REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

UAV-BASED CELLULAR COMMUNICATION: DEEP REINFORCEMENT LEARNING FOR INTERFERENCE MANAGEMENT

17011044 — Uğur Keskin

SENIOR PROJECT

Advisor
Assist. Prof. Dr. Ferkan YILMAZ

June, 2021

ACKNOWLEDGEMENTS

I would like to thank to project's supervisor, Assoc. Prof. Ferkan Yilmaz, for his support and guidance during project. I would also like to thank my parents for all the kindness and support, making this work much easier.

Uğur Keskin

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DQN	Deep Q Network
DQL	Deep Q Learning
MADQL	Multi-Agent Deep Q Learning
DDQN	Double Deep Q Network
DDQL	Double Deep Q Learning
MADDQL	Multi-Agent Double Deep Q Learning
MADRL	Multi-Agent Deep Reinforcement Learning
NN	Neural Network
RL	Reinforcement Learning
FDMA	Frequency-Division Multiple Access

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

UAV-BASED CELLULAR COMMUNICATION: DEEP REINFORCEMENT LEARNING FOR INTERFERENCE MANAGEMENT

Uğur Keskin

Department of Computer Engineering
Senior Project

Advisor: Assist. Prof. Dr. Ferkan YILMAZ

In this article, an artificial intelligence-assisted multi-agent trajectory planning problem is discussed by considering the locations of the users they serve on the ground and the interference caused by the unmanned aerial vehicles to each other. Unlike previous works, this project offers a scalable solution to multiple environments that fits a certain distribution.

Since we wanted to reach a scalable solution for multiple environments, dynamic programming algorithms, Q Learning or greedy method would not give an optimal result due to the complexity of the problem so we tried to solve this problem using multi-agent deep reinforcement learning. To solve this problem, we used two different learning methods, cooperative and independent, based on the multi-agent deep Q learning algorithm.

Keywords: Path planning, trajectory Optimization, unmanned aerial vehicles, multi-agent deep reinforcement learning, deep q networks, cellular communication

ÖZET

İHA TABANLI HÜCRESEL HABERLEŞME: İNTERFERANS YÖNETİMİ İÇİN DERİN PEKİŞTİRMELİ ÖĞRENME

Uğur Keskin

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr. Ögr. Üyesi Ferkan YILMAZ

Bu makalede, birden fazla İnsansız hava aracının, yeryüzündeki servis sağladıkları kullanıcıların konumlarını ve haberleşme kısmında insansız hava araçlarının birbirlerine yol açtığı interferansı göz önünde bulundurarak yapay zeka destekli bir güzergah planlama problemi ele alınmıştır. Bu proje, önceki çalışmalardan farklı olarak belirli dağılımdaki birden fazla çevreye birden ölçeklenebilir bir çözüm sunmaktadır.

Birden fazla çevre için ölçeklenebilir bir çözüme ulaşmak istediğimizden ve problemin karmaşıklığından dolayı dinamik programlama algoritmaları, Q learning veya greedy yöntem optimal bir sonuç vermeyeceğinden dolayı bu problemi çoklu ajanlı pekiştirmeli öğrenme ile çözmeye çalıştık. Bu problemi çözmede çoklu ajanlı derin Q öğrenimi algoritmasını baz alan kooperatif ve bağımsız olmak üzere 2 farklı öğrenme yöntemi kullandık.

Anahtar Kelimeler: Güzergah planlama, insansız hava araçları, çoklu ajanlı derin pekiştirmeli öğrenme, derin Q ağları, hücresel haberleşme

1
Introduction

Unmanned aerial vehicles (UAVs) are highly expected to have crucial roles in wireless communication systems due to their advantages over conventional communication systems. They can provide a wireless communication in an efficient way for numerous practical scenarios due to the fact that unmanned aerial vehicles can move flexibly and freely in space and can easily provide LoS (Line of Sight) communication directly with the user equipments on the ground. Also UAVs can provide an effective wireless connection without needing any network infrastructure and their coverage don't suffer from the effects of tall buildings and disasters, unlike traditional base station systems so that UAVs can be trusted as aerial base-stations to complement and support existing communication infrastructure.

Although UAVs can provide an effective wireless connection with high-range coverage, taking advantage of this feature on a wide-scale deployment requires to solve a highly complex interference management problem. Interference in connection between a user equipment and a UAV serving it, is caused by other UAVs nearby, which is an important factor that UAVs should consider when planning their own trajectories. Moreover, UAVs must take into account different factors like user locations, user movements and area boundaries. Eventually, trajectory optimization with interference management is a challenging problem to address it by traditional algorithms such as greedy methods or dynamic programming.

In this paper, to perform a trajectory planning for UAVs as an aerial base stations while dealing problems such as interference management and transmit rate optimization, Multi-Agent Deep Reinforcement Learning is used. To perform Multi-Agent Deep Reinforcement Learning, a simulation which consists of multiple UAVs as DRL agents and an environment have developed. Environment contains a system model which consists of mobility and transmission models to calculate transmit rates, rewards and states to meet the fundamental needs of reinforcement learning and provide agents to train properly for planning their trajectory optimally even on the environment with different user locations.

2
Literature Review

In this section, previous works related to topics such as UAV trajectory optimization, interference optimization, reinforcement learning approach for UAV trajectory optimization and multi-agent deep reinforcement learning for determining UAV trajectory optimization are examined. In previous researches, reward function, input state for reinforcement learning agent(s), action space of agent(s) and system model is examined in detailed.

2.1 Previous Works

In research [1], multi-agent deep reinforcement learning approach is used to maximize energy efficiency and minimize both interference level and wireless latency caused on the ground network in UAV's path. in research, UAVs' action spaces consists of selecting its transmission level in a discrete way, base station association network and movement direction with fixed altitude. Environment on this research is a square shaped area that consist of discretized area units and UAVs assumed to move with fixed velocity and move along the center of areas so the time required by each UAV to travel between any two unit areas is constant. Terminal state of an UAV in reinforcement learning algorithm in the research is the state when UAV is reached its pre-determined destination. Input state of agents consists of horizontal locations of all agents (UAVs), distance and orientation angle between it and its nearest base stations and orientation angle between agent and its destination. Also an Deep Echo State Network which is a computationally efficient RNN network is used as a deep learning algorithm.

In research [2], unlike [1], situations such as collusions between Uavs and Uavs flying out of environment is not ignored. In order to solve this, agents does not allowed to perform their actions that cause these extreme situations and these actions reflected as penalty in the reward value. In [2], multi agent deep deterministic policy gradient (MADDPG) method due to the continuous action spaces which is a multi agent deep reinforcement learning is applied to jointly maximize the geographical fairness

among the UEs covered, the fairness of UE-load of each UAV, while minimizing the overall energy consumption of UEs by optimizing each UAV's trajectory and offloading decisions. Input state of each UAV consists of horizontal location of itself, distance of other UAVs, served user count and user count rate which serving by UAV itself; Action space consists of flying direction with fixed altitude and flying distance; Reward function is formulated considering geographical fairness and UAV-user fairness index.

In research [3], 3 algorithm used to solve UAV trajectory plan and power plan problems to optimize sum rate of users. First algorithm is a multi-agent q-Learning algorithm which aims to determine the optimal initial deployment location in 3D (x,y,z). Second algorithm is an Echo State Network algorithm used to predict movement of users from a real life data which consists of geographical informations of users and data is collected using Twitter's application programming interface (API). Third algorithm is a multi-agent Q-learning algorithm that used to determine power control and action movement that maximizes total sum rates of users. In proposed environment and system model in this research, reward function returns 1 if UAV's action increases total sum rate of system and returns -1 if not; agents' entire action spaces are consist of 21 element determined by 7 directions (altitude is not fixed unlike the other 2 studies mentioned above) and 3 power levels; An agent's input state consists of its 3 dimensional location and power transmission level. Also in [4], agents have discrete action space and uses a Q-learning algorithm instead of a deep learning method.

