilize these concepts [11,16,32,33] throughout the development of their approach, and they provide effective outcomes when compared to classic trackers [12,13,34] and achieve state-of-the-art results [11,26]. The concept of using correlation filters to resolve the inadequate representation of convolutional and hand-crafted features is retained [5,15,35]. However, due to the fact of learning through a limited number of scale-wise filters and a relatively small number of feature extraction video frames, those methods are not fully functional, resulting in the loss of critical long-term feature representation and temporal information between two or several consecutive frames.

2.1.1. Regression-Based Trackers

In recent years, some researchers have developed a deep regression network-based tracking approach [10,36,37] that uses bounding box regression to identify objects instead of classification models. This technique improves the chances of solving the tracking issue by training the model on trainable datasets using a loss function such as mean-squared or mean-absolute error. The datasets usually consist of the input pictures and the bounding boxes of the frame's object classes. However, if the object moves very quickly across the consecutive frames or if occlusion occurs, this approach may fail or produce poor results during the tracking process. Furthermore, to fully cover the consecutive frames' characteristics and their information, the aforementioned approaches require a more efficient searching algorithm for learning the dataset or environment by utilizing sliding-window or candidate-sampling strategies.

2.1.2. Recurrent Neural Network-Based Tracking

Furthermore, recurrent neural network-based research [36–38] has advocated employing recurrent layers in order to solve the visual object tracking issues. They employ an RNN-based structure combined with an attention mechanism. RNN-based tracking models have not been proven competitive yet on contemporary benchmarks; nevertheless, this technique can obtain higher results by using sequential layers to anticipate objects using the sliding method.

There is also a work by Ning et al. [39] that combines spatially supervised recurrent convolutional neural network integration with the YOLO network [40] architecture for directly detecting object classes. The recurrent neural network will directly regress the YOLO detection output on each frame to retrieve the targeted objects class.

2.2. Deep Reinforcement Learning

Reinforcement learning (RL) is a training approach in which a machine learning model makes judgments in a series of actions while managing the process. It investigates how an agent may learn to make decisions and attain goals in a complex and unpredictable environment. Essentially, this method learns the optimal policy for deciding which sequential acts to perform by maximizing future benefits [41]. Recent popular works [42–45] propose the combination of RL models with deep neural networks in order to improve decision making by representing RL techniques as a policy or value function where the model learns the process interactively from feedback rewards, to improve expected rewards in long-term sequential processes by learning best policy. Several techniques have proposed recovering deep features by learning particular policy tasks [46], which have been evaluated by applying them to Atari games [42] and other [44] approaches, which have been successfully addressed by using a semi-supervised learning methodology. In addition, many models for object localization [47], prediction, and target tracking [48] have been suggested. By integrating CNN, RNN, and RL algorithms, Zhang et al. [49] proposed a network design that was particularly well-suited for addressing the tracking issue. They

used RNN as a top-level CNN feature extraction, focusing on both spatial and temporal restrictions. In addition, the framework was trained in offline mode utilizing an end-to-end reinforcement method.

Deep Q-Networks (DQN) and the gradient policy method are the most well-known deep RL algorithms [23,42]. The Deep Q-Network is an alternate model of the Q-learning algorithm that learns each step of the action values in a given state. It is a model-free method that uses stochastic transitions and incentives to solve problems without requiring any modifications. Alternatively, various DQN algorithm-based designs have been developed, such as Double DQN [50] and Dueling DQN [51], which are better versions of the learning and tracking process in terms of stability.

Another research method is the policy gradient methodology, which learns policy directly by applying gradient descent to optimize the network policy to the projected future reward. Williams et al. [46] proposed the reinforce method, which used a simple and quick reward to measure the policy's value.

3. Deep Reinforcement Learning-Based DQN Agent Drone Algorithm for Visual Object Tracking in a Virtual Environmental Simulation Platform

In this study, we propose a network that is connected to a virtual environment in order to obtain a runtime sequence of video frames and locate targeted objects in each image of the episode. The basic novelty of our algorithm is that it uses one of the most well-known RNNs for learning and predicting from long-term temporal sequences, integrated with action decision techniques inspired by the successful work [52]. RNNs enable the VCE simulation platform to exhibit temporal dynamic behavior by connecting nodes from direct or indirect graphs along a temporal sequence. An integrated deep reinforcement learning network can control actions by using RNN-based training sequences for making action decisions as an output for an object tracking procedure. Figure 2 depicts the overall framework of the proposal, which demonstrates the integrated scheme of the AirSim simulation platform with a virtual environment.

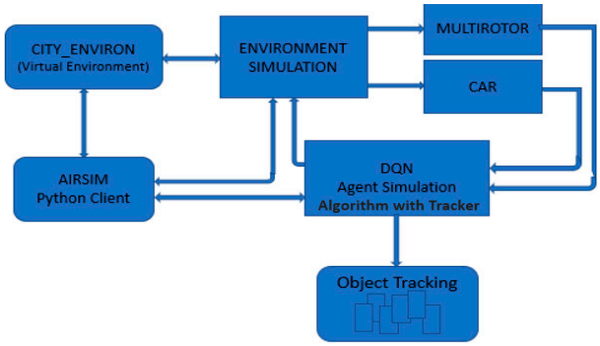

Figure 2. The design depicts a linked virtual world using the AirSim simulation platform to execute network algorithms and evaluate a tracking technique in real-time simulation.

The AirSim simulation platform includes two types of Multirotor (Drone) and Car options to test the approach via connecting AirSim Python Client, as shown in Figure 2. This DQN model discovers the VCE model relatively quickly while exploring and starting to learn the tactic of UAV agents. In this paper, we use a structure in Figure 2 (above) to learn sequential video frames by controlling policies in a variety of simulation environmental conditions. This is achieved by receiving input images and using them as input values for learning, as well as value estimation. The Environment Simulation code in our model connects the VCE and DQN agent simulation network algorithm with tracking via AirSim python client to control the drone simulation during training and tracking. The AirSim API gives an opportunity to run the algorithm on the VCE simulation platform and test it easily without any physical hardware system.

3.1. DQN Network Architecture

The recommended network model architecture incorporates reinforcement learning and a recurrent neural network method. The recurrent network is effective for applying sequential circumstances and for forecasting environmental target attributes. Figure 3, given below, illustrates the network structure of the learning procedure. The DQN network model processing steps with targeted action and state values of the learning procedure are depicted in the diagram below. The taken action and state values will be stored for the next step of training operations as an initial input value. Additionally, preserving learned information allows drone agents to avoid unnecessarily surveying the same location twice. This action may appear to be a repeated procedure, but in a virtual environment, the action and state value will be identical to the preceding one.

