POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Deep Reinforcement Learning for UAV Patrolling in Rescue Operations

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Rafael Bautista Linde**

Student ID: 10784248
Advisor: Prof. Maurizio Bruglieri
Academic Year: 2023-24

Abstract

This thesis arises from the concern about the feasibility and possible optimization of implementing AI and Machine Learning techniques in time-sensitive rescue operations. Aligned with the burgeoning global trend, the UAV sector is pioneering less human-dependent and more effective methodologies for efficient drone deployment in disaster-stricken areas, enhancing rescue operations and emergency communication.

The analysis begins with characterizing the area of interest through a grid map, specifically tailored for UAV planning and exploration methodologies. The selected map outlines the primary monitoring area during volcanic eruptions and it is employed by public safety agencies in the concrete context of Mount Etna.

A detailed analysis of Reinforcement Learning, with a focus on its application to drone patrolling problems, justifies the preference for Deep Reinforcement Learning. Deep Q-Learning is chosen among its variants. The problem formulation involves discretizing the relevance map for UAV traversal and defining a reward policy based on drone actions. The neural network is trained using specific evaluation metrics to achieve homogeneous area coverage with temporal redundancy, prohibition enforcement to stay within the map boundaries, and environmental adaptability.

The search for results progresses from the application of Deep Q-Learning on a homogeneous relevance map to its application on a heterogeneous map, culminating in the acknowledgment of Double Deep Q-Learning as a more suitable tool for UAV rescue problems. Additionally, tuning the hyperparameters of the neural network has been explored to achieve close to optimal results, offering insights for future real-world applications.

Keywords: Reinforcement Learning, UAV, patrolling, Deep Q-Learning

Abstract in lingua italiana

Questa tesi vuole esplorare la possibilità di applicare techniche di Intelligenza Artificiale (AI) e Machine Learning in operazioni di soccorso sensibili al tempo. In linea con la crescente tendenza globale, il settore degli UAV sta sviluppando metodologie meno dipendenti dall'uomo e più efficaci per il dispiegamento efficiente di droni in aree colpite da disastri, migliorando le operazioni di soccorso e la comunicazione d'emergenza.

L'analisi inizia con la caratterizzazione dell'area di interesse attraverso una mappa a griglia, appositamente progettata per metodologie di pianificazione ed esplorazione degli UAV. La mappa selezionata delinea l'area primaria di monitoraggio durante le eruzioni vulcaniche ed è utilizzata dalle agenzie di sicurezza pubblica nel contesto concreto del Monte Etna.

Un'analisi dettagliata del Reinforcement Learning, con un focus sulla sua applicazione ai problemi di pattugliamento dei droni, giustifica la preferenza per il Deep Reinforcement Learning. Tra le sue varianti, viene scelto il Deep Q-Learning. La formulazione del problema prevede la discretizzazione della mappa di rilevanza per il transito degli UAV e la definizione di una politica di ricompensa basata sulle azioni dei droni. La rete neurale viene addestrata utilizzando specifiche metriche di valutazione per ottenere una copertura omogenea dell'area con ridondanza temporale, l'applicazione del divieto di uscire dai confini della mappa e l'adattabilità ambientale.

La ricerca dei risultati procede dall'applicazione del Deep Q-Learning su una mappa di rilevanza omogenea a quella su una mappa eterogenea, culminando nel riconoscimento del Double Deep Q-Learning come uno strumento più adatto per i problemi di soccorso degli UAV. Inoltre, sono stati tarati gli iperparametri della rete neurale per raggiungere risultati prossimi all'ottimo, offrendo spunti per future applicazioni nel mondo reale.

Parole chiave: Reinforcement Learning, UAV, pattugliamento, Deep Q-Learning

Contents

List of Figures

List of Tables

Introduction

At its core, this thesis endeavors to address critical concerns surrounding the integration of AI and Machine Learning techniques into emergency rescue operations, particularly in conjunction with drone deployment. By leveraging these advanced technologies, the aim is to enhance the efficiency and effectiveness of rescue missions while optimizing the allocation of resources available to emergency response teams.

Central to the thesis lies a meticulous examination of the operational landscape to facilitate the seamless integration of drone exploration and planning methodologies. This comprehensive approach underscores the critical importance of aligning technological advancements with the exigencies of emergency response operations, particularly in the aftermath of disasters.

The chosen mapping area, meticulously selected for its relevance in post-disaster scenarios, assumes a pivotal role in surveillance and response coordination efforts spearheaded by public security agencies. As the frontline interface between advanced technological solutions and real-world exigencies, this mapping area represents more than just geographic coordinates; it symbolizes the nexus between cutting-edge innovations and urgent imperatives of saving lives and mitigating the impact of calamities.

In delving into the Reinforcement Learning (RL) field, this thesis wants to improve the drone patrolling methodologies. With a focus on optimizing emergency response strategies, the thesis detects the most effective methodology among several possibilities within the RL domain.

Within the purview of this thesis, the methodology employed for addressing the patrolling problem with a single drone is meticulously crafted to optimize cover-

age effectiveness and temporal efficiency. It involves a structured approach that integrates Reinforcement Learning (RL) algorithms with the specific task of drone patrolling. The methodology unfolds in distinct stages, beginning with the formulation of the problem and the selection of suitable input data representations for the RL algorithms.

Following this, the selected RL algorithms are trained iteratively to associate different states of the environment with optimal actions, with the objective of maximizing coverage while minimizing the time required for patrolling. This training phase is crucial, as it lays the groundwork for the subsequent deployment of the drone in real-world scenarios.

Upon completion of the training phase, the drone is equipped with the learned policy derived from the RL algorithms, enabling it to autonomously navigate the operational environment and execute patrolling missions. As the drone traverses the designated area, it continuously assesses its surroundings and adapts its actions in real-time based on the learned policy.

The introduction aims to elucidate the significance of the undertaken research, articulating the problem statement, the applied methodology, and the originality inherent in this work, contributing to the advancement of both theory and practice in emergency rescue operations.

In order to enhance the comprehensibility of the document, both in terms of content and structure, an outline for the thesis is provided below, accompanied by a brief overview of each chapter.

- **Chapter 1** provides an overview of the research topic, its significance and the research objectives. It outlines the scope and limitations of the study and it introduces the structure of the thesis.
- **Chapter 2** reviews relevant literature and theoretical frameworks, it analyzes previous studies and research findings related to the topic and it identifies gaps, controversies, and areas for further investigation.
- **Chapter 3** deepens into the theoretical framework underlying Deep Rein-

forcement Learning algorithms, starting from a more general approach and progressing towards the method that will ultimately be applied.

- **Chapter 4** provides all the ingredients and the setting upon which the relevant simulations will be developed.
- **Chapter 5** presents the findings of the study based on the collected data and results. It includes tables, figures, and other visuals to illustrate these results.
- **Chapter 6** summarizes the main findings and conclusions drawn from the study. Besides, it offers recommendations for practitioners and policymakers that may conduct future studies.

1 | Problem definition

Wide-scale natural calamities, such as floods, tsunamis, and earthquakes, manifest in diverse locations and bring about devastating repercussions. In the wake of sudden catastrophes, individuals often find themselves in urgent need of external assistance. However, conventional terrestrial communication networks, just like those vulnerable to hurricanes, volcanic eruptions, or earthquakes, may experience partial or complete disruptions. Moreover, the presence of aftershocks poses a potential threat, rendering the reliability of existing wireless communication networks questionable. It becomes imperative to expeditiously establish emergency communication systems.

In the aftermath of disasters, terrestrial transportation routes are frequently obstructed, impeding the arrival of emergency communication vehicles to provide essential services. Consequently, maintaining dependable communication services via ground-based methods for those in post-disaster areas proves to be challenging. Given the intricate conditions that prevail post-disaster, it is essential to swiftly implement an agile solution. In essence, a dependable and adaptable wireless communication infrastructure is required to serve as an emergency network under any complex circumstances following a disaster, catering to both rescue operations and emergency communications. The deployment of Unmanned Aerial Vehicles (UAVs) as aerial base stations offers a potential solution to address this challenge [1].

The role of rescuers and trained dog teams happens to be crucial during a misplacement case. The method to be followed when the individual is stuck in a steep terrain or dense-vegetation zone becomes arduous and lacks accuracy. That is why the initial deployment of such UAVs could be understood as a primary step in the search operations while providing further information about the situation status to the rescue team. In summary, the task of search and monitoring can become

quite challenging, even more so when the extent of the territory to cover is much larger. The human-led search involves time and resources that ultimately become unmanageable, making it a task easily automatable with the deployment of an autonomous vehicle fleet, or alternatively, a single UAV as will be considered in the context of this thesis.

UAVs or drones, as a consequence of its rapid technological development, have been largely used in missions involving reconnaissance or navigating through unknown environments, as far as they can host a wide variety of sensors with low operational costs and high flexibility in the meanwhile. Greater bandwidth, lower latency and higher density of connected mobile networks, work in favor of UAVs deployment, which could even act as aerial base stations to provide wireless communication to terrestrial user equipments (UEs). Somehow, the main objective of UAVs deployment in a rescue operation is to cover more UEs, or equivalently, the larger area possible, in less time while reducing the mean time between visits of a parcel of the territory and considering the cooperation with the still-operative terrestrial base stations (TBSs).

The primary limitation in planning paths for UAVs is the endurance of their batteries. Despite recent advancements in battery technology, the maximum flying distance for small UAVs remains a significant limitation. From a computational perspective, the problem becomes considerably more complex when the limited battery life is taken into account. In fact, some planning issues can be efficiently resolved when battery endurance is considered unlimited but may become NP-hard when power constraints are introduced. An illustrative example of this can be found in [2]. Consequently, the imposition of power constraints gives rise to intricate optimization problems when drones or a team of drones are assigned to navigate within a specific scenario.

Regarding the specific design of the UAV to be considered in this thesis, various models available in the market could meet the requirements of a rescue operation. Among them, the *WingtraOne* model has been chosen as it stands out for its longer endurance and operational range compared to the other similar drones existent in the market right now. Below, the blueprints of the selected UAV as well as a listing

of its features may be found.

Specifications	
Wingspan	1.25 m
Max. Takeoff Weight	4.5 kg
Endurance	∼1 h
Max. Coverage	120 m
Horizontal Accuracy	1 cm
Max. Wind Gusts	65 km/h
Max. Flight Altitude	4,000 m ASL

Table 1.1: *WingtraOne* specifications [3].

Figure 1.1: *WingtraOne* picture.

As a result of all of this, there will be a vehicle that can carry out remote or autonomously computed missions, defining geographic coordinates or waypoints that the vehicle will adhere to through low-level motor control. A trajectory generator (or interpolator) upstream in the control scheme will convert the waypoint commands into position and velocity data at each instant in time. These waypoints are definite, arranged locations on the at-hand map.

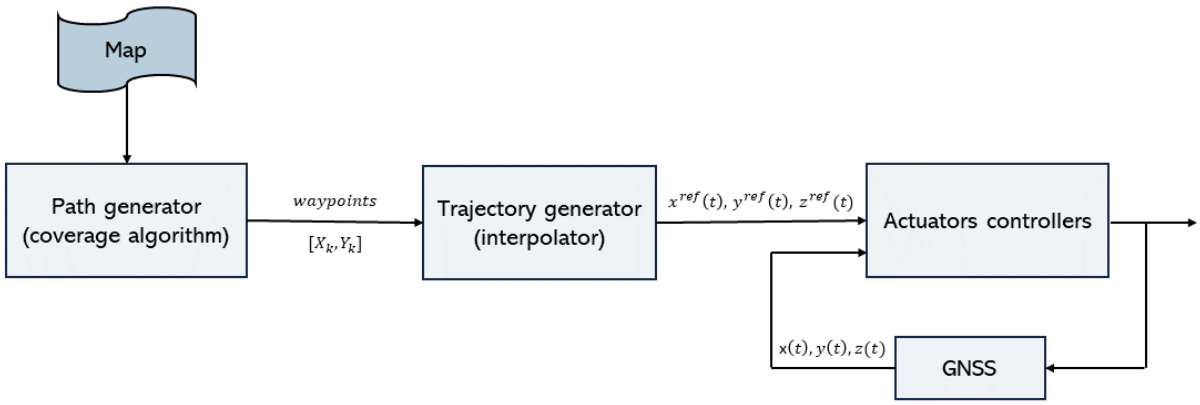


Figure 1.2: UAV control scheme.

In the previous scheme, GNSS stands for Global Navigation Satellite System, which is a constellation of satellites that provides autonomous geo-spatial positioning with global coverage. The most commonly known GNSS is the Global Positioning System (GPS) maintained by the United States, but there are other systems like

GLONASS (Russia), Galileo (European Union), and BeiDou (China).

In the UAV control scheme outlined, GNSS plays a crucial role in providing accurate positioning information to the vehicle. Once the waypoints or geographic coordinates are defined for the mission, the GNSS system enables the UAV to determine its current location in real-time by receiving signals from multiple satellites. This information is then utilized by the trajectory generator to compute the desired path or trajectory that the UAV should follow.

In essence, GNSS acts as the backbone of the navigation system for the UAV, ensuring that it stays on course and adheres to the predefined waypoints or coordinates throughout the mission. By continuously updating the UAV's position, GNSS enables precise control and navigation, facilitating both remote and autonomous operations.

1.1. Thesis rationale

In recent decades, thanks to the evolution of computing capabilities, the scientific and technical community has witnessed the emergence of artificial intelligence. Artificial intelligence, defined as the capacity of machines to autonomously and rationally respond to stimuli, represents a burgeoning and promising field of study. Instances of artificial intelligence and Machine Learning (ML) in well-known cases, such as *Google AlphaZero*, demonstrate that these algorithms can outperform human capabilities.

When confronted with intricate engineering challenges that defy the discovery of an optimal solution, Machine Learning surfaces as a robust approach. It excels at handling statistical uncertainties, addressing incomplete models of reality, and mitigating biases in input data. Machine learning, inclusive of its derivative techniques like Reinforcement Learning, fundamentally operates as a knowledge-inductive learning methodology. These experiential learning mechanisms, whether supervised or unsupervised, offer a unique advantage by providing exceptional flexibility in tackling complex issues without relying on human intuition or expert knowledge of input data.

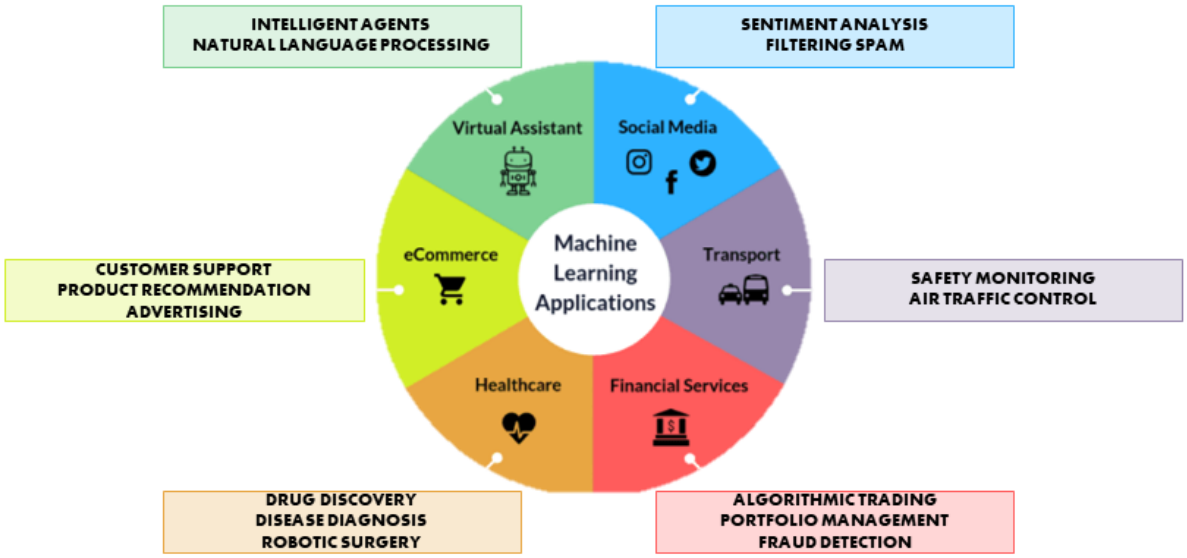


Figure 1.3: Fundamental applications of Machine Learning techniques [4].

Deep Reinforcement Learning, a subset of Reinforcement Learning, presents itself as a promising remedy for addressing these ambiguous conditions. Within this framework, the problem is characterized by a set of rules governing the training process through the common thread of reward mechanisms embedded in all RL algorithms. An evident advantage of employing these algorithms is their capacity to yield notably satisfactory outcomes without the necessity for rigorous problem modeling or the consideration of the diverse constraints typically associated with optimization algorithms. Instead, we succinctly define our expectations of the algorithm's performance by evaluating it through the reward system.

Rescue operations in disaster-stricken areas are often characterized by complex and dynamic environments. These operations require the efficient deployment of unmanned aerial vehicles (UAVs or drones) to locate and reach survivors, assess damage, and determine the safest and quickest paths. Traditional approaches to path planning for drones are limited by the dynamic nature of disaster scenarios and may not always adapt effectively. In this context, it is proposed the implementation of Machine Learning techniques, specifically Deep Reinforcement Learning ones, to significantly enhance the real-time path determination capabilities of drones in rescue operations, as well as decision-making for obstacle avoidance or the efficient allocation of resources for effective rescue missions.

The multi-agent patrolling problem, where several UAVs would be deployed so to aid in the rescue operations, might be understood as a further step or extension in the development of such a thesis as it implies higher complexity in the planification. The utilization of multiple UAVs in such a context allows for more comprehensive coverage and a quicker response in critical areas. Nevertheless, it's crucial to carefully address challenges like collision avoidance and resource allocation to fully harness the potential benefits of this extended patrolling problem, ultimately leading to more effective and responsive multi-agent rescue operations [5].

1.2. Thesis objectives

This study places its emphasis on the top-tier module within the UAV control methodology, shown in Figure 1.2. This module receives a detailed map of the area to be overflown, discretized as a gridded representation, employing a strategic approach to establish waypoints for comprehensive coverage of the surface of interest. The primary objective is to collect extensive data from all regions, with particular attention to areas exhibiting substantial concentrations of pedestrians, vehicles, in essence, endangered people, as this is where rescue operations will need to focus.

In this sense, the **main objectives** of this thesis are:

- **Comprehensive Review**: To conduct a comprehensive review of existing techniques in the realm of exploration through Deep Learning (DL), emphasizing their applicability to patrolling in natural environments.
- **Adaptive Patrolling Algorithm**: Develop an adaptive patrolling algorithm based on Deep Learning techniques, with a focus on effective path determination, no matters the map topography. The algorithm needs to allow for real-time decision-making for the UAV movement within the map, while it would cooperates with the TBs that still provide wireless communication services to UEs.
- **Initial Environmental Monitoring Study**: Undertake an initial study utilizing artificial intelligence techniques for environmental monitoring during rescue missions in natural areas, demonstrating the viability of efficient

solutions to the problem.

- **Parameter Evaluation**: Present and analyze typical learning parameters and their impact on the overall performance of the patrolling algorithm.

The algorithm's design will prioritize autonomy, efficiency, and adaptability for rescue operations in challenging natural settings. As for the **functional requirements** of the algorithm:

- **Comprehensive Area Coverage**: The algorithm should strive to create an efficient patrolling pattern that covers as much of the natural area as possible, ensuring thorough surveillance and rapid response capabilities.
- **Temporal Redundancy**: The algorithm should incorporate temporal redundancy by revisiting previously patrolled areas to ensure updated and accurate data collection.
- **Prohibition Enforcement**: The algorithm must be equipped to reject any actions that could lead the patrolling team into off-limit or dangerous zones.
- **Environmental Adaptability**: The algorithm should demonstrate the flexibility to adapt to varying environmental conditions and topographies, making it a valuable asset for diverse natural environments, specially in post-disaster conditions.
- **Parametrization**: The algorithm should offer parameter configurability to cater to the specific requirements and constraints of different rescue scenarios.

1.3. The patrolling problem

The *Complete Coverage Path Planning (CCPP)* problem is defined as the task of efficiently traversing a continuous or discrete space that is navigable by an agent, with the goal of covering its entire, or nearly entire, area while adhering to criteria related to total surface coverage, minimal redundancy, or path optimality. The concept of complete area coverage represents a recurring and fundamental challenge within the broader field of robotics. For instance, this problem is exemplified in the context of household cleaning robots like Roomba, where the objective is to systematically traverse all surfaces of a pre-mapped area without any omissions. In these cases, the term "complete coverage" signifies the aim to leave no portion

of the mapped area unvisited, with all areas being of equal significance.

