POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Reinforcement Learning Based Strategic Exploration Algorithm for UAVs Fleets

Supervisors
Giorgio GUGLIERI
Simone GODIO

Candidate
Cosimo BROMO

A.Y. 2021/2022

Abstract

Nowadays, autonomous navigation systems are becoming increasingly pervasive in everyday life and work. Unmanned Aircraft Systems (UASs) have been developed in the recent years, conquering different market segments and gaining popularity for their versatility and usefulness. Besides the economic benefits arising from their employments, ranging from crops monitoring in agriculture to fast parcel deliveries, their greatest incentive is the feasibility in hazardous and high risk operations. Their rapid growth is mainly associated to the quick development of algorithms and strategies for autonomous navigation and task execution, involving both traditional approaches and applications of artificial intelligence (AI) algorithms. One of the most challenging but rewarding field of study is the coordinated behavior of a number agents, collaborating to carry out the same high level task as well as distinguished low level objectives. Employment of fleets of Unmanned Aerial Vehicles (UAVs) may be particularly fruitful especially for time-sensitive operations, in which battery autonomy and time minimization are the most stringent requirements. Several applications and needs may exploit all such potentialities, provided that efficient collaboration models and strategies are implemented.
In this thesis, a Reinforcement Learning (RL) approach for coverage planning is presented. The main aim is to efficiently map an environment using a fleet composed of a certain number of UAVs, ranging from 2 to 10, while recognizing and avoiding obstacles. This objective envisages several difficulties, notably those related to collaborative behavior. In fact, each fleet component should autonomously move while taking into account both unexplored areas and other drones positions, in order to avoid mutual collisions and inefficient spreading in the environment. UAVs are trained to accomplish these tasks in a shared environment, by means of Proximal Policy Optimization (PPO) algorithm, a policy gradient method making use of Convolutional Neural Networks (CNNs) for policy and value function approximation. Training procedure is performed through a novel and modified version of PPO, which exploits all agents' trajectories to concurrently update a shared policy function, subsequently tested in a decentralized fashion with a variable number of UAVs. Trained fleets' performance is then assessed in terms of energy consumption, distribution statistics and coverage task accomplishment in simulated test environments.

Acknowledgements

I would like to express my gratitude for Professor Giorgio Guglieri, whose professionality and promptness in providing feedbacks and suggestions during this thesis' work have been of great help.
Furthermore, I want to thank my co-supervisor, Dr. Simone Godio, always available in monitoring the work and in offering useful and fruitful suggestions and assistance, who guided me until the conclusion of this thesis.

Computational resources provided by HPC@POLITO, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it)

Table of Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence
AER Averaged Episodic Return
AMID Average Minimum Distance
AMUD Average Mutual Distance
ANN Artificial Neural Network
APF Artificial Potential Field

CI Confidence Interval
COMA Counterfactual Multi Agent Policy Gradients
CNN Convolutional Neural Network

DDPG Deep Deterministic Policy Gradient
DQN Deep Q-Learning
DNN Deep Neural Network
DRL Deep Reinforcement Learning

EMA Exponential Moving Average

FCU Flight Control Unit
FEC Fleet Energy Consumption
FFNN Feed-forward Neural Network
FOV Field Of View

GA Genetic Algorithm
GAE Generalised Advantage Estimation

HRL Hierarchical Reinforcement Learning

IEC Individual Energy Consumption
IIR Infinite Impulse Response

KL Kullback–Leibler

LS Least Squares

MA Multi-Agent
MADDPG Multi Agent Deep Deterministic Policy Gradient
MAPPO Multi Agent Proximal Policy Optimization
MARL Multi Agent Reinforcement Learning
MDP Markov Decision Process
ML Machine Learning
MPC Model Predictive Control
MTD Mean Travelled Distance

NN Neural Network

OA Obstacle Avoidance

PPO Proximal Policy Optimization

RL Reinforcement Learning
ROS Robot Operating System
RRT Rapidly Exploring Random Tree

SAR Search and Rescue
SGD Stochastic Gradient Descent
SM Set Membership

TD Temporal Difference
TD3 Twin Delayed Deep Deterministic Policy Gradient
TRPO Trust Region Policy Optimization

UAS Unmanned Aircraft System
UAV Unmanned Aircraft Vehicle

Chapter 1

Introduction

1.1 Introduction

The goal of this thesis is to propose a Reinforcement Learning (RL)-based approach to solve the *coverage planning problem* using fleets of Unmanned Aerial Vehicles (UAVs), composed by 2 to 10 units. Efficient coverage strategies provide a series of advantages and applications in different fields, among which Search and Rescue (SAR) missions or aerial patrols, where the employment of autonomous systems reduces the need of human presence in such risky situations. Development of autonomous navigation and coordination of multi-agent systems has been a very heated field of study in the recent years, and it was the terrain of development of different strategies and approaches, taking inspiration from a variety of research areas, including Artificial Intelligence (AI) and Machine Learning (ML). Those research and application domains, which are in continuous and rapid growth, allow to tackle autonomous navigation and task collaboration problems exploiting several mathematical and human-inspired tools, like Convolutional Neural Networks (CNN). Reinforcement Learning is a branch of Machine Learning, devoted to train agents to achieve high performance in accomplishing difficult tasks, without the need to perform explicit programming. In this field, learning process of such decision makers is based on feedback rewards provided as function of the quality of their choices with respect to the intended target to be accomplished, summarized by a properly designed reward function.

In the proposed approach, *coverage planning* is implemented by equipping each UAV of a variable size fleet with a coverage agent, trained using RL strategies, which selects at each time the drone motion direction by processing information about fleet distribution and explored areas. The *obstacle avoidance* task is performed by another RL-agent, that fuses information about motion direction provided by the coverage agent with local obstacle detections performed by simulated range

sensors to provide the definitive motion direction. In real test implementations, this process allows to reduce the computational time that standard and iterative algorithms would require, since trained networks fastly process input data through forward passes. Training and test phases are performed by assuming a constant mutual communication among UAVs, in terms of both respective positions and obstacle detections. Once fleets are trained in shared simulated environments to be fully explored, their performances are assessed in terms of adopted distribution strategies, exploration time and energy consumption.
The proposed algorithm is implemented in Python language, using Python 3.9.5. Mostly used libraries for the code development are Tensorflow 2.4.1 [1], for neural network management and training, numpy 1.21.2 for general purpose calculations, scikit-learn 1.0.2 [2] and pandas 1.4.2 for data analysis. Custom simulation environments are used for training and test phases, taking inspiration by OpenAI Gym framework for 2D simulations, whereas some tests in 3D environments are carried out in Gazebo with ROS for communication and decision-making management, and PX4 Autopilot software for vehicles' control. All the code and models developed during this project are available in my GitHub repository https://github.com/cosimobromo/RL_UAVs_fleets.

1.2 Thesis outline

This thesis is organized as follows: in chapter 2, the state of art of RL applications to autonomous systems is presented, including path planning and obstacle avoidance methods, as well as a collection of applications in the area of multi-agent systems and specific approaches to coverage planning problem. In chapter 3 the most relevant theoretical notions about RL framework are provided, along with a classification of the mostly used RL algorithms. Furthermore, an in-depth analysis of the PPO algorithm follows. In chapter 4, the proposed approach is described with reference to assumptions, agent interactions, reward function shaping and state and action spaces definition. In chapter 5, the training algorithm for RL agents is described, accompanied by the presentation of the learning curves. In chapter 6, the algorithm's test results are presented, either on a large test set of maps, and on specific ones selected as reference. In chapter 7, some results obtained by simulations in 3D environments with ROS are presented as well as some pictures taken during one exploration process. Finally, in chapter 8, conclusions are drawn, along with a discussion about possible further improvements.
At the end of the thesis, in Appendix A there is a brief overview of the most common Artificial Neural Network (ANN) structures and training algorithms, while in Appendix B a formalization of the KL-Divergence, involved in the PPO optimization function, is provided, along with an example of its application.

Chapter 2

State of art

Reinforcement Learning and autonomous single-agent or multi-agent systems constitute a wide field of development and research, characterized by a broad and varied collection of approaches and works. In this chapter, some insights on the major development of such topics are provided, along with some plausible applications of the proposed work.
In Sec. 2.1, some instances of *Reinforcement Learning applications* for control systems and autonomous vehicles are illustrated, along with more detailed applications of such framework to path planning and obstacle avoidance problems in (2.1.1).
In Sec. 2.2, main advantages and challenges in *multi-agent systems* literature are presented, along with specific applications for *coverage planning* in (2.2.1).

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML) aiming at teaching agents to select actions so as to maximize discounted sum of rewards collected over time. Given its high level description and mathematical formulation, RL algorithms and procedures can be easily handled in many research and practical fields, ranging from industrial automation to financial studies.
Some of the earliest successful attempts in RL field include training of agents in playing popular Atari games [3]. This terrain provided outstanding results despite the quite basic algorithms (Q-learning, DQN) and trained agents even outperformed highly expert human players in solving most of the games.
RL is exploited for typical control tasks as well: in [4], Deep Deterministic Policy Gradient (DDPG), a policy gradient based algorithm, is exploited to develop an efficient control law for a nuclear reactor. Additionally, many industrial applications adopt RL control strategies on industrial manipulators carrying out a variety of tasks [5]; physical implementation and control performances are strengthened

by specific training strategies, including domain randomization [6] for increased robustness in filling the simulation-reality gap.
Finally, Reinforcement Learning exhibited notable potentialities in the financial field as well, as for portfolio optimization, even in combination to traditional approaches (like convex mean-variance optimization) [7]. In (2.1.1), an in-depth investigation about RL-based path planning and obstacle avoidance applications from the literature is provided.

2.1.1 Path Planning and obstacle avoidance

Besides the multi-domain approaches and potentialities of Reinforcement Learning, most of the efforts and consequent findings involve employment of RL for autonomous vehicles navigation and control, particularly in terms of path planning strategies. In the path planning literature, a series of methods and algorithms have been proposed (a brief summary is reported in [8]), including A*, D*, RRT, or APF-based planners with traditional control strategies [9, 10]. Path planning methods try to compute optimal paths (with respect to some predefined metrics) connecting a starting position to a target one, while avoiding obstacles during navigation. In Fig. 2.1 a path planned using D* Lite algorithm [11] is displayed, along with starting and ending position.

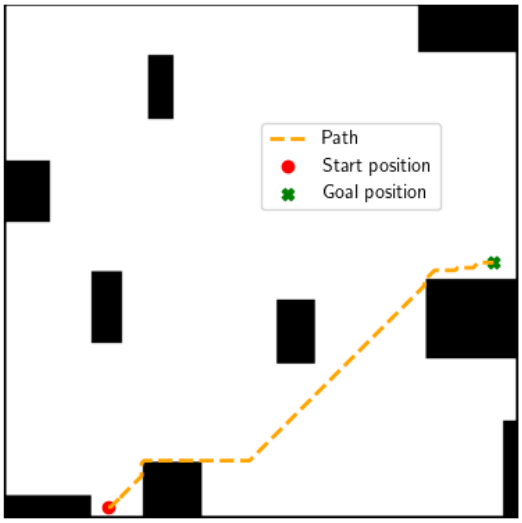


Figure 2.1: Path Planned through D* Lite Algorithm

Some algorithms, including A*, deal with static environments only, while others are more suitable to be employed with dynamic obstacles. With the development of computational resources, in the years alternative path planning methods, especially RL-based ones, have been proposed as well. [12, 13, 14, 15, 16] propose end-to-end local motion planners for ground robots, using heuristic-based reward functions. Specifically UAV-related approaches and applications have shown optimal performances; some of them, see the combination of RL techniques along with traditional approaches, like Model Predictive Control (MPC) [17].

2.2 Multi-Agent Systems

Multi-agent systems constitute a complex field of development, in which besides the benefits raising from the joint employment of different autonomous systems in performing similar tasks simultaneously, a series of complications may arise, especially linked to coordination. Multi-agent systems are implemented in multiple domains, including video games [18, 19] that, despite without practical implications, represent a fruitful development terrain and baseline for research. In addition, multi-agent systems arise in many industrial applications, involving, for example, manipulators coordination in manufacturing processes. The most interesting area, relatively to this thesis topic, concerns autonomous vehicles guidance and control, that may be associated to different purposes, including war, geological or medical related tasks. Applied methodologies may be varied: in [20] a Set Membership (SM) approach for UAVs coordination during targets tracking is described, whereas [21] proposes a Multi Agent SLAM-based exploration with noise robustness, employing on-purpose designed utility functions. With the increasing need of developing special multi-agent algorithms, RL framework extended in this direction, with a series of MA-based algorithms, which are either the extension of single agent methods or on-purpose developed ones. According to Multi Agent Reinforcement Learning (MARL) literature and theory [22], multi-agent problems can be classified in three possible settings type:

- *Purely cooperative settings*: Agents share the same reward and a common goal to be achieved;
- *Purely competitive settings*: Agents are characterized by contrasting purposes, one agent's reward increases to the expenses of another one;
- *Mixed cooperative-competitive settings*: It is the most general case, in which agents tend to maximize their individual rewards while subdued to some shared objectives as well.

For purely cooperative settings, some special-purpose algorithms have been proposed, since a shared objective function allows to treat all interacting agents at the

same way. Among them, Counterfactual Multi Agent Policy Gradients (COMA) [23] exploits agent's equality to simplify value function estimation, while more complex algorithms, built upon their single-agent counterparts, have been adapted for all possible multi agents settings, since agents are allowed to have possibly different objectives and they are trained accordingly [24, 25]. Among recent applications of MARL, combat tasks deserves to be mentioned: in [26] a modified version of Multi Agent Deep Deterministic Policy Gradient (MADDPG) is used, for combat tasks, in a mixed cooperative-competitive environment; in [27] DDPG is exploited in a fully collaborative environment to coordinate a UAVs fleet to reach a target from different initial positions. In (2.2.1) a number of *coverage planning* approaches using MARL are presented.

2.2.1 Coverage Planning

Coverage planning problem consists in finding an efficient exploration strategy of an environment through a number of observers, so that an unknown area can be fully explored. A solved coverage planning problem carries a series of benefits applicable in many applications, basically linked to monitoring, mapping and data gathering. In [28], MADDPG algorithm is applied to a swarm of buoys in order to efficiently spread them so as to avoid their individual influence areas from overlapping and to provide a constant monitoring of the considered underwater environment. The same issue is approached for a small UAVs fleet in [29] and for some ground robots in [30], in which a coordinated strategy for agricultural robots allows to provide constant crops monitoring, with coordinated reallocation if one unit needs to recharge its battery. Some MARL coverage planning approaches are based on maps pre-processing techniques, exploiting for instance Voronoi partitioning [31, 32] in collaboration with RL control strategies.

Chapter 3

Theoretical Background

In this chapter, the main theoretical concepts needed for understanding the proposed methodology are briefly illustrated. In Sec. 3.1 Reinforcement Learning basics and employed mathematical tools are explained; in Sec. 3.2 the wide variety of algorithms and their principal differences will be illustrated and in Sec. 3.3 some details about Proximal Policy Optimization (PPO) algorithm, the core of proposed methodology, will be provided.

3.1 Reinforcement Learning basics

Reinforcement Learning (RL) constitutes a branch of Machine Learning (ML) devoted to train a decision maker, usually referred to as an *agent*, to take actions on the basis of its current state, so that the discounted sum of rewards it receives from the environment over time is maximized. The other branches of ML are *supervised learning*, involving model learning techniques based on input-output labeled data samples, and *unsupervised learning*, specifically devoted to data classification and analysis without any available a priori knowledge, including clustering and dimensionality reduction methods. In Reinforcement Learning, training data is collected as a result of agent-environment interaction, which is usually performed in a simulated fashion. At each step time t, an agent selects and performs an action a_t from a state s_t, which results in a transition to a consequent state s_{t+1} in the next time step $t+1$ and a reward r_{t+1}. Rewards act as feedback signals to agents, providing information about the effectiveness of choosing an action given the current state, in compliance with the objective goal. Reinforcement Learning agents are firstly trained during a specifically devoted phase and then their performance is assessed in the test phase. To fulfill optimality conditions, during the training process an adequate balance between *exploration* of the action space and *exploitation* of the learned strategies must be found. Training process

is made up of a sequence of *episodes* over which agent collects rewards until a terminal condition is reached.

3.1.1 Markov Decision Process

Markov Decision Processes (MDPs) are meant to formalize the sequential decision making process involving agents which learn to maximize a given goal while interacting with an environment. Denoting as *agent* the learner and decision maker and the *environment* whatever does not include the agent itself, using MDPs it is possible to describe consistently all situations in which current actions influence long-term rewards as well as immediate ones [33]. MDPs are *stochastic control processes*, characterized by the *Markov Property* (see Def. 3.1.1).

Definition 3.1.1 (Markov Property)**.** A stochastic control process satisfies the *Markov Property*, and if so it is said to be *Markovian*, if the conditional probability distribution of future states of the process depends on the current state only and not on the past ones.

Definition 3.1.2 (Finite Markov Decision Process)**.** If state and action spaces are *finite*, the Markov Decision Process is said to be a *Finite Markov Decision Process* (Finite MDP).

Agent and environment interactions happen in discrete time: at each time step agent receives as input a representation of the current environment's state s_t, selects an action a_t and, as a consequence of the state-action pair (s_t, a_t), moves to the next state s_{t+1} while receiving a reward r_{t+1}. During this process, agent's trajectory is collected, and in RL settings it is employed for its learning process.

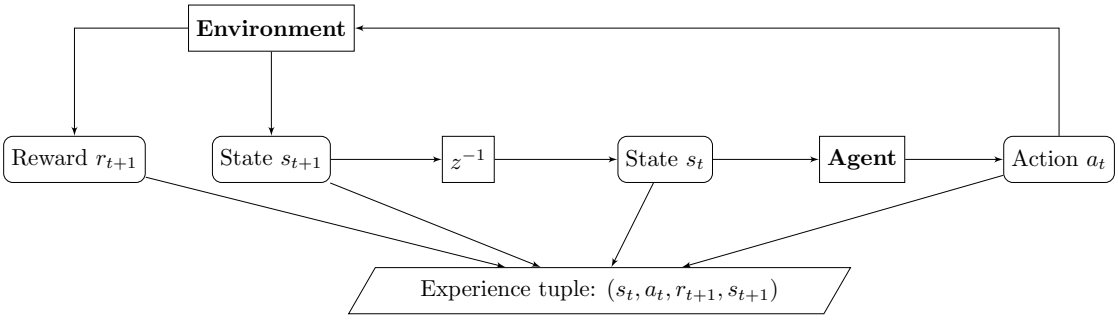


Figure 3.1: Agent-environment interaction, summarized at each time step by an *experience tuple* $(s_t, a_t, r_{t+1}, s_{t+1})$. z^{-1} indicates the one-step delay.

A Markov Decision Process is defined as a 5-dimensional tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where:

- $\mathcal{S}$ is the *state space*, i.e. the set of all possible states;
- $\mathcal{A}(s)$ is the *action space*, i.e. the set of all possible actions available in a given state s;
- $\mathcal{P}(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ is the *transition probability*[1], i.e. the probability that being in a state s and selecting an action a, agent will transition to new state s'. $\mathcal{P}$ is responsible for the definition of the one-step environment dynamic. In a *deterministic* environment, once defined the state-action pair (s,a), the consequent state s' is defined uniquely, i.e. $\exists! s' : \mathcal{P}(s'|s,a) = 1$ and $\forall \tilde{s} \in \mathcal{S} \setminus s' : \mathcal{P}(s,a,\tilde{s}) = 0$
- $\mathcal{R}(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function, mapping the tuple (s,a,s') to a number $r \in \mathbb{R}$.
- γ is the *discount rate*, weighting the future rewards' influence on the *discounted returns* (see Eq. 3.2).

As previously anticipated, in Reinforcement Learning the goal is to maximize the total amount of rewards the agent receives over time, that is generally referred to as the *expected return*, reported in Eq. 3.1.

$$G_t \doteq r_{t+1} + r_{t+2} + \cdots + r_T \tag{3.1}$$

Expected return maximization may fail when it is not possible to guarantee the end of the agent-environment interaction, i.e. when a terminal state may be not reachable ($T = \infty$). That's why, in Reinforcement Learning, the *discounted return* is chosen as maximization objective most of the times [2]. The main idea behind the concept of discounting is to weight future rewards less as considering farther experiences in time. This is done by using a weighting parameter known as *discount rate* $\gamma \in [0,1]$, exponentially decayed as time increases, as in Eq. 3.2. This definition implies that, for $\gamma \to 1$, interest towards future rewards is strengthened and the agent is required to have enhanced forecasting capabilities.

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.2}$$

Discounted return can be even defined in a recursive form, provided that if a terminal state exists at time $t = T$, $G_T = 0$:

$$G_t = r_{t+1} + \gamma G_{t+1} \tag{3.3}$$

[1] The symbol "|" indicates conditional probability. For instance, $p(x|y)$ denotes the probability of a certain event x conditioned on event y.

[2] It is possible to prove that, under some mild conditions on the reward function, mostly related to boundedness of instantaneous rewards, the discounted return is finite even for $T = \infty$.

3.1.2 Policy and Value Functions

In Reinforcement Learning, an agent learns by updating during training process its *policy*, which indicates the way states are mapped to actions. Generally, a policy $\pi(a|s) : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ defines a probability distribution over the action space $\mathcal{A}(s)$ reachable from state s.
Value functions, instead, give a quantitative indication of the discounted return that would be attained from a given state, by following a defined policy π. Specifically, there are three essential value functions in RL: *state-value* function $v_\pi(s)$ (see Def. 3.1.3), *action-value* function $q_\pi(s, a)$ (see Def. 3.1.4), *advantage* function $A_\pi(s, a)$ (see Def. 3.1.5).

Definition 3.1.3 (State-value function)**.**
The state-value function $v_\pi(s) : \mathcal{S} \mapsto \mathbb{R}$ indicates the expected discounted return that would be attained starting from a state s and following a policy π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right] \tag{3.4}$$

Definition 3.1.4 (Action-value function)**.**
The action-value function $q_\pi(a, s) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ indicates the expected discounted return that would be attained starting from a state s, performing action a and following policy π thereafter. It is often referred to as *Q-function* and its numerical value is indicated as *Q-value*.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right] \tag{3.5}$$

Definition 3.1.5 (Advantage function)**.**
The advantage function $A_\pi(a, s) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the difference between the action-value and the state-value function. It provides an indication of the convenience in selecting an action a before following policy π rather than following it immediately.

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \tag{3.6}$$

3.1.3 Optimality and Bellman equation

The target of Reinforcement Learning is to find the *optimal* policy, i.e. the policy π^* ensuring the highest possible expected return.

Definition 3.1.6 (Optimal Policy)**.** A policy π^* is said to be optimal if:

$$\pi^* \geq \pi, \ \forall \pi \tag{3.7}$$

where $\pi^* \geq \pi \Leftrightarrow v_{\pi^*}(s) \geq v_\pi(s) \ \forall s \in \mathcal{S}$.

An optimal policy π^* is such that the optimal state-value function $v_*(s)$ and the optimal action value function $q_*(s,a)$ are:

$$\begin{cases} v_*(s) & = \max_\pi v_\pi(s) \\ q_*(s,a) & = \max_\pi q_\pi(s,a) \end{cases} \tag{3.8}$$

For an optimal policy π^* the *Bellman optimality equations* hold; they are reported for a stochastic environment in Eq. 3.9.

$$\begin{cases} v_*(s) & = \max_a \sum_{s'} p(s'|s,a)[r + \gamma v_*(s')] \\ q_*(s,a) & = \sum_{s'} p(s'|s,a)[r + \gamma \max_{a'} q_*(s,a')] \end{cases} \tag{3.9}$$

In the case of a deterministic environment, as specified in the description of the Markov Decision Process in (3.1.1), the sum reduces to a single element. Bellman equation represents the optimality condition for a policy but also a tool to reach optimality. Notwithstanding the existence of an optimal condition, it is not always guaranteed that policy learning leads to an optimal policy, since RL algorithms make use of approximations and function parametrization, as indicated in (3.1.4) for what concerns policy and value function approximation using Artificial Neural Networks (ANN).

3.1.4 Artificial Neural Networks as function approximator

In most of RL problems, the agent-environment interactions shall be defined in continuous state-action spaces. In discrete state-action spaces, basic RL methods, including Q-learning, employ *Q-tables* that map state-action pairs to Q-values. However, if state and action spaces become larger, their dimensions induce a large raise in complexity, while for continuous spaces the use of a Q-table is absolutely impracticable. If both policy and value functions receive continuous inputs (states and/or actions), they shall be coherently approximated.
Thanks to the rapid and impressing development of computational resources in the recent years, *Artificial Neural Networks* (ANN) are progressively becoming one of the most powerful tool in ML, allowing input-output mapping in both classification and regression problems. In Deep Reinforcement Learning (DRL), deep networks are specifically used for function approximation, of both policy and value functions. The validity of this approach is supported by theoretical foundations, mainly summarized by the universal approximation theorem presented afterwards. For more details about ANN, refer to Appendix A.

Universal approximation theorem

Feedforward Neural Networks (FFNNs) can approximate efficiently several classes of functions. Proofs of their universal approximation capabilities are attributed to

G. Cybenko [34] and K. Hornik [35]. The theorem formulation is given by (3.1.1).

Theorem 3.1.1 (Universal Approximation Theorem)**.** *Let $\psi(\cdot)$ be a nonconstant, bounded and monotonically-increasing continuous function and I_n be the n-dimensional hypercube $[0,1]^n$. $C(I_n)$ is the set of continuous functions on I_n. Given any $\epsilon > 0$ and any function $f \in C(I_n)$, $\exists N \in \mathbb{N}; v_i, b_i \in \mathbb{R}; w_i \in \mathbb{R}^n$, $i = 1, \dots, N$ such that*

$$F(x) = \sum_{i=1}^{N} v_i \psi(w_i^T x + b_i)$$

is an approximation of the function f, i.e.

$$|F(x) - f(x)| < \epsilon, \ \forall x \in I_n$$

The same result holds if replacing I_n with any compact subset of $\mathbb{R}^n$. The specific structure of F can be actually attained, for example, by a FFNN with one hidden layer with sigmoid activation function $\sigma(x) = \frac{e^x}{1+e^x}$ and output layer with linear activation. It is noticeable that Theorem 3.1.1 does not provide any indication about a lower bound for N, but it nevertheless guarantees its boundedness and existance.

3.2 RL Algorithms classification

There are several methods and aspects according to which it is possible to classify RL algorithms. One discriminating factor is the knowledge of the environment model accessible by the agent. The environment dynamics is described by the *transition function* $\mathcal{P}$ as indicated in (3.1.1) but, in most of the cases, the agent does not have any awareness of the state transition probabilities and associated rewards. Obviously, if this knowledge is instead possible, agent will be benefited in terms of long-term prediction accuracies, and sample efficiency, i.e. how much the agent learns by a small amount of experiences. Nevertheless, *model-based* learning techniques, are more difficult in terms of implementation, that's why most RL applications make use of *model-free* RL algorithms. In Fig. 3.2 an overview of the modern RL algorithms is shown, classified with respect to the environment model knowledge.
Focusing on model-free RL algorithm, two approaches are feasible. In Q-learning based algorithms, policy improvement derives from Q-function learning and maximization. In policy optimization methods, instead, policy is updated by maximizing a specific objective function, aiding with value function estimations. After all, many algorithms like Deep Deterministic Policy Gradient (DDPG) or Twin Delayed Deep Deterministic Policy Gradient (TD3) show similarities with both approaches, and cannot be classified definitely in one or the other.