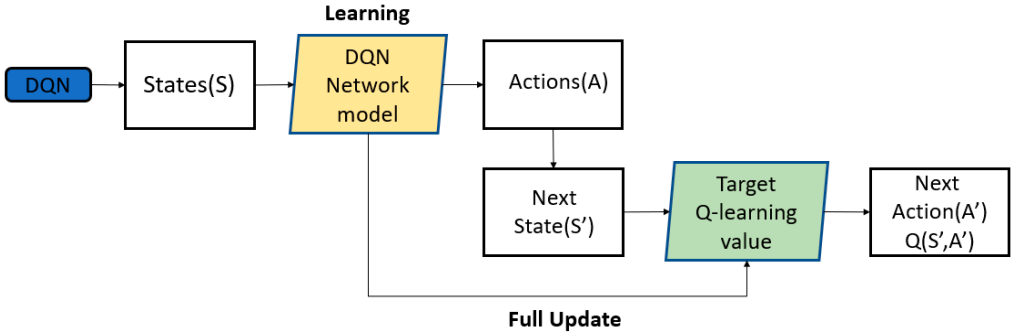

Figure 3. DQN network structure with targeted Q-learning outcomes in operation with states.

Figure 3 depicts a portion of Q-learning for updating the samples or minibatches for a given iteration. In this process, the updated Q-value determines the agent's actions, where the action with the highest targeted Q-learning value is chosen, and the update is accomplished through the use of a Bellman Equation (2) defined as

$$\acute{Q}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot \overbrace{\left(\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{optimum\ future\ value\ estimation} - \underbrace{Q(s_t, a_t)}_{old\ value} \right)}^{disparity\ in\ time}_{intended\ new\ value} \quad (2)$$

where $\acute{Q}(s_t, a_t)$ is the result of the updated Q-network iteration value; $Q(s_t,\ a_t)$ in which the initial old value will be added; α is the learning rate of the training network; r_t is the reward value; in this equation γ relates the rewards to the time domain; and $\max_a Q(s_{t+1}, a)$ is the optimum future value estimation.

In the equation above, the calculation of the updated Q-Network iteration value has been presented. An updated iteration value can be formulated by adding the previous $Q(s_t,\ a_t)$ value and multiplying the learning rate with α, which represents the difference in computation time. The max in this equation represents the maximized actions that help agents to take in the upper arcs of the VCE itself. Suppose, our drone agent was in state s and it took some action a. Because of that action, the environment might land our drone agent in any of the states s_{t+1}, and from these states, it would maximize the action. From these values, the drone agent will choose the action with the maximum Q value: $\max_a Q(s_{t+1}, a)$. Intended new values can be calculated by multiplying the discount factor γ with the maximum Q value and adding to them the reward r_t value.

In our DQN-network concept, the DQN agent contains a replay memory class unit that keeps track of the environment in dynamic mode. All state (s_t), action (a_t), new state (s_{t+1}),

reward (r_t), and done transitions are memorized. This replay memory approach allows us to effectively sample minibatches from preserved values and generate the accurate state representation. In order to obtain the best results from memorized values and keep track dynamically, a responding buffer memory is integrated with the VCE simulation platform.

Figure 4 illustrates the construction of the replay memory unit, where the storage monitoring process is shown. The buffer memory is directly connected to the virtual environment, which adds the required transition to the memory unit. During the sampling process, a random number of map indices of varying sizes are produced in memory, and the returned indices may be retrieved using the "get state" function of the AirSim python client. Minibatches will be generated by using the last saved values from the training process. Replay memory is one of the most critical individual core components of the DQN agent, separating the target Q-network and exhibiting the negative impacts on performance.

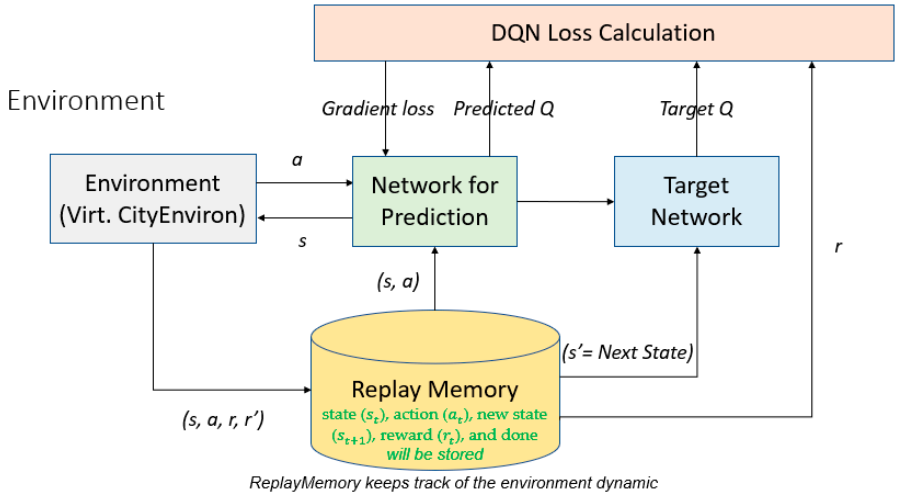

Figure 4. The data flow diagram for a DQN-network model with a responding buffer memory unit and a targeted network that is associated with a virtual environment [53].

As shown in Figure 4, the data flow diagram for the DQN-network model connects a Q-network and a recurrent network design (network for prediction) via the AirSim Python client to a virtual simulation platform. The accumulator keeps track of the N frames to be utilized for agent assessment. We can also compute the DQN loss function by combining forecasted and targeted Q-values, and we can obtain gradient loss output as well. The DQN network uses loss function for updating Q-learning at iteration i as follows [23]:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(\overbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}^{\text{targeted Q value}} - \overbrace{Q(s, a; \theta_i)}^{\text{predicted Q value}}\right)^2\right] \quad (3)$$

where γ denotes the discount factor value of the agent's horizon, θ_i are the parameters of the Q-network at iteration i, and θ_i^- are the network parameters utilized to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) at each defined step and are maintained constant between individual updates.

The underlying buffer with N previous states is stacked along the first axis in the replay memory unit and added to the state preservation. Furthermore, by using the reset function, the whole memory unit is reset to the underlying buffer, with all indexes set to zero (0).

3.2. Deep Q-Agent with Tracking Unit

We propose a DQN agent model that learns by utilizing a recurrent neural network model, whereas the authors in [23] adopted a convolutional neural network model. In our work, the recurrent layers are used to learn the features and information about the virtual simulation environment, while the method proposed in [23] used the convolutional neural network layers for that purpose. In our model, an additional featured approach is used to determine the eventual consequence of final action value. In the sequential environment learning process, the recurrent layers provide accurate predictions of state and action values with generalized data created from policies. Figure 5 illustrates a schematic representation of the DQN agent learning architecture integration with tracking based on recurrent neural networks.