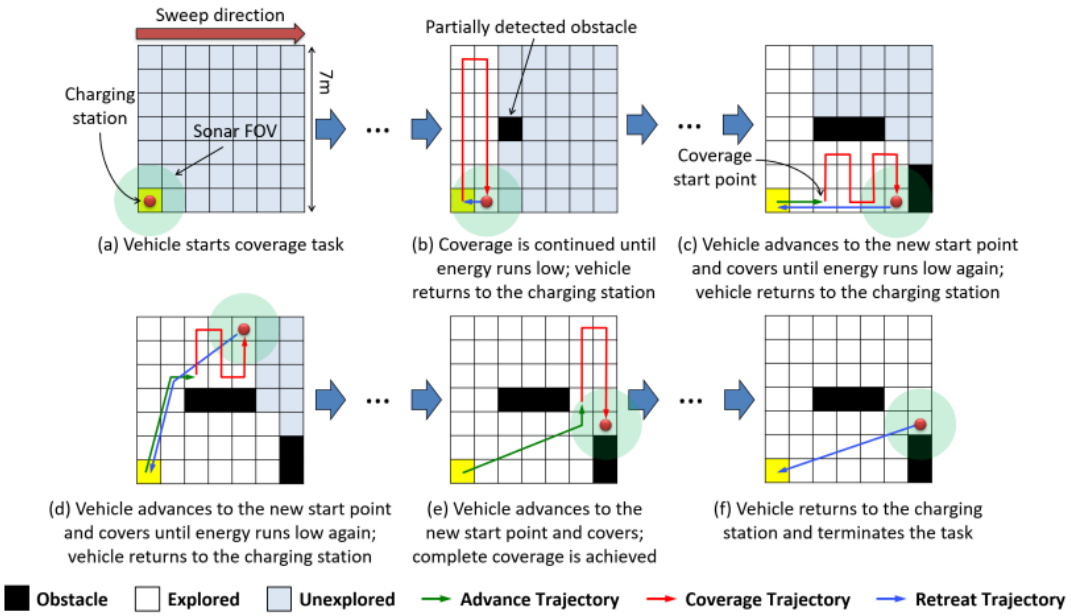


Figure 1.4: An example illustrating the execution of a complete coverage algorithm in an environment containing unknown obstacles [6].

A common characteristic of problems related to complete coverage is the existence of a terminal state in which each area has been visited at least once. Once this terminal condition is achieved, the mapping task can be considered successfully resolved. The specific objectives, whether aimed at reaching this final state while minimizing the repetition of certain areas (thereby minimizing redundancy), depend on the problem's formulation and nature, as illustrated in [7] and [8].

Although also often encountered in fields like robotics, security, and surveillance, the *patrolling problem* or *watchman problem* presents some dissimilarities in their eventual objective. Coverage problems aim to cover all points or areas within a region at least once, while patrolling problems focus on efficiently and periodically revisiting specific locations within the area. To rephrase, coverage is usually a one-time task or operation, ensuring that every point is addressed, while patrolling involves ongoing and repeated visits to maintain security or continued monitoring.

An analogy can be drawn between the patrolling problem and a museum guard: the guard must go through all areas of the museum and also revisit each area

periodically to ensure that no zone is forgotten. In this problem, the need to distinguish between more interesting areas that need to be visited more frequently than less critical ones is addressed [9].

In the concrete case of post-disaster environments, where conditions can be altered every few minutes, visiting each parcel of the region may be as important as redundancy in the acquired information. That is why the patrolling approach will be the one selected for this thesis. Hence, deserted areas at a specific timestep of the monitoring process will be revisited to ensure the absence of people in need for help a posteriori.

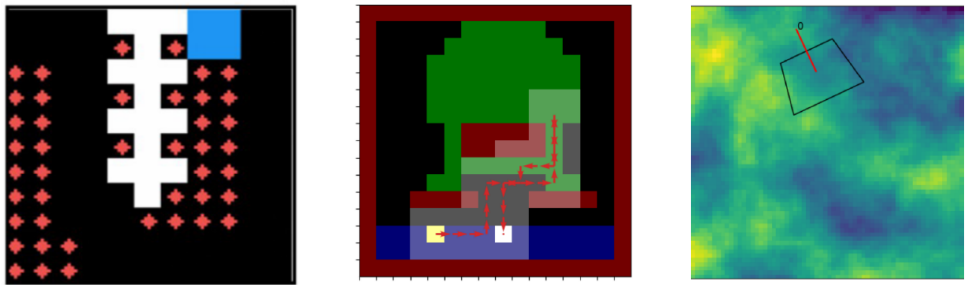
1.4. The environment definition and agent model

In the context of both of these problems, the initial definition of the environment plays a pivotal role. It serves to dictate the permissible movements of the robot or agent and establishes the criteria for boundary conditions and optimality.

Many scholarly works addressing these problems, particularly those utilizing Reinforcement Learning or similar techniques, opt for the simplification of maps. This simplification often involves discretizing two-dimensional spaces into square cells, effectively representing the real-world map. In this representation, it is assumed that the robot can occupy a specific cell or multiple cells within the discrete map, and each of these spaces corresponds to a certain real-world area. It's important to note that a finer grid with smaller cells results in a more accurate representation of the real world; however, it also imposes greater computational demands on the algorithm.

A *relevance map*, in its fundamental representation, takes the form of a two-dimensional, rectangular depiction of the environment under surveillance. This map is structured as a grid, with individual cells assigned specific weights, denoting the significance of observing particular areas — essentially, their level of importance. For coverage-related issues, binary maps are prevalent, employing values of 0 and 1 to distinguish navigable regions from those that are impassable. However, in a more patrolling-oriented context, the value assigned to each cell can

vary, influenced by factors such as the time since the last visit or the inherent likelihood of individuals being present in that specific area.



(a) (b) (c)

Figure 1.5: Relevance map possible representation approaches [7], [8] and [9].

Within this context, various adaptations have been developed to accommodate uncertainties within the map, or, in some instances, to address scenarios where the map is not initially known. In this particular case, an algorithm employing Simultaneous Localization And Mapping (SLAM) is utilized for real-time path planning. This approach serves to optimize energy utilization during coverage and the exploration of uncharted areas.

The agent's role is of paramount importance in shaping the problem definition. Consequently, the problem is categorized as either a single-agent scenario, featuring a solitary robot within the environment, or a multi-agent context, encompassing multiple drones with variable numbers sharing the same spatial domain. In the latter scenario, the algorithm must not only address comprehensive map coverage but also consider the potential for inter-agent collisions and the real-time exchange of valuable information [5].

In practice, the information received by the UAV from its surroundings at any given moment is inherently partial, owing to the constrained reach of its onboard cameras and sensors. Furthermore, the input information can manifest in different formats, such as the agent's positional data on the map or a detailed RGB image of the environment. This latter comprehensive imagery permits the identification of the agent's location and the presence of obstacles, thus enriching the environmental characterization.

2 | State of the art

In the subsequent section, pertinent prior research will be addressed. Acquiring data from sensor devices within an open-air setting imposes demanding limitations on the planning of trajectories for autonomous UAVs. The operational duration of drones is significantly constrained by battery energy, and the intricate urban landscape presents formidable obstacles to navigate.

In the industrial sphere, the utilization of cellular-connected Unmanned Aerial Vehicles (UAVs) for various applications has garnered significant attention and innovation in recent years. A prime example of this technological advancement can be observed in the groundbreaking initiatives led by industry giants such as Amazon Prime Air and Alphabet's Project Wing. These ventures have revolutionized the landscape of delivery services by exploring the potential of UAVs to efficiently transport goods over short to medium distances, thereby offering faster and more flexible delivery options to consumers worldwide [10], [11].

Furthermore, Google's ambitious Loon Project stands as a testament to the versatility of UAVs beyond traditional delivery services. The project, launched in [12], aimed to provide internet connectivity to remote and underserved areas using high-flying balloons equipped with communication technology. In a pioneering effort to bridge the digital divide, Google conducted tests involving the deployment of UAVs for wireless communication, thereby exploring novel solutions to connect the unconnected and expand access to information and opportunities.

The exploration of UAV-based stations has emerged as a focal point of research and development in the aerospace industry, with significant advancements witnessed in recent years. Notably, the literature surrounding drone-based stations has experienced extensive development, with researchers and industry players alike pushing

the boundaries of innovation to maximize the potential of these platforms. This development extends not only to the optimization of individual UAV operations but also to the exploration of multi-agent scenarios involving fleets of UAVs operating collaboratively to achieve complex objectives [13].

The seminal work by Mozaffari et al. provides valuable insights into the design, deployment, and performance analysis of drone small cells, underscoring the transformative impact of UAV technology on communication infrastructure. Their research, presented at the 2015 IEEE Global Communications Conference (GLOBECOM), sheds light on the challenges and opportunities associated with integrating UAVs into existing communication networks, offering a roadmap for future development and deployment efforts in this burgeoning field.

Feature selection plays a pivotal role in Machine Learning, particularly in the domain of victim detection within the context of rescue operations. The ability to discern relevant features from complex datasets is essential for improving the efficiency and accuracy of search and rescue (SAR) efforts. In recent years, advancements in technology, coupled with the proliferation of unmanned aerial vehicles (UAVs) and deep learning techniques, have revolutionized the field of SAR, enabling rapid and precise identification of victims in diverse environments.

One notable study, presented in [14], introduces an autonomous drone designed specifically for search and rescue operations in dense forest areas. Leveraging airborne optical sectioning and real-time image processing techniques, the prototype drone swiftly locates victims amidst challenging terrain. By employing deep learning algorithms for person classification, the system achieves remarkable accuracy in identifying individuals in distress. The primary focus of this research is on expediting time-critical SAR operations, where every moment counts. Moreover, the transmission of minimal data to rescue teams ensures the efficient use of bandwidth, enabling seamless communication and coordination during rescue missions.

Building upon this foundation, [15] proposes a novel approach to expedite SAR operations, particularly in maritime environments. By integrating UAVs with deep learning algorithms, the study aims to enhance the efficiency and effectiveness of

victim detection on the sea surface. The system utilizes GPS location data in conjunction with DL algorithms to identify and locate individuals in distress. Furthermore, the researchers make use of simulated marine environments to generate training data, enabling the development of robust models capable of accurately detecting victims in real-world scenarios.

In a related vein, [16] explores the application of DRL for autonomous search and rescue missions. By harnessing the power of DRL algorithms, the study seeks to empower UAVs with decision-making capabilities, enabling them to autonomously navigate complex environments and identify individuals in need of assistance. This research represents a significant step towards achieving greater autonomy and efficiency in SAR operations, ultimately enhancing the effectiveness of rescue efforts and saving lives.

However, the focus of this thesis lies primarily in achieving redundancy in environmental observation, rather than in the communications established by drones with their remote controls or in the identification of individuals. The aim is to address the issue of patrolling with homogeneous coverage and temporal redundancy, enabling complete decision autonomy for the UAV through DL techniques.

The provided table offers a comprehensive overview of relevant researches on UAVs coverage or patrolling with Deep Learning. Each entry presents key details such as the objective, scenario, state representation, reward scheme, agent type, reinforcement learning (RL) algorithm, estimation function, and hyperparameters employed in the respective studies.

Reference	Scenario	State	Reward	Agent	RL algorithm	Estimation function	Hyperparameters
Lakshmanan, 2020	-**Objective**: CCPP - 120 different **gridmap**-type scenarios. - 3 cell types: visited, unvisited and obstacle.	- **RGB image** of the complete scenario (Fully Observable)	-**Cst. reward** for new visited cell (+1) and for completing task (+50) - **Cst. penalization** for illegal movements (-0.5)	-**Mono-agent** with variable geometry (Tetromino-Like) - 11 actions: 4 translations and 7 shape transformations	- Async. Advantage Actor Critic with Experience Replay (**Off-policy A3C**)	**Secuential Neural Network**: - Convolutional Layer 8x8x32 (ELU) - Convolutional Layer 8x8x32 (ELU) - Convolutional Layer 8x8x32 (ELU) - Fully connected 512 (ELU) - LSTM 256 (Sigmoid)	- Learning Rate = 0.0001 - Discount factor = 0.9 - Exp. Buffer size = 20,000 - Epochs = 10,000
Piciarelli, 2019	- **Objective**: Patrolling drones - **Gridmap**-type scenario with variable interest	- **Visible portion of the map** (agent camera) and agent position	- **Variable reward** based on a dynamic map within 1 (unvisited cell of great interest) and -0.9 (visited cell with minimum interest)	- **Mono-agent** with visual camera - **12 discrete actions**: 6 translations, 4 camera movements and 2 zoom in-/out	- **Double Deep Q Learning** with soft-update of target function	**Secuential Neural Network**: - Convolutional Layer 8x8x16 (ReLU) - Fully Connected Layer 1024 (ReLU) - Fully Connected Layer 1024 (ReLU) - Fully connected 12 (Linear)	- Num. of steps =20
Theile, 2020	- **Objective**: CCPP mono-agent with energy movement minimization - **Gridmap**-type scenario with take-off, landing and obstacles zones	- **5 channels image** of the complete scenario (Fully Observable)	- **Cst. reward**: when a cell is visited for the 1^{st} time - **Cst. penalization**: illegal action, every new step or battery off wothout reaching landing zone	- **Mono-agent** - **5 discrete actions**: 4 movements (N, S, E, W) and landing	- **Double Deep Q Learning** with soft-update of target function	**Secuential Neural Network**: - Convolutional Layer 16x16x5 (ReLU) - Convolutional Layer 16x16x16 (ReLU) - Fully connected Layer 256 (ReLU) - Fully connected Layer 256 (ReLU) - Fully connected Layer 256 (ReLU) - Fully connected Layer 5	- Batch size = 10,000 - Minibatch size = 128 - Learning rate = 0.001 - Discount factor = 0.95 - Epochs = 10,000 - Target update rate = 0.005
Adepegba, 2016	- **Objective**: CCPP multi-agent based on Voronoi map and risk of collision function - **Continuous** scenario	- **Position and velocity of each agent**	- **Reward** calculated as the error between each agent position and its centroid in the Voronoi map (optimum solution)	- **Multi-agent** with continuous actions based on double integrator dynamics	- **Advantage Actor Critic (A2C)**	- **Neural Network (input + hidden + output)**	- Learning rate = 0.0009 - Discount factor = 0.99 - Epochs time = 350 secs
Smith, 2023	- **Objective**: Exploration and mapping of unknown environments - **Scenario**: Real-world outdoor environments with varied terrain and obstacles	- Laser-based **3D point cloud representation** of the surroundings	- **Sparse reward** for uncovering unexplored areas, penalization for collisions	- **Multi-agent** system with heterogeneous capabilities, including aerial and ground vehicles	- Deep Deterministic Policy Gradient (**DDPG**) with prioritized experience replay	- Convolutional Neural Network (**CNN**) combined with Long Short-Term Memory (**LSTM**) for sequence modeling	- Learning rate = 0.0005 - Discount factor = 0.95 - Batch size = 128 - LSTM hidden units = 256

Table 2.1: List of relevant researches on UAVs patrolling with Deep Learning.

3 | Deep Reinforcement Learning

This chapter will provide an in-depth examination of the theoretical foundations that underlie the development and execution of drone patrolling in post-natural disaster scenarios. The recent surge in technological advancements has ushered Artificial Intelligence into diverse domains of application, with Machine Learning, particularly Reinforcement Learning (RL), taking a central role in this thesis. Within this framework, we will conduct a comprehensive analysis of the structure, operation, and metrics of various RL techniques [17]. This analysis will serve to substantiate our choice of methodology and elucidate the implications it holds for the learning process and eventual outcomes.

The great number of automated learning algorithms arisen led to introduce a classification of them. Supervised, unsupervised, and reinforcement learning are three major paradigms within the field of ML, all three aiming to extract knowledge from data, whereas they differ in the nature of the data and the learning objectives.

Supervised learning uses labeled data and aims to minimize prediction errors, while unsupervised learning receives unlabeled data and the focus is on discovering patterns. On the other hand, Reinforcement Learning involves interaction with an environment, being its emphasis on maximizing long-term rewards. That is the reason why it is the preferred approach for real-time decision-making problems, e.g. the UAV patrolling problem in post-disaster scenarios.

RL has gained recent popularity in the field of ML, primarily for its ability to solve learning problems without relying on a predefined model of the environment. It offers a computational approach to understanding and automating goal-directed learning and decision-making. In this framework, an agent's objective is to choose actions in an environment, based on observing the causal relationship between an

action or stimulus and its response, to maximize a numerical signal, known as a reward, which measures the agent's performance. Basically, the agent does not initially know which actions to take, leading to an exploration/exploitation dilemma during the learning process. The agent must try various actions and gradually favor those that seem most effective. This concept is intuitive, and several examples of RL can be found in our daily lives.

In essence, every Reinforcement Learning algorithm operates as follows:

1. The agent interacts with the environment by taking an action.
2. The agent's state changes within the context of the scenario.
3. Depending on whether this change is better or worse for the desired behavior, the agent receives a reward or penalty.
4. The agent evaluates the causality between the action, state, and reward. Over time, it begins to associate actions in a given state as potentially good or bad based on past experience.

Deep Reinforcement Learning (DRL) is a machine learning subfield that combines deep learning (DL) with reinforcement learning (RL) principles to derive optimal solutions through experiential learning. DRL approaches can be categorized into three main groups: model-based, value-based and policy-based, as depicted in Figure 3.1.

Model-based RL relies on providing an environment model to the agent or instructing the agent to learn an environment model for task execution. Model-based learners are often more efficient in terms of required experiences, although it may introduce inaccuracies and imprecision affecting policy and task performance. This can be further divided into two subcategories: "Given the model" and "Learn the model." In the former, algorithms such as AlphaZero and Expert Iteration make use of pre-defined models to simulate the environment dynamics. In contrast, the latter category involves learning the model from data, with techniques like World Models or Imagination-Augmented Agents (I2A).

On the other hand, Model-free DRL dispenses with the need for an explicit model

of the environment and focuses on directly learning policies or value functions from interactions with the environment. This branch can be subdivided into Value-based and Policy-based methods.

In value-based RL, the agent aims to discover the policy that maximizes a value function over a sequence of actions. Policy-based RL, in contrast, seeks to identify the policy leading to the optimal objective function, further subdivided into deterministic and stochastic policies. The former maintains consistent actions across states, while the latter incorporates probabilistic variations in actions. On Policy methods such as Sarsa update their policies based on the current policy, while Off Policy methods like Q-Learning, including its deep variant Deep Q Network (DQN), learn from experiences generated by a different policy.

Policy-based methods, in contrast, directly parameterize the policy and optimize it to maximize expected rewards. This category includes algorithms such as Deterministic Policy Gradient, Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO), and Actor-Critic (AC) methods.

In the next sections, the fundamentals of Deep Q Learning will be discussed, and reasons for its suitability for tracking or patrolling tasks will be presented. In reinforcement training processes, which will be followed in the patrolling problem, the emphasis is not so much on obtaining an immediate high reward through short-term good actions, but rather ensuring that over the entire agent's action time (referred to as an episode), typically marked by the number N of movements allowed by the drone's battery, the reward is maximized at the end.

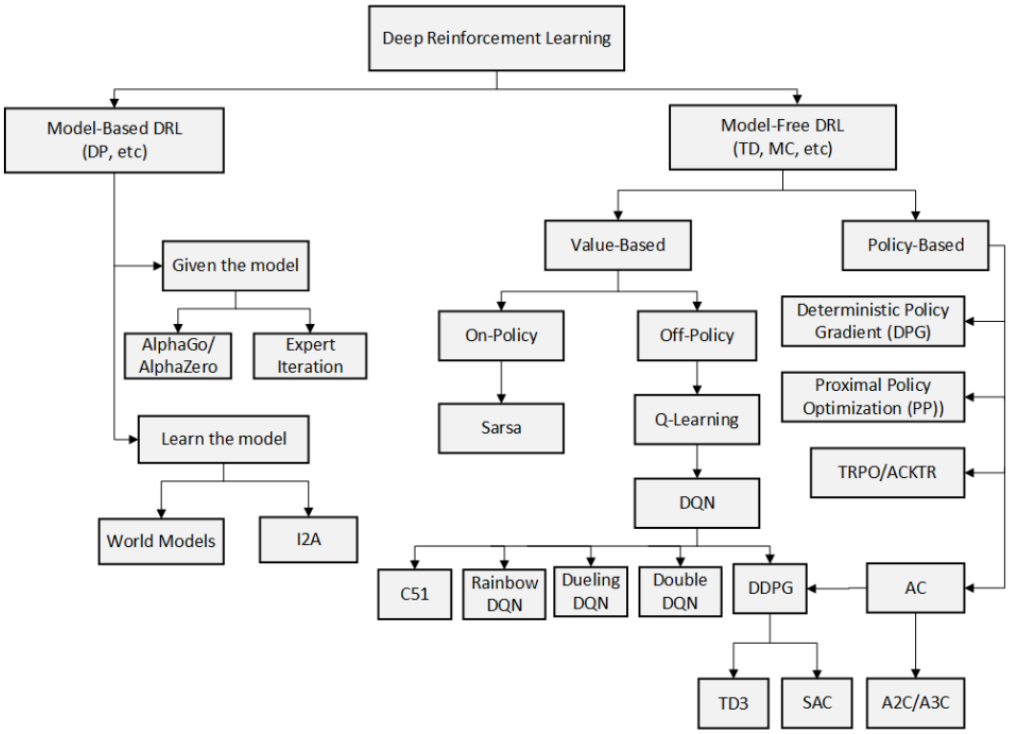


Figure 3.1: Taxonomy of Reinforcement Learning Algorithms [18].

