POLITECNICO DI TORINO

Master of Science Degree in

Communication and Computer Networks Engineering

Machine Learning-driven Management of Unmanned Aerial Vehicles Networks

Academic Supervisors

Prof. Carla Fabiana CHIASSERINI

Prof. Roberto GARELLO

Company Supervisor

Ph.D. Carlo ONGINI

Candidate

Antonio CALAGNA

2020/2021

Abstract

In recent years, Unmanned Aerial Vehicles (UAVs) have been deeply investigated because of the relevant services they can support and the improvements they can provide in different application fields. Examples include last-mile delivery, area monitoring, and infrastructure inspection. UAV-aided communication networks can indeed extend or replace the existing communication infrastructure where such facilities would be difficult or too costly to deploy due to the remote or inaccessible locations, like in the case of areas hit by natural disasters.

In spite of the recent advancements, UAV operations and scenarios introduce unique technical challenges, among which remote control and efficient usage of computational resources emerge as aspects of primary importance. In this context, data-driven approaches such as machine learning represent an effective methodology. UAVs, as well as ground devices with which UAVs communicate, can collect information on the tasks to be executed, the experienced channel propagation conditions, and the extension of the covered area – piece of information that can be leveraged for the design of effective solutions. Importantly, while developing algorithms to address the aforementioned issues, it is also critical to evaluate the system performance through scalable, yet realistic, simulations.

In this work, both objectives are pursued, i.e., to develop *and* implement a machine learning solution that exploits the data collected by UAVs and ground devices in order to achieve coverage maximization, while simulating the system in realistic settings. Realism is brought by the integration with the well-known *ns-3* simulator a statistical channel propagation model.

The addressed scenario includes a set of UAVs, which are meant to assist a number of ground devices in processing tasks and eventually provide the ground devices with an outcome. The UAVs can also communicate with

neighbouring UAVs for either sharing their status information or forwarding tasks if they are too overloaded. Furthermore, a centralized network architecture characterized by UAVs and a base station has been considered; the base station will act as aggregation point of the observations forwarded by UAVs and will provide each one with the action to perform according to a certain policy.

Hence, a deep reinforcement learning model has been developed in order to achieve an efficient, flexible and scalable policy. The algorithm will take as input the positions of the UAVs and of the ground devices, and the computational task demand; in return, it will provide the best action to perform in order to maximize the expected cumulative reward, e.g. coverage or resource utilization maximization.

To conclude, *ns-3* offers a noteworthy simulation environment that allows to efficiently evaluate and compare several scenarios, protocols and applications. This permits to easily encompass all the different aspects that contribute to design quality and network performance. Future work could concentrate on improving the techniques tackled in this thesis and further extending the proposed scenario with higher complexity features that may lead to interesting results and effective performance evaluation. The prospect of being able to merge realistic network simulations with innovative artificial intelligence optimization algorithms serves as a continuous incentive for future research that may also involve other cutting-edge scenarios.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

UAV networks are an emerging technology that has been recently investigated in deep due to the extensive set of application domains where its support can definitely provide an improvement of the experienced service. UAVs can play an active role in the definition of new state-of-the-art services or the redefinition of existing ones, such as package delivery, precision agriculture, surveillance and inspection procedures.
Furthermore, the establishment of a communication network among several UAVs enables the possibility of their interaction and cooperation with the purpose of achieving a final common aim. In this scenario, supplying UAVs with computational capabilities, it is possible to configure a centralized or distributed architecture for decision making. Machine-learning-based frameworks, especially Reinforcement Learning ones, can be therefore exploited for optimization purposes.

On the other hand, UAVs can also serve as aerial low-altitude platforms extending or replacing the existing communication infrastructures in fields where these facilities are hard to be deployed or corrupted by unexpected events. The interaction with an existing networking infrastructure further extends the previously mentioned set of services, drawing the attention of research communities toward next-generation wireless networks and inspiring

telecommunication companies for the definition of new vertical industries solutions. Assuming just one UAV, many BVLOS (Beyond Visual Line of Sight) use cases have been already studied and deployed in real-world scenarios; one of them is infrastructure inspection for which the supervisor is no longer required to be in the specific location of interest and the urban interruption of service that usually an operation like this requires can be avoided. Considering instead a network of several UAVs, the potential is further enhanced: fleets of UAVs can provide computational support for the execution of tasks forwarded by IoT devices or general ground devices, therefore acting as computational aerial servers, performing data collection, partial/full analysis and eventually forwarding the post-processing outcomes to the internet or to the ground devices themselves.

Recent works on UAV networks have demonstrated how the communication challenges and open problems related to the deployment and utilization of this technology are still in an early stage [1]. One of the main challenges regards the proper placement in space of highly dynamic nodes, such as UAVs, within the network in order to optimize the overall performance of the system. Centralized or distributed control solutions can be defined to drive each UAV towards an optimal position. Machine Learning frameworks, especially Reinforcement Learning (RL) approaches, have acquired a significant role in control design application due to the intelligent identification of valid environmental information and complex patterns [2].

RL is one of the most popular machine learning approaches, its aim is to solve Markov Decision Processes (MDP) by making an agent learn how to make decisions according to observations of the surrounding environment and a correspondent value of expected cumulative reward. It has been found to be particularly suited for collaborative tasks in multi-agent systems [3] and performance enhancement in communication networks [4].

1.1 Context and Problem Statement

The purpose of this thesis is the development of a machine learning solution that aims at the optimization of UAVs' positioning and computing tasks assignment while simulating the system in a realistic setting.

A specific context has been considered: fleets of UAVs, cooperating with each other and acting as computational aerial servers, are meant to provide computational support for ground devices' tasks.

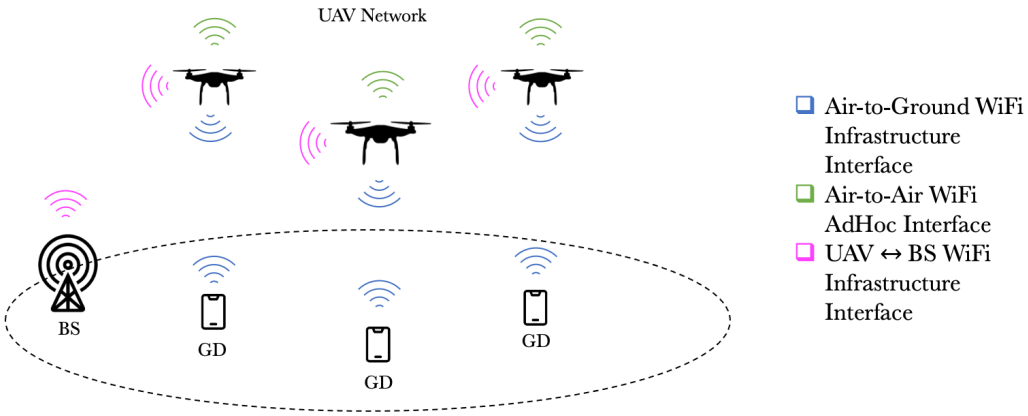

Figure 1.1: Scenario proposed in this work. UAVs are meant to communicate with GDs for task execution and with a BS for positioning policy application. Icons provided by Clea Doltz Patrick McDonnell and Barudak Lier from the Noun Project.

Thus, the first goal is to implement realistic traffic generation and channel propagation models in order to achieve a network simulation as much realistic as possible. In the second instance, this work will develop by introducing a OpenAI Gym interface in order to enhance the simulation with multi-agent Deep Reinforcement Learning (DRL) algorithms that, by exploiting the observation coming from the network simulation, leverages the agents' position and hence optimize coverage and resource utilization. The framework that has been considered relies on a centralized architecture in which a base

station is supposed to collect all information about UAVs and provide them with the best action each one can perform to maximize an expected cumulative reward.

In conclusion, a key feature of this work is the efficient integration of the artificial intelligence framework with a realistic, flexible and scalable network simulation environment provided by *ns-3*. All the modules that have been developed in this work are scenario agnostic, such that they can be easily reused in other contexts or easily extended to provide higher complexity. This integration is supposed to provide researchers with state-of-the-art ways to develop, perform and evaluate realistic network simulation scenarios that exploit AI algorithms for managing the increasing complexity of modern networks.

1.2 Structure

The work developed in this thesis is presented according to the following organization.

Chapter 1 - Introduction

The first chapter illustrates the thesis from an high-level perspective. Here, the main motivations and objectives are introduced and discussed.

Chapter 2 - UAV Network Design: Existing Approaches

The second chapter discusses the main research works that have inspired and contributed to the development of this thesis. The aim of this chapter is to exploit results and considerations from other researchers to highlight the relevance of this work and the innovation it tries to carry out.

Chapter 3 - Background on Mobile Distributed Networks

The third chapter provides a brief description of the main theoretical concepts that are required for an insightful understanding of this work. Reference books are provided for all topics, in case further information would be of interest.

Chapter 4 - Problem Formalization

The fourth chapter presents the needs and the main problems that has inspired this work. A formal perspective is adopted, highlighting the assumptions and the strategies deployed at the basis of the solution proposed. In addition, the algorithms that have been deployed in order to model the proposed scenario are introduced from a formal perspective.

Chapter 5 - Tools

The fifth chapter highlights the main software that has been exploited in the course of this thesis. A description of their main characteristics and functionalities is provided, highlighting the reasons and relevance of their deployment.

Chapter 6 - Implementation

The sixth chapter introduces and discusses the specific implementation for this scenario. An accurate description of the new modules and components that have been produced in this work is provided.

Chapter 7 - Experimental Results and Discussion

The seventh chapter illustrates and compares the experimental results obtained from the simulation of different scenarios. They are all deeply discussed, pointing out the strengths and the limitations of the outcomes.

Chapter 8 - Vodafone Internship

The eighth chapter acknowledges Vodafone Italia as a great source of inspiration for the development of this work. The projects on which Vodafone Business' 5G Program has been investing on are presented, along with a technical discussion of the potential 5G brings in the definition of revolutionary applications and services.

Chapter 9 - Conclusions and Future Works

The final chapter concludes this presentation with a summary of the proposed approach and the resulting outcomes. The main limitations and suggestions for future works and improvements are eventually proposed.

Chapter 2

UAV Network Design: Existing Approaches

Significant inspiration and contribution to the development of this thesis has been brought by former researches and surveys. In the following, a literature review is performed in order to highlight the open problems and the state-of-the-art strategies regarding UAV-based communications, Machine Learning integration and propagation modeling.

2.1 Machine Learning and UAV-based Communication

In recent years, standardization bodies, industries, and academia have been nourishing interest towards the use of UAVs as flying BSs, mobile relays, or autonomous communicating nodes for providing low latency and highly reliable communications in cities, across suburban areas and over rural terrains. A recent study [5] aims at investigating this topic, providing a detailed survey of all relevant research works in which Machine Learning techniques and UAV-based communications cooperate.

The authors specifically focus on the following contributions:

- Overview of AI solutions and their application in UAV-based networks
- Discussion of the main UAV-enhanced wireless communication issues, ranging from physical layer and resource management aspects up to trajectory design, security and public safety matters
- Identification of open issues in order to foster further research for UAVs and AI integration

They essentially outline the importance of radical improvements in wireless networks and the necessity for further advancements based on the application of machine learning in UAVs-based networks. In particular, they deeply investigate four main application areas in which innovative solutions can be gathered:

- Physical Layer issues:
  - Development of accurate channel models and mitigation of path-loss through topology prediction
  - Tacking of severe interference from other UAVs and ground nodes
  - Configuration of transmission parameters towards achieving specific performance targets
- Security and Safety issues:
  - Protection against cyber-attacks across different network layers
  - Discussion on the impact of jamming, eavesdropping and spoofing over VANETs
  - Ensuring privacy when task computation offloading is considered
  - Realtime mapping and avoid trespassing UAVs
- Resource Management and Network Planning:

  - Predict cell quality and select the best serving cell for one node at a given location
  - Reducing unnecessary handovers
  - UAV cooperation and joint optimization of the computation, caching and communication resources
  - Quality of Experience maximization together with energy efficiency

- Position related aspects:
  - Detection and Identification
  - Determine the proper horizontal and/or vertical placement as well as the trajectory that optimizes one or more KPIs

2.2 Channel Model

From a channel propagation point of view. in order to make ns-3 simulation realistic, the work in [6] has been studied in deep and exploited for the definition of the proposed implementation.

The authors in [6] propose a statistical propagation model for predicting the air-to-ground path loss between a low altitude platform (LAP) and a terrestrial terminal. This RF model has the aim of helping designers to simplify simulation calculations, to speed up radio coverage estimation and to facilitate the planning efforts of airborne wireless services.

It assumes isotropic antennas for transmitters and receivers and proposes 12 data sets depending on center frequency and urban environment density. Three different frequencies are considered, i.e. 700 MHz, 2 GHz and 5.8 GHz, and, for each center frequency, a simulation is run over four different urban environments, i.e. Suburban, Urban, Dense Urban and Highrise Urban. Moreover, two different propagation groups are considered, one corresponding to receivers favoring Line-of-Sight condition and the other referring to no

Line-of-Sight scenario, in which the received signal is strongly affected by reflection and refraction. A third group could have been analyzed, considering receivers suffering deep fading due to consecutive reflections and diffractions; nevertheless, it was disregarded because the sample set was limited and results would have been unreliable.

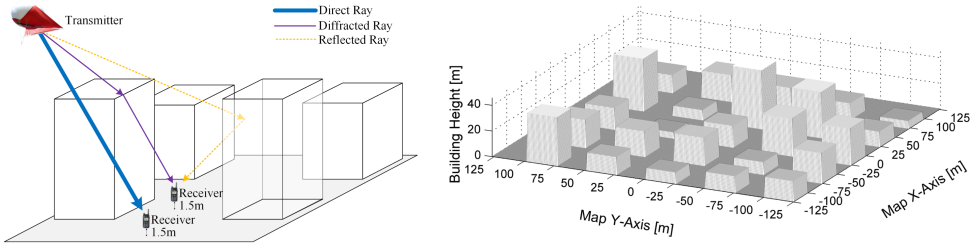

Figure 2.1: On the left, the three different types of simulated rays discussed in [6] are shown. On the right, a computer generated city model based on the ITU-R parameters in [7] and [8]

By curve fitting, using Damped Least Squares (DLS) method, the model parameters have been explicitly defined and reported in Tables B.1 and B.2. The algorithm to implement and utilized the proposed radio model is provided in Algorithm 4. The algorithm consists in an enhancement of the common free space path loss computation, based on the Friis Propagation Law in (2.1).

$$FSPL = 20\log_{10}(d) + 20\log_{10}\left(\frac{f}{1e6}\right) + 20\log_{10}\left(\frac{4\pi \cdot 1e6}{c}\right) \tag{2.1}$$

The total path loss the received signal will be affected by is

$$PL = 10\log_{10}(P_{TX}) - 10\log_{10}(P_{RX}) = FSPL + \xi \tag{2.2}$$

where ξ is the extra path loss depending on the elevation angle of the LAP, the probability of LOS, and the urban density. This model considers ξ normally distributed with mean value μ_ξ and standard deviation $\sigma_\xi(\theta)$. Since the

experimental results did not show a clear dependency of μ_ξ on the elevation angle θ, it is reported as a constant value in the Tables B.1 and B.2. On the other hand, the standard deviation $\sigma_\xi(\theta)$ was strongly dependent on the elevation angle θ, hence, the samples had been fitted to the following formula:

$$\sigma_\xi(\theta) = \alpha \cdot \exp(-\beta \cdot \theta) \tag{2.3}$$

where α and β are frequency and environment dependent parameters available in Tables Tables B.1 and B.2.

Eventually, these parameters are classified in two subsets, the Line-of-Sight Group and the non Line-of-Sight one. Consistently, the mean value and the variance of the excessive path loss are expected to be much bigger in scenarios where a Line-of-Sight path is not available. In general, a channel model that is characterized by many multipath components and one strong LoS one is said to be statistically characterized by a Rician distribution. On the other hand, when no LoS component is available, the channel can be described with a Rayleigh distribution, which usually leads to very fast frequency selectivity and strong Intersymbol Interference that must be counteracted with an equalization processing stage at the receiver [9].

By curve fitting, the authors provided the equation to compute the probability of LoS occurrence:

$$P[LOS] = \gamma \cdot (\theta - \theta_0)^\delta \tag{2.4}$$

where θ is the elevation angle of the low altitude platform and θ_0 is the minimum elevation angle. A minimum elevation angle is introduced because, usually, for $\theta < 15°$, the propagation path may be subject to so many impairments that no reliable transmission can be actually expected.

Authors in [6] present their result in terms of cumulative distribution function versus the path loss. Figure 2.2 the prediction resulting from the mathematical model reproduces the path loss estimations in an accurate

manner with respect to the ray tracing simulation.

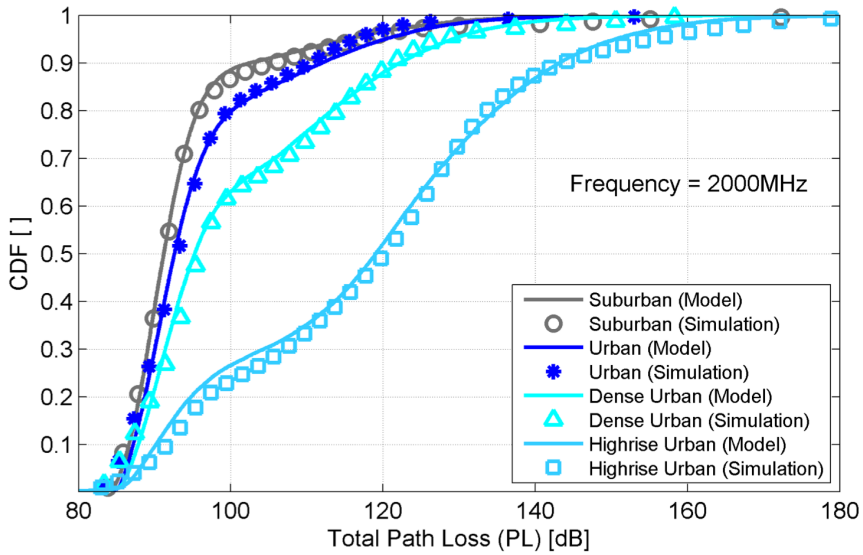

Figure 2.2: Cumulative Distribution Function vs Path Loss comparison between the proposed RF model and ray tracing simulation data in [6]

2.3 AI Integration

Besides acquiring an insight on Machine Learning approaches and especially achieving confidence with Reinforcement Learning strategies [10], one of the main focuses of this work was the integration of ns-3 simulator with RL algorithms running on Python.

