School of Computer Science and Statistics

Connectivity-Aware Quadcopter UAV trajectory design using the Actor Critic Reinforcement Learning approach

Xiangyu Zheng

August 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Computer Science Data Science

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism ‘Ready Steady Write’, located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Xiangyu Zheng Date:August 19, 2022

Abstract

The emergence of unmanned aerial vehicles (UAVs) has provided convenience for people's lives and can also help people working in cities with tasks such as traffic monitoring, multi-target tracking, express delivery, and so on. If the UAV loses signal at work, it will get out of control. Therefore, in order to ensure the working quality of the UAV, optimizing the trajectory of the UAV to ensure the connection quality has become a feasible way.

In recent years, various techniques have been continuously used for trajectory optimization problems, among which graph, dynamic programming, reinforcement learning (RL), Deep-Q-network (DQN), etc. are widely used. However, when these technologies optimize the UAV trajectory, the actions of the UAV need to be preset and fixed. If the action range is larger, the granularity of the trajectory will be larger, and some better flight points may be missed. If the range of motion is small or there are many actions, it will cause computational performance problems. A feasible way is to use the continuous output Advantage Actor Critic (A2C) algorithm, which means that the UAV has a richer action space. It is improved from the Actor-Critic (AC) algorithm and solves the problem of excessive variance of the AC algorithm. We created a simulation environment and compared the performance of the DQN and A2C algorithms on the connectivity-aware UAV path planning problem. The results show that the A2C algorithm is slightly better than the DQN algorithm.

Acknowledgements

Thank you to my family for supporting me living and studying in Ireland.

Thanks to Ivana and my supervisor Boris for their help.

Contents

List of Figures

List of Tables

1 Introduction

This work attempts to optimise Unmanned Aerial Vehicles(UAV) trajectories using the advantage actor critic (A2C) algorithm, with the aim of obtaining better connectivity for UAV on missions in cities. Each UAV has its own agent, which allows the UAV agent to deal more appropriately with the complex and variable variables in urban environments. For example, in the face of building obstacles and limited base station signal coverage. We evaluate the performance and trajectory of UAV under the A2C model and then compare it with the Deep-Q-network (DQN) model. This chapter focuses on the problem we address, the objectives, the assumptions, the contributions, and finally, the structure of the thesis.

1.1 Problem Statement

In recent times, UAVs have played an increasing role in the urban environment, involving the use of various aspects. Examples include the roll-out of dynamic target and multi-target tracking in urban environments[1][2]. monitoring under a wide variety of complex traffic[3]. [4] The use of UAVs for modern photography replaces traditional manual detection of flawed road markings to improve efficiency. The use of UAVs to replace manual inspection of transmission towers for the safety of personnel[5]. More and more researchers are beginning to look at the use of UAVs in smart cities, which seems to indicate a wide range of uses[6][7][8].
These UAVs flying in urban environments have had to face connectivity issues in their missions. In [9] [10], it is pointed out that the Global Navigation Satellite System (GNSS) is not always reliable for navigating in the urban environment of UAVs, while the connectivity base station(BS) can assist the UAVs in their flight. Connectivity for UAVs has therefore become an issue worth investigating, as it is a guarantee that the UAV will perform its job well. In addition, both UAV operators and customers can benefit from this. This work will be devoted to the problem of trajectory optimization using the A2C algorithm for perceptual connectivity.

1.2 Thesis Aims and Objectives

The aim of this work is to optimize connectivity-aware optimized trajectories for UAVs in cities based on A2C while considering various complex and realistic urban environments in the model. This can provide UAV operators and customers with a solution to improve the quality of UAVs performing their tasks. DQN has been used extensively for path optimisation problems in various UAVs with successful results[11][12][13]. This is because it uses a combination of deep neural networks and Q-learning to ensure that the UAV's state and action space are optimised. However, both the action space and the values need to be pre-set, and the values are fixed. This work explores the impact of the continuous output property of the A2C algorithm on trajectory optimisation, which means that by setting the same range of action spaces as the discrete algorithm, more actions can be obtained, potentially leading to a better trajectory. We obtained flight data from the UAV simulator and evaluated it.

1.3 Thesis Contribution

In this work, we attempt to use A2C to optimise connectivity-aware UAV trajectory optimisation problems and compare them with discrete output algorithms such as DQN to reveal the differences in path optimisation and performance.
We also create a UAV simulation environment using gym-pybullet-drone to maximise the complexity of the urban environment and to provide an evaluation of candidate algorithms.

1.4 Document Structure

Chapter 2 introduces the background of reinforcement learning, Q- learning algorithms, neural networks, DQN, and A2C-related techniques. Additional work on UAVs in areas such as trajectory optimization, optimizing connectivity, and background on the use of UAVs in deep reinforcement learning is also presented.

Chapter 3 presents the design of the UAV simulation environment model (including building obstruction and base station signals). The algorithm's reward values, action space, and state space are designed, as well as the pseudo-code for the DQN and A2C algorithms in this experiment.

Chapter 4 describes how to use gym-pybullet-drone to implement the design of Chapter 3, as well as the implementation of other functions and some major functions.

Chapter 5 presents the results of the evaluation of the UAV flight trajectory in the simulation model.

Chapter 6 summarizes the whole thesis and proposes future work.

2 background and Related Work

This chapter introduces the technical concepts of reinforcement learning and deep learning in solving various problems of UAV, such as optimization trajectories and techniques used in optimizing connectivity, and introduces related work. The first part introduces the concept of reinforcement learning and the Q-learning algorithm. The second part introduces deep learning and optimization algorithms. The third part describes the deep reinforcement learning approach. The last part introduces the related work of UAV trajectory optimization problem and optimization connectivity problem and the application of deep reinforcement learning in these fields.

2.1 Reinforcement Learning

Reinforcement learning (RL) uses a trial-and-error approach to allow agents to learn. It is inspired by behaviourist psychology and focused on online learning, which will balance exploration and exploitation. It differs from traditional supervised learning in machine learning. RL does not require any prior information or data to be given, but rather rewards are obtained by interacting with the environment to motivate behaviour. The ultimate goal is to maximize the agent's reward. This is model-free RL, which will be discussed next.

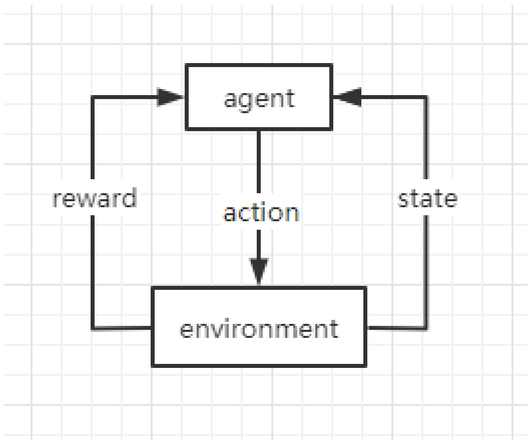


Figure 2.1: Reinforcement Learning

2.1.1 components of reinforcement learning

As shown in Figure 2.1, this is how basic RL works, with the following main elements:

agent

The agent exists as an object, which can be an algorithm or a neural network. It continuously obtains the state from the environment, and returns the policy to the environment according to the observed state. In this scenario, each UAV will have an agent.

environment

An environment is a scene being studied and it contains all the elements that we want to study. For example, the environment of this experiment is a small city model. The environment will continuously receive policies or actions returned from the agent, and based on these feedbacks, it will return to the state and reward the agent.

action

The action plays an integral role between the environment and the agent. It controls the agent and affects reward values and states. Actions can be set depending on the experiment, for example in this scenario the action is the UAV flying in a different direction.

state

State is an important part of reinforcement learning for describing the environment that we care about at a particular time step. The agent is constantly acquiring state from the environment. Thus state influences the intelligence to make decisions.

reward

Reward is arguably the most important aspect of reinforcement learning, as it influences the short- or long-term outcome of the strategies adopted by the intelligence. The environment returns a reward to the agent when the agent observes the state at the next time step, which can be positive or negative, representing either a good or bad action taken. Usually our goal is to optimize this maximum reward value in order to obtain the best policy.

2.1.2 Markov Decision Process model

A common model for RL is the standard Markov Decision Process(MDP), which expresses RL in a complex mathematical form. It can be defined as a five-tuple (λ, A, S, P, R) in discrete time steps t, where:

- λ is the discount factor, where $\lambda \in$ [0,1].
- A represents the set of actions, which contains the actions taken at all time steps, $a_t \in$ A represents the actions taken at time step t.
- S represents the state set, it contains the state at all time steps. $s_t \in$ S represents the state at time t.
- P represents the state transition probability matrix and P(s_{t+1}=s_t|s_t,a_t) denotes the observed probability of transferring state s_t to state s_{t+1} and taking action a_t at time t
- R is the reward function, which is R_t = R(s_t, a_t).

In summary, at each time t, the agent receives the state s_t of the environment and performs the current best action a, passes it to the environment for execution, and then observes the reward R_t. the agent then moves to the new state s_{t+1} based on the state transfer equation P(s_{t+1}=s_t|s_t,a_t). The agent's goal is to optimize long-term total return R, which can be shown as:

$$R_{t+1} = \sum_{t=0}^{\infty} \lambda R_t \tag{1}$$

In which we include the discount factor λ in the calculation in order to prevent unlimited expansion of the reward.

When the agent receives a certain state s_t, it will choose the next action a_t to execute with a certain policy π and then transition to the new state s_{t+1}, This action selection process is called policy and each policy is a state to action mapping function π:S->A. Bellman's equation defines the sum of value functions as follows:

$$V^{\pi}(s) = R(s) + \lambda \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s') \tag{2}$$

The agent needs to find the optimal action policy in the whole range of environments π for state s_{t+1} at the current time t. The optimal function V^* is expressed as follows:

$$V^*(s) = R(s) + \max_{a \in A} \lambda \sum_{s' \in S} P_{sa(s)}(s') V^*(s') \tag{3}$$

Similarly, the optimal policy function π^* S->A is as follow:

$$\pi^*(s) = arg\max_{a \in A} \lambda \sum_{s' \in S} P_{sa(s)}(s') V^*(s') \tag{4}$$

If the agent can obtain the optimal action in each state, then it holds from π^*, Thus the action value function Q_π(s, a) can be obtained :

$$Q^\pi(s,a) = R(s,a) + \lambda \sum_{s' \in S} P(s^{'}|s,a) \sum_{a \in A} \pi(a|s^{'}) Q^\pi(s^{'},a) \tag{5}$$

Then, the Bellman Optimality Equation shows as follows:

$$V^*(s) = \underset{a}{max} Q^*(s,a) = maxR(s,a) + \lambda \sum_{s' \in S} P(s^{'}|a,s) V^*(s^{'}) \tag{6}$$

$$Q^*(s,a) = R(s,a) + \lambda \sum_{s' \in S} P(s^{'}|a,s) \underset{a}{max} Q^*(s^{'},a^{'}) \tag{7}$$

2.1.3 Q-Learning Algorithm

Q-learning is a typical model-free and off-policy RL algorithm. An agent maintains a two-dimensional Q-table and updates the Q-values in it. Q-values are action utility functions that show how good or bad it is to perform certain actions in a certain state and they are stored in the agent. The update function of the Q-value can be expressed as:

$$Q_{t+1}(s,a) = (1-\beta)Q_t(s,a) + \beta(R(s,a) + \underset{a \in A}{\lambda} maxQ_{t+1}(s^{'},a^{'})) \tag{8}$$

β represents the learning rate, the higher its value, the smaller the proportion of historical training results used. λ represents the learning discount rate, the higher its value, the greater the focus on long-term strategies, and vice versa, the focus on immediate benefits.