3.1. Problem setup

The general RL problem is formulated as a discrete-time stochastic control process in which an agent interacts with its environment as follows: The agent begins in a specific state within its environment, denoted as $s_0 \in \mathcal{S}$. At each time step t, the agent receives a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, based on the policy $\pi(a_t|s_t)$, representing the agent's behavior. The agent receives a scalar reward r_t and transitions to the next state s_{t+1} based on the environment's dynamics or model, which is determined by the reward function $\mathcal{A}(s, a)$ and the state transition probability $\mathcal{P}(s_{t+1}|a_{t+1})$, respectively. In an episodic problem, this process continues until the agent reaches a terminal state, at which point it restarts. The return $\mathcal{R}_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ represents the accumulated reward with a discount factor $\gamma \in (0, 1]$. The agent's objective is to maximize the expected long-term return from each state. This problem is defined in discrete state and action spaces but can be extended to continuous spaces with relative ease.

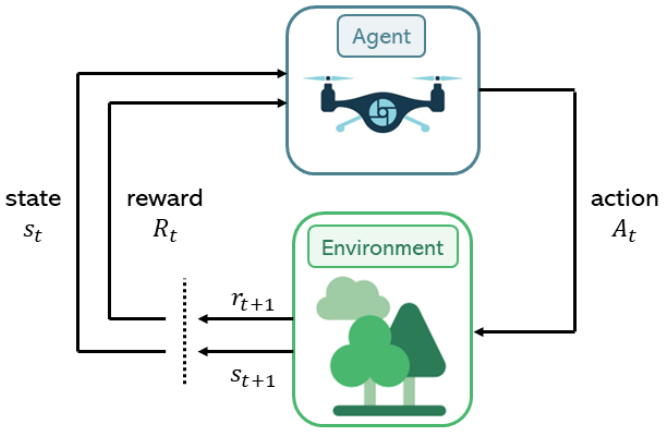


Figure 3.2: The agent-environment interaction in a RL process [19].

3.2. Markov Decision Process

Reinforcement learning leverages the formal structure of Markov Decision Processes (MDPs) to establish the interaction between a learning agent and its environment, which is characterized by states, actions, and rewards. A sequence of states is deemed Markov when the likelihood of transitioning to the subsequent state, s_{t+1}, is solely contingent on the taken action a_t the current state, s_t, and does not rely on any prior states, such as s_1, s_2, ..., s_{t-1}. Besides, if it is time-homogeneous, the probability of the transition results to be independent of t.

Definition 1. Markov Decission Process

A Markov Decission Process is a tuple $(S, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{R})$, where

- S is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is the state transition probability matrix.

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{3.1}$$

- $\gamma \in (0, 1]$ is a discount factor.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ is a reward function.

For reinforcement learning tasks, which break naturally into sub-sequences, called episodes (*episodic* task), the reward function is equal to the sum of individual

rewards:

$$\mathcal{R}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.2}$$

3.3. Q-function and Bellman optimality

The Q-function, also known as action-value function, provides the expected value of the reward starting from state s, taking the action a, and thereafter following policy π.

$$\mathcal{Q}_\pi(s,a) = \mathbb{E}_\pi[\mathcal{R}_t | s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{N} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right] \tag{3.3}$$

Similarly, the state-value function for policy π is the expected reward when starting in s and following π.

$$\mathcal{V}_\pi(s) = \mathbb{E}_\pi[\mathcal{R}_t | s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{N} \gamma^k r_{t+k+1} | s_t = s\right] \tag{3.4}$$

If the Bellman equation is hereafter introduced, the state-value function yields:

$$\begin{aligned} \mathcal{Q}_\pi(s,a) &= \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \mathcal{Q}_\pi(s',a')\right] \\ \mathcal{V}_\pi(s) &= \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \mathcal{V}_\pi(s')] \end{aligned} \tag{3.5}$$

For any MDP there exists an optimal policy π_* for which an optimal value of both the action-value function and the state-value function are attained. Hence, the self-consistency condition given by the Bellman equation for state values leads to:

$$\mathcal{V}_*(s,a) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \mathcal{Q}_\pi(s',a')\right] = \max_a \mathcal{Q}_\pi(s,a) \tag{3.6}$$

The resolution of the MDP will be given once $\mathcal{V}_*$ was found, in other words, the policy $\pi(s)$ that maximizes the reward moving through the different states and the actions that make $\max_a \mathcal{Q}_\pi(s,a)$. Nonetheless, it is immediately detectable that the computational cost of solving the system of equations will be inadmissible as far as each state and action incorporates a new equation to the system, and the MDP

resolution needs of system model to concatenate different states.

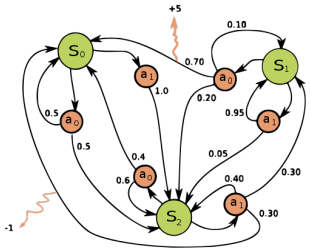


Figure 3.3: Simple Markov Decision Process (MDP) example with three states (green circles), two actions (orange circles) and two rewards (orange arrows) [20].

3.3.1. Q-learning algorithm

Initially, the Q-function remains unknown, and the algorithm's goal is to discover its value. To achieve this, a technique known as Off-policy Time Difference (TD) Control is employed: at each step, the Q-function is estimated by observing the reward from taking action a starting from state s. This eliminates the need to model the underlying dynamics for Q-estimation, enabling state value estimation solely based on experience. In essence, TD estimation is substantially an application of the gradient descent technique to the Q-function with a learning rate parameter, α.

Algorithm 3.1 Q-Learning algorithm

1: $\mathcal{Q} \leftarrow$ Initial empty $\mathcal{A} \times \mathcal{S}$ table
2: end $\leftarrow 0$
3: **while** $end == 0$ **do**
4: step $\leftarrow 0$
5: Reset S and select starting s
6: **while** step $< step_{MAX}$ **do**
7: Choose action a from $\mathcal{Q}(s,a)$ based on $\epsilon - greedy$ policy
8: Take action a and observe r, s'
9: $\mathcal{Q}(s,a) \leftarrow \mathcal{Q}(s,a) + \alpha[r + \gamma\, max\mathcal{Q}(s',a) - \mathcal{Q}(s,a)]$
10: s $\leftarrow$ s'
11: step $\leftarrow$ step+1
12: **if** $\Delta\mathcal{Q} < threshold$ **then**
13: end $\leftarrow 1$

14: **end if**
15: **end while**
16: **end while**

This algorithm will run until the values of Q-table do not change, or at least below a certain threshold. The optimum path is derived from, at each state s, selecting the action a that $\underset{a}{max}\mathcal{Q}_\pi(s,a)$, so that there will be as many tables as actions and states.

One significant advantage of estimating optimal actions through the Q-Learning algorithm, as opposed to directly estimating the optimal value function $\mathcal{V}_\pi(s)$, is the ability to use an arbitrary policy to choose actions at each step while still finding the optimal value of the Q-table. This decouples the problem of the agent's behavioral policy for exploring the state space. Such algorithms that utilize a different behavioral policy (random) from the optimization policy (greedy) are known as Off-Policy algorithms. By selecting an arbitrary behavioral policy, the resolution of the MDP can be achieved. Nonetheless, the fact that the dimension of the Q-function/table increases for each discrete action that can be performed makes this approach infeasible as soon as the size of the problem cease to be negligible.

A paridagmatic example for which the $Q-Learning$ algorithm is understandably applicable is the so-called *Frozen Lake* game. It is a simple environment composed of tiles, where the agent has to move from an initial tile (*start*) to a final one (*goal*). Tiles can be either a safe frozen one or a hole that gets you stuck forever. There are 4 possible actions: left, right, up and down. The agent must learn to avoid holes in order to reach the goal in a minimal number of actions. In this context, the Q-table can be modelled as $16 \times 4 = 64$ discrete values table, a rather low number of elements, that allows for an easy implementation of Q-learning algorithm.

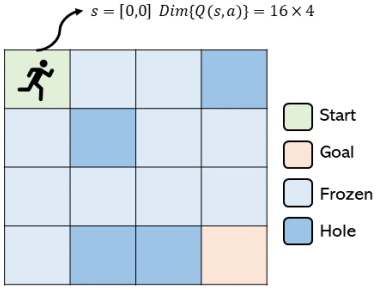


Figure 3.4: Frozen Lake paradigm scheme.

3.3.2. ϵ-greedy policy

The ϵ-greedy policy is a common strategy used in Q-learning which balances exploration and exploitation by occasionally taking random actions (exploration) and mostly choosing the action with the highest estimated Q-value (exploitation). Therefore, with this policy, a balance is sought between reinforcing what has already been learned and discovering new potential optimal paths. Without exploration, the algorithm would become stuck in a local minimum due to a lack of knowledge about the stimulus-response relationship, while without exploitation, the search would be mostly random and without a clear learning component.

The ϵ-greedy policy consists on generating a random number between 0 and 1, such that if the random number is less than ϵ, choose a random action (exploration), and exploitation otherwise. As far as exploration is desired in the first steps of the training phase, ϵ will obey a decreasing tendency as the training advances, ensuring the possibility of exploration even at the very end so to leave space for improvement.

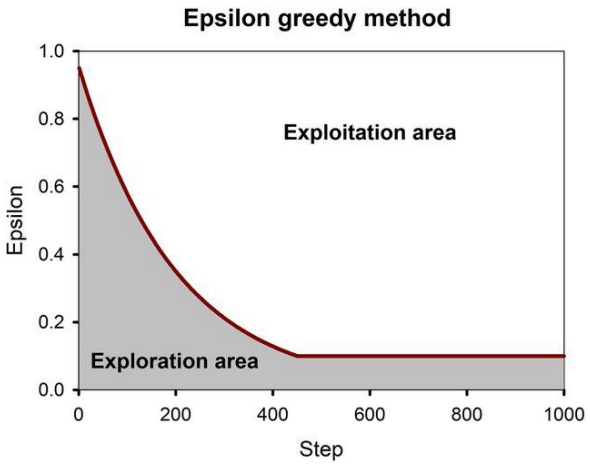


Figure 3.5: ϵ evolution along training [21].

Algorithm 3.2
ϵ-greedy policy

1: $n \leftarrow$ random(0,1)
2: **if** $n < \epsilon$ **then**
3: a $\leftarrow$ sample($\mathcal{A}$)
4: **else**
5: a $\leftarrow \underset{a}{max}\mathcal{Q}(s,a)$
6: **end if**

3.4. Deep neural networks

In previous sections, we discussed the challenges of Reinforcement Learning when dealing with high dimensions, such as a large grid map and a vast number of possible actions or highly variable states that are unsuitable for tabulation, as in the analysis of UAV-captured video frames, each corresponding to an RGB structure with 3840 x 2160 pixels.

Hence, the need arises to find a mathematical tool capable of adequately modeling the Q-function with a realistic computational cost. This is where Deep Reinforcement Learning comes into play. For the first time, in the paper titled "Human-level control through deep reinforcement learning" [22], the concept of using neural networks, particularly deep and convolutional artificial neural networks, to model the Q-function was introduced.

The advantage of employing a neural network lies in its ability to model a complex function with a relatively straightforward arithmetic calculation structure using a variable set of parameters. Consequently, in Deep Reinforcement Learning (DRL), the Q-function could be approximated with rather good results, as follows:

$$\mathcal{Q}(s,a) \approx \mathcal{Q}(s,a,\theta) \tag{3.7}$$

where θ represents a set of parameters of the network, the so-called *weights*.

3.4.1. Basic architecture of neural networks

An Artificial Neural Network draw its inspiration from the concept of emulating biological models of real neurons. In this model, the fundamental unit of information is the neuron or *perceptron*. Each neuron performs a weighted sum operation on input signals received from other neurons upstream. The neural network is composed of multiple interconnected neurons that, through the weighted evaluation of their inputs using weights ($\mathcal{W}$) and biases (b) in a nonlinear function ($\sigma(x)$), can model a mathematical function when its parameters are suitably configured.

In the context of artificial neural networks, a bias term (often denoted as b) is an additional parameter used alongside weights ($\mathcal{W}$) in the calculation performed by each neuron. While weights determine the strength of connections between neurons, biases provide the neural network with the ability to account for situations where all inputs may be zero or when all input weights are zero.

Mathematically, the bias term can be thought of as an offset or threshold that helps in shifting the activation function of a neuron. Without the bias term, the output of a neuron would solely depend on the weighted sum of its inputs, making it difficult to model more complex functions.

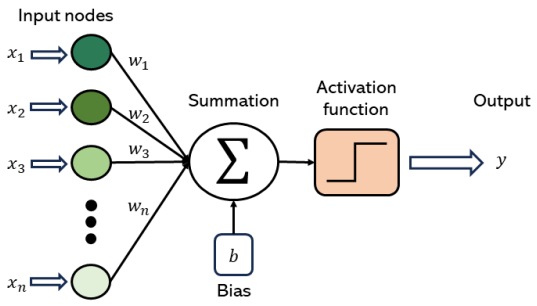


Figure 3.6: Perceptron or neuron architecture.

Definition 2. Activation function

The activation function $\sigma(x)$ operates on the weighted sum of the inputs to the neuron (pre-activation) and produces the output of the neuron (post-activation), introducing non-linearity into the network, which allows the neural network to approximate complex functions, learn hierarchical features from the input data and model complex decision boundaries (crucial for tasks like image recognition, natural language processing...).
Common activation functions used in deep neural networks include the sigmoid function, hyperbolic tangent (tanh), rectified linear unit (ReLU) and the exponential linear unit (eLu).

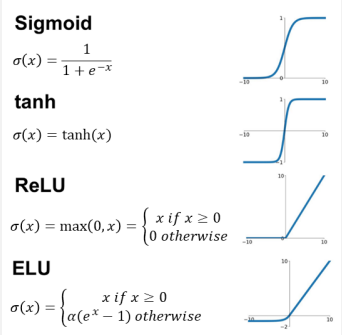


Figure 3.7: Typical activation functions.

This mathematical plasticity property enables, when incorporating multiple layers with numerous neurons, the derivation of a nonlinear function that maps an input set x to an output space y. Fully-connected or dense artificial neural networks are one of the most common ways to structure a neural network, and they are used in tasks involving the classification or identification of elements based on input features x. In the case of Deep Reinforcement Learning (DRL), the input space corresponds to the state (encoded in some manner), and the output is the value of the Q-function for each possible action. If there are $\mathcal{A}$ different actions, we observe an output layer consisting of $\mathcal{A}$ neurons that evaluate $\mathcal{Q}(s, a_i)|_{i \in \mathcal{A}}$.

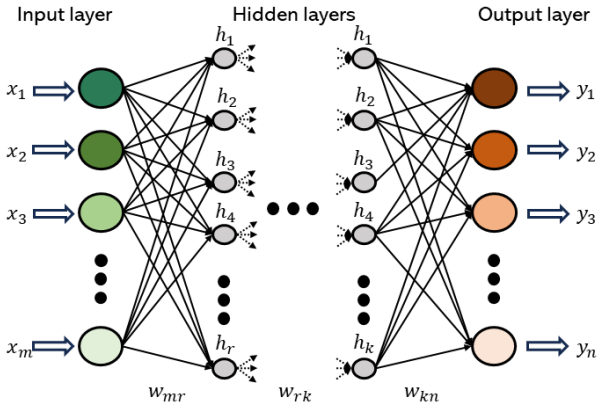


Figure 3.8: Fully-connected neural network architecture.

A neural network involves two fundamental operations:

- **Forward propagation**: *Forwarding* is the computation of the output y given an input x.

$$f_{ij} = \sigma\left[\sum_i \mathcal{W}_{ij} x_i + b_i\right] \tag{3.8}$$

- **Backward propagation**: *Backpropagation* is a fundamental technique used to train artificial neural networks in order to adjust the network's parameters $\theta = [\mathcal{W}, b]$.

  Backpropagation calculates the gradient of the loss function $\mathcal{J}$ with respect to the network's parameters. Hence, the weights and biases can be adjusted in the direction that minimizes the loss for future training examples in the dataset.

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial J}{\partial \theta} \tag{3.9}$$

  where α is the learning rate, in other words, how big is the gradient step.

  The calculation of the partial derivatives of $\mathcal{J}$ is done using the chain rule for each of the layers, computing the derivative first for the last parameters and propagating the sensitivity value upstream through the network (hence its name, backpropagation).

Ultimately, the selection of the cost function, the network's neuron count, and the architectural depth is a decision informed by experience, intuition, and iterative experimentation. The ultimate objective remains the minimization of the cost

function J, which explicitly measures the disparity between estimated and actual values.

3.4.2. Convolutional neural networks

Convolutional Neural Networks (CNNs), also known as ConvNets, are a specialized type of neural network designed for processing grid-like data, such as images and videos. They are commonly used in place of standard neural networks because they offer several advantages when dealing with such structured data: hierarchical feature learning (object recognition) or local receptive fields (focus on specific regions while preserving the spatial relationships between pixels).

The essential operation performed by the network on the prior layer, the *convolution*, could be mathematically defined for an image x with $N \times M$ pixels as follows:

$$x \star \mathcal{K} = y[i,j] = \sum_{m=0}^{M} \left[\sum_{n=0}^{N} x(m,n) \star \mathcal{K}(i-m, j.n) \right] \tag{3.10}$$

where $\mathcal{K}$ is the kernel operator or filter. The primary purpose of the kernel is to extract features from the input data by sliding it across the entire input, one small part (patch) at a time. At each position, the kernel performs an element-wise multiplication with the values in the input data and then sums up the results. This operation creates a feature map or activation map for that specific position, enabling to identify edges, textures, or patterns.

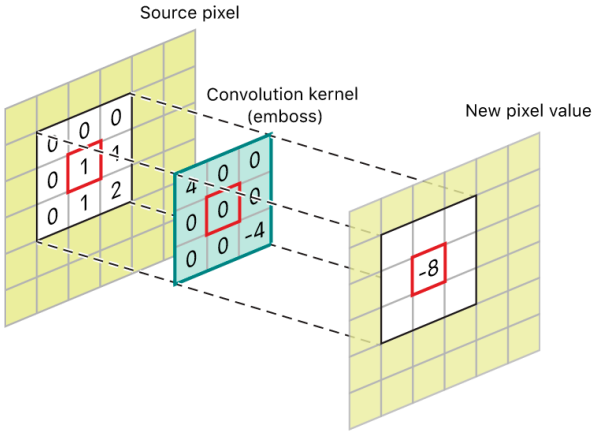


Figure 3.9: Convolution depiction for a given pixel.

Once the image or input structure has traversed through several layers in the convolutional neural network, it is transformed into a smaller image with multiple channels. This transformed image can then be flattened into a vector, and a straightforward logistic regression is applied at the final stage to identify the relevant areas for our classification task. In this logistic regression step, which utilizes the logistic function, the network determines the specific regions within the representation that are essential for classification. It generates an output, which is subsequently compared to the target, and any errors are propagated backward through the network.

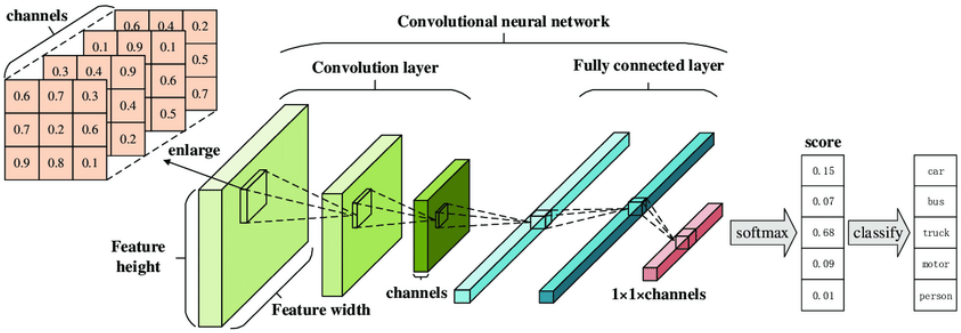


Figure 3.10: Dense CNN scheme example for object classification [23].