Mainly two works have been considered, ns3-gym in [11] and ns3-ai in [12]. The authors in both works share the same motivation; modern communication networks are evolving into extremely complex and dynamic systems for which traditional design approaches are becoming limited. Reinforcement Learning has already proved its effectiveness in communication systems, providing innovative and powerful optimization techniques for scheduling, resource management, congestion control, routing and adaptive video streaming. Nevertheless, the need of real-world environments for a proper training of

the agents, would lead to expensive and time-consuming testbed deployment. Hence, the purpose of both works is providing a framework that can interface a realistic and powerful network simulator (ns-3) with the well-known and agile Open AI Gym framework for RL-based algorithm prototyping.

The ns-3 simulator can be exploited to create a realistic model of the network and configure scenario conditions, such as traffic, mobility, etc. Open AI Gym, basing on the observations collected from ns-3 environment, will produce the best action the agent must leverage in order to maximize a certain expected cumulative reward.

The ns3-gym framework in [11] exploits ZMQ sockets [13] for data transmission between ns-3 and Open AI Gym [14]. An Environment Gateway and an Environment Proxy were defined, respectively on ns-3 and Open AI Gym side, in order to aggregate and systematically exchange observations and actions between the two environments. More details on how ns3-gym works are discussed in Chapter 5.

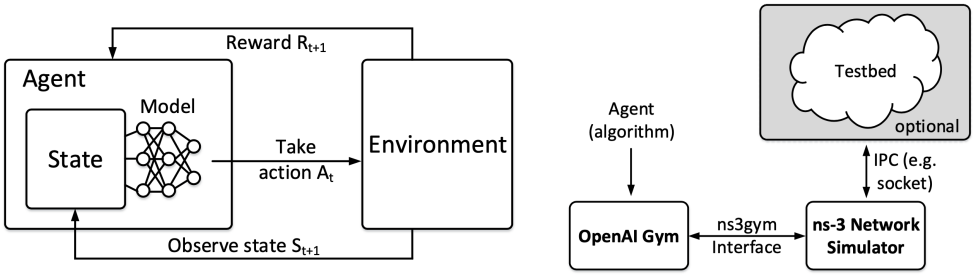

Figure 2.3: On the left, a simple scheme of Reinforcement Learning algorithm is shown. On the right, an high-level scheme of ns3-gym interface is illustrated. Both Figures have been produced by authors in [11]

The ns3-ai toolkit in [12] arises an enhancement of ns3-gym from the average transmission time perspective. Instead of deploying ZMQ sockets, ns3-ai exploits a shared memory technique that allows to achieve a transmission speed seven times faster from ns-3 to AI frameworks and 50-100 times faster the other way with respect to ns3-gym.

In this work, even if the advantages of ns3-ai are clear and inspiring, ns3-gym framework was deployed for its robustness and clarity; from an high-level user perspective, its APIs are much more intuitive and implementation agnostic than the ns3-ai ones.

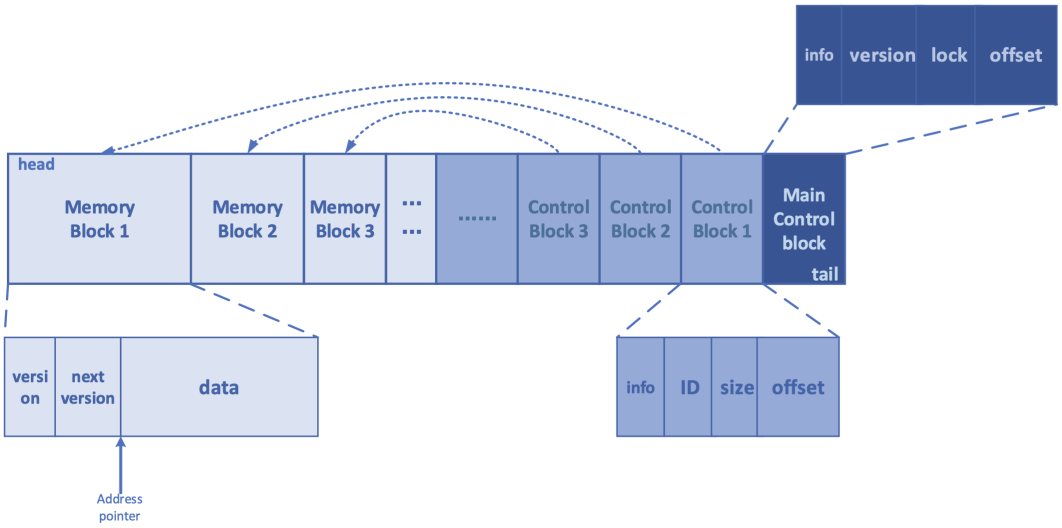

Figure 2.4: Shared memory structure proposed by ns3-ai authors in [12]

Chapter 3

Background on Mobile Distributed Networks

3.1 Unmanned Aerial Vehicles

Unmanned aerial vehicles (UAVs) are aircrafts designed to fly without human assistance on-board. They can either be controlled remotely from a ground operator or they can fly autonomously thanks to artificial intelligence algorithms. Besides the level of autonomy of their flight control, the most relevant classification is based on the the altitude they can reach. Hence, they can be divided into High Altitute Platforms (HAP) and Low Altitude Platforms (LAP) [1]. Devices belonging to the first category are able to fly at altitudes higher than 17km, and they are usually deployed for long term applications. Viceversa, devices belonging to the second category can fly from tens of meters up to few kilometers. LAPs are usually devoted to rapid, flexible and dynamic applications that can range from common use cases, such as area monitoring or infrastructure maintenance, to very time-sensitive scenarios, like natural disasters or emergency rescue.

An important limiting factor for the actual design and deployment of use cases involving UAVs is regulation. For instance, Italy is one of those

countries where flight restriction in urban areas is strictly imposed; acquiring the permissions to fly may be unfeasible from a time perspective or impossible at all depending on the characteristics of the use case.

3.2 Wireless Local Area Networks

IEEE 802.11, also known as WiFi, is a US-based standard released for the first time in 1997. It defines technical specification for physical and data layers' for WLAN implementation. Since then, a series of IEEE 802.11 standards have been proposed to efficiently serve specific scenarios, improve performance, provide higher data rates and better coverage**??**. Some of them are listed below, chronologically:

- **IEEE 802.11:** the original version of the standard. It exploits Direct Sequence Spread Spectrum (DSSS) and Frequency Hopping Spread Spectrum (FHSS) techniques at 2.4 GHz RF and IR, achieving a peak rate of 2 Mbps.
- **IEEE 802.11b:** the most popular standard, operating at both 2.4 GHz and 5GHz. It enhances the original protocol supporting 5.5 and 11 Mbps.
- **IEEE 802.11a:** wireless network bearer operating at a central frequency of 5 GHz and achieving data rate up to 54 Mbps. At the physical layer, Orthogonal Frequency Division Multiplexing (OFDM) is used, each subcarrier can be modulated with BPSK, QPSK, 16-QAM, or 64-QAM, depending on the wireless environment.
- **IEEE 802.11g:** extension with backward compatibility of 802.11b achieving up to 54 GHz in the 2.4 GHz band.
- **IEEE 802.11e:** the approved amendment that defines a set of Quality of Service (QoS) enhancements by making several modifications to the

Media Access Control (MAC) layer. These enhancements, such as packet bursting, enabled better transmission quality for audio and video applications.

- **IEEE 802.11n:** high-speed WLAN protocol based on Multiple-Input Multiple-Output (MIMO) technology. Network throughput is enhanced up to 600 Mbps thanks to antenna diversity and spatial multiplexing, without any additional cost of bandwidth or transmission power.

- **IEEE 802.11r:** enhancement of the original specification enabling fast BSS transition. It has been designed to specifically manage handover in a seamless manner, avoiding disruption of service for wireless devices in motion.

- **IEEE 802.11p:** amendment that introduces wireless access for the vehicular environment. It enhances 802.11 in order to allow efficient Vehicle-to-Vehicle (V2V) or Vehicle-to-Everything (V2X) data exchange.

- **IEEE 802.11ac/ax/ad:** state-of-the-art versions of the standard. They exploit higher carrier frequencies, complex modulation schemes, beamforming techniques and other innovative technologies to achieve extremely high throughput and low latency.

The standard supports two types of wireless interface: infrastructure and ad-hoc modes. In the wireless infrastructure mode, the network consists of a wireless access point (AP) and several wireless stations (STA). The AP coordinates the transmission among stations within its radio coverage area, called Basic Service Set (BSS), and it is responsible for bridging traffic toward the wired LAN.

On the other hand, Ad-Hoc networks represent an evolution of infrastructure mode, consisting of self-organized networks made only by wireless clients communicating directly with each other.

3.2.1 MANETs

A Mobile Ad-hoc Network (MANET) is a collection of hosts equipped with wireless communication capabilities, such as WiFi or Bluetooth, which can communicate with each other without any centralized administration. Each node of the network can communicate directly with all the other nodes within its transmission coverage area (point-to-point). Nodes located outside this area can be reached, if possible, via several intermediate nodes (multi-hop mode).

These networks are deployed in highly dynamic scenarios where the infrastructure is disrupted or not feasible to be implemented, such as natural disasters, military conflicts, emergency situations or simply local events and specific use cases like precision agriculture [15].

3.2.2 VANETs

A specific type of MANET is the Vehicular Ad-Hoc Network (VANET), which is a collection of vehicles provided with mid-range wireless communication technologies, such as the 802.11p WiFi standard. Vehicle-to-vehicle (V2V) and vehicle-to-roadside (V2X) communications architectures co-exist in VANETs with the purpose of providing road safety, navigation and many other applications and services proposed by the Intelligent Transportation Systems (ITS) research [16].

3.2.3 FANETs

Flying Ad-Hoc Networks or FANETs are an extension of MANETs and VANETs that specifically investigate Unmanned Aerial Vehicles (UAVs) ad-hoc wireless networks. FANET is the latest technology that has been proposed for several military and civilian purposes, such as infrastructure extension or surveillance and monitoring of those areas where humans cannot reach [17].

For instance, FANETs allow to relax many constraints imposed by the traditional UAV-Terrestrial Infrastructure centralized architecture, i.e. with only one ground station gathering and processing all information from UAVs in range. However, unlike Air-to-Air communication, the Air-to-Ground communication link is highly affected by urban environment conditions, lack of Line of Sight (LoS) condition and several other impairments. Furthermore, the dynamicity of the flying nodes is strictly constrained by the coverage range of the fixed infrastructure, which also represents a single point of failure for the network, inevitably increasing vulnerability.

A direct communication between the flying nodes of the network enables the possibility of overcoming these problems and introducing other communication approaches relying on the interaction with different kinds of infrastructures, e.g. distributed ones like satellite and cellular networks.

Some examples of the main FANET network architectures and integrations are provided in the following:

- **Centralized Architecture:**
  As previously mentioned, this scenario assumes a UAV direct communication with a ground station, specifically each flying node is connected to a terrestrial control station according to a star topology. This approach enables fault tolerance whenever any of the UAVs fails, easy synchronization and full knowledge of the state of each UAV in range. On the other hand, this solution is not scalable with the number of UAVs since the ground station may become the bottleneck of the network. Moreover, the ground station itself acts as a single point of failure; its breakdown would cause a full service [18] disruption. For simplicity, the assumption carried out in this work considers all the devices always reliable; therefore, the analysis of failure scenarios can be destined for future works.

- **Integration with Satellite Networks:**
  In first instance, communication with satellite networks is considered

when a ground station or a cellular infrastructure are not feasible to be deployed [19]. Secondly, it is important to remark that recently FANETs are attracting considerable interest in terms of Global Navigation Satellite Systems (GNSS) Integration and Augmentation. Integration, also known as Assisted GNSS techniques, refers to the idea of providing a GNSS receiver with additional information through external wireless aiding sources. This information may be related to proprioceptive sensors on board of drones or exteroceptive sensors on ground devices. On the other hand, Augmentation refers to the idea of making UAVs act like pseudolites, i.e. fake satellites transmitting GNSS-like signals with the purpose of improving accuracy and availability. In conclusion, this aiding allows to improve the overall performance of the GNSS receiver, such as submeter precision, the time to first fix (TTFF) and the sensitivity in harsh environments. [20]

- **Integration with Cellular Networks:**
  Connecting UAVs with the cellular networks is among the most widely investigated topics. Although the several advantages and the numerous use cases this solution would enable, UAVs cannot be seen as typical ground user equipment. Low Altitude Platforms undergo a very different radio propagation for which the base stations are not designed for. Recently, Third Generation Partnership Project (3GPP) has concluded its work about the integration of UAVs communication in LTE networks [21]. The application that this thesis will discuss in this thesis would definitely benefit from cellular communication. A connection between a centralized base station and all the ground devices would ease up the initialization process in which UAV must discover all the ground devices asking for service. Nevertheless, since this work aims to provide communication and computational services in critical areas where the infrastructure is not available, the cellular link will not be taken into account. Hence, it will be assumed that a discovery operation had been

already performed offline, such that UAVs are always aware of which ground devices are in need of service and where they are located.

3.3 Network Simulation

In general, performance evaluation of systems and networks is usually performed with three different techniques: mathematical analysis, measurements, and computer simulation [22].

Network simulation is a computer network research technique that exploits a software program to model the behaviour of a network through the realistic interaction of the different network entities and the various applications and services it supports. The need for this technique with respect to the others arises from the unfeasibility of traditional analytical methods to provide an accurate understanding of complex systems' behaviour and hence the impossibility of producing an efficient design of the solution.

Among the several different types of computer simulations, such as discrete-event, continuous, Monte Carlo, trace-driven etc., the most relevant approach in the field of computer networks is *discrete-event simulation*. The key property of discrete-event simulations is that the state of the considered model can only change discretely in time, according to the occurrences of certain *events*.

Discrete-event simulation is used to do research on all layers of computer networks, including signal processing issues in the physical layer, medium access in the link layer, routing in the network layer, protocol issues in the transport layer, and finally, design questions of the application layer.

3.4 Machine Learning

Machine Learning is a branch of Artificial Intelligence that studies and defines computer algorithms for systems that automatically learn and improve their

performance through experience and data.

In recent years, Machine Learning approaches are attracting an increasing interest due to their effectiveness towards complex systems and applications, especially those where it is difficult or unfeasible at all to develop conventional algorithms to perform predictions and decisions without being explicitly programmed.

Machine Learning strategies are traditionally classified into three broad categories, depending on the nature of the signal or feedback available to the learning system:

- **Supervised Learning**:
  The computer is presented with a set of examples made of different inputs and their desired outputs. The goal of the algorithm is learning, basing on this training set, the general rule that maps inputs to outputs. Once the algorithm is trained, it can be tested over a verification test to evaluate its reliability. Hence, this approach is oriented to applications that involve classification or regression procedures.

- **Unsupervised Learning**:
  The learning algorithm is not provided with labels anymore; its goal is to discover hidden patterns or correlation features within the data provided. This strategy is commonly deployed in cluster analysis and data mining applications.

- **Reinforcement Learning**:
  An agent exists within a dynamic environment that he is able to explore and collect information from. Basing on this information, the agent will predict and perform actions that will lead to new observations. The final aim of the agent will be the maximization of an arbitrarily defined cumulative reward by actively interacting with the environment. This approach has proved to be considerably effective in game theory, control theory and simulation-based optimization, among the others.

3.4.1 Reinforcement Learning

This work will focus on the third kind of machine learning paradigms, i.e. Reinforcement Learning.

As previously introduced, Reinforcement Learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. It differs from the other approaches in not needing labelled input/output pairs and sub-optimal actions to be explicitly corrected. The focus is on finding a balance between exploration of uncharted territory and exploitation of current knowledge.

Basic Reinforcement Learning can be introduced as a Markov Decision Process (MDP), i.e. a discrete-time stochastic control process providing a mathematical framework for modelling decision-making operations in situations where outcomes are partly random and partly under the control of the agent.

The MDP under analysis is made of,

- a set of environment and agent states $\mathcal{S}$
- a set of actions the agent can perform $\mathcal{A}$
- probability of transition at time t from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ under action $a \in \mathcal{A}$, i.e. $P_a(s, s') = P[s_{t+1} = s' \mid s_t = s, a_t = a]$
- immediate reward after transitioning from s to s' with action a, i.e. $R_a(s, s')$

By definition, the probability of occurrence of a given state only depends on the previous state of the environment and not the entire history of transitions. This property is known as **Markov Property** (Eq.3.1)

$$P[s_{t+1} \mid s_t] = P[s_{t+1} \mid s_1, \ldots, s_t] \tag{3.1}$$

The agent is essentially characterized by these three components: an a-priori **model** of the environment, a **policy** (π) and a **value function** (V).

- The model, if known, provides the agent with a-priori information regarding the environment. Since it can be either provided or not, model-based and model-free-based approaches can be introduced.

- The policy is the deterministic/stochastic mapping between the observed state and the action the agent will perform.

- The value function represents an expected cumulative value of the reward starting from the current state. While the reward is just an immediate feedback from the environment, the value function shows how the current action will also affect the future rewards.

The purpose of Reinforcement Learning is for the agent to learn an optimal (or nearly optimal) policy [3.2] that maximizes the expected cumulative reward.

$$\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0,1], \pi(a,s) = P[a_t = a \mid s_t = s] \tag{3.2}$$

Hence, the agent must learn to reason about the long-term consequences of its actions, although the immediate reward associated might be negative.

Reinforcement Learning is powerful thanks to the use of samples to optimize performance and function approximation to deal with large environments. Therefore it is exploited in situations where:

- the environment can be described according to a certain model but an analytical solution to the problem under analysis cannot be retrieved.

- the only way to collect information about the environment is to interact with it actively.