Firstly, both the agent and the environment initialize their parameters and enter the first loop, where the agent selects the action in state s_t and executes it based on the ω-greedy policy, passes the action to the environment, observes the new state s_{t+1}, and receives the reward R_{t+1}. The Q value in the Q table is updated, and the next loop is entered. It is worth noting that it is then possible to select the action with the maximum Q value that is updated in the first loop. This exploitation concept makes the best policy based on known information. This can lead to local optimization. Exploration helps to avoid this when the agent selects the action with a non-optimal Q value with some probability of getting more information. Balancing exploration and exploitation greatly impacts the agent's learning; too much exploitation leads to not getting the globally optimal policy, while too much exploration hinders the limited access of the agent to the current reward, leading to a decrease in learning efficiency.

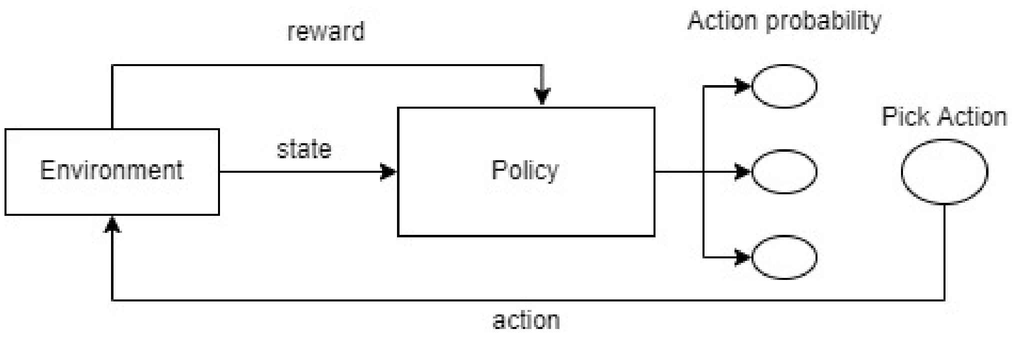


Figure 2.2: Plicy Gradient

2.1.4 Policy Gradient

The Policy Gradient (PG) algorithm is another branch of reinforcement learning, which is a policy-based approach, different from Q-learning, which is based on Q value. The earliest concept was proposed in 1999[14]. As shown in Figure 2.2, PG does not need to calculate the value function but calculates the probability distribution of the random policy output actions from the obtained rewards, and then selects the action with the highest probability to perform the operation. It can select actions over a range of continuous distributions. Usually, a Gaussian probability function is used to represent the probability distribution of continuous actions:

$$\pi_\theta(\alpha|s) \sim N(\phi(s)^T\theta, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\alpha - \phi(s)^T\theta)^2}{2\sigma^2}) \tag{9}$$

where the parameter θ is used to regulate the probability of occurrence of each action. The goal of PG is to adjust the parameter θ so that the expected return of the strategy π_θ is maximized. The key is needed to get the gradient of the return function $J(\pi_\theta)$ about theta, and then use the gradient ascent algorithm to find it. The formula is:

$$\nabla \bar{R}_\theta = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla log p_\theta(a_t^n|s_t^n) \tag{10}$$

2.1.5 Actor Critic

The earliest Actor-Critic (AC) concept comes from 1977[15]. 2012, et al. proposed the first Off-Policy Actor-Critic algorithm[16]. As in Figure 2.3, AC combines the value iteration and policy iteration methods. The actor uses the idea of PG to output actions according to the probability distribution. The critic is based on value and updates the probability of actions according to the current state. AC employs the Q function rather than the cumulative reward $R(\tau)$ of gradient equation 10 in 2.1.4, and the gradient of AC's actor parameters is:

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n-1}^{N} \sum_{t=1}^{T_n} Q^{\pi\theta}(s_t^n, a_t^n) \nabla log \rho_\theta(a_t^n | s_t^n) \tag{11}$$

The critic based on the Temporal Difference error (TD-error) method updates the loss in the squared error between the Q-value and the actual Q-value with the formula:

$$loss = \frac{1}{N} \sum_{n-1}^{N} \sum_{t=1}^{T_n} (r_t^n + \max_{a_{t+1}^n} Q^{\pi\theta}(s_{t+1}^n, a_{t+1}^n) - Q^{\pi\theta}(s_t^n, a_t^n))^2 \tag{12}$$

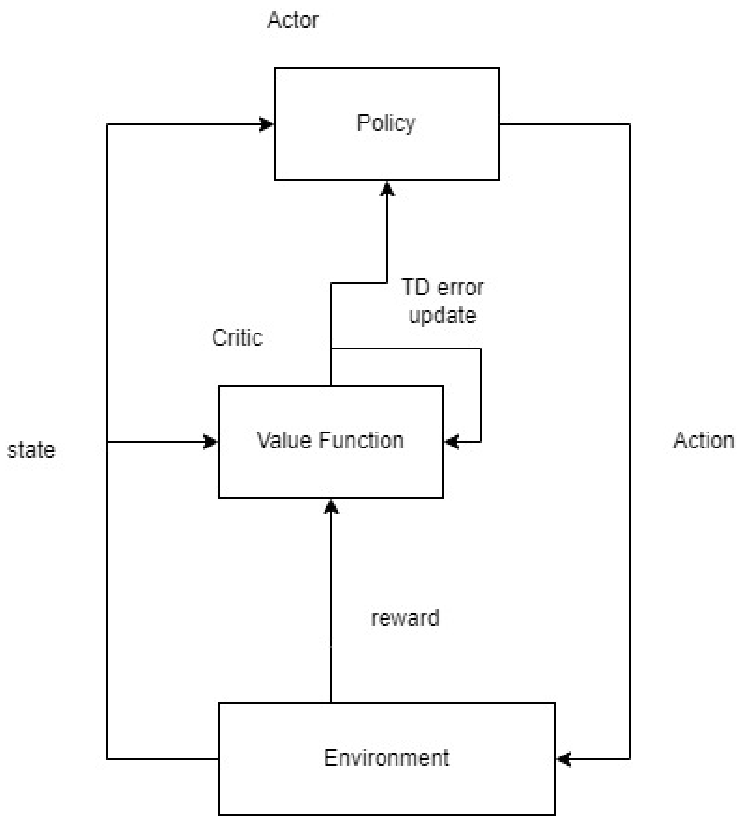


Figure 2.3: Actor Critic

2.2 Deep Learning

Deep learning (DL), a branch of machine learning, is an algorithm based on mimicking the function of the human brain. It uses artificial neural networks(ANN) as a framework to learn specific information[17]. The use of deep learning can be traced back to 1971, Ivakhnenko trained eight layers of neural networks using a data processing group approach[18]. Recently, due to the advancement of computer hardware, important contributions have been made to industries such as speech recognition, computer vision, etc[17].

2.2.1 Back Propagation

Back Propagation(BP) is a significant algorithm in multilayer neural networks. It is built based on the gradient descent method and uses the chain rule of the composite function. The input and output relationship of a BP network is a mapping relationship whose information processing power comes from multiple combinations of simple, nonlinear functions. Therefore, it has a strong function recurrence capability, this is the basis of the BP algorithm. Its learning process consists of forwarding and backward propagation processes. Suppose the desired value is not obtained in the output layer during the forward propagation process. In this case, the sum of the squared differences between the expected value and the output is used as the objective function and given to BP. Layer by layer, the partial derivative of the objective function for each neuron weight is found and used as the objective of the weight vector of the gradient function. The results are then modified, and the learning ends once the desired value is reached[19].

2.2.2 Neural Network and Convolutional Neural Network

ANN, also known as Neural Networks (NNs), processes information by adjusting the interconnected relationships among a large number of complex nodes inside. The earliest concept of NN can be back to 1943 when Walter Pitts and Warren McCulloch first proposed the mathematical model of neurons in the study of neurons. In 1948, John von Neumann worked on a comparison of the structure of the stored-program computer and the human brain.

As shown in Figure 2.4, NN is usually a three-layer model: there is an input layer, a hidden layer and an output layer. Each layer consists of a certain number of computational units. The hidden layer can consist of 0-n layers. The data is input from the input layer, and each layer performs a weighted summation of the input ωx + b and passes it to the next layer, which is called forward propagation, and finally, the output layer outputs the calculation result. The weights ω are obtained by back-propagation of the computing unit. The

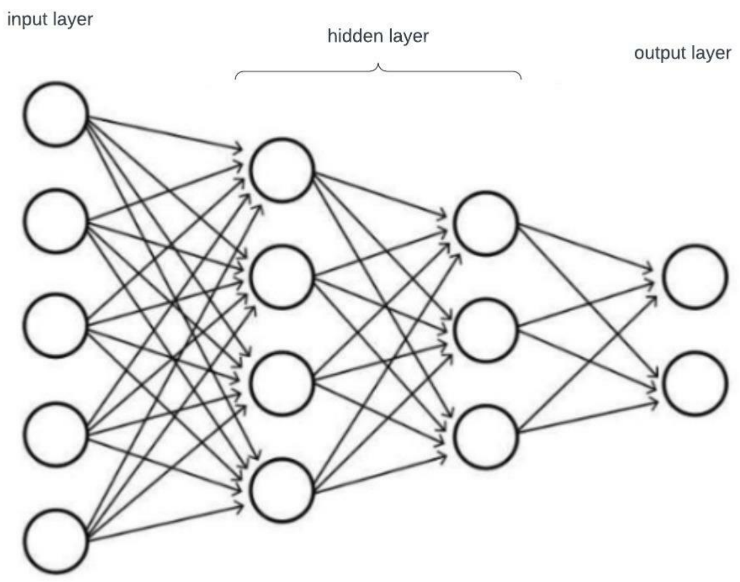


Figure 2.4: a neural network structure

computational unit may also contain activation functions. Common activation functions include sigmoid, tanh, sign, relu, softmax, etc. Figure 2.5 is a simple neural network function processing process.

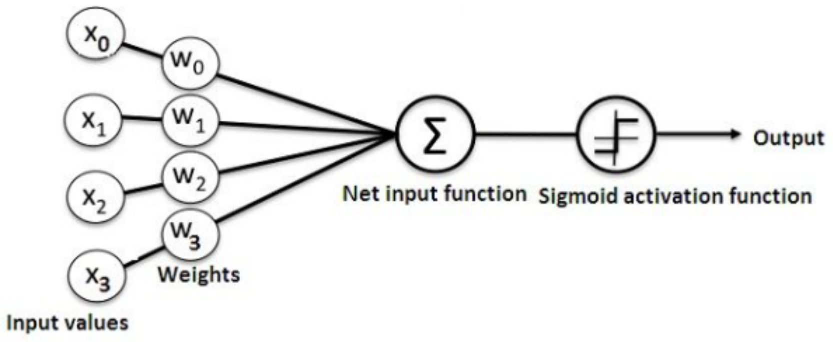


Figure 2.5: Simple NNs function processing

Convolutional neural networks (CNN) is one of the very representative neural network architectures [20], mainly popular in image and speech processing [21][22]. In 1979, Japanese scholar Kunihiko Fukushima proposed the Neocognitron model, which was the earliest CNN concept.[23]. The first CNN was proposed by Alexander Waibel et al. in 1987, which was a time-delayed neural network for speech recognition.[24].

The emergence of CNN is to solve the computationally complex problems of traditional neural networks. Generally, the three concepts of fully connected layer, convolutional layer and pooling layer replace the traditional hidden layer. The convolutional layer includes convolutional kernels to extract features of the input information. Its parameters include stride, padding and kernel size, which determine the output information. As shown in Figure 2.6, When the convolutional layer works, it will regularly scan the input information. The pooling layer then receives the information output by the convolutional layer and performs feature filtering. Generally, the methods of max pooling and average pooling are used. Finally, the fully connected layer performs a nonlinear combination of the extracted features and outputs them. In general, CNN reduces the overall computational cost, retains the details of the original information to the greatest extent, and extracts more abstract features.