3.5. Deep Q-Learning

The Deep Q-Learning algorithm emerges when techniques inherent to deep neural networks are employed in the formulation of what is known as the state-action value function, the purpose of which was elucidated on page 24.

$$\mathcal{Q}(s,a) \approx \mathcal{Q}(s,a,\theta) \tag{3.11}$$

Acquiring proficiency in a sequential decision-making task becomes rather intricate in cases where the number of possible states and actions would result in an unmanageably large Q-table (*dimensionality curse*). Consider the example of a UAV endeavoring to carry out a patrolling operation over a designated territory, discretized as an $N \times M$ grid, which would yield a Q-table with dimensions of $N \times M \times 8$ cells (where 8 corresponds to the drone's ability to move to all adjacent cells).

In an online learning case where, in parallel to learning, the agent gradually gathers

experience in the environment via an exploration/exploitation strategy, the values of the Q network are updated based on the *Time Difference* method.

$$\mathcal{Q}(s,a)^{k+1} = \mathcal{Q}(s,a)^{k} + \alpha(r + \gamma \underset{a}{max}\mathcal{Q}(s',a',\theta)^{k} - \mathcal{Q}(s,a,\theta)^{k}) \tag{3.12}$$

Thus, the *target* function is defined as:

$$\mathcal{Q}_{target} = r + \gamma \underset{a}{max}\mathcal{Q}(s',a',\theta) \tag{3.13}$$

If terms are rearranged now,

$$\mathcal{Q}(s,a)^{k+1} = (1-\alpha)\mathcal{Q}^{k} + \alpha\mathcal{Q}^{k}_{target} \tag{3.14}$$

Hence, by imposing that both the objective function and the Q-function are parameterized as neural networks, the estimation error of the objective function (according to an arbitrary loss function) can be defined. If J_{loss} exhibits a strictly negative gradient and is decreasing with an absolute minimum when $\mathcal{Q}_{target} = \mathcal{Q}$, by updating the neural network parameters, using the gradient descent method, the loss function can be ultimately minimized , thereby achieving a robust estimation of the Q-function.

$$Loss = J_{loss}(\mathcal{Q}_{target}, \mathcal{Q}) \tag{3.15}$$

Definition 3. Loss function

Loss functions are mainly classified into two different categories: Classification loss (predicting the output from the different categorical values) and Regression Loss (predicting continuous values such as weather conditions or the prices of houses).

Hubber loss function is quite fundamental when training a robust neural network as it is less sensitive to *outliers*. It is quite convex in zones where the error is small, whereas the gradient is constant in areas with high errors, so to avoid the explosive gradient problem. The Huber Loss offers the best of two worlds by balancing the MSE and MAE together.

$$L_\delta(\epsilon) = \begin{cases} \frac{1}{2}\epsilon^2 & \text{for} |\epsilon| \leq \delta, \\ \delta(|\epsilon| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{3.16}$$

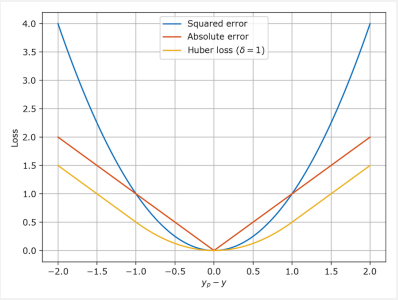


Figure 3.11: Comparison between possible loss functions.

Two important ingredients of the DQN algorithm as proposed by [22] are the use of a target network and buffer replay. This algorithm uses two heuristics to limit the instabilities:

- $\mathcal{Q}_{target}$ in Equation (3.13) is replaced by $\mathcal{Q}(s', a', \theta_k^-)$ where its parameters θ_k^- are updated only every $\mathcal{C} \in \mathbb{N}$ iterations. This prevents from instabilities and divergence.
- In an online setting, the replay memory keeps all information for the last $N_{replay} \in \mathbb{N}$ time steps, where the experience is collected by following an ϵ-greedy policy. The updates are then made on a set of tuples $< s, a, r, s_0 >$ (*mini-batch*) selected randomly within the replay memory. Consequently, it provides the possibility to make a larger update of the parameters, while having an efficient parallelization of the algorithm.

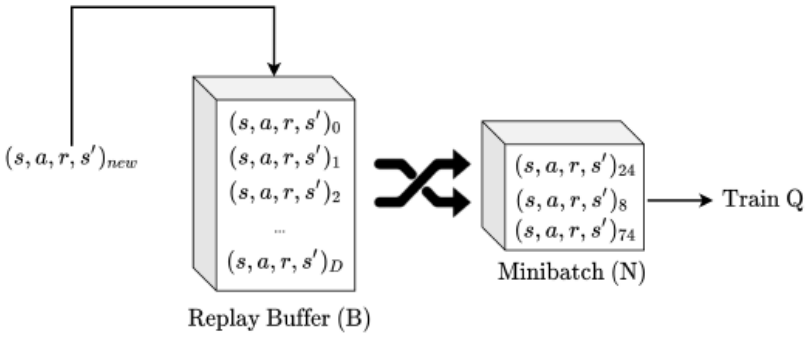


Figure 3.12: Buffer experience scheme.

Algorithm 3.3 Deep Q-Learning algorithm

Random initialization of weights θ in function $\mathcal{Q}(s, a, \theta)$
Initialize $epoch \leftarrow 0$
while $epoch < epoch_{MAX}$ **do**
  step $\leftarrow 0$
  Reset S and select starting s
  while $step < step_{MAX}$ **do**
    Choose action a from $\mathcal{Q}(s, a, \theta)$ based on $\epsilon - greedy$ policy
    Take action a and observe r, s'
    Save tuple (s', r, a, s) in replay memory
    if $N_{experiences} > N_{replay}$ **then**
      $experiences \leftarrow N_{replay}$ experiences (s', a, r, s) from replay memory
      for (s', a, r, s) **in** $experiences$ **do**
        $\mathcal{Q}_{target} = r + \gamma \underset{a}{max} \mathcal{Q}(s', a', \theta)$
        Update θ: $d\theta \leftarrow d\theta + \frac{\partial J(\mathcal{Q}, \mathcal{Q}_{target})}{\partial \theta}$
      end for
    end if
    $\mathcal{Q}(s, a) \leftarrow (1 - \alpha)\mathcal{Q}(s, a) + \alpha \mathcal{Q}_{target}$
    s $\leftarrow$ s'
    step $\leftarrow$ step+1
  end while
  $epoch \leftarrow epoch + 1$
end while

Deep Q-Learning algorithm comes with both advantages and limitations. While it has the potential to yield good results, it is important to acknowledge that we are estimating and training a Q-function through another estimated function, $\mathcal{Q}_{target}$. This technique, known as *bootstrapping*, can occasionally introduce convergence issues in learning and may lead to an overestimation of the target function, resulting in outcomes that deviate from those achieved by other methods.

3.6. Double Deep Q-Learning

The max operation in Q-learning, as expressed in Equation (3.13), relies on the same values both for action selection and evaluation. This can lead to a higher

likelihood of selecting overestimated values, especially when dealing with inaccuracies or noise, resulting in overly optimistic value estimates. Consequently, the DQN algorithm introduces an upward bias.

To address this, the double estimator method employs two estimates for each variable, effectively decoupling the selection of an estimator and its associated value. In the original Double Q-learning algorithm, two separate value functions are learned, and each experience is randomly assigned to update one of the two value functions. This results in having two sets of weights, denoted as θ and θ^-. During each update, one set of weights is used to determine the greedy policy, while the other set is used to determine its corresponding value.

$$\mathcal{Q}_{target} = r + \gamma \mathcal{Q}(s', \underset{a \in \mathcal{A}}{argmax}\, \mathcal{Q}(s', a', \theta), \theta') \tag{3.17}$$

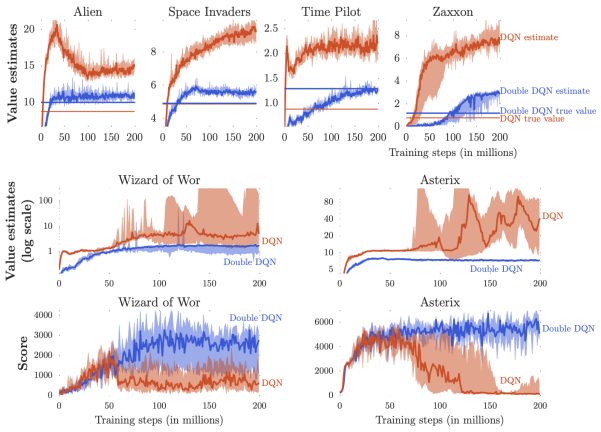


Figure 3.13: Comparison of training results with DQN and DDQN in different scenarios [24].

Algorithm 3.4 Double Deep Q-Learning algorithm

1: Random initialization of weights θ in function $\mathcal{Q}(s, a, \theta)$
2: Copy θ weights from $\mathcal{Q}(s, a, \theta)$ in θ^- for $\mathcal{Q}_{target}(s', a', \theta^-)$
3: Initialize $epoch \leftarrow 0$
4: **while** $epoch < epoch_{MAX}$ **do**
5: $\quad epoch_\theta \leftarrow 0$
6: $\quad$ **while** $epoch_\theta < epoch_{MAX,\theta}$ **do**
7: $\quad\quad$ step $\leftarrow 0$

Reset S and select starting s
while $step < step_{MAX}$ **do**
Choose action a from $\mathcal{Q}(s, a, \theta)$ based on $\epsilon - greedy$ policy
Take action a and observe r, s'
Save tuple (s', r, a, s) in replay memory
if $N_{experiences} > N_{replay}$ **then**
$experiences \leftarrow N_{replay}$ experiences (s', a, r, s) from replay memory
for (s', a, r, s) **in** $experiences$ **do**
$\mathcal{Q}_{target} = r + \gamma \underset{a}{max} \mathcal{Q}(s', a', \theta)$
Update θ: $d\theta \leftarrow d\theta + \frac{\partial J(\mathcal{Q}, \mathcal{Q}_{target})}{\partial \theta}$
end for
end if
$\mathcal{Q}(s, a) \leftarrow (1 - \alpha)\mathcal{Q}(s, a) + \alpha \mathcal{Q}_{target}$
s $\leftarrow$ s'
step $\leftarrow$ step+1
end while
Update θ^-: $\theta^- \leftarrow \theta$
end while
$epoch \leftarrow epoch + 1$
end while

4 | Problem formulation

4.1. Relevance map

The environment constitutes a pivotal aspect in the development of a learning algorithm as it serves as the context within which the agent operates, influencing the success or failure of its mission. The scenario encompasses the following elements, essential for the implementation of a Reinforcement Learning algorithm.

1. **Map type**. The map will be structured as a grid map where each square cell corresponds to a discretized portion of the environment. Cells are distinguished as either accessible, if they fall within the domain of interest for hypothetical rescue operations, or non-accessible if they are out of that area. The size of these cells determines the precision of movement description; a larger size facilitates quicker environmental computations at the expense of realistic space characterization.

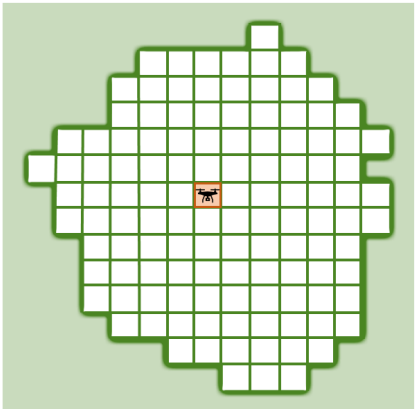


Figure 4.1: Gridmap for the scenario.

This discrete representation of the environment significantly simplifies the computational load during training. Each training step involves the agent advancing on a map cell, requiring recalculations of position, verification of

legal actions, state processing, and weight matrix updates. An excessively large map could notably slow down training, a situation diligently avoided.

2. **Agent-Environment action interface**. The interface between the agent and the environment commences with actions that the agent can execute to alter the state, aiming to obtain a positive reward. Actions represent possible movements the UAV can make, corresponding to movements in the 8 adjacent cells at each moment (following cardinal points: N, S, E, W, NE, NW, SE and SW, as depicted in Figure 4.2).

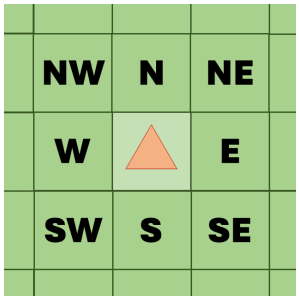


Figure 4.2: Sign convention for the possible movements.

This description of agent movement reflects, more or less realistically, the movement of a real UAV. The training algorithm focuses on identifying waypoints forming the final trajectory. In the meanwhile the low-level module translates cell-to-cell movements into speed and orientation, including the possibility of diagonal movements, even with potentially higher costs, aligning with the real-world scenario.

3. **Reward**. The reward represents the environment's response to the agent's action. The reward policy is designed to reward desired behaviors and penalize illegal or inappropriate actions. The heuristic rule considers efficient exploration as an algorithm allowing the coverage of the maximum possible surface area, with a weighted redundancy criterion. Visiting a recently explored area is considered worse than visiting an unexplored area (much better) or an area visited long ago (not as bad).

4. **Observation interface**. The observation interface is how the agent perceives the state of the environment. The state, within a Markov Decision Process model, may take different forms depending on the algorithm: position in the map as a vector, the map matrix itself, etc. Thus, various representations

of the state have been made possible to study how having more information about the environment affects the training process. Observing only the agent's position differs from having an RGB image providing insights into visited locations and their visit frequency. Both representations have been utilized in this work to assess which one yields better results.

In conclusion, there is a self-made program called CoverageMap.py, which is crucial to this study. This program contains an Environment object designed to simulate scenarios relevant to the research context. For those interested in exploring the detailed implementation of CoverageMap.py and its dependencies, the basis for the source source code is provided in the Appendix of this thesis.

The Environment object encapsulates various methods essential for scenario manipulation. These methods include initialization of the scenario using Environment.reset(), executing actions with Environment.step(action), and visualizing the state of the map using Environment.render(), among others.

It's important to note that while CoverageMap.py serves as the main script, it relies on certain Python Gym libraries to support its functionalities. These libraries, which enable the implementation of reinforcement learning algorithms, are also included in the accompanying source code.

The scenario generated by CoverageMap.py returns a dictionary for each action, providing essential information such as the observation of the state (including the agent's position and weight matrices), the reward generated by the action, and whether the episode has concluded (done).

It's worth mentioning that the design of CoverageMap.py aligns with the fundamental structure of Python Gym library scenarios developed by OpenAI. This design choice facilitates seamless integration with any Reinforcement Learning method designed for Gym. Researchers can utilize this framework to test various algorithms in the context of rescue operations or similar scenarios.

In terms of how the drone observes the region of interest, the camera is assumed

to be fixed in a vertical position, meaning that the camera's main axis aligns with the Z-axis of the drone. Although the battery level could be included as part of the state vector, for simplification, it will be equivalent to setting an initial upper threshold for the number of steps the drone can take before its battery is depleted. Similarly, the relevance map, covering vast distances, leads us to assume that the cell occupied by the UAV at a given time step corresponds to the entire field of view of the drone's camera. The center of this field of view aligns with the projection of the coordinates representing the current position of the drone.

4.1.1. Gridmap generation

To construct the gridmap, a *Python* script utilizing the *OpenCV* library has been devised. This script facilitates the extraction of the contour of the actual area of interest from a map, and through manual adjustment, it selectively isolates the region for discretization based on an user-configurable spatial resolution ratio.

The research presented in this Thesis holds relevance for any natural terrain affected by a natural catastrophe. Nevertheless, henceforth, the subsequent remarks and findings will specifically pertain to a location familiar to individuals residing in Italy. The map employed corresponds to the one utilized by public safety agencies in the context of Mount Etna, outlining the primary area for monitoring during a volcanic eruption.

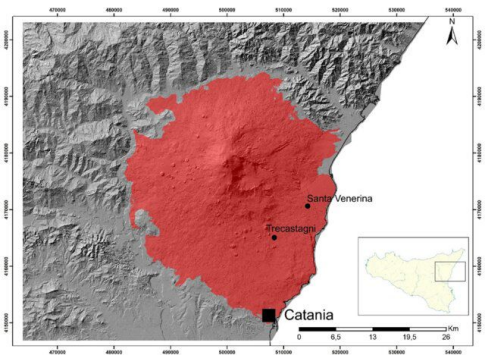


Figure 4.3: Map for Mount Etna.

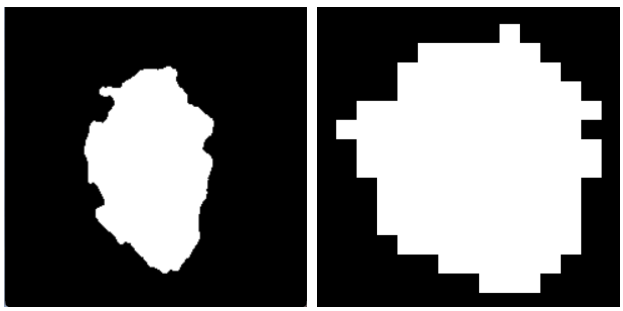
(a) Parameters. (b) Mask. (c) Superposition.

Figure 4.4: Gridmap generation program interface.

The script employs threshold filtering on the image, retaining specific hue, color, and brightness values (pixel values). Subsequently, it applies median filtering to fill in potential gaps in the mask resulting from noise. Lastly, it undergoes scaling according to a linear formula incorporating constants to equate cell width with user-defined meter values.

$$\begin{aligned} px_{width} &= \left\lceil \frac{\text{map width}}{\text{pixel width}} \right\rceil \\ px_{height} &= \left\lceil \frac{\text{map height}}{\text{pixel height}} \right\rceil \end{aligned} \tag{4.1}$$

The resulting pixel relevance map possesses a width of 15 pixels and a height of 16 pixels, with each cell corresponding to a distance in the original map of approximately 500m. The output will be saved as a CSV file containing binary values.



(a) Binary mask. (b) Resulting gridmap.

Figure 4.5: Comparison between initial binary map and its discretization.

4.2. Reward function

The reward law holds a pivotal role within Reinforcement Learning algorithms, serving as the means to model the objective behavior of the agent. A reward law designed to encompass the entire surface must incentive the discovery of unexplored cells (a highly desirable outcome) and also encourage visits to cells that have remained unvisited for a certain duration (a desirable outcome). Moreover, this rule imposes stringent penalties for any illicit actions the agent may attempt, such as engaging in actions that could lead to non-flyable cells or revisiting recently explored cells with readily available recent information (resulting in redundant penalties).

Hence, the objective is to formulate a reward law that translates the state and the executed action into a positive real value when the action aligns with the objective and a negative value when it does not contribute to the goal.

Typically, the reward law is chosen through heuristic criteria, implying that it is crafted with logical rules based on the ultimate goal but generally grounded in fuzzy logic. When quantifying the merit of an action, consideration is given not only to its absolute value but also to its relative value compared to other actions. In this study, it could be posited that an illicit action is five times more detrimental than discovering a new cell but only three times worse than revisiting a cell recently explored.

Excessive penalization of specific actions in this algorithmic paradigm can induce an aversion towards executing those actions in certain states, possibly leading to a systematic avoidance of states located near the boundaries of the map. This situation is unfavorable for coverage purposes. Consequently, penalty values are at times determined through iterative testing, relying on trial and error for adjustments.

4.2.1. Reward for weighted interest matrix

The concept arises that certain regions within the environment may inherently possess greater interest than others. This interest can stem from structural considerations of the terrain or operational factors, such as the impassability of an area, resulting in a lower likelihood of individual presence. Consequently, the absolute

interest matrix, denoted as $\mathbf{R}_{abs}(i,j)$, is defined as a scalar field. For each (i,j) pair, this matrix assigns a value representing the static importance of the corresponding zone, reflecting the interest in that zone regardless of the passage of time.

A constant value for $\mathbf{R}_{abs}(i,j)$ signifies uniform interest across all zones, indicating the objective to traverse each zone equally. The matrix $\mathbf{R}_{abs}(i,j)$ adopts values within the range of 0 to 255, facilitating its representation as either a grayscale matrix or a heat map within the algorithm.