3.4.2 Q-learning

Q-learning is one of the model-free Reinforcement Learning algorithms that allows the agent to learn the real-valued quality of an action in a particular state. It aims at finding the optimal policy for decision making that maximizes the expected value of the total reward over any and all successive steps, starting from the current state. "Q" refers to the function the algorithm consider in order to estimate the quality of a state-action combination: $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. At each time t, the agent selects an action a_t, observes an immediate reward r_t, enters a new state s_{t+1} and Q is updated. The core of the algorithm, described in Algorithm [1], is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information [3.3].

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \tag{3.3}$$

- r_t is the reward achieved when the agent moves from state s_t to s_{t+1}
- $\alpha \in (0,1]$ is the learning rate, i.e. a tuning parameter that influences to what extent newly acquired observations override old information. When $\alpha = 0$, the agent will learn nothing, exclusively exploiting prior knowledge. On the other hand, when $\alpha = 1$, the agent will consider only the most recent information, ignoring prior knowledge.
- $\gamma \in [0,1]$ is the discount factor, i.e. a parameter that determines the importance of future rewards. When γ equals 0, the agent will act short-sighted by only considering current rewards. Viceversa, when γ approaches 1, the agent will strive for a long-term high reward.
- $(1 - \alpha)Q(s_t, a_t)$ is the current value weighted by the learning rate.
- $\alpha\gamma \max_a Q(s_{t+1}, a)$ is an estimate of the optimal future value that can be obtained from state s_{t+1} (weighted by learning rate and discount factor)

Summarizing, the agent explores the environment through trial and error, hence by trying out different actions at different states, learning the expected reward and eventually updating the Q-table. Each Q-value is the maximum expected cumulative reward the agent can achieve by leveraging action a_t at state s_t. Once all Q-values are known, the agent will select at each state the action that provides the highest expected reward; this procedure is called exploitation.

3.4.3 Deep Q Networks

The Deep Q Network algorithm is an enhancement of the previously introduced vanilla Q-learning algorithm. The table that maps the state-action pair to a Q-value is now replaced by a neural network that will instead perform state $\rightarrow$ (action, Q).

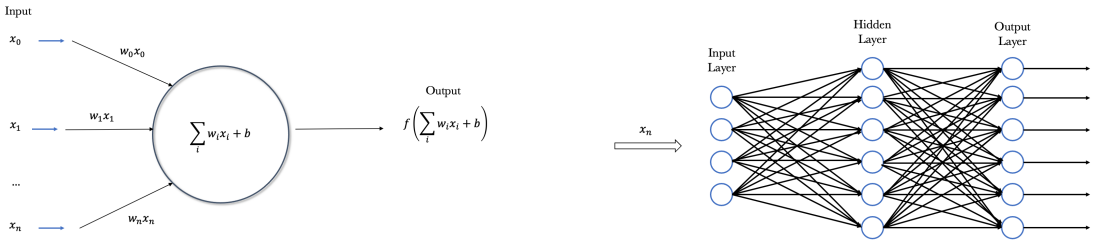

Figure 3.1: On the left, the architecture of a single neuron is reported. On the right, an example of flat fully connected network is shown.

The learning process exploits two neural networks called main and target. They have the same architecture but different weights; periodically the weights from the main are copied to the target one. This leads to higher stability and learning effectiveness.

Input nodes refer to the input state of the agent, while the output nodes represent a possible action the agent can perform. The value inside an output node is the action's corresponding Q-value.

Both networks' update is achieved through Experience Replay, i.e. the act

of storing and replaying old states. This is a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent one to proceed. This approach has proved to minimize correlation in the observation sequence and to avoid skewing the dataset distribution of different states.

The main advantage of Deep Q-learning with respect to the vanilla version is scalability. The traditional Q-learning approach relies on a look-up table system, i.e. the Q-table, to determine at each state which action to perform, which has $\mathcal{S} \times \mathcal{A}$ entries. If the environment presents large or even continuous state and/or action space, this solution may become completely unfeasible, badly affecting memory allocation and the computational time needed to explore each state. This problem can be effectively counteracted by approximating these Q-values with machine learning models such as neural networks, giving birth to the Deep Q Networks approach. The DQN algorithm applying the concept of main/target networks and experience replay is presented in Algorithm 2.

Further knowledge beyond the scope of this work can be retrieved in [10].

Chapter 4

Problem Formalization

As introduced in chapter 1, this thesis aims at the development of a machine learning solution for UAVs' optimal positioning and its integration within a realistic network simulation.

In order to address the aforementioned issues, the problem and the approach proposed in this work are investigated and discussed from a formal perspective.

4.1 Deep Q Networks

Consistently with the former research on existing approaches on UAV Network Design introduced in Chapter 2, Reinforcement Learning is the most promising Machine Learning approach for tackling the proposed optimization problem. Moreover, as previously stated in the background on mobile distributed networks in chapter 3, Markov Decision Processes (MDPs) are the most common and convenient mathematical framework to model decision-making problems where outcomes are partly random and partly under the control of a decision maker.

According to the results provided in [10], Deep Q Networks has been chosen as the most appropriate Reinforcement Learning approach for the

scenario discussed in this work. It has been preferred over the classical vanilla Q-learning algorithm for the intrinsic structure of the problem itself: the deployment of a look-up table for the Q values would badly affect the scalability of the solution proposed, which already suffers a relatively large observation space and the need of a trade-off in terms of environment discretization.

Specifically, a centralized architecture has been considered, which is made of multiple agents, i.e. UAVs, and a base station (BS). All UAVs will periodically send their information to the BS, which in turn will aggregate and process them, selecting the best action each agent should perform in order to achieve a common goal. A fundamental assumption is that the communication link among the UAVs and the BS is fully reliable, hence with no packet loss.

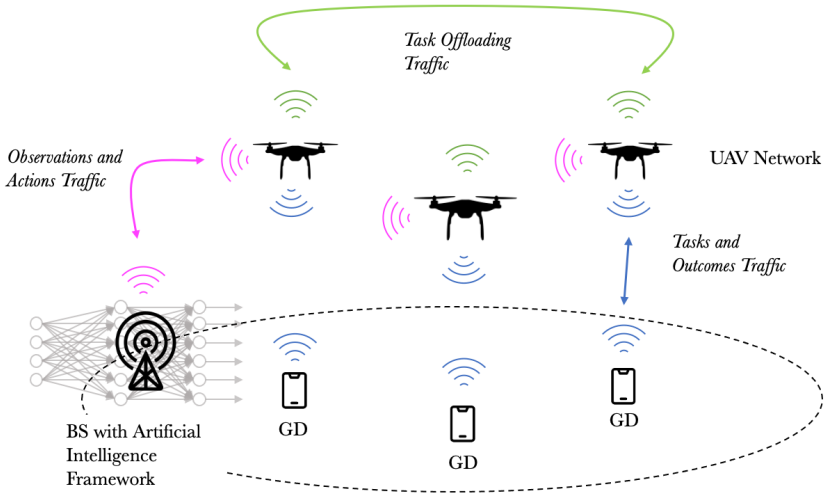

Figure 4.1: Centralized Architecture scenario proposed. The WiFi network interfaces are represented color-wise, along with the arrows representing the different types of traffic generated

4.2 Observation Space

An observation is the arbitrarily defined set of information that the agent is able to gather from the environment. The design of the proper observation space should take into account not only the information needed to achieve a final goal but also whether the agent could realistically gather them or not. For instance, the proposed scenario considers a set of 3 Ground Devices (GDs), each one asking for computational service. Assuming a realistic Air-to-Ground communication link between a UAV and the GD under service, the transmitting power should be as low as possible in order to improve battery consumption and avoid the introduction of jamming/interference in the spectrum shared with other services.

In conclusion, according to these assumptions, it would be unrealistic and unfeasible to assume that each agent knows everything about the environment. Hence, the agent should be able to provide the BS with partial and local information, such as its own position, the position of the GD being served, its level of activity and the number of tasks that have been successfully served.

In this work, the observation consists in two boxes, respectively of size nDrones x 6 and mGroundDevices x 6. At each evaluation step, these boxes are filled according to the information aggregated by the BS and eventually forwarded to the RL algorithm through the ZMQ inter-process communication channel.

The first box is composed by the following information for each UAV considered:

- NodeID: this is the identification number of a specific UAV belonging to the network. All the following information will be related to this specific node.

- Timestamp: this is the time instant of the last update message that has been received by that specific drone. This information can be useful to manage and counteract inconsistencies among the information

aggregated by the BS and the actual state of the drone under analysis. Inconsistencies may arise in case of bursts of packet loss or an inadequately set periodic update and periodic polling.

- Number of Tasks Received and Served: this information is gathered by accessing the Traffic Generator Application running on that node, it states how many tasks have been received and how many of them have been successfully served at that specific time instant. This information can be useful to monitor whether a UAV is overwhelmed by the task rate of a certain Ground Device with respect to its processing rate.

- Position: this information is gathered by accessing the Waypoint Mobility Model installed on that node. This information expresses the coordinates of the position of a specific UAV in the continuous 2D space. Its discretization will be performed directly by the RL algorithm.

Similarly, the second box regards the state of each Ground Device taking place into the network. Since Ground Devices are not supposed to communicate directly with the Base Station, this information are provided by UAVs. If no UAV is serving a certain GD, the BS will not be aware of the updates.

However, one of the main assumptions of this work is that an offline discovery process has been performed in advance, hence the existence and the location of each GD is provided at the begin of each simulation. This box is composed by:

- NodeID: this is the identification number of that specific Ground Device.

- Timestamp: this is the time instant of the last update received. As previously said, this information is useful to check if the BS is up to date with respect to the actual state of the GD.

- Served By: this is the ID of the UAV that is serving that specific GD. This information can be exploited to monitor the overall coverage in

time achieved by UAVs and make the RL algorithm deduce whether a single UAVs is serving more GDs at the same time.

- Task Rate: this is the rate of the Poisson Process describing the task generation, it expresses how many tasks per second the GDs is supposed to forward on average. If the UAV serving that GD has a processing rate much lower than this parameter, some task loss will be inevitably experienced, unless other UAVs come in assistance.

- Position: this is the set of coordinates related to that GD's position at that time instant. In this work, a constant GD position is assumed over the whole simulation. Therefore, since the initial position is known a priori, this information is useless but leaves open the possibility to increase the complexity of the scenario and allow the GDs to have a dynamic behaviour.

4.3 Action Space

The design of the action space is another fundamental aspect that must be addressed while designing the DQN algorithm. An action represents the ways the agents are allowed to interact with the environment. Consistently with the purpose of this work, an action corresponds to a discrete movement the agent can perform within the two-dimensional space that defines the environment.

The number of possible actions and their complexity have a significant impact on the performance of the system; an increasing number of actions inevitably requires additional complexity in the neural network, not only on the output layer but also in the hidden layers, in order to support them and lead to an effective policy. In facts, since a multiple agents scenario is considered and the proposed architecture is centralized, only one network will be responsible for policy training and inference of all the agents in the

network. Therefore, the network must be designed to support states and actions for all the agents of the system, resulting in bad scalability.

Future work may be oriented to realize a distributed architecture in which each agent is provided with his own network and therefore responsible for its own actions.

In this work the following trade-off has been selected: the action space is a nDrones x 1 box in which each entry expresses the index of an action array. Accessing this array at the proposed index, each agent is informed on which type of action it should perform according to the policy in use. Five possible different actions are considered: a movement in the four direction and a stay-still action. Once the BS notifies each agent on which action it should leverage, the discretization step parameter and the Boundaries Handler are exploited to translate the request in an actual position change in the ns-3 continuous space.

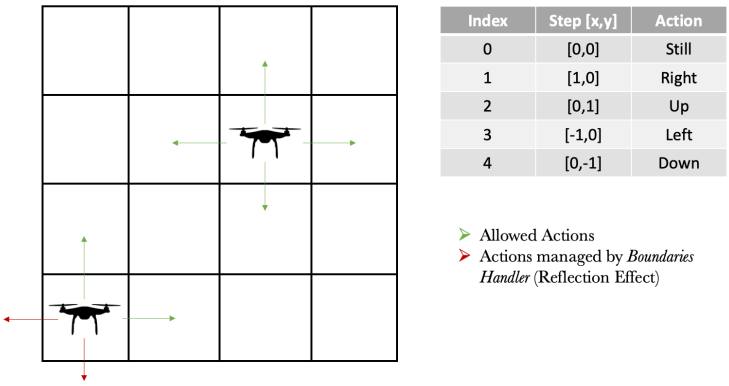

Figure 4.2: Discrete Action Space of the model Markov Decision Proces. At every time step, one of the five possible actions is performed. If the action overcomes the system boundaries, the Boundary Handler will simulate a reflection effect (Algorithm 5)

4.4 Reward Function

The reward function is the most critical feature of the Markov Decision Process design. The reward value associated to each state transition and action pair is the core of the reinforcement learning approach: the fundamental aim of the algorithm will be the maximization of the expected cumulative reward by exploration of the environment and exploitation of acquired knowledge. Furthermore, the performance of the training process strongly rely on how clear the (transition, action, reward) set is expressed at each evaluation.

A significantly long trial and error procedure has been performed in order to determine the optimal reward function for the scenario proposed in this work. Part of these experiments will be discussed in the following sections.

The optimal reward function that has been selected is a binary function that computes, for each ground device, the minimum distance with respect to all UAVs and returns 1 if this value is below a certain threshold, otherwise it returns 0. The final behavior that is expected from this reward function is the minimization of the distance among a Ground Device and the nearest UAV, hence leading to coverage maximization.

The experiments showed that sub-optimal performance are achieved by the training process when the reward function is as much simple as possible; an increase in its complexity leads to improper behavior of the agents during inference procedure. Future work may be devoted to deeply investigate the reasons of this critical complexity scalability and define a methodic strategy to identify the optimal solution.

4.5 Neural Network Design

In the literature there are several examples of neural network design processes for optimization purposes, the results carried out by authors in [23] have been of great inspiration for this work.

According to the features and purposes of the system described up to now, a neural network has been selected as the best solution to model the reinforcement learning policy. This network should accept as input information about the position of UAVs and Ground Devices within the environment and eventually provide on output the Q-value associated with each possible action the agents can perform. Therefore, starting from the relative and absolute positions of the agents within the environment, the network will be trained in order to select at each time the optimal action, i.e. the action that maximizes the expected cumulative reward of the system.

Since the number of UAVs and GDs is supposed to be an arbitrary parameter of the system, the input layer should be designed to be agnostic with respect to that. The best way to address this problem is to organize the position information in two-dimensional maps, i.e. sparse matrices with ones occurring on the cells corresponding to the proper discretized position of each agent. The same strategy could be applied also for other variables, depending on the purpose of that specific scenario, e.g. a map expressing the level of activity of the ground devices or the amount of resource utilization of each UAV.

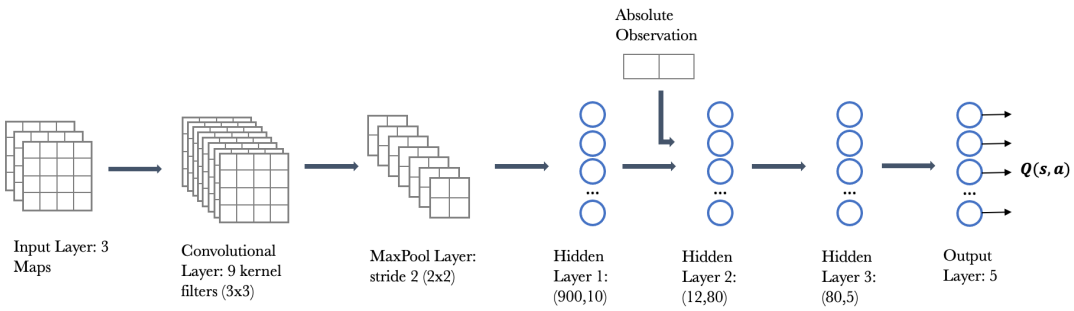

Figure 4.3: Neural Network architecture proposed in this work, from input, i.e. the three state maps, to the output, i.e. the Q value associated to each possible action. The absolute observation provided in the middle of the chain corresponds to the position of the specific agent whose action is going to be determined.

From Machine Learning Theory, Convolutional Neural Networks (CNNs) are indeed the best solution to address problems with two-dimensional structures as input, e.g. they are renown for image classification.

The main perks of CNNs are:

- Local receptive fields: only a subset of cells will be connected to a hidden neuron, this hugely reduces the complexity of the network and exploits the idea that informative elements may be pretty localized rather than uniformly distributed.
- Shared weights: the connections from the local receptive field to each neuron have all the same weights, hence with the convolution kernel acting as a filter. This allows to significantly reduce the complexity of the network in terms of parameters.
- Pooling: pooling layers allow to simplify the information output from a convolutional layer, performing a sort of downsampling. This allows to avoid the exponential increase of the number of parameters when moving from a layer to another.

After an extensive trial and error procedure, the neural network architecture best suited for the purposes has been identified and it is presented in the following. It is reasonable to consider this solution as the best trade off among computational complexity and training performance, although future work could be devoted to further experiments and robustness improvements.

The input CNN is made of one convolutional layer with 9 kernel filters of 3x3 size and one max-pooling layer with kernel size 2x2 and stride of 2. Afterwards, the output of this CNN is passed to the last part of the network, made of three fully connected layers. The two hidden layers are respectively made of 900 and 80 neurons, they allow the implementation of reasonably complex functions while keeping the computational complexity bearable. The output layer has exactly the action space dimension; each neuron refers to one of the possible 5 actions an agent is allowed to perform and the value

associated to that neuron is the Q-value, hence the reinforcement metric related to that action (Figure 4.3).

4.6 Training Process

As introduced in Chapter 3, the aim of the training procedure is for the designed deep Q network to learn a policy that correctly maps the current state of the agent to the action that maximizes the expected cumulative reward. The general algorithms for Q-learning and its DQN enhancement have been introduced in Algorithms 1 and 2. Since a centralized architecture has been considered, a single network will be considered as well; hence, the aim of the training process design is to achieve a unique and scalable policy that is shared by all the agents in the system and fosters their cooperation.

