drones

Article

Application of Reinforcement Learning in Controlling Quadrotor UAV Flight Actions

Shang-En Shen and Yi-Cheng Huang *

Department of Mechanical Engineering, National Chung Hsing University, Taichung 40227, Taiwan; g111061305@mail.nchu.edu.tw
* Correspondence: ychuang66@dragon.nchu.edu.tw

Abstract: Most literature has extensively discussed reinforcement learning (RL) for controlling rotorcraft drones during flight for traversal tasks. However, most studies lack adequate details regarding the design of reward and punishment mechanisms, and there is a limited exploration of the feasibility of applying reinforcement learning in actual flight control following simulation experiments. Consequently, this study focuses on the exploration of reward and punishment design and state input for RL. The simulation environment is constructed using AirSim and Unreal Engine, with onboard camera footage serving as the state input for reinforcement learning. The research investigates three RL algorithms suitable for discrete action training. The Deep Q Network (DQN), Advantage Actor–Critic (A2C), and Proximal Policy Optimization (PPO) were combined with three different reward and punishment design mechanisms for training and testing. The results indicate that employing the PPO algorithm along with a continuous return method as the reward mechanism allows for effective convergence during the training process, achieving a target traversal rate of 71% in the testing environment. Furthermore, this study proposes integrating the YOLOv7-tiny object detection (OD) system to assess the applicability of reinforcement learning in real-world settings. Unifying the state inputs of simulated and OD environments and replacing the original simulated image inputs with a maximum dual-target approach, the experimental simulation achieved a target traversal rate of 52% ultimately. In summary, this research formulates a set of logical frameworks for an RL reward and punishment design deployed with real-time Yolo's OD implementation synergized as a useful aid for related RL studies.

Keywords: quadrotor UAV; reinforcement learning; logic control; target recognition; action decision making

Citation: Shen, S.-E.; Huang, Y.-C. Application of Reinforcement Learning in Controlling Quadrotor UAV Flight Actions. *Drones* **2024**, *8*, 660. https://doi.org/10.3390/drones8110660

Academic Editor: Bo Li

Received: 3 October 2024
Revised: 31 October 2024
Accepted: 5 November 2024
Published: 9 November 2024

1. Introduction

Drones are prone to damage from collisions during operation, which can lead to work stoppages. As a result, drone "automation" has become one of the most prominent areas of focus in the field. For a drone to perform autonomous flight tasks, it must possess three essential capabilities: first, the ability to identify mission targets; second, the ability to avoid obstacles based on the environment; and third, the ability to integrate both target recognition and environmental data to plan the mission path. However, due to the unique nature of a drone's rapid movement through space, the challenges faced by the technologies of visual recognition, obstacle avoidance, and path planning differ significantly from those encountered by other vehicles. In earlier approaches, technicians attempted to achieve drone autonomy by combining sensors with logical decision-making processes, which was not only time-consuming in design but also often complicated by the unpredictability of environmental factors.

1.1. Object Detection

The rise in artificial intelligence technology has brought more innovation and convenience to autonomous flight missions. With the advancement of image recognition technology, object

detection techniques have been increasingly applied in drone target identification tasks. Among the most notable algorithms are YOLO (You Only Look Once) [1], SSD (Single Shot MultiBox Detector) [2], and RCNN (Region Convolutional Neural Network) [3].

Currently, drones commonly utilize LiDAR and depth cameras to create maps and achieve autonomous flight functionality. However, payload limitations and computational efficiency are often challenges that need to be addressed. Therefore, developing a drone control system that integrates 2D vision with reinforcement learning, allowing the drone to recognize targets and navigate through obstacles, could not only reduce the costs associated with carrying LiDAR or depth cameras but also decrease the time required for manual design. This represents a key development direction for autonomous drone flight. Research related to object detection in drone applications is as follows:

Zhengxin Zhang [4] proposed a series of object detection algorithms based on modifications to the YOLO model, known as Drone-YOLO. By modifying the Backbone and Neck sections of the neural network, accuracy was improved by 13.4% in the VisDrone2019 dataset, with the smaller models in the series also demonstrating outstanding performance.

Yongshuai [5] introduced GGT-YOLO, a novel target detection technology for drones engaged in maritime reconnaissance. This approach incorporates a Transformer to enhance feature extraction capabilities, thereby improving the recognition of small or occluded targets. Additionally, GhostNet is employed to replace complex convolutional operations with simpler linear transformations, effectively reducing computational costs on unmanned aerial vehicles. The synergy of these two technologies makes GGT-YOLO particularly suitable for application in the maritime drone detection field.

1.2. Reinforcement Learning

Due to the susceptibility of drones to damage from collisions, it is crucial to establish simulation environments where drones can undergo repeated testing, significantly reducing the cost of crashes. In recent years, reinforcement learning within machine learning has begun to demonstrate decision-making capabilities superior to humans in various gaming fields, which has led to its adoption in drone flight decision-making applications. Kal Backman et al. [6] proposed a method based on reinforcement learning to assist novice drone operators in landing drones. Their training environment was an AirSim simulation environment, and the method was tested in real-world scenarios, resulting in an increase in landing success rates from 51.4% to 98.2%. This demonstrates that as reinforcement learning technology continues to develop, its applications in the drone field will become even broader, providing stronger support for autonomous flight and intelligent decision making in drones. Research related to reinforcement learning in drone applications is as follows:

Mahmoud Abdelkader Bashery Abbass et al. [7] examined the advantages and disadvantages of various simulation environments, including renowned platforms such as Unity ML-Agents, PyBullet, and AirSim. They also created their own simulation environment to implement altitude control for drones using the Advantage Actor–Critic (A2C) algorithm.

Jin-Hyeok Park et al. [8] proposed a tracking technique developed from an object detection technology based on Recurrent Neural Networks (RNNs) and a Deep Q Network (DQN) control model. They integrated AirSim with the Virtual City Environment (VCE) simulation platform to collect target image data, ultimately achieving stable control for drones tracking vehicles and pedestrians.

In the realm of goods delivery, it is anticipated that drones will be widely utilized in the future. To enable drones to autonomously avoid obstacles during the delivery process, Guillem Muñoz et al. [9] tested three reinforcement learning models—DQN, DDQN (Deep Double Q Network), and Dueling DQN—in an AirSim simulation environment. The objective was to learn how to reach target locations within the simulation without colliding with any obstacles, with DDQN achieving the best performance among the models tested.

Drones also encounter moving obstacles during flight. Amudhini P. Kalidas et al. [10] compared three different reinforcement learning models—DQN, PPO (Proximal Policy Optimization), and SAC (Soft Actor–Critic)—in an AirSim simulation environment to assess

their ability to avoid both stationary and moving obstacles without human intervention. The training results indicated that SAC outperformed the other two algorithms, while PPO was the least successful among all the algorithms tested.

In the latest issue of *Nature*, Elia Kaufmann et al., from the Robotics and Perception Group at the University of Zurich [11], studied the autonomous drone system Swift. This system employs a deep reinforcement learning (DRL) algorithm to successfully defeat the current champions in first-person-view (FPV) drone racing, setting new competition records.

1.3. Synergy of Reinforcement Learning and Object Detection of This Study

Based on the aforementioned literature, it can be demonstrated that the application of the YOLO model in drone target identification has significantly increased in recent years, confirming its suitability in this field. Additionally, reinforcement learning has begun to show excellent performance in controlling drones for autonomous obstacle avoidance and path planning, both in simulated and real-world environments. However, most studies focus on comparing the training outcomes of different reinforcement learning models, with limited discussion on how to design appropriate reward and punishment mechanisms. In light of this, our main contributions are as follows:

1. Develop a training system in the AirSim simulation environment that uses 2D images as input and employs three reinforcement learning models—DQN, A2C, and PPO—to train drones for autonomous continuous flight through a target.
2. Discuss the design of reward and punishment methods alongside the training and testing outcomes of the reinforcement learning models, evaluate which model is most suitable for training autonomous drone flight, and summarize a set of appropriate methods for designing reward and punishment methods.
3. Based on the YOLOv7-tiny [12] object detection technology, propose two methods for unifying simulation and real-world state inputs, exploring the design methodologies for employing reinforcement learning to control autonomous drone flight in real-world environments.

2. Experimental Methods

This chapter provides an in-depth exploration of the theories employed in this study. The first section introduces YOLOv7 object detection and the integration with the TensorRT acceleration engine. The second section offers a comprehensive explanation of reinforcement learning (RL), covering its foundational theories as well as the principles, architectures, and training methods of RL models for the DQN, PPO, and A2C.

2.1. YOLOv7 Object Detection

The YOLOv7 neural network architecture builds upon the foundational YOLO structure, incorporating both Backbone and Head components. Its innovations lie in the optimization of the model architecture and the training process. First, it introduces the E-ELAN (Extended ELAN) architecture based on ELAN [13], enhancing network performance while maintaining optimal structure. Second, it employs a training methodology known as bag-of-freebies [12], which increases training costs without adding to inference costs; the techniques of re-parameterization and dynamic label assignment within this framework significantly improve recognition accuracy. Given the need for high recognition efficiency during drone flight, this paper selects the parameter-efficient YOLOv7-tiny as the object detection model.

During inference in object detection, images are input into the model utilizing pre-trained weights for recognition. Consequently, the speed of inference depends on the number of model parameters; to achieve real-time image recognition, it is essential to shorten the inference process to enhance the recognition speed. The TensorRT 10.6 developed and maintained by NVIDIA (Santa Clara, CA, USA) is specifically designed for NVIDIA GPUs and can significantly optimize neural network inference engines, seamlessly integrating with deep learning frameworks such as TensorFlow 2.18, PyTorch 2.5, ONNX 1.17.0, and Caffe. Among these, ONNX (Open Neural Network Exchange) serves as a

standardized format for describing machine learning models, facilitating conversion and deployment across various deep learning tools and frameworks. After streamlining the images through the YOLO with TensorRT, the developed system can integrate with the control system and deploy in real time for objection detection.