Consider now a weighting matrix, denoted as $\mathbf{S} \in \mathbf{R}^(H \times W)$, with values ranging from 0 to 1. This matrix functions to weigh each position of $\mathbf{R}_{abs}$ based on the time elapsed since its last visit. Consequently, the evolution of $\mathbf{S}$ follows the formula outlined below:

$$\mathbf{S} = \begin{cases} \mathbf{S}(i,j)^{t+1} & = & min[\mathbf{S}(i,j)^t + \delta,\ 1] & \text{if} \quad [x,y]_{agent} \neq [i,j] \\ \mathbf{S}(i,j)^{t+1} & = & 0 & \text{if} \quad [x,y]_{agent} = [i,j] \end{cases} \tag{4.2}$$

The relative weighting matrix $\mathbf{R}$ is the element-wise product of the two matrices:

$$\mathbf{R} = \mathbf{S} \circ \mathbf{R}_{abs} \tag{4.3}$$

Hence, a cell that has been visited recently will be assigned a diminished weight, regardless of its inherent interest. Cells progressively restore their significance over time. The reward is once again characterized as the interest gradient between one cell and another.

$$\rho(s') = \begin{cases} \nabla\mathbf{R} = \mathbf{R}(s') - \mathbf{R}(s) & \text{if} \quad s' \neq \text{illegal} \\ -C_{illegal} & \text{if} \quad s' = \text{illegal} \end{cases} \tag{4.4}$$

The reward mechanism can be built as stimulating movements toward less-explored territories, consistently emphasizing that revisiting a previously surveyed region holds precedence over venturing beyond established boundaries. It underscores that revisiting cells forsaken for an extended duration is more favorable than exploring entirely new areas.

Given the algorithm's protracted execution design, it is judicious to incorporate

a decay factor into the reward mechanism. Following the equation involving $\mathbf{S}$, it is established that after a specific number of $1/\delta$ time steps, a cell regains its maximum value in the $\mathbf{R}$ matrix. The algorithm may thus discern it as more advantageous to revisit a cell not accessed for an extended period than an entirely new area. To mitigate this undesired behavior, it is suggested that each visit to a cell deducts a fixed static interest from the $\mathbf{R}_{abs}$ matrix. In this algorithm, each visit reduces a cell's original interest value by 20%. Consequently, even if a cell initially boasts a maximum interest value of 255, after being visited five times, it loses any interest relative to other cells.

The efficacy of this reward policy has been examined in scenarios where the agent operates in manual mode (a program was devised for offline testing of the reward policy). Analogous to a heatmap, the reward is conferred when traversing less-explored regions (areas with high interest, depicted in blue in Figure 4.6). Maximum interest occurs when maximizing the heat gradient from very warm zones (recently visited, less important) to cold zones (less visited, less interesting). Movement from a recently visited cell (current position) to a warm cell generates a low-interest gradient, and vice versa, for example.

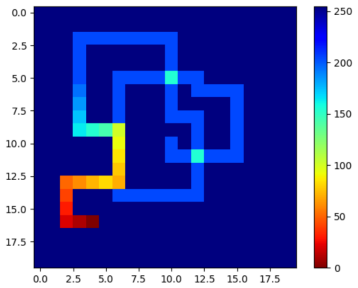


Figure 4.6: Heatmap representation for $\mathbf{R}$ matrix.

Additionally, in RL algorithms, rewards are typically bounded within certain levels. It is necessary to define the maximum and minimum rewards granted for each action of every type. For instance, discovering a new cell generates an interest gradient of 255, resulting in a net reward of 1, while a gradient of 0 leads to a penalty of -0.5. These values, along with the $-C_{illegal}$ parameter, significantly influence the algorithm's scope. Therefore, it would be worthwhile to assess their impact by varying them within certain ranges.

Reward law	
$\mathbf{S} = \begin{cases} \mathbf{S}(i,j)^{t+1} = min[\mathbf{S}(i,j)^t + \delta, 1] & \text{if } [x,y]_{agent} \neq [i,j] \\ \mathbf{S}(i,j)^{t+1} = 0 & \text{if } [x,y]_{agent} = [i,j] \end{cases}$ $\mathbf{R} = \mathbf{S} \circ \mathbf{R}_{abs}$	
Movement	**Reward**
To illegal cell	$-C_{illegal}$
To unvisited cell	r_{max}
To latest visited cell	r_{min}
From position p to whatever p'	$\rho = \frac{r_{max} - r_{min}}{R_{max} - R_{min}}[\nabla\mathbf{R}(p,p') - R_{min}] + R_{max}$

Table 4.1: Summary for reward law.

4.3. Training parameters and evaluation metrics

In the realm of RL, particularly within the domain of DRL, the performance of algorithms is significantly shaped by hyperparameters. Achieving optimal results necessitates a meticulous exploration of values that enhance the training process. This study undertakes a systematic variation of each parameter with respect to a nominal value to discern improvements or deteriorations resulting from it. The nominal values are derived from typical values found in relevant literature, specifically those utilized in [9] and [19].

The critical calibration parameters include:

- **Learning rate**: Extent of the gradient incorporated into the neural network weights (denoted by α in Equation 3.9).
- **Batch size**: Number of transitions from s to s' extracted from memory replay to train the network.
- **ϵ-decay**: Reduction in ϵ with each episode. A higher value implies reduced exploration and extended exploitation time, and otherwise with lower values.
- **Number of neurons**: Quantity of convolutional filters and neurons in the implemented neural network structures, crucial for estimating the Q functions of both the model and the target.
- **Reward parameters**: Comprising values for $C_{illegal}$, r_{max}, and r_{min}, these

parameters specify the penalty for illegal actions, adherence to a low-interest gradient, and the exploration of an unvisited cell, respectively.

The nominal values serve as a baseline for comparison against each modification:

Hyperparameter	Value
Replay Memory Size	1000
Batch Size	256
Learning Rate	1e-3
ϵ-decay	7e-4
γ	0.99
Number of episodes	1500
Movements per episode	300

Table 4.2: Nominal hyperparameters.

To assess the performance of the algorithm, two distinct metrics are defined:

1. **Coverage**: Quantifies the number of newly visited cells relative to the total number of visitable cells on the map. Achieving 100% coverage aligns with the goal of Complete Coverage.

$$\kappa = \frac{\sum_{\forall(i,j)visited}[\mathbf{R}_{abs}(i,j)]}{\sum \mathbf{R}_{abs}} \tag{4.5}$$

   where κ is expressed on a per unit base. In the scenario of homogeneous coverage, where each position of $\mathbf{R}_{abs}$ shares the same value, it simplifies to $\kappa = \frac{m}{N}$ with m denoting the number of visited cells and N representing the total number of cells.

2. **Average effective visit time**: Calculates the average time each cell takes to receive a visit, factoring in the weighted coverage area. The average visit time for each cell is computed as the mean of the waiting times in that specific position:

$$\tau_k^{(i,j)} = t_k^{(i,j)} - t_{k-1}^{(i,j)} \quad \longrightarrow \quad \mu_\tau^{(i,j)} = \frac{1}{n^{(i,j)}} \sum_{k=0}^{n} \tau_k^{(i,j)} \tag{4.6}$$

   with $n^{(i,j)}$ indicating the number of visits to that cell.

Ultimately, the average effective time is determined as the mean of the average waiting times for each visited cell, weighted by its importance level and evaluated based on its covered area.

$$T_{mean} = \frac{1}{m} \sum_{\forall(i,j)} \left[\mu_{\tau}^{(i,j)} \frac{\mathbf{R}_{abs}(i,j)}{max\{\mathbf{R}_{abs}\}} \right] \quad \longrightarrow \quad T_{mean}^{eff} = \frac{T_{mean}}{\kappa} \tag{4.7}$$

5 | Results with Deep Reinforcement Learning

5.1. Results with Deep Q-Learning with homogeneous relevance

The objective is to train the agent to maximize the coverage area while minimizing the average time between visits to cells. The training process involves exploring various network architectures and hyperparameter values, along with different ways of interpreting the scenario's state. In cases of homogeneous coverage, the absolute importance matrix remains constant, as depicted in Figure 5.1. The desired outcome is to achieve maximum coverage with minimal values for the performance metrics κ and T_{eff}.

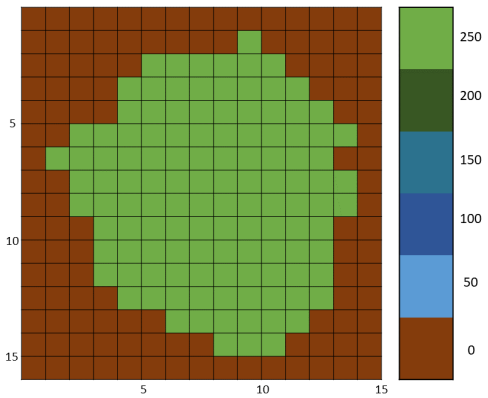


Figure 5.1: Map of absolute importance in the problem of homogeneous coverage.

5.1.1. Outcomes for different state representations

As mentioned in Chapter 3, the state defines how the agent perceives the environment. Initially, two possible states are defined for this problem, the latest providing

a higher level of information than the previous one. It will be demonstrated that the more information the state can capture, the better the final performance:

- s is the agent's position represented as the vector (i, j) within the map.
- s is an RGB image of the map distinguishing between unvisited cells (black), agent's position (red), impassable terrain (green), and visited cells within a grayscale based on its corresponding value in the relative weighting matrix $\mathbf{R}$.

Training with s as position

The neural network employed to estimate both $\mathcal{Q}$ and $\mathcal{Q}_{target}$ will be:

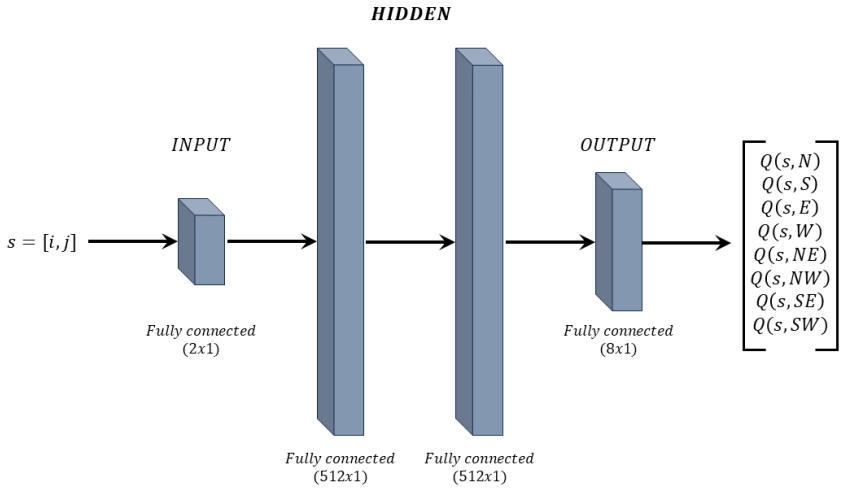


Figure 5.2: Neural network with agent position as input.

It can be observed in Figure 5.3 that the training is unsatisfactory, as evidenced by the attainment of low performance in the later epochs. The rewards peak is below 0, and a maximum of about 55 new cells are explored. Running a 300-move cycle to assess the exploration of Mount Etna over time produces results that are mediocre yet anticipated. The average coverage for 20 games falls short of 40% of the total area, and there is a notable level of redundancy with the continual visitation of the same cells. Additionally, this outcome indicates the occurrence of simplistic loops in movement. The lack of information tends to generate loops aiming to maximize rewards, and these loops typically close once their size is sufficient for the renewed interest in $\mathbf{R}$.

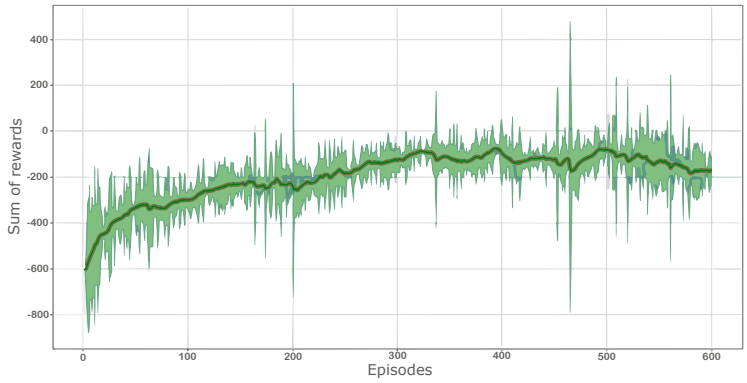


Figure 5.3: Results training with s as position.

By mapping the preferred direction at each position, it becomes apparent that the outcome is a vector map that exhibits a tendency to form closed loops with limited inclination towards overall exploration.

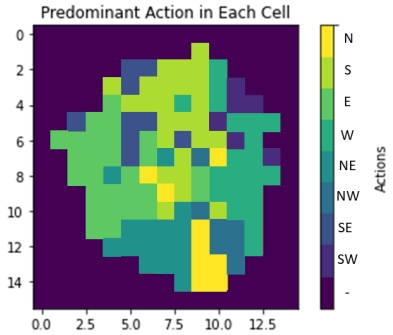
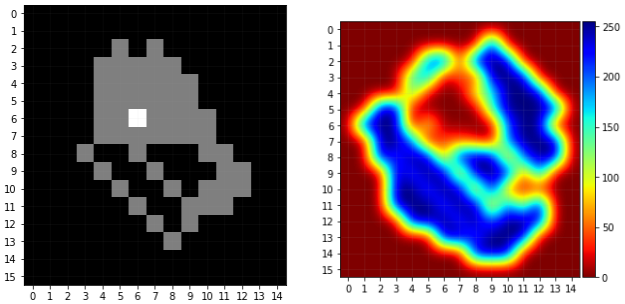


Figure 5.4: Map of preferred directions resulting from training.



(a) (b)

Figure 5.5: Result of best found test with 300 moves using training 1. In (a), the visited cells are shown in gray, and in (b), the matrix $\mathbf{R}$ is presented as an interpolated heat map.

Training with s as complete RGB

In this training, the algorithm is supplied with all the available information in the environment. The program is modified to accept an RGB image (3 channels) as the state, where four types of cells are represented: green for impassable areas, red for the agent's position, black for unvisited cells, and a grayscale range for visited cells indicating their relative interest value, i.e., $\mathbf{R}(i, j)$.

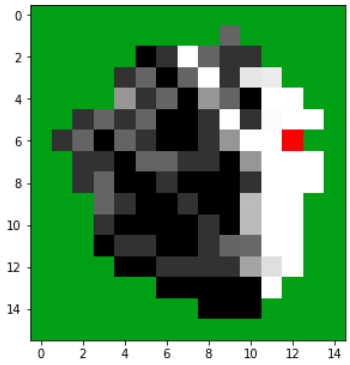


Figure 5.6: RGB image of the state considering the **R** matrix.

To adapt the program and allow the neural network to accept a 3-channel RGB matrix, the convolutional neural network in Figure 5.7 is implemented, serving as the reference in the comparison and sourced from [**piciarelli-2019**].

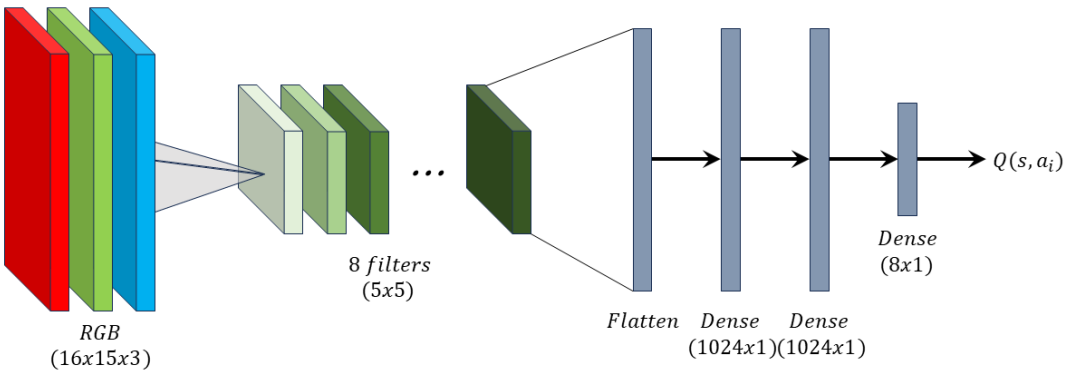


Figure 5.7: Convolutional neural network with RGB image of the scenario as input.

The training result is significantly improved, as the trend is increasing throughout the entire training until it declines at the end. Providing this new information to the algorithm evidently enhances the learning performance. The maximum reward

value reaches 64, and by conducting 20 sessions of 300 moves each, an average coverage of 62% of the total area is achieved.

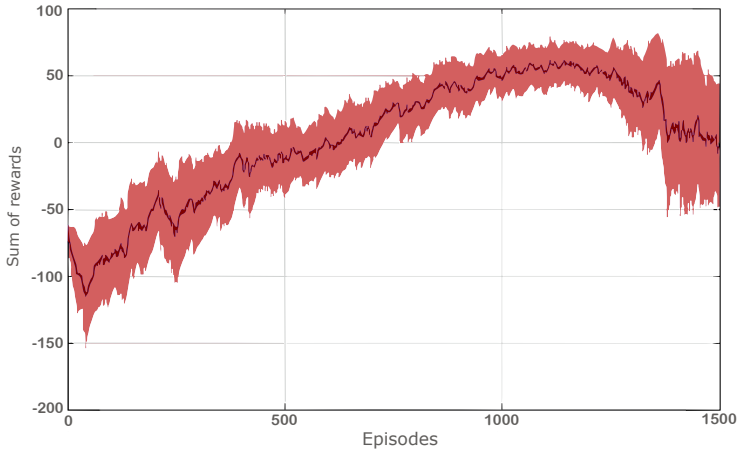
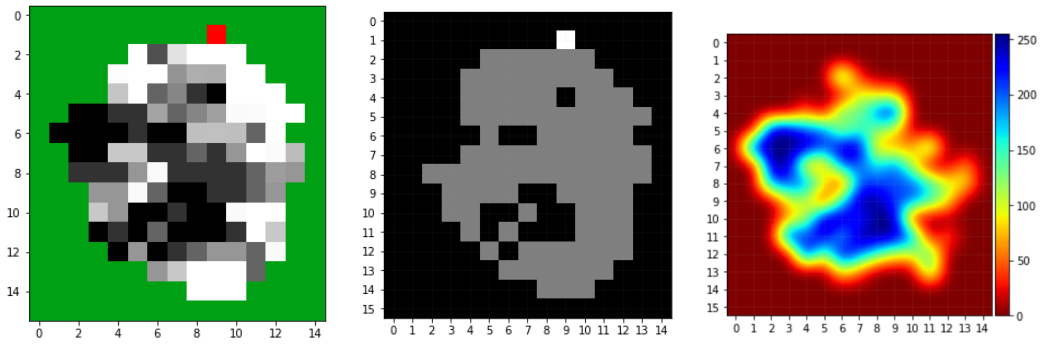


Figure 5.8: Results training with s as complete RGB.

Upon analyzing the outcome of a cycle of 300 possible moves with the resulting training model (the network returning the highest reward, not necessarily the last but close to the last episodes), it is evident that the coverage of Mount Etna is nearly complete:



(a) (b) (c)

Figure 5.9: Result of best found test with 300 moves using training 2. In (a), the complete RGB input map provided to the algorithm is shown, the visited cells are shown in gray in (b), and in (c), the matrix **R** is presented as an interpolated heat map.

With this methodology, 80% of the total area is visited. Additionally, the visitation to cells of interest is significantly more uniform than in the previous case, which

is logical considering that the algorithm now possesses a broader environmental knowledge and can better estimate the optimal value of $\mathcal{Q}$, i.e., a policy $\pi(s)$ closer to the optimum.

Comparison between s observation methods

It has been observed that as the algorithm receives more useful information in the state, the final performance of its task improves. The comparison of rewards can be verified in Figure 5.10. This image needs to be analyzed with caution. The learning outcome is similar in both cases, but when more information about previously traveled paths is available, such as when using the complete state of the map, the uniformity of coverage can also be enhanced.

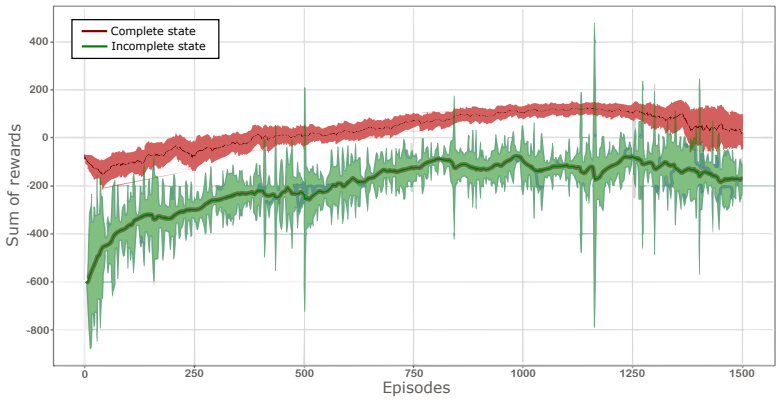


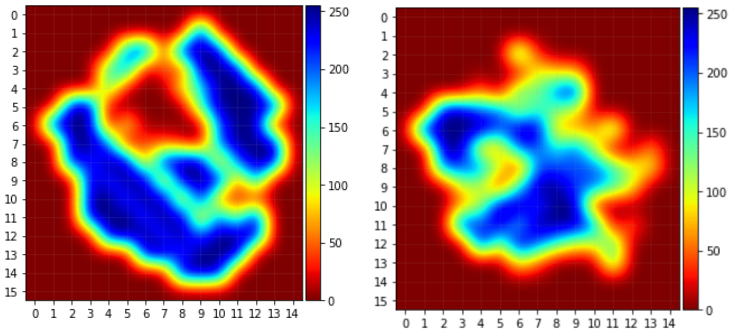
Figure 5.10: Comparison of training results with s as position and s as complete RGB.

