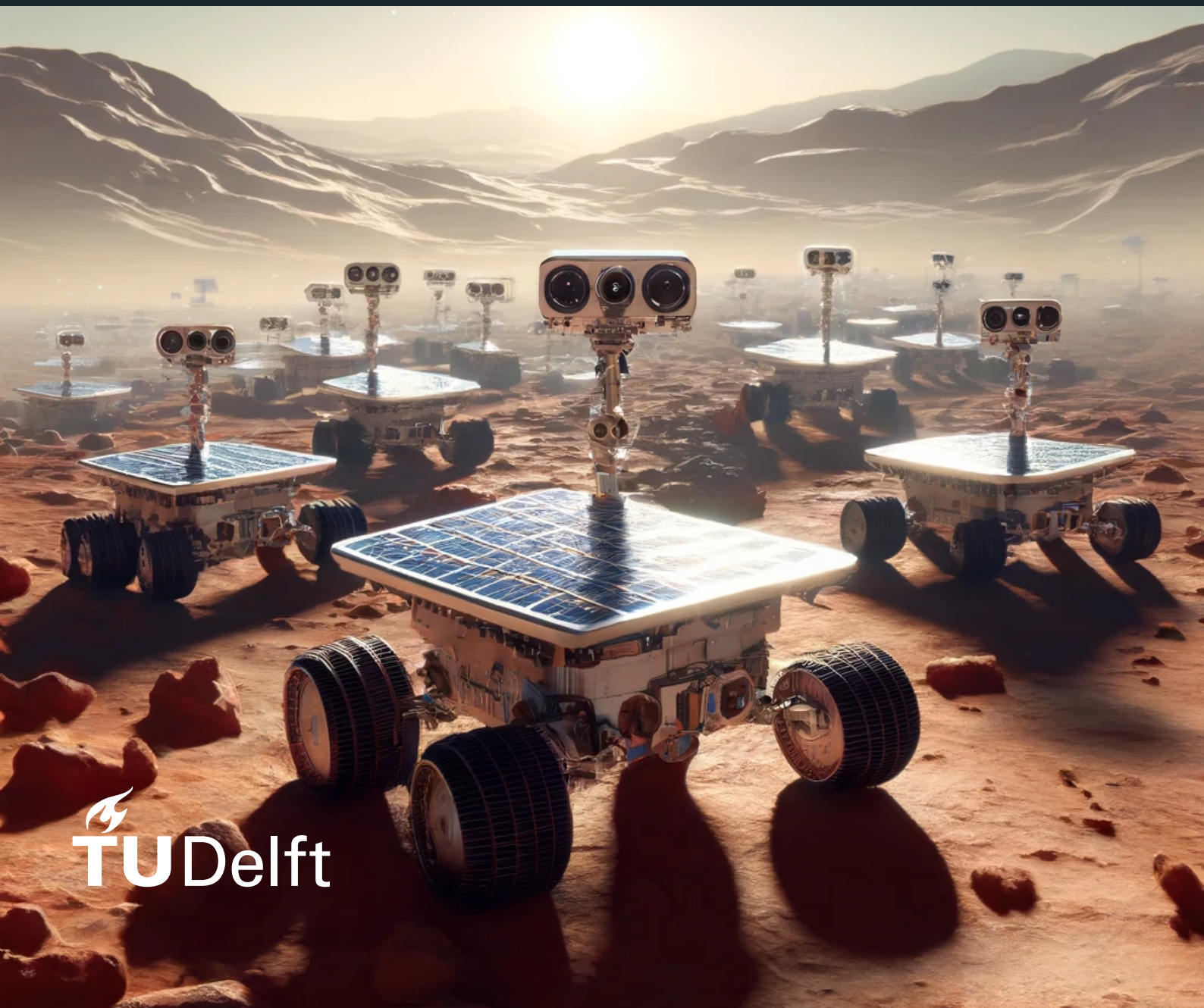
Multi-Agent Reinforcement Learning for Swarm Planetary Exploration

Adrián Menor de Oñate



TU Delft

Multi-Agent Reinforcement Learning for Swarm Planetary Exploration

Thesis report

by

Adrián Menor de Oñate

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on May 24, 2024 at 10:00

Thesis committee:

Chair:	Dr.ir. J. Ellerbroek
Supervisor:	Dr.ir. E. van Kampen
External examiner:	Dr.ing. J. Kober
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	May, 2023 - May, 2024
Student number:	4668901

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Faculty of Aerospace Engineering · Delft University of Technology

Preface

This report summarises my one-year journey delving into the topic of multi-agent reinforcement learning, where I investigated how to improve the intelligence of robot teams with the idea of exploring planetary bodies. My goal has been to try to contribute to the field and learn as much as I can. Multi-agent systems such as the swarms found in nature or human societies fascinate me, with their seemingly magical emergence properties, and I have been lucky to work on this matter for a year. In this regard, I would like to thank my thesis supervisor, Dr.ir. E. van Kampen, who trusted me to dive into this topic, and who gave me a perfect balance between supervision and having the freedom to explore different research directions. The main code used to produce the results in this thesis is made publicly available[1].

Personally, this thesis is the culmination of my TU Delft journey, which started seven years ago. With the bittersweet feeling of saying ”goodbye” to this part of my life journey, I am looking back at all these years, glancing at all the memories that I will now take with me wherever I go. Studying here has never been a given, but a gift. A lot has changed in the kid who first stepped foot into the Faculty of Aerospace Engineering. And this is thanks to you, the people that surround me.

A mi familia. Nunca os podré agradecer todo el cariño que me dais, vuestro apoyo incondicional. Tengo la suerte de poder decir que sois muchos, y aunque no os puedo mencionar individualmente, tened por seguro que con estas líneas os estoy dando las gracias a cada uno de vosotros; primos, tíos, abuelos, y demás acepciones que no están en la definición ”clásica” de familia. Por supuesto, dar especialmente las gracias a mis padres Luis y Mabel, que siempre han hecho todo lo posible e imposible porque crezca como persona. También a vosotros, mi hermano Pablo, Mario, Celia y Paula, junto a los que he crecido, y crezco. A mis abuelos Matías e Isabel, en los que me apoyo mucho. Finalmente, a los que me han visto empezar, pero no terminar; mis abuelos Luis y Paco, os echo de menos. Os quiero mucho a todos.