2.2. Basic Theory of Reinforcement Learning

Reinforcement learning is a goal-oriented learning method that seeks to maximize expected cumulative rewards through interaction with the environment, thereby identifying the optimal actions for the current state. The fundamental elements of reinforcement learning include agent, environment, action, state, reward, and policy. The mathematical model used to describe decision-making problems in stochastic environments is known as the "Markov Decision Process" (MDP), which is a crucial concept in reinforcement learning, representing the interaction process between the agent and the environment, as illustrated in Figure 1.

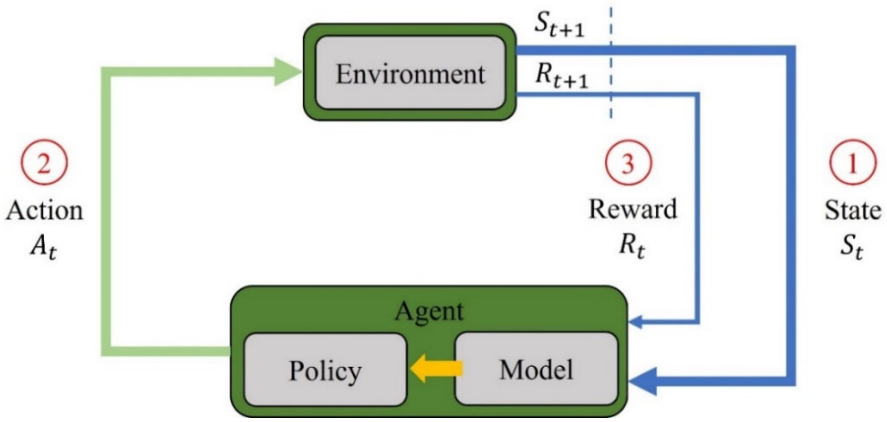


Figure 1. Markov Decision Process model.

At time point t, the environment state is denoted as S_t. The agent, based on its observation of the current state S_t, selects an action A_t from the action set through a policy function $\pi(a|s)$ in Equation (1). Subsequently, the environment, according to the state transition function $P(s'|s,a)$ in Equation (2), provides the next state S_{t+1}, and based on the reward function $R(s'|s,a)$ in Equation (3), it delivers the reward R_t. The agent then makes a new decision based on the updated state following its policy. This cycle of interaction is referred to as a time step, and the agent continues to interact until the episode concludes.

$$\pi(a|s) = P(S_t = s, A_t = a) \tag{1}$$

$$P(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a) \tag{2}$$

$$R(s'|s,a) = R(S_{t+1} = s'|S_t = s, A_t = a) \tag{3}$$

The value function is a key concept in reinforcement learning used to define the quality of learning. It can be divided into two types: the State–Action Value Function Q_π and the state value function V_π, as represented in Equations (4) and (5). The State–Action Value Function indicates the value of the agent performing a particular action in a given state, while the state value function reflects the value of the agent executing all the possible actions in the current state. The objective of reinforcement learning is to identify the optimal value function. Therefore, by evaluating the value defined by the value function, one can employ optimal policy methods to find the highest value. The value function can be reformulated through optimal policy methods in the optimal State–Action Value Function Q_* and the optimal state value function V_*, as shown in Equations (6) and (7).

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{4}$$

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{5}$$

$$Q_*(s,a) = \max Q_\pi(s) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} Q_*(s',a')\right] \tag{6}$$

$$V_*(s) = \max V_\pi(s) = \max_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma V_*(s')\right] \tag{7}$$

In this context, γ represents the discount factor, and $\mathbb{E}_\pi[X]$ is the expected value of the random variable X. The index k refers to the time step, indicating the k-th time in the future from the current time t.

This study will use DQN, A2C, and PPO as the reinforcement learning models for experimentation. The following section provides an introduction to these three models.

2.2.1. DQN (Deep Q Network)

Q-Learning, the predecessor of DQN, trains by continually updating the Q-table to find the optimal decision-making policy. The optimal policy method uses the Bellman Equation, which can maximize or minimize a certain performance indicator, to recursively compute the State–Action Value Function and obtain the optimal value function and policy. The corresponding expression is shown in Equation (8), where i denotes the iteration number and γ is the discount factor.

$$Q_{i+1}(s,a) = \mathbb{E}_\pi\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right] \tag{8}$$

However, when faced with large or continuous state spaces, storing and updating the Q-table becomes highly challenging and impractical. To address these issues, DQN [14] introduced three key techniques: Deep Learning Application, replay buffer, and target network. The algorithm flowchart is illustrated in Figure 2.

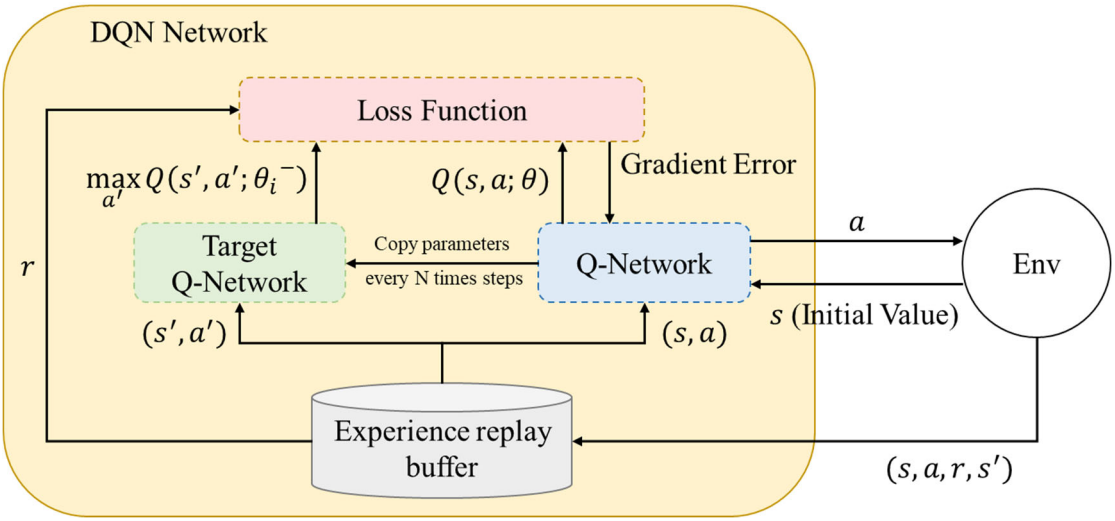


Figure 2. DQN algorithm flowchart [15].

(1) **Deep Learning Application:** DQN integrates the concept of Q-Learning with deep learning by replacing the Q-table with a deep neural network. The environmental state is used as the input to the neural network, and the Q-values for all the possible actions are the output. The Q-values are updated by minimizing the loss function, as shown in Equation (9), using stochastic gradient descent.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^{-}) - Q(s, a; \theta_i)\right)^2\right] \quad (9)$$

Here, θ_i represents the weights of the estimation network, θ_i^{-} denotes the weights of the target network, $U(D)$ is the sample drawn from the experience replay buffer, γ is the discount factor, and i indicates the iteration count.

(2) **Replay Buffer:** The stochastic gradient descent method requires the training dataset to be independent and identically distributed. However, in a simulated environment, data acquisition is continuous, and there are inherent relationships between each piece of data. Moreover, the randomness of exploration causes the data sample distributions generated by different strategies to become disordered and unpredictable. Therefore, DQN proposes the experience replay method, which designs a buffer of a certain size to replace experiences and randomly select data for training, thereby disrupting the continuity between data points.

(3) **Target Network:** Due to the high complexity of the environment, if the Q-value updates during the training process rely solely on the current estimated Q-values, it may result in large gradient fluctuations, leading to instability in the training process. Therefore, in 2015, Volodymyr Mnih et al. [15] proposed the design of two networks: one is the continuously trained estimation network, and the other is the fixed target network. After a certain number of training iterations (N), the parameters of the estimation network are synchronized to the target network. The loss function is calculated using the Q-values from both networks through gradient descent, preventing significant fluctuations in the Q-values during training, thereby contributing to the overall stability of the training process.

2.2.2. A2C (Advantage Actor–Critic)

The combination of the Policy-Based and Value-Based reinforcement learning algorithms is known as the Actor–Critic algorithm, with the notable A2C algorithm integrating the Actor–Critic framework with the Advantage Function $A(s, a)$ to enhance training efficiency and stability. The advantage network is a technique used to reduce the variance in policy gradients, representing how much better a specific action is compared to the average action in a given state, thereby providing a more accurate reflection of the action's quality. The advantage function is expressed as shown in Equation (10), where $Q(s, a)$ represents the action value function, and $V(s)$ represents the state value function.

$$A(s, a) = Q(s, a) - V(s) \quad (10)$$

The Actor–Critic framework uses the Critic network to approximate the state value function $V(s)$ for each observed action, which is then used to compute the policy gradient and update the Actor's policy network. This process reduces the probability of poor "advantage value actions" while increasing the probability of actions with higher advantage values. In other words, the goal of the Actor's policy network is to inform the agent what actions to take, while the Critic's objective is to evaluate the quality of the agent's actions.

The flowchart of the A2C algorithm is shown in Figure 3, and its specific steps are as follows [16]:

- Initialize the "network parameters θ" with random values.
- Use the current "policy π_θ" to explore the "environment" for N steps, storing (s, a, r, and s') in the replay buffer.
- If the episode ends, $R = 0$ or $V_\theta(s_t)$.
- Iterate $i = t - 1 \ldots t_{start}$

$$R \leftarrow r_i + \gamma R$$

Accumulate the "policy gradient" as shown in Equation (11).

$$\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta log\pi_\theta(a_i|s_i)(R - V_\theta(s_i)) \tag{11}$$

- Accumulate the "state value gradient" as shown in Equation (12).

$$\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial(R - V_\theta(s_i))^2}{\partial\theta_v} \tag{12}$$

- Update the network parameters using the accumulated gradients mentioned above, moving in the direction of $\partial\theta_\pi$ and the opposite direction of $\partial\theta_v$. The update process can be further stabilized by employing additional optimization methods, such as RMSProp (Root Mean Square Propagation), to improve training performance.
- Repeat from step (2) until convergence is achieved.