One of the main reasons for this occurrence is that the Deep Q-Learning algorithm, as formulated, faces challenges in dealing with temporal dependencies when the provided state is not fully indicative of the actual state of the environment. With position alone, from one neural network evaluation to another given position, the algorithm finds it impossible to recall past actions. This underscores the idea that the more information the state provides, the better. After all, problem resolution through Q-Learning assumes having a fully observable Markov Decision Process (MPD), which is not the case with the first state form tested here.

Complete state method reaches stabilization later than in the case of position as state. Therefore, it will be a candidate for further testing (bearing in mind that tests can last for hours, and in the early stages of experimentation, this initial

screening can prevent lengthy and unproductive tests).

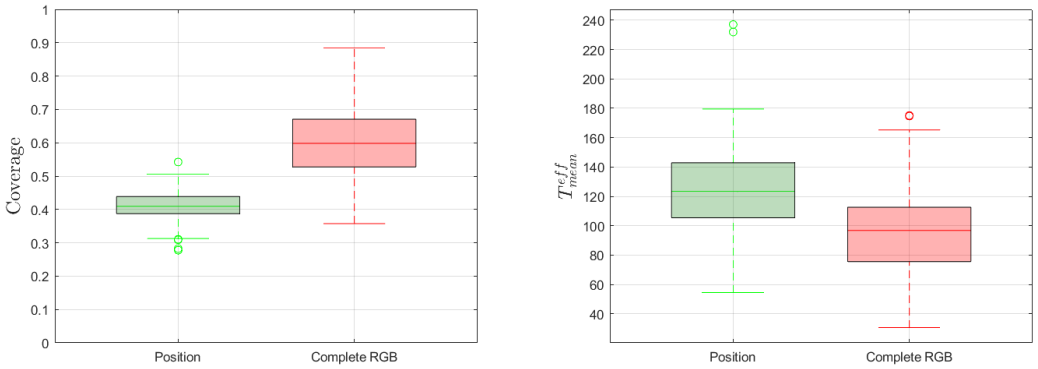
Figure 5.11 clearly shows the difference between both methods: not only does it improve the coverage goal but also the uniformity of coverage. It is equally important to cover the entire mount and do so uniformly, not repeatedly covering some areas and neglecting others. It is evident that the second method is the candidate for improvement and further training over an extended period, as one may encounter less blue zones in the heat map on the right side.



(a) (b)

Figure 5.11: Coverage comparison for the 2 possible inputs to DQN algorithm with 300 moves patrolling.

Lastly, the improvement in performance metrics is visible after conducting 20 tests for each algorithm.



(a) (b)

Figure 5.12: Figures of merit comparison for the 2 possible inputs to DQN algorithm with 300 moves patrolling.

5.1.2. Advantages and disadvantages of Deep Q-Learning

Deep Q-Learning algorithm yields positive results in terms of agent learning to systematically cover the entire visitable space. Episode after episode, the agent learns to take actions that maximize the reward. However, a common phenomenon in Deep Q-Learning, known as *catastrophic forgetting*, can be noted.

This phenomenon can be observed, for instance, in Figure 5.8: beyond a certain point, learned information is forgotten, the network starts to unlearn, and the average reward per episode drops rapidly. This phenomenon hinders the network from being trained for extended periods with increasingly better results since, sooner or later, it will forget what it took so much effort to learn.

Catastrophic forgetting typically occurs in the training of neural networks (and in Machine Learning in general) due to the plasticity of fixed-structure networks, such as the networks used here. The *backpropagation* mechanism employed to train the network can induce significant changes in neuron weights with any alterations in inputs, leading to instability in learning [25]. This characteristic, coupled with the *bootstrapping* approach used in approximating the Q-function (using the estimated Q-function to estimate $\mathcal{Q}_{target}$) in the DQN algorithm and the fact that it deals with an off-policy algorithm, forms a deadly triad that introduces divergences and instabilities in the training process.

5.2. Results with Deep Q-Learning with non-homogeneous relevance

Starting from the results of the study in Section 5.1.1, where the benefits of providing the complete RGB state have been observed, the learning outcomes are now examined for the case of non-homogeneous importance coverage, which aligns more closely with reality. For this purpose, a static interest map $\mathbf{R}_{abs}$ has been created, which will be incorporated into the reward policy with varying levels of interest depending on the geographic area (areas closer to the volcano crater or with higher infrastructure density will have greater relevance).

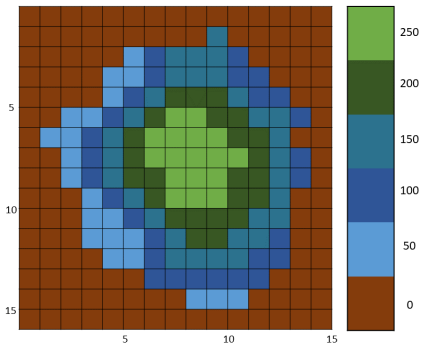


Figure 5.13: Map of absolute importance in the problem of non-homogeneous coverage.

In the training of this network, the phenomena of training instability and catastrophic forgetting are much more pronounced. This phenomenon leads to a very poor final outcome for the non-homogeneous case, prompting further experimentation with finer algorithms, i.e. Double Deep Q-Learning.

The impact of batch size on learning can also be observed. This demonstrates the dependence of the final performance on the training parameters. Training stability will be a trade-off between the effectiveness of the implemented algorithm (DQN, DDQN, ...) and the chosen hyperparameters along with the neural network architecture. As mentioned earlier, it will be shown that with the Double DQN algorithm, this phenomenon is mitigated for both training cases.

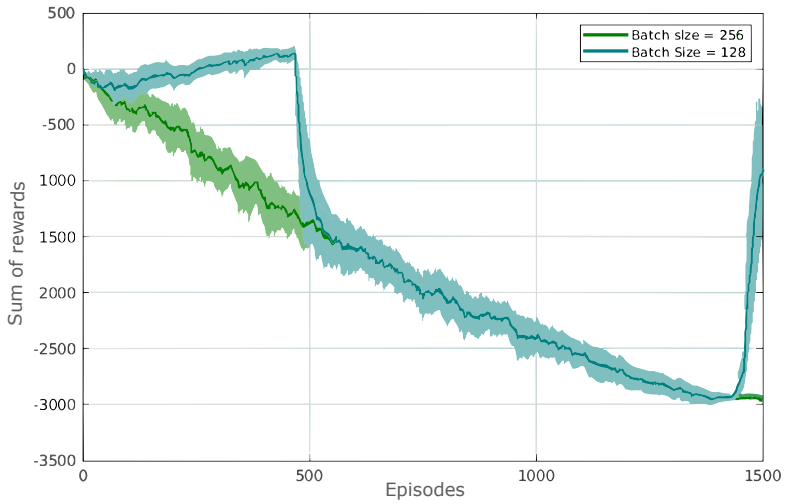


Figure 5.14: Training results for DQN in non-homogeneous case.

5.3. Results with Double Deep Q-Learning with non-homogeneous relevance

It operates under identical nominal conditions and utilizes the reward policy providing optimal results as outlined in Section 5.1.1. The Double DQN algorithm's performance for the non-homogeneous case is demonstrated in Figure 5.15, affirming the proposed hypothesis: duplicating the network for the target function enhances training stability and yields superior outcomes in a general scenario.

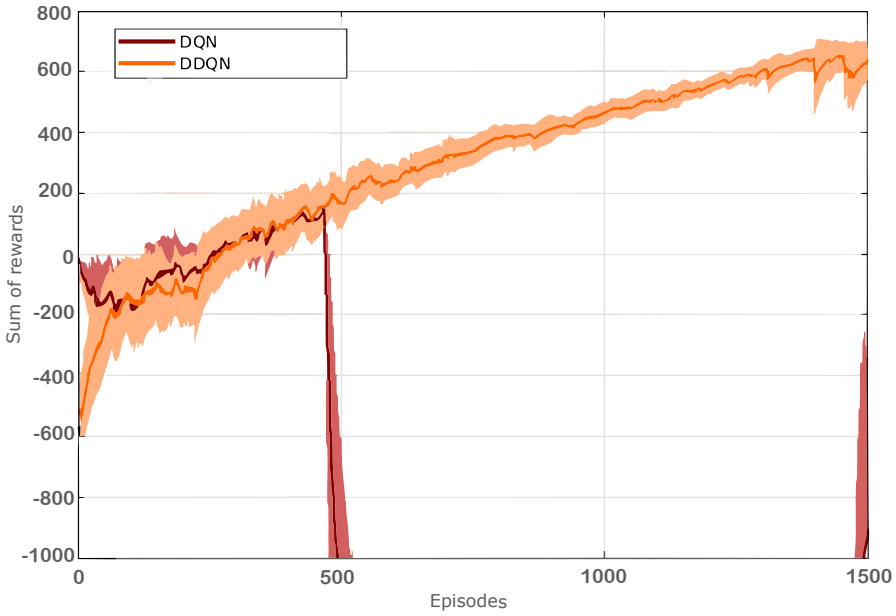
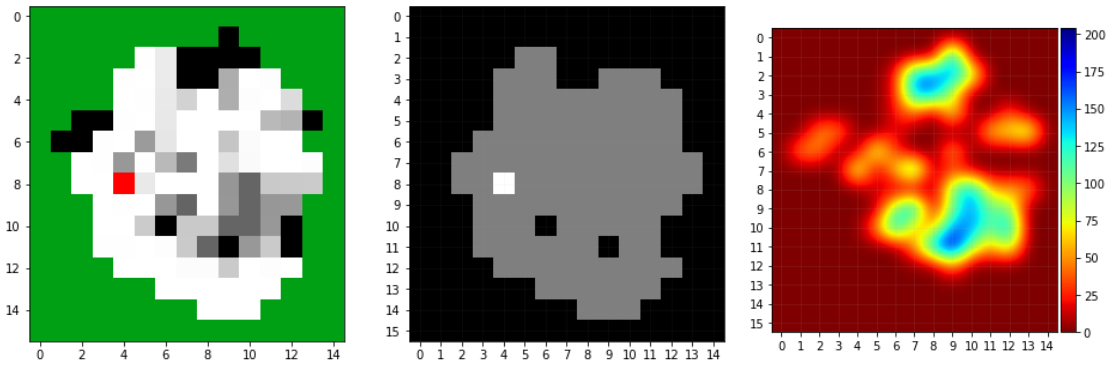


Figure 5.15: Comparison for training results between DQN and DDQN algorithms.

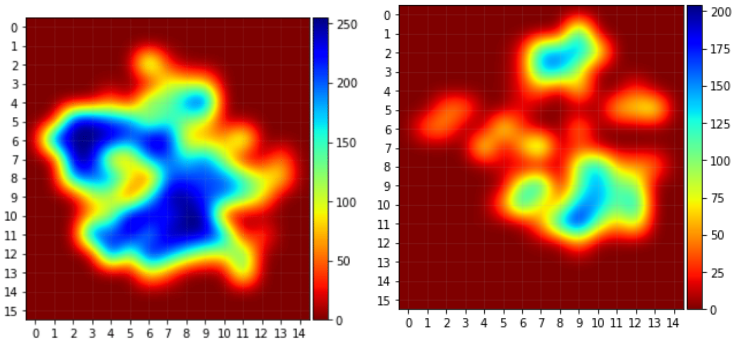
The simple DQN algorithm once again proves to be unstable during training with catastrophic forgetting being notorious after 500 episodes, even more than it was in the homogeneous case. On the contrary, the accumulative reward for DDQN reaches approximately 700 in the final stages, with a more or less monotonous trend of improvement. Given this behavior, the Double DQN algorithm emerges as a potential candidate for extended training, even though adjusting hyperparameters might result in higher rewards.



(a) (b) (c)

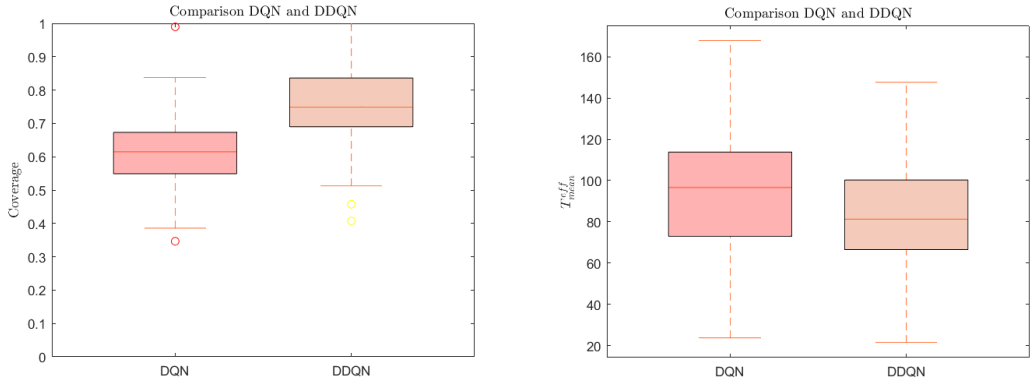
Figure 5.16: Result of best found test with 300 moves using training 3. In (a), the complete RGB input map provided to the algorithm is shown, the visited cells are shown in gray in (b), and in (c), the matrix **R** is presented as an interpolated heat map.

Figure 5.17 compares the agent's behavior using DQN and Double-DQN over an episode of 300 movements. Notably, regions of higher importance on the map receive more frequent visits, forming a learned behavior that tends to disregard less critical areas. This behavior is rationalized by the algorithm's inclination to achieve an acceptable solution by revisiting high-importance regions rather than exploring low-importance areas. Subsequently, the performance metrics from latest training are presented and contrasted with the outcomes from DQL algorithm.



(a) (b)

Figure 5.17: Comparison of heat maps obtained using DQN (a) and DDQN (b) algorithms.



(a) (b)

Figure 5.18: Figures of merit comparison for DQN and DDQN algorithms with 300 moves patrolling.

The results indicate that the ultimately trained algorithm outperforms a simple DQN network of 300 steps. The coverage reaches a maximum of 91%, and the effective mean time yields a minimum of 73 movements.

5.3.1. Advantages and disadvantages of Double Deep Q-Learning

Examining the advantages and disadvantages of Double Deep Q-Learning, it is evident that the algorithm effectively eliminates catastrophic forgetting in Mount Etna exploration. This advantage proves pivotal in achieving efficient learning, as satisfactory results are attained with 1500 episodes without compromising previously acquired knowledge. The implementation cost of the Double DQN algorithm is appreciable but not excessive, requiring only the duplication of the Q network into a target network. The latter is a non-trained network that periodically receives the trained weights, imposing negligible computational overhead and slightly increased memory usage.

In conclusion, Double-DQN emerges as a robust solution for the exploration of Mount Etna, demonstrating its efficacy within the specified training parameters.

5.4. Analysis of hyperparameters tuning

The pages above has highlighted that not only introducing a more complex RL algorithm will have comparative advantages in the obtained results, but also the value of the training neural network's hyperparameters will significantly influence these results. Therefore, this section is primarily focused on contrasting to what extent the adjustment of hyperparameters introduces substantial variability in the autonomous behavior of the drone. Once the superiority of DDQN over its simple-type algorithm has been demonstrated, the different parametric studies conducted from this point onward will be applied to the case of a non-homogeneous relevance map, varying a single parameter while keeping the others constant.

5.4.1. Batch size

The neural network has been trained, starting from the nominal values and imposing different values for the batch size: 32, 64, 128, 256 (nominal) and 512.

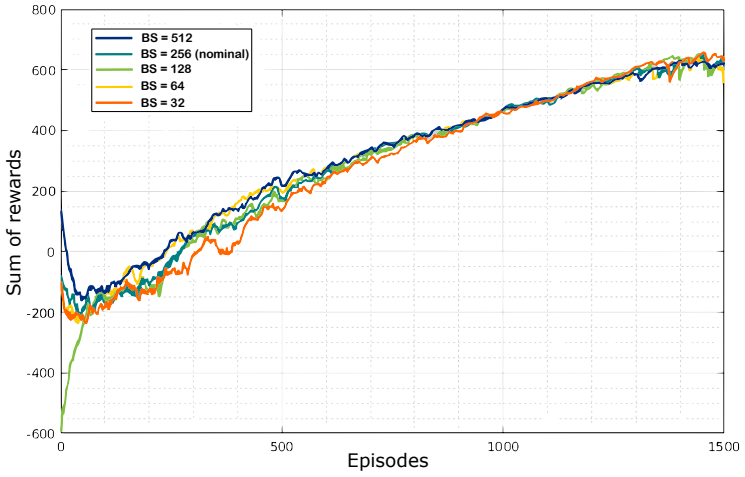


Figure 5.19: Training results for various batch sizes.

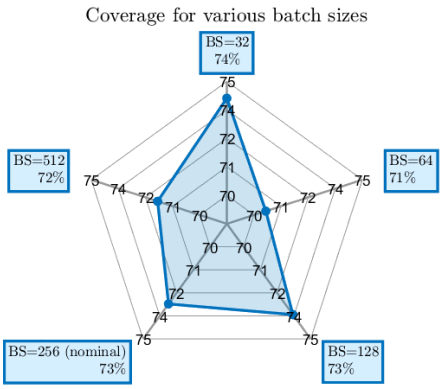


Figure 5.20: Mean coverage of 20 episodes for various batch sizes.

The obtained outcome goes in line with the findings presented in [26], where, in contrast to common intuition, an increased batch size does not necessarily lead to enhanced agent learning. The learning process is intricately linked to the ratio between the number of experiences sampled at each training iteration (batch size) and the capacity of the memory replay. As indicated by the conclusions drawn in [26], a high ratio tends to align with an on-policy approach, resulting in diminished training efficacy and augmented computational expenses. Conversely, an excessively low ratio implies that the training process encounters a scarcity of associations (s, a, s', r) during each iteration, thereby yielding suboptimal performance.

5.4.2. Learning rate

The learning rate plays a crucial role in determining the speed at which the gradient of the cost function J is applied to each parameter of the neural network. An excessively large value for this parameter may introduce instabilities during training, while an overly small value can result in exceptionally slow convergence, possibly leading to a suboptimal solution. In this context, four different values of the learning rate have been experimented with: 10^{-1}, 10^{-2}, 10^{-3} (nominal) and 10^{-4}.

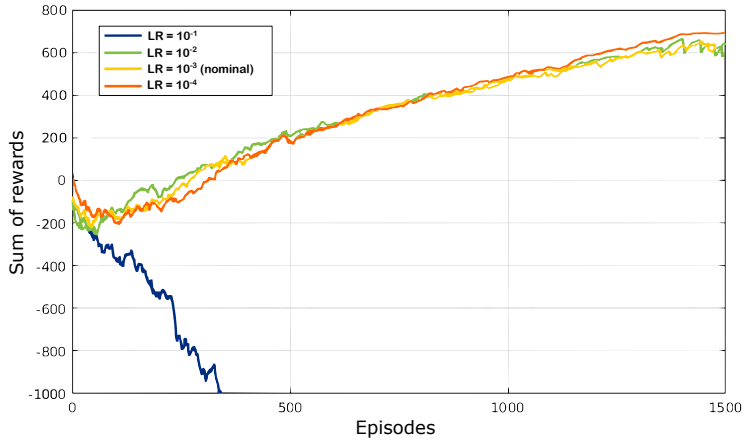


Figure 5.21: Training results for various learning rates.

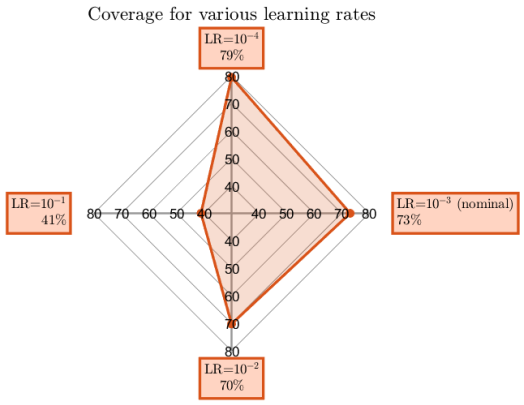


Figure 5.22: Mean coverage of 20 episodes for various learning rates.

It is observed that for a learning rate value that is excessively large, the training becomes unstable, indicating the need to keep it below the nominal value.