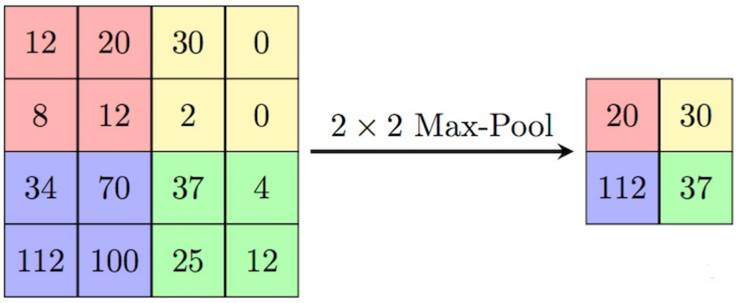


Figure 2.6: max pooling

2.2.3 Optimization Algorithms

Gradient Descent

Gradient Descent(GD) is an iterative method, often used in neural networks, which continuously iterates to compute the gradient of the loss function j(θ), determine the distance between a specific direction of the point and the target, and finally find the minimum and the associated parameters. A common form is Batch Gradient Descent(BGD). It uses all samples at once for gradient update but brings the problem that it is very computationally intensive when the number of samples is large. Mini-batch Gradient Descent(MBGD) and Stochastic Gradient Descent(SGD) can solve this problem properly, and they are also prevalent GD algorithms[25].

SGD randomly takes one sample for updating and then selects the following sample. Thus the iteration speed is significantly increased. The problem is the decrease in accuracy and the possibility of convergence to a local optimum[25].

MBGD is between batch gradient descent and random gradient descent, in which a batch of samples is randomly selected for updating during the training process, ensuring the algorithm's iteration speed and guaranteeing better accuracy. However, improper selection of batch size can also bring some performance problems[25].

There are also some common algorithms used in neural networks such as Adagrad, RMSprop, Adam.

Adagrad

Adagrad was proposed by Duchi et al[26]. It allows the learning rate to be adaptive. Unlike SGD it can automatically adjust the learning rate to fit different parameters. The learning rate iteration equation is:

$$\alpha_t = \frac{\alpha_0}{\sqrt{\sum_{i=1}^{t-1} \frac{df}{dx}(x_i)^2} + \epsilon} \tag{13}$$

Where a_0 is a custom global learning rate that speeds up training by dividing by the square root of the sum of squares of historical gradients, which is different for each parameter and can smooth out jitters. ϵ is to prevent errors caused by dividing by zero.

RMSprop

Geoff Hinton et al. proposed the Root Mean Square Propagation (RMSprop) algorithm[27], which is an improvement of Adagrad. one of the limitations of Adagrad is that it may lead to minimal learning rates for each parameter in the later stages of learning, which can lead to a decrease in learning speed and possibly failure to find the global optimal solution. The learning rate formula is as follows:

$$\begin{aligned} Sd\omega_t =& \lambda Sd\omega_{t-1} + (1-\lambda)(d\omega_{t-1})^2 \\ \alpha_t =& \frac{\alpha_0}{\sqrt{Sd\omega_t} + \epsilon} \end{aligned} \tag{14}$$

RMSProp uses the concept of the exponentially weighted moving average, Sdω_t denotes the gradient momentum accumulated in the first t-1 rounds of the loss function, and λ is an exponentially controlled total and current quantity.

Adam

Kingma et al. proposed the Adam algorithm[28], which is one of the more popular deep learning optimizers. It combines the advantages of the momentum gradient descent algorithm and the adaptive learning rate algorithm, which can not only alleviate the gradient oscillation problem, but also adapt to the sparse gradient oscillation. Adam's iteration formula is as follows:

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\Delta f(x_t) \tag{15}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)\Delta f(x_t)^2 \tag{16}$$

where m_t is the first-order moment-weighted mean of the historical gradient, which is similar to finding the L1 parametrization of the historical gradient. It applies a correction to the historical gradient to obtain a stable gradient update value. v_t is the second-order weighted mean of the squared historical gradient, similar to finding the L2 parametrization of the historical gradient, and the idea comes from the adaptive learning rate algorithm, which is designed to make the learning rate different for each parameter. However, since the first and second moments are initialized with zero values, they are biased, and they are corrected for:

$$\widehat{m} = \frac{m_{t+1}}{(1 - \beta_1^t)} \tag{17}$$

$$\widehat{v} = \frac{v_{t+1}}{(1 - \beta_2^t)} \tag{18}$$

Finally, the gradient descent formula is obtained:

$$x_{t+1} = x_t - \alpha \frac{\widehat{m}}{\sqrt{\widehat{v}} + \epsilon} \tag{19}$$

In the iterative process, a parameter has been updated with a slight gradient, indicating that the gradient change is relatively stable and the learning rate can be increased. A small value will be obtained by formula 15 and 16, and the learning rate will become larger by modifying formulas 17 and 18. The opposite is true if the parameters are iterated with a large gradient.

2.3 Deep Reinforcement Learning

2.3.1 Deep-Q-network

In the Q-learning algorithm, the Q table is the core. The Q value of a certain action in all states is stored here to provide guidance for the action. This only applies when the action and state dimensions are not high, and they are discrete states. Once actions and states are high-dimensional and continuous, the Q-table will become very huge, resulting in inefficient queries. An example is Go, which has as many as 10^{170} move possibilities, and the storage and query of the Q table become unavailable. DQN was born[29].DQN performed dimension compression on the Q table and adopted Value Function Approximation:

$$Q\phi(s,a) \approx f(s,a,k) \tag{20}$$

Where k represents the input parameters of the function f, which is used to approximate the distribution of Q values, function f is the neural network.

A major improvement point of DQN is the addition of a target Q network to the Q network for predicting the realization of Q. The update formula of Target Q is:

$$targetQ = r + \gamma * Qmax(s^{'}, a^{'}, \theta) \tag{21}$$

The loss function is obtained from the target Q and Q estimation, generally using the mean square error:

$$L(\theta) = E[(targetQ - Q(s,a,\theta))^2] \tag{22}$$

The Q estimation function will be updated each time, while the update of the target will lag, After that all the data of the Q network will be transferred and copied to the target network, so that the target Q is fixed for a period of time, making the algorithm update more stable and avoiding oscillations.

Another important improvement is the introduction of the experience replay mechanism. Behaviour policy acquires experience after exploring or executing an action and stores it in the experience pool, which can be represented as (s,a,r,s'). Target-policy then randomly selects an experience to update the network from the experience replay pool. This advantage is that it can cut the correlation between experiences in the experience pool and improve learning efficiency. Secondly, each experience can be learned repeatedly to improve data usage. Figure 2.7 shows the DQN execution process.

2.3.2 Advantage Actor Critic

A2C is improved from AC and is a synchronous product of Asynchronous Advantage Actor-Critic (A3C). As shown in Figure 2.8, in the A2C algorithm, the actor is responsible for outputting the probability of each action, and the critic outputs the action value Q, both of which are implemented by neural networks. It optimizes the problem of excessive variance of the AC algorithm by adding a baseline V function to the gradient. This makes the training process more stable. As shown in A2C's gradient equation 23:

```
Algorithm  DQN execution process
Initialize function Q, target Q function $\hat{Q} = Q$
 1: for each episode do
 2:   for each time step t do
 3:     For a given state $s_t$, perform action $a_t$ based on Q $\varepsilon$-greedy policy.
 4:     Get the reward $r_t$, and update the state $s_{t+1}$.
 5:     Store $(s_t, a_t, r_t, s_{t+1})$ into the experience pool.
 6:     Sample (usually in batch) from the pool of experience $(s_i, a_i, r_i, s_{i+1})$.
 7:     The target value is targetQ $= r_i + max_a \hat{Q}(s_{i+1}, a)$.
 8:     Update the parameters of Q so that $Q(s_i, a_i)$ is as close to targetQ as possible.
 9:     Copy the Q value to $\hat{Q}$ every j steps
10:   end for
11: end for
```

Figure 2.7: DQN execution process

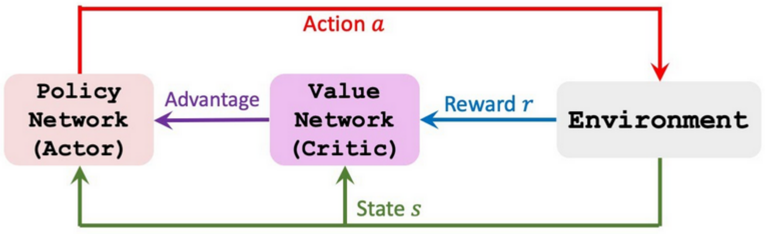


Figure 2.8: Advantage Actor-Critic

$$\nabla \bar{R}(\theta) = \frac{1}{N} \sum_{n-1}^{N} \sum_{t=1}^{T_n} (Q^{\pi\theta}(s_t^n, a_t^n) - V^{\pi\theta}(s_t^n)) \nabla log \rho_\theta(a_t^n | s_t^n) \tag{23}$$

However, this requires two neural networks to evaluate Q and V separately, and a transformation is needed:

$$Q^{\pi}(s_t^n, a_t^n) = r_t^n + V^{\pi\theta}(s_t^n)) \tag{24}$$

Where r_t is the current reward. The loss function of the critic can be obtained:

$$L(\theta) = \frac{1}{N} \sum_{n-1}^{N} \sum_{t=1}^{T_n} (r_t^n + V^{\pi}(s_{t+1}^n) - V^{\pi}(s_t^n))^2 \tag{25}$$

Compared with DQN, as in Figure 2.9, the A2C algorithm uses a parallel architecture where each process interacts independently with its environment to get unique and unrelated experiences, which breaks the coupling between the experiences and also plays a comparable effect to experience replay. Since there is no experience pool, it saves memory compared to DQN.

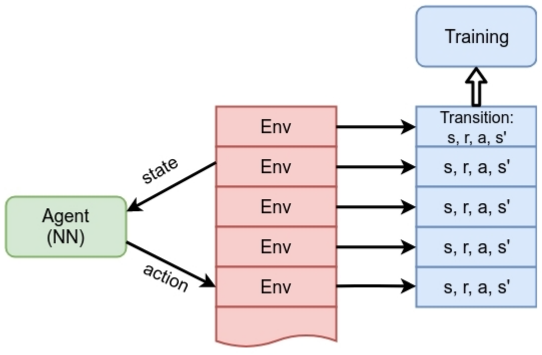
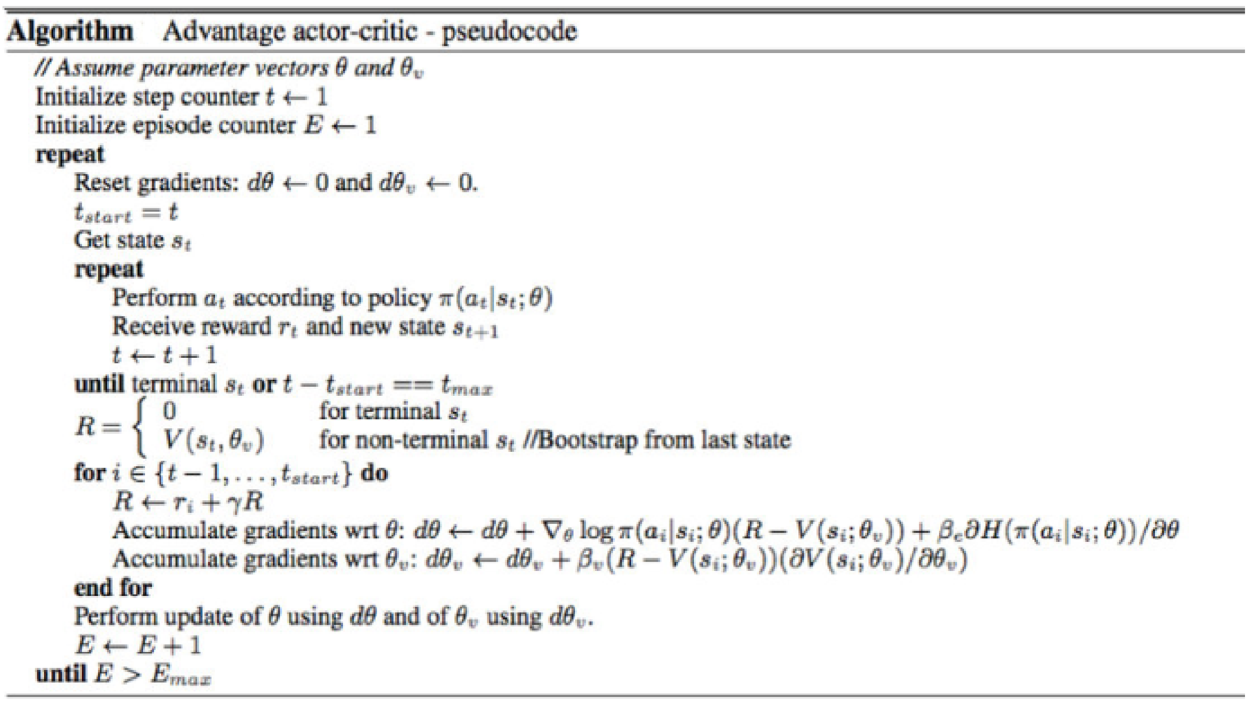


Figure 2.9: A2C Parallel Architecture

The A2C execution process is outlined in Figure 2.10.