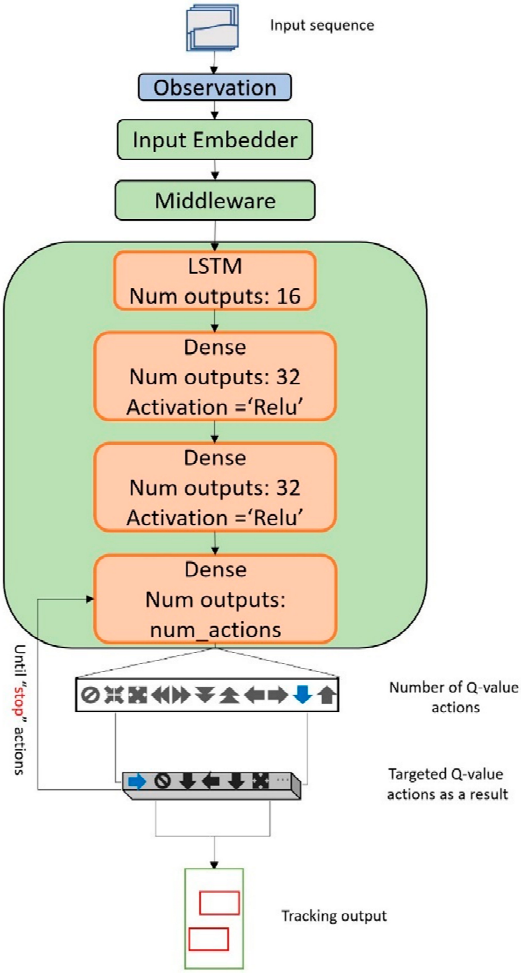

Figure 5. The implementation of the deep Q-neural network agent model initiation with the tracking process by calculating the intersection over union of two predicted bounding boxes from targeted action values.

In the first step of implementation, we configure the parameters of the DQN agent model with initial setup, where an action value model will be used by the agent to interact with the virtual simulation environment. A target model is used to compute the target Q-values in training, and is updated less frequently to increase the stability of the learning procedure of the DQN agent. The network model is built using a sequential mode by applying an LSTM layer with a deeply connected dense layer to change the dimensions of the vector from 64 to 32 with the activation-relu function. The network outputs will be state, Q, and action values to provide the drone agent with crucial information. A number of Q-value actions allow the agent to select the next action to perform in regard of the

current state of the environment. Subsequently, the drone agents' next targeted action will be chosen dependently by following predicted feature information of the trackable objects via LSTM network that are stored in a long–short term memory unit. Correspondingly, the drone agent takes best targeted action Q-values while tracking objects over time.

The network model's observation unit allows the agent to observe the number of Q-values through action functions on the old state. If finished, the process will be reset in the network's short-term memory, where it gives summary outcomes about one-time exploration episodes and summaries of learning procedures and appends to the network's long-term memory.

Network training's output summary includes total rewards, average MaxQ, duration, average loss, and timesteps of the full training episodes. Number of action Q-values emphasize different types of movement in terms of tracking objects in different trajectories. Given signs in a network model represent the direction of the drone agent's interaction with object locations and movement. For example, the following signs on the flow chart present (starts from right sight) up, down, left, right, two times up and down, two times left and right, resizing the object on the inner and outer side, and stopping action Q-values, respectively. Targeted action Q-values shown in Figure 5 are taken after predicting object locations relatively.

3.3. Training the DQN Network Model

The training process allows the agent to train itself to better understand the environment dynamics and compute the expected rewards for the next state ($t + 1$). Additionally, it allows the agent to update the expected reward at step t according to the first training episode outcomes of the network model. The target expectation is computed through the targeted Q-values of the actions, which is a more stable version of the action value for increasing training stability. In actual cases, the target network is a frozen copy of the action value network updated as regular intervals. After the training process, the network clips all positive rewards at 1 and all negative rewards at -1, leaves 0 rewards unchanged, trains the network again for the next episode, updates the target network, and eventually saves the network output files into a fixed path. There is an extension of the train function that calls the batch generation and graph computations for sampling random minibatches of transitions from the replay memory unit. We set the hyperparameters and their default values for the deep Q-agent before training the network. At the end, we summarize all the learned values of total reward, average MaxQ, duration, and average loss, and save them into the adjusted path location.

Training Dataset

For the training of RNN-based object prediction and DQN network modules, we created our own dataset from the VCE simulation model. We captured images for training and testing purposes, totaling 727 and 195 images, respectively. To ensure compatibility with the training and testing processes, all images were manually labeled. Images include two types of objects: pedestrians and cars. Overall, 922 images were used for the training and testing process, which is sufficient to produce accurate outputs for analyzing proposals. Training parameters were fine-tuned with several different values while examining the algorithm under different conditional changes. Additionally, the tracking algorithm was tested in runtime on a simulation platform to observe the tracking performance.

3.4. Tracking Baseline of the DQN Network Model

In the tracking implementation section, we utilized a supervised approach to identify object classes, information, and properties, and we combined it with the DQN agent simulation network architecture. While testing the suggested tracker, the tracking approach employed a pretrained object classifier model to detect targets from a virtual simulation platform. Our proposed tracker was implemented by integrating a DQN network as a sequential decision-making procedure with a drone agent in the VCE simulation plat-

form. Moreover, the network model observation part represents the virtual environment sequences and recurrent network-based architecture layers and provides an output with predicted bounding box locations in each frame, as shown in Figure 5. We first trained our network to predict target classes in proper action to provide environment states. The network was then updated with a deep q-learning approach to ensure that the drone agent continued to learn effectively from high-dimensional virtual simulation environment inputs via end-to-end reinforcement learning.

The network gives the output prediction from learning process, and we integrated tracking units by calculating the intersection over union (IoU) of bounding boxes. The origin from the Cartesian coordinate system at the center of the right frame and in the top position was considered as a positive axis. Then, the coordinates of the intersection rectangle were determined by identifying the maximum and minimum values. The intersection area of the two axis-aligned bounding boxes was always considered as an axis-aligned bounding box value. Then, we computed the area of both axis-aligned bounding boxes. Intersection over union was calculated by taking the computational intersection area and dividing it by the sum of the prediction plus ground truth areas minus the intersection area. The result is asserted to be a value between zero and one. The next step was to interpret the action sequence using a simulation environment and tracking calculation activity, which was applied to the interpreted action sequence.

The reward function was calculated as a scaled sum of the Euclidean distance between the center of the bounding box and the center of the frame, intersection over union of bounding boxes, and an imaginary box centered with parameters threshold height and weight. The completed portion will be determined by taking reward values at predetermined intervals. The final stage was to create a reinforcement agent to configure the specified parameters and test the algorithm using virtual environment simulation model inputs.

4. Experiment Results and Discussion

4.1. Datasets for Evaluation: VisDrone2019 and OTB-100

There are various opensource datasets available for measuring and evaluating the suggested method, as well as comparing with other state-of-the-art models. Applying the identical image or video sets technique to the algorithm reveals the advantages and weaknesses of the recommended method while evaluating it among other state-of-the-art models in different criteria of the learning field. VisDrone2019 [54] and OTB-100 [55] datasets are opensource dataset benchmarks for evaluating measures and competing in different challenges for object detection/tracking techniques.