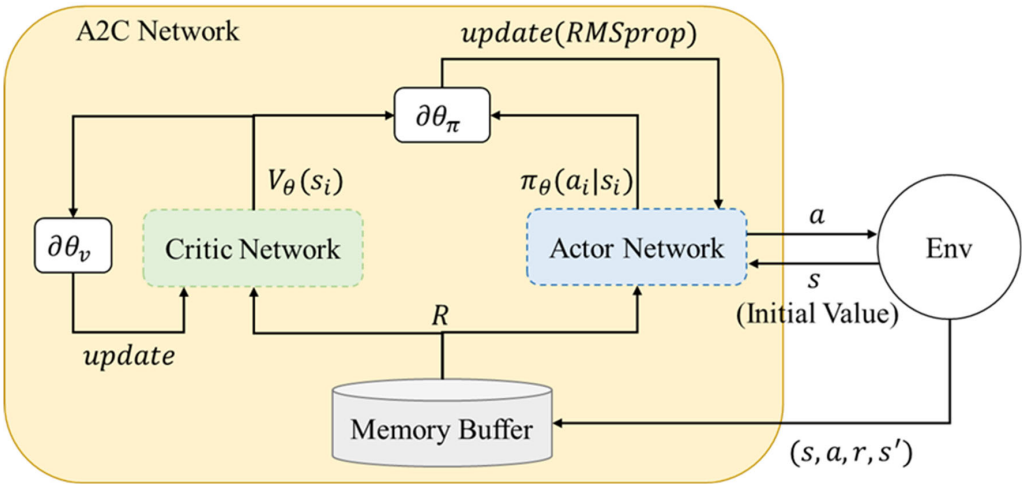


Figure 3. A2C algorithm flowchart [17].

2.2.3. PPO (Proximal Policy Optimization)

The advantages of Policy-Based reinforcement learning methods, compared to Value-Based approaches, lie in their ability to learn the policy itself directly, resulting in a faster learning process and improved decision-making performance for continuous actions in space. Among these methods, the Proximal Policy Optimization (PPO) algorithm, proposed by OpenAI and designated as a foundational test on their official website, is currently the most renowned Policy-Based reinforcement learning method. Inspired by Trust Region Policy Optimization (TRPO), PPO integrates three techniques: policy gradient, importance sampling, and Kullback–Leibler divergence (KL) [18], addressing the sampling efficiency issues inherent in basic policy gradient methods while avoiding the extensive computational demands of TRPO. The following section will outline the innovative aspects of the PPO algorithm [19].

- **Applying importance sampling to the policy gradient algorithm:** The most common policy gradient method parameterizes the policy π and fits it to π_θ, optimizing the parameters through a gradient ascent approach. The objective function $L(\theta)$ is expressed in Equation (13). Here, $\hat{A}_t$ represents the estimate of the advantage function, and $\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}$ denotes the expected value over a finite batch, with θ being the parameters of the policy network.

$$L(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\left[\pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{13}$$

Since the updated samples (s_t, a_t) required by the gradient algorithm must be generated by the current policy π_θ, using previous data to update the policy necessitates

aligning it with the distribution of samples generated by the new policy. Consequently, PPO employs importance sampling to introduce a weighting factor, reformulating the new objective function $L(\theta)$ as presented in Equation (14), where $\pi_\theta(a_t|s_t)$ represents the current policy and $\pi_{\theta'}(a_t|s_t)$ denotes the old policy.

$$L(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}\hat{A}_t\right] \tag{14}$$

This approach effectively constrains the ratio between the new and old policies, thereby limiting the size of the updates.

- **Introducing Kullback–Leibler divergence:** A limitation of importance sampling is that the numerator and denominator of the weighting factor should not differ significantly, as this can compromise stability. PPO addresses this issue by incorporating KL divergence to constrain the importance factor, effectively reducing computation and ensuring the stability of the algorithm. The specific formulation for $L_{ppo}(\theta)$ is presented in Equation (15), where β is the penalty coefficient and θ' represents the parameters of the old policy network.

$$L_{ppo}(\theta) = L(\theta) - \beta KL(\theta, \theta') \tag{15}$$

The main flowchart of the PPO algorithm is illustrated in Figure 4. PPO employs an Actor–Critic architecture, where the agent first acquires the state s_t from the environment and subsequently generates an action policy a based on this state, parameterizing the policy to fit $\pi_\theta(a_t|s_t)$. The environment then produces the next state s_{t+1} and the reward r_t in response to the action policy a. Each training iteration records $(s_t, a_t, r_t,$ and $s_{t+1})$ for the Actor–Critic network to compute the loss function. The Critic network is responsible for evaluating the value of the current policy by estimating the state value function $V_\mu(s)$ and updating the parameters of the Critic network through the loss function $L^V(\mu)$, where μ denotes the parameters of the value network. Stochastic gradient descent (SGD) is employed for this process, which cycles to update the policy iteratively.

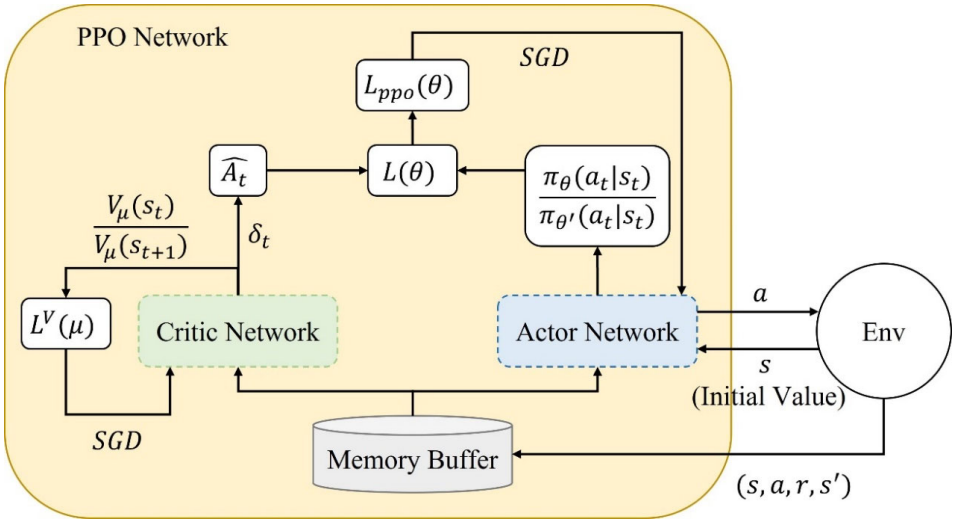


Figure 4. PPO algorithm flowchart [19].

3. Experimental Design

3.1. Simulation Environment Construction

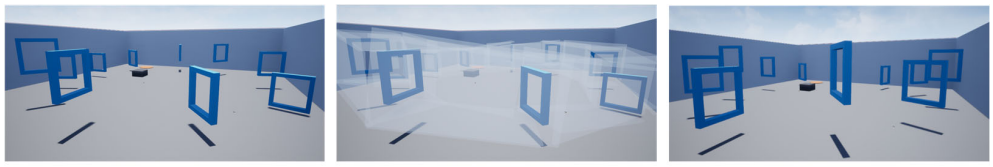
3.1.1. Drone Simulation

This study utilizes the AirSim (Redmond, WA 98052-6399, USA) quadcopter simulator, set up within a simulated environment built on Unreal Engine, to conduct experiments. The AirSim Python API (Version 1.8.1) is employed to retrieve drone flight control data and

onboard images by configuring the IP address. The flight control data include the drone's world coordinates in the virtual environment, flight commands (forward, backward, left, right, ascend, descend, rotate right, and rotate left), various motion speed control parameters, and collision detection information. The virtual drone's onboard image specifications are 360 × 240 × 3 pixels, with a field of view (FOV) of 82.6 degrees, providing colored image output.

3.1.2. Environment Simulation

This study utilizes version 4.27.2 of Unreal Engine to construct the simulated environment, which measures 100 × 100 square meters. Within this area, 11 square frames, each measuring 100 × 100 × 15 cubic centimeters, are positioned as targets for the drone to navigate through the square frame. The design of these targets is based on the targets used in the AirSim Drone Racing Lab [20] simulation competition, with their dimensions set to ten times the length and width of the AirSim simulated drone. The complete environment is illustrated in Figure 5a, with each target spaced approximately 5 m apart. To leverage the exploratory nature of reinforcement learning while preventing the drone from exploring unintended areas, invisible walls are established around the flight path. If the drone deviates from the flight path and collides with an invisible wall, it is classified as a failure, thereby reducing the collection of erroneous data. Figure 5b depicts the arrangement of the invisible walls as for the electronic fence in practice.



(a) (b) (c)

Figure 5. (**a**) Picture of the training simulation environment. (**b**) The arrangement of the invisible walls as the electronic fence. (**c**) Test simulation.

To assess whether the trained weights have successfully enabled the drone to navigate through the targets, this study designed an additional testing environment that is different from Figure 5a, as illustrated in Figure 5c. The size of the area and the number of targets are the same as in the training environment, but the positions of the targets differ. This setup allows for the evaluation of the same trained weights to determine if the drone can still successfully traverse the targets despite the altered placement.

A total of 11 tests will be conducted, with the drone generated at the center point of the corresponding target for each test. Subsequently, the weights will be utilized for flight testing, with the objective of continuously navigating through the subsequent 11 targets, effectively attempting to complete one full circuit of the area. If the drone successfully traverses all 11 targets or collides with the boundaries or any untraversed targets, resulting in a test failure, the current phase will be immediately halted, and the next test will commence. Throughout the testing process, data will be recorded on the number of targets the drone successfully traverses, the traversal rate, the number of collisions, the number of untraversed targets, and the number of successful circuits. These statistics will be compiled at the conclusion of each test phase.

3.2. Reinforcement Learning Settings

This study is based on the foundational settings established in the drone reinforcement learning example provided on the Stable Baselines3 [21] website. Adjustments were made according to the learning characteristics of the DQN, A2C, and PPO algorithms.

3.2.1. Agent, Target, State, and Action Design

This study aims to train a drone using reinforcement learning so that it can identify targets and traverse them in the shortest possible time, relying solely on the colored images

obtained from its onboard camera while maintaining flight through all 11 targets. Based on this objective, the agent is designated as the drone within the simulated environment, with the target being the traversable area of the targets. The design of the state, actions, and rewards is relatively complex, and will be explained sequentially.