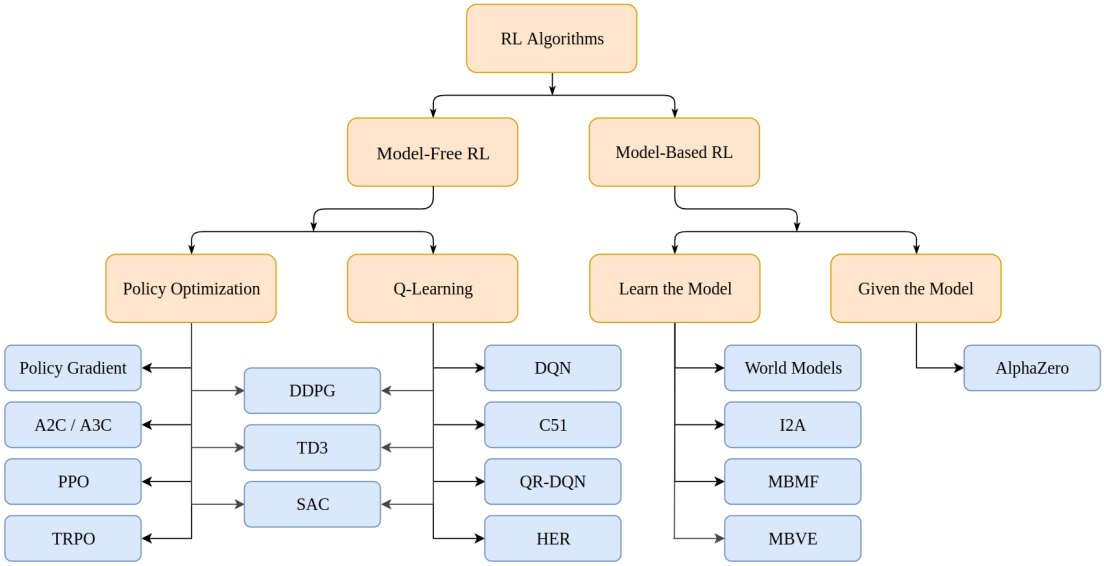


Figure 3.2: Taxonomy of algorithms in modern RL. Image taken by [36].

3.2.1 On-policy vs off-policy methods

On the basis of the learning process details, algorithms can be defined as:

- *On-policy*: These algorithms, as the name suggests, update the same policy that is used to take actions and to collect experiences on which learning takes place. Among on-policy algorithms, PPO appears as one of the most stable ones. In general, on-policy algorithms exhibit better learning stability, but they may be not optimal in terms of exploration capabilities.

- *Off-policy*: In these algorithms, the policy employed to record experiences during the training process differs from the one which is progressively updated. For example, in DDPG or TD3, the policy under training is used to take actions, then corrupted by correlated noise sequences which affect transitions and recorded experiences. In some other algorithms, including DQN, actions are sometimes chosen greedily with the policy (the ones associated to the highest Q-value), sometimes randomly to promote exploration. Even in this case, the learned policy differs from the one used to take actions and record experiences. Off policy algorithms usually show better exploration capabilities, at the cost of more unstable learning processes and convergence issues.

3.2.2 Actor-critic methods

Most of modern RL algorithms are *actor-critic* methods, making use of two models, an *actor* and a *critic*, to represent respectively the policy and the value function. They allow to learn a state-value (or action-value) estimator, the critic, which provides update directions for the policy, the actor. In Fig. 3.3 the actor-critic framework is depicted. If the critic represents the state-value function (in algorithms like PPO), critic model is trained by regression exploiting states (as inputs) and rewards (for discounted return computation). If the critic models the action-value function (as in DDPG, TD3), its inputs are both states and actions, while rewards are used for output computation.

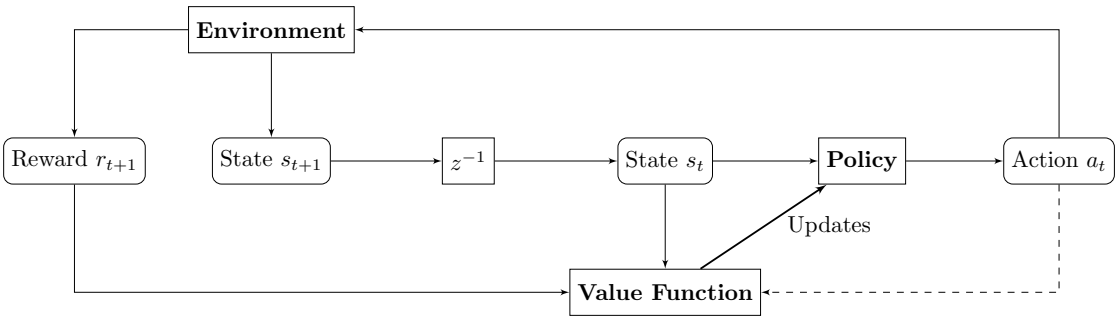


Figure 3.3: Actor-Critic framework details. z^{-1} is the one-step delay.

3.3 Proximal Policy Optimization (PPO)

In this section, a detailed analysis of the Proximal Policy Optimization (PPO) algorithm is presented, along with the main concepts and techniques employed in the presented work. PPO is a recent algorithm, presented by OpenAI in 2017 [37], empowering Trust Region Policy Optimization algorithm (TRPO) with a lower implementation complexity and an enhanced sample efficiency. PPO is an on-policy algorithm, that can be used in environments with both discrete or continuous state-action spaces and it is an actor-critic method, in which the critic approximates the state-value function. As TRPO, PPO improves the policy avoiding too large update steps, possibly causing instability and performance degradation. This can be done using either a *KL-divergence* constraint (refer to Appendix B) or a *clipped surrogate objective* or both.
Let θ_{old} and θ represent the policy parameter before and after an update step. The probability ratio $r_t(\theta)$ in a given time t is reported in Eq. 3.10.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{3.10}$$

The probability ratio $r_t(\theta)$ is exploited in the PPO objective function, as shown in Eq. 3.11.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t))] \tag{3.11}$$

This objective function penalizes excessive policy changes by clipping the probability ratio $r_t(\theta)$ between $[1-\epsilon, 1+\epsilon]$, while taking the minimum between the unclipped objective and the clipped one induces conservativeness. $\hat{A}_t$ is the estimated advantage (see (3.3.1) for advantage terms estimation procedure), whose sign implicates two possible situations:

- $\hat{A}_t > 0 \Rightarrow q_\pi(s_t, a_t) > v_\pi(s_t)$: The selected action brings higher rewards: policy π shall be updated in that direction;
- $\hat{A}_t < 0 \Rightarrow q_\pi(s_t, a_t) < v_\pi(s_t)$: The selected action brings lower rewards: policy π shall move away from that direction.

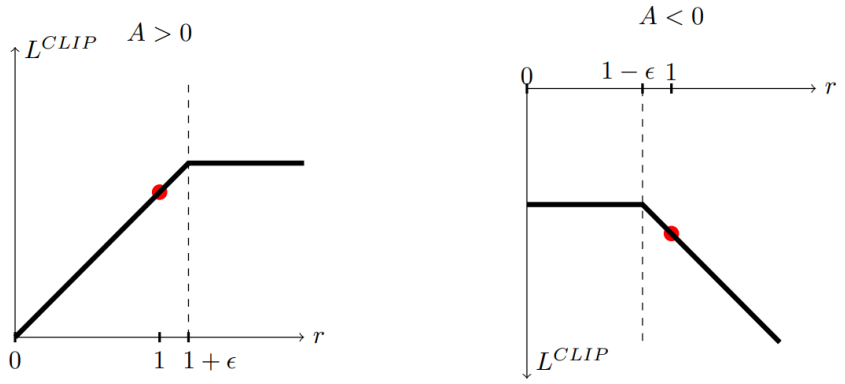


Figure 3.4: Plots of L^{CLIP} for positive and negative advantages, as function of the probability ratio $r_t(\theta)$. Image taken by [37].

In Fig. 3.4 the clipped objective $L^{CLIP}(\theta)$, for both signs of the advantage, is plotted as function of the probability ratio $r_t(\theta)$. For positive advantages, clipping penalizes excessive policy changes in the direction of significant probability ratio increase, while for negative advantages clipping avoids significant policy changes in the direction of critical probability ratio decrease. KL-Divergence constraint can be introduced either as penalty in the objective function or as a constraint to the optimization problem. In this thesis, the constrained version only will be presented as it has been used for the algorithm development. The complete optimization problem to be solved is indicated in Eq. 3.12, with δ being the KL-divergence

threshold.

$$\begin{aligned} &\max_{\theta} \hat{\mathbb{E}}_t \Big[\min \Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \Big) \Big] \\ &\text{s.t. } \hat{\mathbb{E}}_t \Big[KL\Big[\pi_{\theta_{old}}(\cdot \mid s_t), \pi_{\theta}(\cdot \mid s_t)\Big]\Big] \leq \delta \end{aligned} \tag{3.12}$$

The optimization objective for value function regression is reported in Eq. 3.13, with Q being the overall training epoch length.

$$\phi = \arg\min_{\phi} \frac{1}{Q} \sum_{t=0}^{Q} \Big(V_{\phi}(s_t) - \hat{R}_t \Big)^2 \tag{3.13}$$

Being the PPO algorithm an actor-critic method, the critic model is trained by regression, fitting the value function to the *rewards-to-go* concurrently to policy optimization. Rewards-to-go correspond to discounted returns, computed on the basis of the experienced transitions and gathered rewards. From a practical point of view, they can be computed by means of an Infinite Impulse Response filter (IIR); their definition is reported in Eq. 3.14, indicating with T a generic episode length.

$$\hat{R}_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \tag{3.14}$$

The pseudocode of the PPO algorithm with clipped objective function and KL-Divergence constraint is reported in Alg. 1, adapting [38].

Algorithm 1 PPO with clipped objective and KL-constraint

Initialize: policy parameters θ_0, value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ running $\pi_k = \pi(\theta_k)$ in the environment
  Compute rewards-to-go $\hat{R}_t$
  Compute advantage estimates $\hat{A}_t$ using $\hat{V}_{\phi_k}$ (see 3.3.1)
  Update policy by solving the optimization problem indicated in (3.12)
  Fit value function by regression on mean-squared-error, as in (3.13)
end for

Because of the multi-UAV settings described in this thesis, the reported algorithm constitutes a simplified version of the truly implemented one, that has been suitably modify to address the multi-agent settings (see Sec. 5.1.1). Anyhow, the original algorithm is reported for completeness.

3.3.1 Generalized Advantage Estimation (GAE)

Advantages' estimates are needed for the computation of the PPO clipped objective function. Recalling the definition of advantage function (see Eq. 3.6), at a given time step, advantage can be estimated in several ways, as reported in Eq. 3.15.

$$\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t) \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 \hat{V}_\phi(s_{t+2}) - \hat{V}_\phi(s_t) \\
&\vdots = \vdots \\
\hat{A}_t^{(\infty)} &= r_t + \gamma r_{t+1} + \gamma^2 \hat{V}_\phi(s_{t+2}) + \cdots - \hat{V}_\phi(s_t)
\end{aligned} \tag{3.15}$$

The superscript indicates the amount of time steps between the current one t and the one in which state-value estimation is carried out. In general, being the critic model an approximation of the state-value function $\hat{V}_\phi \neq V_\pi$, advantage estimates are biased. $\hat{A}_t^{(j)}$ with $j \to 1$ are generally subject to lower variance but higher bias, since there are few terms in the sum, and estimate relies heavily on the state-value estimation (which can be biased). If $j \to \infty$, the number of terms in the sum rises linearly with j, this makes such estimates more susceptible to higher variance but reduces the bias [39].
Typically, a value j is chosen in order to find a compromise between variance and bias, but Generalized Advantage Estimation (GAE) allows to concurrently exploit all available information by considering an *exponentially-weighted average* of all estimates. Generalized Advantage Estimate $\hat{A}_t^{GAE(\gamma,\lambda)}$ is defined in Eq. 3.16, as function of the parameter $\lambda \in [0, 1]$, tuning the compromise between bias and variance.

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots) \tag{3.16}$$

Notice that, being $\lim_{t=0}^{\infty} \lambda^t = \frac{1}{1-\lambda}$ if $\lambda < 1$, if all estimates are perfectly equal, the Generalized Advantage Estimate $\hat{A}_t^{GAE(\gamma,\lambda)}$ coincides with them, as formalized in Eq. 3.17.

$$\hat{A}_t = \hat{A}_t^{(1)} = \cdots = \hat{A}_t^{(\infty)} \implies \hat{A}_t^{GAE(\gamma,\lambda)} = \hat{A}_t \tag{3.17}$$

Expanding expressions in Eq. 3.15 and denoting with $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ the TD residual, the advantages estimates can be reformulated as in Eq. 3.18.

$$\begin{aligned}
\hat{A}_t^{(1)} &= \delta_t^V \\
\hat{A}_t^{(2)} &= \delta_t^V + \gamma \delta_{t+1}^V \\
&\vdots = \vdots \\
\hat{A}_t^{(\infty)} &= \sum_{k=0}^{\infty} \gamma^k \delta_{t+k}^V
\end{aligned} \tag{3.18}$$

Then, expanding Eq. 3.16 in Eq. 3.19 and exploiting the TD-residuals, it is possible to obtain the final form of the Generalized Advantage Estimate $\hat{A}_t^{GAE(\gamma,\lambda)}$ as demonstrated in Eq. 3.19.

$$\begin{aligned}\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\delta_t^V(1+\lambda+\lambda^2+\dots)+\gamma\delta_{t+1}^V(\lambda+\lambda^2+\dots)+\\ &\quad \gamma^2\delta_{t+2}^V(\lambda^2+\lambda^3+\dots)+\dots) = \\ &= (1-\lambda)\left(\delta_t^V\frac{1}{1-\lambda}+\gamma\lambda\delta_{t+1}^V\frac{1}{1-\lambda}+(\gamma\lambda)^2\delta_{t+2}^V\frac{1}{1-\lambda}+\dots\right) = \\ &= \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}^V\end{aligned} \tag{3.19}$$

From Eq. 3.19 it appears evident how it is possible to compute the GAE as the discounted sum of the TD residuals, with discount rate $\gamma\lambda$. λ must be tuned opportunely in order to find a suitable trade-off between bias and variance. Despite in the GAE the discount rate is the product of γ and λ, those parameters shall be tuned independently, since having different purposes and meanings.

Chapter 4

Methodology

In this chapter, a detailed description of the proposed approach for the project development is presented. This chapter is organized as follows: in Sec. 4.1 the main assumptions and simplifications are reported; in Sec. 4.2 the general settings of the algorithm are illustrated, in Sec. 4.3 and 4.4, the coverage and obstacle avoidance agents are described; in Sec. 4.5 the reference UAV and model for energy consumption analysis are summarized.

4.1 Assumptions

In this section, the adopted assumptions are described. It is worth specifying that most of them can be considered sufficiently reasonable and the algorithm can be deployed in a real implementation with only limited adaptations.

4.1.1 Fleet Size

The entire work has been developed trying to make it fastly and efficiently adaptable to variable fleet sizes. In order to assess performances and exploration capabilities in terms of fleet dimension, all training and testing simulations have been performed with a number of UAVs ranging from 2 to 10, in particular $N = 2, 3, 4, 5, 6, 8, 10$. Even larger fleets can be trained and tested using the same framework, with just small adaptations in terms of input states.

4.1.2 Information Exchange

Given that the main purpose of this thesis is to solve the coverage planning problem, in order to address it and test the developed model without additional complexities, an *instantaneous* and *noise-free* data exchange framework is considered. This

allows to decouple the major objective of this thesis from other plausible problems, which can be anyway addressed using special-purpose approaches in parallel to the proposed algorithm. Possible robustness to noise affecting UAVs' localization data is anyway introduced in the definition of the input states to the coverage agents (see Sec. 4.3).

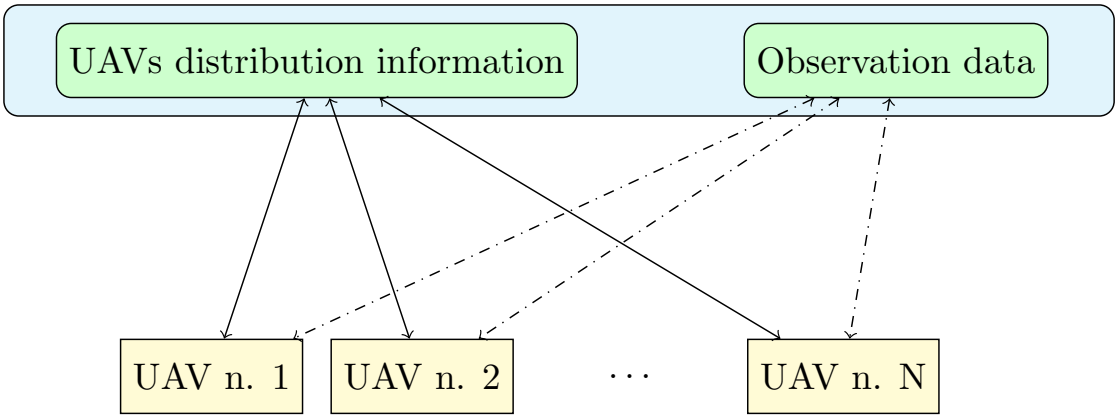


Figure 4.1: Scheme reporting the centralized architecture for data exchange in the fleet. Each UAV shares its position and observed data with a centralized storage and processing unit.

In Fig. 4.1 the data exchange framework employed in this thesis is shown, representing a centralized storage and processing unit, to which UAVs send instantaneous information about their positions in the map and observations they gather during the exploration, accessible at each time instant and without any transfer delay.

4.1.3 Exploration Environment and Maps

All simulated environments, with an obstacle occupancy ranging from 0 to 25 % of the overall surface area, are built upon binary matrices M of dimensions 100×100, defined in Eq. 4.1.

$$M \in \{0,1\}^{100\times100} : M_{ij} = \begin{cases} 0, & \text{if in position } (i,j) \text{ no obstacle is present} \\ 1, & \text{if in position } (i,j) \text{ an obstacle is present} \end{cases} \tag{4.1}$$

Potentially, such maps can approximate any squared real field with any dimensions, assuming a different approximation precision of the matrix cells. As it commonly happens in ML settings, 2 sets of maps were created: a *training set* composed by 2700 maps, and a test set, made up of 300 maps. All of them were created by randomly placing obstacles, in compliance with the following requirements:

- All obstacles are *rectangular*, this choice was made specifically for shape prediction purposes, as specified in (4.1.5);
- Obstacle dimensions must be between the 8 and the 20 % of the map size and their height is greater than UAVs' flight altitude;
- The distance in l_∞ norm between two obstacles' centers must be greater than or equal to the maximum sum of the obstacles' dimensions, multiplied by a safety factor $q = 1.55$. This value was determined by trial and error, in order to place obstacles distant enough to allow drones to move among them.

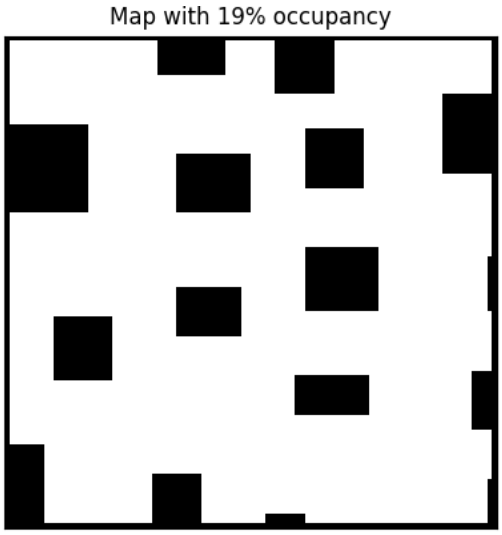


Figure 4.2: Example of a map with 19 % obstacle occupancy

In Fig. 4.2 a randomly generated map with 19% obstacle occupancy is shown, while in Alg. 2 the creation procedure of the simulated maps is described.
In order to be adaptable to different field dimensions, each UAV position X is defined as a vector with normalized coordinates with respect to the map dimension (see Eq. 4.2).

$$X_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \in [0,1] \times [0,1],\ i = 1, \ldots, N \tag{4.2}$$

In Fig. 4.3 the reference system that is adopted throughout the thesis is shown alongside coordinates of a 2 UAVs fleet.

4.1.4 UAVs' Field of View

It is assumed that UAVs move at a fixed flight altitude, thus their motion is limited to a 2D plane. Allowing altitude changes, the occurrence of collisions obviously

Algorithm 2 Maps creation procedure

for each map **do**
  Initialize: zero matrix $M \in \mathbb{R}^{100\times100}$, list of obstacles
  while obstacle occupancy $<$ desired obstacle occupancy **do**
    Randomly choose new obstacle dimensions d^* and center position c^*
    for obstacle in list of obstacles **do**
      if $||c^* - c_{\text{obst}}||_\infty \leq q \cdot \max\left(d_x^* + d_{\text{obst,x}}, d_y^* + d_{\text{obst,y}}\right)$ **then**
        break
      else if all obstacles were checked **then**
        Place new obstacle and add it to list of obstacles
      end if
    end for
  end while
end for

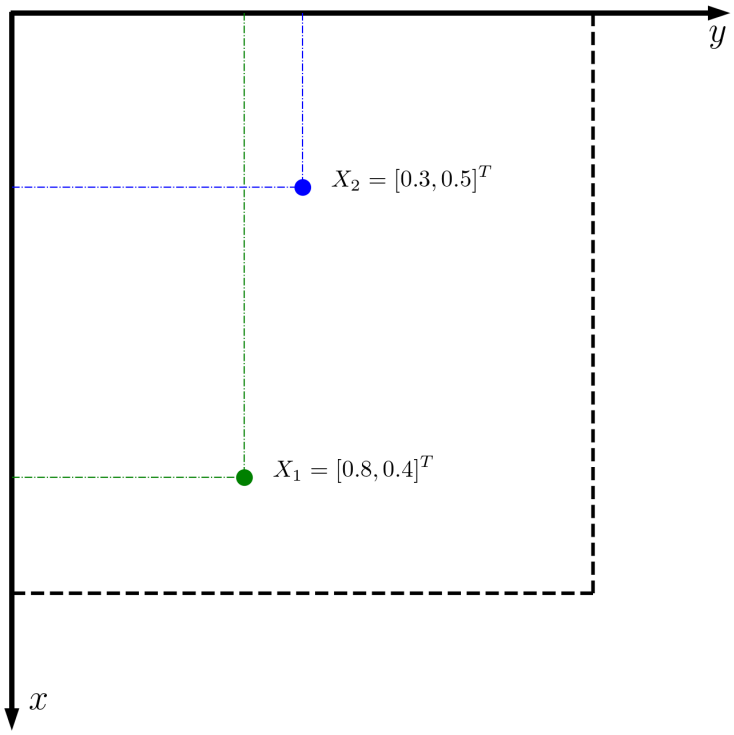


Figure 4.3: Adopted reference system, along with 2 UAVs coordinates in the map

reduces, but this more stringent requirement is anyway assumed. Reasonably considering that fleets share the same sensors and cameras, the observable area size

is the same for all fleet units, and it is considered to be a squared region. Intuitively, given that larger exploration areas should correspond to higher flight altitudes, the observable area dimensions are fixed in terms of number of cells, while their real dimensions scale with respect to the environment size. In all training simulations, fixed footprint dimensions of 11×11 cells are considered, while model performances are assessed in test simulations by varying them from 11×11 to 15×15 cells. The assumption of squared observable areas derives by considering that each UAV's *Field of View* (FOV) forms a pyramid with half-angle $\theta = \theta_1 = \theta_2$, as shown in Fig. 4.4.

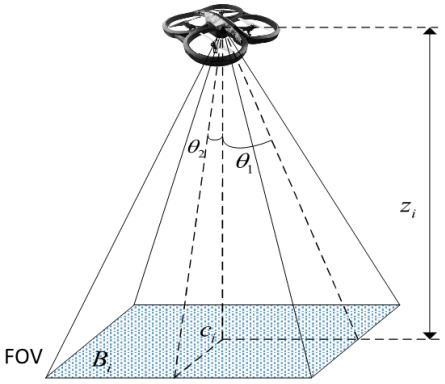


Figure 4.4: UAV's Field of View forming a pyramid with half-angles θ_1, θ_2 with ground. Image taken by [29].

With reference to Fig. 4.4, if $\theta_1 = \theta_2 = \theta$, the relation among footprint dimension f_{dim}, FOV half-angle θ and flight altitude z is reported in Eq. 4.3.

$$f_{dim} = 2z \tan \theta \tag{4.3}$$

Choosing an half-angle $\theta = 45°$, in Tab. 4.1 the flight altitudes matching with the analyzed footprint dimensions and plausible environment sizes are reported.

4.1.5 Obstacle Shape Prediction

Obstacles are detected by simulated range sensors, scanning the space all around the UAVs with a maximum detection range equal to half the footprint dimension. Given the impossibility of detecting whatever is beyond the obstacle surface, once UAVs communicate all information they gather during motion to the centralized storage, a *shape prediction algorithm* is run in order to predict the interior areas of the obstacles, undetectable by the fleet. The shape prediction is carried out thanks to scikit-image Python package [40].
Merging information about the drones location with obstacle detections, reconstructed map is firstly updated with the position of their countours, and then it

Map dimension [m]	$\mathbf{f_{dim}}[-]$	$\mathbf{f_{dim}}[\mathbf{m}]$	**Flight altitude** [m]
$10m \times 10m$	11×11	$1.1m \times 1.1m$	0.55
	13×13	$1.3m \times 1.3m$	0.65
	15×15	$1.5m \times 1.5m$	0.75
$50m \times 50m$	11×11	$1.1m \times 1.1m$	2.75
	13×13	$1.3m \times 1.3m$	3.25
	15×15	$1.5m \times 1.5m$	3.75
$100m \times 100m$	11×11	$1.1m \times 1.1m$	5.5
	13×13	$1.3m \times 1.3m$	6.5
	15×15	$1.5m \times 1.5m$	7.5
$500m \times 500m$	11×11	$1.1m \times 1.1m$	27.5
	13×13	$1.3m \times 1.3m$	32.5
	15×15	$1.5m \times 1.5m$	37.5
$1000m \times 1000m$	11×11	$1.1m \times 1.1m$	55
	13×13	$1.3m \times 1.3m$	65
	15×15	$1.5m \times 1.5m$	75

Table 4.1: Plausible flight altitudes compatible with footprint dimensions and map sizes, assuming $\theta = 45°$.

is processed so that unreachable locations are marked as occupied obstacle. The assumption of rectangular obstacles, as anticipated in (4.1.3), is functional to the use of this package; further adaptations would allow more complex shapes to be suitable for shape prediction. In Fig. 4.5 an example of the shape prediction algorithm applied to a local UAV observation is shown.

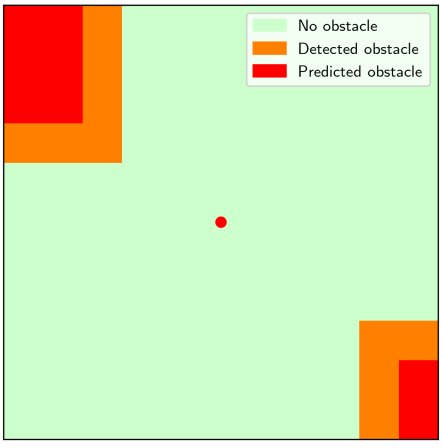


Figure 4.5: Shape prediction algorithm application to a local UAV observation.

4.1.6 Mixed Competitive-Collaborative Settings

With reference to Sec. 2.2, in which the different settings for multi-agent systems literature are reported, the coverage planning problem, as solved in this thesis, falls into *mixed cooperative-competitive settings*. In fact, the global task of exploring the entire environment is achieved through a competitive behavior learned by the UAVs, each one trying to maximize its coverage to the disadvantage of the others, while the mutual distancing and collision avoidance originates collaborative intents. This problem, which at a first glance can seem purely cooperative, is therefore solved developing competitive attitudes mixed with collaborative purposes (see Sec. 4.3 for a more detailed explanation, specifically in terms of reward function shaping).