Algorithm Advantage actor-critic - pseudocode

```
// Assume parameter vectors $\theta$ and $\theta_v$
Initialize step counter $t \leftarrow 1$
Initialize episode counter $E \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta)$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t+1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t,\theta_v) & \text{for non-terminal } s_t \text{ //Bootstrap from last state} \end{cases}$
    for $i \in \{t-1,\ldots,t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta$: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \partial H(\pi(a_i|s_i;\theta))/\partial\theta$
        Accumulate gradients wrt $\theta_v$: $d\theta_v \leftarrow d\theta_v + \beta_v(R - V(s_i;\theta_v))(\partial V(s_i;\theta_v)/\partial\theta_v)$
    end for
    Perform update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
    $E \leftarrow E+1$
until $E > E_{max}$
```

Figure 2.10: A2C pseudocode

2.4 Related Work

This section reviews some recent papers related to the problem addressed in this paper, starting with a description of how researchers have addressed the trajectory optimization problem. Then some papers on UAV trajectory optimization for BS connectivity are presented. Finally, it explains how deep reinforcement learning works for UAV connectivity and UAV trajectory optimization for connectivity.

2.4.1 The trajectory optimization problem of UAV

Trajectory planning of UAVs is a widely studied problem. Researchers try to solve UAV energy saving, connectivity, and other issues by optimizing trajectories. They use different algorithms, as shown in Table 2.1, and this section introduces some related works.

Table 2.1: Related work on UAV trajectory optimization problem

Research	approaches/algorithms	details
[32]	Online method based on Artificial Forces	3D trajectory planning in city environment
[33]	bisection search technique ,block coordinate descent and dynamic programming	2D trajectory planning in Internet of Things(IoT) networks
[34]	dynamic programming	3D trajectory optimization in for cargo delivery
[35]	SARSA	26 actions in 3D IoT data collection Scenes
[36]	double q-learning	2D trajectory planning for UAV base stations (UAV-BS)

[32]The problem faced is that the UAV may not be able to reach the specified mission area within the specified time because of the battery limitation. they proposed an online method to optimize the 3D UAV trajectory problem for the UAV to move rapidly toward Training and Recharging Areas (TRA). This online algorithm is a movement control algorithm that senses the environment through the sensors on the UAV, calculates the Euler distance between the obstacles and the UAV, and dynamically adjusts the cruising altitude, speed, etc. of the UAV, which serves to optimize the trajectory and also avoids collision with the obstacles.

[33]proposes a method based on the block coordinate descent (BCD) and the bisection search technique, aiming to address the data dissemination problem of UAV-BSs in post-disaster areas. Considering the wired battery of the UAV, it is necessary to fly over nodes where data may not be available to optimize trajectories to save mission completion time and energy. First, using the binary search technique, the flight time is constantly searched to know if the requirements of the IoT nodes are satisfied, and then a decision is

found by BCD for given flight duration. This decision is the UAV trajectory, the problem of energy optimization.

[34] uses dynamic programming(DP) to solve the freight UAV energy-efficient trajectory and to guarantee UAV and cellular connectivity. They formulate this problem as a multi-objective problem. The scenario is formulated as a grid cell shape, and the UAV has eight directions. Then the endpoints are set and iterated continuously, in which the distance flown by the UAV is counted and the total energy consumption. It is judged that the on-board energy cannot be exceeded, and the set disconnect rate cannot be exceeded. The final result is obtained. This is an NP-hard problem.

[35] uses State-Action-Reward-State-Action (SARSA) to improve the flight energy efficiency of UAV in IoT data collection scenarios. SARSA is a TD-based algorithm. They propose 26 actions, which make the scenario more challenging, and then model the scenario as an MDP model with the state space as the velocity and position of the UAV and the reward is quotient of total throughput and total energy consumption of the UAV over time. The final optimized trajectory is obtained through continuous learning for comparison with the Q-learning algorithm.

[36] introduces an method base on double q-learning to optimize the movement trajectory of UAV-BSs and considers the nodes' priority to be served as a combinatorial optimization problem. The UAV-BSs and the nodes to be served are randomly generated in the scenario, the service priority is according to the residual energy level of the nodes, and the UAV is considered as a Q-learning agent. Action is expressed as the location to the next node requiring service, and the state space has UAVs and the location of nodes with node priority service levels. Rewards are penalties for service delays and flight energy consumption. They use double q-learning to avoid getting into local optimization.

2.4.2 The connectivity problem of UAV

Improving the BS connectivity of UAVs air-to-ground is also one of the key research directions. On the one hand, UAVs need good connectivity when performing tasks, improving throughput and thus performing tasks better. On the other hand, the problem of path loss caused by obstacles is also a big challenge.

[37]introduced a mobility management model based on a Q-learning method to reduce the number of switching of connected ground base station (GBSs) while ensuring the connectivity of UAVs and GBSs. They used the reference signal received power (RSRP). As a measure of connectivity, path-loss is taken into account. The connectivity is increased by adjusting the downtilt (DT) of the GBSs. Therefore, the angle of DT adjustment is chosen

Table 2.2: Related work on connectivity problem

Research	approaches/algorithms	details
[37]	Q-learning	Improving UAV connectivity to ground BS
[38]	TD learning	trajectory planning to improve connectivity
[39]	graph search	3D trajectory optimization in for cargo delivery

as actions in the MDP model, with a total of 8 ranges between [-2,12] at intervals of 2 degrees. The reward is then a combination of the weights of the RSRP and ground user equipment (GUE) ratios, and the state space is the spatial segment of the UAV flight.

[38]demonstrates a TD learning-based approach to trajectory planning in cities that minimizes UAV task completion time while maintaining good connectivity to the cellular network and avoiding cellular coverage gaps. The simulation environment consists of a grid. The location of the UAV defines the state space of the model, the actions are the flight direction of the UAV, and the reward is whether the area to which the UAV travels has signal coverage, denoted by -1 and 1.

[39]Uses the A star algorithm, the UAV trajectory design problem is formulated as a method of seeking the shortest path, weighing the path length against the quality of the connection. This is a 2D scenario where the UAV has eight directions to move in. The connectivity is measured in terms of connectivity outage ratio (COR), connectivity outage duration (COD) and signal coverage.

2.4.3 UAV application of Deep Reinforcement Learning

Recently, deep reinforcement learning(DRL) has become a popular technique for studying UAVs. This is because the UAV environment and its parameters are becoming more complex and variable, and neural network techniques can be adapted to this environment, making the simulated environment more similar to the real one.

[40]Uses DQN to optimize the connectivity of UAVs with BSs in cities, fully considering the density of buildings, the influence of Line-of-Sight (LoS) caused by buildings on connectivity, and increasing it by adjusting the height of UAVs. UAV acts as an agent to obtain information from the environment, and the state space consists of the density of BS, the density of buildings, the Signal-to-Interference-and-Noise Ratio (SINR), the distance

Table 2.3: Related work of UAV using Deep Reinforcement Learning

Research	approaches/algorithms	details
[40]	DQN	Optimize connectivity in the city
[41]	DDQN	Optimize UAV trajectory to improve connectivity
[42]	DQN	Optimize UAV trajectory to save energy, under the constraints of remaining power, safe distance
[43]	DQN	Optimal trajectory problem for energy saving and air-ground communication balance in 2D environment

between UAV and base station, and the height of UAV. Actions have three up, down and unchanged. The reward is SINR.

[41]proposes a 3D trajectory optimization method based on DDQN to avoid areas with weak signal coverage of BSs. A 3D coverage map is first constructed to store the expected interruption probability for each coordinate. Then the UAV flight is optimized by DDQN. Since the simulator environment coordinates have been constructed, the authors used 26 directions as UAV flight manoeuvres. The reward is a combined value consisting of the relative distance of the UAV at time step t-1 and t and the focus of the UAV, which is to make the UAV fly to the endpoint.

[42]designed a scheme to design the UAV trajectory, in order to maintaining a reliable connection with the BS at all times for the UAV during the mission. The signal diagram is first constructed based on the SINR signal strength, and then the DQN-based algorithm is used to find the best action for each step to find the best trajectory that guarantees connectivity. Eight directions of 2D are used as actions.

[43]is a UAV and GBSs connection energy optimization problem, considering connectivity, flight energy consumption, and other issues. First, a channel knowledge map (CKM) is specified for the scenario. The shortest path is obtained using path discretization and convex optimization techniques and algorithms. Then, the previous environmental problems are considered and combined into an MDP model using DQN, and let the UAV as an agent to interact with the environment to solve the problem.

2.5 Summary

This chapter covers some basic concepts from reinforcement learning to deep reinforcement learning and also introduces the concepts and basic features of the DQN and A2C algorithms used in this experiment. This experiment explores the feasibility of the A2C algorithm for optimizing UAV trajectories based on the features of the two algorithms. The specific design and implementation will be discussed in Chapters 3 and 4. Then, the use of UAV for trajectory optimization, connectivity, and deep learning is discussed.

3 Design

This chapter describes the layout of buildings and base stations in the UAV flight scenario, the base station signal model, and Modeling the connectivity-aware trajectory optimization problem of UAV as MDP. Firstly, the states, action spaces and reward functions of DQN and A2C in the UAV context are explained and defined, in addition to showing the algorithm execution process of the two algorithms in this experiment.

3.1 Buildings Distribution Model

In this experiment, buildings and BS (obstacles) are generated by the simulator at a fixed location. Buildings will have different heights and volumes. They are represented by 1 or 0 in the scene as blocking or non-blocking to the UAV flight, which the UAV may traverse or fly over them. The environment layout will be described in section 5.2. The building also affects the line-of-sight (Los) of the UAV, and also causes signal attenuation, non-Lineof-Sight (NLoS). Details will be introduced in 3.2.

3.2 UAV And Base Station Connection Model

We assume that there is a set U=$\{u_1,u_2,...u_i\}$ of UAV positions and a set of BSs $\{b_1,b_2,...b_j\}$, and the UAV can automatically connect to a BS with a better signal. Our goal is to optimize the trajectory to improve the user experience during UAV flight. We therefore use SINR to measure the signal acquired when the UAV is connected to the BS. The function it gives is as follows[44]:

$$SINR_t = \frac{\rho\eta\left(\theta\right)\kappa\left(\phi_{i,j}\right)c\left(x_{i,j}^2+\Delta\gamma^2\right)^{-\beta_{t_s}/2}}{I+\tau^2} \tag{1}$$

where ρ is transmit power from BS, $\Delta\gamma$ is the height difference between the UAV and the BS, βt_s is the pathloss exponent, where t_s is whether the UAV has LoS or NLoS for its service BS, $x_{i,j}$ is the vertical distance between the drone and the base station, c is the near-field path loss, τ^2 is the noise power, η is the directional antenna gain of the UAV, This experiment assumes that the UAV is always aligned to the BS to gain, it is given as $\eta(\theta)$=16 π/θ^2 , κ is the antenna gain of the BS. μ is the angle function between UAV and BS which can be defined as[44]:

$$\kappa(\phi_{i,j}) = \frac{1}{N_t} \frac{sin^2 \frac{A_t \pi}{2} sin(\phi_{i,j})}{sin^2 \frac{\pi}{2} sin(\phi_{i,j})} \tag{2}$$

Assuming that the BS has a three-sector antenna, where A_t represents a Uniform Linear Array (ULA) of how many elements each antenna has. $\phi_{i,j}$ is the arctangent function of $\Delta\lambda$ and UAV and BS vertical distance $x_{i,j}$ such that $\phi_{i,j}$ = arctan($\Delta\lambda$ / $x_{i,j}$).