I am extremely thankful for my friends; the Foulkeslanders, Bachelor friends, the ones from Spain, Switzerland... you know who you are. I am very grateful for sharing these past years with you all, and I feel extremely lucky to have met such incredible people whom I admire and learn so much from. We have built a ”family” abroad, shared highs and lows, and grown so much together. I hope that our friendships last forever, or until my simulations converge, which is still a lot of time.

Muchas gracias a todos, os quiero muchísimo.

[1] https://github.com/adrianmenor/MAPPO_for_swarm_planetary_exploration

Contents

Nomenclature

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
CADRE	Cooperative Autonomous Distributed Robotic Exploration
CNN	Convolutional Neural Network
CTDE	Centralised Training, Decentralised Execution
DDPG	Deep Deterministic Policy Gradient
DEC-MDP	Decentralised Markov Decision Process
DEC-POMDP	Decentralised Partially Observable Markov Decision Processes
DLR	German Aerospace Center
DNN	Deep Neural Network
DP	Dynamic Programming
DRL	Deep Reinforcement Learning
FF	Feed Forward Neural Network
HPC	High Performance Computing
IPPO	Individual PPO
LIDAR	Light Detection and Ranging
MaCMAS	Methodology Fragment for Analysing Complex Multi-agent Systems
MAPPO	Multi-Agent PPO
MARL	Multi-Agent Reinforcement Learning
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean Squared Error
NASA	National Aeronautics and Space Administration
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RQ	Research Question
SAC	Soft Actor-Critic
TD	Temporal Difference
TRL	Technology Readiness Level

List of Symbols

δ	TD-error
t	Time-step
γ	Discount factor
π	Policy
A	Action space
a	Action
q_π	Action-value function of π
S	State space
s	State
v_π	Value function of π

List of Figures

List of Tables

1 Introduction

Over the past half century, space exploration missions have relied on single space vehicles [1]. These vehicles tend to be highly complex and expensive, and can compromise the exploration mission if a failure occurs.
Because of their resiliency, low cost, and adaptability, teams and swarms of autonomous robots are being considered to undertake future space exploration missions [2]. In this context, a swarm of space vehicles is a *"collection of often smaller and simpler, autonomous vehicles that coordinate in a decentralized manner to achieve a common goal"* [2]. Exploration tasks performed by a single, complex vehicle, could be done by a collection of simpler vehicles instead.
These swarms have the potential to yield reduced cost and greater risk tolerance by using several, simpler assets, that can perform the mission [2]. Additionally, launch costs can be reduced by gradually launching the assets, and also using them as secondary payloads [2]. When looking at the nature of the exploration missions, swarms open the possibility of increasing the value of the space mission, and *"open a new world of science exploration"* [3], allowing for performing measurements and explorations that were previously impractical or impossible; executing large spatial aperture missions such as seismological investigations, fast terrain measurements, interferometry, exploration of Moon and Mars lava tubes, etc, as recognized by the German Aerospace Center (DLR) [4] and NASA's JPL [2]. In fact, the potential for swarm exploration missions is reflected in one of the next launches to the Moon, in 2024, where NASA will deploy its CADRE robots [3] for the first time, shown in Figure 1.1.

Figure 1.1: Two CADRE rovers tested at NASA's JPL Mars Yard, August 2023. Retrieved from [5].

Moreover, using swarm autonomous missions allows for exploring deeper into the cosmos. As suggested by NASA [3], establishing communication with Earth is a practical choice for lunar missions, given the swift transmission of signals. However, the delay caused by waiting for ground control to assess the data, make decisions, and send instructions consumes valuable time. Furthermore, for missions to more distant locations like Mars and Jupiter's moon Europa, communication can be significantly delayed, making reliance on ground-based transmissions cumbersome. Enabling robots to collaborate and explore autonomously would significantly enhance the scientific outcomes of such missions [3].

Despite its potential and recent successes, the technology readiness level (TRL) of swarm space exploration missions is still very mature, requiring the development of new technology in a diversity of areas [2].

At the same time, the field of artificial intelligence is experiencing a booming period of discoveries and developments. Particularly, this research investigates the application of multi-agent reinforcement learning (MARL) on swarm missions, intending to contribute to their TRL. This field has immense potential, as shown by the rapid increase in research it is experiencing, observed in Figure 1.2.

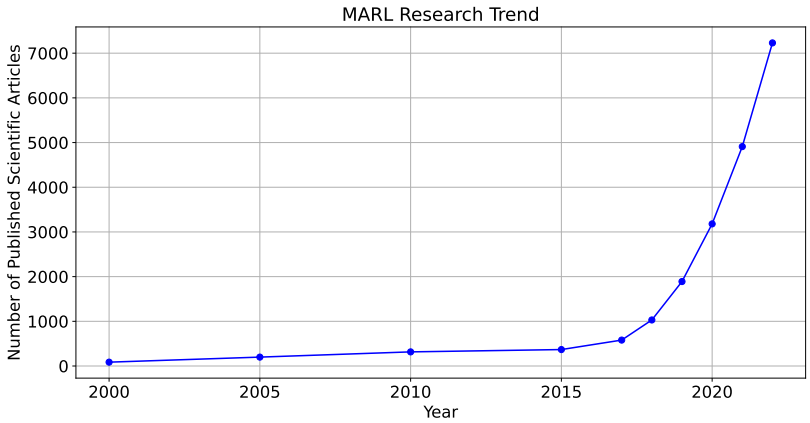


Figure 1.2: Number of scientific articles related to multi-agent reinforcement learning, per year, according to Google Scholar.

Multi-Agent Reinforcement Learning

Reinforcement learning (RL) stands as a cornerstone in the evolution of artificial intelligence and autonomous systems. Its historical journey spans several decades, witnessing remarkable advancements and milestones.