4.2 General Settings

In order to solve the coverage planning problem effectively, two RL agents jointly collaborate in order to efficiently choose the motion direction while avoiding obstacles. The split of these tasks into two subproblems, taking inspiration from the typical Hierarchical Reinforcement Learning (HRL) settings [41], allows to solve the general problem by training the devoted agents independently and to assess their collaboration performances. Ideally, the overall problem can be even solved by using a learning agent devoted both to coverage planning and obstacle avoidance tasks, but to avoid misleading multi-objective reward functions, this hierarchical pattern is selected as more suitable to efficiently address both intents. Furthermore, since during exploration the discovered and predicted obstacles' positions are marked as already covered areas, the coverage planning agent alone already makes use of this information to suitably define a motion direction (see Sec. 4.3), which will be just refined by the obstacle avoidance agent (see Sec. 4.4). The two employed agents, which run locally on each fleet unit and acquire data from the shared centralized storage and processing unit, are specifically defined as:

- **Coverage agent**: On the basis of the fleet distribution and of the previously explored areas, encoded in a progressively updated coverage map, selects a temporary motion direction, which is passed to obstacle avoidance agent before being applied;
- **Obstacle Avoidance agent**: It merges information about the coverage agent's decision with local obstacle detections and outputs the definitive motion direction.

The interaction between the agents is depicted in Fig. 4.6, describing the UAV-environment interaction for a single fleet unit. Once the selected motions are

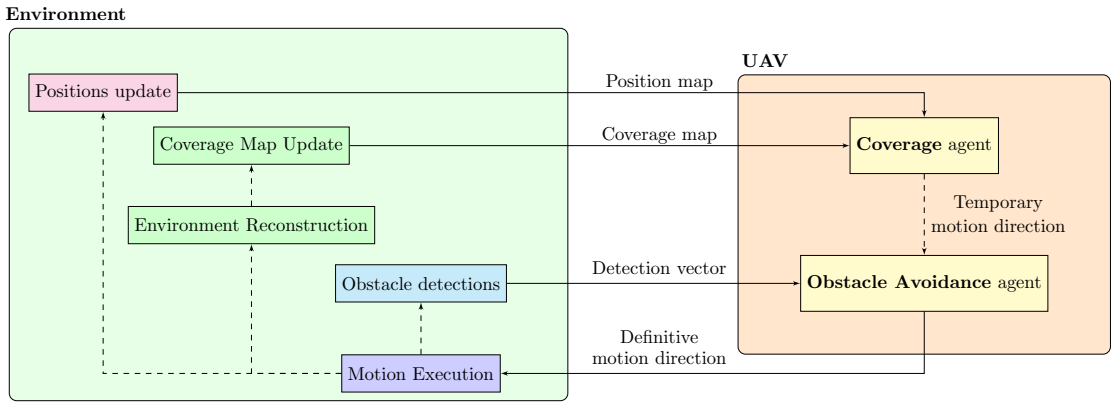


Figure 4.6: Single UAV interacting with the environment, outputting the selected motion direction as a result of input processing and agents' decision process.

executed by all UAVs, their positions are updated in parallel, their observations are fused for environment reconstruction and obstacles are detected in the new locations.

4.3 Coverage Agent

The coverage agent is the high-level decision maker, running on each UAV, in charge of selecting a suitable motion direction on the basis of the fleet distribution and of the already explored areas. The state and action spaces of the coverage agent are specified in (4.3.1). The training procedure of this agent, obtained with a modified version of the PPO algorithm, is outlined in (5.1.1).

4.3.1 State and Action spaces

The *state space* definition of the coverage agent comes from the need of taking into account contemporarily both the UAVs positions in the unknown environment and the coverage statistics at a given step. In the view of avoiding the input state dimension to modify as function of the fleet size, a suitable encoding of such information is employed, choosing as input *two stacked matrices*:

- **Position map**: It encodes the UAVs location in the map by means of a matrix $P \in [-1,1]^{100\times100}$, which is specifically processed for each UAV at each time step, so that the coverage agent can discriminate between the current UAV it is running on and the other ones of the fleet. Starting from a zero matrix, in correspondence to the current UAV a positive bi-variate normal distribution with mean value $\mu = (x, y)$ and covariance matrix $\Sigma = \sigma^2 I_2$ is added, with

$\sigma = 2.5$, whereas negative distributions with the same μ and Σ are added in correspondence to the other UAVs positions. In Eq. 4.4 the probability density function of a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ is reported, while in Fig. 4.7 the workflow of a position map construction is shown.

$$p(x, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp -\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \tag{4.4}$$

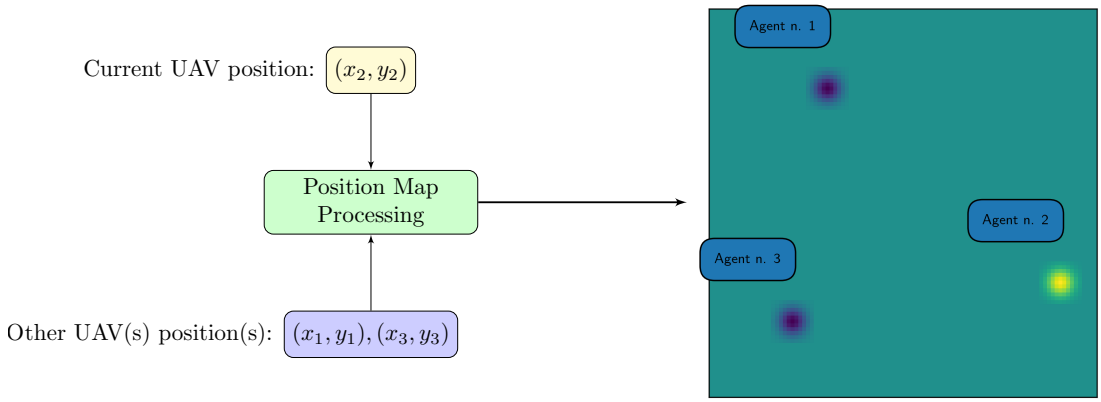


Figure 4.7: First layer (position map) of the input state to UAV n. 2, processed as function of the UAVs locations.

Obviously, the N processed maps at a given instant change in the sign of the gaussian bells only, as it is illustrated in Fig. 4.8, which displays the position maps processed at the same instant, for all members of a 3 UAVs fleet. A 3D representation of a position map is reported in Fig. 4.9b.

- **Coverage map**: It is a binary map $C_{\text{map}} \in \{0, 1\}^{100 \times 100}$ indicating the distribution of already explored areas of the environment during simulation.

$$[C_{\text{map}}]_{ij} = \begin{cases} 0, & \text{if cell } (i,j) \text{ has } \textit{not} \text{ been covered} \\ 1, & \text{if cell } (i,j) \text{ has been covered} \end{cases} \tag{4.5}$$

The coverage map is updated at each step by putting at 1 the cells corresponding to explored areas, i.e. the ones observed by the UAVs according to their trajectories and footprint dimensions (4.1.4). An example of coverage map is reported in Fig. 4.9a.

Coverage agent maps the input states to discrete actions, represented by integer numbers ranging from 0 to 8 to which motion directions are associated as reported in Tab. 4.2. While action 0 corresponds to no motion, actions $1-8$ correspond to

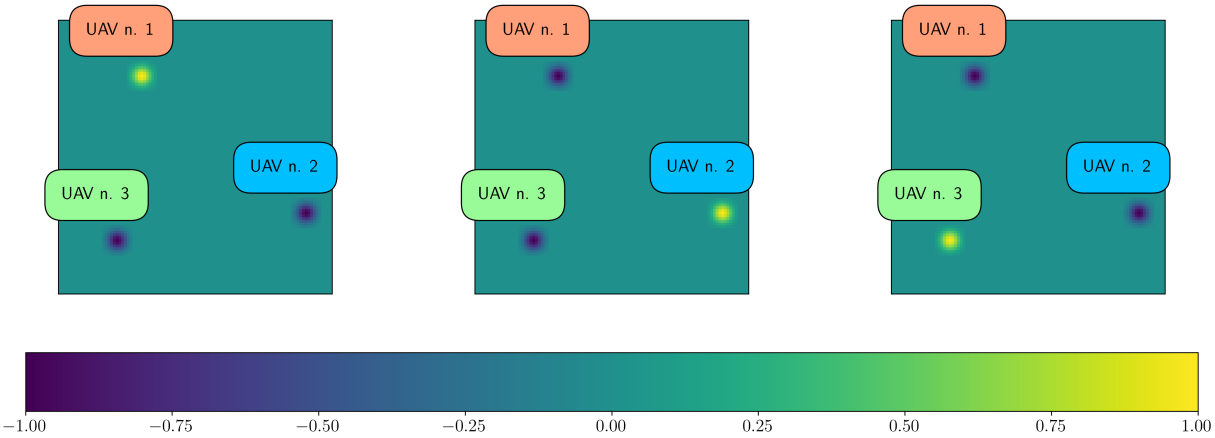
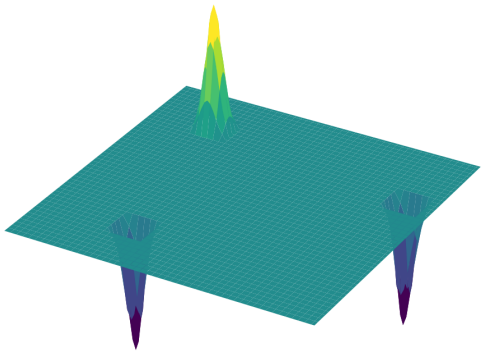


Figure 4.8: Position maps for a 3 UAVs fleet.

(a) Example of a coverage map.



(b) 3D representation of a position map

Figure 4.9

fixed-length displacements (of 4 cells, as a result of trial and error tuning) to the corresponding directions.
Actually, it is worth pointing out that these motion direction shall be intended as local waypoint assignments rather than direct velocity commands to be applied, to be reached with special purpose path-planning methods for smoother trajectories and attitude changes. More specifically, the coverage agent does not output directly the action number, but it returns $P = 9$ real values z_i for each action a_i (logits), which are then mapped to probabilities using the *softmax function*, as reported in Eq. 4.6. The action selection is then performed by sampling from this categorical distribution.

$$\text{prob}(a = a_j \mid s) = \frac{e^{z_j(x)}}{\sum_{i=0}^{P-1} e^{z_i(x)}} \tag{4.6}$$

Action number	Motion type	
0	Don't move	$\cdot$
1	Move up	$\uparrow$
2	Move down	$\downarrow$
3	Move left	$\leftarrow$
4	Move right	$\rightarrow$
5	Move up left	$\nwarrow$
6	Move up right	$\nearrow$
7	Move down left	$\swarrow$
8	Move down right	$\searrow$

Table 4.2: Correspondence between the discrete actions and motion directions selected by the coverage agent.

4.3.2 Actor and Critic models

Coherently with the chosen inputs and outputs of the coverage agent, as indicated in (4.3.1), CNNs are used as policy and value function approximators. In Fig. 4.10 a graphical representation of the selected architecture is illustrated.

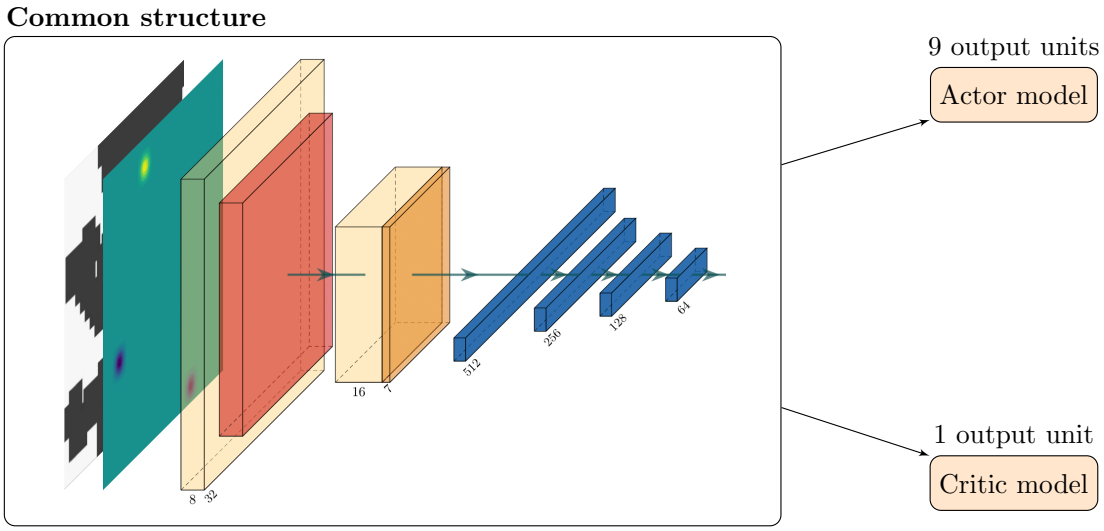


Figure 4.10: CNN structure for actor and critic models of the coverage agent.

Actor and critic models share the same structure until the output layer, consisting of $P = 9$ units for the actor and of 1 unit only for the critic model. The layered input state is firstly processed by 2 convolutional layers, the former followed by a

Max-Pooling layer and the latter by a batch normalization layer. A series of fully connected layers then follows, until the output layers. A more detailed description of the CNN structure is reported in Tab. 4.3, indicating layers, dimensions, activation functions and other additional hyperparameters.

Layer Type	**N. units/Filter dimension**	**Stride**	**Activation Function**
Convolutional	$8 \times (5 \times 5)$	(3,3)	ReLU
MaxPool	(3×3)	(2,2)	-
Convolutional	$16 \times (3 \times 3)$	(2,2)	ReLU
Batch Norm.	-	-	-
Flatten	-	-	-
FC	512	-	ReLU
FC	256	-	ReLU
FC	128	-	ReLU
FC	64	-	ReLU

Table 4.3: Detailed CNN common architecture of actor and critic models.

Weights and biases of the fully connected layers are initialized at the beginning of training process by means of He Normal initialization [42], which consists in randomly sampling initial weights from a normal distribution with mean value 0 and standard deviation $\sqrt{\frac{2}{n_l}}$, where n_l is the number of units of the preceding layer.

4.3.3 Reward Function

As anticipated in (4.1.6), the common objective of fully covering unknown environments is achieved in this thesis by stimulating a competitive approach among the fleet members: each one tries to maximize its own explored area, while keeping at a safety distance from the other fleet members. The reward is thus independently assigned to each UAV during training simulations as a function of its chosen action and consequent interaction with the environment. Before defining the chosen reward function, the *coverage percentage* and *individually increased percentage of covered area* are formalized in Def. 4.3.1 and Def. 4.3.2.

Definition 4.3.1 (Coverage percentage)**.**
The *coverage percentage* of an environment is the amount of already observed/predicted area with respect to the overall surface. It can be computed as function of the coverage map C_{map} by means of Eq. 4.7.

$$\nu = \nu(C) = \frac{\sum_{i=1}^{100} \sum_{j=1}^{100} C_{i,j}}{100 \times 100} \tag{4.7}$$

Definition 4.3.2 (Individually increased percentage of covered area)**.**
The *individually increased percentage of covered area* of the i-th UAV at a given time step t, henceforth denoted as $\Delta_i(t)$, is the increase in coverage percentage of environment that would be attained if the i-th UAV moved according to its selected action a_i and all other UAVs were kept fixed, i.e. $a_j = 0,\ j \neq i$. Denoting with $C_{\text{map,i}}$ the updated coverage map after that i-th agent moves and with $C^*_{\text{map}}(t)$ the initial coverage map at time t, before any motion, $\Delta_i(t)$ is reported in Eq. 4.8, while its computation procedure is outlined in Alg. 3.

$$\Delta_i(t) = \nu(C_{\text{map, i}}) - \nu(C^*_{\text{map}}(t)) \tag{4.8}$$

Algorithm 3 $\Delta_i(t)$ computation at time t

Temporarily store initial coverage map $C_{\text{map}}(t)$ and UAVs positions $x(t)$:

$$C^*_{\text{map}} \leftarrow C_{\text{map}}(t)$$
$$x^*_i \leftarrow x_i(t),\ i = 1, \ldots, N$$

for $i = 1, \ldots, N$ **do**
  Move i-th UAV according to action a_i and update C_{map}
  Compute $\Delta_i(t) = \nu\Big(C_{\text{map}}\Big) - \nu\Big(C^*_{\text{map}}\Big)$
  Restore i-th UAV position: $x_i(t) \leftarrow x^*_i$
  Restore coverage map: $C_{\text{map}}(t) \leftarrow C^*_{\text{map}}$
end for
Move all UAVs to new positions $x(t+1)$ according to selected actions
Update and store coverage map $C_{\text{map}}(t+1)$

To train agents in the achievement of these concurrent goals, a suitable reward function is defined as the sum of a *coverage* and *distribution* contribution, as in Eq. 4.9.

$$r = r_{cov} + r_{dist} \tag{4.9}$$

The *coverage contribution reward* r_{cov} is specified in Eq. 4.10; weighting $\Delta_i(t)$ through a coverage gain $K_\Delta = 100$.

$$r_{cov,i} = \begin{cases} K_\Delta \Delta_i, & \text{if } \Delta_i > 0 \\ K_\Delta \Delta_i + 10, & \text{if } \Delta_i > 0 \text{ and } \nu(C_{\text{map}}) > C_{th} \\ -0.1, & \text{otherwise,} \end{cases} \tag{4.10}$$

A positive reward, proportional to Δ_i is granted for actions providing a strictly positive coverage percentage increase, while a small negative reward (-0.1) is

provided if action doesn't add further explored areas. If UAV's action leads to a full coverage of the environment, achieved when $\nu(C_{\text{map}}) > C_{th}$, coverage agent receives an additional reward equal to (+10). C_{th} changes as function of the fleet size from 95% to 99%, further details will be provided in (5.1.1).
The *distribution contribution reward* r_{dist} is specified in Eq. 4.11. It is a function of the l_2-norm distances between couples of UAVs, computed as function of their positions X_i, $i = 1, \ldots, N$. A penalty is applied when mutual distance falls below a certain threshold.

$$r_{dist,i} = \begin{cases} -0.5, & \text{if } \exists j \neq i : \|X_i - X_j\|_2 \leq 0.1 \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

Notice that, for consistency and applicability to variable environment sizes, both positions and distance values are normalized. Obviously, in the case of larger environments the threshold can be lowered, but it is kept unchanged for all simulations to consider a more stringent condition.

4.4 Obstacle Avoidance Agent

The obstacle avoidance (OA) agent is the low-level decision maker in charge of selecting the definitive motion direction, once the temporary selection is outputted by the coverage agent and local obstacles are detected. A detailed description of state and action spaces of the OA agent are reported in (4.4.1), while for the detailed training procedure refer to (5.2.1).

4.4.1 State and action spaces

The state space definition of the obstacle avoidance agent comes from the need of fusing information about obstacle detections and the coverage agent selection. The input state is thus the concatenation of two arrays:

- **A priori action**: it is the action selected by the coverage agent, reported in one-hot encoding. Since there are $P = 9$ possible actions, this information is converted into an array with 9 entries, all zero except for the one corresponding to the selected action. All possible configurations are reported for completeness in Tab. 4.4.

- **Obstacle detections**: Obstacle detections are performed by means of simulated range sensors, displaced all around the UAVs, mutually shifted by 45°, with a detection range corresponding to half the footprint dimension. Obstacle detections are summarized in a binary array with 8 entries, assuming value:

Action	One-Hot Encoding
0	$[1,0,0,0,0,0,0,0,0]$
1	$[0,1,0,0,0,0,0,0,0]$
2	$[0,0,1,0,0,0,0,0,0]$
3	$[0,0,0,1,0,0,0,0,0]$
4	$[0,0,0,0,1,0,0,0,0]$
5	$[0,0,0,0,0,1,0,0,0]$
6	$[0,0,0,0,0,0,1,0,0]$
7	$[0,0,0,0,0,0,0,1,0]$
8	$[0,0,0,0,0,0,0,0,1]$

Table 4.4: One-Hot encoding conversion of the coverage agent action inputted to OA agent.

- 0: if no obstacle is detected along i-th direction;
- 1: if obstacle is detected along i-th direction.

Since UAVs move at the same flight altitude, obstacle detections are performed in the motion plane; this implies that other fleet members in the detection range are detected and marked as obstacles as well. A graphical representation of the detection directions is shown in Fig. 4.11.

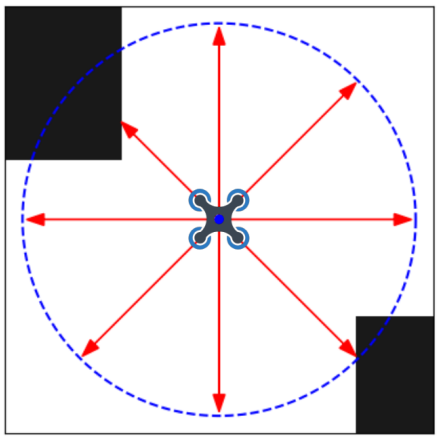


Figure 4.11: Obstacle detection directions. Numbering of detection directions follows the motion directions one in Tab. 4.2. The represented condition produces an obstacle vector $[0,0,0,0,1,0,0,1]$.

The action space of the obstacle avoidance agent resembles the coverage agent's one. In particular, the obstacle avoidance agent outputs the discrete action probabilities

just like the coverage agent does, in compliance with the action-motion direction correspondance already described in Tab. 4.2.

4.4.2 Actor and Critic models

As for the coverage agent, the obstacle avoidance actor and critic models share the same neural network architecture, except for the output layers, which consists of 9 units for the actor model, approximating the policy function, and of 1 unit for the critic model, approximating the state-value function as PPO algorithm is implemented for training process.
Being the input state a binary array, the neural network architecture is made up of a series of fully connected layers, with ReLU activation function. Weights initialization is performed in the same way as for the coverage agent architecture, using He Normal initialization (see 4.3.2). In Fig. 4.12 the neural network architecture of actor and critic models of the OA agent is shown, more detailed information are provided in Tab. 4.5.

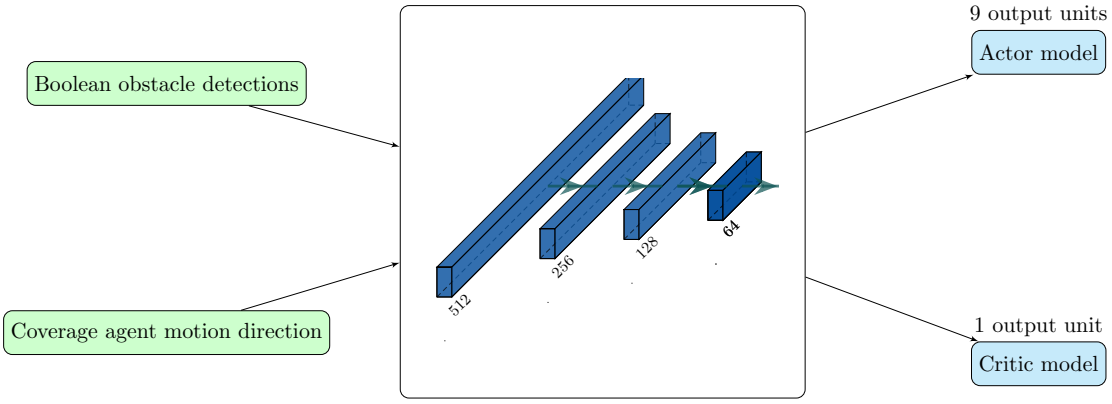


Figure 4.12: Graphical representation of the NN structure for actor and critic models of the obstacle avoidance agent, with common inputs and different outputs.

Layer Type	N. units	Activation Function
FC	512	ReLU
FC	256	ReLU
FC	128	ReLU
FC	64	ReLU

Table 4.5: Detailed NN common architecture of actor and critic models for the obstacle avoidance agent.

4.4.3 Reward function

Since the obstacle avoidance task is not influenced by the multi-agent settings, the training procedure of the OA agent is performed with a single UAV moving in the environment, as it is outlined in (5.2.1). The main objective of the obstacle avoidance agent is to confirm the action selected by the coverage agent whenever no obstacle is detected along the involved direction, on the contrast an action change is required when the coverage agent's one would lead to obstacle collision. In compliance with the assumption that OA agent is devoted to just refine the coverage agent's decision, it incurs greater penalties the farther the final position is from the original one, i.e. the one that would be attained by following the coverage agent's decision.
Defining:

- $\tilde{a}$, as the action selected by the coverage agent;
- a, as the action selected by the OA agent;
- $\tilde{X}$, as the position that would be attained following $\tilde{a}$;
- X, as the position that would be attained following a;
- $\tilde{o}, o$, as a binary flags indicating if along directions corresponding respectively to actions $\tilde{a}, a$ obstacles have been detected;

the reward function chosen for the OA agent is reported in Eq. 4.12.

$$r = \begin{cases} +1, & \text{if } a = \tilde{a} \text{ and } o = \tilde{o} = \text{False} \\ 1 - \frac{\|X - \tilde{X}\|_2}{\max_{X,\tilde{X}} \|X - \tilde{X}\|_2}, & \text{if } a \neq \tilde{a} \text{ and } o = \text{ False and } \tilde{o} = \text{ True} \\ -0.5, & \text{if } a \neq \tilde{a} \text{ and } o = \tilde{o} = \text{ False} \\ -1, & \text{if } \begin{cases} a = \tilde{a} \text{ and } o = \tilde{o} = \text{ True, or} \\ a \neq \tilde{a}, o = \text{ True} \end{cases} \end{cases} \tag{4.12}$$

The maximum reward (+1) is achieved when selecting the same action as the coverage agent, in an obstacle-free direction, while the minimum reward (-1) penalizes actions leading to obstacle collisions. A small negative reward is provided if the OA agent selects an action differing from the coverage agent's one, despite it would have not led to any collision. Finally, when OA agent selects $a \neq \tilde{a}$ because of obstacle detections, a decreasing reward with the increasing of the distance between X and $\tilde{X}$ is provided; the penalty is normalized with respect to the maximum possible value that $\|X - \tilde{X}\|_2$ can achieve, i.e. when a and $\tilde{a}$ correspond to opposite directions. This quite complex reward function allows to drive training towards selecting obstacle-free directions without discarding coverage planning purposes.

4.5 Reference UAV for energy consumption

In the view of a pratictal implementation of the designed algorithm, an energy consumption analysis of the fleet is carried out, so as to examine either the capability of completing the exploration task as well as residual autonomy once the environment is fully covered.

Figure 4.13: Front view of DJI Mavic 2 Pro [43], selected as reference UAV for energy consumption analysis.

The UAV selected as reference model for consumption evaluation is the DJI Mavic 2 Pro [43] (see Fig. 4.13), which is a commercial drone with 0.91 kg takeoff weight and a maximum (declared) flight distance of 18 km. It is worth specifying that this choice is merely related to its technical specifics, including energy consumption data which are employed for the presented analysis. This reference model may represent a varied class of UAVs employable for the implementation of the algorithm, for which it is reasonable to assume approximately the same autonomy as well as similar energy consumption. Among its specifics reported in Tab. 4.6 [44], the key parameter for the presented analysis is the *Energy per meter (Epm)* value, indicating the average burned energy per traveled distance.
In [44], a survey of energy consumption models for drones is proposed. As well as the contribution specifically related to horizontal motion, a variable amount of autonomy is burned because of *hovering*, *ascent* and *landing*. In the proposed approach, which is intended to provide just a rough estimate of the overall energy consumption, a linear autonomy decay with respect to the travelled distance only is assumed, as reported in Eq. 4.13, indicating with e the residual autonomy and s the travelled distance.

$$e(t) = e(0) - Epm \cdot s(t) \tag{4.13}$$

Drone Model	Mavic 2 Pro
Takeoff weight [kg]	0.91
Max Flight Time	31 min at 6.94 m/s
Max Hover Time [min]	29
Max Speed [m/s]	20
Max Flight Distance [km]	18
Battery Energy [kJ]	213.4
Energy per meter (Epm) [J/m]	11.9

Table 4.6: Specifics of DJI Mavic 2 Pro, a commercial drone [44] selected as the reference model for energy consumption analysis.