In Equation 1, I is the total interference power of the base stations in the set of areas near the base station to which the UAV directional antenna is aligned. This area is an annulus with an angle of θ, the major and minor radius can be expressed respectively as[44]:

$$R = \begin{cases} \dfrac{|\Delta\lambda|}{tan(|\phi_{i,j}| - \theta/2)} & if\ \theta/2 < |\phi_{i,j}| < \pi/2 - \theta/2 \\ \dfrac{|\Delta\lambda|}{tan(\pi/2 - \theta)} & if\ |\phi_{i,j}| > \pi/2 - \theta/2 \\ \infty & otherwise \end{cases} \tag{3}$$

$$r = \begin{cases} \dfrac{|\Delta\lambda|}{tan(|\phi_{i,j}| + \theta/2)} & if\ |\phi_{i,j}| < \pi/2 - \theta/2 \\ 0 & otherwise \end{cases} \tag{4}$$

The values of all parameters are described in Table 4.2 in chapter 4.

Finally we express the SINR in dB to measure the signal of the UAV at a certain location[45]:

$$SINR_{dB} = 10log_{10}(SINR) \tag{5}$$

3.3 Environment

The path planning problem for UAVs is formulated as an RL problem. agents and environment interactions in UAVs. Firstly the environment is the connectivity-aware flight path in discrete time t. The UAV on a mission faces obstructions such as buildings in an urban environment, and the base station signal limits the situation. The agent observes the state $s_t \in S$ from environment in time t, then chooses the best action $a_t \in A$, takes the action and the agent gets a reward and updates the value, then transfers to the next state $s_t + 1 \in S$.

3.3.1 State

The state is obtained by the agent observing the environment according to time t. The state space is defined as: UAV's position, UAV's target point, building's position, the SINR value received at the UAV's current time, that is, S={L_{UAV},T_{UAV},L_{build},SINR},Among them, L_{UAV} contains the location information of the UAV, which is composed of x, y, and z coordinates. T_uav is a predefined coordinate x,y,z. L_{build} is generated when the simulator initializes the scene in 3.1. The SINR is first initialized by the simulator to initialize the position of the BS, and then the SINR value received by the UAV at t is calculated by the formula in 3.2.

3.3.2 Action

Since this paper optimizes the 2D path, in order to make the UAV motion more flexible, we define the DQN action set A to have 8 actions, as shown in Figure 3.1. That is, the UAV can be moved to: Southeast, Northeast, Southwest, Northwest, East, South, West, North. The action set is set as: A={ES,EN,WS,WN,E,S,W,N,}.The number represents a distance in each direction, for example, direction 1 represents a move of 1 to each of the x and y axes.

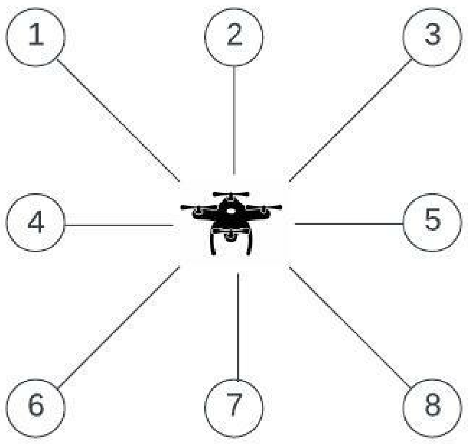


Figure 3.1: UAV eight directions

We define the action space of the A2C algorithm as a=[x,y], where x ∈ [-1,1], y ∈ [-1,1], the algorithm selects a value from the range of the set x and y according to the probability distribution, and combines them into a vector. Possible actions For example a=[x=0.5,y=-0.5].

3.3.3 Reward

The reward is the return obtained after the agent performs the action. In this scenario, since the UAV needs to fly to the destination and signalaware, the reward is set as a combination of the distance from the UAV's present position to the destination and the weight of the SINR. According to the signal formula from the base station in 3.2, the position of the UAV is obtained in the time segment of t1, and the SINR value of the present position of the UAV is calculated, and the larger the value is, the larger the reward value is obtained. The final combination into the reward formula is as follows:

$$R_t = \begin{cases} \omega_1 d_t + \omega_2 s_t, \textit{if } \ B(t) = 0 \\ -1, \textit{if } \ B(t) = 1 \end{cases} \tag{6}$$

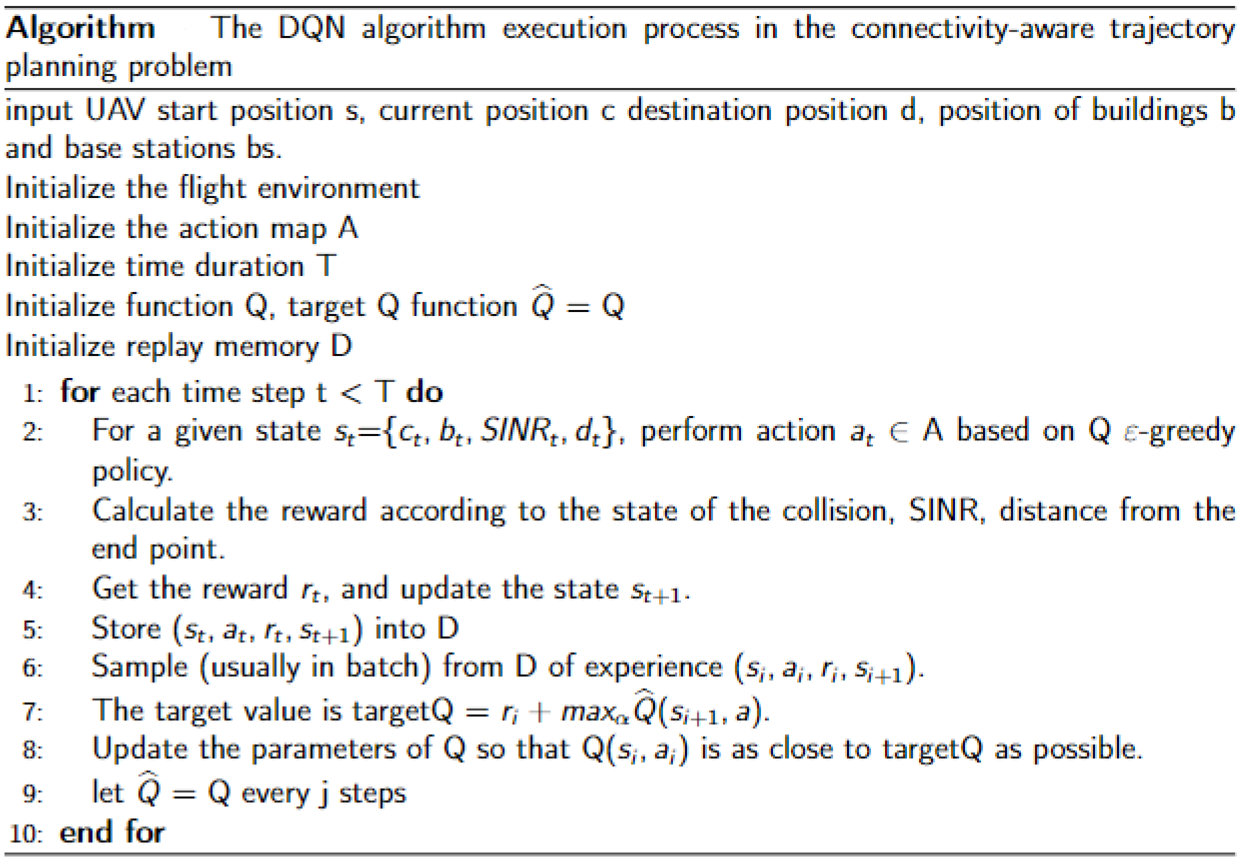
Where d represents the distance from the UAV to the destination, and s represents the signal value of the SINR from section 3.2. ω_1 and ω_2 are coefficients that control the trade-off between the distance from the UAV to the destination and the SINR, B(t) is a function to calculate whether the UAV collides with a building. A value of 1 indicates that a building blocks the action taken, and a penalty is taken. We use the difference between the next action and the current action to calculate the reward, which requires us to predict the best action in advance. Since the action of DQN is fixed, we can traverse the trade-off between the calculated distance to the endpoint and the SINR value, and in the A2C algorithm In the derivation method, we adopt the method of derivation.

3.4 Training Design

This section presents how UAV agents are trained using DQN and A2C under the connectivity-aware trajectory design problem, in addition to the design of pseudocode.

3.4.1 DQN

Figure 3.2 explains how the DQN trains the UAV to find connectivity-aware trajectories, showing the experience playback, and the target network is featured in the algorithm. First, the UAV agent observes the environment state s_t, and the neural network calculates the Q value for each action a $\in$ A, based on the state. The a four tuple of state (r_t , s_{t+1} , s_t , a_t) are saved in the experience replay pool. Next, if the experience pool memory is full, the loss function is calculated using the comparison of the Q value and the target value, and the better one is replaced by the poor one. The target network updates lag behind the main network to make the learning more stable. In addition, DQN will collect the tentative actions of UAV a certain number of steps before learning, which is to collect data and store it in the experience pool.



Algorithm The DQN algorithm execution process in the connectivity-aware trajectory planning problem

```
input UAV start position s, current position c destination position d, position of buildings b
and base stations bs.
Initialize the flight environment
Initialize the action map A
Initialize time duration T
Initialize function Q, target Q function $\widehat{Q} = Q$
Initialize replay memory D
 1: for each time step t < T do
 2:     For a given state $s_t$={$c_t, b_t, SINR_t, d_t$}, perform action $a_t \in A$ based on Q $\varepsilon$-greedy
        policy.
 3:     Calculate the reward according to the state of the collision, SINR, distance from the
        end point.
 4:     Get the reward $r_t$, and update the state $s_{t+1}$.
 5:     Store $(s_t, a_t, r_t, s_{t+1})$ into D
 6:     Sample (usually in batch) from D of experience $(s_i, a_i, r_i, s_{i+1})$.
 7:     The target value is targetQ = $r_i + max_a \widehat{Q}(s_{i+1}, a)$.
 8:     Update the parameters of Q so that Q$(s_i, a_i)$ is as close to targetQ as possible.
 9:     let $\widehat{Q}$ = Q every j steps
10: end for
```

Figure 3.2: DQN execution process in UAV flight environment

3.4.2 A2C

Figure3.3 shows how A2C trains the UAV agent. Unlike DQN, the agent takes continuous actions from the actor-network, which is a probability distribution. We do not need to set the set of actions in advance as in DQN, but only need to give a range. A2C uses the critic network to measure the merit of this action.

3.5 Summary

This chapter introduces the environment design and layout for UAV connectivity-aware path optimization, as well as the training design and pseudocode of the two algorithms, A2C and

DQN, including the design of state space, action space and reward value. All specific parameters will be given in Chapter 4.