5.4.3. ϵ-decay

The decay value of for ϵ determines the level of greediness in the behavior policy according to the epsilon-greedy algorithm. Three decay values have been tested, each causing ϵ to reach its minimum (0.01) over a complete set of episodes. These three values are: 0.002 (ϵ reaches its minimum in episode 500), 0.001 (ϵ reaches its minimum in episode 1000), and 0.0007 (ϵ reaches its minimum in episode 1400).

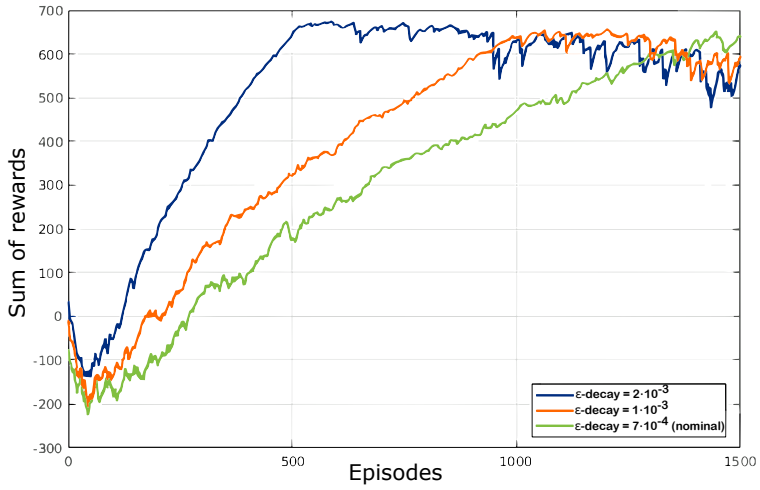


Figure 5.23: Training results for various ϵ-decays.

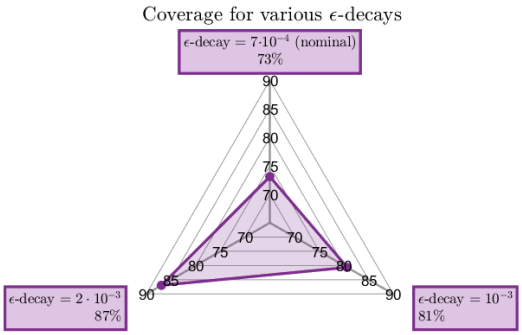


Figure 5.24: Mean coverage of 20 episodes for various ϵ-decays.

It can be observed that the smaller the value of ϵ-decay, the earlier the curve stabilizes. A very high value could result in a lack of exploration, leading to a failure to generalize learning due to a lack of previous experiences. A very small value leads to very slow but more stable training, as it explores many more actions in search of the optimal Q policy. Nonetheless, in this specific application, it is noted that nominal training can be improved (in terms of speed and performance) by increasing the value of ϵ-decay.

5.4.4. Neural network size

Experimentation has been conducted by increasing and decreasing the size of the neural network while maintaining its design structure to observe how the quantity of neurons and convolutional filters affects training. An excessively large network will have many neurons with weights that tend toward 0, while a very small network lacks plasticity to generalize the optimal Q function. Therefore, it should be adjusted to an intermediate value.

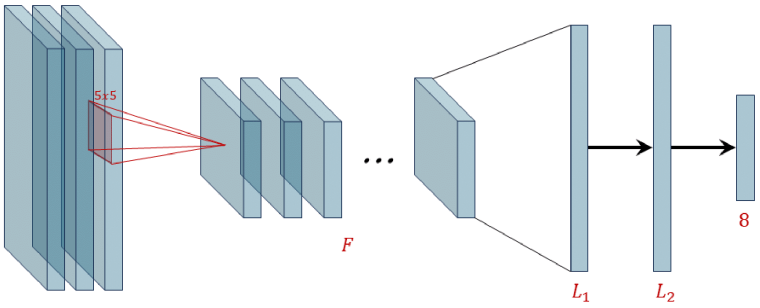


Figure 5.25: Neural network structure and size (F, L_1, L_2).

Size	Number of filters F	Linear layer neurons L_1	Linear layer neurons L_2
x2	16	2048	2048
x1 (nom.)	8	1024	1024
0.5	4	512	512
x0.25	2	256	256
x0.1	1	128	128

Table 5.1: Trained neural network sizes.

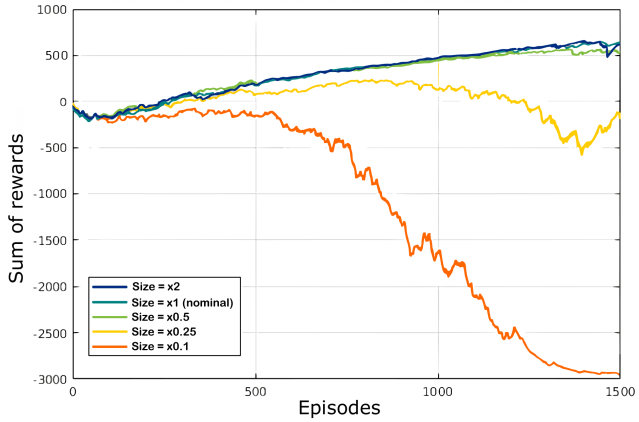


Figure 5.26: Training results for various neural network sizes.

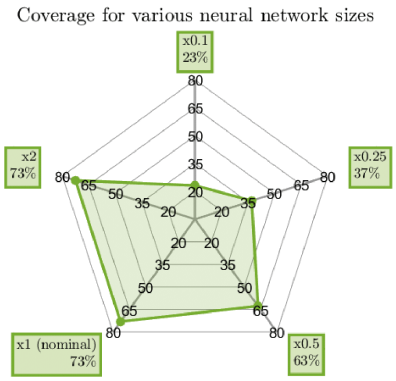
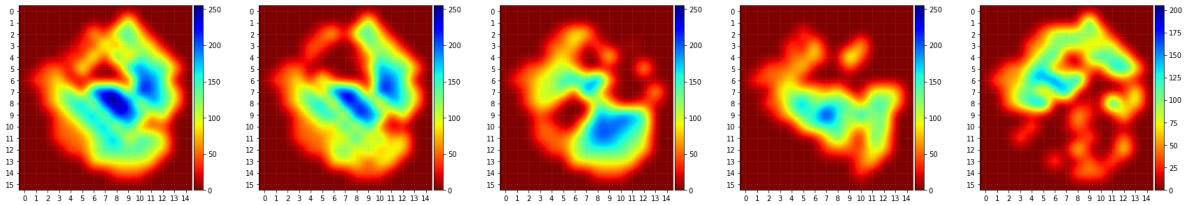


Figure 5.27: Mean coverage of 20 episodes for various neural network sizes.

It is observed that satisfactory results are obtained with the nominal network size. A reduction in size leads to instability in the network training, while an increase in size does not yield superior results. Hence, it is concluded that the initially proposed structure is adequate for achieving the desired outcomes.

The coverage map for the different neural network sizes will be represented in this case, where a higher variability in outcomes is obtained.



(a) Size = x0.1 (b) Size = x0.25 (c) Size = x0.5 (d) Size = x1 (e) Size = x2

Figure 5.28: Comparison of heat maps obtained using DDQN with various neural network sizes.

Once the hyperparameters tuning study has been performed, the values that yielded superior results in terms of coverage could be considered simultaneously. This way the heat map obtained from a simulation close to the optimal one would be as the one shown in Figure 5.29, with $\kappa = 90\%$ and $T_{mean}^{eff} = 83$ movements.

Hyperparameter	Value
Replay Memory Size	1000
Batch Size	32
Learning Rate	1e-4
ϵ-decay	2e-3
γ	0.99

Table 5.2: Tuned hyperparameters.

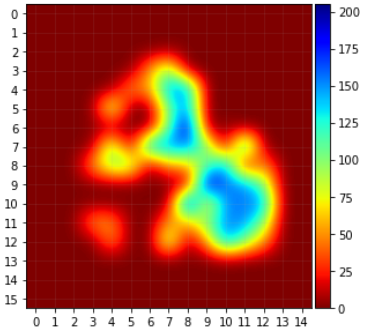


Figure 5.29: Heat map obtained using DDQN algorithm with close to optimal hyperparameters.

5.5. Application to avalanche problem

Using Double Deep Q Learning for path planning of a UAV during patrolling has proven to be highly effective and adaptable in the environmental context of Mount Etna in the previous sections. This methodology, indeed, was initially developed for the challenging terrain of Mount Etna, where volcanic eruptions pose a constant threat. The algorithms created have demonstrated their utility and robustness with high values of effective coverage and mean effective time averaged over an important number of patrolling problem samples. However, the applicability of this methodology extends beyond volcanic landscapes to regions prone to other natural disasters, such as floods, hurricanes, fires and so on.

In this section, the methodology developed as the subject of the Thesis will be applied to another ecosystem of diverse nature in which avalanches are natural

disasters with a high probability of occurrence.

A region with a history of avalanches and suitable terrain for UAV patrolling simulations could be the Swiss Alps. The Swiss Alps experience frequent avalanches due to their steep slopes, high altitude, and heavy snowfall. This makes them an ideal location for studying avalanche dynamics and assessing the effectiveness of UAV patrols in mitigating risks.

Switzerland has a well-established network of avalanche research institutes and monitoring stations that provide valuable data on avalanche activity, snowpack conditions, and terrain characteristics. This data can be integrated into simulation environments to enhance realism. In fact, the map that will be used for the development of simulations in this context comes from a Swiss security institution, identifying mountainous areas where avalanche probabilities are higher or lower.

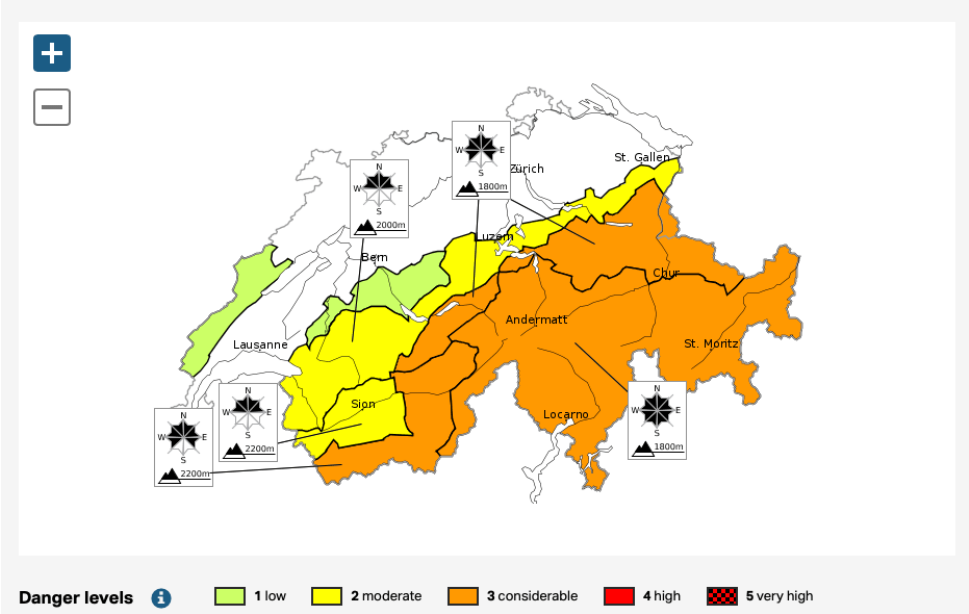


Figure 5.30: Map for Swiss Alps.

Just as was done for Mount Etna, the map undergoes a process of pixels extraction that are of interest for research through threshold filtering.

(a) Parameters. (b) Mask. (c) Superposition.

Figure 5.31: Gridmap generation program interface for Swiss Alps case.

The only added complexity encountered here is that one of the areas of interest during avalanche rescue searches is completely disconnected from the others, which represents a discontinuity in the treatment of the problem. This is a problem that is currently unresolved in the developed algorithms, and as a workaround solution, it is chosen to consider that the cells that would connect both areas of the map are also included in it with minimal interest in the weighting matrix.

All in all, the Swiss Alps, renowned for their majestic beauty, also present unique challenges due to the risk of avalanches, especially in the winter months. Patrolling these areas effectively is crucial for early detection of avalanche risks and ensuring the safety of residents and tourists. Traditional patrolling methods may be limited in their coverage and efficiency, particularly in vast and rugged terrains. Here is where the adaptability and effectiveness of Double Deep Q Learning shine.

The algorithm's ability to learn and adapt to dynamic environments makes it well-suited for patrolling tasks in avalanche-prone areas. By leveraging historical data and real-time feedback, the UAV can navigate the terrain, identify potential risks, and optimize its patrol route to maximize coverage and minimize response time. Moreover, the algorithm's Reinforcement Learning framework enables it to continuously improve its decision-making process based on past experiences, ensuring more efficient and informed patrolling over time.

Furthermore, demonstrating the efficacy of Double Deep Q Learning in diverse en-

vironments like the Swiss Alps and Mount Etna serves as a descriptive validation of its versatility and generalizability. It underscores that the algorithm's success in one specific context, such as Mount Etna, is not merely coincidental but rather indicative of its broader applicability across different geographical and environmental conditions.

Based on the findings presented in previous sections, which highlighted the advantages of utilizing the full RGB state and scenarios featuring non-uniform importance coverage, mirroring real-world conditions more closely, the focus now shifts to evaluating the learning outcomes in the context of the Swiss Alps. To achieve this, an equivalent static interest map, denoted as Rabs, has been formulated for this case study. This map will be integrated into the reward policy, assigning different interest levels based on geographical factors (differentiation of areas by color according to the probability of avalanche occurrence), thereby reflecting varying degrees of significance.

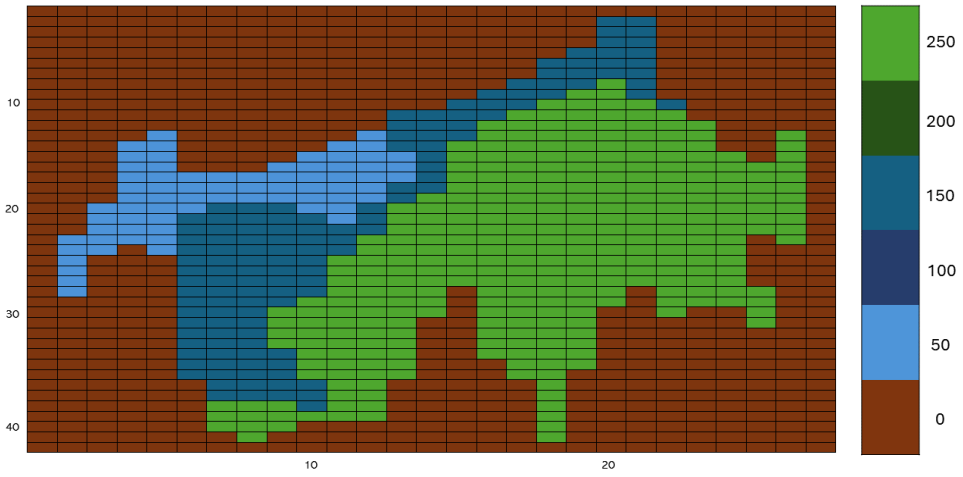
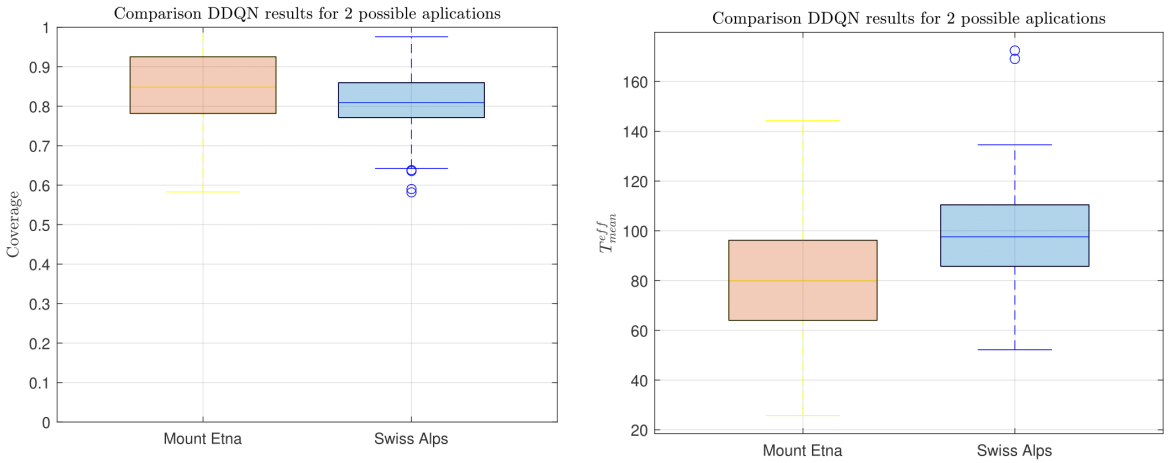


Figure 5.32: Map of absolute importance in non-homogeneous coverage of Swiss Alps.

Since a comprehensive analysis has previously been conducted on the effect of modifying hyperparameters on the problem's performance metrics, those values that lead to results close to optimal for the case of Mount Etna will be utilized. The aim is to ascertain whether the methodology employed in the Thesis is robust and widely applicable, with the hyperparameters listed in Table 5.2 not only being valid for the case from which they were extracted but also yielding similar results for other maps of interest.

Figure 5.33 compares the figures of merit from agent's behavior using DDQN over a sample of 500 episodes with 300 movements each within the scenario of Mount Etna, applicable for volcanic eruptions rescue missions, and Swiss alps, valid for avalanche rescue operations.



(a) (b)

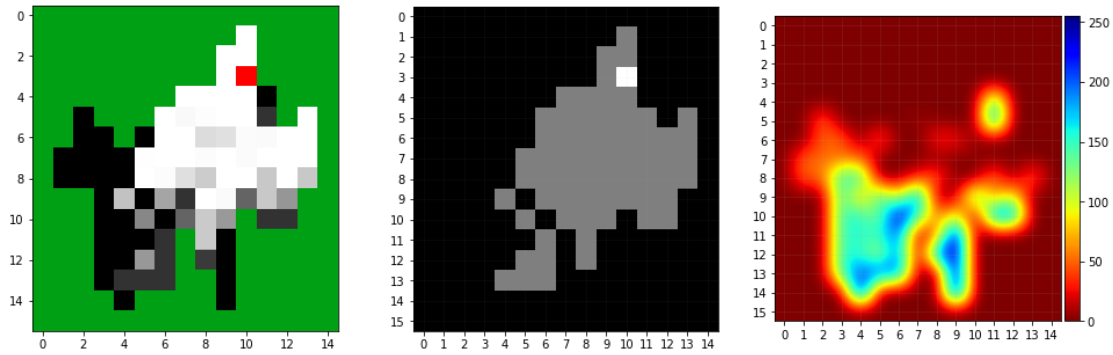
Figure 5.33: Figures of merit comparison for DDQN algorithm applied to Mount Etna and Swiss Alps scenarios.

The results presented demonstrate how, indeed, both the coverage of the area of interest and the mean effective cell visitation time achieve similar values in both cases. Specifically, κ takes on an 82% for the nominal problem for which the tuned hyperparameters were calculated, while the decrease is negligible in this regard for the Swiss Alps, with $\kappa = 80\%$, maintaining a more than significant coverage. The variation is slightly more notable when discussing the mean effective visitation time as the increase in this figure is 15%. However, it should be understood that the hyperparameters were optimized based on coverage without considering T_{eff}^{mean}, hence it is understandable that this performance metric is subject to greater variability. Nevertheless, if an improvement in both κ and T_{eff}^{mean} is desired, the parametric analysis would need to be extended to a multiparametric study with the complexity that entails.

In any case, the results obtained by applying the designed UAV patrolling algorithm based on Double Deep Q Network to another scenario of different typology

than the one initially applied are equally more than satisfactory; one could say, very similar to what was expected. With all this, the proposed methodology demonstrates its robustness in the face of environmental variation and allows us to assume that it should be valid for any other environment in which a rescue mission is necessary, regardless of the type of environment or natural disaster that triggers the deployment of operations.

As a final conclusion, the heat map of the patrolling episode for which the best coverage is achieved for the Swiss Alps map is attached, reaching a maximum of $\kappa = 85\%$.



(a) (b) (c)

Figure 5.34: Result of best found test with 300 moves for Swiss Alps scenario. In (a), the complete RGB input map provided to the algorithm is shown, the visited cells are shown in gray in (b), and in (c), the matrix $\mathbf{R}$ is presented as an interpolated heat map.

6 | Conclusions and future developments

As a final **conclusion** to this Thesis, it has been observed that the Reinforcement Learning methodology, specifically Deep Reinforcement Learning, thanks to its versatility allows training in tasks that, until now, could only be addressed with a complex model and significant computational power. The merits of these techniques emerge in UAV guidance, where there are tasks with complex solutions involving large action spaces and practically infinite states: the Deep Q-Learning algorithm can directly work with past experience without the need to model the environment. Furthermore, being an *off-policy* algorithm, it can find the optimal strategy regardless of the behavior policy used to explore the state-action space.