First, the state is designed to align with the research objectives, utilizing only the first-person view (FPV) colored images as the input state. Figure 6 illustrates the state design for the scenario of drone flight with the first-person view when it is operated along the environment observed by the third-person view. However, due to the differences between the simulated and real-world environments, the state input through the first-person perspective shall vary, rendering the image data from the simulated environment unsuitable for real-world applications. Therefore, when the drone in the simulated environment should successfully complete its task using its perspective view, as shown in Figure 6, this study will employ YOLOv7-tiny to identify the targets as deployed in the first-person view. The recognition results will then be displayed on a blue screen measuring 360 × 240 pixels with a pink dot, as illustrated in the YOLO detection result in Figure 6. This allows for reinforcement learning to bridge the gap between the simulated and real-world environments.

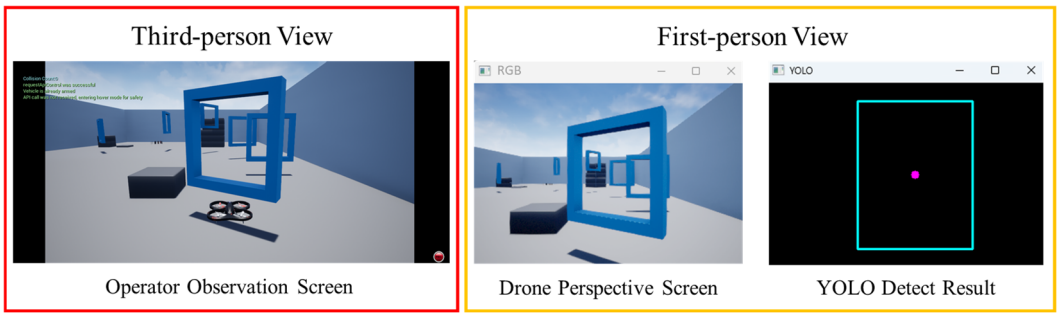


Figure 6. State design for the scenario of drone flight with the first-person view when it is operated along the environment with the third-person view.

This study employs a discrete action approach, designing nine types of drone flight maneuvers. These actions encompass forward movement, as illustrated in Figure 7, and are defined as follows:

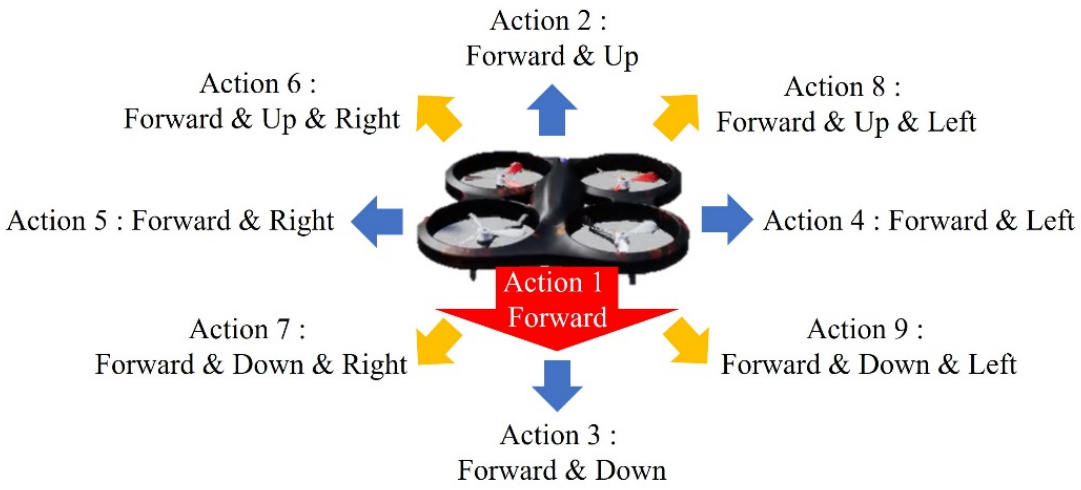


Figure 7. Action design.

3.2.2. Reinforcement Learning Parameters

The parameters utilized for training the DQN, A2C, and PPO agents will be based on the initial parameters provided on the Stable-baselines3 official website. Below, I will

enumerate three parameters that are common to at least two of the algorithms, as shown in Table 1, along with a brief explanation. Additional parameters for each algorithm can be referenced in the documentation on the Stable-baselines3 official website.

Table 1. DQN, A2C, and PPO parameter.

Reinforcement Learning Model	DQN	A2C	PPO
Learning Rate	0.00025	0.007	0.0003
Gamma	0.99	0.99	0.99
Seed	1	1	1
Total Time Steps	150,000	150,000	150,000
Batch Size	32	-	64
Policy Kwargs	None	None	RMSProp
N steps	-	5	2048
GAE Lambda	-	1.0	0.95
VF Coefficient	-	0.5	0.5

3.2.3. Hardware and Software System Specifications

This study involves the graphic rendering of the simulation environment, reinforcement learning, and training and testing with YOLOv7-tiny, all of which require extensive complex computations. Consequently, the hardware will employ GPU and CPU collaboration to expedite the training process while maintaining the quality of the experiments. The GPU utilized is the GeForce RTX 3060, and the CPU is the Intel(R) Core(TM) i7-8700.

In terms of software development, Python is employed as the programming language for the system, specifically version 3.10, leveraging its open-source libraries to implement various functionalities. Pytorch (Version 2.4.1) serves as the framework for image object detection and deep reinforcement learning; OpenCV is utilized for image preprocessing and display; ONNX and TensorRT are employed for converting and optimizing the trained weights; Gym [22] is used to construct the simulation environment and facilitate interaction with reinforcement learning; and Stable-baselines3 is utilized for coding various deep reinforcement learning algorithms. Figure 8 illustrates the relationship between the software and hardware utilized in this study.

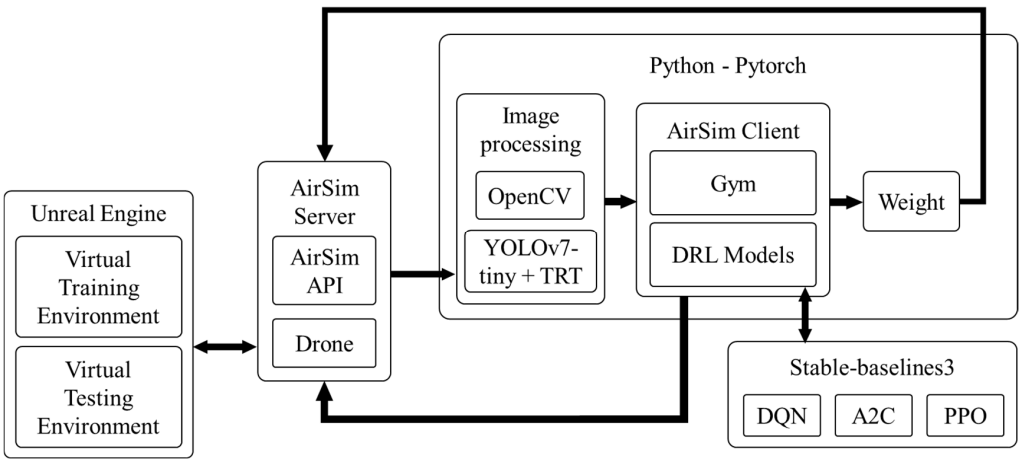


Figure 8. Software architecture integration diagram.

3.2.4. Experimental Simulation Processes

To achieve this study's research objective, designing a reward and penalty reinforcement learning (RL) flowchart system is depicted in Figure 9. The first step is to establish

communication between the simulation environment and the RL training program, which involves interactions between the agent and the environment. A reward and penalty mechanism is developed, followed by sequential training by algorithms using DQN, A2C, and PPO. The training results are evaluated to determine whether the average episode steps and average episode rewards increase with the number of training steps and converge to a stable value. If these evaluation criteria are met, a flight simulation test is conducted to analyze the effectiveness of the reward mechanism at this stage. The test has two objectives: first, to assess whether the target frame is successfully traversed, and second, whether the drone completes a full route of the field. In practice, the route dimension of the field should be confined. As an example, for drone safety or the prevention of GPS communication loss, the electronic fence should be assigned to the field no matter how the drone automation is conducted. So, considering the boundary condition for the route of the drone's flight, a field should be imposed and designed in the reward mechanism. If two conditions for successfully traversing the frame and a full route of the field are not completed, the reward mechanism is redesigned. Once the proposed reward mechanism enables the RL model to converge to the desired criterion and the two objectives (successful drone flight passing through the box frame and circle around the complete route of 11 box frames) are successfully achieved in the test environment, the finalized reward design method will be summarized based on the experimental results. Finally, the feasibility of practical application will be evaluated by integrating YOLOv7-tiny image recognition functionality with the developed reinforcement learning control.

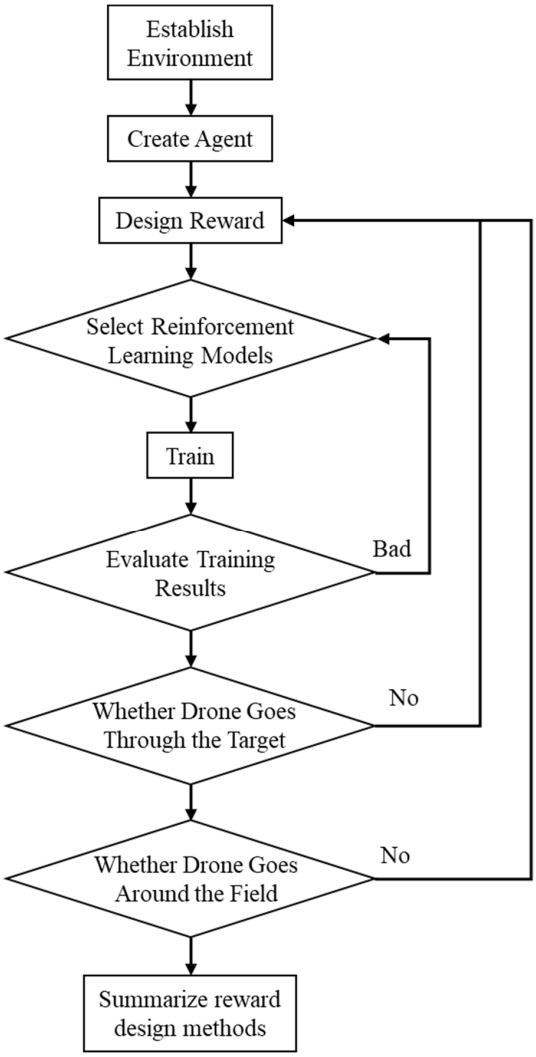


Figure 9. Flowchart for the reinforcement learning system design.