Algorithm The A2C algorithm execution process in the connectivity-aware trajectory planning problem

```
input UAV start position s, current position c destination position d, position of buildings b and base stations bs.
Initialize the critic nets $V_\alpha$ and actor nets $\pi_\theta$.
Initialize the flight environment
Initialize time duration T
Observe the initial state $S_0$.
1: for each time step t < T do
2:    Initialize empty episode minibatch.
3:    for each sample episode i do
4:       Select an action $a_i$ from actor $\pi_\theta$.
5:       Perform the action $a_i$.
6:       Observe the next state $s_{i+1}$.
7:       Get the reward $r_{i+1}$
8:       Store($s_i$,$a_i$,$r_{i+1}$) in the episode minibatch.
9:    end for
10:   if $s_t$ is not terminal then
11:      Set R=$V_\alpha(S_t)$.
12:   else
13:      R=0.
14:   end if
15:   Reset gradient $d_\theta$=0 and $d_\alpha$=0
16:   for each episode i $\in [n-1, 0]$ do
17:      Update the discounted sum of rewards R = $r_i + \lambda$R
18:      Accumulate the policy gradient using the critic $V_\alpha$: $d_\theta \leftarrow d_\theta + \Delta_\theta log\pi\theta(s_i, a_i)(R - V_\alpha(s_i))$
19:      Accumulate the critic gradient: $d_\alpha \leftarrow d_\alpha + \Delta_\alpha(R - V_\alpha(s_i))^2$
20:   end for
21:   Update the actor and the critic with the accumulated gradients: $\theta \leftarrow \theta + \mu d\theta$, $\alpha \leftarrow \alpha + \mu d\alpha$
22: end for
```

Figure 3.3: A2C execution process in UAV flight environment

4 Implementation

This chapter describes the Deep Reinforcement Learning of UAV as an agent, the construction of the flight environment, and how they interact.

4.1 Simulation Environment

The simulation used in this experiment is gym-pybullet-drones[46], a bybullet-based simulation software explicitly developed for quadrotor UAVs, which also integrates open AI and provides many advanced and popular reinforcement learning algorithms for the simulator. It provides an API interface for controlling the UAV's flight, as well as accessing the UAV's current position information, motor speed, and other data. Besides, it also supports user-defined environment layout, such as adding obstacles, collisions between UAV and obstacles, collision detection, and other functions. To facilitate support for reinforcement learning, it provides APIs for customising the settings of reward, state space, and action space. Gym-pybullet-drones define a frequency (Hz), which means the number of actions the UAV will perform in a second in the simulated environment.

4.2 classes and functions

As shown in Figure 4.1, the code structure of this experiment is divided into two main parts: (1) environment and (2) algorithm. Python 3.9 was used for this experiment, and stable_baseline3 was used as the support for the algorithm library. Each of these classes needs to implement its own different methods.

4.2.1 Environment

The environment section is composed of three files: flyWithSingalAviray, baseSingleAgentAviary, and baseAviary. flyWithSingalAviray is a custom environment that inherits from baseSingleAgentAviary and from baseAviary. This makes it easier to expand. The main functions of these three files are:

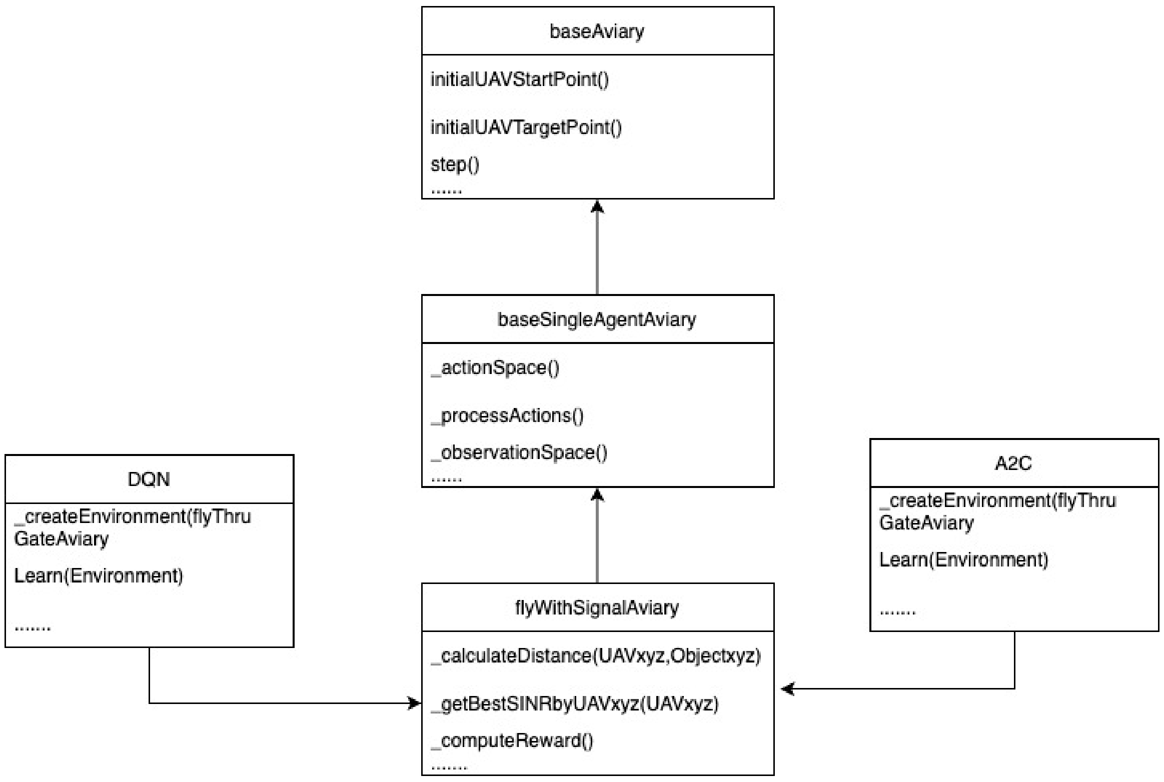


Figure 4.1: Class diagram for UAV flight

baseAviary

Its function is to record some basic information about the UAV and its environment, such as the initial position of the UAV, target point, gravity of environment, duration, clock frequency, coordinates of obstacles and base station, etc. The main methods are:

- step(action) Pass in the current action. First, the simulator executes this action to make the UAV fly to the target point. The environment transfers the state s_{t+1} according to this action and calculates the reward value and returns it. Then the observed state is recalculated.
- _housekeeping() This method is executed when the simulator ends the last episode and it resets all the information in the environment, such as the initial position of the drone, information about the buildings and the environment time.
- _getDroneStateVector() Composing a vector of current state information makes it easy for the caller to get the current state at any time.

baseSingleAgentAviary

This file defines the methods of the public part, such as the setting of the action space, passing the action to the console for execution, the definition of the state space, etc. The main methods are:

- _actionSpace() This method customizes different action spaces according to different algorithms. The A2C algorithm in this experiment belongs to the continuum output, directly defining the action space as a=[x,y], where x ∈ [-1,1] and y ∈ [-1,1]. We define DQN as discrete numbers for the action space, that is, 0 to 7. Eight numbers represent eight directions. As the simulator environment needs to input x and y axes as action inputs, we need to define a dictionary in Python to map numbers to actions. Details can be seen in Table 4.1. Columns x and y, respectively, represent the distance performed by an action on the axis. But this distance has to consider the environment's execution frequency and speed.

Table 4.1: DQN output and action mapping

action number	x	y
0	1	1
1	0	1
2	-1	1
3	-1	0
4	1	0
5	-1	-1
6	0	-1
7	1	-1

- _preprocessAction(action) This method really passes the action into the controller for execution, and it is called in step.
- _observationSpace() This method defines the state space and also supports the addition of custom states.
- _computeObs() which is used to calculate the state space and normalize the values of each state to facilitate algorithm calculation.

flyWithSingalAviray

User-defined environments, reward values, etc. are implemented here. The methods implemented are:

- _addObstacles() This method supports adding desired objects to the environment, such as adding blocks to simulate buildings and base stations.
- _computeReward() It calculates the reward value according to the formula in 3.3.3 and returns it to the agent. The agent judges the quality of the action according to

the size of the value, which affects the agent's path planning.

- _computeDone() Here the conditions are set for the UAV flight to jump out of the boundary, which can be set to accelerate the training by ending the training of the current episode early.
- _getBestSINRbyUAVxyz(UAVxyz) The parameters UAV coordinate points x, y, and z are required to calculate the SINR value received from the base station for the current UAV position according to the formula in 3.2. The base station position is added by the method _addObstacles(). The detailed parameters are shown in Table 4.2[44].

Table 4.2: signal model parameters

Parameters	Value
transmit power from BS ρ	40W
UAV antenna beamwidth θ	45°
UAV height λ	100m
BS height	20m
LoS pathloss exponent β_L	2.1
NLoS pathloss exponent β_N	4
Near-field pathloss c	-38.4dB
Noise power τ^2	$8 * 10^{-13}$W
BS antenna elements A_t	8

4.2.2 Algorithems

Both the A2C and DQN algorithms are implemented through stable_baseline3, and both need to pass in the created environment to achieve interaction.

Some important parameters of A2C are:

- environment
- learning rate α
- discount factor γ
- entropy coefficient e

The Entropy coefficient is used to adjust the randomness of A2C actions.

Some important parameters of DQN are:

- environment
- learning rate α

- discount factor γ
- buffer size M
- batch size S
- learning starts L
- exploration fraction E
- target update interval T

Both algorithms call the learn() method to learn the environment, passing in a time.

4.3 Summary

In this chapter, we first introduce the simulator used for this experiment. Next, we describe the creation of the custom environment and the functions implemented. Finally, the implementation of the algorithm in the stable_baseline3 used for the agent is presented.

5 Evaluation

In this chapter, we will evaluate the performance of the A2C algorithm in connectivity-aware trajectory planning and use DQN as a comparison. First, we introduce the environment of the simulation experiment and the hyperparameters used by the two algorithms. It also shows the paths of the two algorithms in the simulated environment and the metrics to measure the experiments. Finally, the results are analyzed.

5.1 Metrics

- Trajactory It can be measured in two aspects. The first aspect is the intuitive performance of the trajectory. The second aspect is the distance of the trajectory. Because the A2C algorithm is significantly larger than the DQN in the number and range of actions. Trajectories may appear smoother and distances may be shorter.
- SINR(dB) It can be measured from the average SINR performance of the UAV during the flight, the highest SINR value and the lowest value.

5.2 Environment

The scene parameters of this experiment are shown in the table. The experimental range is 500m x 500m, the UAV flight height is 100m, the base station height is 20m, and the building height is 80m to 110m. We assume that the UAV can automatically connect to a BS with a good signal during the flight We set 3 starting points for comparison and set a The endpoint. If the UAV reaches the endpoint within a certain range, the flight ends. In addition, the SINRdB signal heat map is added, and the parameters are shown in the table 4.2. The final scene is shown in Figure 5.1.

Table 5.1: UAV flight environment parameters

Parameters	Value
Environmental range	500m x 500m
UAV flight height	100m
BS height	20m
UAV speed	30m/s
BS height	20m
Buildings height	80m to 110m
Number of drones	1
number of buildings	3
Number of BS	3
starting points	1:[x=0m,y=0m] 2:[x=70m,y=0m] 3:[x=200m,y=0m]
end position range	[x=250m,y=500m], Radius of 50m

5.3 Algorithem Hyper-parameters

The hyperparameters used by A2C are shown in Table 5.2.
Among them, our reward value setting is more inclined to the current action, so the discount factor is set to 0.9. The entropy coefficient is set to 0.0001 to make the algorithm explore new actions to a certain extent, and the default value is set to 0 to not explore.

Table 5.2: hyperparameters of A2C

Parameters	Value
learning rate α	0.0007
discount factor γ	0.9
entropy coefficient e	0.0001

The hyperparameters used by DQN are shown in Table 5.3
Because the reward function strategy is the same as in A2C, the discount factor is also 0.9. To ensure a specific action exploration, the exploration fraction is set to 0.2. Learning starts is set to 5000 to start training after collecting 5000 steps of data.