The VisDrone2019 [54] dataset was collected by the AISKYEYE team at the Lab of Machine Learning and Data Mining, Tianjin University, China. The dataset benchmark consists of 288 video clips formed by 261,908 frames and 10,209 static images, captured by various drone-mounted cameras and covering a wide range of aspects including location (taken from 14 different cities separated by thousands of kilometers in China), environment (urban and rural), objects (pedestrian, vehicles, bicycles, etc.), and density (sparse and crowded scenes). Notably, the dataset was collected using various drone platforms (i.e., drones with different models), in different scenarios, and under various weather and lighting conditions. These frames are manually annotated with over 2.6 million bounding boxes of targets of frequent interests, such as pedestrians, cars, bicycles, and tricycles. Some important attributes including scene visibility, object class, and occlusion are also provided for better data utilization.

The full OTB-100 [55] benchmark contains 100 sequences from recent literatures that address an extensive evaluation of the state-of-the-art online object tracking algorithms with various evaluation criteria to understand how these methods perform within the same framework. The authors initially constructed a large dataset with ground-truth object positions and extents for tracking, and introduced the sequence attributes for the performance analysis. Additionally, they integrated most of the publicly available trackers into one code library with uniform input and output formats to facilitate large-scale performance evalua-

tion. Moreover, extensive evaluation was conducted by the performance of 31 algorithms on 100 sequences with different initialization settings. By analyzing the quantitative results, an effective approach for robust tracking was identified and provided along with potential future research directions in this field.

4.2. Evaluation and Discussion

The proposed reinforcement learning-based object tracking algorithm has been explored and evaluated using AirSim [22], a well-known simulation platform that is highly useful and advantageous for exploring freely in a realistic environment simulation. The implementation of the proposed algorithm was connected to the VCE simulation platform with AirSim Python Client that was accomplished by Microsoft developers.

Firstly, we trained the network to learn the object classes with environmental feature information, so that it was ready for training with a reinforcement learning approach. After completing the object classification learning process, the output was used in an integrated tracking framework to track objects with a drone agent. Figure 6 shows the evolution of the learning rate and loss function of the trained tracker. It can be seen that learning rate image on the left side illustrates small variation of the learning outcome; due to the environmental conditions, that drone flew in an extensive area and provided a small learning rate. Alternatively, on right side, loss functions results increase during training epochs relatively by learning a large-scale area of the virtual environment.

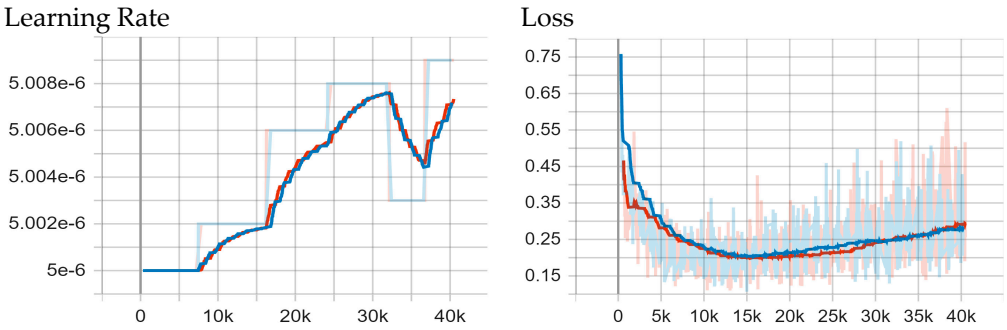

Figure 6. Output of the Learning Rate (**left**) and Loss (**right**) function. The blue and red lines represent actual and targeted learned values, respectively. The left side of the learning rate figure y axis represents given parameter learning rate variation value and the x axis represents training epoch numbers. The right side of the image y axis represents loss variation values and the x axis is training epoch value. The shadow copies of trained variables and ops that maintain a moving average of the trained variables in their shadow copies are shown.

This training was conducted in offline mode to understand the characteristics and details of the object class. For the training method, numerous sequential image sets taken from a realistic virtual world were employed as inputs. Only VCE realistic virtual model pictures were used in the trials to train the DQN tracker, which is based on a recurrent neural network. The figures shown below represent the average predicted action-value MeanQ and MaxQ computed over the held-out set of states.

Figure 7 shows the training epochs outcome for computing average action values for MeanQ/MaxQ that examines learning and predicting procedures of the recurrent neural network-based DQN tracker. Figure 7 (left) portrays the evolution of the training procedure by using the virtual realistic images as training data. Figure 8 represents the evolution of both the epsilon score and the used replay memory during the tracker training process. The epsilon greedy exploration graph represents a chosen random action with a probability of epsilon that exploits the best-known action value and can probably reach nearly one epsilon value. Figure 8 (right) displays replay memory, also known as the replay buffer or experience buffer, and contains a collection of experience tuples $(s, a, r, s^{'})$ with training information. They were added gradually to the buffer as we interacted with the virtual

realistic environment. This act of sampling a small batch of tuples from the replay buffer in order to learn the environment is also known as experience replay. It allowed us to learn the environment more deeply with individual tuples multiple times and make better use of learning experiences during training.

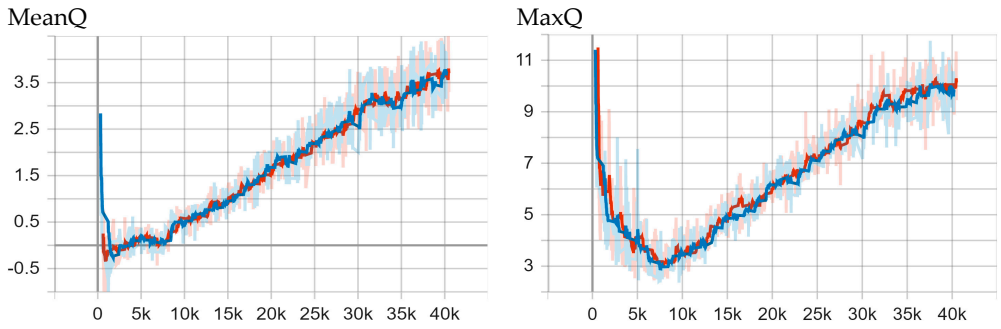

Figure 7. The average predicted action value of the MeanQ (**left**) and MaxQ (**right**) values with two colored lines (red—target and blue—DQN value loss). The MeanQ figure (left) represents the variation values of the average Q on the y axis along with the number of training epochs, and MaxQ on the right figure shows the maximum achieved Q value variation while training on the y axis jointly with the number of epochs on the x axis. Shadow variables are copies of the main train variables that are used to maintain the moving average. The idea is that they follow the main variable like a shadow.