4. Experimental Results and Discussion

4.1. YOLOv7-Tiny Object Detection Experiment Results

The dataset required for this object detection research was captured by using an onboard camera from a simulated environment of the drone. In a virtual environment, a total of 2142 images with dimensions of 360 × 240 pixels were captured. After annotation with the LabelImg 1.8.6 software, the dataset was divided into training, validation, and test sets in a ratio of 80%, 10%, and 10%, respectively. The training and validation sets were fed into a desktop computer equipped with a GeForce RTX 3060 GPU for simulation. The training process was set to 300 epochs, with a batch size of 16. After the training process, the generated weight files were tested on both the validation and test sets, achieving an accuracy of over 90% for mAP@0.5. To further enhance object detection speed, the trained weights were optimized and converted using ONNX and TensorRT, improving recognition efficiency. The object detection system's accuracy (mAP) and efficiency (FPS) results are shown in Table 2.

Table 2. mAP and FPS testing results.

YOLOv7-Tiny		
Validation	mAP@0.5	0.996
	mAP@0.5:0.95	0.816
Test	mAP@0.5	0.995
	mAP@0.5:0.95	0.8
Original FPS		47.09
TensorRT FPS		91.7

4.2. Reinforcement Learning Reward Design Experimental Results

The success of reinforcement learning training is closely related to the design of rewards and punishments. Therefore, this study proposes three reward design methods including the random initialization method, random center method, and continuous round method that consider the drone's position and detail how to modify the boundary conditions and rewards for the experimentation. To bridge the gap between the simulated environment and the real world, this research introduces the YOLOv7-tiny object detection technology and employs two methods to synchronize the input states of both environments. This chapter will focus on the reward and punishment design methods, the various input states, and the experiments involving the different reinforcement learning algorithms.

4.2.1. Random Initialization Method (RIM)

The initial condition design for the RIM involves randomly placing the drone at the center of any target at the start of each training episode, facing the next target. The drone then begins its exploration training. The termination conditions are set as follows: the episode ends when the drone passes through the next target, collides with environmental obstacles, or incurs excessive time penalties. To determine whether the drone has crossed the target, this study records the exact center coordinates (x_t, y_t, z_t) of all the targets and sets a spherical region as the basis for determining whether the drone has passed through a target. A conceptual design diagram is shown in Figure 10, where the gray area represents the spherical region used to assess whether the drone has passed through the target.

As shown in Figure 10, during each training episode, the environment will continuously record the drone's current coordinates (x_d, y_d, z_d) and calculate the absolute distance D_{abs} between the drone and the center of the target. The formula for this calculation is provided in Equation (16).

$$D_{abs} = \sqrt{(x_d - x_t)^2 + (y_d - y_t)^2 + (z_d - z_t)^2} \tag{16}$$

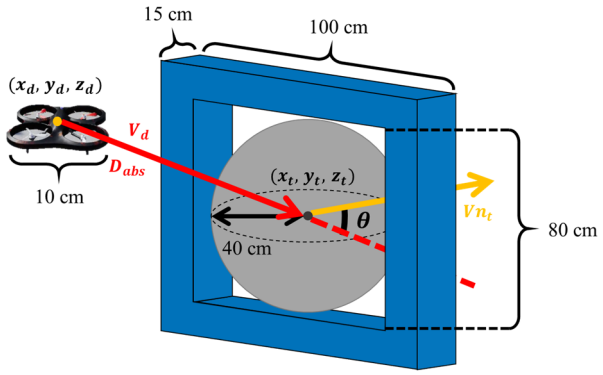


Figure 10. Target passing area conceptual design diagram with the absolute distance arrow for approaching the frame and the normal vector arrow for leaving the frame.

When the distance value is less than 40 cm, the environment will grant a reward of +100 and end the training episode. By subtracting the current D_{abs} from the distance at the previous time step, D_{abs_before}, the difference d is obtained, as shown in Equation (17). This difference allows the system to determine whether the drone's current action is moving it closer to or farther from the target. The environment uses d as the basis for reward or penalty, enabling the drone to learn to fly toward the center of the target by observing the score changes at each step.

$$d = \left(D_{abs} - D_{abs\_before}\right) \tag{17}$$

Using a spherical shape as the judgment area introduces a margin of error when determining whether the drone has passed through the target. When the drone crosses near the four corners of the target, there is a possibility of false negatives, where the drone is incorrectly judged as not having passed through the target. To address this issue, the RIM incorporates a time penalty term, T_m, into the training episode, as shown in Equation (18).

$$T_m = \text{Episode time} \tag{18}$$

As the duration of the episode increases, the penalty term will gradually grow. If the drone crosses the target but is not recognized as having done so, causing the training to continue, and the environment records that the current penalty term exceeds 0.2—indicating that the drone has made 40 decisions—the training episode will be forcibly terminated to prevent system training defects. Additionally, when the drone touches the target or an invisible obstacle, the environment will assign a penalty of −100. The boundary conditions and rewards for the RIM are shown in Table 3.

Table 3. Boundary conditions and reward of RIM.

Boundary Conditions	Rewards
1. Initial conditions: The drone is randomly generated at the center point of any target. 2. Ending conditions: a. The drone passes through the target. b. Hitting an environmental obstacle. c. The time deduction item is greater than 0.2.	1. Cross target: +100 2. Difference in distance from target center: $+d$ 3. Time deduction item: $-T_m$ 4. Hitting obstacles: −100

Using the RIM as the reward method, training was conducted with three reinforcement learning algorithms: DQN, A2C, and PPO. The mean episode length and mean reward graphs for the three training results are shown in Figure 11. From the results of the mean episode length (bottom of Figure 11), it can be observed that DQN, A2C, and PPO converged at 31.0, 33.4, and 32.1, respectively. Compared to A2C, both PPO and DQN exhibited less

fluctuation upon convergence, with PPO converging earlier than DQN. Specifically, PPO began showing an upward convergence trend around 20,000 time steps, while DQN did not display a similar trend until approximately 50,000 time steps.

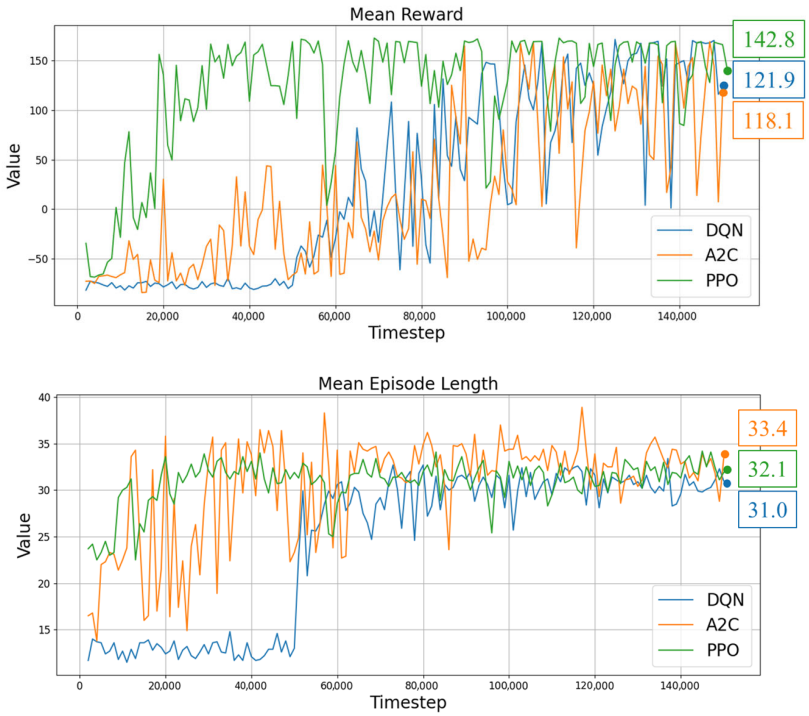


Figure 11. RIM mean episode length graph and mean reward graph.

Regarding the mean reward (top of Figure 11), DQN, A2C, and PPO converged at 121.9, 118.1, and 142.8, respectively. The degree of convergence fluctuation, from the smallest to the largest, is as follows: PPO, DQN, and A2C. Although all three methods demonstrated an upward trend in average episode reward scores, the amplitude of variation exceeded 50, indicating that the rewards obtained by the agent in each episode were unstable. This instability complicates the predictability and control of the model's performance, resulting in a decreased reliability of the overall training process.

The trained weights of DQN, A2C, and PPO were applied to the testing environment to evaluate their decision-making outcomes. The test results using the RIM are shown in Table 4. From the test results, it can be observed that the target traversal rates for all three reinforcement learning algorithms were below 25%, indicating that they failed to effectively pass through the target consecutively.

Table 4. DQN, A2C, and PPO test results with RIM.

RIM Test Results			
Items	**DQN**	**A2C**	**PPO**
Target traversal number (rate)	17 (14%)	17 (14%)	26 (21.4%)
Target collision number	4	9	4
Target untraversed number	7	2	7
Target circuit number	0	0	0

4.2.2. Random Center Method (RCM)

Due to the tendency of the random initialization method to result in the UAV colliding with the edges of the target, it is necessary to modify the reward for passing through the target by introducing a reward and punishment design related to the target's center.