In this research, we aimed to maximize overall sum rate of system to be scaled to multiple environments generated with a certain distribution, by optimizing each agent's own networks. In order to do this, we used multi-agent deep reinforcement learning (MADRL) approach with 2 different training methods. Formulation of transmission model on this research is derived from [3]. Reward function, user movement model and state input structure is formulated differently from all researches mentioned above.

3
Feasibility Study

In this section, feasibility studies are mentioned in order to determine the requirements of the project. In the technical feasibility part, Software and Hardware Feasibility was mentioned. The software environments, libraries and tools used in Software Feasibility, the hardware requirements for the system in Hardware Feasibility, the licenses required by the project in Legal Feasibility, the time required for the realization of the project in Time Feasibility, and the costs required for system implementation in economic feasibility.

3.1 Technical Feasibility

Technical Feasibility, Software Feasibility and Hardware Feasibility are explained in 2 separate sections.

3.1.1 Software Feasibility

During the development of the project, Python programming language was preferred due to its high reusability, resource abundance, the convenience of artificial intelligence libraries and its easy readability. Keras library supported by Tensorflow and Tensorflow has been used to create, train and evaluate Deep Learning models in a more understandable way.

3.1.2 Hardware Feasibility

Since training a single neural network is a time-consuming, demanding and costly process, a high amount of processing power is required. Considering that simulation consist of multiple agents and each of agents have their own memory and network, system may need up to 6 hours for just one experiment. The system required for the training process to run is shown in Table 3.1 .

CPU	İntelCore i7-7700 HQ
GPU	Nvidia GTX 1050 Ti
Operating System	Windows 10
RAM	20 GB

Table 3.1 Minimum Hardware Requirements

Developer Cost	24000 TL
Hardware Cost	15000 TL
Total Cost	39000 TL

Table 3.2 Costs

3.2 Legal Feasibility

All frameworks and software tools used in this project are open source and free for commercial use.

Keras library is licensed under The MIT License (MIT)[5].

Tensorflow Library is licensed under The Apache License 2.0[6].

3.3 Economic Feasibility

The cost spent in training RL agents, simulating the environment and installing the system is as in the table 3.2.

3.4 Time Feasibility

The project is planned to be completed in the spring semester. Since the project is a one-person project, there is no distribution of tasks. The gant diagram is shown in figure 3.1.

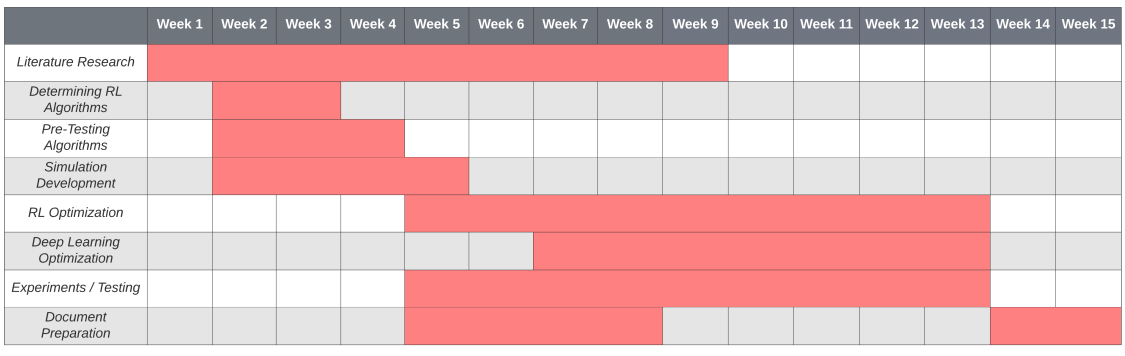

Figure 3.1 Gant Diagram

4
System Analysis

In this section, the entities that the system needs, the working principle of environment, the method used in the training of agents and system models are explained.

4.1 Reinforcement Learning

Reinforcement learning (RL) is a machine learning area about how smart agents should act in an environment to maximize the cumulative reward which determined by the environment and problem. Reinforcement learning is one of the three basic machine learning paradigms as well as supervised learning and unsupervised learning. unlike other two machine learning paradigms, reinforcement learning does not need pre-labeled data on neither training nor testing phases. Instead, it aims to optimize the policy function which determines agent's actions according to the current state, to make agent obtain the optimal reward.

In order to optimize agent's policy, agent experience various states by acting randomly (exploring) or acting according to its policy and stores a tuple which consist of state S, the action A in state S, the reward R obtained by agent from environment as a result of action decision A on state S, next state N after taking action A in state S and a boolean T whose value is true if agent reaches a terminal state, by interacting with environment. In figure 4.1, interaction between agent and environment is visualized basically.

Deciding between taking action by exploration or exploitation is a common trade-off problem in reinforcement learning. Without exploration, agent doesn't discover new states enough to optimize its reward which makes agent unaware of opportunities. On the other hand, exploitation is also necessary for agent to use its experiences which is required to reach optimal reward. In order to solve that problem, approaches such as "epsilon greedy" are considered for determining decision between exploration and

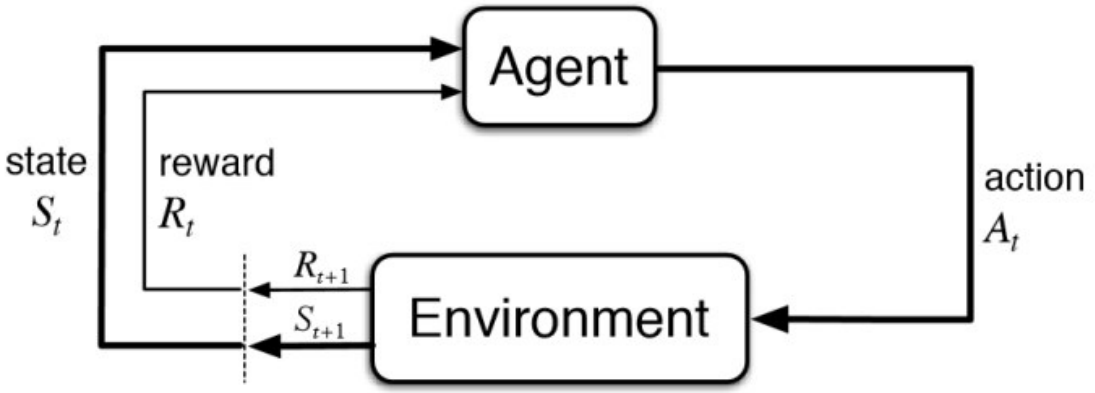

Figure 4.1 Agent-Environment Relation

exploitation randomly in many different ways.

4.1.1 Multi-Agent Reinforcement Learning

Considering real-world problems with high complexity that require multiple entities in the same environment to solve or reach a more optimal solution, it is appropriate to use multi-agent reinforcement learning approach. In this approach, multiple agents in same environment can be cooperative, competitive or mixed.