During the training procedure, for each episode, the agents will interact with the environment at the same time, mutually affecting the results of their decisions. The actions, for each evaluation step, are selected according to a ε-Greedy Policy, which has been proven to be the best balance for exploration and exploitation procedures. At the beginning of the training process the agents will purely explore the environment, performing random actions only. Eventually, the noise will be gradually reduced, up to the end of the training process in which the network should fully exploit the acquired knowledge. For each (state, next state) transition, a certain value of cumulative reward is associated and the whole set is appended to the replay buffer, whose batches are exploited to train the network. Eventually the target network's parameters are updated according to the Polyak averaging.

The parameters deeply affecting the training performance are:

- Number of Training Episodes: the appropriate amount of episodes needed for the network to achieve good training performance can be found by trial and error procedures.
- Simulation Time: each episode corresponds to an entire simulation of

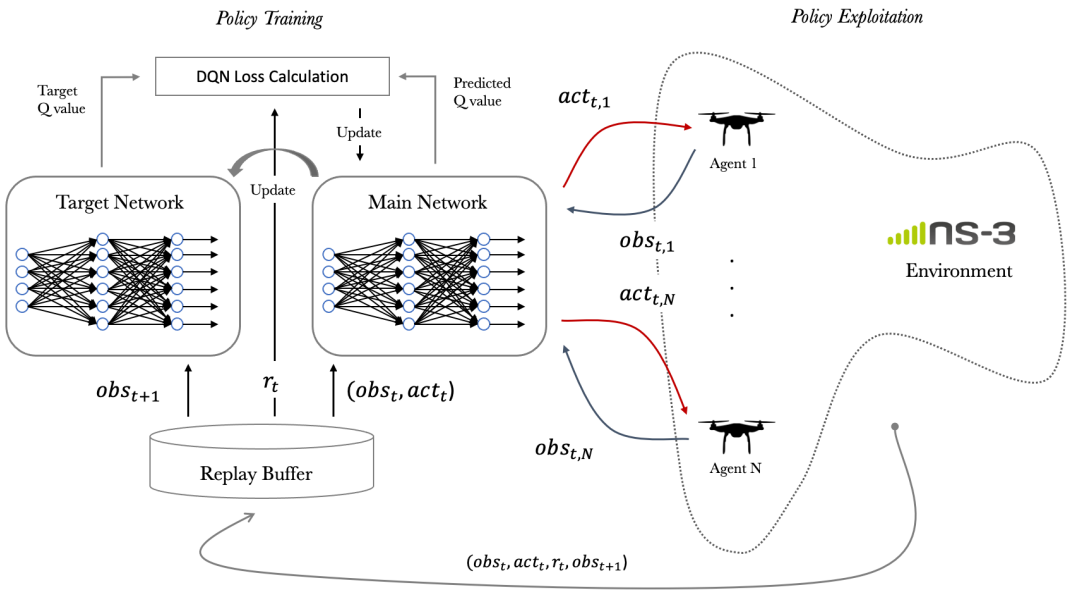

Figure 4.4: DQN Training Process for the proposed multiple agents system. On the right, the agents provide their observation and receive a corresponding action according to the current policy determined by the DQN algorithm. On the left, the training scheme is shown, which relies on the replay buffer and the two main and target networks as discussed in Chapter 3

a certain duration. Longer simulations allow better exploration of the environment but it may significantly slow down the training process.

- Evaluation Step Time: this is the periodic sampling performed by algorithm on the agents, observations are collected, reward is computed and actions imposed.

- Discount Factor $\gamma \in [0,1]$: the discount factor is a parameter that expresses the importance of cumulative future rewards over the immediate instantaneous ones. When γ equals 0, the agent will act short-sighted by only considering current rewards. Viceversa, when γ approaches 1, the agent will strive for a long-term high reward.

- Epsilon $\varepsilon \in [0,1]$: the epsilon parameter is involved in the ε-greedy

policy for the selection of the action to perform at each evaluation step. ε can be interpreted as the probability according to which a random action must be performed (exploration) rather than one based on the acquired knowledge (exploitation).

- Pure exploration limit: previous studies proved that a pure exploration phase at the beginning of the training process may lead to better performance. Starting from this episode, at each evaluation, the ε will be decreased, e.g. according to $\varepsilon_{t+1} = \varepsilon_t * 0.99995$.

- Minimum Epsilon ε_{min}: this is the minimum value imposed for ε, even in the pure exploitation mode there will be a certain number of random actions according to this probability. This can be interpreted as an intentional addition of noise in the action performed, in the following chapters the benefits of this action will be discussed.

- Batch Size: the batch size defines the number of samples that will be propagated through the network. Using many small sets of samples for training the network enables lower memory requirements and typically faster training. On the other hand, the main disadvantage of mini-batches is the less accurate estimate for the gradient, which may lead to unexpected fluctuations on the results. The proper value must be determined for each specific scenario by trial and error procedures.

- Polyak Weight τ: this parameter is involved in the Polyak Averaging process performed when updating the weights of the target neural network according to those from the main one.

- Experience Replay Buffer Size: the larger the experience replay buffer, the less likely correlated elements will be sampled, hence the more stable the training of the network will be. However, a large experience replay buffer also requires a lot of memory and it might produce a much slower

training. The proper value for this parameter depends on the specific scenario, it should be determined by a fine tuning procedure.

- Bounds and Discretization: these parameters express how fine the discretization of the continuous space is and the overall boundaries of the environment. A rough discretization leads to lower complexity in the network but lower accuracy when re-mapped onto the continuous space. A proper trade off must be determined in order to achieve the best performance, however it is clear that, no matter how fine the discretization is, a quantization error will always occur.

Chapter 5

Tools

5.1 Network Simulator 3 (ns-3)

ns-3 is a free and open-source discrete-event network simulator based on C++ programming language and Unix network architectures.

Although ns-3 is based on C++ at its core, every API can be exported as Python bindings, allowing users also to write python scripts rather than C++ programs. In addition, components like pybindgen, castxml and pygccxml are provided by default and allow the C++ libraries' parsing and binding generation.

One of the hallmarks characterizing ns-3 is the realism of its models, making implementation closer to the actual software they represent. In fact, ns-3 has been designed with the purpose of simulating networks architectures in a way that resembles as much as possible the Unix internal and application interfaces (e.g. device drivers and sockets). This also emphasizes ns-3's emulation capabilities and its feasibility to be used on testbeds with real devices and applications.

The WAF build system is a Python-based framework that ns-3 exploits for configuring, compiling and installing applications.

In the following, the main components of ns-3 simulations are presented:

- **Nodes:** They represent a network host; hence they can realistically contain lists of network devices, applications and objects in general. This intuitive representation is made possible by the aggregation system, which will be discussed later on.

- **NetDevices:** They implement functionalities to send and receive data. Sub-classes of NetDevices include PointToPoint, CSMA and Wifi. The simplest ones come by default with a MAC and Physical layer implementation, while more complex ones, like Wifi LANs, require explicit definition from channel properties up to MAC protocols' parameters.

- **Channels:** They define a connection between a set of nodes. These nodes, sharing the same channel object, will therefore belong to the same broadcast domain. For instance, all nodes in a LAN share a single CsmaChannel and all nodes in a WiFi network should have the same YansWifiChannel.

- **Packets:** They represent the quantum unit for information exchange among nodes. Like real communications, packets are made by headers, trailers and payload, all stored in a byte buffer. In addition, they can also have tags and metadata added to them:

  - **Tags:** set of arbitrary, user-provided data structures (e.g., per-packet cross-layer messages, or flow identifiers)
  - **Metadata:** describes types of headers and trailers that have been serialized, optional and usually disabled by default.

- **Applications:** They represent the high-level traffic generators, creating packets, hence forwarding them to lower layers of the stack, or listening for packets and eventually processing them. The simplest examples of application are:

  - UdpEchoApplication: it generates packets of a certain size with

an arbitrary time interval. This is usually deployed to test if the communication is properly functional.

- OnOffApplication: it generates traffic according to an on-off pattern. During the "Off" state, no traffic is generated. During the "on" state, a constant bitrate (CBR) traffic is generated, characterized by the specified "data rate" and "packet size".

- **Mobility Models:** They specify the behaviour of nodes in terms of mobility. Nodes can be stationary or moving according to a certain model, e.g. RandomWayPoint or RandomWalk.

Since, by definition, simulation time moves discretely from event to event, a scheduler has been introduced to manage the event execution fully. It enqueues new events when created and dequeues them whenever the simulator engine is ready to move the system to its next state. Thus, the engine maintains a sorted list of future events which are progressively dispatched and served by calling the proper event handler subroutine. In ns-3, every member function for any object has the potential to be an event handler since it can easily create and schedule new events.

All ns-3 objects inherit from their base class ns3::Object several useful features such as:

- dynamic run-time object aggregation
- an attribute system
- smart-pointer memory management

Given a certain entity, such as a node, aggregation allows to retrieve pointers to all its connected objects (e.g. Mobility Model, NetDevices or Applications) without explicitly modifying the base class to do so. This feature enables flexibility in design, efficiency in memory usage and simplicity in object representation.

An Attribute represents a template value of the system that can be connected to an underlying variable or function of the corresponding object. The attribute system is a unified and agile way to handle initialization and access to the variables of an object (e.g. setting or getting a variable during the execution without explicitly programming methods to access the corresponding pointers).

Smart pointers are template objects belonging to the ns-3 reference-counting system. They behave like normal pointers from a syntax perspective, but they are designed to avoid memory leaks, automatically managing the heap deallocation whenever the reference count goes to zero.

In order to produce an alignment with the common input/output standards, such as ASCII or pcap traces, ns-3 has been equipped with a tracing system that enables agile post-processing of simulation data, with the aim of evaluating and monitoring the workflow and the performance of the system under analysis.

TraceSources causes a function with a specific signature to be called whenever a certain event occurs, e.g. packet reception or wifi association. The user-defined routine to execute is called TraceSink.

5.1.1 Native Modules

In the following, the main libraries that have been exploited in this work are introduced.

- core-module: it provides the main ns-3 functionalities, such as event management, the scheduling system, random variables and the attribute system.

- network-module: it manages several aspects of the network stack, such as topology management, physical layer definition, MAC and IP layer protocols, queuing systems, packet management and socket factories.

- applications-module: it contains all the helpers needed to install native applications, such as OnOff, Echo, 3GPP HTTP, etc.

- mobility-module: it provides all the objects and methods needed for position allocation and native mobility models installation, such as Random Walk, Waypoint, Gauss-Markov, Hierarchical, etc.

- internet-module: it handles the IP layer protocols, the routing management and the interface with the transport layer.

- yans-wifi-module: it provides an accurate implementation of IEEE 802.11 standard, based on the Yet Another Network Simulator project [24].

- propagation-module: it enables the implementation of a realistic channel model in terms of loss and delay experienced, such as FixedRSS, Friis Propagation, 3GPP indoor/outdoor models, etc.

- flow-monitor-module: it provides a user-friendly system that allows to gather aggregated flow-based statistics over the whole simulation.

5.1.2 New Modules

In addition to the modules natively available and previously introduced, new modules have been produced from scratch as part of this work.

These modules have been implemented according to a scenario agnostic approach and consistently with the official contribution guide provided in [25]. This strategy enables an efficient and agile reuse of these modules for future works, such that they can be deployed for even completely different scenarios. Furthermore, they can even be published as an open-source contribution to ns-3, such that anyone can exploit them without the need of modifying the ns-3 original source code.

Here they are introduced:

- air2ground-propagation-loss-model: this module provides a probabilistic propagation loss model that realistically resembles an actual Air-to-Ground communication channel. The inspiration for the algorithm and its parameters comes from the work in [6]. This model is an enhancement of the native Friis propagation-loss-module, introducing a stochastic characterization of an extra path loss depending on the urban environment and the elevation angle of the flying vehicles. Specifically, the authors in [6] empirically retrieved several sets of parameters, each one related to a specific urban environment density. These parameters allow to compute the probability of Line of Sight for a specific communication link and to describe the extra path loss affecting the propagation as a Gaussian random variable with certain mean value and variance.

- traffic-generator-module: this module provides a server/client pair of applications that allows to model a realistic task/processing traffic generation process. From the client perspective, traffic is generated according to an arbitrary inter-packet time. The default definition of the inter-packet time is an Exponential Random Variable with parameter λ, resulting in a Poisson Stochastic Process description of the traffic generated by the client. From the server perspective, once a packet has been received, it is assumed that a certain amount of time for its processing must be allocated. This allows to resemble the computational cost required for the server to perform a certain task forwarded by the client. The default description of this processing time is again a Poisson Process with parameter μ. For the sake of simplicity, the server is assumed not capable of parallel processing; a new task will be processed once the last one has been successfully accomplished. Eventually, once the processing operation has been completed, the server forwards a packet back to the client. This packet's payload will provide an outcome related to the previously assigned task.

- ai-interface-module: this module consists in the cooperation among three

different applications, a client/server pair of traffic generators and a gym interface framework. The objective of the module is the full implementation of the interface among ns-3 and the OpenAI Gym algorithm. The client application is meant to run on the same node performing server operations for the traffic-generator module. In fact, a periodic interrupt callback system is deployed for the sampling of that node's statistics, e.g. its position, the number of tasks received/served and the ground devices it is serving along with its statistics. These statistics will then take part in the observation for the reinforcement learning algorithm. Once this information has been aggregated and serialized, it is forwarded to the server as part of a packet's payload. Therefore, the client generates a constant bitrate traffic defined according to an arbitrary constant inter-packet time. From the server perspective, these packets are processed and their information parsed and accumulated, ready to be forwarded to the AI algorithm. When the AI algorithm determines which actions the agents should perform in order to maximize the expected cumulative reward, the server will handle the production of a proper packet in order to inform the client. Eventually, the client will receive the packet, parse it, and apply the action suggested by the AI algorithm. The gym interface component is responsible for establishing the ZMQ inter-process communication channel needed for the exchange of observations, actions and rewards among ns-3 and the OpenAI Gym algorithm.

5.2 PyTorch

As declared in the official paper [26] and on its website [27], PyTorch is a Python package that mainly provides two high-level features:

- Tensor computation: multi-dimensional data structures that can run on GPUs and hence speed computational time up.

- Deep neural networks built on a tape-based autograd system, i.e. an automatic differentiation engine that powers neural network training.

Compared to other popular frameworks, PyTorch introduces a dynamic way of building neural networks based on using and replaying a tape recorder. This technique, i.e. reverse-mode automatic differentiation, allows to change the way the network behaves arbitrarily, with zero lag or overhead. Furthermore, its GPU-Ready Tensor and Neural Networks libraries are the hallmarks of this module, providing efficient accelerated computing capabilities and agile neural networks definition.

In conclusion, PyTorch is usually chosen as deep learning framework because it enables fast, flexible experimentation and efficient production through a user-friendly front-end, distributed training, and ecosystem of tools and libraries.

5.3 ns3-gym

In order to follow the major trend in network research to merge ns-3 simulations with RL algorithms, the authors in [28] introduced the ns3-gym module. This toolkit has been designed to enable the interaction among OpenAI Gym and ns-3 network simulator by establishing a ZMQ-socket-based Inter-Process Communication channel.

The purpose of the authors was to define a scenario agnostic framework where the realistic simulation provided by ns-3 could be exploited as the environment of a RL algorithm where the agent performs exploration and exploitation procedures. In order to make policy learning as much effective as possible, real-world training would be necessary; however, this is usually not feasible because of testbed cost, setup time and safety reasons. Since ns-3 has been designed to reflect communication systems' behaviour realistically, this model fosters the prototyping of innovative RL-based networking solutions without buying and setting up costly testbeds. On the other hand, since ns-3

natively supports testbed implementation, this module allows the seamless porting of the simulated RL solution onto a real-world system, in order to eventually evaluate the consistency with the simulated scenario.

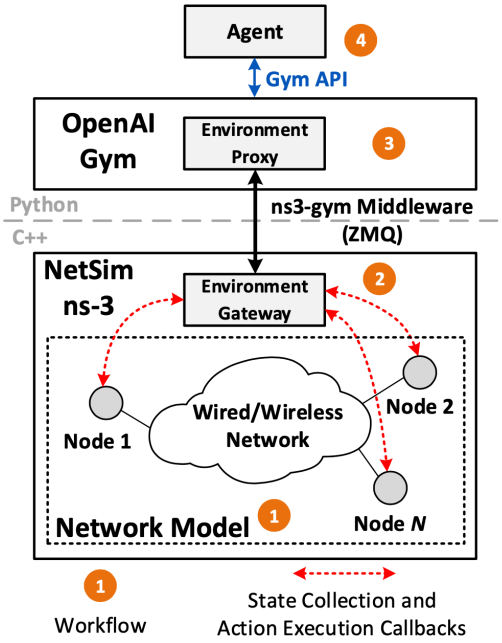

Figure 5.1: Architecture of ns3-gym framework [11]

Figure 5.1 depicts the architecture of ns3-gym. It consists of an ns-3 framework (1), an OpenAI Gym framework (4) and ns3-gym middleware (2 and 3). The first defines the environment and the agents interacting with it; the second defines the reinforcement learning algorithm that, basing on the observations, will provide the optimal action that maximizes the cumulative reward for the agent. The middleware between the two frameworks provides a ZMQ socket inter-process communication channel that enables messages exchange among them (i.e. observations and actions). It is composed of the Environment Gateway (2) and the Environment Proxy (3).

On the ns-3 side, the Environment Gateway is responsible for gathering the agents' states and interpreting the actions received from the RL algorithm. The main APIs exposed in ns-3 for the gateway implementation are:

- *GetObservationSpace()* and *GetActionSpace()* initialize the observation/action data structures and establish a common ground for both frameworks to exchange messages properly.
- *GetObservation()* gathers the simulation variables involved in the observation definition and collects them in the predefined data structures, ready to be forwarded to the AI framework;
- *GetReward()* evaluates the reward achieved during the last agent's state transition;
- *GetGameOver()* checks whether the agent has reached the game-over condition or not, hence if the episode must be terminated or not;
- *ExecuteActions(action)* handles the reception of the action data structure from the AI framework and maps it into the corresponding action the agent must perform.

Similarly, on the AI framework side, the Environment Proxy will manage the reception of the observations from the ns-3 environment and eventually forward them to the RL algorithm through Gym APIs. Viceversa, it will collect the action that has been selected by the RL algorithm and forward it to the ns-3 simulation through the ZMQ communication channel.