Thus, the new RCM retains the initial and termination conditions, as well as most of the reward and penalty conditions, from the RIM. The only modification is the reward for passing through the target, which is now based on the drone's position relative to the center of the target. The closer the drone passes to the center, the higher the reward. The specific design process is as follows: When defining the spherical area for the target, in addition to recording the center point of each target, the normal vector Vn_t of the target is also recorded. When the drone touches the spherical area, the environment calculates the vector V_d from the drone to the center of the target by subtracting the drone's coordinates $(x_d,\ y_d,\ z_d)$ from the target center coordinates $(x_t,\ y_t,\ z_t)$, as shown in Equation (19). Then, using the dot product formula, the $\cos\theta$ between the vector V_d and the normal vector Vn_t is calculated, and the θ value is determined, as shown in Equation (20). From the θ value, it can be observed that the closer the drone passes to the center of the target, the smaller the θ value. Conversely, the closer the drone passes to the edge of the target, the larger the θ value. The environment uses this characteristic to assign a reward score ranging from 0 to +100. A conceptual design diagram is shown in Figure 10. The boundary conditions and rewards for the RCM are presented in Table 5.

$$V_d = (Vx_d,\ Vy_d,\ Vz_d) = (x_t - x_d,\ \ y_t - y_d,\ \ z_t - z_d) \tag{19}$$

$$\theta = \cos^{-1}\left(\frac{\vec{V_d} \cdot \vec{Vn_t}}{\left|\vec{V_d}\right| \times \left|\vec{V_{nt}}\right|}\right) \tag{20}$$

Table 5. Boundary conditions and reward conditions of RCM.

Boundary Conditions	Rewards
1. Initial condition: The drone is randomly generated at the center point of any target. 2. Ending conditions: a. The drone passes through the target. b. Hitting an environmental obstacle. c. The time deduction item is greater than 0.2.	1. Crossing target: Convert rewards through θ, ranging from 0 to +100. 2. Difference in distance from target center: $+d$ 3. Time deduction item: $-T_m$ 4. Hitting obstacles: -100

Using the RCM as the reward method, training was conducted with three reinforcement learning algorithms: DQN, A2C, and PPO. The mean episode length and mean reward graphs for the three training results are shown in Figure 12. From the plot of mean episode length (bottom of Figure 12), it can be observed that DQN converged at 30.6, while A2C failed to demonstrate effective improvement; PPO converged at 31. The PPO achieves convergence earlier than the DQN. PPO shows a higher gain convergence than the low gain convergence in DQN. In terms of the plot of mean reward (top of Figure 12), DQN converged at 170.5, A2C again did not show effective enhancement, and PPO converged earlier at 143.7 than DQN.

The training result graphs indicate that compared to the RIM, the RCM exhibited smaller fluctuations during training, suggesting that the average steps and rewards per episode became more stable. Although the mean episode length remained unchanged, the mean reward showed an increase of 48.6 for DQN and 0.9 for PPO. This demonstrates that the additional reward mechanism introduced by the RCM effectively enhanced the training performance.

The trained weights of DQN, A2C, and PPO were applied to the testing environment to evaluate their decision-making outcomes. The test results using the RCM are shown in Table 6. From the test results, it can be observed that the weights trained by A2C failed to control the drone to pass through the target, while the target traversal rates for DQN and PPO improved by 1.7% and 5.9%, respectively, compared to the results from the RIM.

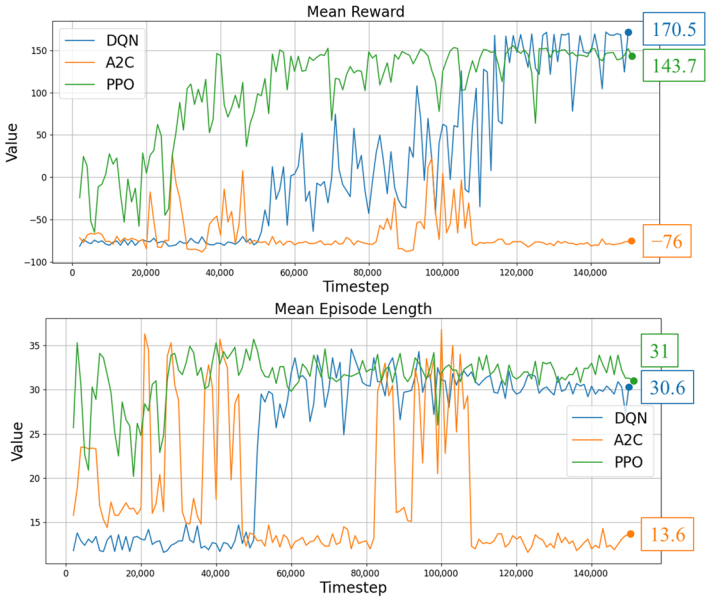


Figure 12. RCM mean episode length graph and mean reward graph.

Table 6. DQN, A2C, and PPO test results with RCM.

RCM Test Results			
Items	**DQN**	**A2C**	**PPO**
Target traversal number (rate)	19 (15.7%)	0 (0%)	33 (27.3%)
Target collision number	3	0	2
Target untraversed number	8	11	9
Target circuit number	0	0	0

4.2.3. Continuous Round Method (CRM)

In deep reinforcement learning algorithms, there must be a balance between the episode length and reward to ensure effective convergence towards the desired results. The CRM adjusts the termination conditions by limiting each training episode to the space between two targets while training the drone to consecutively pass through the target. The termination conditions for the CRM are the same as those in the RIM, but adjustments are made to the initial conditions. The drone is not regenerated at a random target at the start of every episode; instead, it is only regenerated in a random area if it collides with the environment's boundary or the edge of a target. If the drone successfully passes through a target, although the episode ends and restarts, the drone is not regenerated randomly but continues its exploration from its current position. A conceptual design diagram is shown and deployed in Figure 13. The boundary conditions and rewards for the CRM are presented in Table 7.

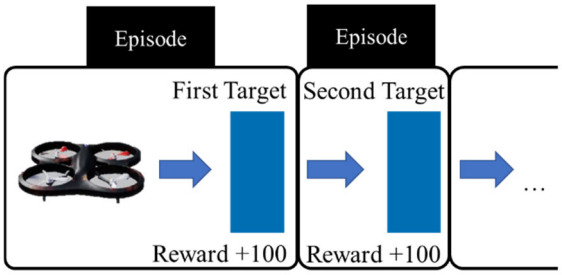


Figure 13. RCM target passing area conceptual design diagram.

Table 7. Boundary conditions and reward conditions of CRM.

Boundary Conditions	Rewards
1. Initial conditions: a. Randomly generated: Collision and time deduction items are too large. b. Continuous flight: Cross the target in the last round. 2. Ending conditions: a. The drone passes through the target. b. Hitting an environmental obstacle. c. The time deduction item is greater than 0.2.	1. Crossing target: Convert rewards through θ, ranging from 0 to +100. 2. Difference in distance from target center: $+d$ 3. Time deduction item: $-T_m$ 4. Hitting obstacles: -100

Using the CRM as the reward method, training was conducted with DQN, A2C, and PPO reinforcement learning algorithms. The mean episode length and mean reward graphs are shown in Figure 14.

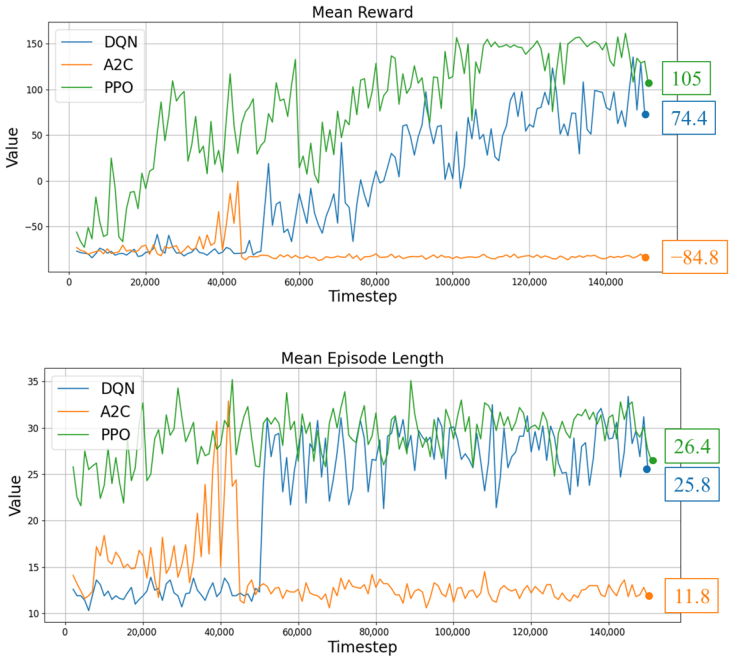


Figure 14. CRM mean episode length graph and mean reward graph.

It is evident that A2C still fails to effectively improve the values of both training results. From the mean episode length graph (bottom of Figure 14), DQN converged at 25.8 and PPO at 26.4, with PPO achieving convergence earlier than DQN. Compared to the training results of the RCM, these values decreased by 4.8 and 4.6, respectively, indicating that the CRM can complete the traversal of the target frame with fewer steps, thereby enhancing flight efficiency.

In the mean reward graph (top of Figure 14), it can be observed that PPO began to improve its values from the early stages of training, ultimately converging at 105, whereas DQN did not start to show improvement until 52,000 time steps, converging finally at 74.4. This indicates that both PPO and DQN can effectively train the drone to traverse the target frame, but PPO achieves better reward scores with a similar number of steps. Furthermore, from the value changes observed in Figure 14, the differences in the values are approximately 10 and 50, respectively, demonstrating better convergence stability compared to the RCM.

The trained weights of DQN, A2C, and PPO were applied to the testing environment to evaluate their decision-making outcomes. The test results using the CRM are shown in Table 8. It can be observed that A2C, due to its inability to improve scores during training, still failed to effectively pass through the target. In contrast, DQN and PPO both demonstrated improved target traversal rates, with one and five full-field flights, respectively. Therefore, this study selects the CRM as the final reward design method and continues using the stable and converging DQN and PPO algorithms for the subsequent experiments involving changes to state inputs.

Table 8. DQN, A2C, and PPO test results with CRM.

CRM Test Results			
Items	**DQN**	**A2C**	**PPO**
Target traversal number (rate)	48 (39.7%)	0 (0%)	86 (71.1%)
Target collision number	1	0	3
Target untraversed number	9	11	3
Target circuit number	1	0	5

4.2.4. Maximum Single Target Method (MSTM)

To apply the simulation results to a real-world environment, it is essential to unify the drone's image reception in both the simulated and physical environments. This study proposes using YOLOv7-tiny combined with TensorRT object detection technology to recognize targets in both environments, and then mapping the detection results onto a black screen. This approach unifies the image input for both the simulation and real-world environments. In the simulation training, the target the drone is expected to pass through is always the largest target in the image. Therefore, the MSTM involves using object detection technology to identify the largest target, which is then rendered on a black screen and used as the state input for training. The state design for the MSTM is illustrated in Figure 15.