Learning by interacting with an environment can be arguably regarded as one of the most natural ways of learning [6, p. 2]. It can be observed in nature, for example, when an infant is trying to learn how to walk it may not have an explicit teacher, but by **interacting** with the environment (with which it has a sensorimotor connection) it learns how to stand up, and eventually move. This occurs in an iterated fashion, with the infant trying to stand up and balance itself in different ways, and **observing** how its actions influence how well it can reach the goal of walking.

Similarly, RL is the *computational* approach to mimic such learning by interaction that is often found in nature [6, p. 2].

MARL applies reinforcement learning to systems with several agents. Multi-agent systems consist of a group of autonomous, interactive entities that all share a common environment, which they perceive with sensors and in which they can act.

MARL has the potential to learn or discover control policies that are difficult or virtually impossible to find by pre-established heuristics or priors, by interacting with a simulated environment and receiving a performance-based feedback signal (the reward) which can be designed from the mission requirements. In this sense, the found policies can make a collection of individual agents achieve a global goal.

Research Gap

Despite the advancements in the field of MARL, its application in the domain of space exploration remains largely unexplored. This research aims to bridge this gap by investigating the potential utilization of MARL techniques in space exploration, with the objective of contributing substantive insights and advancements to this emerging area of study.

With this in mind, Section 1.1 discusses the research questions formulated to investigate using MARL for space exploration. Following that, Section 1.2 delineates the boundaries within which these questions will be addressed. Finally, Section 1.3 explains the report structure.

1.1. Research Formulation

The present work aims at fulfilling the following research objective:

Research Objective

Improve the technology readiness level of swarm space missions by developing a multi-agent reinforcement learning framework to discover swarm control policies.

To achieve the research objective, the following research questions have been established:

Research Question 1

How can reinforcement learning be used to contribute to the autonomy of swarm space exploration missions?

Research Question 2

What are the main requirements of swarm space exploration missions?

1. Which swarm space exploration requirements can-not be satisfied with reinforcement learning given the available resources?
2. Among the different tasks of a space mission, upon which task(s) should this research be focused, given the suitability of reinforcement learning and the available resources?

Research Question 3

Which reinforcement learning setup and algorithm(s) satisfy these requirements?

1. Which of the identified swarm-mission requirements are satisfied by existing MARL algorithms applied in this field?
2. Which architecture/hierarchy is more suitable for swarm space exploration?
3. What are the current state-of-the-art reinforcement learning algorithms applied to swarms?
4. How should the value and policy functions be approximated?
5. What are the limitations of using single-agent reinforcement learning algorithms such as SAC and PPO directly on swarms?
6. Which type of reward function promotes better learning (in terms of sample efficiency, or convergence capabilities)?

Research Question 4

How can the selected reinforcement learning algorithm be verified?

1. How can adaptive and swarm systems be verified?
2. How can reinforcement learning systems for swarm exploration be verified?

Research Question 1 delves into the applicability of reinforcement learning within the domain of swarm space exploration. After this, **Research Question 2** inspects the nature of swarm space exploration missions, intending to provide a clear problem formulation within which to develop a MARL framework. Following, **Research Question 3** delves into the specifics of aligning the fields of MARL and space explo-

ration; inspecting critical MARL characteristics that can lead to the ultimate development of a successful learning algorithm. Lastly, **Research Question 4** explores how to verify the developed technology.

1.2. Scope of Research

The scope of the research conducted in this thesis is twofold. Firstly, it aims to bridge the alignment gap between space exploration and MARL. This entails investigating how MARL techniques can be effectively applied to space exploration scenarios. Secondly, the research delves into the offline learning of swarm control policies in a specifically developed simulated environment resembling a swarm exploration problem. This involves developing and evaluating algorithms that enable agents to achieve exploration objectives. A particular emphasis is placed on the algorithmic aspect of the problem, wherein various state-of-the-art solutions are scrutinized and compared to then develop a MARL algorithm that is specific to the nature of the problem.

The preliminary analysis investigates the aforementioned alignment problem and proposes a specific swarm system architecture. This study does not produce an exhaustive development of such a system, instead, it uses it to provide a starting point for envisioning a swarm planetary mission and contributing to it in a place with a low TRL.
Furthermore, a few state-of-the-art single-agent RL algorithms are used to solve a simple multi-agent control problem, with the intention of understanding the fundamental limitations of naively using RL in multi-agent domains.

Secondly, the later part of the thesis focuses on the development and assessment of a MARL algorithm. The generated algorithm is not intended to be readily applicable in real missions, but a significant step forward in the field.
Similarly, the verification efforts in this research are focused on ensuring that the developed algorithms and framework function as intended. Delving into a full verification and validation of a swarm space exploration system is thus considered to be outside of the scope of this work, although further verification efforts have been studied in the literature study.

1.3. Report Structure

This report comprises four main parts, each contributing to a comprehensive understanding of the research conducted.

Part I, the Scientific Article, serves as the core of the report, offering a concise exploration of the research. It commences with a concise background, providing contextual information on the swarm exploration problem. Following this, the problem formulation is explained, delineating the objectives and challenges addressed by the research. The developed algorithm is then described in detail. Finally, the main results obtained from this research are presented, offering insights into the outcomes and implications of the study.

Part II focuses on the Literature Study, which serves as a foundational component of the research. This section delves into the fields of space exploration, swarming, and reinforcement learning, examining existing literature and state-of-the-art algorithms. Additionally, it performs a preliminary analysis of algorithms and reward functions, implementing single-agent RL algorithms in a simple multi-agent task, and proposes a study plan and swarm system architecture.

Part III comprises Additional Results, providing supplementary insights and analyses to complement the core research. Here, the overall methodology employed in the research is discussed, offering clarity on the approach utilized. Furthermore, additional results, hyperparameter tuning, and verification and validation processes are discussed.

Lastly, Part IV is the Closure, examining the research questions, findings, and contributions. This section provides critical reflections on the outcomes of the study, offering insights into the implications and significance of the research. Additionally, recommendations are highlighted for future research, identifying potential research directions in the field.