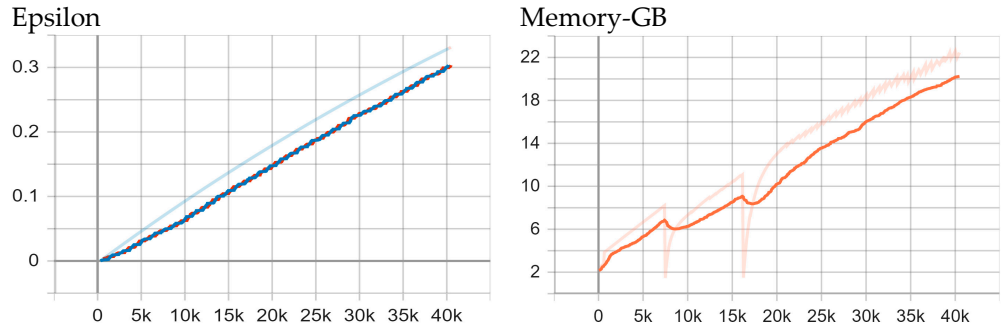

Figure 8. The average achieved epsilon score (Epsilon, **left**) and used replay memory (Memory-GB, **right**) value during the training process (red—target and blue—DQN value loss in epsilon graph; orange—used replay memory unit in GB). Epsilon graph shows the balances exploration and exploitation by choosing randomly in y axes that refers to the probability of choosing to explore, while x axes depict the training epochs. The memory graph represents the used memory space during training in the y axes as well as training epochs in the x axes.

In summary, the basic idea of using replay buffer memory, or experience replay memory, is to take advantage of a strong experience and use a random subset of the experience to update the Q-network. Rather than using the last single experience outcome during the tracking process, this action was originally used for the learning tuples of observation state, action, reward, done flag, and next state parameters to keep the obtained transitions from the virtual environment.

After an offline training process of the tracker, the network was updated and connected to the virtual realistic environment in order to be tested in real time. A drone agent gave several output parameter results (shown in Table 1), which were configured before the testing process, illustrating the virtual drone agent's behavior during the testing process as well as providing summarizing episode (Table 2) results.

Table 1. DQN agent drone testing output parameters.

Parameters	Episodes (Randomly Chosen)									
	276	**279**	**287**	**315**	**333**	**382**	**768**	**815**	**921**	**999**
Delta X	17.08	67.51	23.14	43.24	51.45	31.46	45.12	18.53	14.36	24.47
Delta Y	−56.3	−43.16	−53.64	−66.56	−61.03	−68.00	−64.96	−59.21	−51.65	−74.82
Delta Z	−20.89	−11.40	−20.66	−10.07	15.00	18.31	−0.36	−20.66	−20.46	−8.57
UAV Vel	0.77, −0.83, 1.03	1.29, −0.1, 0.12	1.36, −0.79, −0.75	0.63, −0.36, 0.61	1.19, −0.12, 0.10	0.12, −0.07, 1.59	1.80, −0.70, 1.13	0.64, −0.92, −1.01	0.89, −0.73, −1.55	0.83, −0.89, 1.42
UAV Pos	17.54, −56.78, −40.51	68.23, −43.26, −30.36	23.94, −54.10, −40.12	43.61, −66.79, −28.74	52.16, −61.09, −3.96	31.54, −68.05, 0.11	46.15, −65.35, −18.72	18.91, 59.74, −40.28	14.91, −52.07, −40.41	24.99, −75.36, −26.72
Distance XY	60.01	80.49	59.69	80.07	80.46	75.50	80.26	63.33	54.87	80.04
Distance Z	21.49	11.34	21.09	9.72	15.06	19.13	0.29	21.26	21.39	7.70
Reward XY	0.24	−0.006	0.25	−0.0009	−0.005	0.05	−0.003	0.20	0.31	−0.0005
Reward Z	−73.56	−9.66	−68.01	−6.98	−20.33	−45.96	−0.05	−70.27	−72.16	−4.667
Reward (+T)	−73.31	−9.67	−67.75	−6.99	−20.335	−45.91	−0.06	−70.06	−71.84	−4.668
Reward	−10	−10	−10	−10	−10	−10	−10	−10	−10	−10
Action RL	(0.25, 0, 0) +m/s	(0, −0.25, 0) +m/s	(0.25, 0, 0) +m/s	(0, −0.25, 0) +m/s	(0, −0.25, 0) +m/s	(0.25, 0, 0) +m/s	(0, −0.25, 0) +m/s	(0, −0.25, 0) +m/s	(0, −0.25, 0) +m/s	(0, −0.25, 0) +m/s
Done	1	1	1	1	1	1	1	1	1	1

Table 2. Action-value based testing output of the DQN tracker.

Summary	Episodes (Randomly Chosen)									
	276	**279**	**287**	**315**	**333**	**382**	**768**	**815**	**921**	**999**
Time step	1924	1947	2001	2188	2300	2604	5170	5633	5912	6341
Duration	28	89	29	69	68	97	47	33	22	60
Epsilon	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Total Reward	−609.86	−995.15	−640.83	−1084.84	−360.74	−475.99	−341.57	−665.84	−402.24	−1586.36
Avg_Max_Q	0.1837	0.1839	0.1834	0.1958	0.1258	0.0874	0.1352	0.1549	0.1442	0.1733
Avg_Loss	0.052	0.063	0.062	0.069	0.051	0.047	0.066	0.061	0.073	0.071
Mode	random	random	random	random	random	random	random	random	random	random

Table 1 depicts the parameters of the drone agent while testing it in the VCE model. Information given in Table 1 provides the drone agent's coordinates while learning the environment in random episodes as well as rewards of the DQN model drone. In every randomly chosen episode of the training, the location parameters show the drone agent's tracking path in a virtual environment. Reward XY, Z, and +T parameters emphasize the drone agent's adaptation to environmental conditions while analyzing it in sequential actions. It can be seen that while training the realistic VCE, the drone agent obtains negative values by flying in different directions because of environmental space and conditions; behavior of the VCE model affects the result of the rewards. Additionally, Table 2 shows a summarized action Q-value result during the testing process.

The result of the tracker in the realistic VCE model is presented in Table 2. The summary results of randomly selected episodes, such as time steps, duration, total reward, average max Q-value, and average loss results of the testing procedure, are displayed in this table. The drone traveled in a random area and identified the item sites to track during the testing phase, which means that the drone flew in a random location and identified the object locations to track.

4.3. Comparison and Qualitative Results

The suggested tracking model was evaluated using VisDrone2019 [54] and OTB-100 [55] to compare it to recent state-of-the-art object tracking models based on deep reinforcement learning techniques. These datasets are available on the internet in a variety of image and video sets, including training, testing, and challenge sets.