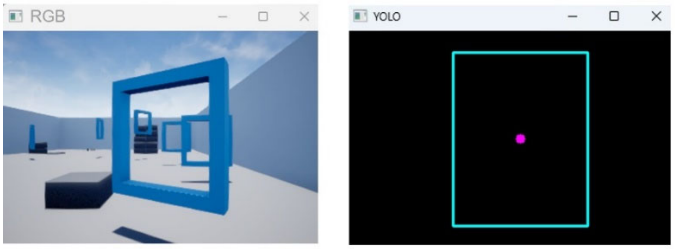


Figure 15. State design for the MSTM.

Using the CRM as the reward method and the MSTM as the state input, the DQN and PPO reinforcement learning algorithms were trained. The mean episode length and mean reward graphs are shown in Figure 16. In the mean episode length graph, DQN converged at 26.3 and PPO at 26.5, while in the mean reward graph, DQN converged at 39.8 and PPO at 76. In comparison to the CRM training results during the simulation phase, these values represent decreases of 34.6 and 29, respectively.

From the two training result graphs, it can be observed that PPO began to improve its values early in the training process, whereas DQN did not start to show improvement until 51,000 time steps. Although PPO exhibited a downward trend in mean reward during the later stages of training, when comparing the final convergence scores, PPO still demonstrated superior training performance relative to DQN.

The trained weights of DQN and PPO were applied to the testing environment to evaluate their decision-making outcomes. The test results using the MSTM combined with the CRM are shown in Table 9. Due to the state input being less rich compared to the

previous simulation phase, the target traversal rates for both DQN and PPO decreased compared to the best results from the CRM alone.

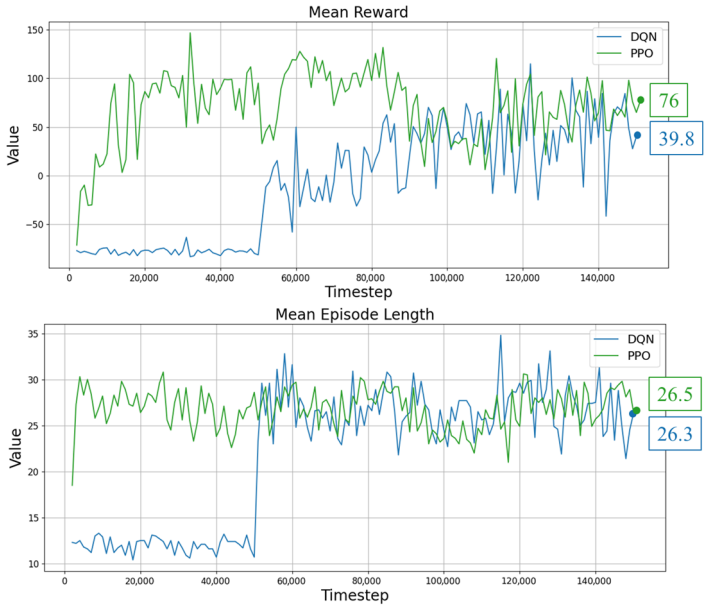


Figure 16. MSTM mean episode length graph and mean reward graph.

Table 9. DQN and PPO test results with MSTM + CRM.

MSTM + CRM Test Results		
Items	**DQN**	**PPO**
Target traversal number (rate)	27 (22.3%)	47 (38.8%)
Target collision number	3	1
Target untraversed number	8	9
Target circuit number	0	1

4.2.5. Maximum Two Target Method (MTTM)

Since the primary and subsequent targets for the drone will be the two largest targets in the frame, this study introduces the MTTM. In this method, the detection results displayed on the black screen are modified to include the two largest targets, with different colors distinguishing their sizes. The largest target is marked in blue, and the second largest in pink. If there is only one target in the frame, it will be displayed in blue. This ensures that the input state for training includes information on the next two targets. A conceptual design diagram for the MSTM is shown in Figure 17.

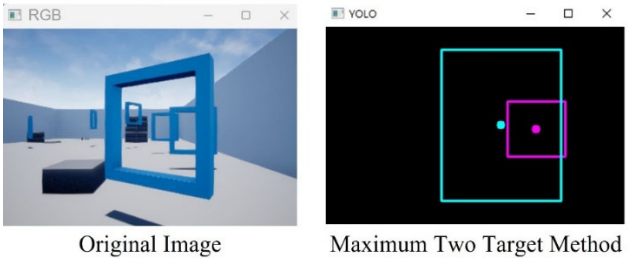


Figure 17. State design for the MTTM.

Using the CRM as the reward method and the MTTM as the state input, the DQN and PPO reinforcement learning algorithms were trained. The mean episode length and mean

reward graphs are shown in Figure 18. In the mean episode length graph, DQN converged at 26.1 and PPO at 28.1. In the mean reward graph, DQN converged at −19.5, revealing a fluctuation of over 50 in the later stages of training, which indicates instability during DQN training. In contrast, PPO converged at 106.1, demonstrating an improvement of 30.1 compared to using the MSTM as the state input. Moreover, the difference from the CRM training results in the simulation phase was only 1.1, indicating that PPO, when trained using the MTTM as the state input, can achieve performance comparable to that of the CRM during the simulation phase.

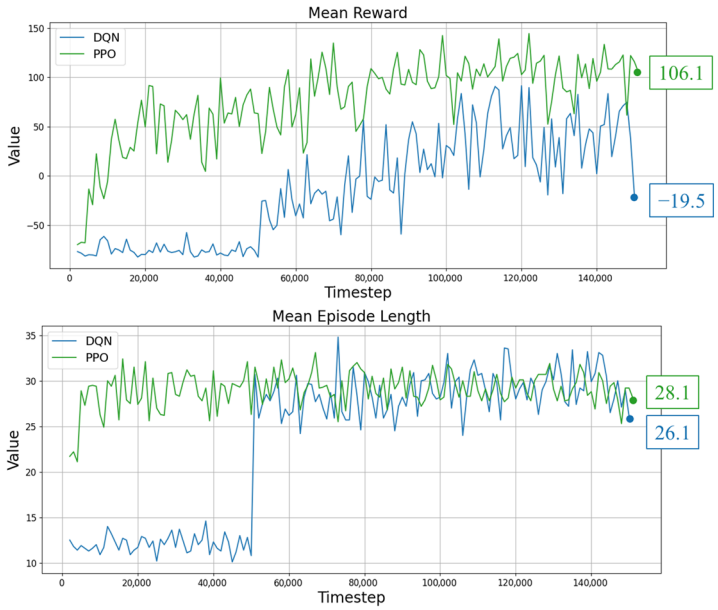


Figure 18. MTTM mean episode length graph and mean reward graph.

Therefore, it can be inferred that PPO is more suitable than DQN for reinforcement learning training algorithms designed with the MTTM as the state input and CRM as the reward mechanism.

The trained weights of DQN and PPO were applied to the testing environment to evaluate their decision-making outcomes. The test results using the MTTM combined with the continuous episode method are shown in Table 10. In the testing results, PPO achieved a target traversal rate of 52.1% and completed two full-field flights. The PPO preserved the high gain control phenomena both in the traversal and circuit numbers. The leading look-ahead ability can provide the drone with agile motion during the flight and hold stable learning dynamics during the flight mission. Although this performance is not as high as the results from the CRM during the simulation phase, it still demonstrates a success rate exceeding 50%. This indicates that using YOLOv7-tiny object detection to unify the state inputs of both the simulation and physical environments is a feasible approach for training drones to pass through targets in reinforcement learning contexts.

Table 10. DQN and PPO test results with MTTM + CRM.

MTTM + CRM Test Results		
Items	**DQN**	**PPO**
Target traversal number (rate)	10 (8.2%)	63 (52.1%)
Target collision number	3	6
Target untraversed number	8	3
Target circuit number	0	2

5. Discussions

This study summarizes three reward design methods and the testing results of various reinforcement learning models, with trajectory recordings illustrated in Figure 19. It is evident that models trained using the A2C algorithm, regardless of the reward design method employed, are unable to effectively control the UAV's continuous target cruising. This outcome is closely related to the training process, as observed from the training results with A2C across each reward and punishment method, shown in Figures 11, 12 and 14, which indicate that A2C fails to effectively acquire reward scores. In contrast, when employing the DQN and PPO algorithms in conjunction with the CRM for training, the results displayed in Figure 14 reveal that PPO achieves a higher mean reward under nearly identical mean episode lengths. Moreover, the flight test results depicted in Figure 19 demonstrate that the flight path chosen by PPO is more inward compared to DQN, and it approaches the center point more closely when passing through the target frame. Therefore, it can be inferred that PPO, when combined with the CRM as a reward and punishment training method, exhibits superior altitude control stability and enables faster and safer target traversal compared to DQN. Consequently, it can be concluded that PPO combined with CRM is the most suitable reinforcement learning control method identified in this study.