Chapter 6

Implementation

The previous chapters have been devoted to an introduction of the problems tackled in this work, the approach and strategy proposed and the tools exploited to achieve some results and conclusions. This chapter highlights the methodology that has been pursued to design and implement the proposed solution.

6.1 Python Environment

Two main Python frameworks have been implemented; the first handles the policy training and the latter exploits a previously trained network to perform inference simulations.

Both of them obviously share and implement the same problem formalization structure discussed in Chapter 4.6: for each timestep, a certain observation space is gathered from the ns-3 environment, it is processed, an action space is returned, and, eventually, the reward function corresponding to this state transition is evaluated.

In the following, a description of the policy training framework is provided.

The first step regards the processing of the information boxes coming from ns-3; the most significant operation is the discretization process that maps

the continuous position information to the discrete maps that will be later passed to the CNN for training. Specifically, three maps are produced from each observation: UAVs' position, Ground Devices' position and their level of activity, i.e. the task rate of the Poisson distributed traffic generation process.

The next step is the selection of the action to perform; according to the ε-greedy policy algorithm, an instance from a uniform random variable in [0,1] is extracted and compared with the ε parameter defined during the initialization. If the value is lower than ε, a random action is performed (exploration), otherwise exploitation is performed. In detail, the current state is passed to the neural network, along with the position of the specific UAV that is supposed to perform the action, and the final action index will be determined by the neuron of the output layer with the highest Q-value.

The ns3-gym method *step(self, action)* will exploit the ZMQ inter-process communication channel to forward the action to the ns-3 framework for its actual execution. Once the action has been performed, a new complete observation of the next state is gathered, along with the reward achieved with the current state transition. Eventually, the set state, next state, action and reward is stored in the Replay Buffer for later policy training.

Once the Replay Buffer becomes full, the training procedure starts: a batch of a certain size is sampled from the replay buffer and passed to PyTorch methods that will handle the training of the main network. Eventually, Polyak averaging is performed in order to update the target network parameters and the ε is updated according to the strategy imposed during initialization.

6.2 ns-3 Environment

As already stated in the previous chapter, the definition of an ns-3 environment has the purpose of bringing as much realism as possible in the simulations. Each element of the system must be defined as a node of a

network with a specific set of physical layer characteristics, network devices, protocols and applications.

In this chapter, the main features of the proposed ns-3 framework are introduced.

6.2.1 Network Architecture

The scenario proposed in this work consists of three types of nodes: Drones (UAVs), Ground Devices (GDs) and Base Stations (BSs). UAVs will be able to establish three different communication links, the first with the GDs for task processing, the second with other UAVs for task offloading and the third with the BS for periodic observation/action exchange. In order to achieve them, each UAV has been equipped with three IEEE 802.11a network devices:

- WiFi 802.11a - Infrastructure Mode: the UAV is configured to act as an Access Point that will be able to serve one or multiple GDs at the same time acting as WiFi Stations.
- WiFi 802.11a - Infrastructure Mode: the UAV is configured to act as a WiFi station which will pair with the corresponding AP interface on the BS.
- WiFi 802.11a - Ad-Hoc Mode: the communication among UAVs is implemented with an Ad-Hoc interface for simplicity and consistency with the absence of classification among UAVs.

From the previous information, it is clear that both the BS and the GDs are equipped with a WiFi 802.11a interface in Infrastructure mode, respectively acting as AP and STA.

Each one of these devices relies on a specific Physical Layer definition characterized by Transmission Power, Frequency Channel Number, Propagation Delay Model and Propagation Loss Model. The WiFi Channel Numbers

are selected in order to avoid inter-channel interference among the three network devices, hence sufficiently spaced to avoid spectrum overlapping. The transmission power can be set arbitrarily at the beginning of the simulation; the default values are consistent with the real hardware capabilities of the devices: UAVs and GDs should transmit with the lowest power possible in order to maximize the energy efficiency while the BSs is realistically allowed to support higher values for the transmission power [29].

6.2.2 Channel Model

For all Air-to-Ground communication links, the channel model presented in [6] and described in Algorithm 4 is adopted. The Urban Density and the Elevation Angle are arbitrary parameters that can be imposed at the beginning of the simulation during the initialization procedure.

Since the Air-to-Air communication channel is supposed to be affected by fewer impairments, the ideal Friis Propagation Loss model is implemented, as described in Algorithm 3.

6.2.3 Mobility Models

As already stated in the Problem Formalization (Chapter 4.6), the UAVs must be able to receive messages from a BS containing the action they should perform according to the policy imposed by the Reinforcement Learning algorithm.

ns-3 offers a native mobility model that allows to asynchronously schedule the next position a certain node should reach, i.e. WayPoint Mobility Model. The initial position of all UAVs is selected according to a Random Disc Position Allocator: they lay on a circle with the origin of the reference system as its centre, arbitrary constant radius $|\rho|$ and angle θ extracted from a uniform random variable in $[0,2\pi]$. This disc will also be interpreted as the boundary of the whole environment, such that the agents should not go

beyond it. All the following positions, aka waypoints, will be scheduled in the *ai-interface-client* application whenever a packet is received from the BS. Since Python AI framework operates within a discretized bi-dimensional space, each action will correspond to moving an agent from a cell to an adjacent one; hence, every new waypoint will correspond to covering a distance equal to the space discretization factor imposed as initialization parameter.

All GDs and the BS are instead characterized by a Constant Position Mobility Model; future work may be devoted to the introduction of dynamic behaviour for the GDs and to training performance comparison with respect to this simplified scenario.

An important assumption that is important to underline is the fact that the BS is aware, at the beginning of each simulation, of the initial positions of all UAVs and GDs. It is like having performed a prior offline exploration of the environment in order to identify which GDs are asking for service and store their location. Future work may be oriented to removing this assumption and moving to a more realistic solution in which this discovery is performed in real-time.

6.2.4 Applications

As already introduced previously, there are three different types of traffic that are generated within the proposed scenario. One regards the task/outcome exchange among UAVs and the GDs under service, another is related to the periodic observation/action exchange among UAVs and the BS, and the last one is an ideal constant bit rate traffic that resembles a constant information exchange among UAVs in the same radio range.

Specifically, the applications installed on all UAVs are:

- *traffic-generator-server*: this application is devoted to task reception from GDs. Whenever a task is received, a simulated processing time needed for its execution is extracted from a random variable arbitrarily

distributed, e.g. Exponential by default with parameter μ. After the processing time elapsed, an outcome packet is sent back to the proper GD. In other words, this application generates a Poisson distributed traffic going from UAVs to GDs. It is important to highlight that, in order to avoid inconsistencies, each UAV is assumed not capable of parallel processing features. Hence, an internal FIFO queuing system has been implemented in order to guarantee that only one task is executed in each time instant and, at the same time, ensure that the application is able to listen for other tasks without dropping them.

- *ai-interface-client*: the aim of this application is to forward the observations to the BS periodically and to handle the reception of the action messages in return. Regarding the observations, the Inter-Packet Time can be configured as a random variable with arbitrary distribution; in this case, a constant deterministic value has been preferred. A periodic interrupt system has been implemented in order to access and update the information required for the composition of an up-to-date observation. On the other hand, whenever an action message is received, the application will handle its parsing and translation into an actual waypoint.

- *on-off-application*: the native on-off application module has been exploited to roughly simulate a constant bit rate traffic among UAVs in the same radio range, which may resemble task offloading operations or collaborative computing.

Consequently, GDs are characterized by a *traffic-generator-client* application, whose purpose is to generate a Poisson distributed traffic towards UAVs, resembling a sequence of tasks to be executed. Thanks to its agnostic implementation, the inter-packet time variable can be easily changed from Exponential with intensity λ to any preferred distribution that is supported

natively by ns-3. Concurrently, the application is able to listen for packets coming back from UAVs, which resemble the outcomes related to the previously forwarded tasks.

Eventually, the BS is provided with a *ai-interface-server* application, whose aim is to receive the observations from all the UAVs in the system, update its internal aggregated maps, establish a ZMQ inter-process communication channel with the AI framework, and forward action messages back to the UAVs.

Figure 6.1: Overview of the framework design. The block on the left presents the problem formalization in Chapter 4.6. The block on the right illustrates the information flow among the two frameworks, i.e. ns-3 and Open AI Gym.

6.2.5 Handlers

Many of the functionalities and results that have been presented in this thesis have been achieved thanks to the implementation of a set of various algorithms:

- **Handover Handler**: this routine allows to keep track in real-time of the handovers occurring whenever a UAV stops serving a certain GD and moves towards another one. The WiFi *AssociationLog* trace source is exploited: whenever an AP $\leftrightarrow$ STA pairing occurs at the MAC layer, an interrupt callback routine is executed. This routine is in charge of extracting the Node IDs involved in this operation and inform the corresponding applications. The *traffic-generator-client* application running on a GD will be informed about the new UAV in radio range, i.e. the corresponding IP address is set as the destination for all the future tasks generated. The *ai-interface-client* application running on a UAV will be informed as well in order to add the new GD to the list of the GDs currently served by that UAV. Similarly, the *DeassociationLog* trace source is exploited to perform the inverse operation: whenever the AP $\leftrightarrow$ STA pairing is lost, the corresponding GD is removed from the list of the UAV involved.

- **Boundaries Handler**: whenever an action is produced by the Reinforcement Learning algorithm, before actuating it, it is fundamental to check if this action may make the agent overcome the boundaries of the reference system. This handler provides two possible approaches on addressing this issue. The first is the possibility to rescale an action, if this results in a boundary overcome, or stay still if needed. The second option is to simulate a reflection effect: the agent reaches the boundary and the remained amount of distance to cover is performed towards the centre of the circle. The algorithm is presented in Algorithm 5.

- **Packet Monitoring**: the huge set of trace sources provided by ns-3

allows the production of output logging files that can be later post-processed in order to gather some statistics. The simulator natively offers the possibility to trace everything occurring in the system with *pcap* traces or *ASCII* logs; however, part of this work has been devoted to producing a tracing system that could export only the information that was actually of interest and therefore avoid processing irrelevant and redundant information. All the Physical Layer trace sources have been exploited, i.e. an interrupt is received whenever a packet has been transmitted, received or dropped on a certain network device. The packet involved can be inspected and its information arbitrarily printed onto an output file.

- **Mobility Monitoring**: similarly, the *CourseChange* trace source, related to the Mobility Model module, has been exploited in order to log the time evolution of the positions assumed by nodes of the system. This handler has been particularly useful in the initial connectivity and performance evaluation tests, in which a Random WayPoint Mobility Model was deployed.

- **SNR Tracing**: the Signal-to-Noise Ratio is a fundamental metric of telecommunication systems that allows to evaluate from a quantitative perspective the quality of the communication channel. This parameter is usually related to the Bit Error Rate guarantees that must be fulfilled as a requirement of a certain communication system. Tracing this quantity allows to acquire further knowledge from a technical perspective of the communications occurring during the simulation and ease up design processes if necessary.

- **Vehicle Visualizer**: the native network visualizer provided by ns-3 is definitely not sufficient to fully illustrate the characteristics of a designed system. In particular, it does not allow to change nodes' colour or icons, making the representation pretty messy and definitely not flexible or

agile. Part of the work in [30] was devoted to the development of a visualizer based on NodeJS that allows to address the aforementioned issues. A comparison among the two tools is provided in Figures 6.2 and 6.3; the visualizer from ms-van3t enables the introduction of icons for each node and an underlying geographic map that can be chosen according to arbitrary coordinates. This definitely increases the quality of experience when presenting a simulated scenario and enables the possibility to compare multiple environments, e.g. suburban or dense urban, in a clearer and more effective way.

Figure 6.2: Vehicle Visualizer from ms-van3t [30]

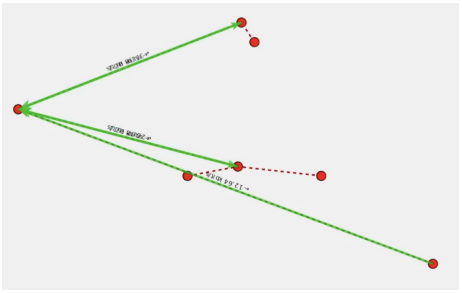

Figure 6.3: PyGraphViz native module

6.3 Post-processing Environment

A wide library of MATLAB functions has been developed in order to read, parse and process the output logs produced from ns-3.

The set of statistics that can be gathered can be classified into two main types: those aggregated over the whole simulation and those aggregated over an arbitrary time window, e.g. every 200 packets.

For the sake of clarity, the following Figures are related to simpler scenarios and have been reported just as an example of the potential of these post-processing algorithms for future work.

For the first class, the following statistics can be achieved:

- Transmission Power P_{TX} vs Throughput, SNR, Delay or Packet Loss
- SNR vs Throughput with variable Bin size

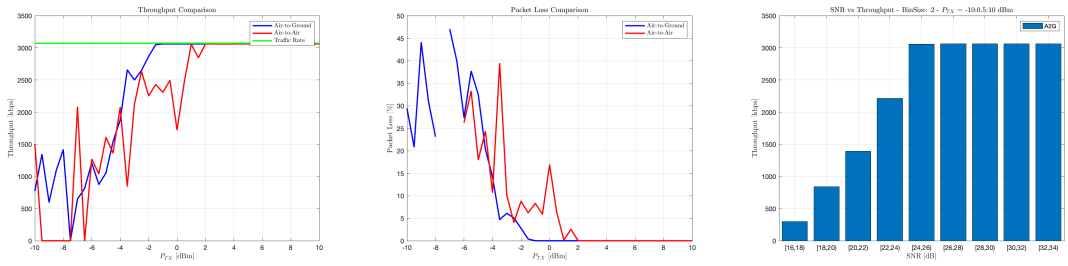

Figure 6.4: Statistics related to multiple simulation instances, each one with a different transmission power. A simple Random WayPoint Scenario and CBR traffic generation are considered

Figures in 6.4 highlight the consistency among the metrics under analysis: increasing the transmission power, the quality of the communication increases, achieving higher SNR values and consequently higher throughput and lower packet loss.

With respect to second class, the following statistics can be gathered:

- Time vs Throughput, SNR, Delay or Packet Loss

- Distance UAV $\leftrightarrow$ GD vs SNR
- SNR vs Throughput with variable Bin size
- Nodes trajectories in time

Figures in 6.5 allow to qualitatively appreciate the consistency among the relative distance of the two UAVs and the achieved SNR. Obviously, due to the 3D nature of the first plot, more effort is definitely required to make the comparison actually appreciable.

Figures in 6.6 consistently show that shorter distances between a UAV and a GD under service correspond to higher SNR achieved. Moreover, the proposed algorithm takes into account handovers too; that is the reason why multiple UAVs are shown on the left plot. The second plot highlights the lower mean value and the huge variance that characterize the throughput random process; the reason is related to the deployment of a realistic A2G Propagation Loss model and the fact that the worst-case scenario is imposed. Low elevation angles will produce higher non LoS probability and a Highrise Urban environment will correspond to harsher propagation conditions.

Eventually, it is worth mentioning the last class of statistics related to the application layer rather than the physical one. The application *traffic-generator* is considered, set to generate a Poisson distributed traffic for task forwarding according to parameter λ and task processing according to parameter μ.

- Poisson Process stair plot
- End-to-end Delay at Application layer
- Task Loss

Figures in 6.7 clearly depict the properties of Poisson Processes: non-negative, never decreasing, no deterministic information superimposed, independent and stationary increments. Consistently with the expectations, the

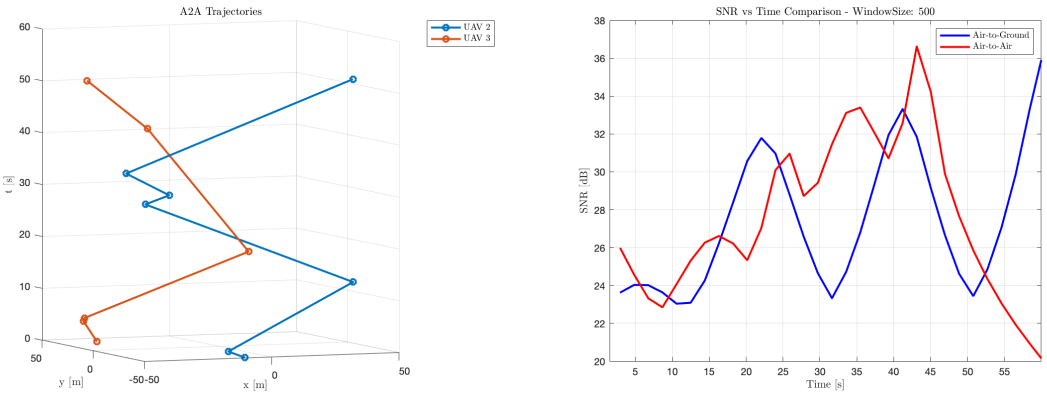

Figure 6.5: Statistics related to a Suburban environment scenario with Elevation Angle $\theta = 50°$. The plot on the left is a 3D plot reporting the trajectories of the two UAVs involved in A2A communication. The plot on the right highlights the achieved SNR in time for both A2G and A2A communications.

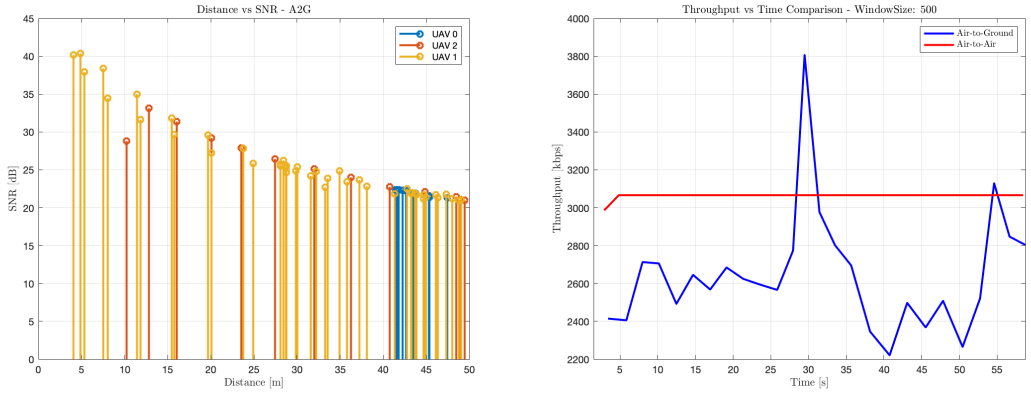

Figure 6.6: Statistics related to a Highrise Urban environment scenario with Elevation Angle $\theta = 50°$. The plot on the left compares the UAV $\leftrightarrow$ GD distance with the SNR experienced by the UAV that is serving. The plot on the right reports the achieved throughput in time for both A2G and A2A communications.

expected number of packets in the long run is directly proportional to the hazard rate.