Due to multiple agents being in the same environment, reinforcement learning with multiple agents has many difficult problems compared to reinforcement learning with a single agent such as the challenge is that the environment is non-stationary since the actions of the agents in current state affect each other and all of the agents may behave differently from previous trials [7]. In order to get rid of the non-stationary problem, the state information to be given per step to each UAVs should cover the environment state in a way that does not cause any ambiguity in the absolute Q function, so UAVs should not be able to get multiple different reward values from same state information and same action [8]. Mentioned situation causes agents to have to make the optimal action about more states than usual in a more dynamic environment which means agents in multi-agent reinforcement learning have more workload than single agent in the basic reinforcement learning.

4.2 Q-Learning

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. By using a Q-table which is a state-action value matrix that contains the value of taking an action A in state S, agent determines its policy for the state it is in. In training phase of Q-Learning, Q-table is getting updated as agent

experiences many different states and results with its actions, to the extent that the agent can make the right decision, considering the future rewards[9]. The fact that the q-learning algorithm is based on predicting the future reward until the end makes it more successful than the greedy method in complex problems.

Formula for Q-value expressed as:

$$NewQ(s,a) = Q(s,a) + \alpha\left[R(s,a) + \gamma \max Q'\left(s',a'\right) - Q(s,a)\right]$$

where α is learning rate, R(s,a) is the reward, γ is the constant called discount rate, $\max Q'(s',a')$ is the expected future reward.

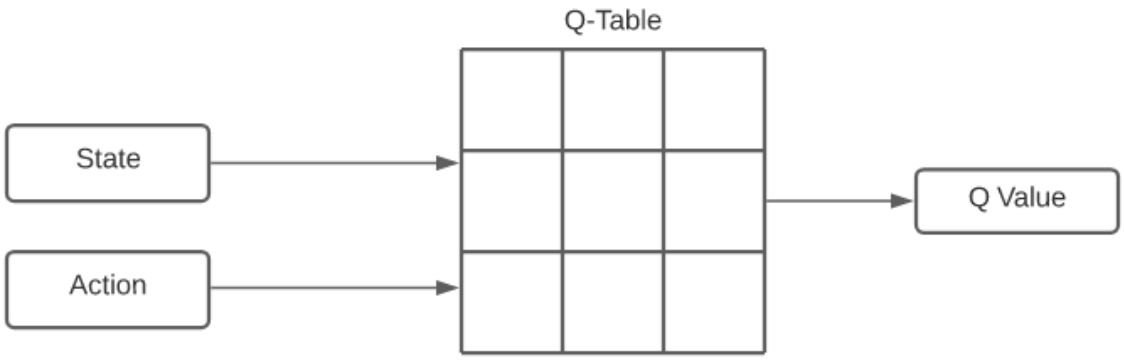

Figure 4.2 In Q-learning, the value of Q is obtained from the Q-Table by giving action and state

4.2.1 Deep Q Learning

Since environments with large state space can cause a serious resource problem in Q-Learning by requires too much space for q-table, basic q-learning can only be used in problems with limited state space. To get rid of resource problem and improve learning performance, using a neural network to estimate action-state values instead of a Q-Table is one of the solution for it[10].

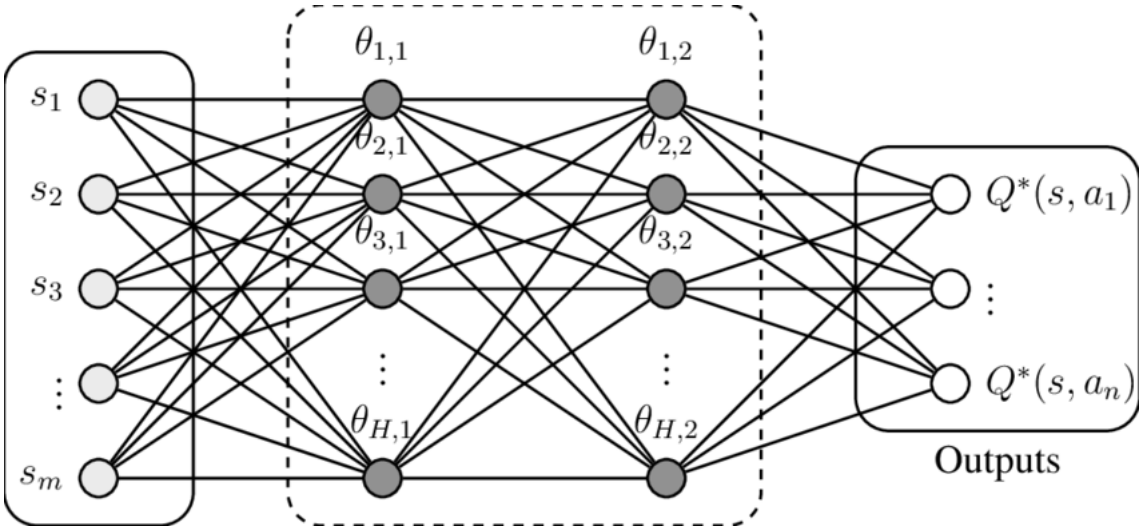

Figure 4.3 Deep Q Network uses a neural network to map state-action pairs to corresponding Q value instead of a Q-Table

Deep Q-Learning is an approach that takes advantage of both deep learning and q-learning. In Basic Q-Learning, a table maps each state-action pair to its corresponding Q-value. In deep q-learning, instead of a table, a neural network maps input states to the action with Q values which makes deep Q-Learning more appropriate in complex environments which have high dimensional discrete or continuous spaces.

4.2.1.1 Double Deep Q Network

Due to the popular Deep Q Network, overestimating the action values in many states, Double Q Network decouples action selection and value estimation by using two Q Networks instead of single one. In basic Deep Q-Learning algorithm, single Q network both selects action and estimates value of future steps and that causes network to estimate value overoptimistic [11]. Depending on the environment, double Q-learning generally gets higher performance against basic Q-learning.

5
System Design

In this chapter, system model, the developed multi-agent environment and applied reinforcement learning algorithms are explained in detail,

5.1 System Model

In system model, the downlink of UAV supported wireless communication networks are considered. Multiple UAVs serves users on the ground as aerial base stations. Each user communicates with using their user equipments (UE) with UAV and gets served by nearest UAV dynamically. UAVs assumed to be connected to a network by satellite and each UAV communicates simultaneously with users served by themselves by using FDMA system. Also in order to solve energy problem of UAVs, their energy assumed to supplied by using laser charging from a power base station which will not be detailed in the problem formulation.

5.1.1 Mobility Model

For the purpose of keeping environment more realistic, users are able to move during the flying period of UAVs. Because the UAVs are faster than the users and to ensure the continuity of the simulation defined in a discrete way, the movement period of the users is fixed according to the user-UAV velocity ratio. All users have action vector consists of moving right, left, forward, back and remaining static with same length but with a different sequence. Paths users follow is predetermined before simulation randomly by using a random distribution on each users' own action vector so that environment's user mobility model is stationary and accurate. UAVs moves

5.1.2 Transmission Model

In transmission model, formulations described on [3] are used. The Overall achievable sum rate at time unit t can be described as

$$R_{\text{sum}} = \sum_{n=1}^{|N|} \sum_{k_n=1}^{K_n} r_{k_n}(t)$$

where N is set of UAVs, K_n is the total user count which serving by UAV n and $r_{kn}(t)$ is the instantaneous achievable rate of user k_n at time t which is expressed by bps/Hz and expressed as

$$r_{k_n}(t) = B_{k_n} \log_2\left(1 + \Gamma_{k_n}(t)\right)$$

where B_{kn} is the bandwidth of user k_n and $\Gamma_{k_n}(t)$ is the signal-to-noise-plus-interference ratio (SINR) of user at time t.