Figure 9 compares the performance of the proposed tracking methodology with that of recent state-of-the-art trackers, such as ADNet [52] and ASRL Track [56], which are both based on the DRL (Deep Reinforcement Learning) strategy. Comparison was carried out by testing video sets of the two public datasets VisDrone2019 [54] and OTB-100 [55] respectively. The graphic above depicts precision performance with regard to local error threshold outcomes in two public datasets. Table 3 (below) shows the precise results of the numerical comparison:

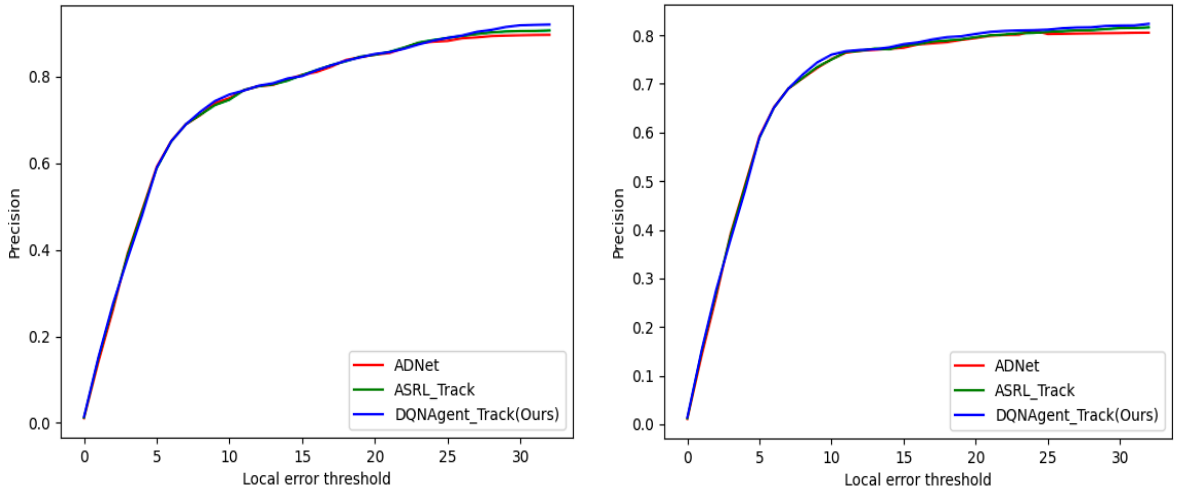

Figure 9. In this plot, comparison results of three different types of RL-based recent works are shown. The red (ADNet [52]), green (ASRL_Track [56]), and blue (ours) colors show the result of comparison in testing the (**a**) VisDrone2019 and (**b**) OTB-100 datasets respectively.

Table 3. Comparison results of recent tracking algorithms with proposed method on VisDrone2019 and OTB-100 datasets.

Algorithms	VisDrone2019			OTB-100		
	Precision	**FPS**	**IOU**	**Precision**	**FPS**	**IOU**
ADNet [52]	89.69%	4.45	0.599	80.52%	4.03	0.492
ASRL_Track [56]	90.70%	6.94	0.602	81.62%	6.41	5.99
Ours	92.07%	9.11	0.785	82.35%	7.18	0.662

In Table 3, the results of a comparison of recent RL-based object tracking approaches and the proposed model are emphasized. The table compares the accuracy, frames per second (FPS), and intersection over union (IoU) outcomes of two current approaches with our model. Among these approaches, our strategy outperformed the others when tested on the identical open-source datasets. For the testing process, video inputs were used to examine the tracking capabilities of the approaches. Moreover, we provide some qualitative results below in Figure 10.

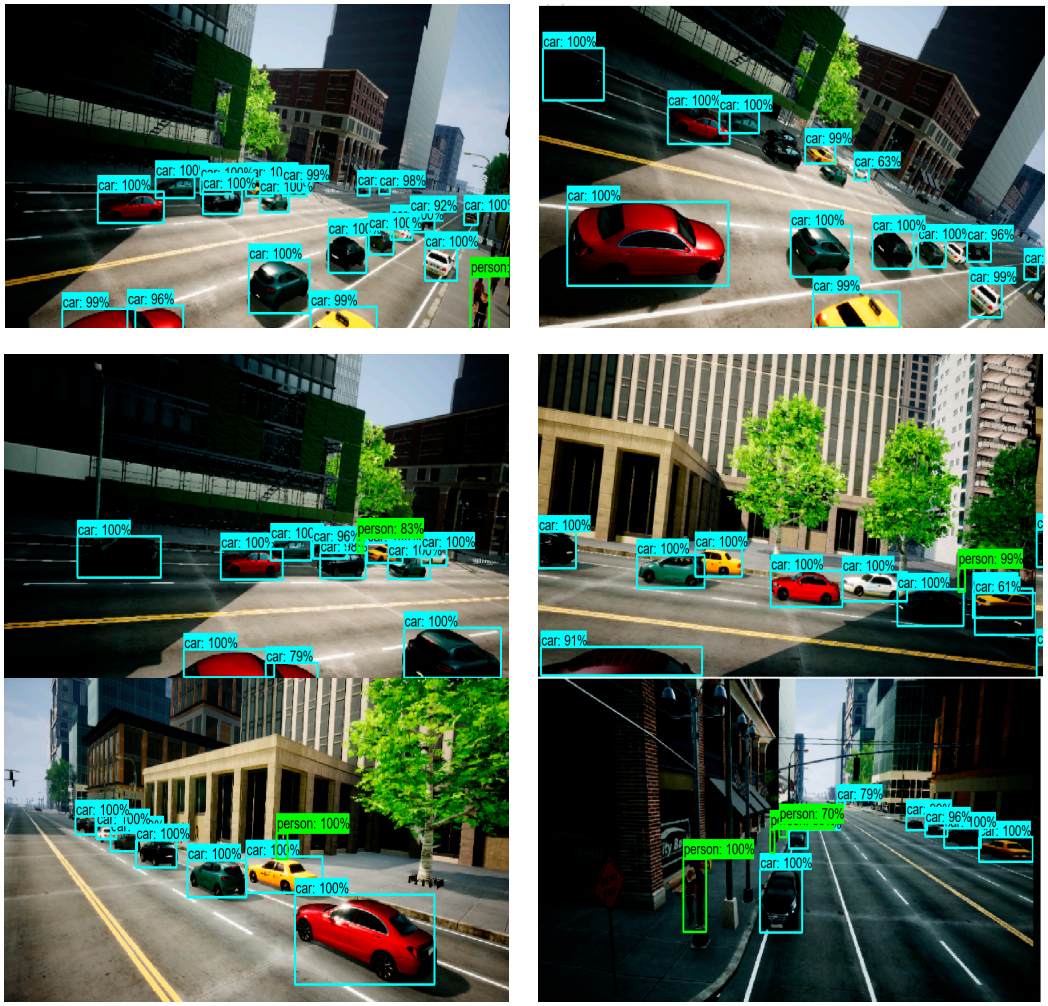

Figure 10. Qualitative results of the proposed tracking algorithm with two object classes: person and car.