Further details and obtained results will be described in Chap. 6, in which model performance is assessed also in energetic terms.

Chapter 5

Agents training

This chapter is addressed to provide an in-depth analysis of the training processes of both coverage and obstacle avoidance agents, accompanied by a description of the selected hyperparameters and of the main learning curves and results.
In Sec. 5.1, which is devoted to the coverage agent training, the PPO algorithm adapted for multi-agent settings is reported and described, along with the learning performances, while in Sec. 5.2 the single-agent PPO implementation for the obstacle avoidance agent is analyzed.

5.1 Coverage agent

As already anticipated in Sec. 4.3, given the multi-agent settings affecting the coverage planning problem, a modified version of the PPO algorithm is implemented for training the devoted agent. In [19], one of the first implementations of the Multi Agent Proximal Policy Optimization (MAPPO) algorithm is applied to multi-agent games in either cooperative and competitive settings. However, such scenarios typically involve a fixed number of agents, with well-defined characteristics and goals. In this thesis, instead, because of the willingness to validate the presented approach for a wide range of fleet sizes, special multi-agent algorithms would have been impracticable in terms of complexity and applicability. In fact, most of the multi-agent actor-critic algorithms make use of an actor and (at least one) critic network for each agent under training, thus requiring at least $2N$ networks to be trained concurrently during the training procedure. With 10 UAVs fleets, 20 CNNs would have been trained concurrently at each learning epoch, requiring huge memory allocation and computational resources and consequently long training times. Furthermore, such an approach would be wasteful given the complete equality among UAVs in terms of performances and roles, since it would lead to train N policies that should ideally converge to a common one. In Fig. 5.1 the

typical Multi-Agent Actor-Critic framework is represented. Depending on the algorithm, the critic model can either approximate the state-value or action-value function.

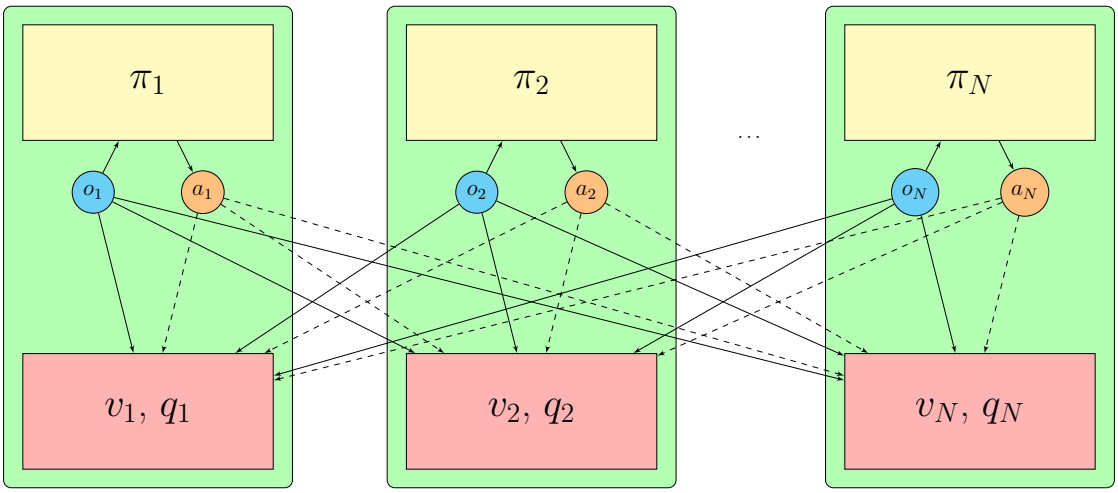


Figure 5.1: Multi-Agent Actor-Critic framework. In this picture it is shown how each policy π_i outputs action a_i as function of the observation o_i. Critic models estimate the value function by concurrently processing all agents' observations (and eventually all their actions in some algorithms).

Exploiting these simplifications, the single-agent PPO algorithm was modified so that each learning epoch is made up of trajectories collected by running the same version of the policy function on all agents, and all such trajectories are concurrently exploited as training experiences, useful for updating both policy π and value function v_π. The adopted framework is graphically represented in Fig. 5.2.

5.1.1 Training procedure

The training process of the coverage agent is repeated for each analyzed fleet dimension, specifically for $N = 2, 3, 4, 5, 6, 8, 10$. Before describing the training algorithm, it is worth providing a brief description of the main hyperparameters:

- *Epoch Length* T: It is the length of the training epochs, i.e. the number of experience tuples collected (for each agent) before a new policy and value function update. A learning epoch may consist of several episodes, depending on the general episode length and on the potential reaching of a terminal condition;
- *Maximum Episode Length* τ_{max}: It is the maximum number of steps that can

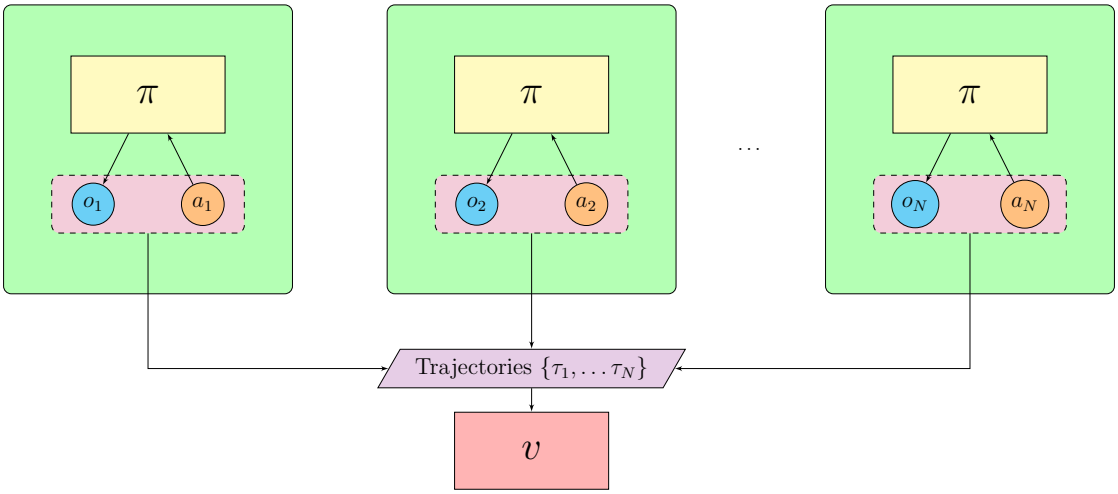


Figure 5.2: Modified PPO framework employed in this thesis. Multiple instances of the same policy π contribute in collecting trajectories, exploited for training both v and π at the end of each learning epoch.

be completed during an episode, that is reached if no terminal state occurs before;

- *Discount rate* γ: It weights the future rewards in the computation of the discounted return (see Eq. 3.2);
- λ: it is a tuning parameter for GAE, involved in the exponential moving average computation;
- α_{actor}: it is the learning rate of the stochastic gradient descent algorithm applied to update the actor model;
- α_{critic}: it is the same as α_{actor}, but applied to update the critic model;
- δ: It is the KL-divergence threshold (see Appendix B) involved in the constrained version of the PPO optimization problem, reported in Eq. 3.12;
- ϵ: It is the clip ratio, involved in the clipped objective function of the PPO algorithm, reported in Eq. 3.11;
- Q_a: it is the number of training epochs for the CNN approximating the policy function (actor model), i.e. the number of times in which forward and backward pass are applied to the entire buffer;
- Q_c: it is the same as Q_a but for the critic model (value function) training;

- *Coverage threshold* C_{th}: it is the percentage of the overall ground surface over which the environment is considered fully covered. A coverage percentage greater than or equal to the coverage threshold implies the achievement of a terminal state.
- *Footprint dimension* f_{dim}: it is the amount of area that each UAV can observe at a given step.

Some of the listed hyperparameters are fixed regardless of the number of UAVs and their values are reported in Tab. 5.1, while some others are the result of trial and error tuning made with respect to the fleet size and they are listed in Tab. 5.2.

Hyperparameter Symbol	**Description**	**Value**
T	Maximum epoch length	Variable
τ_{max}	Maximum episode length	Variable
γ	Discount rate	0.99
λ	Tuning parameter for GAE	0.97
α_{actor}	Actor learning rate	1e-4
α_{critic}	Critic learning rate	1e-3
Q_a	Actor training iterations	Variable
Q_c	Critic training iterations	Variable
δ	KL Divergence threshold	0.015
ϵ	Clip ratio	0.2
C_{tr}	Coverage threshold [%]	Variable
f_{dim}	Footprint dimension	11×11 cells

Table 5.1: List of training hyperparameters for Coverage Agent. Corresponding value is provided if common to all fleet dimensions, refer to Tab. 5.2 instead.

It is noticeable how the main optimization-related hyperparameters, except for Q_a and Q_c, do not change as function of the fleet size. This is also a consequence of PPO algorithm's data efficiency and robustness [38], with a reduced need of hyperparameters tuning. Modification of the episode length is needed, since lower size fleets intuitively need higher exploration times for reaching full coverage than larger ones may require. Moreover, C_{th}, selected as 99% for all fleets with more than 3 UAVs, is lowered for 2 and 3 UAVs fleets, respectively to 95 and 97 %, because of the difficulties encountered in completing explorations with insufficient fleet sizes within constrained number of steps. In Alg. 4 the coverage agent training procedure is described in detail.
It is important to highlight how some training episodes may be executed partly in one epoch and partly in the following one. Epochs consist of several episodes,

N	**T** [n. steps]	τ [n. steps]	$\mathbf{Q_a}$	$\mathbf{Q_c}$	$\mathbf{C_{th}}$ [%]
2	1200	600	3	3	95
3	2000	400	10	10	97
4	2000	400	10	10	99
5	1800	300	10	10	99
6	1800	300	10	10	99
8	1800	200	10	10	99
10	1800	200	10	10	99

Table 5.2: Training hyperparameters reported as function of the fleet size N.

Algorithm 4 Coverage Agent Training - General procedure for N agents

```
Initialize: Actor and Critic parameters θ_0 and φ_0 and networks π_θ0 and v_φ0
for k = 1, 2, ... do
    Initialize N buffers
    for t = 1, ..., T do
        if no episode is running then
            Reset the environment and start new episode
            Place agents randomly in the map
        end if
        for agent i = 1, ..., N do
            Sample action a_i(s_i) using policy π_k = π(θ_k)
            Move to next state s'_i and compute reward r_i
            Store experience in i-th agent's buffer
        end for
    end for
    for agent i = 1, ..., N do
        Load i-th agent's buffer
        Compute rewards-to-go R̂_t and advantage estimates Â_t
        Update π_{k+1} consistently with Eq. 3.12
        Fit the value function by regression on mean squared error, as in Eq. 3.13
    end for
end for
```

this allows to exploit experiences collected at different times of the exploration process and enables a greater forecasting ability. The policy update procedure, which is performed consistently with the optimization problem in Eq. 3.12, is outlined in Alg. 5, describing the process occurring at the end of each training epoch. Specifically, policy is updated with Q_a cycles of forward and backward

passes, unless the KL-divergence computed between policies of consequent training iterations overcomes the selected threshold δ, as reported in Alg. 5.

Algorithm 5 Coverage Agent Training - Policy Update

$\pi_{k+1,0} \leftarrow \pi_k$
for $j = 1, 2, \ldots, Q_a$ **do**
  Update $\pi_{k+1,j}$ through SGD to maximize PPO objective (Eq. 3.11)
  if $KL(\pi_{k+1,j-1}, \pi_{k+1,j}) \geq \delta$ **then**
    break
  end if
end for

5.1.2 Training results

The learning process, repeated for fleet sizes N = 2,3,4,5,6,8,10, is evaluated in terms of the overall returns gained per episode and of coverage and distribution statistics. In order to analyze the learning curves obtained for variable N, the episodic returns collected by the agents are averaged over the number of UAVs in the simulated episodes, thus introducing the *Averaged Episodic Return* (AER), which is defined in Def. 5.1.1.

Definition 5.1.1 (Averaged Episodic Return)**.**
The Averaged Episodic Return (AER) is the sum of the rewards collected by each agent during the episode, averaged over the number of agents. Let T be the length of a generic training episode, its AER is defined as in Eq. 5.1.

$$\text{AER} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} r_i(t), \tag{5.1}$$

where $r_i(t)$ is the i-th agent's reward collected at time t.

In Fig. 5.3 the AER for all training episodes is plotted against the episode number, counted starting from the beginning of the training process.
It is noticeable how larger fleets are affected by lower steady state values of the AER. This is in accordance with the reward function that is chosen for the coverage agent: the coverage component of the reward is linked to the individual contribution in exploration. Obviusly, an increasing number of UAVs results in a decrease of the explorable portion of the environment for each fleet unit, entailing a reduced individually gainable reward. In addition, an increased population involves higher probabilities of mutual collisions or unsafe mutual approaches. All learning curves show a convergence after about 8×10^3 episodes, after which policy is stable and does

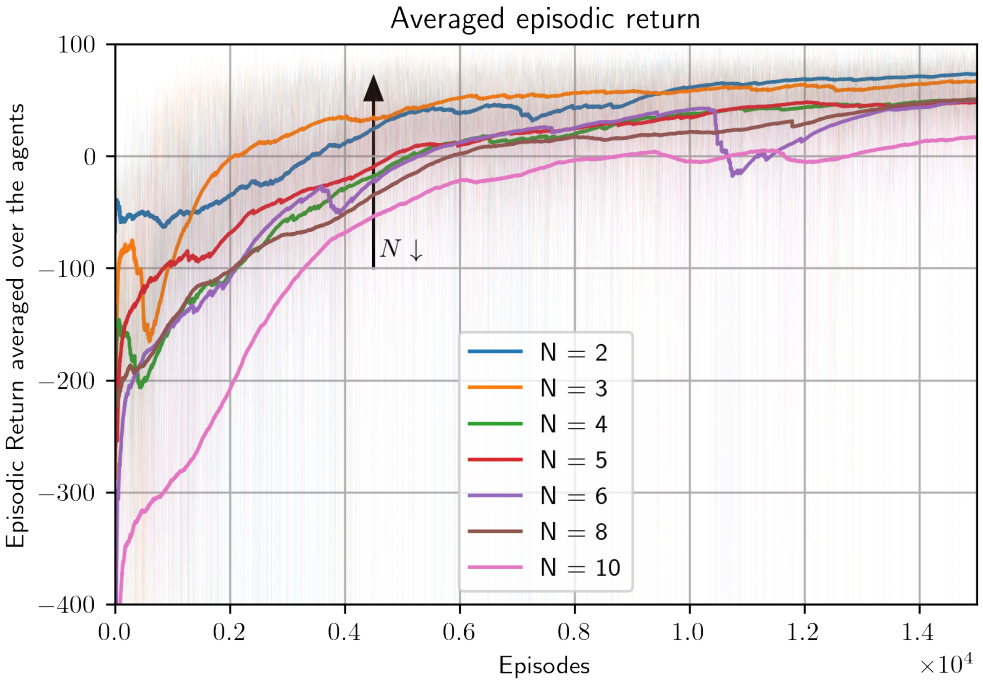


Figure 5.3: Filtered (by means of EMA) Averaged Episodic Return (AER) during the learning process, for different fleet sizes $N = 2,3,4,5,6,8,10$.

not show further remarkable enhancements. The learning phase is characterized by initial policy divergence from the randomly initialized one, after which training proceeds smoothly until convergence.

Even though the AERs trends provide an elegant validation and proof of objective accomplishment, it is nonetheless necessary to analyze the learning process performances by means of specific coverage and distribution statistics gathered during the simulated episodes. In Fig. 5.4, the percentage of explored surface at the end of each training episode is reported during the learning process.

The curves show consistency with the coverage thresholds reported in Tab. 5.2. During the first training phase, the untrained fleets don't manage to reach the coverage threshold target within the maximum episode length. Thanks to policy learning, in a very reduced number of episode, convergence is achieved, with faster learning rates for bigger fleets. Since the trends resemble logarithmic behaviors, a Least-Squares (LS) fitting is performed, by assuming an approximating function $\hat{\nu}$ of the training episode e, resulting in a polynomial plus logarithmic approximation, as specified in Eq. 5.2. The approximated trends are reported in Fig. 5.4 using

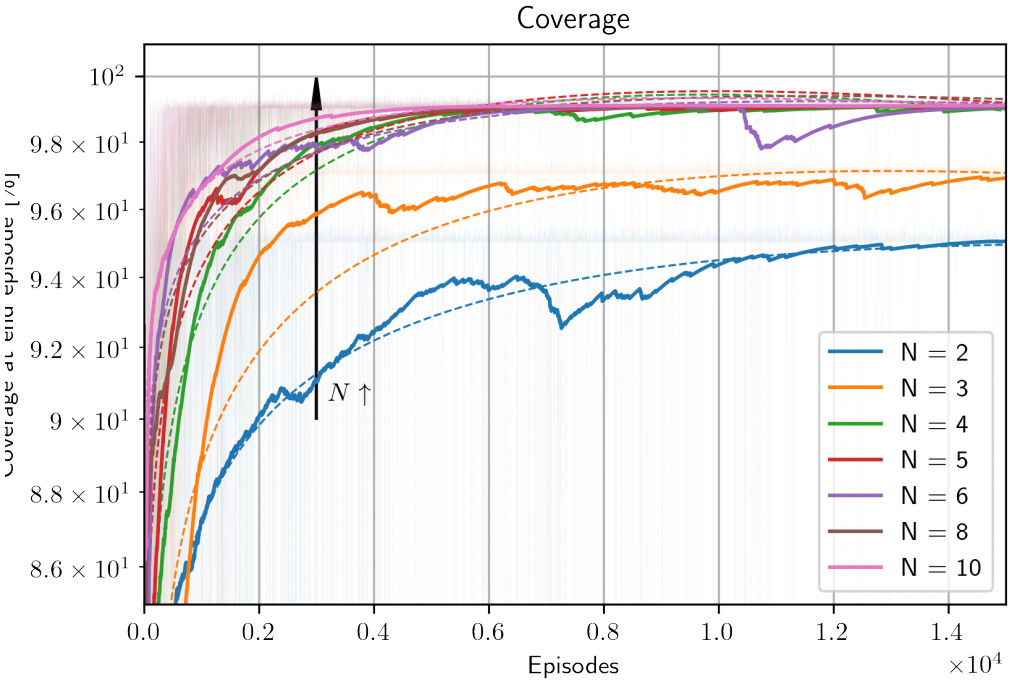


Figure 5.4: Filtered (by means of EMA) coverage percentage at the episode end (semi-logarithmic plot) during the learning process, for different fleet sizes $N = 2,3,4,5,6,8,10$. Dashed lines result by LS approximation, as indicated in Eq. 5.2.

dashed lines.

$$\hat{\nu}(e) = a_0 + a_1 e + a_2 log(e) \tag{5.2}$$

The LS estimates of $a = \begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix}^T$ are reported in Tab. 5.3.
The first episodes of the training process, characterized by a limited coverage percentage as reported in Fig. 5.4, end because the maximum episode length τ_{max} is exceeded before reaching a terminal state, attained when $\nu(C_{\text{map}}) \geq C_{th}$. Despite the fast convergence of the final coverage percentage during training, the episode length continues to decrease even after, as a result of the policy improvement reflected in Fig. 5.3.
In fact, in Fig. 5.5 the episodic length over the training episodes is plotted, for all fleet sizes. The starting values correspond to the maximum episode lengths, as reported in Tab. 5.2. As expected, the episode length decreases with larger fleets, since they manage to reach the target coverage percentage in lower exploration times.

N	a_0	a_1	a_2
2	5.89e+01	-2.45e-04	4.13e+00
3	5.35e+01	-3.98e-04	5.15e+00
4	6.10e+01	-4.8e-04	4.69e+00
5	6.87e+01	-3.76e-04	3.76e+00
6	7.84e+01	-2.10e-04	2.48e+00
8	7.65e+01	-2.34e-04	2.74e+00
10	8.27e+01	-2.11e-04	2.03e+00

Table 5.3: Least Squares estimates of the convergence trends parameters, applied to $N = 2,3,4,5,6,8,10$.

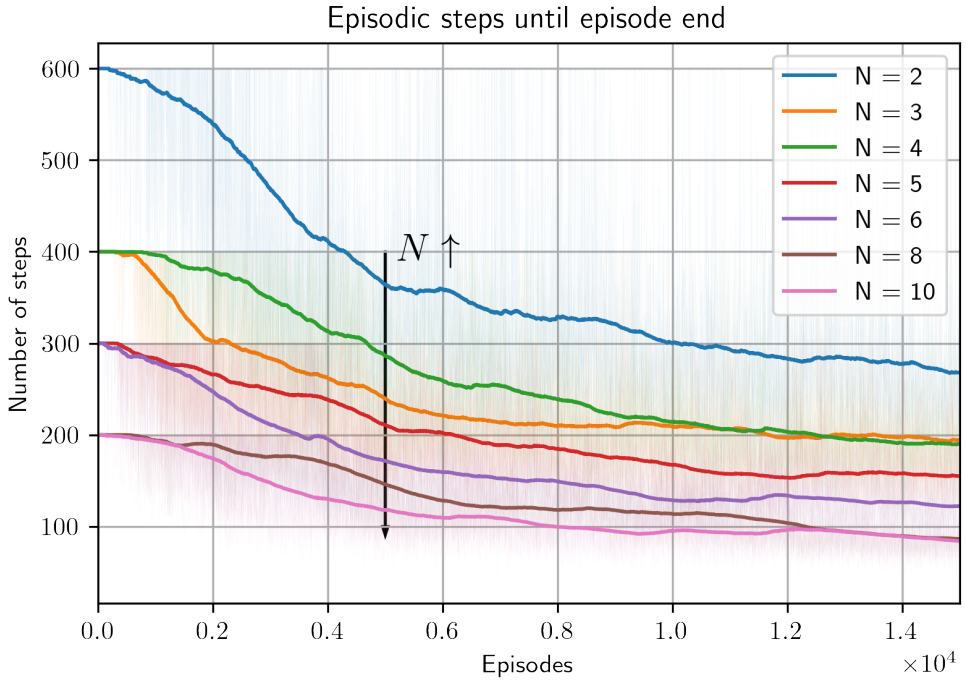


Figure 5.5: Filtered (by means of EMA) episodes lengths during the learning process, for different fleet sizes $N = 2,3,4,5,6,8,10$.

Alongside the reported metrics, policy improvement reflects in terms of strategic distribution as well. Spreading efficiency in the environment is evaluated by means of two main metrics: the *average minimum distance* (AMID, see Def. 5.1.2) and the *average mutual distance* (AMUD, see Def. 5.1.3).

Definition 5.1.2 (Average Minimum Distance)**.**
The Average Minimum Distance (AMID) of an episode is the mean over its length τ of the minimum distance between couples of UAVs, recorded at each step. Its formulation is reported in Eq. 5.3 denoting with $X \in [0,1] \times [0,1]$ the generic UAV position.

$$\text{AMID} = \frac{1}{\tau} \sum_{t=0}^{\tau} \min_{i \neq j} \|X_i(t) - X_j(t)\|_2 \tag{5.3}$$

Definition 5.1.3 (Average Mutual Distance)**.**
The Average Mutual Distance (AMUD) of an episode is the mean over its length T of the average value of distances among couples of UAVs, recorded at each step. Its formulation is reported in Eq. 5.4 denoting with $X \in [0,1] \times [0,1]$ the generic UAV position.

$$\text{AMUD} = \frac{1}{\tau} \sum_{t=0}^{\tau} \frac{1}{N(N-1)} \sum_{i \neq j} \|X_i(t) - X_j(t)\|_2 \tag{5.4}$$

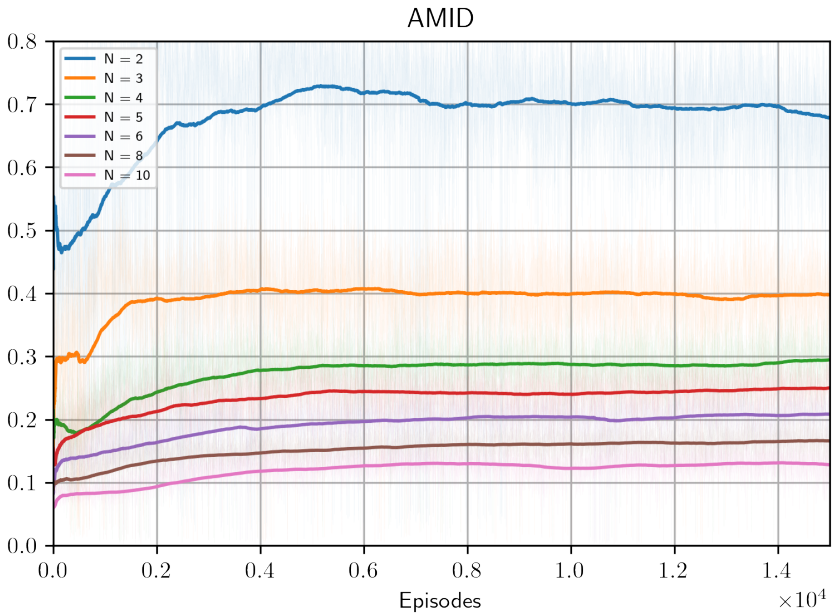


Figure 5.6: Filtered (using EMA) AMID trends over the training process, reported for $N = 2,3,4,5,6,8,10$.

The AMUD gives indication about the effectiveness of the coverage planning algorithm in keeping drones appropriately spaced to promote parallel exploration, while the AMID takes into account how much the algorithm is effective in avoiding critical

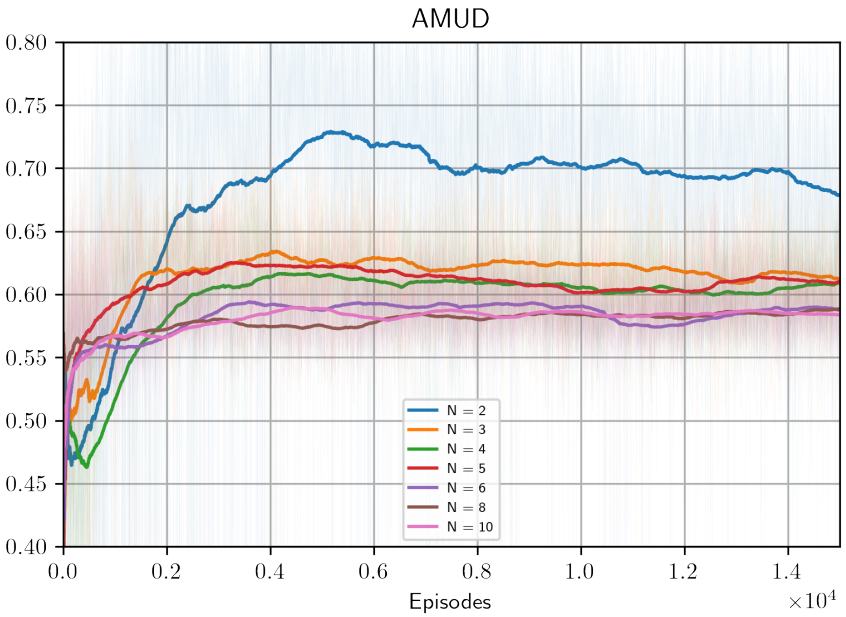


Figure 5.7: Filtered (using EMA) AMUD trends over the training process, reported for $N = 2,3,4,5,6,8,10$.

approaches. It is worth highlighting that, since the positions X are normalized, both AMID and AMUD are invariant with respect to the environment size.
In Fig. 5.6 and 5.7, AMID and AMUD for the training episodes are reported, for all fleet sizes. Both metrics are characterized by a fast improvement in the very initial phase of training, with their values rapidly increasing before converging. AMID changes consistently as function of the fleet size, nevertheless even for the largest fleets ($N = 10$), learning process allows to exceed the critical value of distance (0.1) under which agents are penalized during the process. Concerning AMUD, besides showing a more remarkable enhancement during the first learning phase, it is of special relevance the fact that even the largest fleets manage to plan coverage with an efficient distribution although their increase in number.