SINR is a quantity used to give upper bounds on channel capacity in wireless communication systems, theoretically and in our problem, it can be expressed as

$$\Gamma_{k_n}(t) = \frac{p_{k_n}(t) g_{k_n}(t)}{I_{k_n}(t) + \sigma^2}$$

where $p_{k_n}(t) = P_n(t)/|K_n|$ is the transmit power allocated to user k_n at time t where $P_n(t)$ is transmit power of UAV n at time t and $|K_n|$ is the number of users serving by UAV n. $\sigma^2 = B_{k_n} N_0$ where N0 denoting the power spectral density of the additive white Gaussian noise. $I_{k_n}(t)$ is the interference between user k_n and all UAVs except the one it is serving by (UAV n). Power gain of channel between user k_n and UAV n given by

$$g_{k_n}(t) = {K_0}^{-1} d_{k_n}^{-\alpha}(t) \left[P_{\text{LoS}} \mu_{\text{LoS}} + P_{\text{NLoS}} \mu_{\text{NLos}}\right]^{-1}$$

where d_{k_n} is the euclidean distance between user k_n and UAV n, $K_0 = \left(\frac{4\pi f_c}{c}\right)^2$ with α as a path loss component, c is speed of light, f_c is the carrier frequency, μLoS and μNLoS are the attenuation factors. Line-Of-Sight(LoS) and Non-Line-Of-Sight(NLoS) conditions are assumed to encountered randomly and probability of LoS denoted as $P_L oS$ which can be expressed as

$$P_{\text{Los}}\left(\theta_{k_n}\right) = b_1 \left(\frac{180}{\pi} \theta_{k_n} - \zeta\right)^{b_2}$$

where b1, b2 and ζ is constant. $\theta_{k_n}(t) = \sin^{-1}\left(\frac{h_n(t)}{d_{kn}(t)}\right)$ is the elevation angle between UAV n and user k_n. Inherently, $P_{\text{NLoS}} = 1 - P_{\text{LoS}}$.

Finally, interference $I_{k_n}(t)$ expressed as

$$I_{k_n}(t) = \sum_{n' \neq n} p_{k_{n'}}(t) g_{k_{n'}(t)}$$

5.2 Algorithm and Environment Design

In order to create an environment which includes multiple UAVs (RL Agents) and users, a three dimensional space with an horizontally square shaped ground area is defined.

Each user's initial location is defined randomly in ground and UAVs' initial positions are predetermined in the same proximity to the center of the space. UAVs' altitudes are fixed so they have 5 action options (left,right,forward,back,remaining static) and all UAVs starts from same x and y but successive altitudes at 1 meter intervals. If there is more than one UAV closest to the user, the one closest to him in the horizontal plane is selected. Also environment is discretized to area units.

In this work, various action spaces, environmental parameters and neural network topologies have been experimented. State representation to be given to agents as input can be expressed as $s(t) = \{\{x_i(t), y_i(t)\}_{i \in N}, r(t), M(t)\}$ which consists horizontal coordinates of all UAVs, remaining time of trial and horizontal user-environment location matrix. Also hyper-parameters of reinforcement learning and deep learning methods such as learning rate, gamma, initial epsilon, neural network topology, epsilon decay rate, minimum epsilon tried to be optimized.

5.2.1 Reward Calculation

In order to keep reward fair for all states, reward of all agents are determined by the difference of system's total sum rate between current state and the state after the action taken by agent.

$$reward = R_{sum}(new) - R_{sum}(old)$$

where R_{sum} is the total sum rate of system which is explained in system model.

5.2.2 Implementation of Multi-Agent Deep Reinforcement Learning

To implement a multi-agent deep reinforcement learning, model-free, deep q learning based algorithms are used due to the discrete action spaces and continuous state space.

5.2.2.1 Deep Q Network

Experience replay algorithm with DQN is described below.

Algorithm 1 DQN Training with Experience Replay

procedure TRAIN(UAVs)
  for all UAV n in UAVs **do**
    5. Experience Replay:
    UAV gets random batch of past experiences $B \in M_n$ with size bs
    for each experience tuple of time j in B_n **do**
      Set $y_j = \begin{cases} r_j & \text{if } d_j \text{ is True} \\ r_j + \gamma \max(Q_n(s_{j+1})) & \text{else} \end{cases}$
      Apply a gradient descent step on $\left(y_j - Q_n(s_j)\right)^2$
    6. Adaptive Greedy:
    if $\epsilon_n > minEpsilon$ **then**
      $\epsilon_n \leftarrow \epsilon_n * epsilonDecayRate$

5.2.2.2 Double Deep Q Network

In DDQN, experience replay differs from DQN by using a second network called target network which updates its weights slower than and have the same topology with main network to estimate values of future rewards instead of using same network. DQN tends to overestimate q values which leads to unstable training and DDQN is proposed to avoid this problem. Experience replay algorithm of DDQN is described below.

Algorithm 2 DDQN Training with Experience Replay

procedure TRAIN(UAVs)
  for all UAV n in UAVs **do**
    5. Experience Replay:
    for each experience tuple of time j in B_n **do**
      Set $y_j = \begin{cases} r_j & \text{if } d_j \text{ is True} \\ r_j + \gamma \max(Q'_n(s_{j+1})) & \text{else} \end{cases}$
      Apply a gradient descent step on $\left(y_j - Q_n(s_j)\right)^2$
    6. Adaptive Greedy:
    if $\epsilon_n > minEpsilon$ **then**
      $\epsilon_n \leftarrow \epsilon_n * epsilonDecayRate$
    7. Target Network Update:
    $Q'_n \leftarrow Q_n$

5.2.2.3 Multi-Agent Independent Deep Reinforcement Learning

The main idea behind independent deep reinforcement learning depends on the assumption that if all agents optimize their q functions, the global reward will also be optimized [3]. By having cooperative and homogeneous setting, many optimization problems can easily provide this assumption.

Because multi-agent independent DRL is a method with decentralized training and decentralized execution, it is highly scalable but can be lack of optimizing hard and rare states in some robust environments. Algorithm of method is decribed below.

Algorithm 3 Multi-Agent Independent Deep Reinforcement Learning

procedure SIMULATE($environment, iterationCount, stepCount, collectStepCount$)
  for i $< iterationCount$ **do**
    Environment resets its state
    $isCollectStep \leftarrow i * stepCount <= collectStepCount$
    for t$<stepCount$ **do**
      for all UAV n **do**
        1. Input:
        UAV n receives state information s_n^t from environment
        2. Action Selection:
        UAV n selects a random action with probability ϵ
        Otherwise, UAV n selects an action $a_n^t = argmax(Q_n(s_n^t))$
        3. State Update:
        if the location after the action is against the limits **then**
          Reward $r_n^t \leftarrow PENALTY$
        else
          Environment updates location of UAV n based on a_n^t
          $r_n^t \leftarrow reward(\mathrm{a}_n^t, s_n^t)$
        Observe state s_n^{t+1}
        $d \leftarrow$ True if UAV is in terminal state at s_n^{t+1} else False
        4. Store Experience:
        UAV n stores experience tuple $m = s_n^t, a_n^t, r_n^t, s_n^{t+1}$,d in its memory M_n
    if isCollectStep == False **then**
      TRAIN(UAVs)

5.2.2.4 Multi-Agent Deep Reinforcement Learning with Cooperative Learning

Cooperative learning method allows UAVs to take advantage of the networks of subsequent UAVs during training. After agent n's action executed, for making agent n to take advantage of the subsequent agent m, agent n stores the state input received by agent m as its next state and during experience replay, agent n uses agent m's network to predict future rewards by giving stored next play to network of agent m. UAV with the higher index replays its experiences sooner in order to

Since this method is a method with assisted training and decentralized execution, it is both scalable and can achieve more optimal results in the long run. The disadvantage of this method is that it may take longer for any agent to converge than the independent method, since a single agent's network affects the learning of other agents. Algorithm of method is decribed below.