Figure 10 shows the qualitative results of the tracking model with two types of targeted objects. As we can see in Figure 10, the percentages of predicted and targeted object classes indicate how well the object classes were predicted. The result in Figure 10 demonstrates a good performance in terms of predicted and tracked object classes. We tested our proposed algorithm only in this virtual environment, and it performed much better compared to the other models, which, in the majority, have been tested with real video sequences. We have tested our algorithm only with the default simulation environment case, which is the normal weather condition.

5. Conclusions

In this research, we have proposed a novel tracking technique that integrates the virtual simulation platform VCE using the AirSim python client that performs in virtual realistic environment as a drone agent. In order to predict and track the objects and to learn the environment, we used a mutually integrated recurrent neural network-based DQN tracker that was trained with several virtual image-based video sequences. The AirSim simulation platform allowed us to test our model in a virtual environment, gather crucial feature information, and identify the object classes autonomously. AirSim API allowed us to test our DQN agent-based tracker easily by connecting it directly to the virtual drone simulation platform. Even though there are several challenges in terms of a three-dimensional virtual realistic model environment, our proposed model was successfully trained and achieved better performance with a recurrent prediction-based network integrated with an action decision technique. Our model can work autonomously

in a virtual simulation model by applying deep RL agent solutions. Additionally, we tested our model with two different data sets, VisDrone2019 and OTB-100, and compared our model performance with recent state-of-the-art techniques. Testing evaluation showed that our proposed technique displayed better performance among recent methods, with 92.07% and 82.35% precision in VisDrone2019 and OTB-100 data sets, respectively.

In our future works, we plan to improve the performance by using fine-tuning methodology and performing more simulation experiments with different weather conditions. Moreover, we plan to test our model with other open-source video sequences in order to compare it with other conventional RL-based DQN trackers.

Author Contributions: Conceptualization, J.-H.P.; funding acquisition, K.F., S.-H.L. and K.-R.K.; investigation, J.-H.P.; methodology, J.-H.P.; project administration, K.F., S.-H.L. and K.-R.K.; software, J.-H.P., K.F., S.-H.L. and K.-R.K.; supervision, K.F., S.-H.L. and K.-R.K.; validation, K.F., S.-H.L. and K.-R.K.; writing—original draft, J.-H.P.; writing—review and editing, J.-H.P., K.F. and S.-H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Brain Korea 21 project (BK21).

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: The original VisDrone2019 and OTB-100 datasets are available online at https://paperswithcode.com/dataset/otb (accessed on 1 January 2022) and http://aiskyeye.com/download/ (accessed on 1 January 2022). These datasets used for comparing with algorithm performance with recent state-of-the-art models.

Acknowledgments: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2020R1I1A306659411, 2020R1F1A1069124) and MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2022-2020-0-01797) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ma, H.; Lin, Z.; Action, S.T. FAST: Fast and accurate scale estimation for tracking. *IEEE Signal Process. Lett.* **2019**, *27*, 161–165. [CrossRef]
2. Zhang, Y.; Wang, C.; Wang, X.; Wen-Jun, Z.; Liu, W. FairMOT: Fairness of detection and re-identification in multi object tracking. *Int. J. Comput. Vis. (IJCV)* **2021**, *129*, 3069–3087. [CrossRef]
3. Liu, Q.; Liu, B.; Wu, Y.; Li, W.; Yu, N. Real-time online multi-object tracking in compressed domain. *IEEE Access* **2019**, *7*, 76489–76499. [CrossRef]
4. Wang, G.; Luo, C.; Sun, X.; Xiong, Z.; Zeng, W. Tracking by instance detection: A meta-learning approach. In Proceedings of the Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 6288–6297.
5. Danelljan, M.; Hager, G.; Khan, F.S.; Felsberg, M. Convolutional features for correlation filter based visual tracking. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Washington, DC, USA, 7–13 December; pp. 58–66.
6. Iswanto, I.A.; Chao, T.W.; Li, B. Object tracking based on Meanshift and Particle-Kalman filter algorithm with multi features. *Procedia Comput. Sci.* **2019**, *157*, 521–529. [CrossRef]
7. Xie, J.; Stensrud, E.; Skramstad, T. Detection-Based Object Tracking Applied to Remote Ship Inspection. *Sensors* **2021**, *21*, 761. [CrossRef]
8. Riahi, D.; Bilodeau, G. Online multi-object tracking by detection based on generative appearance models. *Comput. Vis. Image Underst.* **2016**, *152*, 88–102. [CrossRef]
9. Lan, X.; Yuen, P.C.; Chellappa, R. Robust MIL-based feature template learning for object tracking. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 September 2017.
10. Hare, S.; Golodetz, S.; Vineet, V.; Cheng, M.; Hicks, S.L.; Saffari, A.; Torr, P.H. Struck: Structured output tracking with kernels. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 2096–2109. [CrossRef]
11. Park, E.; Ju, H.; Jeong, Y.M.; Min, S. Tracking-learning-detection adopted unsupervised learning algorithm. In Proceedings of the 7th International Conference on Knowledge and System Engineering (KSE), Ho Chi Minh City, Vietnam, 8–10 October 2015.
12. Bertinetto, L.; Valmadre, J.; Henriques, J.F.; Vedaldi, A.; Torr, P.H. Fully-convolutional siemence networks for object tracking. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016; pp. 850–865.