The study conducted in this Thesis has been applied to tracking Mount Etna during volcanic eruptions. However, the presented methodology is applicable to any other imagined environment. Just as the goal of environmental adaptability has been achieved, comprehensive area coverage, temporal redundancy, and parametrization have also been accomplished.

Indeed, the flexibility and adaptability of the methodology stand out as key strengths, enabling its seamless integration into diverse settings beyond volcanic monitoring. The robustness of the approach is exemplified by its ability to effectively address complex challenges inherent in tracking dynamic phenomena such as volcanic activity. Moreover, the methodology's scalability ensures its suitability for environments with varying degrees of map shape, complexity and uncertainty.

One of the notable achievements of this study lies in its ability to achieve comprehensive area coverage, a critical aspect in scenarios requiring thorough surveillance

and monitoring. By employing advanced reinforcement learning techniques, the developed methodology excels in optimizing resource allocation and maximizing coverage efficiency, thereby ensuring comprehensive monitoring of the target area.

Furthermore, the incorporation of temporal redundancy enhances the reliability and resilience of the monitoring system, mitigating the impact of uncertainties and disruptions. This temporal redundancy not only contributes to the robustness of the methodology but also enables continuous monitoring and adaptive decision-making in dynamic environments.

Parametrization emerges as another significant accomplishment of this research, allowing for fine-tuning and optimization of the methodology to suit specific environmental conditions and operational requirements. The parametrization process involves careful calibration of various algorithmic parameters to achieve optimal performance, thereby enhancing the efficacy and efficiency of the monitoring system.

As for the methodology followed, the progressive increase in the complexity of the addressed problem has revealed that the DQN algorithm is susceptible to the phenomenon of *catastrophic forgetting*, where prior learning becomes somewhat useless going forward. This was particularly evident when the relevance map with a heterogeneous matrix was introduced, justifying the introduction of a methodology that, although slightly more computationally costly, yields more robust and beneficial results. Thus, the performance metrics used for evaluating the effectiveness of the methods show a notable improvement from DQN with position as the input state vector ($\kappa = 42\%$ and $T^{eff}_{mean} = 124$) to the latest proposed implementation with DDQN with s as a complete RGB ($\kappa = 73\%$ and $T^{eff}_{mean} = 81$).

Additionally, it has been demonstrated that DQN and DDQN algorithms heavily depend on the values of the hyperparameters used. In this regard, a brief tuning of the DDQN algorithm has led to significantly improved results, prompting future research to dedicate time to finding a suitable set of hyperparameters for each case.

In essence, while the primary focus of this Thesis may have been on tracking Mount

Etna during volcanic eruptions, its broader implications extend far beyond this specific application. The methodology presented herein embodies a versatile and adaptable approach to environmental monitoring and surveillance, with potential applications spanning diverse domains such as disaster response, environmental conservation, and urban planning.

In this sense, by applying the developed methodology to a scenario different from the one initially proposed, such as the Swiss Alps, where search and rescue operations after avalanches are commonplace, its robustness has been demonstrated. The designed algorithm has proven to be not only a useful tool for a single environment focused on post-volcanic eruption rescue, but also applicable to many other situations where UAVs can perform patrolling operations. This is a crucial aspect when testing the algorithm, as it would make little sense to develop a method useful for only one real-world problem. The final part in the results section thus demonstrates the wide range of study possibilities that open up after initiating this Thesis, with a possible future line of research being the study of a larger number of scenarios, with diverse maps in terms of typology and shape, to improve the algorithm in areas that can be enhanced and consolidate the observed performance metrics.

As for potential avenues for **future research**, the obtained results forecast a whole new range of exploration regarding Reinforcement Learning algorithms. New paths are opening up in this field. Policy Optimization algorithms like Advantage Actors Critic (A2C) and Asynchronous Advantage Actors Critic (A3C) stand out as promising candidates, focusing on finding optimal policies rather than Q-function values. Particularly in multi-agent scenarios, where joint exploration is crucial and collisions must be considered, these algorithms show remarkable efficiency. This represents a fundamental area for improvement, especially in real-world applications such as drone fleet operations, where optimized coverage results are essential.

Another important avenue is the inclusion of concurrent neural networks to enhance final performance. Concurrent neural networks allow feedback from the neurons' output to detect and integrate temporal dependencies from one movement to another. Besides, the inclusion of uncertainty levels originating from sensors in the

learning process might also be considered, which should be reflected in a new form of the heat map. The uncertainties of the scenario necessarily transform the problem into a non-deterministic decision-making process with temporal observation dependencies.

The last step for the real-world applicability of the entire proposed methodology would be to integrate the two algorithms (homogeneous and non-homogeneous coverage) into the UAV. A control scheme (see Figure 6.1) is proposed where, utilizing sensor data and algorithms, a map detailing relative importance is generated, prioritizing areas with higher pedestrian probability. In subsequent iterations, the non-homogeneous planning algorithm is employed, facilitating real-time monitoring of the environment by emphasizing areas of greater importance detected during each search.

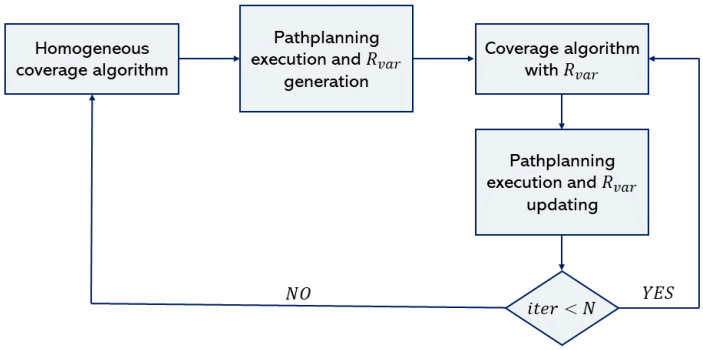


Figure 6.1: Proposed algorithm for implementation in a real-world application.

Bibliography

[1] S. Zhao, K. Ota, and M. Dong. "UAV Base Station Trajectory Optimization Based on Reinforcement Learning in Post-disaster Search and Rescue Operations". In: *Journal of Disaster Management* 11.2 (2022), pp. 45–60. URL: http://arxiv.org/abs/2202.10338.

[2] O. Aichholzer et al. "Scheduling Drones to Cover Outdoor Events". In: *Proceedings of the European Workshop on Computational Geometry (EuroCG)*. Mar. 2020. URL: http://www.example.com/proceedings/eurocg2020.

[3] Wingtra AG. *WingtraOne Technical Specifications*. n.d. URL: https://wingtra.com/mapping-drone-wingtraone/technical-specifications/.

[4] IASbaba. *Day 49 - Q 1. What is machine learning? What are its applications? - TLP-IASbaba*. Jan. 2019. URL: https://tlp.iasbaba.com/2019/01/day-49-q-1-what-is-machine-learning-what-are-its-applications/.

[5] S. Yanes Luis, D. Gutiérrez Reina, and S. Toral. "A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The Ypacaraí Lake Patrolling Case". In: *IEEE Access* 9 (2021). DOI: 10.1109/access.2021.3053348. URL: https://doi.org/10.1109/access.2021.3053348.

[6] Z. Shen, J. Wilson, and S. Gupta. "$\epsilon^\star$+: An Online Coverage Path Planning Algorithm for Energy-constrained Autonomous Vehicles". In: *Global Oceans 2020: Singapore – U.S. Gulf Coast*. Oct. 2020. DOI: 10.1109/ieeeconf38699.2020.9389353. URL: https://doi.org/10.1109/ieeeconf38699.2020.9389353.

[7] A. K. Lakshmanan et al. "Complete coverage path planning using reinforcement learning for Tetromino based cleaning and maintenance robot". In: *Automation in Construction* 112 (2020). DOI: 10.1016/j.autcon.2020.103078. URL: https://doi.org/10.1016/j.autcon.2020.103078.

[8] M. Theile et al. "UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2020. DOI: 10.1109/iros45743.2020.9340934. URL: https://doi.org/10.1109/iros45743.2020.9340934.

[9] C. Piciarelli and G. L. Foresti. "Drone patrolling with reinforcement learning". In: *13th ACM/IEEE International Conference on Distributed Smart Cameras*. Sept. 2019. DOI: 10.1145/3349801.3349805. URL: https://doi.org/10.1145/3349801.3349805.

[10] Amazon. *Amazon Prime Air's First Customer Delivery*. 2016. URL: https://www.youtube.com/watch?v=vNySOrI2Ny8.

[11] J. Stewart. *Google Tests Drone Deliveries in Project Wing Trials*. 2014. URL: https://www.bbc.com/news/technology-28964260.

[12] S. Levy. *How Google Will Use High-Flying Balloons to Deliver Internet to the Hinterlands*. 2013. URL: https://www.wired.com/2013/06/google-internet-balloons/.

[13] M. Mozaffari et al. "Drone Small Cells in the Clouds: Design, Deployment and Performance Analysis". In: *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2015, pp. 1–6.

[14] D. C. Schedl et al. "An autonomous drone for search and rescue in forests using airborne optical sectioning". In: *Science Robotics* 6 (2021), eabg1188. DOI: 10.1126/scirobotics.abg1188.

[15] T. D. Trong et al. "A Novelty Approach to Emulate Field Data Captured by Unmanned Aerial Vehicles for Training Deep Learning Algorithms Used for Search-and-Rescue Activities at Sea". In: *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*. 2021, pp. 288–293. DOI: 10.1109/ICCE48956.2021.9352109.

[16] J. G. Cárcamo Zuluaga et al. "Deep Reinforcement Learning for Autonomous Search and Rescue". In: *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. 2018, pp. 521–524. DOI: 10.1109/NAECON.2018.8556642.

[17] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. 1998. URL: http://portal.acm.org/citation.cfm?id=551283.

[18] A. T. Azar et al. "Drone Deep Reinforcement Learning: a review". In: *Electronics* 10.9 (2021), p. 999. DOI: 10.3390/electronics10090999. URL: https://doi.org/10.3390/electronics10090999.

[19] S. Bhatt. *Reinforcement Learning 101 - towards Data Science*. Apr. 2019. URL: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.

[20] Wikipedia Contributors. *Proceso de decisión de Markov*. 2023. URL: https://es.wikipedia.org/wiki/Proceso_de_decisiondeMarkov.

[21] I. Sajedian, H. Lee, and J. Rho. "Double-Deep Q-Learning to Increase the Efficiency of Metasurface Holograms". In: *Scientific Reports* 9 (2019), p. 10899. DOI: 10.1038/s41598-019-47154-z.

[22] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[23] K. Xu, B. Song, and F. Sun. "A deep similarity metric method based on incomplete data for traffic anomaly detection in IoT". In: *Applied sciences* 9.1 (2019), p. 135. DOI: 10.3390/app9010135. URL: https://doi.org/10.3390/app9010135.

[24] H. Van Hasselt, A. Guez, and D. Silver. "Deep Reinforcement Learning with Double Q-Learning". In: 30^{th} *AAAI Conference on Artificial Intelligence*. 2016, pp. 2094–2100.

[25] A. Cahill. “Catastrophic Forgetting in Reinforcement-Learning Environments”. PhD thesis. University of Otago, 2010.

[26] W. Fedus et al. “Revisiting Fundamentals of Experience Replay”. In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

Appendix A

This appendix includes the Python code developed by the author of the Thesis, which has served as a basis for the development of variations thereof, with which to obtain all the results collected in previous sections.

In particular, this is the code that applies Double Deep Q Learning to the patrolling problem on the map for Mount Etna. It takes as input the most complete input possible, the RGB image in which 4 possible cells are distinguished: green for impassable areas, red for the agent's position, black for unvisited cells, and a grayscale range for visited cells indicating their relative interest value, i.e. $R(i, j)$.

```
"""
Created in 2024 by Rafael Bautista Linde
"""

# Import the libraries for general purpose #
import numpy as np
import sys
from collections import deque
from tqdm import tqdm

# Libraries for NNs (Keras y Tensorflow) #
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten, Conv2D, LSTM,
   TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.backend import clear_session
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from scipy.signal import lfilter

# Import the specific scenario #
```

```
import EtnaMap

# Eliminate the previous Keras sesion just in case #
clear_session()

# Load the scenario #
env = EtnaMap.Environment()

# Reset the scenario #
env.reset()

# Define what the state is #
def do_step(env,action):

    obs, reward, done, info = env.step(action)

    state = env.render()

    return state, reward, done, info

def reset(env):

    env.reset()

    state = env.render()

    return state

class Agent:

    """
    Class Agent where the training functions, neural networks and
       other stuff are included
    """

    def __init__(self,env,optimizers): # Class builder #

        # Initialize the atributes for the problem structure #
        # State size #
```

```
        self._state_size = (16,15,3)
        # Input size #
        self._action_size = 8
        # Optimizer for DNN #
        self._optimizer = optimizer
        # Cost function #
        self.loss_fn = keras.losses.Huber()
        # Initialize replay memory #
        self.replay_memory = deque(maxlen = 10000)
        # Parameters for soft-updating #
        self.tau = 0.125
        self.soft_update = 1 # ACTIVATED. If deactivated -->
           normal DDQN #
        # Number of episodes to update target network #
        self.target_episode_update_freq = 100

        # Reinforcement Learning hyperparameters #
        # Discount rate #
        self.discount_rate = 0.95 # Weight of future reward wrt
           latest one #
        self.epsilon = 0.995

        # Deep Neural Network #
        self.model, self.target = self._build_compile_model()

    # Define the agent method (how to accumulate experience in
       memory reward) #

    # Function which saves a step in memory #
    def append_memory(self, state, action, reward, next_state,
       done):
        self.replay_memory.append((state,action,reward,next_state,
           done))

    # Method to create DNN to estimate target function and Q
       function: #
    def _build_compile_model(self):

        # Model 1 - Only Conv2D #
```

```
        model = Sequential()
        model.add(Conv2D(8, kernel_size=(5, 5), strides=(2, 2),
                         activation='elu',
                         input_shape=(16,15,3)))
        model.add(Flatten())
        model.add(Dense(1024, activation='elu'))
        model.add(Dense(1024, activation='elu'))
        model.add(Dense(8, activation='linear'))

        model.compile(loss = 'Huber', optimizer = self._optimizer)

        target = keras.models.clone_model(model)
        target.set_weights(model.get_weights())

        return model, target

    # Take an action depending on the epsilon-greedy policy #
    def take_action(self,state):

        # Epsilon-greedy policy #
        if np.random.rand() <= self.epsilon:
            return env.action_space_sample()
        else:
            q_values = self.model.predict(state[np.newaxis])
            return np.argmax(q_values[0])

    # Experiences sampling #
    def sample_experiences(self,batch_size):
        indices = np.random.randint(len(self.replay_memory), size=
           batch_size)
        batch = [self.replay_memory[index] for index in indices]
        states, actions, rewards, next_states, dones = [
            np.array([experience[field_index] for experience in
               batch])
            for field_index in range(5)]
        return states, actions, rewards, next_states, dones

    def make_a_training_step(self,batch_size):
```

```
        # Experiences sampling from batch #
        experiences = self.sample_experiences(batch_size)

        # Return values #
        states, actions, rewards, next_states, dones = experiences

        # Predict Q values based on target function #
        next_Q_values = self.model.predict(next_states)

        best_next_actions = np.argmax(next_Q_values, axis = 1)

        next_mask = tf.one_hot(best_next_actions, self.
           _action_size).numpy()

        # Compute target #
        max_next_Q_values = (self.target.predict(next_states) *
           next_mask).sum(axis=1)


        # Evaluate target function - if DONE the discount rate
           does not apply because the experience was already done
           #
        target_Q_values = (rewards +  self.discount_rate *
           max_next_Q_values)

        # Reshape
        target_Q_values = target_Q_values.reshape(-1, 1)

        # Mask
        mask = tf.one_hot(actions, self._action_size)

        # Tape method to save the network forwarding and compute
           the descending gradient #

        with tf.GradientTape() as tape: # Compute the gradient
            all_Q_values = self.model(states)
            Q_values = tf.reduce_sum(all_Q_values * mask, axis=1,
               keepdims=True)
```

```
            # Loss function #
            loss = tf.reduce_mean(self.loss_fn(target_Q_values,
               Q_values))
        # Cost gradient #
        grads = tape.gradient(loss, self.model.trainable_variables
           ) # Gradient step
        # Apply gradient adjusting the parameters #
        self._optimizer.apply_gradients(zip(grads, self.model.
           trainable_variables))
        return loss

    def update_target(self,episode):

        if self.soft_update == 0:
            # Return directly the network every
               target_episode_update_freq number of  episodes #
            if(episode%self.target_episode_update_freq == 0):
                Drone.target.set_weights(Drone.model.get_weights()
                   )
                print("The model network has been dumped into the
                   target network.")
            else:
                pass
        # If soft update, pass the filtered parameters #
        else:
            weights = self.model.get_weights()
            target_weights = self.target.get_weights()
            for i in range(len(target_weights)):
                target_weights[i] = weights[i] * self.tau +
                   target_weights[i] * (1 - self.tau)
                self.target.set_weights(target_weights)



# Training #

optimizer = Adam(learning_rate = 0.001)
Drone = Agent(env,optimizer)
```

```
np.random.seed(42)
tf.random.set_seed(42)

"""Training parameters:
1.   Batch size
2.   Number of episodes
3.   Number of movements in each episode
"""

batch_size = 256
num_of_episodes = 1500
steps_per_episode = 300
best_score = -100000 # Maximum value - infinite
r_buffer = []
fr_buffer = []
epi_buffer = []
filtered_reward = 0
first = 1
loss = -1

"""Neural network summary"""

Drone.model.summary()
Drone.target.summary()

"""Training execution"""

fig = plt.figure(figsize=(16, 8))
fig.show()
fig.canvas.draw()
plt.xlim([0,num_of_episodes])

for episode in tqdm(range(0,num_of_episodes),desc='Rec: {:.2f} '.
   format(filtered_reward)):

    # Reset the scenario #

    state = reset(env)
```

```
    rew_episode = 0

    epi_buffer.append(episode+1)

    # Start an episode #
    for step in range(steps_per_episode):

        # Decrease epsilon #
        Drone.epsilon = max(1 - episode / num_of_episodes, 0.1)
        # Select an action #
        action = Drone.take_action(state)

        # Perform the step and observe its reward #
        next_state, reward, done, info = do_step(env,action)

        # Save the result from this action #
        Drone.append_memory(state,action,reward,next_state,done)

        # The latest state becomes the current one #
        state = next_state

        # Total sum of rewards #
        rew_episode += reward

        if done:
            break

        if len(Drone.replay_memory) > batch_size:
            loss = Drone.make_a_training_step(batch_size)

    # Load the reward from the latest episode in buffer #
    r_buffer.append(rew_episode)

    if first == 1:
        filtered_reward = rew_episode
        first = 0
    else:
        filtered_reward = 0.95*filtered_reward + rew_episode*0.05
```

```
    fr_buffer.append(filtered_reward)

    if rew_episode > best_score: # If the record is broken
        best_weights = Drone.model.get_weights() # Save the best
           DNN
        best_score = rew_episode # Update record
        print("\nEP: {} --- New record with Reward: {} and loss
           value{:06.2f}".format(episode,best_score,loss))

    # Soft Update of target network #
    Drone.update_target(episode)

    # Plots #
    plt.plot(epi_buffer,r_buffer,'b',alpha=0.2)
    plt.plot(epi_buffer,fr_buffer,'r')
    plt.grid(True, which = 'both')
    plt.pause(0.001)
    fig.canvas.draw()

"""When training is done, save the weights from the best trained
   DNN"""

# Load the best trained model #
Drone.model.set_weights(best_weights)
Drone.model.save("DoubleDeep2D_Etna_Model_BEST.h5")

print("\n Done! \n")

np.savetxt("reward_DoubleDQN_2D.csv", r_buffer, delimiter=",")
plt.figure(figsize=(16, 8))

# Plot the filtered reward #
plt.plot(epi_buffer,fr_buffer,'b')

# Standard deviation #
std = np.sqrt(np.power(np.asarray(fr_buffer) - np.asarray(r_buffer
   ),2))
std = lfilter(np.asarray([0.05]), np.asarray([1,-0.95]), std)
```

```
plt.fill_between(epi_buffer,fr_buffer+std,fr_buffer-std, color = '
    blue', alpha=0.3)
plt.grid(True, which = 'both')

plt.xlabel("Episode", fontsize=14)
plt.ylabel("Sum of rewards", fontsize=14)
plt.show()
```

Listing 6.1: Basis for Python code.