Algorithm 4 Experience Replay in Cooperative Multi-Agent Training

procedure TRAIN(UAVs)
  for all UAV n in UAVs in reverse order **do**
    $m \leftarrow (n+1) mod(len(UAVs))$
    5. Experience Replay:
    for each experience tuple of time j in B_n **do**
      Set $y_j = \begin{cases} r_j & \text{if } d_j \text{ is True} \\ r_j + \gamma \max(Q_m(s_{j+1})) & \text{else} \end{cases}$
      Apply a gradient descent step on $\left(y_j - Q_n(s_j)\right)^2$
    6. Adaptive Greedy:
    if $\epsilon_n > minEpsilon$ **then**
      $\epsilon_n \leftarrow \epsilon_n * epsilonDecayRate$
=0

5.2.2.5 Neural Network Topology

For UAVs to be scalable for multiple environments, the state input must be non-stationary even users' initial locations and movement strategies changes, therefore agents takes fully-observable environment-related state inputs such as 2D matrix that presents horizontal user map on the ground, in our case so that additional part that consist of an input, pooling and flatten layer added to network topology and its output concatenated with the first input layer. Neural network topology for environment-scalable setting is visualized below in 5.1

Every time environment is reset, users' initial locations are randomly determined according to Poisson Point Process distribution and movement strategies randomly determined according to normal distribution.

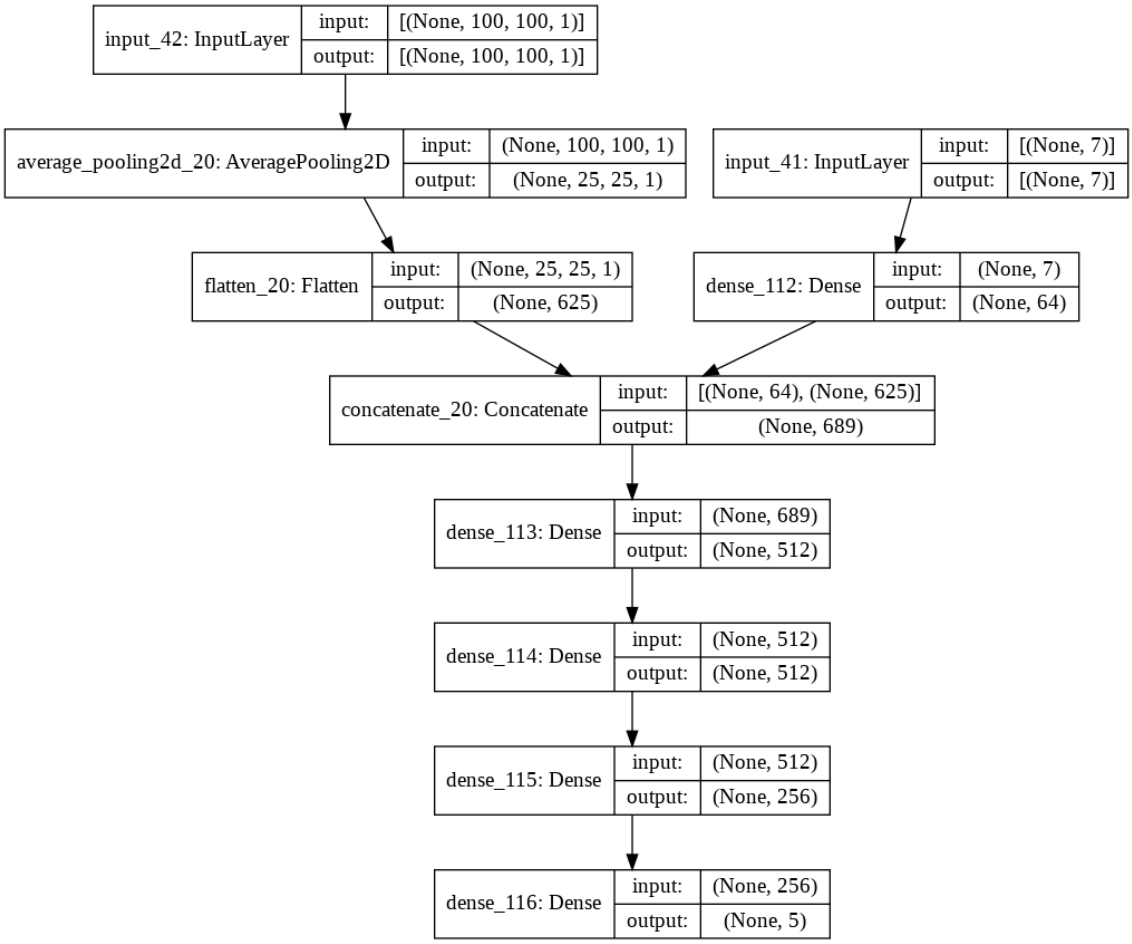

Figure 5.1 Topology of Environment-Scalable Deep Q Network (Environment assumed to have 3 UAVs)

6
Implementation

6.0.1 Simulation Tool

A demo application was made to run the trained models on the location data of the users on earth given by the person using the program. In this demo application, the user selects the path of a trained model folder and the file containing the user data from the dialog. Then user starts the simulation by clicking the "Start Simulation" button. Also user can choose to generate random location data by ticking the "Random Environment" checkbox.