Part I

Scientific Article

Multi-Agent Reinforcement Learning for Swarm Planetary Exploration

A. Menor de Oñate*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Exploring planetary bodies using robot swarms can potentially increase the value of the exploration missions; enabling the execution of novel measurements and explorations previously deemed impractical or unattainable. Despite its potential, the technology readiness level of planetary swarms is not very mature. This work uses multi-agent reinforcement learning to find control policies that allow swarms to autonomously explore unknown areas in a decentralized manner, contributing towards the technology readiness of the field. A multi-agent proximal policy optimization (MAPPO) algorithm is proposed for this end, where the policy uses LIDAR perception information, and the input of the value function contains local and global environment information. The algorithm finds control policies that achieve cooperation behaviors and generalize to unseen swarm sizes and environments learning with simple, sparse reward functions. Moreover, different types of reward functions, value inputs, and environment configurations are investigated. Compared with the state-of-the-art in the field, MAPPO can learn with a larger number of agents, more complicated environments, and using sparse rewards instead of dense ones.

Keywords: **Multi-agent PPO, Planetary Exploration, Swarm Intelligence.**

Nomenclature

a.u.	=	Arbitrary Units
CNN	=	Convolutional Neural Network
CTDE	=	Centralised Training, Decentralised Execution
DEC-MDP	=	Decentralised Markov Decision Process
DEC-POMDP	=	Decentralised Partially Observable Markov Decision Processes
IPPO	=	Individual PPO
LIDAR	=	Light Detection and Ranging
MaCMAS	=	Methodology Fragment for Analysing Complex Multi-agent Systems
MAPPO	=	Multi Agent PPO
MARL	=	Multi-Agent Reinforcement Learning
MDP	=	Markov Decision Process
ML	=	Machine Learning
POI	=	Point of Interest
POMDP	=	Partially Observable Markov Decision Process
PPO	=	Proximal Policy Optimisation
RL	=	Reinforcement Learning
TRL	=	Technology Readiness Level

I. Introduction

Over the past half century, planetary exploration missions have relied on single space vehicles [1]. These vehicles are highly complex and expensive and can compromise the exploration mission if a failure occurs.
Because of their resiliency, low cost, and adaptability, teams and swarms of autonomous robots are being considered to undertake future space exploration missions [2]. In this context, a swarm of space vehicles is a *"collection of often*

*MSc. Student, Faculty of Aerospace Engineering, Control and Simulation Division, Delft University of Technology.

smaller and simpler, autonomous vehicles that coordinate in a decentralized manner to achieve a common goal" [2]. Exploration tasks performed by a single, complex vehicle, could be done by a collection of simpler vehicles instead.

Swarms unveil the possibility of increasing the value of the planetary exploration missions, and *"open a new world of science exploration"* [3], allowing for performing measurements and explorations that were previously impractical or impossible; executing large spatial aperture missions such as seismological investigations, fast terrain measurements, interferometry, exploration of Moon and Mars lava tubes, etc, as recognized by the German Aerospace Center [4] and NASA's JPL [2].
As outlined in *Intelligence in Future NASA Swarm-based Missions* [5], to achieve the aforementioned missions, swarms must be **autonomous**, **intelligent**, and have the capability of **learning from their environment**.

In this regard, the technology for deploying swarm missions for planetary exploration is not fully mature. In *Space Vehicle Swarm Exploration Missions: A Study of Key Enabling Technologies and Gaps* [2] it is highlighted that most of the technology readiness levels (TRLs) for performing planetary exploration, mapping and sampling, and cooperative task recognition and allocation are not very mature (3-5 TRL), or not available/ in conceptual stages of development (1-2 TRL).

In this work, multi-agent reinforcement learning (MARL) is studied to contribute to the readiness of swarm missions. MARL belongs to the field of machine learning (ML) methods, specializing in finding control strategies that can make a collection of individual entities achieve a global goal. As such, it has the potential to learn or discover control policies that are difficult or virtually impossible to find by pre-established heuristics or priors, by interacting with a simulated environment and receiving a performance-based feedback signal (the reward) which can be designed from the mission requirements.

However, the MARL optimization problem remains an extremely challenging task [6–8], with actor-critic MARL architectures (see section II) being used in the state-of-the-art of the field. A multi-agent extension of the Deep Deterministic Policy Gradient RL algorithm has been proposed in state-of-the-art studies in MARL technologies for planetary exploration [9].
However, in this architecture, the agent training scalability is severally limited (5 agents or less), the robustness of the learned policies is not addressed, and the reward function is dense; thus heavily relying on heuristics which may limit the potential of the swarm.

This work proposes a MARL algorithm that achieves salient state-of-the-art performance in the field of MARL applied to swarm planetary exploration in the domains of agent scalability, agent perception, reward function flexibility, and policy analysis while considering NASA mission requirements. The algorithm is a multi-agent extension of Proximal Policy Optimisation (PPO) which uses a simulated LIDAR for the individual perception of each agent, and a global environment tensor processed by a convolutional neural network modeling the critic of the agents.

The contributions of this study are summarised as follows:

- A framework is proposed to improve the intelligence of swarm missions, integrating the field of MARL with NASA swarm architectures, and targeting areas with low TRLs. Specifically, the objective is obtaining local cooperative guidance policies that achieve a global mission goal, and investigating the usage of MARL for policy discovery in complex settings. This is the first time NASA swarm mission requirements and MARL solutions for this end are analyzed.
- A Multi-Agent PPO algorithm specifically tailored for swarm planetary exploration is developed. Different from other works in MARL, there is no explicit use of dense reward functions, thus potentially allowing for the implementation of reward functions derived directly from mission requirements. Furthermore, the algorithm also allows for homogeneous and heterogeneous agent populations and operates in continuous action spaces.
- A planetary swarm exploration environment is developed, which has more complexity than the environment used in [9] regarding the amount and placement of targets, agents, and obstacles, where the proposed algorithm displays state-of-the-art performance, also achieving state-of-the-art agent scalability and local policies that can adapt to environments and swarm sizes that have not been seen during training, and show emergent behaviors.
- Different algorithm, environment, and reward function configurations are examined to assess how they affect learning performance, policy generalisability, and gain insights into the MARL problem, thus further improving the TRL of swarm exploration missions.