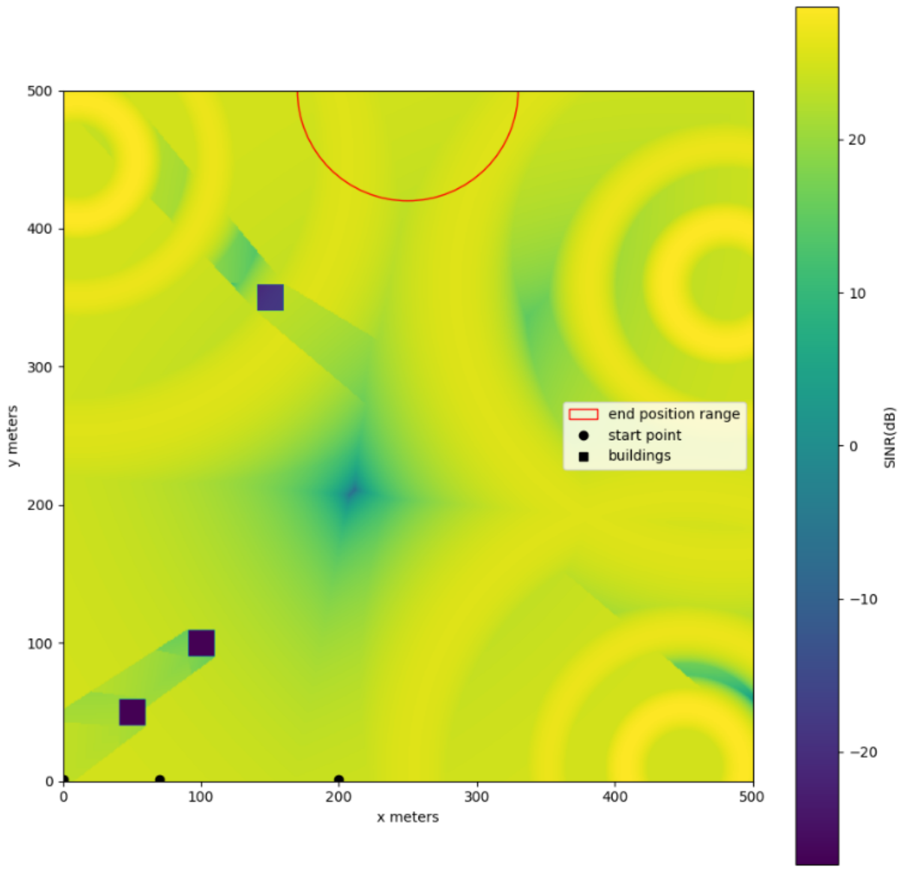


Figure 5.1: flight environment

Table 5.3: hyperparameters of DQN

Parameters	Value
learning rate α	0.0001
discount factor γ	0.9
buffer size M	1000000
batch size S	32
learning starts L	5000
exploration fraction E	0.2
target update interval T	10000

5.4 Results and Analysis

We use three starting points to test the connectivity-aware path planning problem of the UAV. [0 m, 0 m], [70 m, 0 m], and [200 m, 0 m] are the three starting points.
Figures 5.2, 5.3, and Table 5.4 show the comparison of the two algorithms with a starting point of [0m, 0m]. We can see that the average SINR of the A2C algorithm is significantly

better than that of the DQN algorithm during the flight. They all got about the same best signal, while the A2C algorithm sensed a poorer signal. The A2C algorithm flies a relatively small distance and reaches the end point range. As can be seen from Figure 5.2, the DQN algorithm flies the broken line. Because the action is fixed, it loses a lot of actions that can save distance.
Figures 5.4, 5.5, and Table 5.5 show the trajectories and results of the two algorithms at the starting point [70m, 0m]. Similar to the starting point [0,0], the average SINR value of UAV under the A2C algorithm is significantly higher than that of the DQN algorithm, but this The second flight is longer, which may be because the UAV has learned the vector of signal strength, which can be seen from Figure 5.4, the last segment of the A2C path.
As can be seen from Figures 5.6, 5.7 and Table 5.6, for the path from the starting point [200m, 0m], the A2C performance is better than the DQN algorithm, except that the best signals obtained are equal.

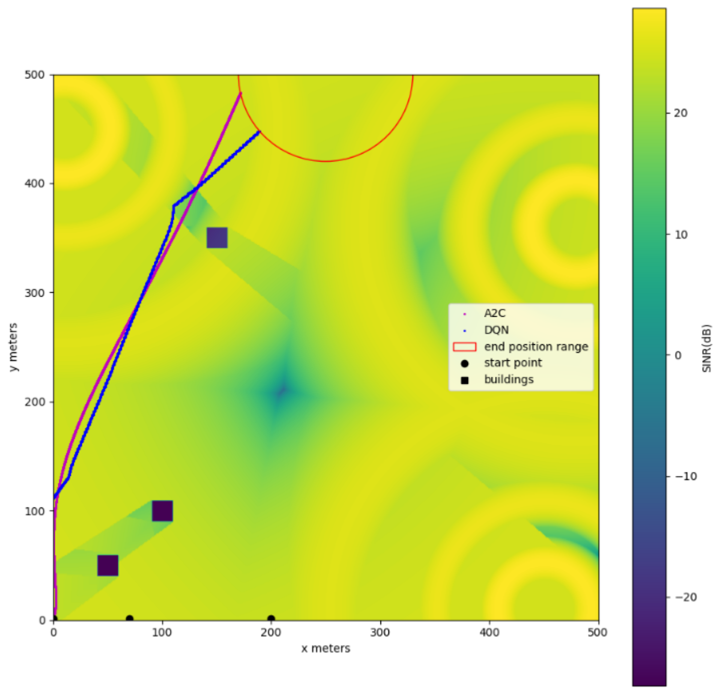


Figure 5.2: DQN and A2C paths are compared at the starting point [0, 0]

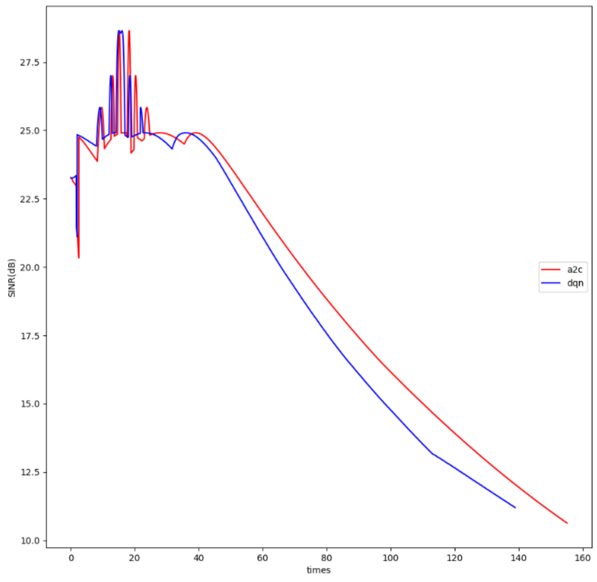


Figure 5.3: The strength of SINR (dB) is compared between DQN and A2C during UAV flight, the starting point is [0, 0]

Table 5.4: Result analysis for the starting point [0,0]

Results	A2C	DQN
average SINR(dB)	19.02	17.19
best SINR(dB)	28.65	28.65
worst SINR(dB)	10.64	11.21
flight distance	521m	529.3m

Table 5.5: Result analysis for the starting point [70,0]

Results	A2C	DQN
average SINR(dB)	18.89	15.29
best SINR(dB)	24.91	24.91
worst SINR(dB)	10.95	11.37
flight distance	489.9m	472.2m

Table 5.6: Result analysis for the starting point [200,0]

Results	A2C	DQN
average SINR(dB)	15.00	12.23
best SINR(dB)	20.08	20.08
worst SINR(dB)	11.39	10.12
flight distance	480.8	487.1m

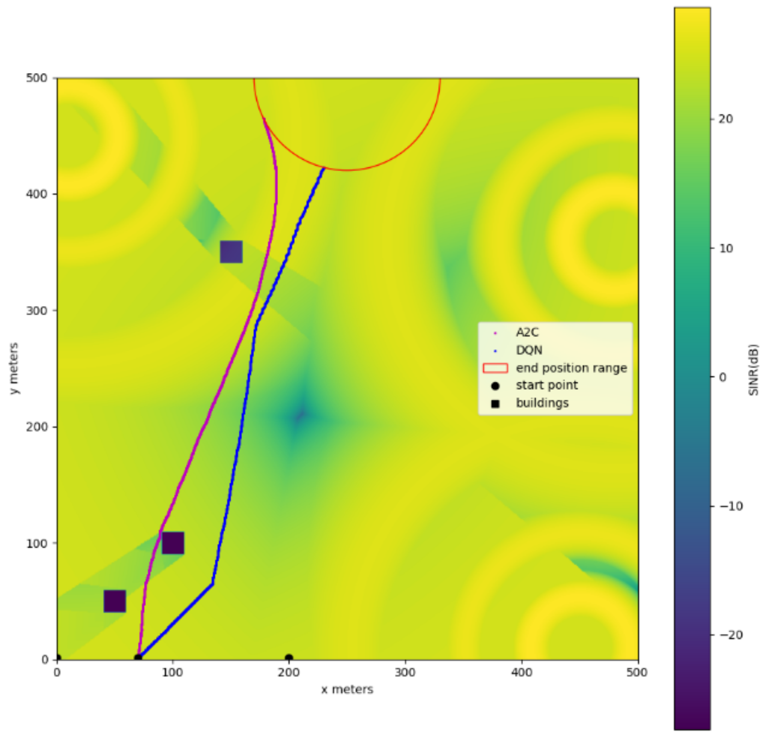


Figure 5.4: DQN and A2C paths are compared at the starting point [70, 0]

5.5 Summary

We describe in this chapter the experimental setting, the metrics used to measure the results, and the hyperparameters used by the A2C and DQN algorithms. The experimental results show that the A2C algorithm maintains a significantly stronger signal strength than the DQN algorithm during the entire flight of the UAV.

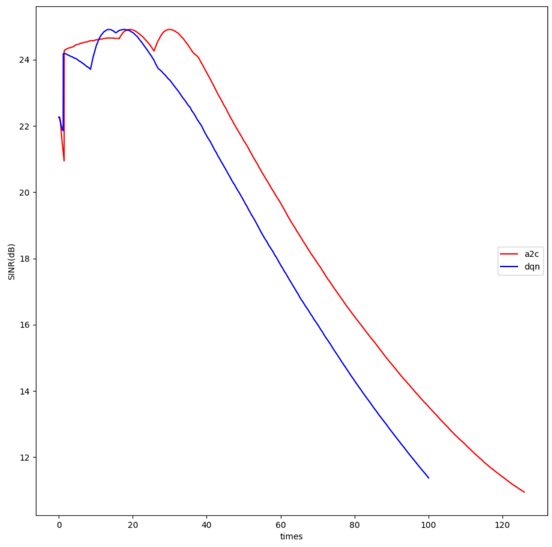


Figure 5.5: The strength of SINR (dB) is compared between DQN and A2C during UAV flight, the starting point is [70, 0]

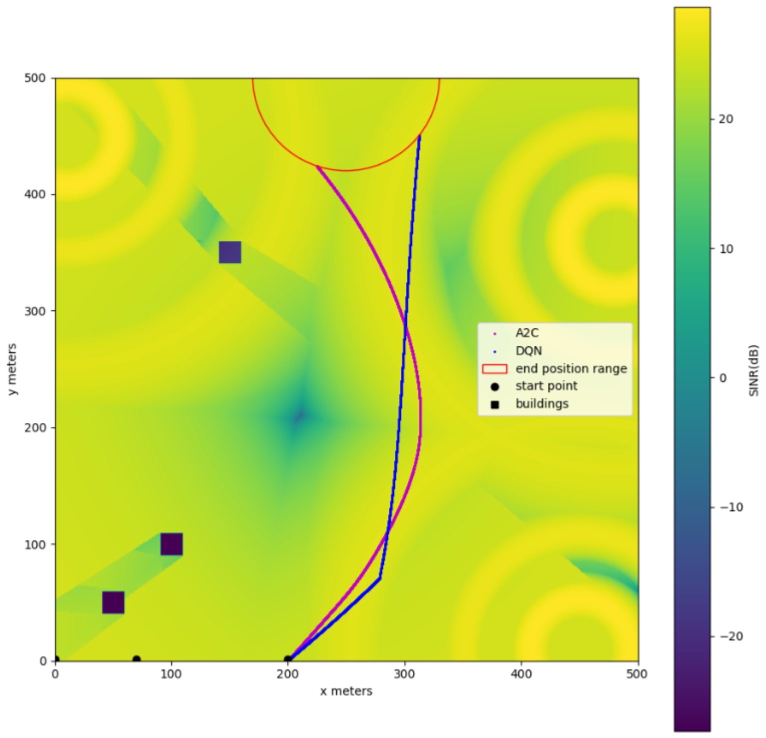


Figure 5.6: DQN and A2C paths are compared at the starting point [200, 0]