5.2 Obstacle Avoidance Agent

The obstacle avoidance agent, being just a low-level decision maker refining each UAV's temporary motion, is trained and acts according to local information about the current UAV only. In compliance with its local effects, the training procedure of the obstacle avoidance agent is performed in single-agent settings, simulating a single UAV moving in random environments as the ones described in (4.1.3)

with variable obstacle occupancies. The training is performed by exploiting of the original version of the PPO algorithm (see Sec. 3.3), defining the state and action spaces as in (4.4.1) and in compliance with the reward function defined in Eq. 4.12. The training settings for the OA agent shall be such to expose it to all feasible conditions it can face during the test condition. Specifically, for this purpose using the trained coverage planning as high-level action generator during the OA agent training would be ineffective, since any biased behaviour may reflect in the impossibility of facing all plausible conditions. That's why in this training process, while obstacle detections are properly simulated by means of range sensors, the high-level motion direction, which in test conditions is the outcome of the coverage agent, is generated randomly, by sampling from a uniform probability distribution U over all possible actions. This ensures that, in terms of the input state component related to the temporary motion direction, all possible motion directions are faced with asymptotically identical frequencies and random detections are generated by randomly displaced obstacles in the environment. Terminal state of episodes correspond to obstacle collisions or to exceeding the maximum episode length. In Fig. 5.8, this framework with random direction selection is schematized, while in Fig. 5.9, the uniform probability distribution U for the possible input actions is displayed.

5.2.1 Training procedure

The training procedure of the OA agent is tuned by means of hyperparameters selected by trial and error, whose values are reported in Tab. 5.4. Given that OA agent is trained by means of PPO algorithm as for the coverage one, a detailed description of their meaning is already reported in (5.1.1). Notice that the maximum episode length is set to 300 steps: an optimal policy should be able to avoid detected obstacles for the entire episode, early stopping occurs only in the case of collisions. Unintended ending before reaching maximum length implies a reduction of both the episodic return (see Eq. 3.1) and of its discounted version (see Eq. 3.2).
In Alg. 6 the training procedure of the OA agent is reported, performed by means of PPO algorithm.

5.2.2 Training results

The learning statistics of the obstacle avoidance agent refer mainly to episodic return curves. In Fig. 5.10 the episodic return, computed as the sum of collected rewards during each episode ($\sum_{t=0}^{\tau} r(t)$), is plotted for the overall learning process. In about 3000 episode, the episodic return shows policy convergence to a steady state value which is slightly lower than the maximum achievable 300. Actually, it is important to take into account that the maximum attainable episodic return

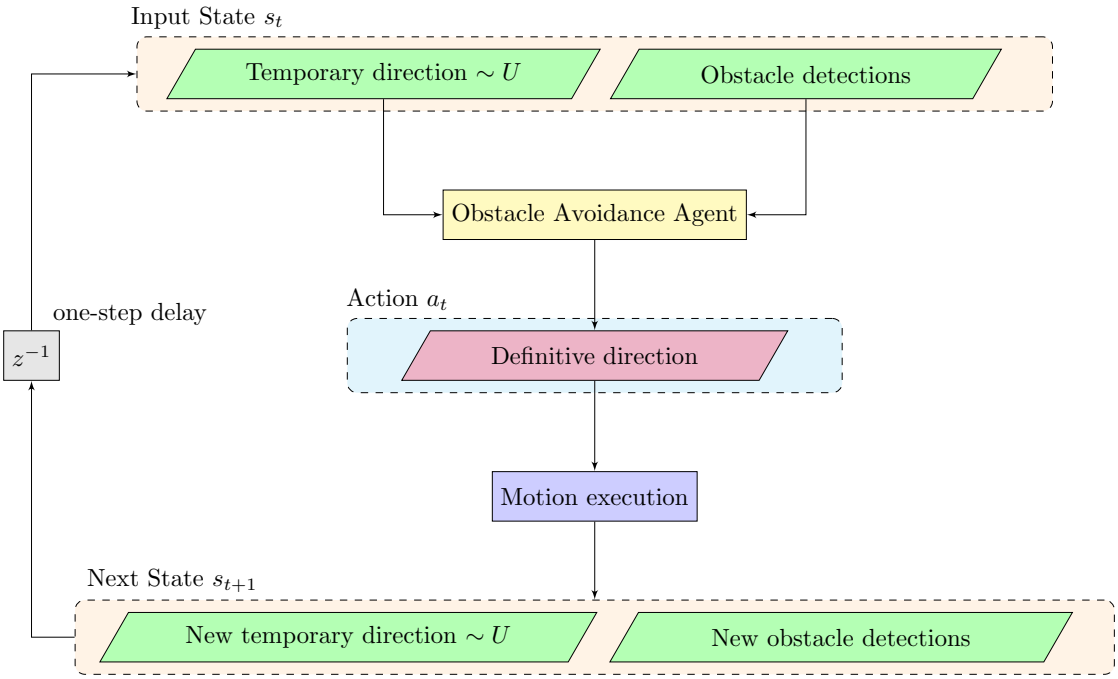


Figure 5.8: Obstacle Avoidance Agent's inputs and outputs during training. z^{-1} represents the one-step delay block. Temporary directions are always sampled by uniform distributions U (as reported in Fig. 5.9).

Hyperparameter Symbol	Description	Value
T	Maximum epoch length	1500
τ	Maximum episode length	300
γ	Discount rate	0.9
λ	Tuning parameter for GAE	0.97
α_{actor}	Actor learning rate	3e-4
α_{critic}	Critic learning rate	1e-3
Q_a	Actor training iterations	20
Q_c	Critic training iterations	20
δ	KL Divergence threshold	0.01
ϵ	Clip ratio	0.2

Table 5.4: List of training hyperparameters for Obstacle Avoidance Agent.

can be achieved only when no obstacle is present, since a unitary reward (see reward function in 4.12) is provided at each step (the OA agent simply confirms the coverage agent's decision). When obstacles are present, even selecting the best

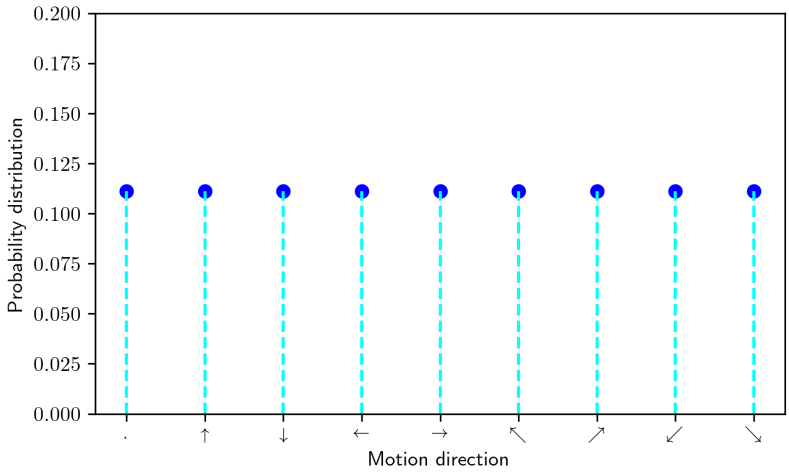


Figure 5.9: Uniform Probability Distribution U over the action space. Each possible motion direction (among the ones reported in Tab. 4.2) is sampled with probability $p = 0.11$.

Algorithm 6 Obstacle Avoidance Agent Training Process

Initialize: Actor and Critic parameters θ_0 and ϕ_0 and networks π_{θ_0} and v_{ϕ_0}
for $k = 1, 2, \ldots$ **do**
  Initialize agent's buffer
  for $t = 1, \ldots,$ T **do**
    if no episode is running **then**
      Randomly pick an environment map from training set
      Place agent in a random position
    end if
    Sample temporary motion direction $\tilde{a} \sim U$
    Detect obstacle by simulated range sensors
    Concatenate to build the input state s_t
    Sample definite motion direction $a \sim \pi_{\theta_k}(s_t)$
    Move agent according to a and compute reward r_{t+1}
    Store experience in the buffer
  end for
  Compute rewards-to-go $\hat{R}_t$ and advantage estimates $\hat{A}_t$
  Update π_{k+1} consistently with Eq. 3.12
  Fit the value function by regression on mean squared error, as in Eq. 3.13
end for

action allowing to move to the nearest obstacle-free direction, the attained reward will be surely < 1 even though the best possible selection is made. This implies that, except for the (extremely rare) case in which no obstacles are detected for the entire episode, the episodic return will be necessarily < 300. When convergence is achieved for the OA agent, episodes end after reaching their maximum length, thus validating the trained agent's effectiveness.

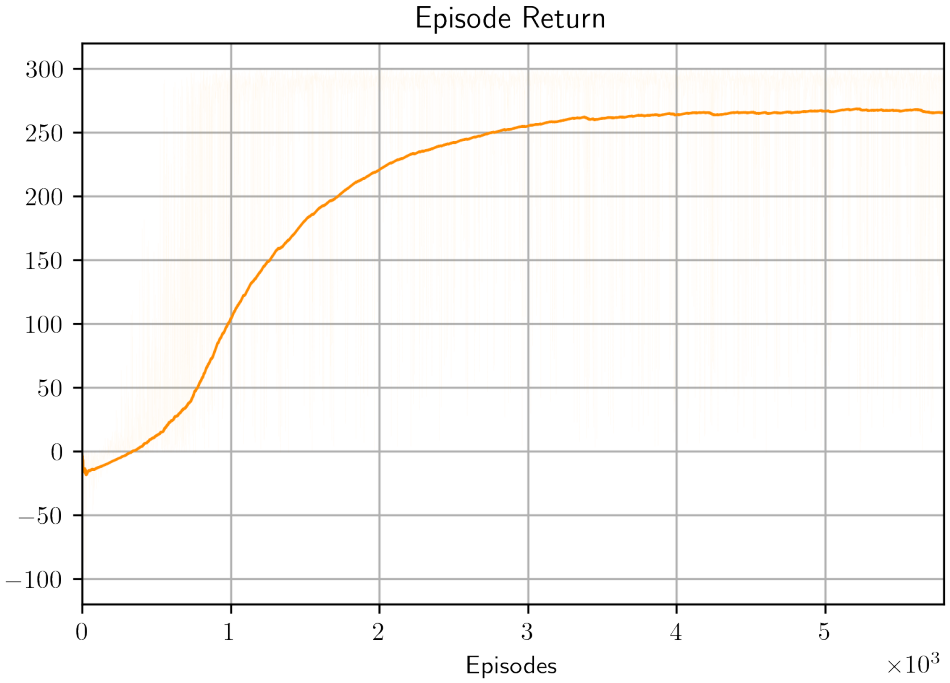


Figure 5.10: Filtered (using EMA) learning curve of the obstacle avoidance agent, showing episodic returns as function of the number of episodes from the beginning of the simulation.

Chapter 6

Simulations and test

In this chapter the performance of the proposed methodology is evaluated by means of several test simulations, in which the combined behaviour of the coverage and obstacle avoidance agents is analyzed. The aim of this chapter is mainly to prove the effectiveness of trained agents in solving the coverage planning problem, and thus to validate the training approach proposed in Sec. 5.1. The simulations make use of randomly generated environment, tests are carried out by testing agents' performance on a test set consisting of 300 test maps with different complexities, allowing to gather large quantities of simulation data, that are then exploited for analysis. Some maps are then chosen as reference, to provide a detailed analysis of the fleet behavior in plausible operating environments. In Sec. 6.1, the test procedure is explained in detail, in Sec. 6.2 the employed metrics for performance assessment are described, in Sec. 6.3 the main results obtained on the large test set of environments are presented, in Sec. 6.4 some selected case studies are analyzed and in Sec. 6.5 statistics of an exploration example are described.

6.1 Simulation algorithm

The test simulations are carried out loading the trained models of coverage and obstacle avoidance agents (i.e. the trained ANN), and selecting actions by means of forward passes of input data across the networks. Training process returns a collection of weights and biases for the policy networks (and for the value functions as well), saved in h5 data files that are then loaded in test phase, during which no change of such parameters is performed. A schematic representation of these different settings is shown in Fig. 6.1.
In order to test the fleet performance in more stringent conditions, at the beginning of each test simulation, all UAVs are positioned in a little portion of the map. This allows to evaluate their capability in spreading in spite of this unfavourable starting

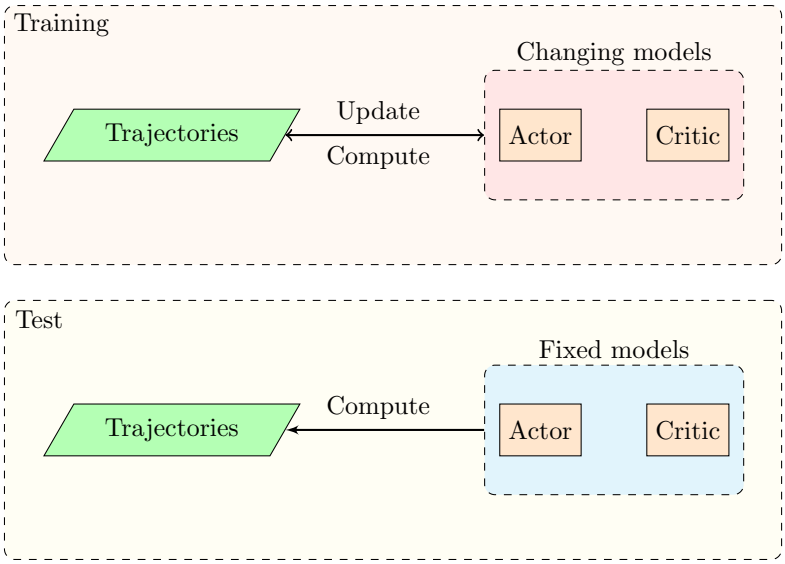


Figure 6.1: Schematic representation of the training and test procedures. In training phase, models are used to compute trajectories through which they are updated in turn. In test phase, fixed models are used to compute the trajectories, without any further update.

condition, which is graphically shown in Fig. 6.2.

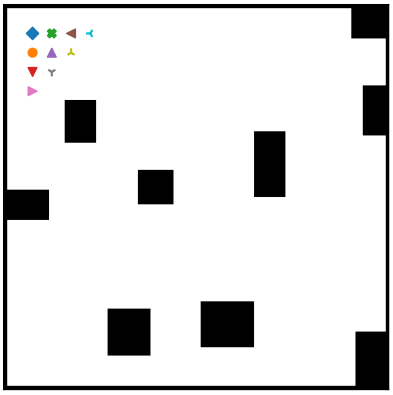


Figure 6.2: Representation of the initial fleet distribution in a test map.

Once the environment is initialized, positions and initial observations are sent to a centralized storage and processing unit, which provides, at each step, useful information to all fleet units. From the collected trajectories, various results are

extrapolated, in compliance with the metrics of interest, reported in Sec. 6.2. In Fig. 6.3, a flowchart describes the test algorithm, summarizing the main steps involved in this phase. A more detailed description of this procedure is presented in Alg. 7.

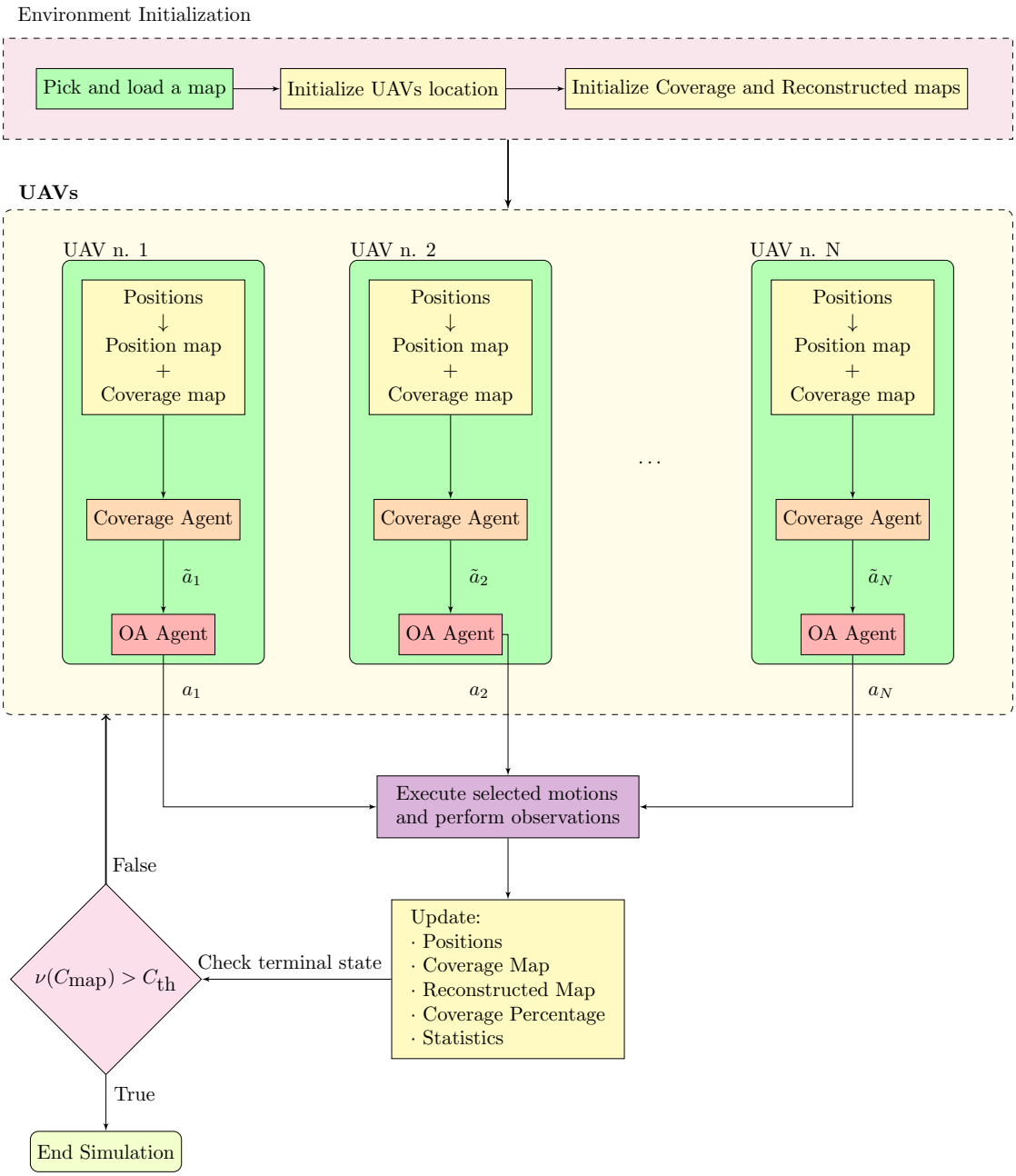


Figure 6.3: Flowchart of the simulation test algorithm.

Algorithm 7 Simulation Algorithm

Pick and load an environment map for testing
Initialize UAVs location according to a tree-shaped structure
Initialize C_{map} and reconstructed environment data
while $\nu(C_{\text{map}}) < C_{\text{th}}$ **do**
  for agent $i = 1, \dots N$ **do**
    From positions create i-th UAV's position map
    Stack position and coverage maps $\Rightarrow \tilde{o}_i$
    Sample temporary motion direction $\tilde{a}_i$ (*coverage agent*)

$$\tilde{a}_i \sim \pi_{\text{coverage}}(\tilde{a}|\tilde{o}_i)$$

    Concatenate one-hot encoding of $\tilde{a}_i$ with obstacle detections $\Rightarrow o_i$
    Sample definitive motion direction a_i (*OA agent*)

$$a_i \sim \pi_{\text{oa}}(a|o_i)$$

  end for
  Move all UAVs according to selected actions
  Update positions
  Observe the environment from current positions
  Update coverage map C_{map}
  Update reconstructed map with gathered information
  Update statistics
end while
End simulation

Once the environment is initialized and all UAVs are placed, the exploration procedure goes until the coverage percentage of the environment, computed as $\nu(C_{\text{map}})$ (see Eq. 4.7), exceeds the threshold C_{th}. Its value, variable with respect to the fleet size, corresponds in test phases to the one already reported for the training procedure (see Tab. 5.2). At each time step, each UAV processes the (normalized) positions of the fleet and creates its position map, which is then stacked to the coverage map (common to all fleet members). Agents' policies are invoked to compute action probabilities distributions, from which actions are sampled. After the fleet moves, coverage map is updated, taking into account the new explored areas, while exploiting obstacle detections and shape prediction algorithm (see (4.1.5)) a sensed version of the environment map is build. An alternative approach would perform action selection by selecting the one associated to the maximum probability instead of sampling it from the policy distribution, as reported in Eq.

6.1.

$$a = \max_{a} \pi(a|s) \tag{6.1}$$

Actually, despite at the beginning of the training process the output distributions are quite flat, trained models output low entropy ones, thus the two approaches would converge to the same results. The sampling approach is actually preferred since, in the case where multiple actions are suggested with approximately equal probability, it adds randomness to the exploration process.

6.2 Performance evaluation metrics

The choice of evaluation metrics for performance is made taking into account the most influential aspects for a practical implementation of the proposed algorithm. Those metrics include the exploration time needed for achieving full coverage (reported in number of steps τ) as well as distribution-related parameters including the average minimum distance (AMID) and average mutual distance (AMUD), already mentioned in training statistics (see Def. 5.1.2 and 5.1.3). Moreover, data gathered during simulation tests is exploited to provide an energy consumption analysis, which is mostly based on the on-purpose defined *efficiency parameter* $\chi(\alpha)$, whose detailed definition is provided in (6.2.1). An overview of the employed metrics is reported in Tab. 6.1.

Purpose	**Metric**	**Symbol or computation**
Exploration time	Number of steps	τ
Distribution	AMID	(Eq. 5.3)
	AMUD	(Eq. 5.4)
Energy Consumption	MTD	$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\tau}\|X_i(t) - X_i(t-1)\|_2$
	FEC	(Eq. 6.2)
	IEC	(Eq. 6.3)
	$\chi(\alpha)$	(Eq. 6.4)

Table 6.1: Table containing the evaluation metrics, grouped by purpose.

6.2.1 Efficiency parameter

The efficiency parameter $\chi(\alpha)$ is designed in order to provide indications of the exploration efficiency of the fleet in terms of both exploration time and energy consumption. Ideally, both terms should be minimized in the view of improving the coverage planning, nonetheless they may be in contrast to each other: a decrease

in exploration time may require some UAVs to reach far locations, that would be otherwise explored by other UAVs in a later time but without big displacements. More accent on the energy consumption, instead, would imply longer hovering times for the sake of energy saving. Notice that the efficiency parameter does not enter training process: the proposed algorithm is tested in consumption terms a posteriori using this metric. In fact, the gathered data during test simulations allows to build a tool that, depending on the desired trade off between fuel consumption and exploration time, provides the fleet size guaranteeing highest efficiency. In fact, larger fleets would undoubtedly consume more energy, but they will achieve the target coverage percentage much earlier.
On the basis of the energy model, proposed in Sec. 4.5, the Fleet Energy Consumption (FEC) is defined in Eq. 6.2, accounting for displacement consumption only, and discarding hovering, landing and ascent contributions.

$$\mathrm{FEC} = \sum_{i=1}^{N} Epm \sum_{t=1}^{\tau} \|X_i(t) - X_i(t-1)\|_2 = N \cdot Epm \cdot \mathrm{MTD} \tag{6.2}$$

From the Fleet Energy Consumption (FEC) it is possible to estimate the average Individual Energy Consumtion (IEC) as reported in Eq. 6.3.

$$\mathrm{IEC} = \frac{\mathrm{FEC}}{N} = Epm \cdot \mathrm{MTD} \tag{6.3}$$

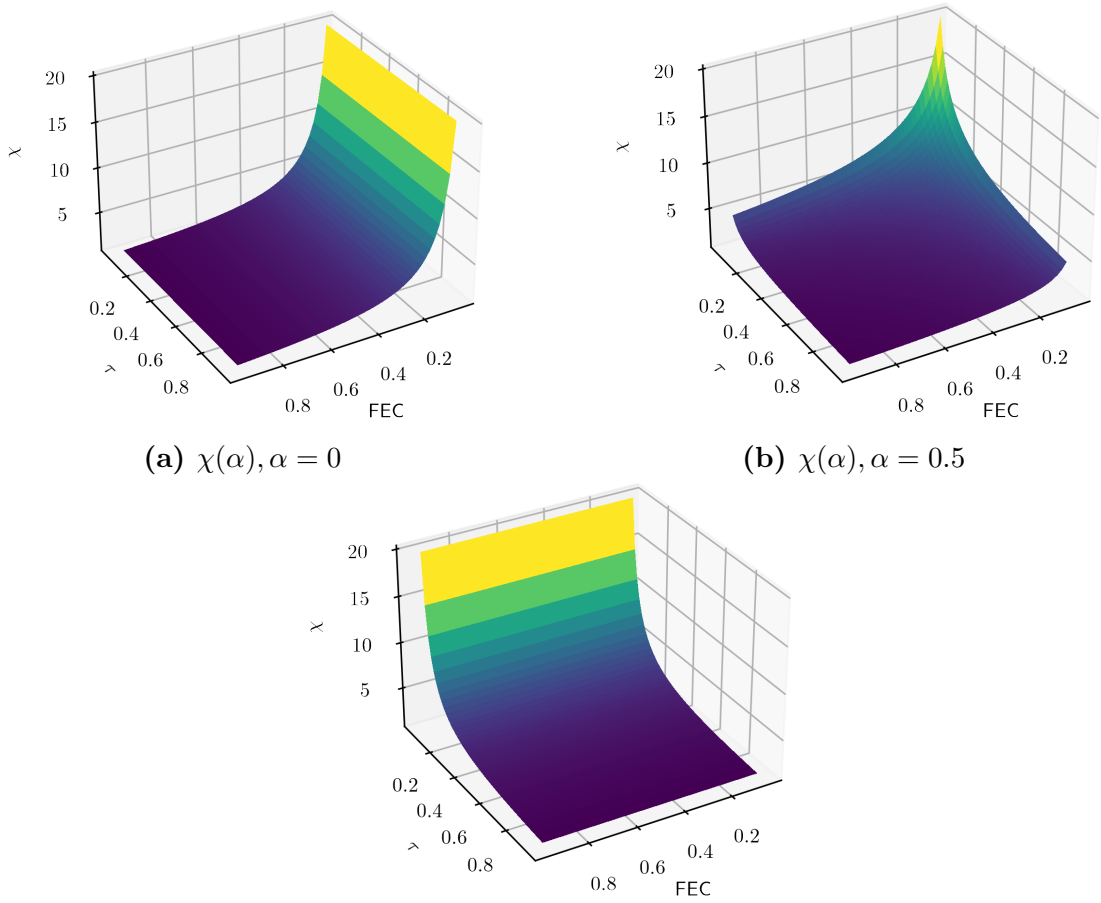
The efficiency parameter χ, for a given trade-off parameter α is computed, as function of the FEC and of the exploration time (τ), using Eq. 6.4.

$$\chi(\alpha) = \tau^{-\alpha} \cdot (\mathrm{FEC})^{\alpha-1},\ \alpha \in [0,1] \tag{6.4}$$

The trade-off parameter α weights the exploration time and fuel consumption objectives:

- $\alpha \to 0$: Interest is shifted towards energy consumption minimization;
- $\alpha \to 1$: Interest is shifted towards exploration time minimization.

An intermediate value accounts for both objectives concurrently. It is important to highlight that, for a coherent analysis, normalized values for τ and FEC shall be used in the efficiency parameter computation (more details will be provided in 6.3.3).
In Fig. 6.4 the efficiency parameter χ is plotted as function of FEC and exploration time τ (both normalized in the interval [0,1]) for $\alpha = 0, 0.5, 1$. While with $\alpha = 0$ or 1, χ depends uniquely on FEC or τ, $\chi(\alpha = 0.5)$ is simultaneously (and with equal weights) dependent on both of them.