II. Background

This section provides relevant background information on the knowledge blocks on which the proposed algorithm is based. In particular, single-agent RL, actor-critic methods, and MARL are discussed. Next, relevant prior studies are cited to offer contextual insight drawn from the aerospace and machine learning fields and highlight the focus of this research.

A. Single-Agent RL

Reinforcement learning (RL) is a sub-field of machine learning that does not fully lie in the supervised or unsupervised learning category. Supervised learning requires a set of labeled data examples that are used for training as well as assessing the extrapolation/generalization capabilities of the ML model. In interactive scenarios, it is often impractical to have such labeled data [10, p. 2]. On the other hand, unsupervised learning usually deals with finding structures that are hidden in data that is unlabelled [10, p. 2]. This is also different from RL; although uncovering structure in an agent's experience can certainly be useful in RL, *"by itself does not address the reinforcement learning agent's problem of maximizing a reward signal"*[10, p. 3].

In RL, an agent interacts with an environment with the objective of maximising a numerical reward function. In this sense, the agent can observe the environment and take actions that will change the state of this environment. The state of the environment affects the value of the reward function.

Disregarding the agent and the environment, three elements of RL that are important in this work:

- The **policy** is the mapping from the environment states to the actions to be taken in those states [10, p. 7]. Specifically, a policy π is a mapping from each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(a|s)$ of taking action a when in state s [10, p. 70].
- The **reward signal** defines the goal in the RL problem. The environment sends the agent a reward signal depending on the state achieved by the agent and the current action that is taken. The goal of the agent is to maximize the total reward.
- The **value function** indicates the total amount of reward an agent can expect in the future [10, p. 7]. Particularly, for Markov Decision Processes (MDPs) the value of a state under a policy π is defined in Equation 1, where the discount factor γ is a number between 0 and 1 used to give more or less importance to future rewards R [10, p. 70].

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{1}$$

B. Actor-Critic Methods

There are myriad possible RL architectures to guide the learning of a successful control policy. Here, **actor-critic** is described, given the developed algorithm is of this form. Actor-critic methods are RL methods where the policy (actor) and value (critic) functions are represented independently [10, p. 257]. The information outputted from the critic is also included in the objective function that needs to be maximized by the actor. In this research, deep artificial neural networks are used to model these functions, and the output from the critic is used to update the actor via including advantage estimation (see algorithm Equation 1) in the objective.

Furthermore, the algorithm proposed is on-policy. On-policy algorithms update on the **current** policy used during the training episode. Moreover, two strengths of actor-critic are the minimal computational requirements needed to select actions, as well as its ability to learn stochastic policies (selecting the optimal probabilities of selecting various actions) [10, p. 259], something that is useful on non-Markov cases (as is the case of MARL).

C. MARL

MARL applies the aforementioned schemes of RL to systems with several agents. Multi-agent systems consist of a group of autonomous, interactive entities that all share a common environment, which they perceive with sensors and in which they can act through actuators. In this work, collaborative swarms are considered.

MARL scenarios have certain distinctive characteristics compared with the traditional RL paradigm. The fundamental design of MARL algorithms considers aspects from Temporal Difference methods, Game Theory, and Direct Policy Search [6, p. 13]. Classic RL theory is grounded in the assumption of MDPs. In MARL this assumption is often not valid [6], as the problem becomes a partially observable MDP (POMDP), a decentralized MDP (DEC-MDP), or as

is the case in this study, a decentralized, partially observable MDP (DEC-POMDP), which is notoriously difficult to solve and can require super-exponential time in the worst case [7, 8]. This is because DEC-POMDPs model problems in which multiple agents, each possessing partial observations of the environment, make decisions simultaneously to achieve a common objective.

MARL presents some additional challenges compared to classic single-agent reinforcement learning, which results in the need to modify the structure of the learning algorithms, and the perception of the agents. Here we discuss some of the most salient characteristics/challenges of MARL:

- In multi-agent scenarios, the environment gets affected by the actions of all agents, making it non-stationary. Consequently, each agent encounters a dynamic learning challenge where the optimal policy continually shifts in response to changes in the policies of other agents.
- MARL algorithms with centralized training often suffer from the *"curse of dimensionality"* [6], where the input size of the value functions grows exponentially as the number of agents is increased since each new agent adds its own variables to the environment.
- The training stability of the agents (the convergence to a stationary policy) needs a trade-off with the adaptation capabilities with respect to the changing behavior of the other agents [6]. Thus if the policy rapidly updates to react to the other agents, it is likely to have an oscillatory performance during training.
- In accordance with the aforementioned information, the overall learning stability of MARL algorithms remains a challenging task [6].
- MARL settings are susceptible to fall in local minima (which can be related to fall in a suboptimal Nash equilibrium).
- In this study sparse reward functions are allowed (since we propose a framework to find control policies in situations where the problem information is limited), and this can further hinder learning performance.

MARL algorithms typically lie within two frameworks: centralized and decentralized learning [11]. Here both options are explored; the proposed algorithm has a policy that only perceives a local state of the environment, and value functions that can contain the global information during training. Combining a local policy with a global value function is referred to as **Centralised Training, Decentralised Execution** (CTDE). CTDE is usually selected as the approach to solving the DEC-POMDP in state-of-the-art MARL algorithms [12], especially in actor-critic architectures.

Having value functions that use some form of global information helps relax the issue of the environment being non-stationary, but if this global information scales poorly as more agents are added to the environment, the input to the value function becomes too large, leading to the aforementioned curse of dimensionality.

D. Related Work

In recent research, the field of MARL has witnessed significant progress. Specific to planetary and space exploration, [9] showcased the effectiveness of a multi-agent deep deterministic policy gradient (MADDPG) algorithm, achieving successful learning outcomes in simple planetary exploratory tasks with five agents or less, and dense reward functions. Additionally, NASA explored deep Q-learning and Advantage Actor-Critic in [13] for routing and link selection in networks comprising 100 nodes.