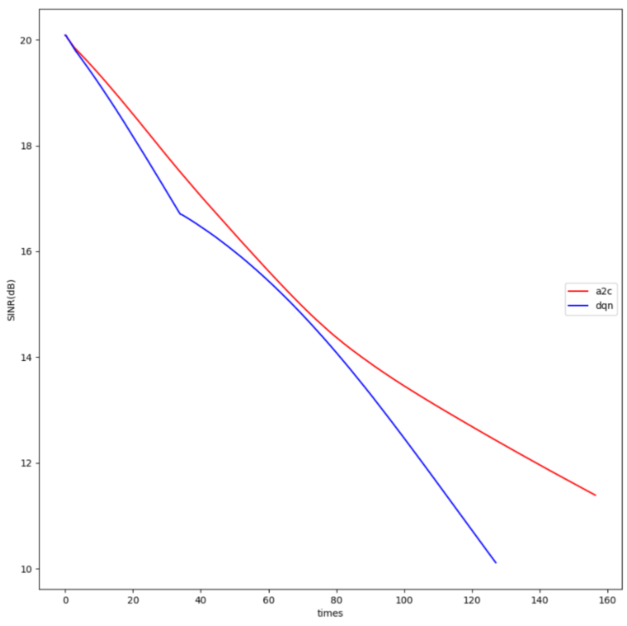


Figure 5.7: The strength of SINR (dB) is compared between DQN and A2C during UAV flight, the starting point is [200, 0]

6 Conclusions and Future Work

In this chapter, we summarize the entire content of this work, and then look forward to future work.

6.1 Conclusion

Chapter 1 introduces the current environment of UAV, the motivation, goals, and contributions behind our work.

Chapter 2 details the background of the DQN and A2C technologies we used, as well as UAV's work related to path planning, connectivity, and deep reinforcement learning for connectivity and connected path planning.

In Chapter 3, we design the experiments, including the experimental environment design, action space, state space, and reward design of the deep reinforcement learning algorithms.

Chapter 4 describes the simulation tools we used, the specific code to be implemented, and the hyperparameters needed to implement deep reinforcement learning.

In Chapter 5 we specify the various parameters and hyperparameters used for the experiments, including the environment and the algorithm. Finally, we demonstrate through experimental data that the A2C algorithm is superior to DQN in connectivity-aware path planning.

6.2 Future Work

It can also be seen from the results of 5.4 that the A2C algorithm has some shortcomings in the accuracy of reaching the endpoint, and the trajectory is not perfect. In future work, we will continue to work on UAV connectivity trajectory optimization. Study the optimization problem of the A2C algorithm, explore the inadequacies of the A2C algorithm, and try to use more advanced deep reinforcement learning algorithms.

Bibliography

[1] S. Bhagat and P. B. Sujit, "UAV Target Tracking in Urban Environments Using Deep Reinforcement Learning," 2020 International Conference on Unmanned Aircraft Systems (ICUAS), 2020, pp. 694-701, doi: 10.1109/ICUAS48674.2020.9213856.

[2] Xiaofei Wang, Hui Zhao, Tong Han, Huan Zhou, Cong Li, A grey wolf optimizer using Gaussian estimation of distribution and its application in the multi-UAV multi-target urban tracking problem, Applied Soft Computing, Volume 78, 2019, Pages 240-260, ISSN 1568-4946.

[3] Butilă, E.V. and Boboc, R.G., 2022. Urban Traffic Monitoring and Analysis Using Unmanned Aerial Vehicles (UAVs): A Systematic Literature Review. Remote Sensing, 14(3), p.620.

[4] Bu, T., Zhu, J. and Ma, T., 2022. A UAV Photography–Based Detection Method for Defective Road Marking. Journal of Performance of Constructed Facilities, 36(5), p.04022035.

[5] Wang, J., Wang, G., Hu, X., Luo, H. and Xu, H., 2020. Cooperative transmission tower inspection with a vehicle and a UAV in urban areas. Energies, 13(2), p.326. Islam, A. and Shin, S.Y., 2019, July. BHMUS: blockchain based

[6] secure outdoor health monitoring scheme using UAV in smart city. In 2019 7th international conference on information and communication technology (ICoICT) (pp. 1-6). IEEE.

[7] Zhao, Y., Wang, T., Zhang, S. and Wang, Y., 2020. Towards minimum code dissemination delay through UAV joint vehicles for smart city. IET Communications, 14(15), pp.2442-2452.

[8] Sharma, R. and Arya, R., 2022. UAV based long range environment monitoring system with Industry 5.0 perspectives for smart city infrastructure. Computers & Industrial Engineering, 168, p.108066.

[9] Zhang, G. and Hsu, L.T., 2019. A new path planning algorithm using a GNSS localization error map for UAVs in an urban area. Journal of Intelligent & Robotic Systems, 94(1), pp.219-235.

[10] Delamer, J.A., Watanabe, Y. and Chanel, C.P., 2021. Safe path planning for UAV urban operation under GNSS signal occlusion risk. Robotics and Autonomous Systems, 142, p.103800.

[11] Liu, L., Tian, B., Zhao, X. and Zong, Q., 2019, August. UAV autonomous trajectory planning in target tracking tasks via a DQN approach. In 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR) (pp. 277-282). IEEE.

[12] Zhang, J., Yu, Y., Wang, Z., Ao, S., Tang, J., Zhang, X. and Wong, K.K., 2020, August. Trajectory planning of UAV in wireless powered IoT system based on deep reinforcement learning. In 2020 IEEE/CIC International Conference on Communications in China (ICCC) (pp. 645-650). IEEE.

[13] Wang, L., Wang, K., Pan, C., Chen, X. and Aslam, N., 2020. Deep Q-Network Based Dynamic Trajectory Design for UAV-Aided Emergency Communications. Journal of Communications and Information Networks, 5(4), pp.393-402.

[14] Sutton, R.S., McAllester, D., Singh, S. and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12.

[15] Witten, I.H., 1977. An adaptive optimal controller for discrete-time Markov environments. Information and control, 34(4), pp.286-295.

[16] Degris, T., White, M. and Sutton, R.S., 2012. Off-policy actor-critic. arXiv preprint arXiv:1205.4839.

[17] Deng, L. and Yu, D., 2014. Deep learning: methods and applications. Foundations and trends® in signal processing, 7(3–4), pp.197-387.

[18] Carrio, A., Sampedro, C., Rodriguez-Ramos, A. and Campoy, P., 2017. A review of deep learning methods and applications for unmanned aerial vehicles. Journal of Sensors, 2017.

[19] Lillicrap, T.P., Santoro, A., Marris, L., Akerman, C.J. and Hinton, G., 2020. Backpropagation and the brain. Nature Reviews Neuroscience, 21(6), pp.335-346.

[20] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J. and Chen, T., 2018. Recent advances in convolutional neural networks. Pattern recognition, 77, pp.354-377.

[21] Frid, A., Kantor, A., Svechin, D. and Manevitz, L.M., 2016, November. Diagnosis of Parkinson's disease from continuous speech using deep convolutional networks without manual selection of features. In 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE) (pp. 1-4). IEEE.

[22] Chen, Q., Xu, J. and Koltun, V., 2017. Fast image processing with fully-convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 2497-2506).

[23] Fukushima, K., 1979. Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron. IEICE Technical Report, A, 62(10), pp.658-665.

[24] Waibel, A., 1987. Phoneme recognition using time-delay neural networks. Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE). Tokyo, Japan.

[25] Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[26] Duchi, J., Hazan, E. and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[27] Hinton, G., Srivastava, N. and Swersky, K., 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 14(8), p.2.

[28] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[29] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. nature, 518(7540), pp.529-533.

[30] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016, June. Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

[31] Eom, S., Lee, H., Park, J. and Lee, I., 2019. UAV-aided wireless communication designs with propulsion energy limitations. IEEE Transactions on Vehicular Technology, 69(1), pp.651-662.

[32] Natalizio, E., Zema, N.R., Di Puglia Pugliese, L. and Guerriero, F., 2019, November. Download and fly: An online solution for the UAV 3D trajectory planning problem in smart cities. In Proceedings of the 9th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications (pp. 49-56).

[33] Yang, H., Ruby, R., Pham, Q.V. and Wu, K., 2021. Aiding a disaster spot via multi-UAV-based IoT networks: energy and mission completion time-aware trajectory optimization. IEEE Internet of Things Journal, 9(8), pp.5853-5867.

[34] Cherif, N., Jaafar, W., Yanikomeroglu, H. and Yongacoglu, A., 2021, June. Disconnectivity-Aware Energy-Efficient Cargo-UAV Trajectory Planning with Minimum Handoffs. In ICC 2021-IEEE International Conference on Communications (pp. 1-6). IEEE.

[35] SILVIRIANTI, S. and Shin, S.Y., 2022. Energy Efficient Multi-dimensional Trajectory of UAV-aided IoT Networks with Reinforcement Learning.

[36] Hoseini, S.A., Bokani, A., Hassan, J., Salehi, S. and Kanhere, S.S., 2021, January. Energy and service-priority aware trajectory design for UAV-BSs using double Q-learning. In 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC) (pp. 1-4). IEEE.

[37] Chowdhury, M.M.U., Saad, W. and Güvenç, I., 2020, June. Mobility management for cellular-connected UAVs: A learning-based approach. In 2020 IEEE international conference on communications workshops (ICC Workshops) (pp. 1-6). IEEE.

[38] Zeng, Y. and Xu, X., 2019, December. Path design for cellular-connected UAV with reinforcement learning. In 2019 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

[39] Yang, H., Zhang, J., Song, S.H. and Lataief, K.B., 2019, April. Connectivity-aware UAV path planning with aerial coverage maps. In 2019 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-6). IEEE.

[40] Fonseca, E., Galkin, B., DaSilva, L.A. and Dusparic, I., 2020. Adaptive height optimisation for cellular-connected UAVs using reinforcement learning. arXiv preprint arXiv:2007.13695.

[41] Xie, H., Yang, D., Xiao, L. and Lyu, J., 2021. Connectivity-aware 3D UAV path design with deep reinforcement learning. IEEE Transactions on Vehicular Technology, 70(12), pp.13022-13034.

[42] Chen, Y.J. and Huang, D.Y., 2021. Joint Trajectory Design and BS Association for Cellular-Connected UAV: An Imitation-Augmented Deep Reinforcement Learning Approach. IEEE Internet of Things Journal, 9(4), pp.2843-2858.

[43] Zhan, C. and Zeng, Y., 2022. Energy Minimization for Cellular-Connected UAV: From Optimization to Deep Reinforcement Learning. IEEE Transactions on Wireless Communications.

[44] Galkin, B., Fonseca, E., Amer, R., DaSilva, L.A. and Dusparic, I., 2021. Reqiba: Regression and deep q-learning for intelligent uav cellular user to base station association. IEEE Transactions on Vehicular Technology, 71(1), pp.5-20.

[45] Błaszczyszyn, B. and Keeler, H.P., 2015. Studying the SINR process of the typical user in Poisson networks using its factorial moment measures. IEEE Transactions on Information Theory, 61(12), pp.6774-6794.

[46] Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A. and Schoellig, A.P., 2021, March. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 7512-7519). IEEE.