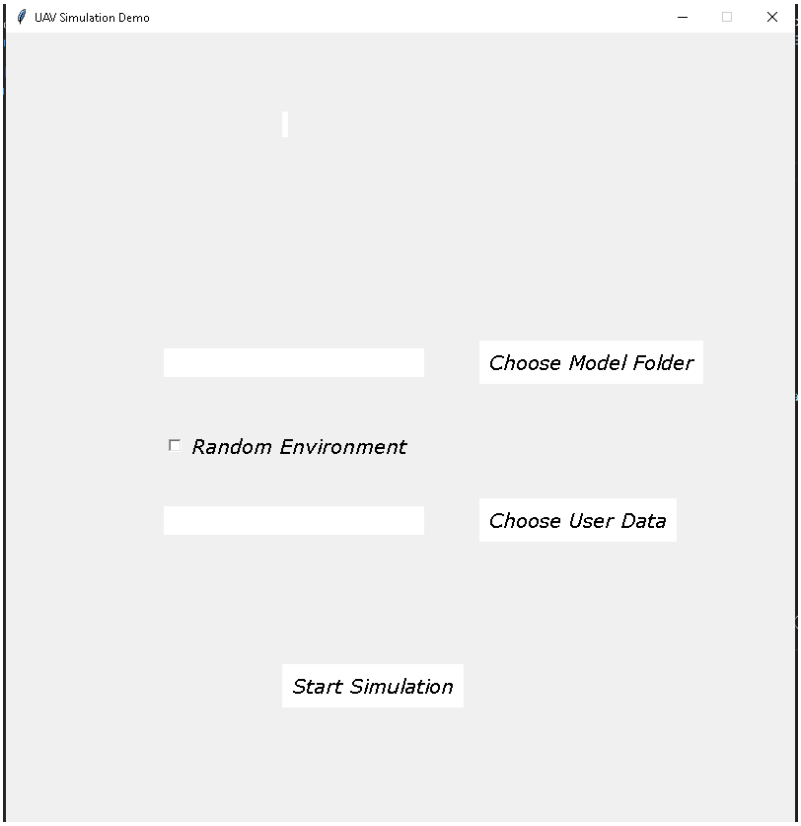

Figure 6.1 Home window of simulation software

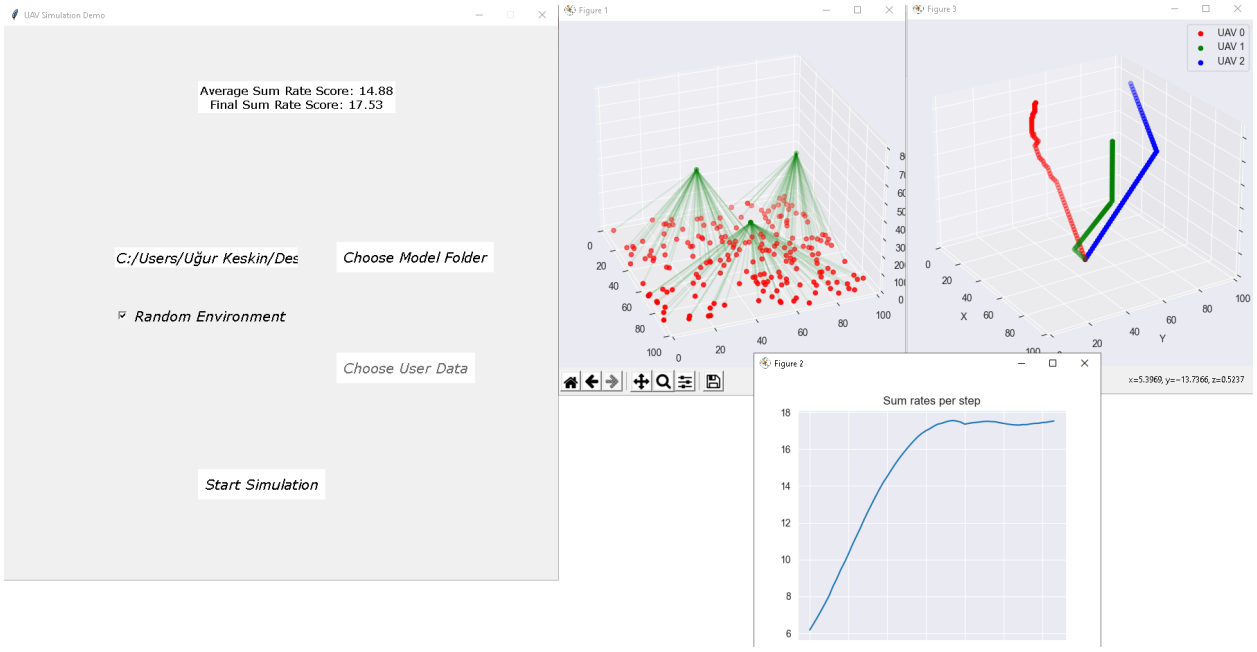

Figure 6.2 Screenshot of software after the end of simulation

After the simulation starts, the location of the UAVs and their connection with the users on the ground, the instantaneous total sum rate of that step and the trajectories of the UAVs along the way can be seen from the graphs in real time. Charts are interactive and images can be saved, perspectives of 3D charts can be changed by dragging interactively. After simulation ends, instantaneous sum rate at the last state and average sum rate of trial can be seen in window. Screenshots of software can be seen in 6.2 and 6.1.

7
Experiment Results

7.1 Parameters

7.1.1 Simulation Parameters

The environment is treated as a square-shaped area with a side of 100 units. Each unit corresponds to 20 meters. The movement speed of users is fixed and considered as 1 meter per second and the speed of UAVs is considered as 20 meters per second, so while UAVs move 1 unit per step, users move 1 unit per 20 steps.

Initial horizontal location of UAVs are same and in the center of the environment area. UAVs have different altitudes 1 meter apart also UAV with highest altitude's altitude is 400 meter. Simulation parameters are shown in Table 7.1.1

Parameter	Description	Value
N	Number of UAVs	3
K	Number of users	200
P	Transmit power of UAVs	0.08W
B	Bandwidth	1MHz
N_0	Noise power spectral	-170dBm/Hz
f_c	Carrier frequency	2GHz
α	Path loss exponent	2
u_{LoS}	Additional path loss for LoS	3dB
u_{NLoS}	Additional path loss for NLoS	23dB
$b1, b2, \zeta$	Environmental parameters	0.36, 0.21, 0
$UNIT$	Meter value corresponding to 1 unit	20
ALT	Initial altitude of highest UAV (in meters)	400
$iterationCount$	Number of iterations for simulation	100
$stepCount$	Number of steps for simulation	64

7.1.2 Training Parameters

Parameters of deep reinforcement learning are shown in Table 7.1.2. Each UAV takes a state input from environment at each step. State input consist of the horizontal locations of each UAV and the time information to avoid non-stationary states by considering both user movements (with giving time information) and UAV movements (with giving UAV locations). At the beginning of each iteration, The environment is reset by changing user and UAV locations to their initial positions. Huber loss function was chosen because it is less sensitive to outliers than the mean squared error.

Parameter	Description	Value
α_{lr}	Learning Rate	1e-06
γ	Discount Rate	0.95
ϵ	Initial epsilon greedy	1
BS	Batch Size	10
$LossFunction$	Loss Function of Neural Networks	Huber
$minEpsilon$	Minimum Epsilon	0.1
$epsilonDecayRate$	Epsilon greedy decay rate	0.96
$PENALTY$	Penalty Coefficient	-100

7.2 Results

7.2.1 Metric of Evaluation

The success of system is measured by the average of the total sum rates of all steps in trial and the sum rate at the last step of trial. In test trial, agents don't stores or replays experiences. After each training trial on random environment, agents performs on a test trial on a particular environment which is also created before with same random distributions. Sum rate of each step is calculated after all users and UAVs have performed their actions in that step. Starting locations of UAVs are always the center of environment so their initial locations are (50,50,400), (50,50,399) and (50,50,398).

7.2.2 Comparison of Results

Evaluation scores after each iteration can be seen below in Figure **??** and 7.2. Also sum rates per step during the trial where UAVs gets the highest score can be seen below in Figure 7.3 and 7.4.