(a) $\chi(\alpha), \alpha = 0$

(b) $\chi(\alpha), \alpha = 0.5$

(c) $\chi(\alpha), \alpha = 1$

Figure 6.4: Efficiency parameter $\chi(\alpha)$ plotted for $\alpha = 0, 0.5, 1$, as function of the number of steps τ and of the Fleet Energy Consumption (FEC).

6.3 Entire test set

In order to validate the proposed approach by testing it on a variety of environments, with different complexities, all trained models (all fleet sizes) are tested on a set composed of 300 maps, randomly generated according to the procedure and assumptions reported in (4.1.3). These maps, all approximated by means of 100×100 matrices, show an obstacle occupancy (defined as the surface occupied by obstacles over the overall size) ranging from 0% to 25%. In Fig. 6.5, the distribution of the obstacle occupancy over the test maps is reported for completeness. All results presented in this section are obtained by assuming the most stringent condition for the footprint dimension $f_{dim} = 11 \times 11$ cells; in (6.3.4) some extended outcomes about the metrics dependence on the observable area size are provided.

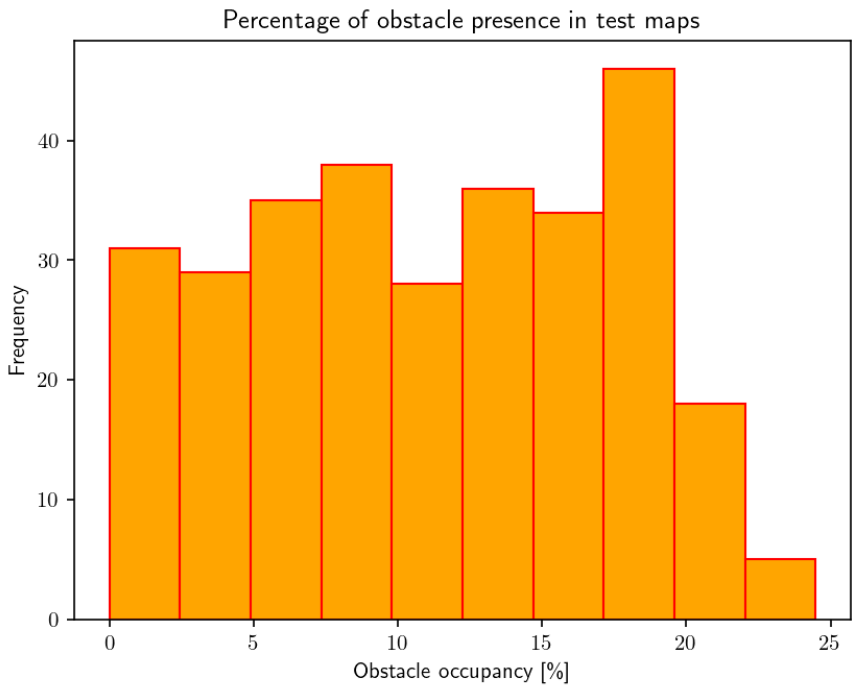


Figure 6.5: Histogram showing the distribution of obstacle occupancies in test maps, reported in terms of percentage of occupied area with respect to the environment size.

6.3.1 Exploration time

In order to analyze the exploration efficiency and the time needed to fulfill coverage objective, the number of steps, i.e. the number of motions during simulation, is analyzed as function of the fleet dimension.

In Fig. 6.6, the average episode length (in number of steps) is plotted, as function of the fleet size N. Specifically, in order to focus on the average trend, neglecting the influence of environmental complexity, the figure shows, for each N, the average episode length, computed on the basis of all simulations carried out in test maps. Fleets consisting of 2 units only, as expected, require much longer times to complete exploration than larger ones. In fact, 3 UAVs fleet manage to achieve full coverage in almost half the time needed by couples of UAVs. Up to $N = 6$, τ decreases monotonically: this intermediate dimension results in being optimal in terms of exploration time. Fleet sizes with $N > 6$, do not show a further reduction in τ, indeed there is even a slight increase of exploration time for $N = 8$. This behaviour is the consequence of an excess of fleet units collaborating in exploration. In fact,

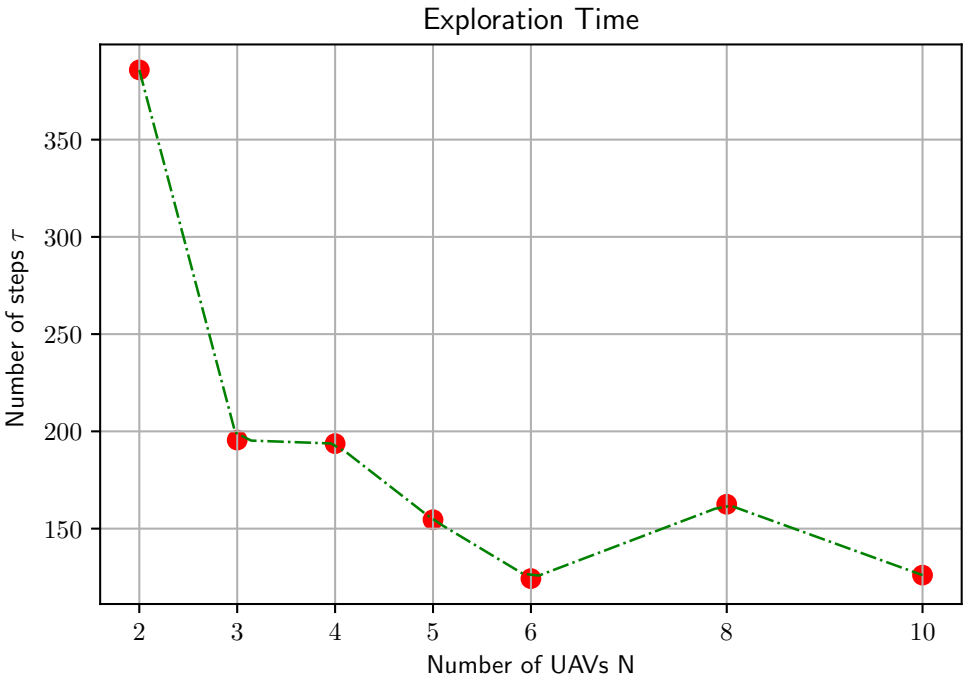


Figure 6.6: Plot showing the average exploration time (in number of steps τ) as function of the fleet size N.

notwithstanding the presence of many UAVs may aid in work division and sharing, an excess of multiple explorers requires substantial collaborative planning in terms of distribution that penalizes the achievement of the main objective, in concordance with the statistics reported in (6.3.2).

6.3.2 Distribution statistics

Regarding distribution-related metrics, the Average Mutual Distance (AMUD) and Average Minimum Distance (AMID) are computed for each fleet size, during the exploration of the map in the test set. The statistics involving the AMID parameter are in agreement with the distribution-related contribution in the reward function of the coverage agent. In fact, referring to Eq. 4.11, the application of a negative penalty to couples of UAVs nearer than the normalized distance 0.1, reflects in AMID being greater than this threshold, for all fleet sizes. This is shown in details in Fig. 6.7, in which both the AMID distribution (on the left) and the average values, along with $\pm 1 \cdot \sigma$ confidence intervals (on the left) are displayed.
The average value of the AMID monotonically decreases with the fleet size, with

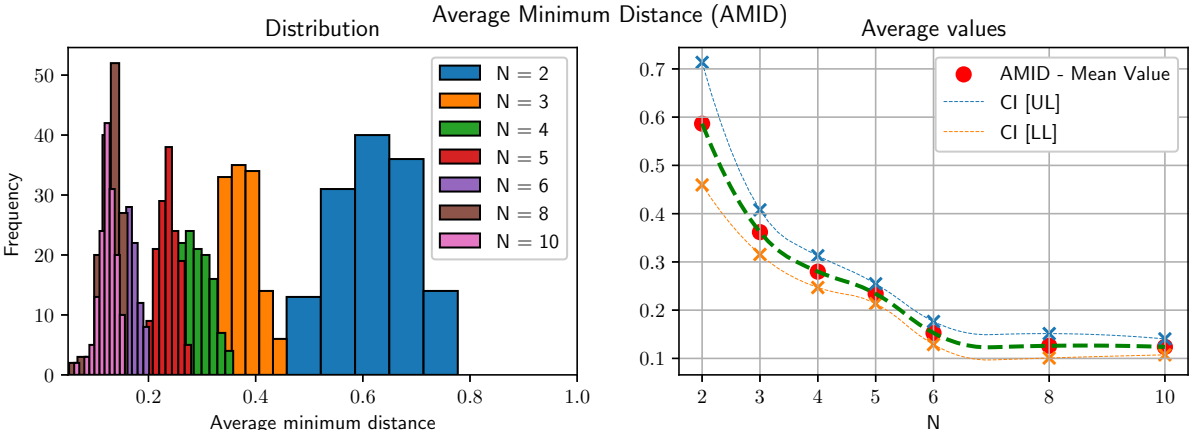


Figure 6.7: Representation of the results involving AMID parameter. The empirical distribution is reported on the left, while the average values with $\pm 1 \cdot \sigma$ confidence intervals are plotted on the right.

$N = 6$ representing the elbow point, after which the AMID average value remains roughly steady. As it is more evident in the bar plot, the variance of this parameter is greater for smaller fleets: this is reasonable because a larger N implies greater probability of mutual approaches, thus restricting the AMID value in a smaller region nearby the threshold inherited by the reward function. In any case, the presented results validate the capability of the coverage agent in mutual distancing, which already reduces the need of OA agent in avoiding mutual collisions. In Fig. 6.8 a similar analysis, concerning the AMUD parameter, is graphically shown.

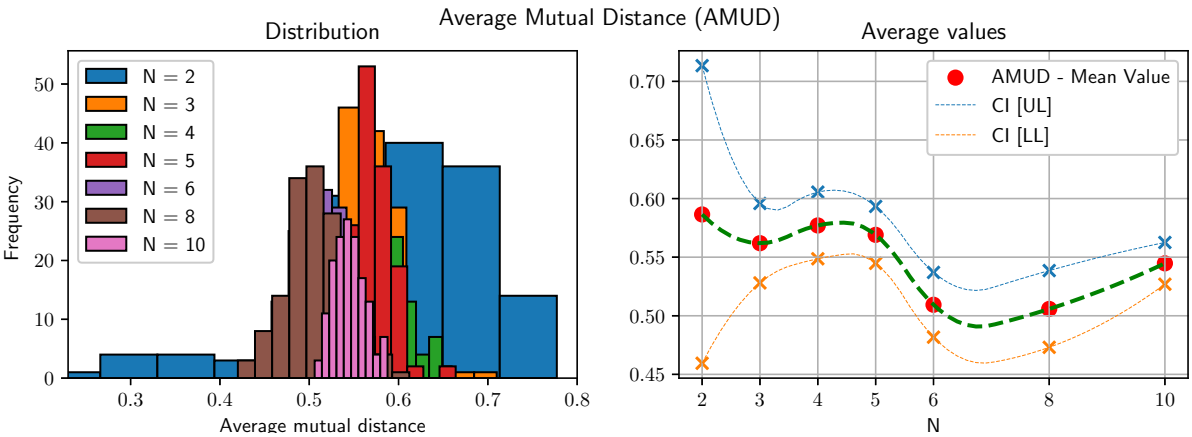


Figure 6.8: Representation of the results involving AMUD parameter. The empirical distribution is reported on the left, while the average values with $\pm 1 \cdot \sigma$ confidence intervals are plotted on the right.

In this case, the average values do not change consistently as function of the fleet size, varying in the range between 0.5 and 0.6. For $N > 3$, the fleet size does not influence the variability of AMUD values on the different maps, while a greater variance affects 2 UAVs fleets. This is partly due to the fact that unique distance evaluated in such low-size fleets (AMUD and AMID coincide) greatly depends on the map complexity and peculiarities.

6.3.3 Energy consumption

By means of the MTD recorded during test simulations, the Fleet Energy Consumption (FEC) and Individual Energy Consumption (IEC) are computed in order to evaluate the algorithm efficiency in energy terms. Specifically, the FEC provides useful information in terms of overall fleet power usage, whereas the IEC allows to assess the capability of the algorithm in achieving full coverage without running out of autonomy on the single units. MTD metric, and consequently the FEC and IEC, is computed by considering grounds with dimensions 1 km $\times$ 1 km.

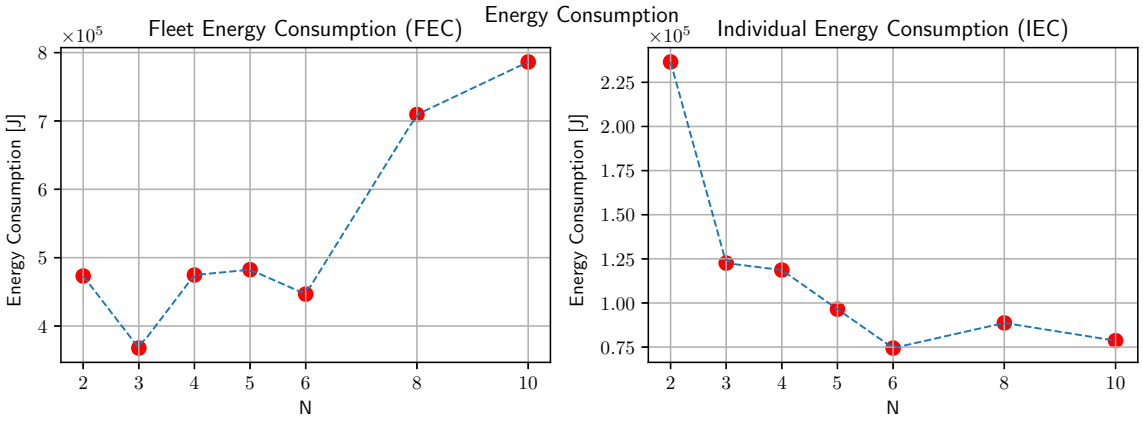


Figure 6.9: Representation of the results FEC and IEC. Both are computed on the basis of the MTD and are expressed in J.

In Fig. 6.9 both metrics are plotted as function of the fleet size, reporting the registered consumption, averaged over all test simulations. The FEC parameter does not change substantially until $N = 6$, meaning that despite the increase in fleet size, the required energy per unit reduces because of limited exploration time and traveled distance involving each UAV, while larger fleets ($N = 8, 10$) require almost double the power consumption required by smaller ones. Concerning the IEC, its trend suggests a general reduction of the per unit consumption with the fleet size, still strengthening the selection of fleet sizes with $N = 6$ as the most efficient condition, in compliance with the proposed algorithm. In Tab. 6.2, the average IEC values are reported, for all fleet sizes. Recalling that the selected

reference UAV for energy consumption analysis (see Sec. 4.5) is empowered by a 213 kJ battery, the algorithm is capable to fulfill coverage objective for all fleet sizes with $N \geq 3$ without power issues. Anyway, recalling the big environment size (1km $\times$ 1 km), it is reasonable that 2 UAVs alone could not manage to fully cover such environment within their power constraints.

Fleet size N	**IEC [kJ]**
2	236.5
3	122.6
4	118.6
5	96.47
6	74.44
8	88.68
10	78.64

Table 6.2: Average Individual Energy Consumption (IEC) results in test maps.

Using the processed results, it was possible to compute the efficiency parameter $\chi(\alpha)$ for all fleet sizes and for different values of the trade-off parameter α, by averaging out the FEC and exploration time τ over the test simulations carried out in all test maps of the set. In Fig. 6.10, efficiency parameter $\chi(\alpha)$ is plotted, as function of the fleet size, for different values of $\alpha \in [0,1]$.

As anticipated in (6.2.1), the FEC and τ parameters are normalized in the computation of the efficiency parameter so that, changing α value, the interest towards energy efficiency or time minimization shifts accordingly without scaling issues.

In the case of low α, which corresponds to a greater interest in minimizing the overall energy consumption, lower size fleets are preferred, specifically 3 UAVs fleets show the highest efficiency parameter. On the other hand, higher α values, consider low size fleets much more inefficient, preferring larger ones. It is evident how the best compromise refers to medium size fleets ($N = 6$), for which the efficiency parameter achieves above-average values as well as a limited variation as function of α, that is the consequence of high capability of satisfying optimal performances in terms of both energy consumption and exploration time.

It is worth to remark that selected motion directions do not properly constitute a path planning method, rather they enable near waypoint assignments, to be reached by calling specific end-to-end path planning methods. That's why, even though this analysis estimates the energy consumption in a sufficiently reliable way, the effect of local motion planners may provide some variations on the numerical values.

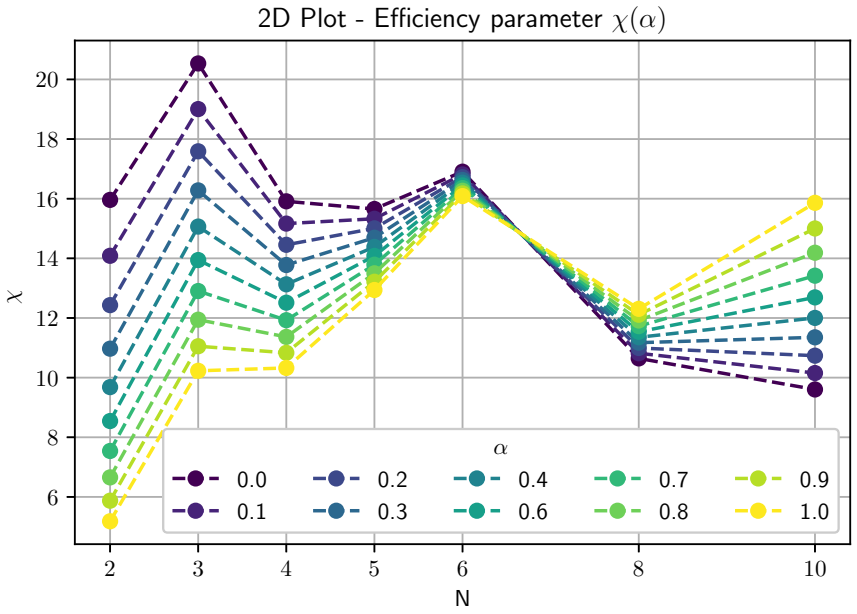
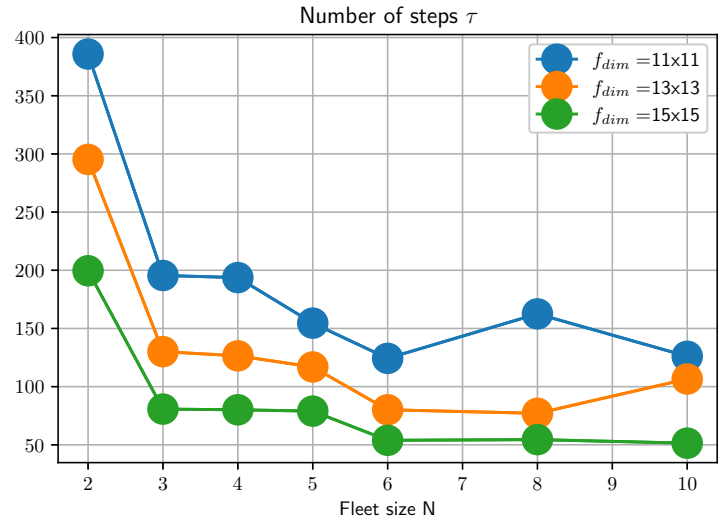


Figure 6.10: Efficiency parameter $\chi(\alpha)$ for different α, as function of the fleet size.

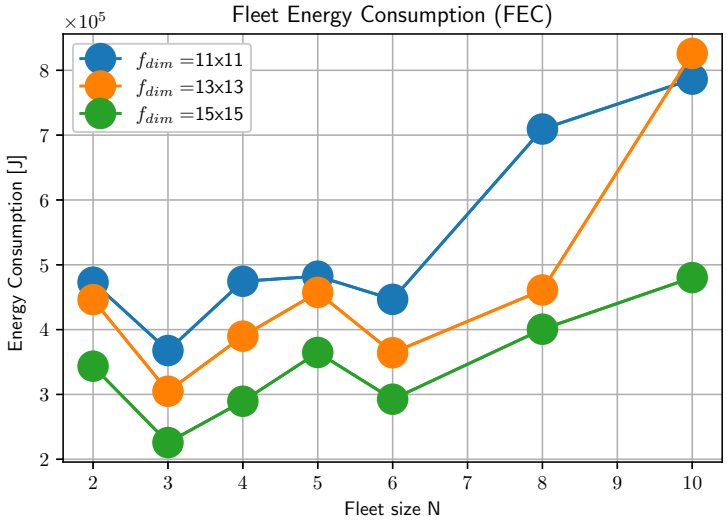
6.3.4 Dependence on observable area size

To confirm expectations about performance metrics variation with respect to the observable area size as well as applicability of the algorithm with variable operating conditions, all fleet sizes were tested on all maps of the test set (see introductory part of Sec. 6.3) with different footprint dimensions, from 11×11 to 15×15 cells. The intuition that fleet performances shall improve, assuming wider fields of view (FOVs), is confirmed by the simulation results.

In Fig. 6.11a the average exploration time (in number of steps τ) recorded on the test maps is plotted as function of the fleet size, for different assumptions on the footprint dimension. Increasing observable areas correspond to lower exploration times, reaching for the largest dimension (15×15 cells) just 50 steps needed to accomplish the goal, even with medium size fleets ($N = 6$). This is also correlated to what is shown in Fig. 6.11b in which the FEC is plotted for the various FOV assumptions. Enhanced sensing specifics reflect in lower energy consumption, for all fleet sizes.



(a) Exploration time for different fleet sizes, compared as function of the FOV dimension.



(b) FEC for different fleet sizes, compared as function of the FOV dimension.

Figure 6.11: Results on test maps with varying FOV assumptions.

6.4 Case studies

To judge the fleet performances on environments which can be representative of complex urban areas, reference maps with increasing complexity were selected

as case studies, characterized by 0 to 20% of their surface occupied by obstacles, shown in Fig. 6.12.

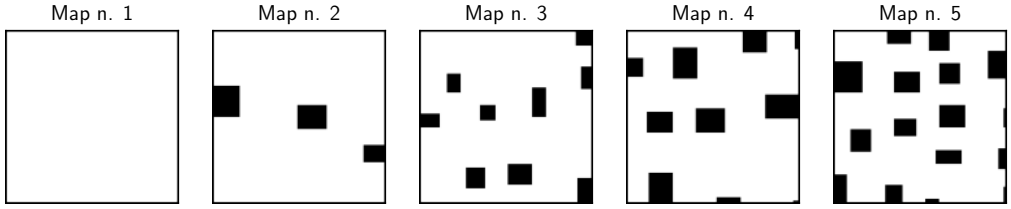


Figure 6.12: Reference maps selected as case studies, with obstacles occupying from 0 to 20 % of their surface.

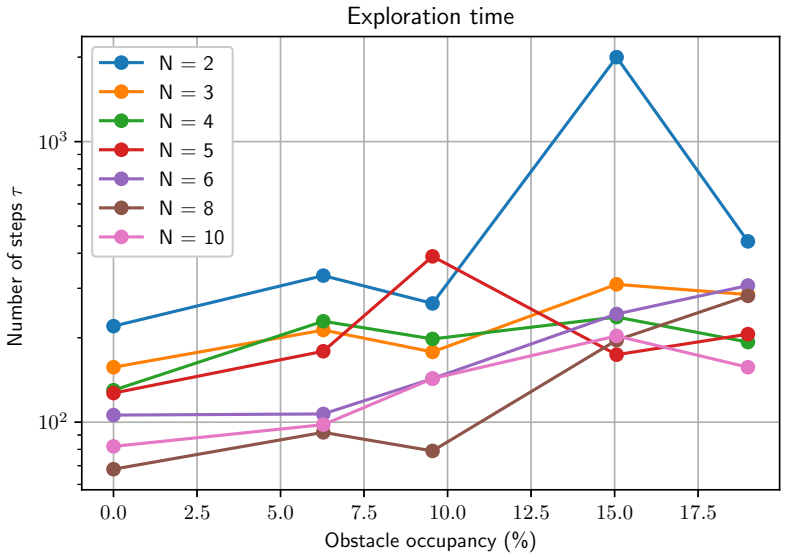
For these maps, the performance assessment is conducted in the same way as for the entire test set, as specified in Sec. 6.1, and the evaluation metrics are selected among the ones already reported in Tab. 6.1. The precise amount of obstacles is indicated in Tab. 6.3.

Map number	**Obstacle presence** [%]
1	0
2	6
3	10
4	15
5	19

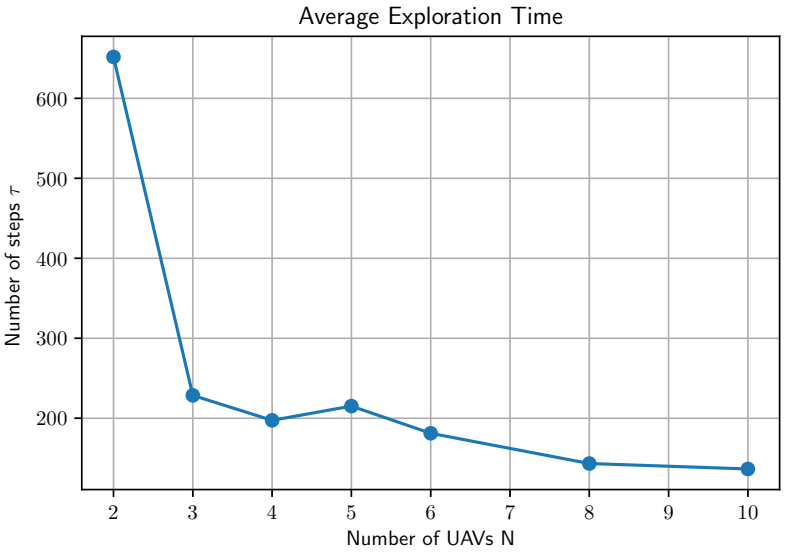
Table 6.3: Percentage of surface occupied by obstacles in maps selected as case studies.

6.4.1 Exploration time

The exploration time, indicated in terms of number of steps before reaching full coverage, is displayed in Fig. 6.13a for all fleet sizes, as function of the obstacle occupancy, while in Fig. 6.13b the exploration time, averaged over the case studies, is reported as function of the number of fleet units N.
According to the general trend, it is evident how to smaller fleets correspond longer explorations, with only a limited dependence on the map complexity, whose increase implies rises in exploration time, for all fleet sizes. Anyway, increasing complexity does not cause substantial performance degradation, since the corresponding rise in exploration time is limited and shows only a slight increase.



(a) Semi-log plot of the exploration time in number of steps τ, as function of the amount of surface occupied by obstacles, in the reference maps.



(b) Plot of the exploration time in number of steps τ, as function of the fleet size N, averaged over the reference maps.

Figure 6.13: Reference maps: exploration time

6.4.2 Distribution statistics

The distribution efficiency is evaluated in terms of AMID among UAVs, during the simulated exploration tests.