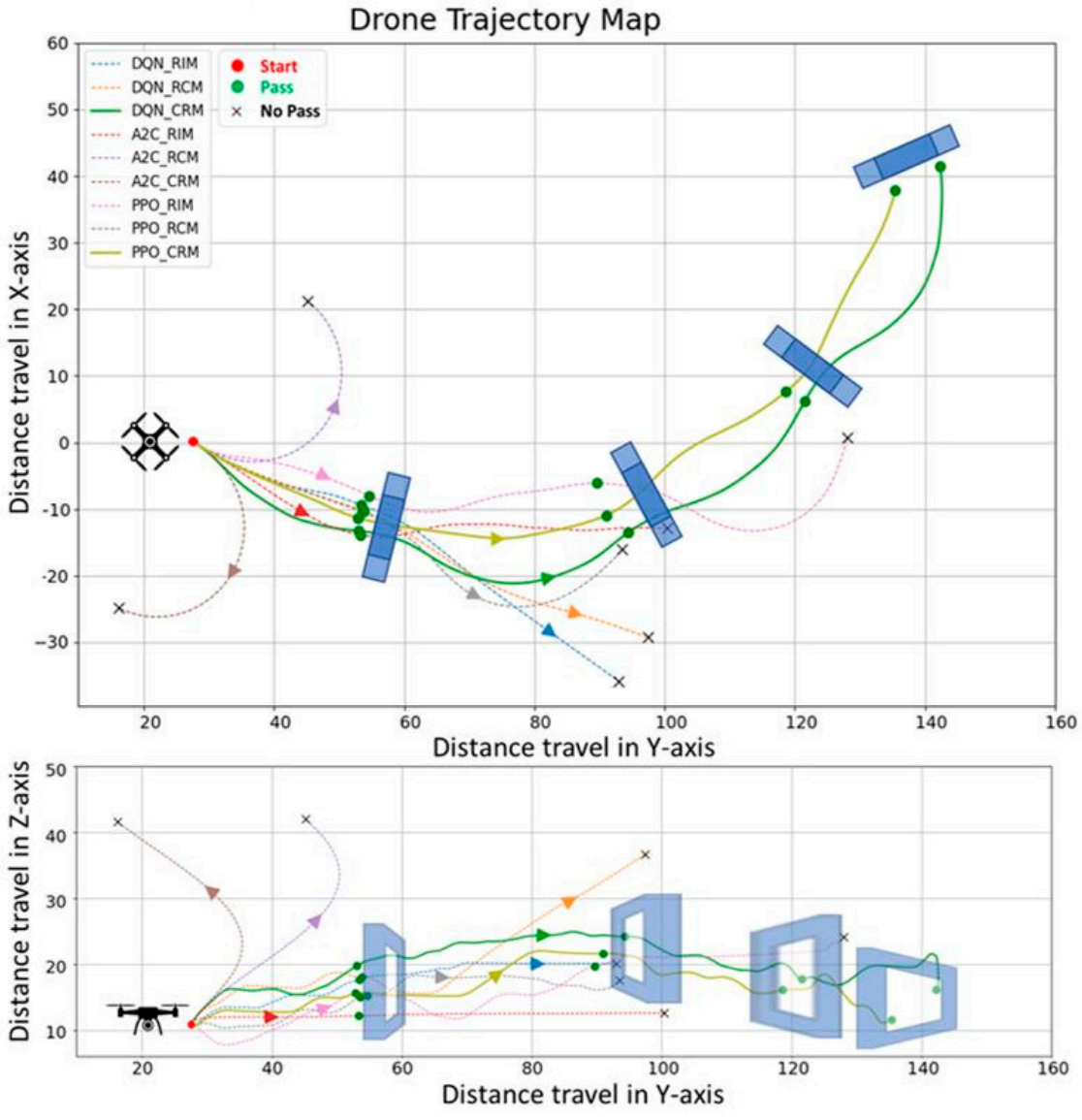


Figure 19. Drone trajectory records.

Based on the experimental process of this study, which progresses from the design of the RIM to RCM and ultimately to CRM while considering the practical OD's applications in UAV flight, we can draw the following conclusions: Developing suitable reward methods requires the sequential consideration of four aspects. First, establish reward and punishment scores for the primary objective, which will be awarded only once per episode and will carry a significant value. Second, design a supplementary reward and punishment method, wherein the agent receives continuous minor rewards or punishments based on its proximity to the target during each action. Third, set boundary conditions according to the context, restricting the number of steps per episode within a defined range during the agent's continuous interactions. Finally, design state inputs that align with real-world situations. The design process can be summarized as shown in Figure 20.

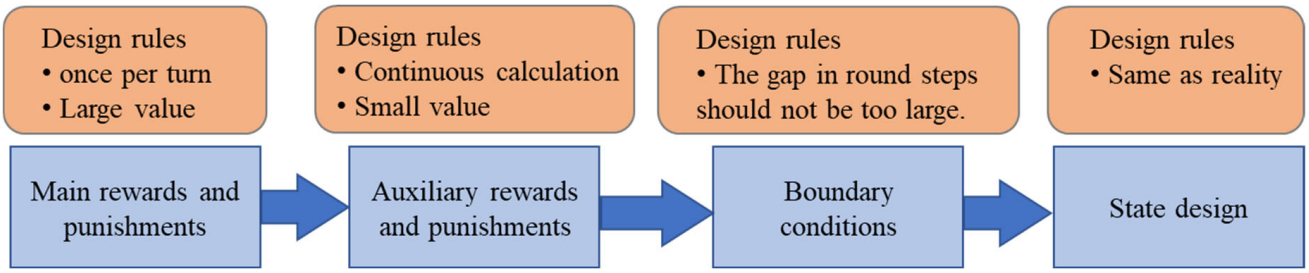


Figure 20. Reward method design process.

6. Conclusions and Future Work

6.1. Conclusions

This study established a simulated environment using AirSim and Unreal Engine to replicate the scenario of a drone traversing a target within the environment. Through reinforcement learning control, the functionality of the drone achieving automatic continuous traversal of multiple targets was realized. The experimental results can be viewed at the following video link: https://youtu.be/rsS0WlDHrlQ (26 September 2024). The contributions of this study are as follows:

- A simulated environment for drone flight was established, encompassing both the drone and the targets. This training environment supports three reinforcement learning algorithms (DQN, A2C, and PPO) for training the drone's decision-making flight maneuvers, facilitating reinforcement learning control for the task of traversing the target.
- The PPO algorithm combined with the CRM emerged as the most suitable reinforcement learning algorithm and reward method for this study's tasks. In contrast, A2C proved unsuitable for training within this research environment due to its instability. The training results effectively converged the mean episode length to 27, with a target traversal rate reaching 71%, successfully completing five full laps of the designated area.
- The YOLOv7-tiny object detection technology was employed to link, apply, and deploy the reinforcement learning state inputs for both the simulation and physical environments, examining the effects of two different state inputs on the drone's action decision making. The results indicated that the MTTM, utilized as a state input, achieved a target traversal rate of 52%, which represented a 13% increase in success rate compared to the MSTM, making it more suitable as the state input for a reinforcement learning decision system in physical environments.

6.2. Future Outlook

- Optimize the foundational setup of the reinforcement learning training system to provide a more comprehensive training experience, which includes enhancing state reading speed, expanding action spaces, and optimizing boundary designs.

- Investigate a broader array of reward methods, and design diverse training and testing environments to improve the success rate of the drone in traversing targets and its generalization capabilities in different contexts.
- Increase flight experiments in real-world environments to assess the dynamic performance of the drone in authentic settings, thereby addressing discrepancies between the simulated and physical environments.
- Given that the drone is unable to carry heavy laptops and can only accommodate lightweight edge computing devices, future efforts will focus on integrating reinforcement learning with object detection technologies for deployment on edge computing devices.

Author Contributions: Conceptualization and methodology, Y.-C.H. and S.-E.S.; software, S.-E.S.; validation, S.-E.S.; investigation, Y.-C.H.; resources, Y.-C.H.; data curation, S.-E.S.; writing—original draft preparation, S.-E.S.; writing—review and editing, Y.-C.H.; supervision, Y.-C.H.; project administration, Y.-C.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Available under request.

Conflicts of Interest: No conflicts of interest.

References

1. 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, Las Vegas, NV, USA, 27–30 June 2016.
2. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot Multibox Detector. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016.
3. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014.
4. Zhang, Z. Drone-YOLO: An Efficient Neural Network Method for Target Detection in Drone Images. *Drones* **2023**, *7*, 526. [CrossRef]
5. Yongshuai, L.; Yuan, H.; Wang, Y.; Xiao, C. GGT-YOLO: A Novel Object Detection Algorithm for Drone-Based Maritime Cruising. *Drones* **2022**, *6*, 335. [CrossRef]
6. Backman, K.; Kulić, D.; Chung, H. Reinforcement Learning for Shared Autonomy Drone Landings. *Auton. Robot.* **2023**, *47*, 1419–1438. [CrossRef]
7. Abbass, M.A.B.; Kang, H.S. Drone Elevation Control Based on Python-Unity. Integrated Framework for Reinforcement Learning Applications. *Drones* **2023**, *7*, 225. [CrossRef]
8. 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. [CrossRef]
9. Muñoz, G.; Barrado, C.; Çetin, E.; Salami, E. Deep Reinforcement Learning for Drone Delivery. *Drones* **2019**, *3*, 72. [CrossRef]
10. Kalidas, A.P.; Joshua, C.J.; Md, A.Q.; Basheer, S.; Mohan, S.; Sakri, S. Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. *Drones* **2023**, *7*, 245. [CrossRef]
11. Kaufmann, E.; Bauersfeld, L.; Loquercio, A.; Müller, M.; Koltun, V.; Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature* **2023**, *620*, 982–987. [CrossRef] [PubMed]
12. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 7464–7475. [CrossRef]
13. Wang, C.Y.; Liao, H.Y.M.; Yeh, I.H. Designing Network Design Strategies Through Gradient Path Analysis. *J. Inf. Sci. Eng.* **2022**, *39*, 975–995.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602.
15. Mnih, V.; Kavukcuoglu, 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]
16. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *Proc. Mach. Learn. Res.* **2016**, *48*, 1928–1937. Available online: https://proceedings.mlr.press/v48/mniha16.html (accessed on 19 June 2016).
17. Wang, J.X.; Kurth-Nelson, Z.; Kumaran, D.; Tirumala, D.; Soyer, H.; Leibo, J.Z.; Hassabis, D.; Botvinick, M. Prefrontal cortex as a meta-reinforcement learning system. *Nat. Neurosci.* **2018**, *21*, 860–868. [CrossRef] [PubMed]
18. Kullback, S.; Leibler, R.A. On information and sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86. [CrossRef]
19. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.

20. Madaan, R.; Gyde, N.; Vemprala, S.; Brown, M.; Nagami, K.; Taubner, T.; Cristofalo, E.; Scaramuzza, D.; Schwager, M.; Kapoor, A. AirSim Drone Racing Lab. *arXiv* **2020**, arXiv:2003.05654.
21. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *J. Mach. Learn. Res.* **2021**, *22*, 1–8. Available online: http://jmlr.org/papers/v22/20-1364.html (accessed on 11 November 2021).
22. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. Openai gym. *arXiv* **2016**, arXiv:1606.01540.