Figure 7.1 Average sum rate scores per iteration

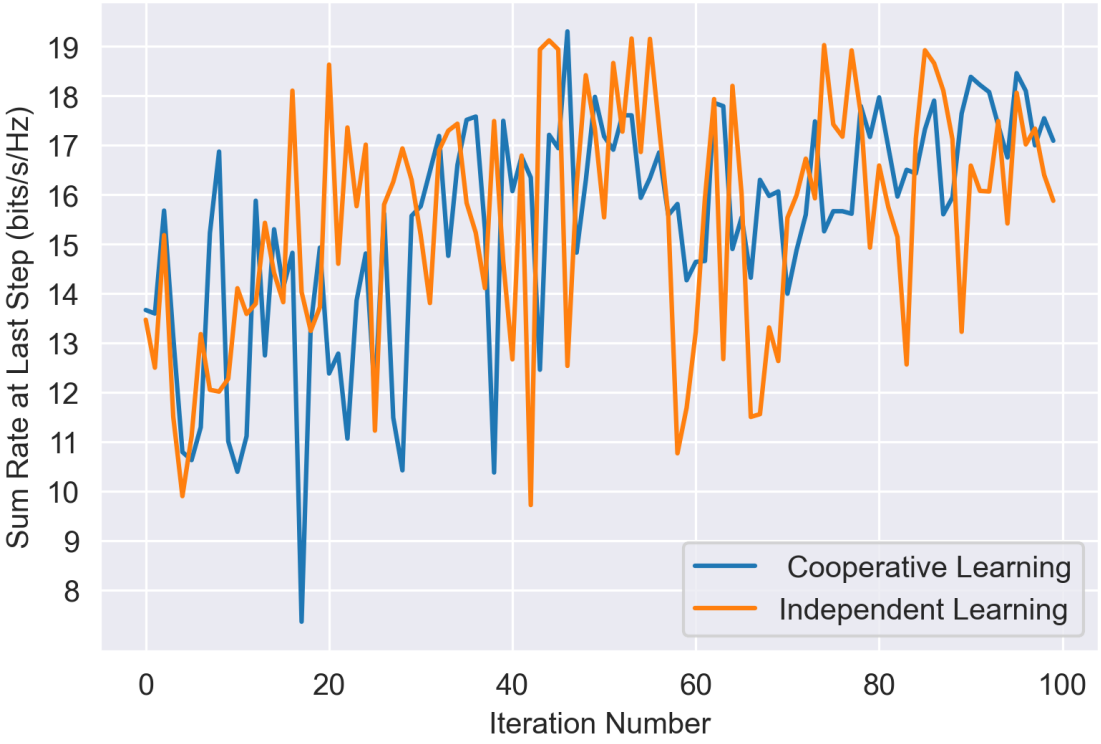

Figure 7.2 Highest instantaneous sum rate on last step scores per iteration

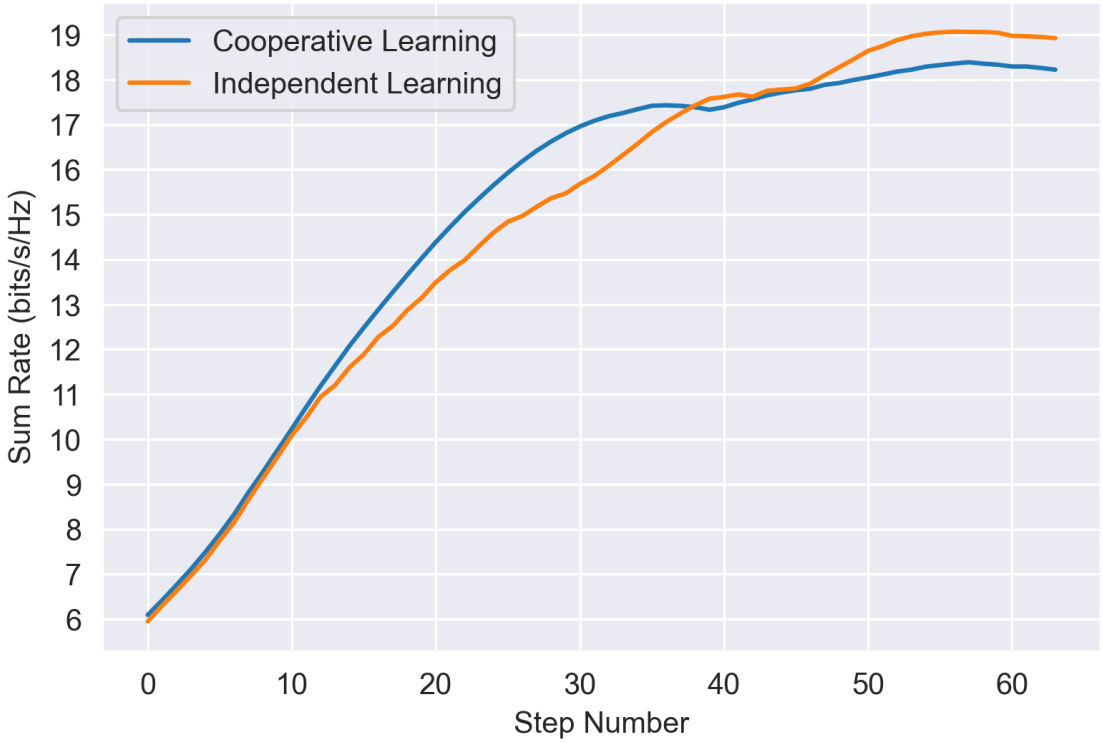

Figure 7.3 Sum rates per step during trial with the highest average sum rate score

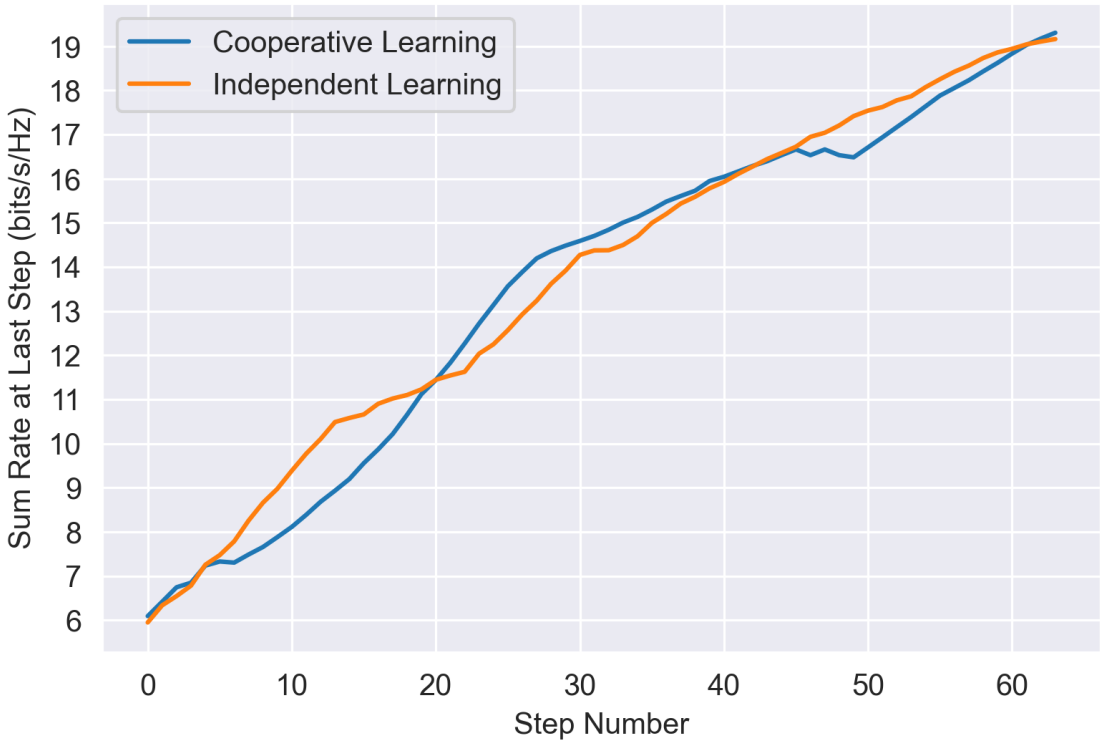

Figure 7.4 Sum rate of steps during the trial with highest last sum rate

Last location (location on the last step of trial with the highest average sum rate score) of UAVs and their association with users' can be seen in 3D chart 7.5. Comparison of Independent MADRL and Cooperative MADRL results shown in Table 7.2.2.