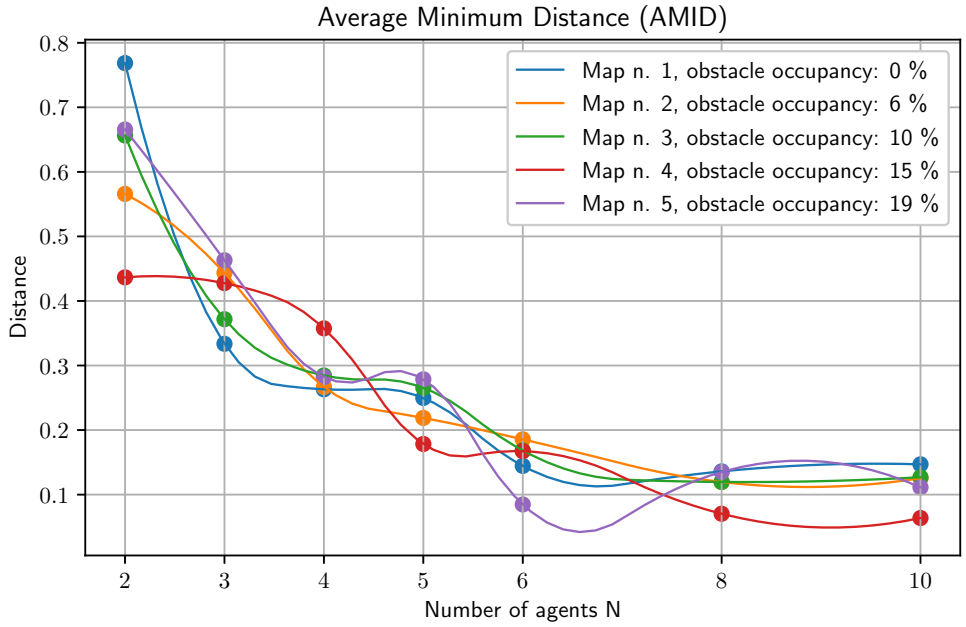


Figure 6.14: Plot of the Average Minimum Distance AMID as function of the fleet size, for all reference maps.

In Fig. 6.14 the AMID is reported, as function of the number of fleet units N, for all maps reported in Fig. 6.12. As well as the common expected reduction in AMID with increasing fleet sizes, caused by a larger number of UAVs exploring the same environment, it is notable how this parameter keeps almost always over the threshold (0.1) appearing in the reward function (see Eq. 4.9), except for the largest fleets exploring the most complex environments, for which the presence of this value slightly below the threshold is still acceptable. In general, specifically for simpler maps, AMID decreases monotonically with N.

6.4.3 Energy consumption

Apart from the FEC and IEC parameters, which are in line with the trends reported for the entire test set in (6.3.3), the results gathered by simulating the proposed model in the reference maps permits to draw a series of conclusions about energetic efficiency, exploiting the efficiency parameter $\chi(\alpha)$, whose formalization is provided

in Eq. 6.4. In Fig. 6.15 the efficiency parameter $\chi(\alpha)$ is reported, for all fleet sizes and for all selected maps.

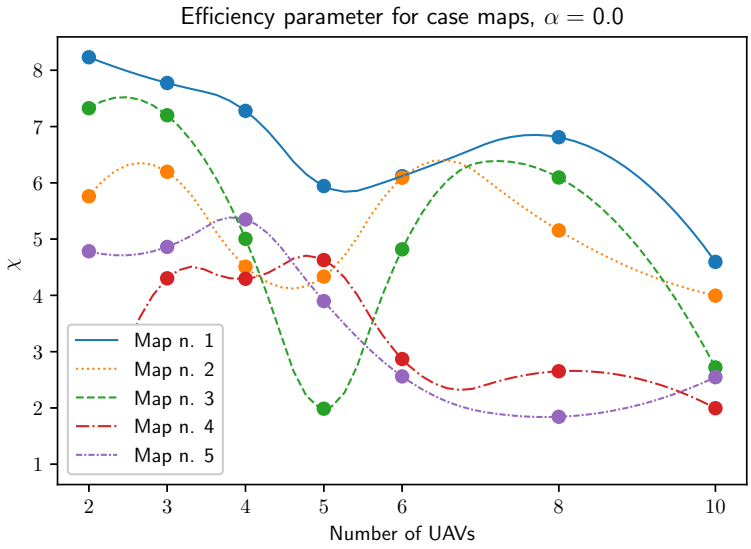


Figure 6.15: Plot of efficiency parameter $\chi(\alpha)$, for $\alpha = 0$, for the different maps, with variable fleet size N.

Recalling that $\alpha \to 0$ makes the efficiency parameter biased toward fleet consumption minimization, the obtained results suggest that low-medium size fleets manage to be more efficient in these terms than bigger size ones, specifically in the case of complex environments. In Fig. 6.16, the efficiency parameter with $\alpha = 0.5$ is plotted, in the same conditions as for Fig. 6.15.
In this case, after feature normalization, efficiency weights similarly exploration time and FEC minimization, showing different scenarios with respect to the environment complexity. Larger fleets are more efficient in these terms in simpler maps, while for environments with higher obstacle density, low-medium size fleets are preferred. The situation changes with $\alpha = 1$, corresponding to time minimization interest only. In this specific case, reported in Fig. 6.17, larger fleets are indicated to be more efficient in reducing the exploration time than larger ones, independently from the environment complexity.

6.5 Exploration example

In order to provide an example of the algorithm application, in this section the exploration process of map n. 3 (see Fig. 6.12), carried out by a medium-size fleet ($N = 4$) will be described and analyzed in detail.

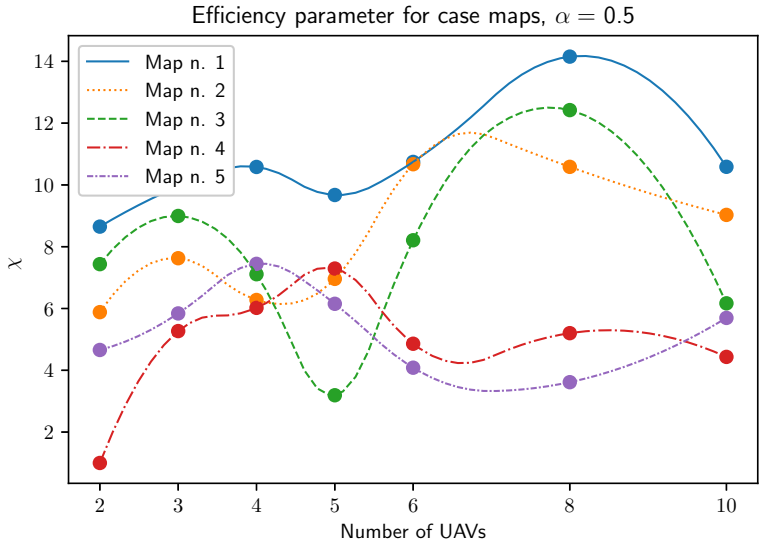


Figure 6.16: Plot of efficiency parameter $\chi(\alpha)$, for $\alpha = 0.5$, for the different maps, with variable fleet size N.

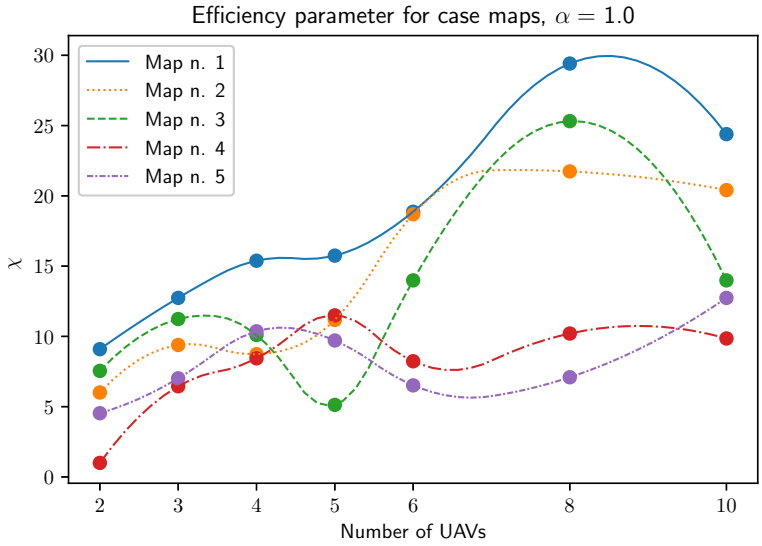


Figure 6.17: Plot of efficiency parameter $\chi(\alpha)$, for $\alpha = 1$, for the different maps, with variable fleet size N.

In Fig. 6.18 the UAVs positions, along with the reconstructed environment, are shown, at different phases of the exploration. Light blue areas correspond to

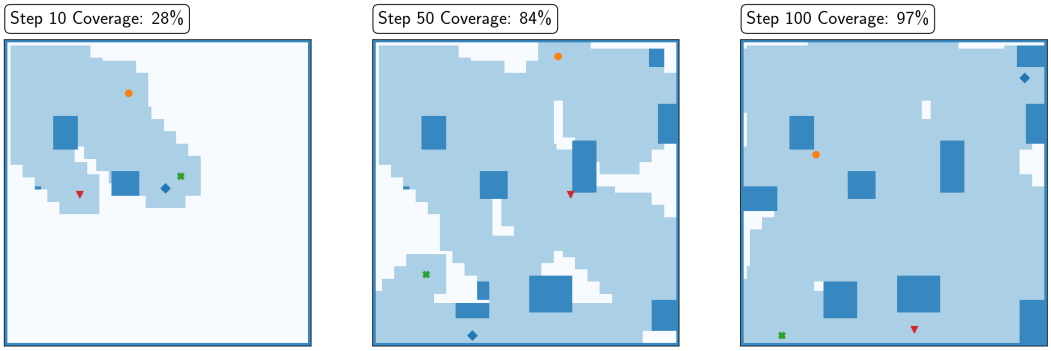


Figure 6.18: Graphical representation of the exploration process over time. Environment map is progressively built by the exploring UAVs.

obstacle-free explored regions, whereas dark blue zones indicate the presence of detected/reconstructed obstacles. This simulation is conducted by assuming an intermediate observable area dimension (13×13 cells). In order to analyze the UAVs contribution in reaching the coverage objective, in Fig. 6.19 the cumulative sum of individual coverage contributions $\sum_{t=0}^{t^*} \Delta_i(t), i = 1, \ldots, 4$ is plotted, against the episode step t^*, until full coverage.

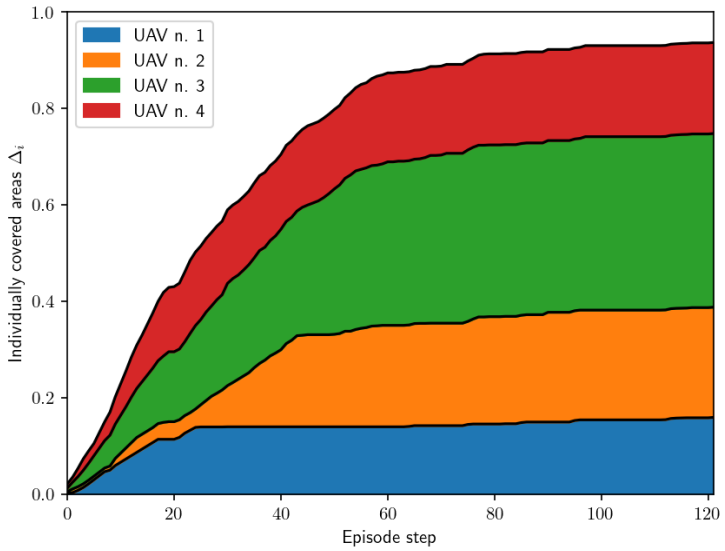


Figure 6.19: Plot of the cumulative individual contributions in coverage as function of the episode step.

It is notable how the individual contributions of the fleet units in coverage at the episode end are approximately comparable, in spite of slight differences, which can be traced mainly to the starting conditions, already reported for a larger fleet in Fig. 6.2. As expected, the initial exploration phase is characterized by larger coverage increases, followed by a reduction in rate increase which makes the exploration proceed approximately steadily until completion. The initial limited contribution of UAV n. 2 in coverage is mainly related to its unfavorable position in the fleet: during the initial phase of exploration, before starting to operate in unknown regions, it moves in the same direction as the other drones, which precede it before the fleet spreads efficiently in the environment.

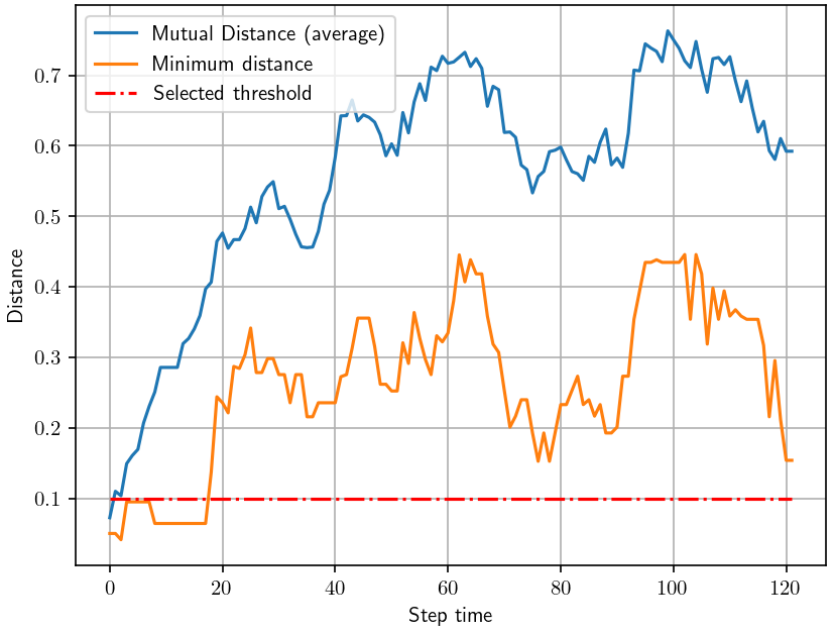


Figure 6.20: Plot of the average mutual distance and of the minimum distance (both recorded at each step) during map exploration.

UAVs distribution as function of time can be observed in Fig. 6.20, in which, at each time step, the mean mutual distance and the minimum reciprocal distance are plotted, along with a reference line indicating the threshold, related to the reward function definition in terms of distribution (see Sec. 4.3.3). It is evident how during the first phase of simulation, when the fleet starts to explore and the units distribute in the environment, the average mutual distance rises sharply and then it oscillates around 0.6, thus indicating how UAVs explore different areas at the same time. With reference to the minimum distance, after the initial phase in which it remains below the threshold, due to the starting conditions, it exceeds the limit value and remains above it throughout the simulation, as it is desirable.

Chapter 7

ROS Simulation

In this chapter, an implementation of the proposed coverage planning model in a 3D environment is presented. Specifically, the simulation is carried out in Gazebo [45], a 3D open source simulator and ROS [46], that is a meta-operating system providing tools and libraries for the development of robotic applications. For the control of UAVs, an open source software, PX4 Autopilot [47], communicates with ROS using the MAVROS package.

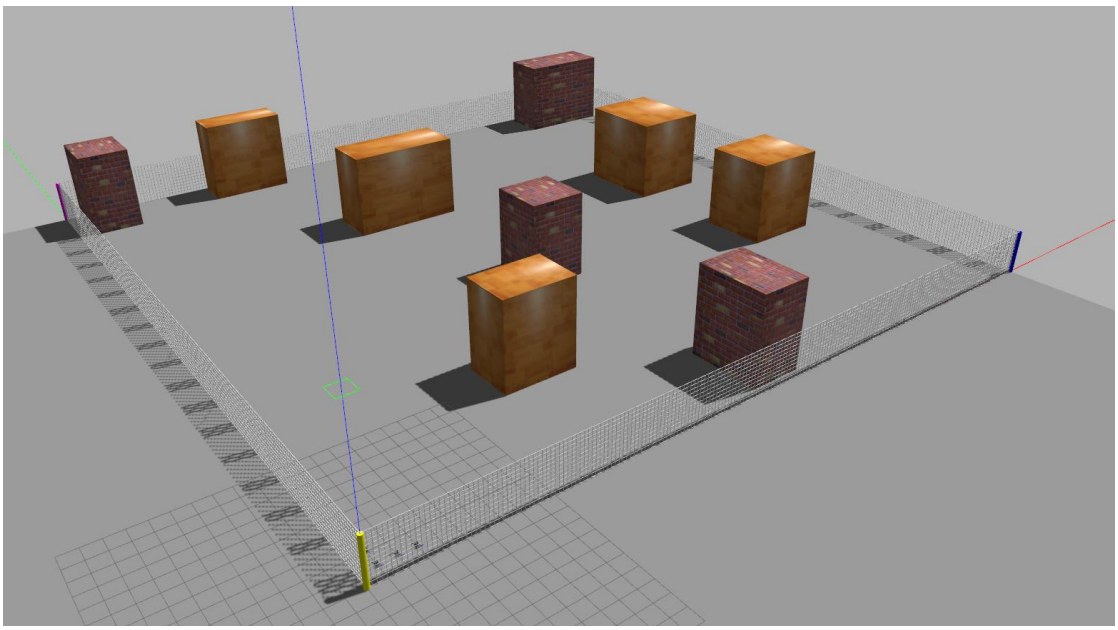


Figure 7.1: Graphical representation in Gazebo of the simulation environment.

The simulation environment is a 3D extension of the third map belonging to the selected case studies (see Fig. 6.12), which was appropriately implemented in

Gazebo environment, as shown in Fig. 7.1. The flight altitude is commonly set to 4 m, while obstacles are all 6 m high. As already mentioned in (4.1.3), the binary maps used for training and test phase of the RL agents can hypothetically approximate any squared real field; in this case the simulation ground occupies an area of size $50 \times 50m$. To manage take-off, landing and path planning between consequent waypoints, as many ROS nodes as fleet units are initialized, and interact with Flight Control Units (FCUs) implemented in PX4 Autopilot software. The starting condition of the fleet is represented in Fig. 7.2, showing 4 UAVs in idle state, waiting for the take-off phase.

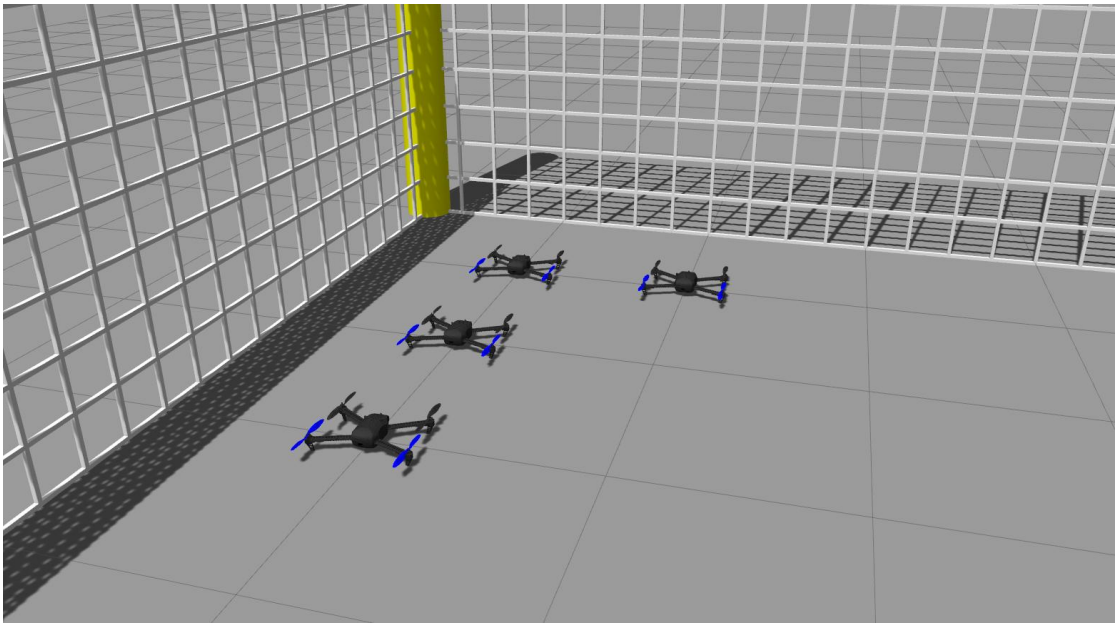


Figure 7.2: Starting condition of the fleet. UAVs are in idle state, before take-off.

In Fig. 7.3 the final part of the take-off phase is shown; UAVs are represented while moving to the first waypoints computed using the trained agents. In Fig. 7.4 the fleet is shown during the simulated exploration phase.
In order to validate the proposed model and check possible discrepancies with respect to the test conditions, which do not take into account the effect of the flight controllers, some ROS nodes were added to gather distribution data during exploration and compute the average mutual distance and the minimum distance among UAVs at each time step. In Fig. 7.5 such statistics are plotted, along with all the recorded distances among all possible couples of UAVs. Considering that the simulated conditions are the same as the ones reported in Sec. 6.5 and specifically in Fig. 6.20, it is possible to notice how they show the same general trend, with smoother changes, mainly linked to high frequency data acquisition and to the

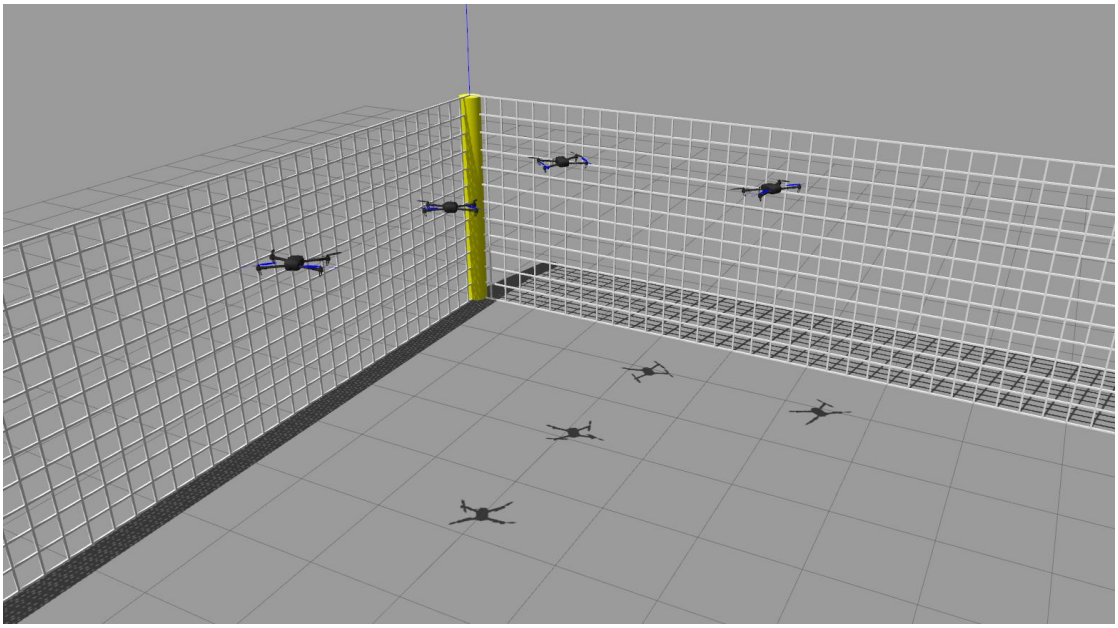


Figure 7.3: UAVs at the end of their take-off phase.

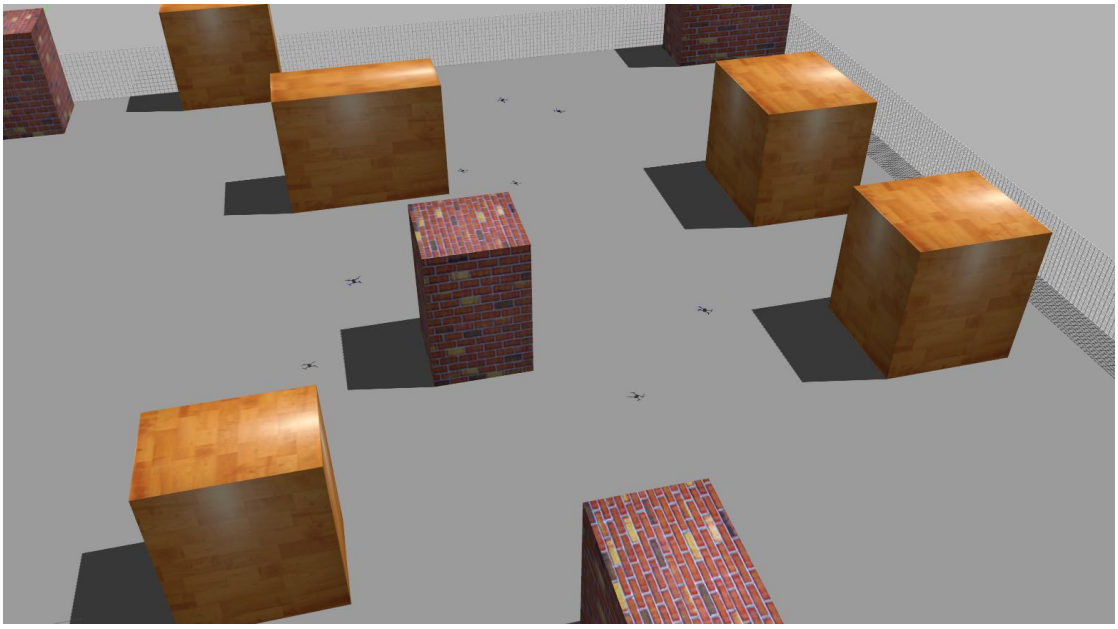


Figure 7.4: Fleet exploration during simulation.

presence of path planning methods between waypoints instead of abrupt position changes.

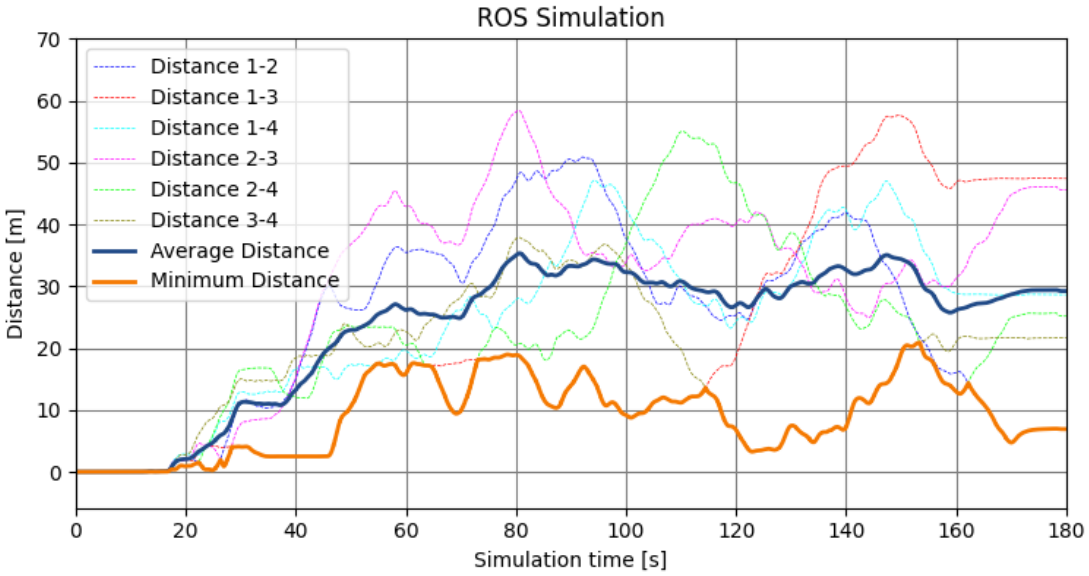


Figure 7.5: Plot showing the average and mutual distance among UAVs during the exploration phase, computed on the basis of the UAVs positions resulting by simulation.

Chapter 8

Conclusions and further studies

The proposed coverage planning model has shown interesting and encouraging results in terms of both target accomplishment and distribution statistics. The novel approach, which exploits the multi-agent settings to train a shared policy function in order to prevent scalability issues as well as to reduce the training time, allows to train fleets with widely varying sizes using an easily scalable approach. This modified version of the PPO algorithm, in conjuction with the power of convolutional neural networks, manages to plan coverage efficiently, without facing mutual collisions. Furthermore, the joint implementation of the obstacle avoidance agent, allows to prevent collisions with fixed obstacles while following the coverage-driven indications. It is worth remarking that the motion selection, although it implies small displacements, is anyway limited to a restricted set of directions which introduce in the deployed trajectories sharp attitude and motion direction changes. It is worth remarking that the proposed work does not aim at developing a path planning model, rather the outputted motion directions can be interpreted as waypoint generators, close to the current positions, to be reached to boost the exploration by means of already developed and existing models. That's why movement between waypoints shall be managed by means of special-purpose controllers and path planners, as done in Chap. 7 exploiting PX4 Autopilot software. The usage of pre-trained neural networks in real test implementations, enables fast decision-making without requiring online optimization. From this work, it appears significantly evident the advantage of using RL-based techiniques to solve complex and multi-objective problems by shaping a reward function, without needing to explicitly program each agent's motion. On the other hand, trial and error tuning is still necessary due to the multitude of methods and approximations involved, before obtaining stable and efficient learning convergence.