Furthermore, studies in MARL have demonstrated the efficacy of PPO in multi-agent scenarios, often outperforming other algorithms and/or achieving state-of-the-art performance in both collaborative and adversarial settings [11, 14]. While both CTDE and decentralized PPO formats have been explored, the latter has shown suboptimal performance compared to alternatives like QMix in specific environments such as SMAC [11]. Moreover, leveraging deep reinforcement learning in multi-agent settings with visual inputs has been investigated in [15]. Recently, multi-agent PPO (MAPPO) has also been applied to cooperative UAV trajectory design [16].

This research focuses on the specific problem of planetary exploration, using multi-agent PPO to find control policies in exploratory environments. Moreover, and specific to this work, several reward functions, environments, and swarm sizes are examined to foster the finding of exploratory policies that have emergent behaviors and generalize to unseen environments and swarm sizes.

III. Problem Formulation

This section describes how the MARL problem is formulated; first discussing where the policies obtained through the MARL algorithm can be deployed in a planetary mission, and later defining the formal optimization problem that needs to be solved for this end.

A. MARL Within the Planetary Exploratory Mission

In this study, a focus has been placed on contributing to areas with low TRLs [2], in particular; exploration, mapping and sampling, and cooperative task recognition and allocation. In this manner, a swarm exploration traceability model is proposed (see Figure 1), following NASA's *Methodology Fragment for Analysing Complex Multi-agent Systems* (MaCMAS) [17]. MaCMAS generates models at various abstraction levels to manage system complexity incrementally. This approach links detailed micro-level models with abstracted macro-level models, ensuring property verification across different system levels using formal methods. Additionally, MaCMAS offers techniques to refine and abstract models for comprehensive layer completion. Using MaCMAS for the mission design provides a clear reference of the area of contribution in this work, also highlighting other areas where research can be done.

In this manner, this study does not produce an exhaustive development of such a MaCMAS system, instead, it uses it to provide a starting point for envisioning a swarm planetary mission and contributing to it in a place with a low TRL. In the traceability model in Figure 1, six abstraction levels are envisioned; ranging from top-level mission planning, to agent-level control. Within each level, tasks/roles are defined; establishing a mission plan, identifying the task(s) that need to be executed, assembling a team of robots to do so, etc.

The MARL algorithm in this work aims at contributing to specific mission configurations, connected with black arrows in Figure 1. In such configurations, the mission entails using a swarm of land-based or airborne robots, all of the same type, who explore Points of Interest (POIs, which are also referred to as targets), and where the robots in the swarm can sense each other, but not directly communicate. Particularly, this work proposes to deploy MARL at the cooperation strategies level, a level of abstraction above agent-level control (see Figure 1), and for which there is a 3-5 TRL [2]. This way, the commands from the policy to the robot are guidance actions, stating where should the robot go next, based on its local perception. With these mission configurations, the swarms could explore complex terrains such as caves or lava tubes.

Here, the focus is on contributing to the following NASA requirements [2] (that is, not necessarily having complete fulfillment of all requirements, but showing that through MARL and the discovered policies, contributions are with respect these requirements):

Req 1. Exploration, mapping, and sampling

1) **Autonomy**: the platforms should be able to operate for hours to days with no humans in the loop.
2) Relative pose estimation.
3) Formation keeping: the vehicles should be able to assess and, in many applications, control their relative location to a high degree of accuracy.
4) **Distributed estimation and cooperative mapping** to build real-time maps of the observed quantities and enable adaptive sampling strategies.
5) Distributed inter-agent communication.

Req 2. Cooperative Task and Task Allocation

1) **Recognise tasks** that should be performed based on environmental cues observed by the agents.
2) Assign such tasks according to the **agents' states and capabilities**.

As mentioned, from a mission perspective, **the goal is to obtain a control policy that can be used for the guidance command of robots in swarms performing planetary exploration.** This policy needs to be obtained using reward functions that follow straightforwardly from the mission requirements and require as little heuristics or human input as possible. In this way, scientists can potentially find policies that surpass human design or provide new insights into the control problem.