Figures in 6.8 highlight the E2E Delay and the Task Loss experienced. The average E2E Delay for the UAVs' outcomes is smaller, probably due to the fact that smaller packets sizes have been simulated. The second plot compares the task loss with different processing rates μ [packets/s], assuming a fixed task rate λ [packets/s]. The percentage of tasks that has not been served within the simulation consistently decrease with the processing rate. If $\lambda < \mu$, the server can cope with the task rate: all tasks will be served successfully. On the other hand, if $\lambda > \mu$, the server can't cope with the task rate in terms of processing time: some tasks will be left unserved.

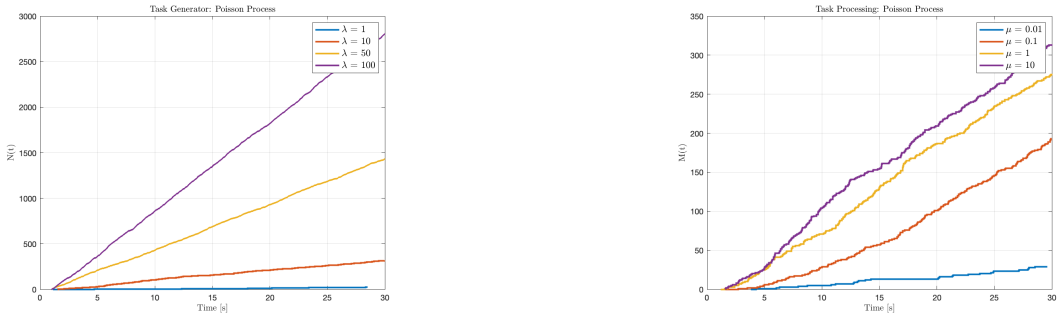

Figure 6.7: Trajectory of Poisson Processes: number of events in time

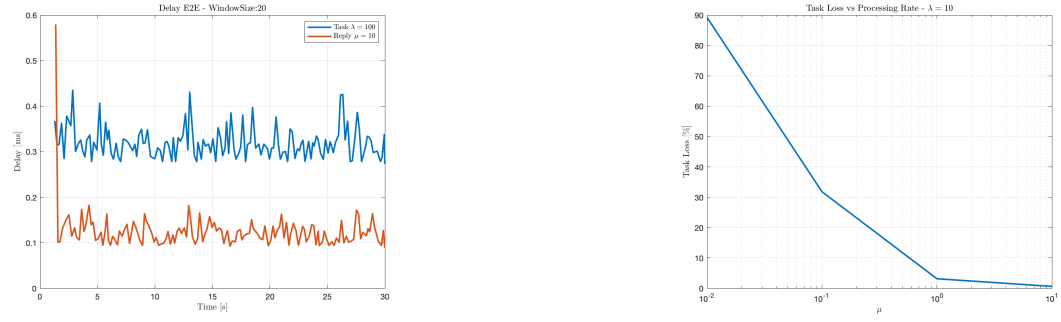

Figure 6.8: End-to-end delay at application layer and task loss experienced for fixed λ and different μ

Chapter 7

Experimental Results and Discussion

In previous chapters, both ns-3 and Python environments have been introduced and deeply analyzed, highlighting purpose, strategy and implementation of the proposed solution. In first instance, the purpose of this chapter is to briefly discuss the main limitations of the system which have been encountered during the several experiments carried out. Specifically, some of the most important training sessions and their results are reported, underlining the common critical aspects of machine learning in general and the limitations of the specific solution proposed.

In second instance, the best policy achieved is exploited to retrieve a set of statistics that can represent and describe the performance of the system.

7.1 Training Experiments

As introduced previously, this section is devoted to the introduction and brief discussion of some of the experiments that have been conducted during the trial and error procedure to find the best hyper-parameters for the training session.

Each figure of the following reports the evolution of the cumulative reward achieved in each episode and averaged according to the following algorithm: for each episode, the running reward is updated according to Equation 7.1 and eventually stored for plot purposes every 10 episodes. This allows to smooth out the reward curve and highlight its fundamental characteristics: the increasing trend and the maximum value achieved.

It is worth mentioning that each training experiment could last up to 10 hours each. Exploiting more powerful computational resources, especially introducing a GPU, leads to shorter training times (about 5 hours). Future work may concentrate on improving this timing issue and hence enable the possibility to perform multiple tests with higher efficiency and agility.

$$runningReward \leftarrow 0.9 \cdot runningReward + 0.1 \cdot episodeReward \quad (7.1)$$

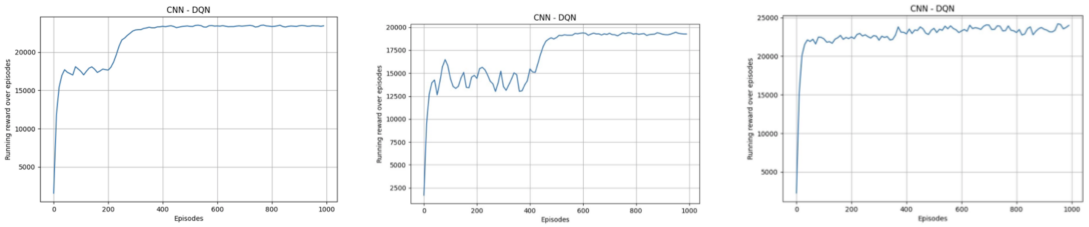

Figure 7.1: Cumulative Reward Problem, Complex Reward Unsuitability, Exploding Gradients Effect

- **Figure 7.1**:
  - Cumulative Reward Problem: a reward function aiming at the minimization of the maximum UAV↔GD distance leads to improper results; due to the algorithm's critical aspects, the reward over the whole simulation may always be related to the same UAV↔GD pair, making all the actions performed by other UAVs completely unaccounted for.

  - Complex Reward Unsuitability: in the attempt to solve the previous problem, the reward function has been extended to higher complexity. The achieved maximum reward is even lower than the previous case; a possible conclusion is that the algorithm is not robust enough to support complex reward.
  - Exploding Gradients Effect: if the reward function produces large values, the training process may face one of the most common learning problems, i.e. exploding gradients effect.

- **Figure 7.2**:
  - NaN Inconsistencies: a reward function that produces infinitely large values and tiny ones in the same context leads to inefficient learning.
  - Noisy Inference: in order to address the previous issue, clipping could be implemented. However, the inference phase appears to be particularly noisy and characterized by a decreasing trend.
  - Complex Reward Unsuitability: going back to complex reward functions, this time ensuring that exploding gradients effect does not occur, still brings no benefit. This corroborates the idea that the algorithm is not robust enough to support them.

- **Figure 7.3**:
  - Discretization Mismatch: a binary reward is deployed: if a UAV is within radio range of a GD, 1 is returned. However, if the discretization parameter is not designed properly, mismatches due to ns-3 continuous space and python discrete one improper mapping may arise.
  - Noisy Inference: once the discretization mismatch has been corrected, the maximum reward achieved is consistent, although extremely noisy.

  - Competition Effect: increasing the batch size significantly improved the noisy behaviour, leading to an increasing trend during the inference phase. Good performance is achieved if dithering technique is applied during verification.

- **Figure 7.4**:
  - Improper Batch Size: following the idea of the previous experiment, the batch size is further increased. This, however, led to no learning achieved at all.
  - Vanishing Gradients: it is commonly renowned that normalizing the reward leads to faster and better training sessions. This is not the case since vanishing gradients effect occurs.
  - Reduced Reward Range: going back to the best result up to now and trying to reduce the radio coverage boundary in order to ensure no competition effect among agents. This leads to weird knowledge loss and unreliable behaviour on inference.

- **Figure 7.5**:
  - Large Reward Range: larger radio range limits appear to lead to better performance, but, actually, this is an illusion since agents will appear to be converged when they are actually pretty far from the corresponding GD.
  - Algorithm Scalability: going back to the best result up to now and trying to reduce the discretization complexity. It seems that smaller scales are better managed by the algorithm, however, this inevitably corresponds to high inaccuracy and high quantization error.
  - Reward Range Sensitivity: staying on a smaller scale and increasing distance among GDs to avoid competition brings back the agents to noisy, unreliable and inconsistent behaviour during the inference phase.

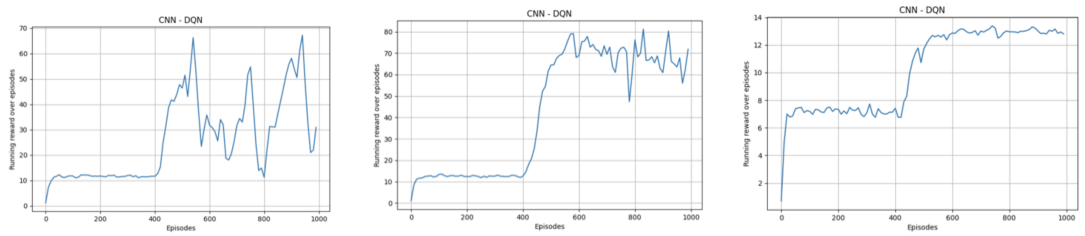

Figure 7.2: NaN Inconsistencies, Noisy Inference, Complex Reward Unsuitability

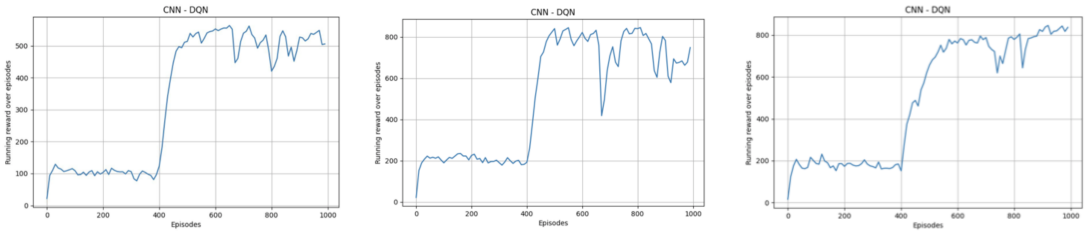

Figure 7.3: Discretization Mismatch, Noisy Inference, Competition Effect

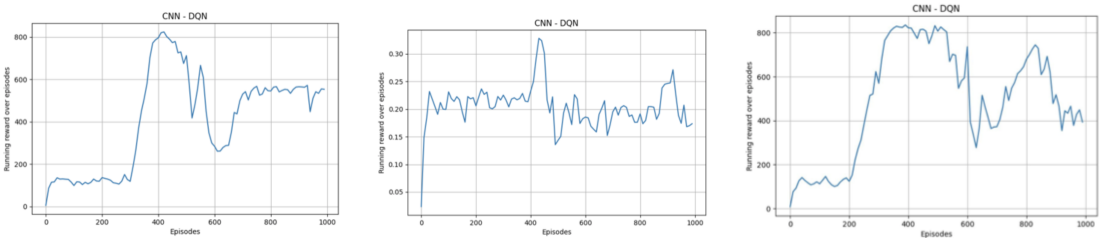

Figure 7.4: Improper Batch Size, Vanishing Gradients, Reduced Reward Range

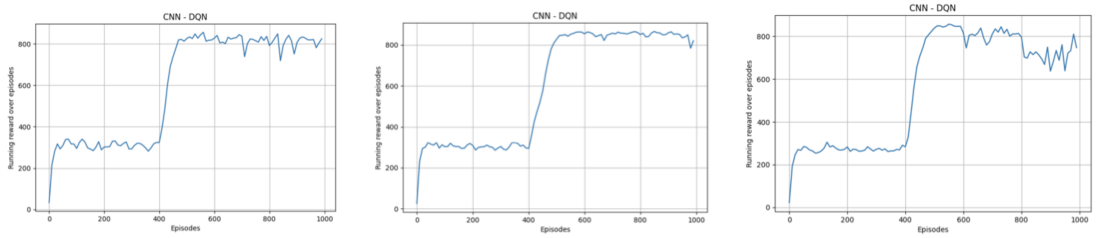

Figure 7.5: Large Reward Range, Algorithm Scalability, Reward Range Sensitivity

7.2 Performance Evaluation

This section aim at introducing and discussing the final experimental results achieved in this work. From the long set of training sessions experiments, the one that achieved the best performing policy has been selected and exploited to produce some statistics. The simulation parameters corresponding to the chosen policy and shared by all the following experiments are reported in Table 7.1. The final reward function selected is shown in Equation 7.2, where x_j is the distance between a GD and the nearest UAV, i.e. Equation 7.3, and ρ_0 is an arbitrary parameter expressing the minimum distance within which a UAV can be sufficiently considered in radio range of a certain GD. In other words, the RL algorithm will aim at the minimization of the distance between a GD and its nearest UAV.

$$r_t = \sum_j (x_{j,t} \leq \rho_0) \quad \forall j \in \{1, \ldots, \text{nGDs}\},\ t \in \{1, \ldots, \text{simTime}\}, \tag{7.2}$$

$$x_j = \min_i (d_{GD_j - UAV_i}) \quad \forall i \in \{1, \ldots, \text{nUAVs}\} \quad \forall j \in \{1, \ldots, \text{nGDs}\} \tag{7.3}$$

The first result that is worth mentioning is the reward achieved by the system episode by episode in Figure 7.6. This curve has been obtained by repeating the training procedure six times and averaging the results. A 90 % confidence interval is assumed when evaluating the standard deviation estimate. As expected, the training process starts in pure exploration mode, i.e. the agents exclusively perform random action with the purpose of acquiring knowledge about the environment they are in. Starting from episode 200, the probability of random actions starts decreasing iteration by iteration; the agents are now progressively exploiting the acquired knowledge and simultaneously improving it further. Around episode 300, a steady state

is reached; the network is not learning anymore, reaching an average reward value per episode equal to 800. This value is consistent: since three GDs and three UAVs have been considered, the maximum reward that can be obtained is given by $nGD \cdot simTime = 900$.

Parameter	**Value**
nDrones	3
nGroundDevices	3
Simulation Time	300 s
Training Episodes	1000
Starting ε	1
Minimum ε	0.1
Episode for Exploitation start	200
Epsilon Decrement Law	$\varepsilon_{t+1} = \varepsilon_t \cdot 0.99995$
Discount Factor γ	0.97
Batch Size	175
Polyak Weight τ	0.01
Experience Replay Buffer Size	15e3
Evaluation Step Time	1 s
Bounds	20
Discretization	5

Table 7.1: Simulation Parameters

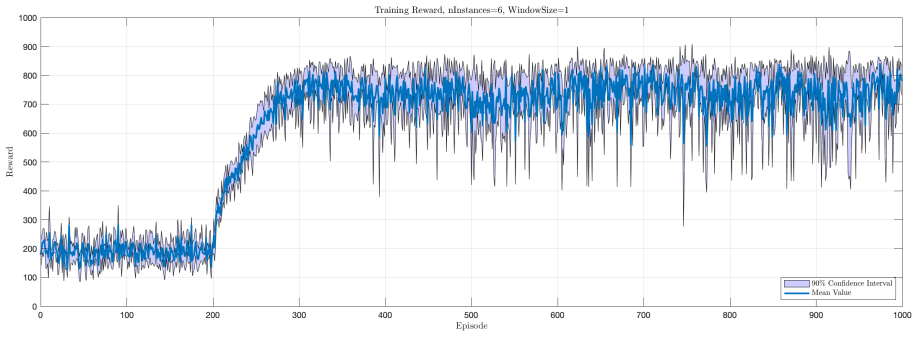

Figure 7.6: Reward achieved for each episode

On the other hand, what was not expected at all is the amount of noise

experienced. Randomly, during the course of the inference phase, i.e. for episodes > 300, some simulations end up with meagre rewards, e.g. 500, meaning that convergence of UAVs toward the GDs was not achieved. Exploiting the visualizers introduced in Figures 6.3 and 6.2, it becomes clear that despite most of the times the algorithm works, making the agents converge successfully, other times this does not happen. The agents happen to get stuck in their position, oscillating or competing with each other. A solution to this issue has been identified: introducing a certain probability of random actions in the inference phase may lead agents to states where the behaviour they have to leverage is clearer. The procedure of intentionally introducing noise to achieve better performance is commonly known as dithering, a technique usually deployed in Digital Signal Processing field.

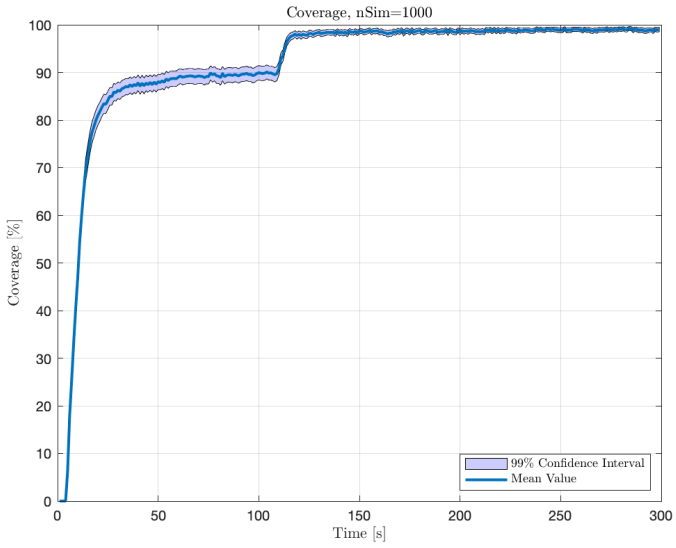

Figure 7.7: Average coverage percentage vs Time

Previous considerations are further corroborated by results in Figure 7.7. This plot depicts the average coverage percentage achieved by the system in time. It has been retrieved repeating the inference session a thousand times and collecting, for each one of them, a coverage log expressing how many GDs

were served at a certain time instant. The average among these thousand instances has been evaluated and a 99% confidence interval assumed for the standard deviation computation. Consistently with the expectations, the coverage percentage grows rapidly from 0% to 90% within the first 50 s. In most simulations, the algorithm is able to bring the agents to convergence in few seconds. For those instances in which the agents may get stuck in critical states, dithering solves the problem: a probability of 0.1 for random actions is imposed, meaning that, statistically, one action over a hundred will be random, and, luckily, it will bring the agents out of the critical state. That is probably why convergence suddenly grows up almost to 100% starting from about 115 s. Actually, a more smooth growth toward total convergence was expected; the fact that almost deterministically the algorithm converges at 115 s leads to the idea that the network might actually be learning to do so. Future work may be devoted to investigating the reasons why this occurs and how to prevent it.