13. Held, D.; Thrun, S.; Savarese, S. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016; pp. 749–765.
14. Nam, H.; Han, B. Learning multi-domain convolutional neural networks for visual tracking. *arXiv* **2015**, arXiv:1510.07945.
15. Farhodov, X.; Oh-Heum, K.; Kwang-Seok, M.; Oh-Jun, K.; Suk-Hwan, L.; Ki-Ryong, K. A New CSR-DCF Tracking Algorithm based on Faster RCNN Detection Model and CSRT Tracker for Drone Data. *J. Korea Multimed. Soc.* **2019**, *22*, 1415–1429.
16. Gordon, D.; Farhadi, A.; Fox, D. Re^3: Real-time recurrent regression networks for visual object tracking of generic objects. *IEEE Robot. Autom. Lett. ICRA* **2018**, *3*, 788–795. [CrossRef]
17. Hong, S.; You, T.; Kwak, S.; Han, B. Online tracking by learning discriminative saliency map with convolutional neural network. In Proceedings of the International Conference on Machine Learning PMLR, Lille, France, 6–7 July 2015.
18. Wang, N.; Li, S.; Gupta, A.; Yeung, D.Y. Transferring rich feature hierarchies for robust visual tracking. *arXiv* **2015**, arXiv:1501.04587.
19. Kristan, M.; Pflugfelder, R.; Leonardis, A.; Matas, J.; Cehovin, L.; Nebehay, G.; Felsberg, M.; Joni-Kristian, K.; Danelljan, M.; Drbohlav, O.; et al. The eighth visual object tracking VOT2020 challenge results. In Proceedings of the Visual Object Tracking Workshop 2020 at ECCV, online, 23–28 August 2020.
20. Fan, H.; Miththanthaya, H.A.; Rajan, S.R.; Liu, X.; Zou, Z.; Lin, Y.; Ling, H. Transparent object tracking benchmark. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 10734–10743.
21. Donahue, J.; Hendricks, L.A.; Guadarrama, S.; Rohrbach, M.; Venugopalan, S.; Saenko, K.; Darrell, T. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 2625–2634.
22. Qasim, A.B.; Pettirsch, A. Recurrent neural networks for video object detection. *arXiv* **2020**, arXiv:2010.15740.
23. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Springer: Cham, Switzerland, 2018.
24. Mnih, V.; Koray, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
25. Bhatt, D.; Patel, C.; Talsania, H.; Patel, J.; Vaghela, R.; Pandya, S.; Modi, K.; Ghayvat, H. CNN variants for computer vision: History, architecture, application, challenges and future scope. *Electronics* **2021**, *10*, 2470. [CrossRef]
26. Wang, Y.; Velswamy, K.; Huang, A.B. Long-Short Term Memory Recurrent Neural Network Based Reinforcement Learning Controller for Office Heating Ventilation and Air Conditioning Systems. *Processes* **2017**, *5*, 46. [CrossRef]
27. Kristan, M.; Matas, J.; Leonardis, A.; Felsberg, M.; Cehovin, L.; Fernandez, G.; Vojir, T.; Hager, G.; Nebehay, G.; Pflugfelder, R. The visual object tracking vot2015 challenge results. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Washington, DC, USA, 7–13 December 2015; pp. 1–23.
28. Guo, D.; Wang, J.; Cui, Y.; Wang, Z.; Chen, S. SiamCAR: Siamese fully convolutional classification and regression for visual tracking. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 6269–6277.
29. Salscheider, N.O. Object tracking by detection with Visual Motion Cues. *arXiv* **2021**, arXiv:2101.07549v1.
30. Maras, B.; Arica, N.; Ertuzun, A.B. Object tracking by combining tracking-by-detection and marginal particle filter. In *Proceedings of the 24th Signal Processing and Communication Application Conference (SIU), Zonguldak, Turkey, 16–19 May 2016*; IEEE: Piscataway, NJ, USA, 2016.
31. Chahyati, D.; Fanany, M.I.; Arimurthy, A.M. Tracking people by detection using CNN features. *Procedia Comput. Sci.* **2017**, *124*, 167–172. [CrossRef]
32. Wang, N.; Shi, J.; Yeung, D.Y.; Jia, J. Understanding and diagnosing visual tracking systems. In Proceedings of the IEEE International Conference on Computer Vision, Washington, DC, USA, 7–13 December 2015; pp. 3101–3109.
33. Rahman, M.; Ahmed, R.; Laishram, L.; Kim, S.H.; Jung, S.K. Siamese high-level feature refine network for visual object tracking. *Electronics* **2020**, *9*, 1918. [CrossRef]
34. Zhang, K.; Liu, Q.; Wu, Y.; Yang, M.-H. Robust visual tracking via convolutional networks. *arXiv* **2015**, arXiv:1501.04505.
35. Xiao-Qin, Z.; Run-Hua, J.; Chen-Xiang, F.; Tian-Yu, T.; Wang, T.; Peng-Cheng, H. Advances in deep learning methods for visual tracking: Literature review and fundamentals. *Int. J. Autom. Comput.* **2021**, *18*, 311–333.
36. Danelljan, M.; Robinson, A.; Khan, F.S.; Felsberg, M. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016.
37. Kahou, S.E.; Michalski, V.; Memisevic, R. Ratm: Recurrent attentive tracking model. *arXiv* **2015**, arXiv:1510.08660.
38. Gan, Q.; Guo, Q.; Zhang, Z.; Cho, K. First step toward model-free, anonymous object tracking with recurrent neural networks. *arXiv* **2015**, arXiv:1511.06425.
39. Farhodov, X.; Moon, K.; Lee, S.; Kwon, K. LSTM network with tracking association for multi-object tracking. *J. Korea Multimed. Soc.* **2020**, *23*, 1236–1249.
40. Ning, G.; Zhang, Z.; Huang, C.; He, Z.; Ren, X.; Wang, H. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv* **2016**, arXiv:1607.05781.
41. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2016; pp. 779–788.

42. Sutton, R.S. *Introduction to Reinforcement Learning*; MIT Press: London, UK, 2015; Volume 135.
43. Luo, W.; Sun, P.; Zhong, F.; Zhang, T.; Wang, Y. End-to-end active object tracking and its real-world deployment via reinforcement learning. In Proceedings of the Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–19 June 2019.
44. Nabati, R.; Harris, L.; Qi, H. CFTrack: Center-based Radar and camera fusion for 3D multi-object tracking. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 3–8 January 2021.
45. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. Mastering the game of go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
46. Barreto, A.; Hou, S.; Borsa, D.; Silver, D.; Precup, D. Fast reinforcement learning with generalized policy updates. *Proc. Natl. Acad. Sci. USA* **2020**, *117*, 30079–30087. [CrossRef] [PubMed]
47. Caicedo, J.C.; Lazebnik, S. Active object localization with deep reinforcement learning. In Proceedings of the IEEE International Conference on Computer Vision, Washington, DC, USA, 7–13 December 2015; pp. 2488–2496.
48. Jayaraman, D.; Grauman, K. Look-ahead before you leap: End-to-end active recognition by forecasting the effect of motion. *arXiv* **2016**, arXiv:1605.00164.
49. Zhang, D.; Maei, H.; Wang, X.; Wang, Y. Deep Reinforcement Learning for Visual Object Tracking in Videos. *arXiv* **2017**, arXiv:1701.08936.
50. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. *arXiv* **2015**, arXiv:1509.06461.
51. Wang, Z.; de Freitas, N.; Lanctot, M. Dueling network architectures for deep reinforcement learning. *arXiv* **2015**, arXiv:1511.06581.
52. Yun, S.; Choi, J.; Yoo, Y.; Yun, K.; Choi, J.Y. ADNet: Action-Decision Networks for visual tracking with deep reinforcement learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
53. Nair, A.; Srinivasan, P.; Blackwell, S.; Alcicek, C.; Fearon, R.; de Maria, A.; Panneershelvam, V.; Suleyman, M.; Beattie, C.; Petersen, S.; et al. Massively parallel methods for deep reinforcement learning. *arXiv* **2015**, arXiv:1507.04296. Available online: http://arxiv.org/abs/1507.04296 (accessed on 10 September 2021).
54. Zhu, P.; Wen, L.; Du, D.; Bian, X.; Hu, Q.; Ling, H. Vision meets drones: Past, present and future. *arXiv* **2020**, arXiv:2001.06303.
55. Wu, Y.; Lim, J.; Yang, M. Object tracking benchmark. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*; IEEE: Piscataway, NJ, USA, 2015; Volume 37, pp. 1834–1848.
56. Go'zen, D.; O'zer, S. Visual object tracking in drone images with deep reinforcement learning. In Proceedings of the 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021.