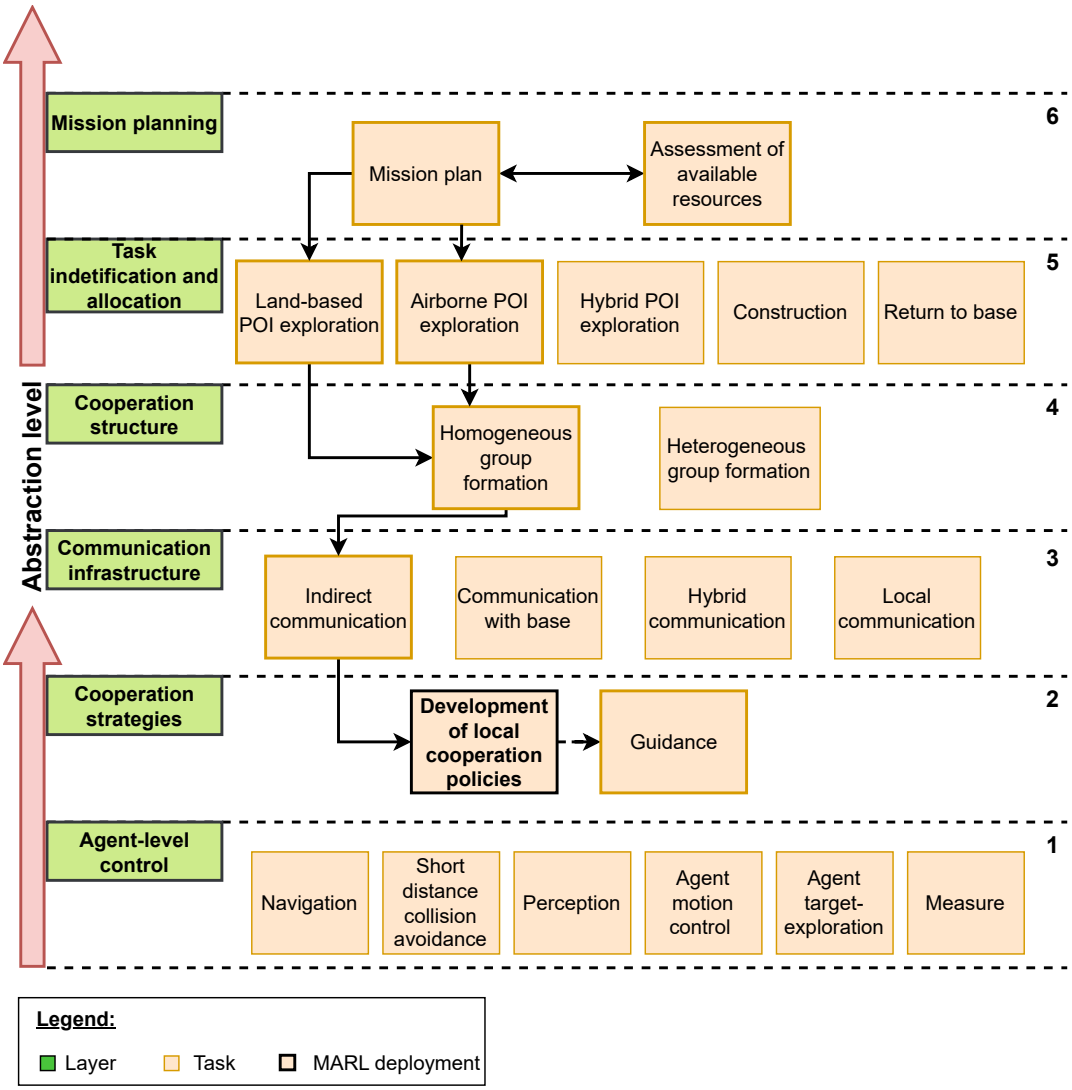


Fig. 1 Proposed MARL traceability model within NASA's MaCMAS methodology [17].

B. MARL Optimisation Problem

The collaborative planetary exploration problem is phrased as a study of decentralized partially observable Markov decision processes (DEC-POMDP)[7] with reward functions shared across all agents (which can depend on global or local performance, or a combination of both).
Furthermore, this study deals with homogeneous agent populations (see Figure 1, abstraction level 4) where agents have similar observation and action spaces, and act according to the same policy π. We do not make restrictive assumptions for homogeneous agent populations, hence allowing for future extensions with heterogeneous agent swarms.

In this problem formulation, denoted by $\langle S, A, O, R, P, n, \gamma \rangle$, the state space S encapsulates the possible states of the planetary environment. A is the shared action space for each agent i. Each agent's policy local observation o_i is represented as $o(s; i)$, where s is the global environment state. The shared reward function can take the form $R(s, A)$, $R(s_i, A)$, or $R(s_i, s, a_i, A)$; accounting for local or global performance, or a combination of both.
The transition dynamics of the environment are captured by $P(s'|s, A)$, denoting the probability of transitioning from state s to s' given the joint action $A = (a_1, \ldots, a_n)$. The discount factor γ weighs future rewards against immediate rewards. Agents deploy policies $\pi_\theta(a_i|o_i)$, parameterized by θ, to generate actions a_i based on their local observations.

The fundamental objective from the MARL pespective is to jointly optimise the discounted accumulated reward $J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T}\sum_{i=1}^{n}\gamma^t R_t^i\right]$.

IV. Exploration Environment

In this work, a physically simulated two-dimensional exploration environment in continuous space and discrete time is used. This environment consists of N agents, M target locations (which simulate strategic locations that need to be explored during the mission), and Z obstacles that should be avoided. All agents, targets and obstacles inhabit a physical location in space and possess physical characteristics such as size. Once a target is explored it disappears from the environment. Also, agents take actions **simultaneously**.

Specifically, agents take guidance actions; stating the desired $\Delta x, \Delta y$ (change in horizontal and vertical position) to which to move next, but are also affected by interactions with the environment which, if they occur, result in a perturbed $\Delta x, \Delta y \rightarrow 0, 0$ (if the agents are taking an action which will move them within the bounds of an obstacle, their change in position is overwritten to not change, hence preventing agents from going through obstacles). The maximum allowed change in horizontal and vertical position is set to two distance units, and the action space consists of continuous actions. The transition dynamics for an agent i are shown below:

$$s_t^{(i)} = \begin{bmatrix} x \\ y \end{bmatrix}_t^{(i)} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \end{bmatrix}_{t-1}^{(i)} \text{ or } \begin{bmatrix} x \\ y \end{bmatrix}_{t-1}^{(i)} \text{ if an obstacle collision occurs} \tag{2}$$

In addition, each agent receives a reward signal which can solely be based on local or global performance, or a combination of both. Agents are rewarded when a target is found, and penalised for colliding with other agents or obstacles. Notice that unlike other works in MARL [9], we **do not** make the reward function dense by quantifying the norm distance to the nearest target, or similar heuristics. Although using such heuristics can enhance the learning convergence, it hinders the finding of policies that might outperform human-based knowledge, which is one of the final goals of using MARL for policy discovery.

The environment is terminated when all the targets are found, or a maximum number of timesteps is reached. Lastly, we use environments that have noticeably more complexity than the state-of-the-art in MARL planetary exploration [9] regarding the amount and placement of targets, agents, obstacles, and the nature of the reward functions. Figure 2 shows such an environment.