The last result presented is the task drop rate experienced by all GDs in time (Figure 7.8). Also in this case a thousand simulations were performed and application layer logs collected and processed. The strategy deployed to produce these logs and the approach for their post-processing is deeply discussed in Chapter 6. Differently with respect to previous results, this time more scenarios were considered, changing the urban conditions or the transmission power.

- **Best-case scenario**: Suburban environment and relatively high transmission power, i.e. $P_{TX} = -5$dBm. Propagation conditions are simulated to be almost ideal, pretty much resembling the free space path loss from the Friis equation (Algorithms 3 and 4). Hence, low communication impairments and losses are expected.

- **Worst-case scenario**: Highrise Urban environment and relatively low transmission power, i.e. $P_{TX} = -10$dBm. Propagation conditions are simulated to be the harshest possible, the probability of LoS will be

lower with respect to the previous case and the corresponding extra path loss will much higher.

- **Intermediate-case scenario**: Suburban environment and relatively low transmission power, i.e. $P_{TX} = -10$dBm. Even though the propagation conditions simulated are the best possible, a lower transmission power has been considered. This means that there is the possibility that the experienced SNR is not sufficiently high to guarantee reliable performance.

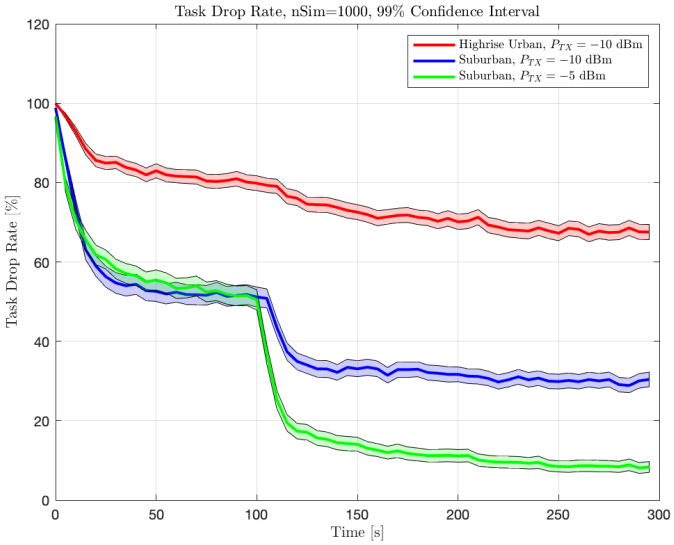

Figure 7.8: Task Drop Rate vs Time

As expected, Figure 7.8 clearly shows that, for the worst-case scenario, although convergence is achieved, the task drop rate is particularly high. Therefore, there is no doubt that, for harsh propagation conditions like this one, some telecommunication improvements must be engineered. Simulating higher transmission power is for sure possible, however, the energy consumption and interference constraints will realistically mark a limit. An example could be to deploy a reliable transport layer protocol, e.g. TCP, that could handle the retransmission of packets if any packet loss occurred. The

other two scenarios are consistent with the expectations too: the considered increase in transmission power leads to almost 20% benefit in task drop rate. Moreover, as already discussed for Figure 7.7, the lowest rates are achieved on average after 115 s, i.e. when full convergence of UAVs over the GDs is obtained. Considering the characteristics of the proposed system, its non-idealities and its critical aspects, a 10% task drop rate for the best-case scenario can be considered adequate.

Chapter 8

Vodafone Internship

8.1 Introduction

Great source of inspiration for the development of this thesis must be acknowledged to Vodafone Business Italia and, specifically, the 5G Program team. Over the last few years, Vodafone Italia has invested a considerable amount of resources in the development of use cases that could provide a significant impact on our society. The main focus is the exploitation of 5G, i.e. the state of the art standard for mobile communication systems, in order to pursue the technological revolution that has been foreseen for the next decade. The fundamental feature that makes 5G technology remarkable with respect to previous standards is the achievement of potential application fields and services that were completely unimaginable up to now.

They can be essentially classified into three main sections:

- **Enhanced Mobile Broadband**: the aim is the achievement of high throughput and high spectral efficiency. The ideal peak data rate achievable is in the order of Gbps, enabling ultra high-quality streaming services as well as augmented and virtual reality experiences.

- **Ultra Reliable and Low Latency Communication**: it is possible

to guarantee an E2E latency below 10 ms and a FER below 10^{-9}, enabling applications where transmission errors or delay excess can not be tolerated, e.g. autonomous vehicles communications or remote surgery.

- **Massive Machine Type Communication**: the capacity of the network in terms of simultaneous connections have been improved up to 1 million per squared kilometre. This enables dense Internet of Things scenarios, characterized by the cooperation of multiple sensors and devices which require good coverage, aim at exchanging short messages and are focused on energy efficiency.

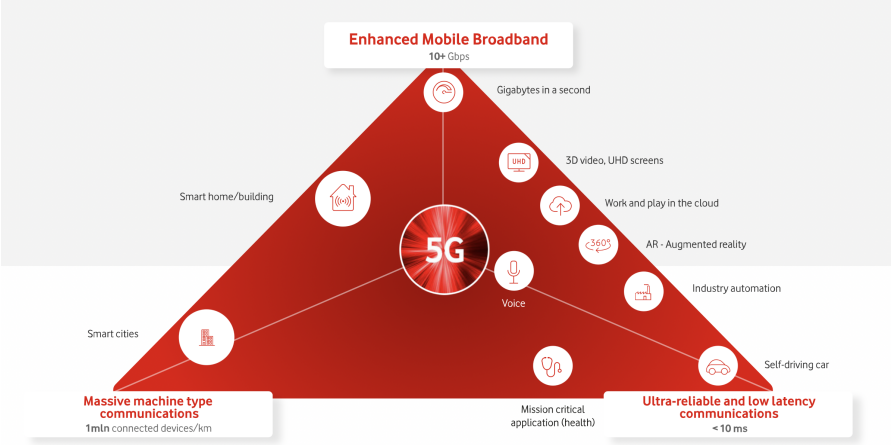

Figure 8.1: 5G Features. Sources: ITU, *vodafone5g.it*

Obviously, these features cannot be achieved at the same time; depending on the specific application needed by vertical industries and its QoS requirements, the parameters of the network, at every layer of the network stack, will be properly designed and tuned. In conclusion, 5G should not be seen as a single standard but as a set of recommendations covering different use cases. The principal characteristics of 5G are briefly introduced in the following:

- **Multiple Frequency Spectrums**: from a communication system perspective, according to the Shannon Theorem, the most effective way to

increase the amount of bandwidth available is moving to higher central frequencies, although this inevitably results in worse coverage and the need for cellular densification. Multiple center frequencies could be exploited depending on the specific requirements: 700 MHz for offering wide coverage in urban and rural areas, 27 GHz for high-performance data transmission and 3.6 GHz as a trade-off among the two.

- **Orthogonal Frequency Division Multiplexing**: OFDM, together with Cyclic Prefix technique, is the most deployed multi-carrier system technique. It allows to efficiently counteract frequency selectivity of the wireless channel, schedule resource allocation in a flexible way, guarantee no inter-channel interference (ICI) and no inter-symbol interference (ISI), and enable single-tap equalization for channel estimation. All these features come with a reasonably low computational complexity thanks to the Fast Fourier Transform Algorithm.

- **Massive MIMO**: working at higher frequencies leads to smaller antenna sizes; the coexistence of multiple input and multiple output antennas enables digital signal processing techniques such as spatial diversity, beamforming and spatial multiplexing.

- **Multi Access Edge Computing**: the network is now enhanced with localized computing capabilities that allow to process data at the edges of the network, resulting in shorter data paths and faster response time.

- **Software Defined Networking and Network Function Virtualization**: SDN enables the abstraction of network resources, making them machine-independent and ready for general-purpose hardware. NFV exploits this new physical resources perspective to define software-based network functions and combine them in order to produce services. In order words, while SDN decouples data and control plane, NFV decouples the physical network equipment from the network functions running on them. The proper orchestrations of these two technologies

lead to **Network Slicing**, i.e. the possibility to define multiple vertical industry solutions with specific requirements that are fulfilled while properly sharing the same physical resources at the same time.

8.2 Vodafone 5G Trials

Vodafone has been involved in multiple projects funded by MiSE (*Ministero dello Sviluppo Economico*) and promoted by several endorsers.

- **Milan, 5G European Capital**: the Vodafone 5G Trials, assigned by the MiSE after Vodafone won the national call for Milan, is backed by an investment of 90 millions euro and the involvement of 38 partners. As a result, Milan has become the European capital of 5G, with 41 services across 7 industries.

- **Vodafone 5G Genova**: Vodafone is bringing four 5G projects to Genova related to public transportation and infrastructure security. The use of 5G technology will allow Genova to accelerate digital transformation and evolve into a true city of the future.

- **Catanzaro 4.0**: Vodafone is part of 'Catanzaro 4.0', a trial project funded by MiSE and promoted by Comune and Provincia of Catanzaro in collaboration with the Magna Graecia University, Polo Bio Tecnomed and PMI Igea Soluzioni. The project objective is to valorize the historical, cultural and enviromental assets of the territory using 5G solutions, allowing citizens and tourists access to innovative information and recreational services.

- **BASE-5G**: Vodafone is part of BASE-5G, one of the winning projects of the "Call Hub Ricerca e Innovazione" by the Regione Lombardia and lead by Politecnico di Milano and in collaboration with Akka Italia, LIFE, YAPE and Anotherreality. The project objective is to develop the

Smart City of the future, smart areas that use Vodafone 5G to interact with citizens, offering intuitive interfaces depending on their needs.

8.3 5G Program for Vertical Solutions

The main fields of applications and services that have been being investigated in the 5G Program are:

- **Digital Divide**: the use of 5G technology as a Fixed Wireless Access solution allows people, businesses and public offices to use evolved, interactive and engaging services in Digital Divide areas where the deployment of fixed infrastructure is not feasible.

- **Education and Entertainment**: 5G enables the possibility to access video streaming content in highly dynamic mobility seamlessly and to be integrated with augmented reality devices for Tourist 4.0 and Education 4.0 solutions.

- **Manufacturing and Industry 4.0**: 5G can be exploited as the underlying communication system for many different applications ranging from Smart Agriculture, Connected Drones for Infrastructure Maintenance, Augmented Reality for assisted maintenance, Collaborative Robotics, and Last Mile Logistics.

- **Mobility and Transport**: Assisted Driving solutions aim at enlarging the driver visual range extension in case of partial or blind visibility exploiting connected vehicles real-time video exchange and signaling.

- **Health and Wellbeing**: URLLC features of 5G enable the deployment of a robotic telesurgery system that allows a surgeon to operate a patient remotely.

- **Security and Surveillance**: 5G enables the adoption of smart video-surveillance systems that can automatically detect suspicious or anomalous behaviours, increasing safety in public spaces. Other solutions range from video surveillance on board of law enforcement vehicles, automatically detecting licenses, or the deployment of drones for surveillance of sensitive or inaccessible areas due to emergency scenarios or natural disasters.

- **Smart Energy and Smart City**: the coexistence of a huge number of interconnected devices and sensors fosters the definition of Smart City solutions and cutting-edge applications to improve people's quality of life.

8.4 Internship Activities

This six-month internship activity has been mainly devoted to the analysis of the previously announced use-cases and to the active involvement in the development of existing ones regarding Manufacturing and Industry 4.0.

Several training sessions have been organized in order to analyze in deep the core strategies and technologies on which the use cases are designed from a networking perspective:

- **Real-time streaming protocols**, e.g. WebRTC, and comparing their performance with legacy standards.

- **Network Traversal Services**, e.g. ICE (Interactive Connectivity Establishment), STUN (Session Traversal Utilities for NAT), TURN (Traversal Using Relays around NAT)

- **Data Center Virtualization**, analyzing the functionalities and services provided by VMware vSphere Suite in terms of resources virtualization and orchestration.

- **IaaS to CaaS Evolution**: an introduction to Docker and Kubernetes has been provided, highlighting the advantages of container-based solutions with respect to VM-based ones.
- **Virtual Private Networks**, analyzing how to configure VPNs, investigating how to make them compliant with Cyber Security Standards and the reasons why it could be of interest for the development of a certain application.

Significant attention has been devoted to the development of these use cases:

- **Last Mile Logistics**: this use case focuses on the deployment of a self-driven ultra-light electric vehicle for urban delivery purposes. The solution is based on the low latency features of 5G, enabling the possibility for a pilot to remotely control the drone if necessary in a seamless way. Moreover, Multiple Access Edge Computing is exploited to run the self-drive algorithm and the fleet management applications, hence maximizing the energy efficiency on board of the drone. An additional feature that has been recently investigated is the possibility for the drone to interact with an IoT-enhanced environment, such as Smart Elevators.
- **UAVs for Smart Logistics**: this use case aims at providing a fast, efficient and reliable way to transport sensitive items within the campus of a renowned hospital in Milan. For instance, UAVs could be exploited to deliver samples from the hub to the proper laboratory autonomously. One of the main focuses is the development of a collision helper algorithm that enables the coexistence of multiple UAVs in the same area. Whenever an obstacle or an imminent collision with another UAV is detected, a pilot in a control room will receive a signal and take control remotely of the UAV in question. During the flight, an application running on MEC constantly monitors the drone, acquiring telemetry in real-time thanks to 5G technology.

- **UAVs for Infrastructure Maintenance**: the aim of this use case is the development of a solution for the inspection and monitoring of public infrastructures. A UAV, enhanced with multiple sensors, e.g. UHD Camera, Infrared Camera, LIDAR, etc., allows to monitor a building or an infrastructure efficiently, reducing the time usually needed to organize and perform operations like these, avoiding urban service disruption, and improving safety since human intervention can be scheduled only if necessary. The drone exploits 5G communication to continuously transmit information during the flight, which will be processed in near real-time by MEC applications. Multiple applications are expected to run at the same time: one handles the remote control of the drone, another aggregates the information and prepares a 3D model of the infrastructure, another may apply a Machine Learning model for an AI Analytics procedure.

Eventually, considerable interest has been devoted to the **Vodafone IoT Academy**, a joint project between Vodafone Italia and Politecnico di Torino with the purpose of merging high-quality academic formation with wide business perspectives and experienced technical support. The groups of students are organized in the most heterogeneous way possible; one of the expected skills to develop is the ability to establish an interdisciplinary cooperation while seeking innovative solutions. The students have been initially provided with two main topics, Smart Agriculture and Connected Drones; the purpose of the Academy is for them to formalize, design, develop and actually implement an innovative application regarding these topics. Although the objective is to foster imagination, innovation and creative thinking, the solution should also fulfill the business requirements to make it attractive and its implementation actually feasible.

Chapter 9

Conclusions

Unmanned Aerial Vehicles are widely considered to be among the most interesting and promising technologies of recent years. Their potential in terms of applications, services and quality of life enhancement has been attracting widespread interest, both from the research world and business companies. In light of recent advancements and the rapid rise in the use of UAVs, concerns have arisen about the unique technical challenges UAV operations and scenarios introduce. The design of an effective methodology for dealing with optimization problems and the evaluation of system performance through scalable, yet realistic, simulations are indeed recognized as critical topics that deserve to be properly investigated.

In this work, both objectives are pursued, i.e. to develop and implement a machine learning solution that allows to achieve the optimal UAV positioning in terms of coverage maximization and resource utilization while simulating the system in realistic settings. In facts, *ns-3* simulation offers a noteworthy simulation environment that allows to evaluate and compare several scenarios, protocols and applications efficiently. This permits to easily encompass all the different aspects that contribute to design quality and network performance. An efficient integration among ns-3 and artificial intelligence frameworks is renowned as a challenging and intriguing research

field since it is supposed to provide researchers with state-of-the-art ways to manage the increasing complexity of modern networks. The addressed scenario includes a set of UAVs, which are meant to assist a number of ground devices in processing tasks and eventually provide the ground devices with an outcome. This work presents a deep reinforcement learning model devoted to assessing a centralized policy that, basing on the observations provided by UAVs, establishes the best action each one should perform to maximize the expected cumulative reward, i.e. coverage optimization. In order to provide the algorithm with an environment as much realistic as possible, an ns-3 simulation framework has been developed, exploiting both native features and a set of new scenario agnostic modules and applications. For example, a statistical channel propagation model for low altitude platforms has been implemented, aiming at the realistic prediction of the path loss experienced by network devices basing on urban environment properties and the elevation angle.

A long trial and error session has been performed in order to identify the optimal parameters and reward function for the deep q-learning network algorithm. Although the maximum reward achieved during inference is consistent, the experiments highlighted several limitations that should be considered and addressed in future works, especially those regarding critical states and the lack of robustness. Despite that, the final performance evaluations and experimental results are definitely consistent and particularly promising: thanks to the proposed policy, all agents are able to converge successfully within the end of each simulation; hence a wide set of meaningful statistics could be gathered.

Taken together, these findings demonstrated that the integration of AI frameworks with realistic network simulations provides a powerful tool for investigating complex network scenarios. The proposed approach and methodology in this work has the potential to lead researchers to new state-of-the-art ways to design and validate innovative and cutting-edge solutions.

9.1 Future Works

In spite of the fact that there are several limitations in the proposed algorithm, given the potential demonstrated, it is reasonable to believe that this work could be a springboard for further research.

The most important limitation lies in the fact that the algorithm showed weaknesses in terms of robustness with respect to small changes in the simulated environment or in the training parameters. Future studies are therefore suggested in order to investigate more sophisticated neural network architectures and carry out multiple experiments and debugging sessions in order to identify which part of the framework chain could be responsible for the limitations discussed.