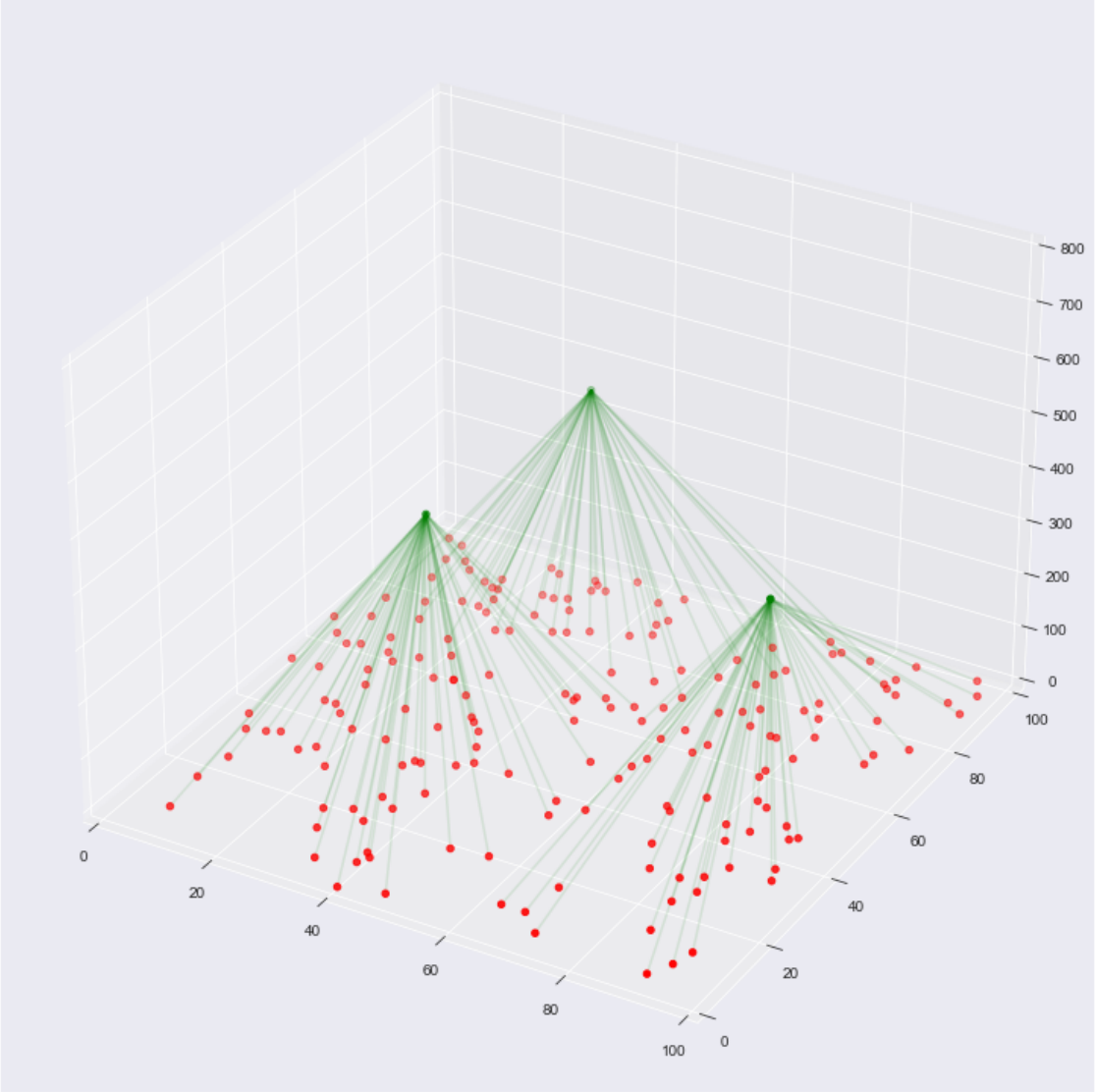

Figure 7.5 Last locations of UAVs (green) and their association with users on the ground (red) that provides highest instantaneous sum rate on last step

MADRL Method	Highest instantaneous sum rate	Highest average sum rate
Independent	19.16	14.85
Cooperative	19.30	15.04

As can be seen in figures above, Cooperative MADRL is more stable than Independent MADRL but it gets lower scores on first iterations which may be due to the fact that unlike independent MADRL, UAVs (agents) learn by interacting with each other by using each other's network to predict future rewards even agents are unstable and prone to make wrong decisions in the first iterations.

7.3 Conclusion and Possible Further Improvements

In conclusion, cooperative MADRL method provides more stable and successful way than independent MADRL. Trained models gets high scores on environments which their user location distribution fits the poisson point process distribution. At the end of the experiment, while 3 different UAVs move away from each other to avoid interference, they design their trajectory to take into account the location of the users. In greedy approach which reviews the output of all joint action combinations of UAV for current step, UAVs flies inseparably to avoid interference but they fails to increase their average and highest instantaneous score more than 8 bits/s/Hz because the problem absolutely need agents that afford interference in some steps to achieve better scores. Dynamic Programming and Q-Learning may not have achieved optimal results in this problem because they do not provide a scalable solution for multiple environments and are difficult to adapt to multi-parameter non-discrete state inputs.

Trajectory design of UAVs as an aerial base stations is very detailed and difficult process. Taking cognizance of interference, user locations, user movements and time makes trajectory design more complicated. To make available to get better results on distinct, tougher environments, models must be trained on a lot of distinct environments. Another ways to increase success of MADRL are increasing batch size, increasing iteration number, tuning parameters such as epsilon decay rate, initial epsilon greedy, minimum epsilon greedy learning rate, discount rate and using prioritized experience replay buffer. Increasing batch size and iteration number significantly increases duration of training that is also the reason for batch size was set low in this project.

References

[1] U. Challita, W. Saad, and C. Bettstetter, "Cellular-connected uavs over 5g: Deep reinforcement learning for interference management," *CoRR*, vol. abs/1801.05500, 2018. arXiv: 1801.05500. [Online]. Available: http://arxiv.org/abs/1801.05500.

[2] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "Multi-agent deep reinforcement learning-based trajectory planning for multi-uav assisted mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 73–84, 2021. DOI: 10.1109/TCCN.2020.3027695.

[3] X. Liu, Y. Liu, Y. Chen, and L. Hanzo, "Trajectory design and power control for multi-uav assisted wireless networks: A machine learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7957–7969, 2019. DOI: 10.1109/TVT.2019.2920284.

[4] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learning-based resource allocation for uav networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 729–743, 2020. DOI: 10.1109/TWC.2019.2935201.

[5] (), [Online]. Available: https://github.com/keras-team/keras/blob/master/LICENSE.

[6] (), [Online]. Available: https://github.com/tensorflow/tensorflow/blob/master/LICENSE.

[7] K. Zhang, Z. Yang, and T. Basar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *CoRR*, vol. abs/1911.10635, 2019. arXiv: 1911.10635. [Online]. Available: http://arxiv.org/abs/1911.10635.

[8] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," 2019. arXiv: 1906.04737 [cs.LG].

[9] O. Buffer, O. Pietquin, and P. Weng, "Reinforcement learning," *CoRR*, vol. abs/2005.14419, 2020. arXiv: 2005.14419. [Online]. Available: https://arxiv.org/abs/2005.14419.

[10] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018. arXiv: 1811.12560. [Online]. Available: http://arxiv.org/abs/1811.12560.

[11] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. arXiv: 1509.06461 [cs.LG].

Curriculum Vitae

FIRST MEMBER

Name-Surname: Uğur Keskin
Birthdate and Place of Birth: 25.05.1999, Samsun
E-mail: ugurkk2599@gmail.com
Phone: +90 545 811 49 60
Practical Training:
SIEMENS
Oredata

Project System Informations

System and Software: Windows, Python
Required RAM: 8GB
Required Disk: 12GB