Although the completeness of the proposed approach and the satisfactory results that have been achieved, it is anyway possible to outline a series of improvements and extensions which can be applied to this work:

- Despite the proved reliability and effectiveness of the collaboration between coverage and obstacle avoidance agents, splitting various objectives among hierarchical policies does not guarantee optimality in general terms. In fact, while a single policy may be ideally trained up to optimality, multiple policies are not guaranteed to do so. Developing an agent which is capable to plan coverage while taking into account obstacle detections would be preferrable, assuming that all objectives are correctly accounted for in the reward function and the input states provide a complete representation of the agent-environment condition.

- The presented approach assumes an ideal condition of constant and instantaneous information exchange among UAVs. In the reality, especially in large simulation fields, such assumption may be unrealistic. The proposed approach may be extended by exploiting specific communication models or by performing expectations about other UAVs movements. In fact, given that the same policy function is used in each fleet unit, by knowing previous positions and coverage statistics, it is possible to predict other UAVs motions by calling the on-board policy with modified inputs. This procedure may show limitations in the case of trajectory modification because of unseen and not communicated obstacles, but assuming temporary delays or communication interruptions, they are expected not to diverge from the real behavior.

- As already mentioned, since the coverage planning output directions shall be interpreted more as target waypoint generators rather than motion directions to be directly applied, the application of specific path planning algorithms, either more traditional ones, as well as RL-based approaches, would help in end-to-end local planners, providing smoother trajectories without abrupt attitude changes. Another possible approach would be to modify the action space to a continuous one, enabling it to directly output smoother paths as a path planning method would do as well as taking into account the UAVs' heading directions in input states.

- UAVs are fixed to fly at the same altitude and they are characterized by equal characteristics in terms of observability and sensors. Allowing them to move vertically would surely enhance their exploration capabilities, both in terms of changing observable footprint dimension as well as in a reduced possibility of mutual approaches. A 3D extension with variable flight altitudes would also require to change the input states and detection assumptions.

Appendix A

Artificial Neural Networks

A.1 Introduction

Artificial Neural Networks (ANNs) are bio-inspired tools, taking inspiration from functionality of human and animal brains. In an ANN, each elementary unit, called *neuron*, linearly combines several input signals coming from other neurons, and returns a single output value, after a nonlinear activation function. ANNs can be exploited both for classification and regression problems and, due to nonlinearities in each unit, they represent complex non-linear form of hypotheses parametrized with respect to connection weights W and biases b, computed as a result of a training procedure. A typical Feed-forward Neural Network (FFNN) structure is shown in Fig. A.1.

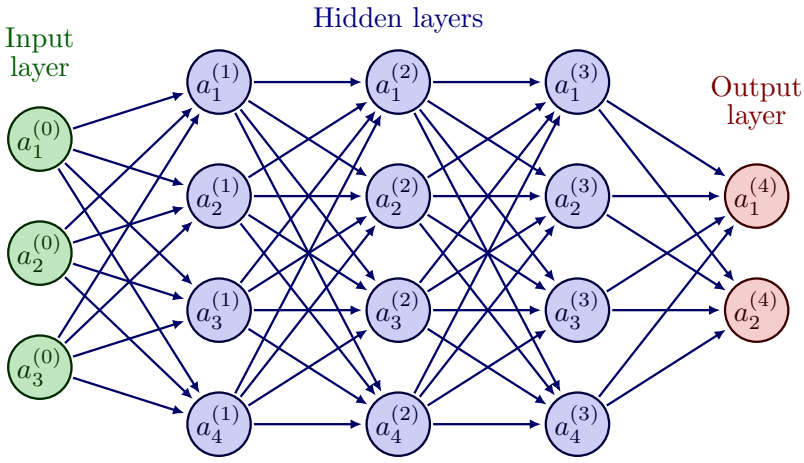


Figure A.1: Feed-forward Neural Network with 3 inputs, 2 outputs, and 3 hidden layers. A FFNN does not have any loops or cycles.

A.2 Elementary Unit: Neuron

The elementary unit of a neural network, called *neuron*, performs 2 operations:

- Linear combination of its inputs (to which, usually, a bias term is added);
- Nonlinear activation function.

Denoting with $f(\cdot)$ the activation function, and being $b \in \mathbb{R}; x, w \in \mathbb{R}^n$, the output y of a single neuron with n inputs is computed according to Eq. A.1. In Fig. A.1 a neuron with $n = 3$ inputs and a bias term is schematized.

$$y = f(w^T x + b) \tag{A.1}$$

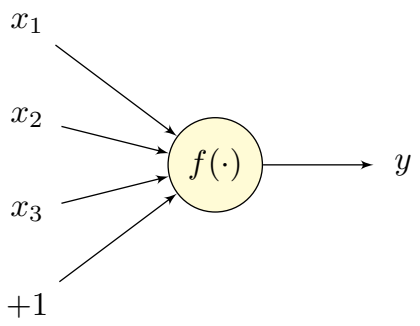


Figure A.2: Neuron with 3 inputs and a bias term.

A.2.1 Activation functions

Commonly used activation functions in ML settings are reported in Tab. A.1, along with their representation in Fig. A.3.

Sigmoid σ	$f(z) = \frac{e^z}{1+e^z}$
Hyperbolic Tangent (tanh)	$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
Rectified Linear Unit (ReLU)	$f(z) = \max(0, z)$

Table A.1: Typical Activation Functions $f(\cdot)$.

A.3 Structure and modeling

An ANN is formed by connecting together several neurons, and it is composed by three main layer types:

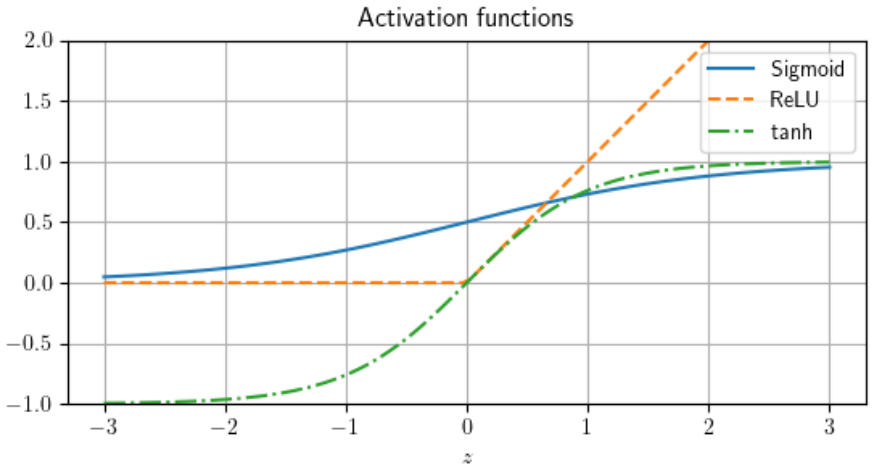


Figure A.3: Plot of sigmoid, tanh and ReLU activation functions.

- *Input layer*: It receives the input values;
- *Hidden layer(s)*: They are responsible for data processing, by means of tunable parameters W, b, trained in the view of a loss function minimization;
- *Output layer*: It includes the output values of the NN, computed through the preceding layers.

With reference to figure A.1, and considering a generic layer l, denote with s_l the number of its units. Each layer is associated to parameters $(W^{(l)}, b^{(l)})$, where $W_{ij}^{(l)}$ denotes the weight factor relating unit j in layer l with unit i in layer $l+1$. For a generic layer l, $W^{(l)}$ is thus a matrix $W \in \mathbb{R}^{(s_{l+1}, s_l)}$. $a_i^{(l)}$ denotes the activation of i-th node in the l-th layer, and it is computed as in equation A.2.

$$a_i^{(l)} = f\left(\sum_{j=1}^{s_{l-1}} W_{ij}^{(l-1)} a_j + b_i^{(l-1)}\right) = f\left(W^{(l-1)} a^{(l-1)} + b^{(l-1)}\right) \tag{A.2}$$

These assumptions hold when there are no cycles and loops, as it happens in any FFNN. The presence of several hidden layers makes a NN a Deep Neural Network (DNN).

A.4 Training and Backpropagation Algorithm

Once network topology and parameters (W, b) are defined, NN output is computed through a *forward pass* of the input values throughout the network layers and weights. On the basis of the network output and of a suitably chosen loss function,

the network parameters can be updated in order to minimize the loss value J. The optimization algorithm to be solved is depicted in equation A.3.

$$\min_{W,b} J(W,b) \tag{A.3}$$

In general, J is not a convex function of W and b, thus implying that most optimization problems do not solve up to global optimality. Training procedure is usually performed by means of *stochastic gradient descent* (SGD) method, which reduces computational time of classical gradient descent algorithm through a gradient approximation based on a limited subset of training samples. In this technique, at each iteration step, weights are updated by moving towards descending direction of the (approximated) gradient of the loss function, as reported in Eq. A.4, indicating with $\tilde{\nabla}_W$ and $\tilde{\nabla}_b$ the approximated gradient with respect to W and b respectively. η is the learning rate, weighting the parameters update at each optimization step.

$$\begin{aligned} W &\leftarrow W - \eta \tilde{\nabla}_W J(W,b) \\ b &\leftarrow b - \eta \tilde{\nabla}_b J(W,b) \end{aligned} \tag{A.4}$$

Neural Network weights are typically initialized to small random values near zero for symmetry breaking and gradient computation is performed exploiting the *backpropagation algorithm*, which is based on the chain rule for derivatives computation. Backpropagation algorithm consists of 3 main phases, repeated until convergence:

- *Forward pass*: inputs are propagated across the current network, and outputs are computed;
- *Loss computation*: On the basis of the computed outputs and of a suitably chosen loss function, loss value is calculated;
- *Backward pass*: From the computed loss, gradients are calculated by moving backwards in the layers and exploiting the chain rule.

A.5 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a class of ANNs, usually employed for image processing and analysis. Their advantage is an increased efficiency in hierarchical pattern recognition and analysis. With respect to fully connected layers, in which the huge amount of weights and biases could lead to overfitting, convolutional layers make use of *local filters*, which are applied to the input features

by sliding along them, and do not require each neuron in a layer to be connected to all neurons in the preceding one. Filters in CNNs average over the input layers by dot products between local input with filter weights, subject to training. In each convolutional layer, the number of filters equals the one of *feature maps* created, and unlike fully connected layers, filter weights do not change by sliding, thus providing a reduced computational burden during training. Besides the number of feature maps, *strides* constitute another important hyperparameter, denoting the amount of pixels the filters shift during their application. Among convolutional layers, typically activation functions and pooling layers are interposed.

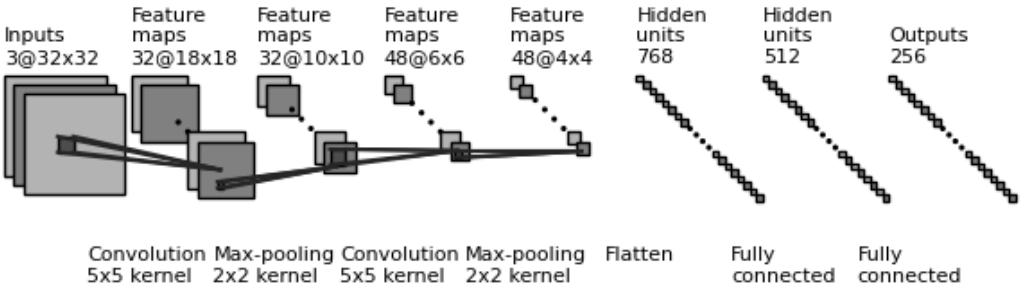


Figure A.4: Typical Convolutional Neural Network structure.

In Fig. A.4 a CNN with 3-layer input features is shown. The first convolutional layer employs 32 filters with kernel size 5×5, while in the second layer 48 filters of dimension 2×2 are used, before a series of fully connected layers.

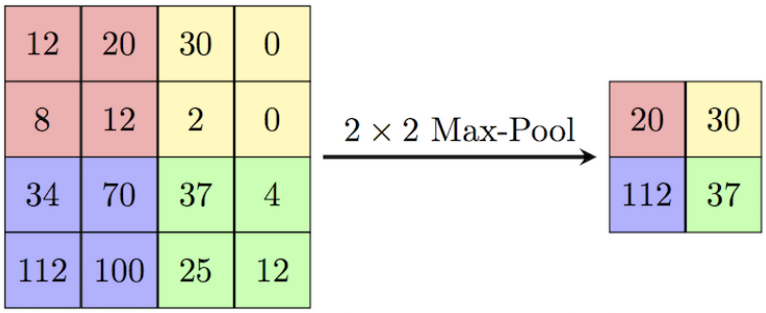


Figure A.5: Max-Pool applied to 4×4 input map, with kernel size 2×2 and stride (2,2). Image taken by [48].

Pooling layers are used in CNNs for dimensionality reduction, downsampling feature maps before the subsequent layers. Typically, the mostly used pooling layer is the *Max-Pooling*, but in some applications *Avg-Pooling* layers are applied as well. In Fig. A.5 a numerical example of *Max-Pool* applied to a 4×4 input map is shown.

Appendix B

Kullback-Leibler (KL) Divergence

KL-Divergence measures the statistical distance between 2 probability distributions $p(x)$ and $q(x)$. Let $p(x)$ the *true* distribution and $q(x)$ be the *approximated* version of $p(x)$; since they are both probability distributions, the conditions reported in Eq. B.1 always hold.

$$\int_{-\infty}^{\infty} p(x)dx = \int_{-\infty}^{\infty} q(x)dx = 1;\ p(x), q(x) > 0,\ \forall x \tag{B.1}$$

The KL-divergence is defined, in the discrete space, as in Eq. B.2, while its continuous counterpart is reported in Eq. B.3 [49].

$$D_{KL}(p(x)||q(x)) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{B.2}$$

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \tag{B.3}$$

Despite it gives indications about a distance, it cannot be considered a norm. In fact, in general $D_{KL}(p(x)||q(x)) \neq D_{KL}(q(x)||p(x))$, i.e. it is *asymmetrical*. Moreover, it does not satisfy the *triangle inequality* but the *positive definiteness* holds: $D_{KL}(p(x)||q(x)) = 0 \geq 0$, $D_{KL}(p(x)||q(x)) = 0 = 0 \Leftrightarrow p(x) = q(x)$.
In figure B.1 an histogram plot of 10000 randomly generated values sampled from a Normal distribution $q \sim \mathcal{N}(\mu = 0, \sigma = 1)$ is shown, alongside the theoretical probability distribution calculated according to Eq. B.4; both used to analyze the KL-divergence resulting by approximating the theoretical Gaussian distribution using the experimental one.

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{B.4}$$

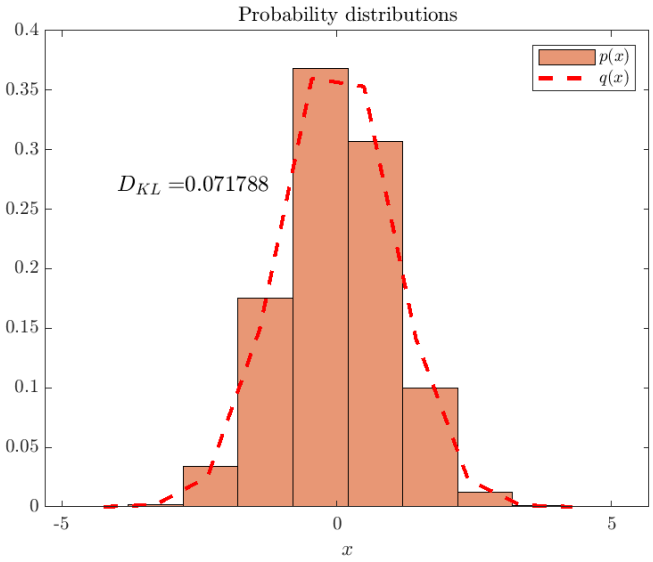


Figure B.1: Histogram plot of randomly generated data, along with theoretical Gaussian Distribution.

In Tab. B.1 the values of $p(x)$ and $q(x)$ are reported, used to compute the KL-divergence $D_{KL}(p(x)||q(x)) = 0.071788$.

x	$p(x)$	$q(x)$
-4.25e+00	4.79e-05	6.00e-05
-3.29e+00	1.72e-03	2.17e-03
-2.35e+00	2.51e-02	3.38e-02
-1.40e+00	1.49e-01	1.75e-01
-4.54e-01	3.60e-01	3.68e-01
4.94e-01	3.53e-01	3.07e-01
1.44e+00	1.40e-01	1.00e-01
2.39e+00	2.28e-02	1.25e-02
3.34e+00	1.51e-03	8.00e-04
4.29e+00	4.04e-05	1.00e-05

Table B.1: $p(x)$, $q(x)$ distribution values

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 2).

[2] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 2).

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. «Playing Atari with Deep Reinforcement Learning». In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602 (cit. on p. 3).

[4] Jing Li, Yanyang Liu, Xianguo Qing, Kai Xiao, Ying Zhang, Pengcheng Yang, and Yue (Marco) Yang. «The application of Deep Reinforcement Learning in Coordinated Control of Nuclear Reactors». In: *Journal of Physics: Conference Series* 2113.1 (2021), p. 012030. DOI: 10.1088/1742-6596/2113/1/012030. URL: https://doi.org/10.1088/1742-6596/2113/1/012030 (cit. on p. 3).

[5] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. «Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates». In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385 (cit. on p. 3).

[6] Xiaoyu Chen, Jiachen Hu, Chi Jin, Lihong Li, and Liwei Wang. «Understanding Domain Randomization for Sim-to-real Transfer». In: *CoRR* abs/2110.03239 (2021). arXiv: 2110.03239. URL: https://arxiv.org/abs/2110.03239 (cit. on p. 4).

[7] Ruan Pretorius and Terence van Zyl. *Deep Reinforcement Learning and Convex Mean-Variance Optimisation for Portfolio Management*. 2022. DOI: 10.48550/ARXIV.2203.11318. URL: https://arxiv.org/abs/2203.11318 (cit. on p. 4).

[8] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. «A Survey of Path Planning Algorithms for Mobile Robots». In: *Vehicles* 3.3 (2021), pp. 448–468. ISSN: 2624-8921. DOI: 10.3390/vehicles3030027. URL: https://www.mdpi.com/2624-8921/3/3/27 (cit. on p. 4).

[9] Nicoletta Bloise, Elisa Capello, Matteo Dentis, and Elisabetta Punta. «Obstacle Avoidance with Potential Field Applied to a Rendezvous Maneuver». In: *Applied Sciences* 7.10 (2017). ISSN: 2076-3417. DOI: 10.3390/app7101042. URL: https://www.mdpi.com/2076-3417/7/10/1042 (cit. on p. 4).

[10] Alfian Ma'arif, Oyas Wahyunggoro, and Adha Imam. «Artificial Potential Field Algorithm Implementation for Quadrotor Path Planning». In: *International Journal of Advanced Computer Science and Applications* 10 (Jan. 2019). DOI: 10.14569/IJACSA.2019.0100876 (cit. on p. 4).

[11] Sven Koenig and Maxim Likhachev. «Fast replanning for navigation in unknown terrain». In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 354–363 (cit. on p. 4).

[12] Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj, and Mahbube Ardani. «Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments». In: *CoRR* abs/2005.13857 (2020). arXiv: 2005.13857. URL: https://arxiv.org/abs/2005.13857 (cit. on p. 5).

[13] Chen Xia and A. KAMEL. «A Reinforcement Learning Method of Obstacle Avoidance for Industrial Mobile Vehicles in Unknown Environments Using Neural Network». In: Jan. 2015, pp. 671–675. ISBN: 978-94-6239-101-7. DOI: 10.2991/978-94-6239-102-4_136 (cit. on p. 5).

[14] Matej Dobrevski and Danijel Skočaj. «Deep reinforcement learning for mapless goal-driven robot navigation». In: *International Journal of Advanced Robotic Systems* 18.1 (2021), p. 1729881421992621. DOI: 10.1177/1729881421992621. eprint: https://doi.org/10.1177/1729881421992621. URL: https://doi.org/10.1177/1729881421992621 (cit. on p. 5).

[15] Umer Khan. «Mobile Robot Navigation Using Reinforcement Learning in Unknown Environments». In: *Balkan Journal of Electrical and Computer Engineering* 7 (Dec. 2020), pp. 235–244. DOI: 10.17694/bajece.532746 (cit. on p. 5).

[16] Oualid Doukhi and Deok-Jin Lee. «Deep Reinforcement Learning for End-to-End Local Motion Planning of Autonomous Aerial Robots in Unknown Outdoor Environments: Real-Time Flight Experiments». In: *Sensors* 21.7 (2021). ISSN: 1424-8220. DOI: 10.3390/s21072534. URL: https://www.mdpi.com/1424-8220/21/7/2534 (cit. on p. 5).

[17] Colin Greatwood and Arthur Richards. «Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control». In: *Autonomous Robots* 43 (Oct. 2019). DOI: 10.1007/s10514-019-09829-4 (cit. on p. 5).

[18] Oriol et al. Vinyals. «Grandmaster level in StarCraft II using multi-agent reinforcement learning.» In: *Nature* 575,7782 (2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z (cit. on p. 5).

[19] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. «The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games». In: *CoRR* abs/2103.01955 (2021). arXiv: 2103.01955. URL: https://arxiv.org/abs/2103.01955 (cit. on pp. 5, 38).

[20] Léon Reboul, Michel Kieffer, Hélène Piet-Lahanier, and Sébastien Reynaud. «Cooperative guidance of a fleet of UAVs for multi-target discovery and tracking in presence of obstacles using a set membership approach». In: *IFAC-PapersOnLine* 52.12 (2019). 21st IFAC Symposium on Automatic Control in Aerospace ACA 2019, pp. 340–345. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.11.266. URL: https://www.sciencedirect.com/science/article/pii/S2405896319312182 (cit. on p. 5).

[21] Nesrine Mahdoui, Vincent Fremont, and Enrico Natalizio. «Communicating Multi-UAV System for Cooperative SLAM-based Exploration». In: *Journal of Intelligent and Robotic Systems* 98 (May 2020). DOI: 10.1007/s10846-019-01062-6 (cit. on p. 5).

[22] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. «Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms». In: *CoRR* abs/1911.10635 (2019). arXiv: 1911.10635. URL: http://arxiv.org/abs/1911.10635 (cit. on p. 5).

[23] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. «Counterfactual Multi-Agent Policy Gradients». In: *CoRR* abs/1705.08926 (2017). arXiv: 1705.08926. URL: http://arxiv.org/abs/1705.08926 (cit. on p. 6).

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. «Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments». In: *CoRR* abs/1706.02275 (2017). arXiv: 1706.02275. URL: http://arxiv.org/abs/1706.02275 (cit. on p. 6).

[25] Hassam Ullah Sheikh and Ladislau Bölöni. «Multi-Agent Reinforcement Learning for Problems with Combined Individual and Team Reward». In: *CoRR* abs/2003.10598 (2020). arXiv: 2003.10598. URL: https://arxiv.org/abs/2003.10598 (cit. on p. 6).

[26] Xiaolong Wei, Lifang Yang, Gang Cao, Tao Lu, and Bing Wang. «Recurrent MADDPG for Object Detection and Assignment in Combat Tasks». In: *IEEE Access* 8 (2020), pp. 163334–163343. DOI: 10.1109/ACCESS.2020.3022638 (cit. on p. 6).

[27] Yuxie Luo, Jia Song, Kai Zhao, and Yang Liu. «UAV-Cooperative Penetration Dynamic-Tracking Interceptor Method Based on DDPG». In: *Applied Sciences* 12 (Feb. 2022), p. 1618. DOI: 10.3390/app12031618 (cit. on p. 6).

[28] Maryam Kouzehgar, Malika Meghjani, and Roland Bouffanais. «Multi-Agent Reinforcement Learning for Dynamic Ocean Monitoring by a Swarm of Buoys». In: *CoRR* abs/2012.11641 (2020). arXiv: 2012.11641. URL: https://arxiv.org/abs/2012.11641 (cit. on p. 6).

[29] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Luan Van Nguyen. «Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage». In: *CoRR* abs/1803.07250 (2018). arXiv: 1803.07250. URL: http://arxiv.org/abs/1803.07250 (cit. on pp. 6, 23).

[30] Mohammadreza Davoodi, Javad Mohammadpour Velni, and Changying Li. «Coverage Control with Multiple Ground Robots for Precision Agriculture». In: *Mechanical Engineering* 140.06 (June 2018), S4–S8. ISSN: 0025-6501. DOI: 10.1115/1.2018-JUN-4. eprint: https://asmedigitalcollection.asme.org/memagazineselect/article-pdf/140/06/S4/6384102/me-2018-jun4.pdf. URL: https://doi.org/10.1115/1.2018-JUN-4 (cit. on p. 6).

[31] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. «Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning». In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14413–14423. DOI: 10.1109/TVT.2020.3034800 (cit. on p. 6).

[32] Adekunle A. Adepegba, Md. Suruz Miah, and Davide Spinello. «Multi-Agent Area Coverage Control Using Reinforcement Learning». In: *FLAIRS Conference*. 2016 (cit. on p. 6).

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 8).

[34] G. Cybenko. «Approximation by superpositions of a sigmoidal function». In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274 (cit. on p. 12).

[35] Kurt Hornik. «Approximation capabilities of multilayer feedforward networks». In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T (cit. on p. 12).

[36] OpenAI. *Kinds of RL algorithms*. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html (cit. on p. 13).

[37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. «Proximal Policy Optimization Algorithms». In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (cit. on pp. 14, 15).

[38] OpenAI. *Proximal Policy Optmization*. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#pseudocode (cit. on pp. 16, 41).

[39] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. «High-Dimensional Continuous Control Using Generalized Advantage Estimation». In: (). URL: https://arxiv.org/abs/1506.02438 (cit. on p. 17).

[40] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. «scikit-image: image processing in Python». In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453 (cit. on p. 23).

[41] Bernhard Hengst. «Hierarchical Reinforcement Learning». In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 495–502. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_363. URL: https://doi.org/10.1007/978-0-387-30164-8_363 (cit. on p. 25).

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification». In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852 (cit. on p. 30).

[43] «DJI Mavic 2 Support Page». In: (). URL: https://www.dji.com/it/mavic-2/info#specs (cit. on p. 36).

[44] Juan Zhang, James F. Campbell, Donald C. Sweeney II, and Andrea C. Hupman. «Energy consumption models for delivery drones: A comparison and assessment». In: *Transportation Research Part D: Transport and Environment* 90 (2021), p. 102668. ISSN: 1361-9209. DOI: https://doi.org/10.1016/j.trd.2020.102668. URL: https://www.sciencedirect.com/science/article/pii/S1361920920308531 (cit. on pp. 36, 37).

[45] *Gazebo*. URL: https://gazebosim.org/home (cit. on p. 74).

[46] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. URL: https://www.ros.org (cit. on p. 74).

[47] *PX4 Autopilot: Open Source Autopilot for Drones*. URL: https://px4.io/ (cit. on p. 74).

[48] Computer Science Wiki. *Max-pooling / Pooling — Computer Science Wiki,* 2018. URL: https://computersciencewiki.org/index.php?title=Max-pooling_/_Pooling&oldid=7839 (cit. on p. 84).

[49] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining concepts and techniques, third edition*. 2012 (cit. on p. 85).