In addition, it is worth remarking that the centralized architecture proposed in this work should be compared with a distributed one. As discussed in previous chapters, in light of the experiments conducted, it seems fair to believe that a distributed solution could lead to better state transitions and reward association pairs and therefore to performance enhancement.

Eventually, another suggestion for future work is to concentrate on improving the scalability of the system: a significant contribution would be to determine a policy as much scenario-agnostic as possible, in order to avoid repeating the time-consuming training sessions for each possible configuration and therefore achieve the first step towards an actual baseline solution. Improving time efficiency is for sure a way to foster a large number of experiments and to deploy an agile methodology that could lead to efficient design solutions.

Appendix A

Reinforcement Learning Algorithms

A.1 Q-learning and DQN

Algorithm 1 Q-learning: Learning function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Require:
States Set $\mathcal{S}$
Actions Set $\mathcal{A}$
Reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
State transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
Learning rate $\alpha \in (0, 1]$
Discounting factor $\gamma \in [0, 1]$
procedure Q-Learning($\mathcal{S}$, $\mathcal{A}$, R, T, α, γ)
Initialize $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
while Q is not converged **do**
Start in state $s_t \in \mathcal{S}$
while s_t is not terminal **do**
▷ Select the best action according to a defined policy
e.g., $a_t \leftarrow \arg\max_{a \in \mathcal{A}} Q(s, a)$
▷ Compute the immediate reward
$r_t \leftarrow R(s_t, a_t)$
▷ Define the new state
$s_{t+1} \leftarrow T(s_t, a_t)$
▷ Update Q function
$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a))$
▷ Update state
$s_t \leftarrow s_{t+1}$
end while
end while
return Q
end procedure

Algorithm 2 Deep Q Network with Experience Replay and ε-greedy Exploration Strategy

Initialize:
Replay Memory D to capacity N
Main Neural Network Q_θ arbitrarily
Target Neural Network $Q_{\theta'}$ with $\theta' = \theta$
for *episode* $\in [1, M]$ **do**
  Initialize the state s_0
  Define a Bernoulli Random Variable ξ for which $P(\xi = 1) = \varepsilon$
  for $t \in [1, T]$ **do**
    Extract instance of ξ
    if $\xi == 1$ **then**
      Select action a_t randomly
    else
      $a_t \leftarrow \arg\max_a Q(s_t, a, \theta)$
    end if
    Get next state s_{t+1} and reward r_t
    Store the transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample a random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set the temporal difference $y_j = r_j + \gamma \arg\max_a Q(s_{j+1}, a; \theta')$
    Update θ through gradient descent step on $(y_j - Q(s_j, a_j; \theta))$
    Update θ' through Polyak averaging: $\theta' = \tau \cdot \theta' + (1 - \tau) \cdot \theta, \tau \in [0,1]$
  end for
end for

Appendix B

Channel Model Algorithms

B.1 Friis Propagation Loss Model

Algorithm 3 Free Space Received Power Computation

Initialize:
Carrier Frequency f [Hz]
Distance between TX and RX d [m]
Transmitting Power P_{TX} [dBm]
Free Space Path Loss $FSPL$ [dB]
Define:
Speed of Light in vacuum $c \approx 3e8$ [m/s]
Carrier Wavelength $\lambda = c/f$ [m]
Assume:
Antenna TX and RX Gain $G_{TX} = G_{RX} = 1$
Compute:
Retrieve Mobility Models for TX and RX, respectively a and b
Update distance: $d = a \rightarrow$GetDistanceFrom (b) [m]
$FSPL = 20\log_{10}(d) + 20\log_{10}(\frac{f}{1e6}) + 20\log_{10}(\frac{4\pi \cdot 1e6}{c})$ [dB]
Return:
$P_{RX} = P_{TX} - FSPL$ [dBm]

B.2 Probabilistic Loss Model for LAP

Algorithm 4 Probabilistic Channel Model Received Power Computation

Initialize:
Carrier Frequency f [Hz]
Distance between TX and RX d [m]
Elevation Angle θ [deg]
Minimum Elevation Angle θ_0 [deg]
Urban Density (Sub Urban, Urban, Dense Urban, Highrise Urban) ρ
Line of Sight Condition (LOS, nLOS, Random) ψ_0
Transmitting Power P_{TX} [dBm]
Free Space Path Loss $FSPL$ [dB]
Extra Path Loss ξ [dB]
Define:
Speed of Light in vacuum $c \approx 3e8$ [m/s]
Carrier Wavelength $\lambda = c/f$ [m]
Parameter μ_ξ: mean value for extra path loss
Parameters α, β: for extra path loss standard deviation
Parameters γ, δ: for probability of LOS computation
Assume:
Antenna TX and RX Gain $G_{TX} = G_{RX} = 1$
Compute:
if ψ_0 == Random **then**
    Retrive γ, δ according to Tables B.1 or B.2 basing on f and ρ
    Compute Probability of LOS: $P = \gamma \cdot (\theta - \theta_0)^\delta$
    Update ψ according to probability P
end if
Retrive μ, α, β according to Tables B.1 or B.2 basing on ψ, f and ρ
Compute Standard Deviation for Extra Path Loss ξ: $\sigma_\xi = \alpha \cdot \exp(-\beta \cdot \theta)$
Update Free Space Path Loss $FSPL$ according to Algorithm 3
Extract ξ from $\mathcal{N}(\mu_\xi, \sigma_\xi^2)$
Return:
$P_{RX} = P_{TX} - FSPL - \xi$ [dBm]

700 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	0.0	0.6	1.0	1.5
(α, β)	(11.53,0.06)	(10.98,0.05)	(9.64,0.04)	(9.16,0.03)
(γ, δ)	(0.77,0.05)	(0.63,0.09)	(0.37,0.21)	(0.06,0.58)

2000 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	0.1	1.0	1.6	2.3
(α, β)	(11.25,0.06)	(10.39,0.05)	(8.96,0.04)	(7.37,0.03)
(γ, δ)	(0.76,0.06)	(0.6,0.11)	(0.36,0.21)	(0.05,0.61)

5800 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	0.2	1.2	1.8	2.5
(α, β)	(11.04,0.06)	(10.67,0.05)	(9.21,0.04)	(7.15,0.03)
(γ, δ)	(0.75,0.06)	(0.56,0.13)	(0.33,0.23)	(0.05,0.64)

Table B.1: LOS Parameters from [6] obtained with curve fitting method

700 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	18	17	20	29
(α, β)	(26.53,0.03)	(23.31,0.03)	(30.80,0.04)	(32.13,0.03)
(γ, δ)	(0.77,0.05)	(0.63,0.09)	(0.37,0.21)	(0.06,0.58)

2000 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	21	20	23	34
(α, β)	(32.17,0.03)	(29.60,0.03)	(35.97,0.04)	(37.08,0.03)
(γ, δ)	(0.76,0.06)	(0.60,0.11)	(0.36,0.21)	(0.05,0.61)

5800 MHz				
	Suburban	**Urban**	**Dense Urban**	**Highrise Urban**
μ	24	23	26	41
(α, β)	(39.56,0.04)	(35.85,0.04)	(40.86,0.04)	(40.96,0.03)
(γ, δ)	(0.75,0.06)	(0.56,0.13)	(0.33,0.23)	(0.05,0.64)

Table B.2: nLOS Parameters from [6] obtained with curve fitting method

Appendix C

Handlers Algorithms

C.1 Boundaries Handler

Algorithm 5 Rescale actions or simulate reflection effect

deltaStep, *currentPos* and *newPos* are 3D Vectors
ρ is the radius of the circle that defines the boundary of the system
deltaStep has been derived from the action message
newPos has been computed as *currentPos* + *deltaStep*
while *excessBoundaries* **do**
Compute distance d among newPos and the origin
  if $d \leq \rho$ **then**
    $excessBoundaries$ = false
    Add the new WayPoint w to the UAV Mobility Model
  else
    if $reflectionEn$ == false **then**
      Rescale *deltaStep* of one unit and update *newPos*
    else
      Compute the coordinates of the nearest point
      belonging to the boundary circle
      if $deltaStep.x$!= 0 **then**
        Action performed along the x-axis
        $|boundary.x| = \sqrt{\rho^2 - currPos.y^2}$
        Compute *newPos* and Add a new Waypoint w
        Compute the remainder distance to be covered
        $|\delta d|$ = $|deltaStep.x|$ - $|boundary.x - currPos.x|$
        Compute the angle for the direction
        $\angle \delta d = \tan^{-1} |currPos.y/boundary.x|$
      end if
      Mutatis mutandis for actions along y-axis
      if $currPos.y > 0$ and $currPos.x > 0$ **then**
        $newPos.y = newPos.y - |\delta d| \cdot \sin \angle \delta d$
      end if
      Mutatis mutandis for the other quadrants
      Add the new WayPoint w to the UAV Mobility Model
      $excessBoundaries$ = false
    end if
  end if
end while

Bibliography

[1] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, Young-Han Nam, and Mérouane Debbah. «A tutorial on UAVs for wireless networks: Applications, challenges, and open problems». In: *IEEE communications surveys & tutorials* 21.3 (2019), pp. 2334–2360 (cit. on pp. 2, 15).

[2] Paulo V Klaine, João PB Nadas, Richard D Souza, and Muhammad A Imran. «Distributed drone base station positioning for emergency cellular networks using reinforcement learning». In: *Cognitive computation* 10.5 (2018), pp. 790–804 (cit. on p. 2).

[3] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. «Multi-agent reinforcement learning: An overview». In: *Innovations in multi-agent systems and applications-1* (2010), pp. 183–221 (cit. on p. 2).

[4] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. «Applications of deep reinforcement learning in communications and networking: A survey». In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3133–3174 (cit. on p. 2).

[5] Petros S Bithas, Emmanouel T Michailidis, Nikolaos Nomikos, Demosthenes Vouyioukas, and Athanasios G Kanatas. «A survey on machine-learning techniques for UAV-based communications». In: *Sensors* 19.23 (2019), p. 5170 (cit. on p. 7).

[6] Akram Al-Hourani, Sithamparanathan Kandeepan, and Abbas Jamalipour. «Modeling air-to-ground path loss for low altitude platforms in urban environments.» In: *2014 IEEE Global Communications Conference*. IEEE. 2014 (cit. on pp. 9–12, 46, 54, 92).

[7] ITU-R. «Rec. P.1410-2 Propagation Data and Prediction Methods for The Design of Terrestrial Broadband Millimetric Radio Access Systems». In: (2003) (cit. on p. 10).

[8] ITU-R. «Rec. P.526-8, Propagation by Diffraction». In: (2003) (cit. on p. 10).

[9] Sergio Benedetto and Ezio Biglieri. *Principles of digital transmission: with wireless applications*. Springer Science & Business Media, 1999 (cit. on p. 11).

[10] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. «An introduction to deep reinforcement learning». In: *arXiv preprint arXiv:1811.12560* (2018) (cit. on pp. 12, 27, 28).

[11] Piotr Gawłowicz and Anatolij Zubow. «Ns-3 meets OpenAI gym: The playground for machine learning in networking research». In: *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 2019, pp. 113–120 (cit. on pp. 12, 13, 49).

[12] Hao Yin, Pengyu Liu, Lytianyang Zhang, Liu Cao, Yayu Gao, and Xiaojun Hei. «NS3-AI: Enable Applying Artificial Intelligence to Network Simulation in ns-3.» In: (2020) (cit. on pp. 12–14).

[13] *ZeroMQ*. ZeroMQ Contributors. URL: `https://zeromq.org/socket-api/`. (cit. on p. 13).

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. «OpenAI Gym». In: *arXiv preprint arXiv:1606.01540* (2016) (cit. on p. 13).

[15] Vicente Mayor, Rafael Estepa, Antonio Estepa, and German Madinabeitia. «Deploying a reliable UAV-aided communication service in disaster areas». In: *Wireless Communications and Mobile Computing* 2019 (2019) (cit. on p. 18).

[16] George Dimitrakopoulos and Panagiotis Demestichas. «Intelligent transportation systems». In: *IEEE Vehicular Technology Magazine* 5.1 (2010), pp. 77–84 (cit. on p. 18).

[17] Sandeep Raj, VK Panchal, Prem Chand Vashist, and Rajiv Chopra. «FANETs: Current Trends and Challenges». In: *2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC)*. IEEE. 2019, pp. 472–475 (cit. on p. 18).

[18] Manlio Bacco, Pietro Cassará, Marco Colucci, Alberto Gotta, Mario Marchese, and Fabio Patrone. «A survey on network architectures and applications for nanosat and UAV swarms». In: *International Conference on Wireless and Satellite Systems*. Springer. 2017, pp. 75–85 (cit. on p. 19).

[19] Mario Marchese, Aya Moheddine, and Fabio Patrone. «IoT and UAV integration in 5G hybrid terrestrial-satellite networks». In: *Sensors* 19.17 (2019), p. 3704 (cit. on p. 20).

[20] Iain Sheridan. «Drones and global navigation satellite systems: current evidence from polar scientists». In: *Royal Society open science* 7.3 (2020), p. 191494 (cit. on p. 20).

[21] Azade Fotouhi, Haoran Qiang, Ming Ding, Mahbub Hassan, Lorenzo Galati Giordano, Adrian Garcia-Rodriguez, and Jinhong Yuan. «Survey on UAV cellular communications: Practical aspects, standardization advancements, regulation, and security challenges». In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3417–3442 (cit. on p. 20).

[22] Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and tools for network simulation*. Springer Science & Business Media, 2010 (cit. on p. 21).

[23] Federico Venturini, Federico Mason, Francesco Pase, Federico Chiariotti, Alberto Testolin, Andrea Zanella, and Michele Zorzi. «Distributed reinforcement learning for flexible UAV swarm control with transfer learning capabilities». In: *Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. 2020, pp. 1–6 (cit. on p. 34).

[24] Mathieu Lacage and Thomas R Henderson. «Yet another network simulator». In: *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. 2006, 12–es (cit. on p. 45).

[25] *Manual for ns-3 contribution*. The ns-3 developers, 2019. URL: `https://www.nsnam.org/docs/manual/html/new-modules.html` (cit. on p. 45).

[26] Adam Paszke et al. «Pytorch: An imperative style, high-performance deep learning library». In: *arXiv preprint arXiv:1912.01703* (2019) (cit. on p. 47).

[27] *PyTorch*. PyTorch Contributors, 2019. URL: `https://pytorch.org/docs/stable/index.html` (cit. on p. 47).

[28] Piotr Gawlowicz and Anatolij Zubow. «ns3-gym: extending OpenAI gym for networking research». In: *CoRR* (2018) (cit. on p. 48).

[29] Chi Harold Liu, Xiaoxin Ma, Xudong Gao, and Jian Tang. «Distributed energy-efficient multi-UAV navigation for long-term communication coverage by deep reinforcement learning». In: *IEEE Transactions on Mobile Computing* 19.6 (2019), pp. 1274–1285 (cit. on p. 54).

[30] Marco Malinverno, Francesco Raviglione, Claudio Casetti, Carla-Fabiana Chiasserini, Josep Mangues-Bafalluy, and Manuel Requena-Esteso. «A Multi-Stack Simulation Framework for Vehicular Applications Testing». In: *Proceedings of the 10th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*. DIVANet '20. Alicante, Spain: Association for Computing Machinery, 2020, pp. 17–24. ISBN: 9781450381215. DOI: 10.1145/3416014.3424603. URL: https://doi.org/10.1145/3416014.3424603 (cit. on p. 60).

Acknowledgements

È mio desiderio approfittare di questo breve spazio per esprimere la mia più sincera gratitudine nei confronti dei docenti e delle persone che sono rimaste al mio fianco per l'intera durata di questo gratificante percorso.

Mi ritengo onorato di aver potuto perseguire un percorso scolastisco così ricco. La passione per la conoscenza e la cultura è solo un seme, un principio primo che richiede costante nutrimento e dedizione. Questo plus ultra va riconosciuto ai docenti, il cui ruolo è spesso sopravvalutato ma di cui ho sempre riconosciuto professionalità e competenze insuperabili. Rivolgo, a nome di tutti, un pensiero e profonda stima alla mia relatrice Carla Chiasserini per avermi concesso l'opportunità di procedere sotto la sua ala.

Eterna e metafisica gratitudine desidero che sia rivolta all'arte in tutte le sue forme. Come l'epigrafe del Teatro Massimo ricorda, essa *"Rinnova i popoli e ne rivela la vita"*.

La filosofia, mia inestimabile compagna, insegna che la finitezza del linguaggio demarchi un limite invalicabile. L'idea che io possa esprimere in parole la totalità di sentimenti ed emozioni rivolte ai miei cari appare a me tracotante. Farò dunque del mio meglio, demandando a successive forme di comunicazione non verbale il completamento dei miei pensieri.

In primis, è mia intenzione rivolgere estremo affetto alla mia famiglia. Ringrazio i miei genitori per i loro insegnamenti e per il supporto incondizionato dedicatomi in ogni contesto. Non posso non ribadire i miei elogi per la loro equilibrata razionalità e il loro contagioso coraggio, sperando un giorno

di ereditarne anche solo una piccola parte. Ai miei nonni dedico questo breve lungo viaggio; ringrazio chi, ahimè, ha concluso il suo iter terreno, rimembrandone l'amore puro e senza misura e ringrazio chi l'iter lo vive tutt'ora, ammirando e traendo ispirazione dalla loro saggezza, dall'amore per la natura e per il mondo classico, ricordandomi sempre di *seguir virtute e canoscenza*.

Ai miei amici, infine, attribuisco il valore di casa, luogo in cui manifestare la propria identità con inestimabile libertà. Tante piccole mie famiglie sono sparse per l'intera penisola, ognuna con le sue affascinanti peculiarità e sempre unite ed intramontabili. A Giulia, Giovanni, Silvia e Annalia, grazie per essere stati come fratelli, siete esseri speciali e avrò cura di voi.

Infine, a Gabriele la dedica più onerosa, solo *il* poeta può intellegere a pieno il nostro rapporto, dunque, citandolo, *l'amore ha l'amore come solo argomento*. Con te capii di aver trovato la persona che poteva condividere le mie vette senza inorridire dei miei abissi.

A tutti quanti, grazie per aver dato colore al mio piccolo universo.

Noi siamo infinito.