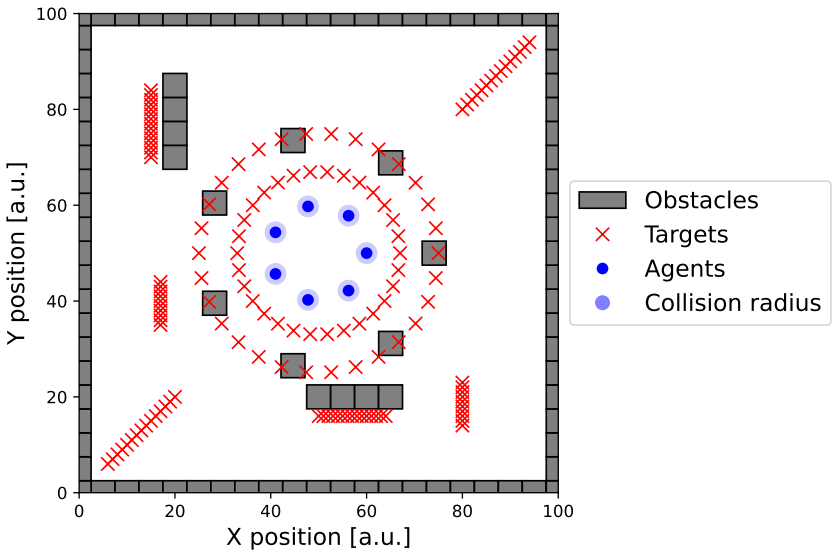


Fig. 2 Example exploration environment with seven agents (the circles), targets (the x's), and obstacles (the rectangles). Notice that targets are sometimes placed on top of obstacles, and hence can not be reached by the agents.

V. Learning Algorithm

To solve the distributed swarm control problem, a multi-agent extension of the PPO algorithm is proposed (see Algorithm 1) which is a variant of the algorithm used in [11], tailored for the collaborative exploration environment discussed in section IV, and including features to mitigate the MARL challenges listed in subsection II.C; discussed in this section and section VI.

A schematic of the learning process is shown in Figure 3. In this context, the agents in the swarm interact with the environment and collect experiences. These experiences are then used to learn a policy π_θ and a value function $V_\phi(s)$; two separate neural networks model these functions. These neural networks are shared across agents here since homogeneous swarms are studied, and one single buffer collects the experiences of all agents each time-step, effectively using **experience-based parallelization**. This technique enhances the sample efficiency of the algorithm as well as its performance [11]. However, the algorithm can be extended to heterogeneous agent settings by rolling different data buffers D, actor and critic networks per agent type, and generating agent-type-specific objectives and loss functions. Furthermore, the value function is only used during training (that is, once training is done, only the policy neural network needs to be implanted in the real robots) and is deployed for variance reduction.

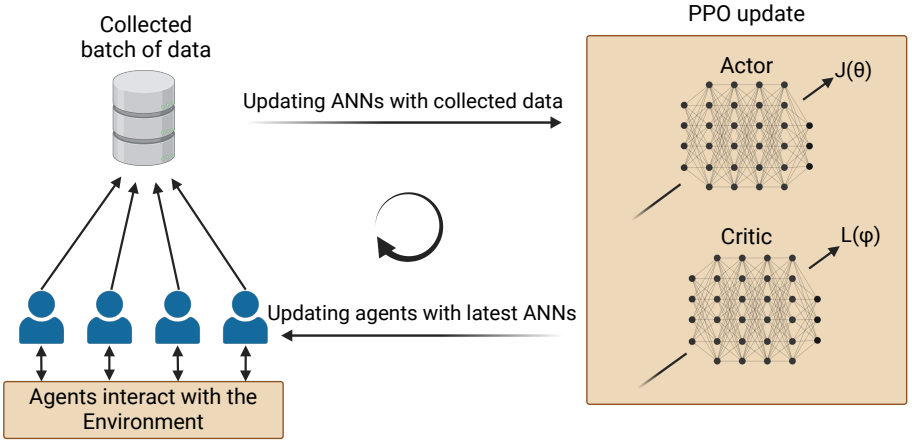


Fig. 3 Schematic of the multi-agent PPO learning process.

As shown in Algorithm 1, the objective of the policy is to maximize the objective function $J(\theta)$, also shown in Equation 3:

$$J(\theta) = \frac{1}{|D_k|T_\tau} \sum_{\tau \in D_k} \sum_{t=0}^{T_\tau} \min\left(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)} \hat{A}^{\pi_{\theta_k}}(O_t, a_t), g(\epsilon, \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}) \hat{A}^{\pi_{\theta_k}}(O_t, a_t) \right) \tag{3}$$

In particular, the first term inside the `min` function operates on the ratio between the probabilities of taking an action under the new policy $\pi_\theta(a_t|o_t)$ and the old policy $\pi_{\theta_{old}}(a_t|o_t)$, and scaling by the advantage estimate $\hat{A}^{\pi_{\theta_k}}(O_t, a_t)$ (which measures the advantage of taking a certain action compared to the average action, and is calculated using the critic). The second term inside the `min` function is the clip function g, which saturates the scaling of the policy ratio by a factor $1 \pm \epsilon$.

This way, the `min` function computes the minimum of two values: the policy-scaled advantage estimate and the clipped version. If the behavior of the new policy is very different from the previous policy, the `min` function selects the clipped term, thus not promoting such an abrupt policy change.

By imposing this constraint, the PPO algorithm balances the need for policy improvement with training stability, preventing overly aggressive policy updates that could lead to divergence or instability while still allowing for performance improvements.

Furthermore, in this work the policy neural networks output deterministic actions (during deployment, the policies don't describe probability distributions). This is done to find a deterministic policy that can potentially be certified in future planetary applications. However, to make the algorithm explore during training, the outputs of the policy parameterize the mean vector of a multivariate Gaussian distribution with a predefined covariance matrix from which to

sample actions. This exploratory noise is denoted as $\mathcal{N}$ in Algorithm 1. Notice that exploration techniques such as using entropy in the loss function are thus not implemented in this scenario.

Similar to the policy function, the goal of the value function is to minimise the loss function $L(\phi)$ (see Equation 4). In this sense, a converged critic will have a good estimation of the value of being in a certain state, given the current policy. In multi-agent settings, this is a complex task, given that the environment is non-stationary, and having an accurate expectation of discounted rewards is thus difficult. Selecting an optimal observation of the state for the value function is crucial for obtaining robust learning, and section VI delves into the proposed implementation of the paradigm of Centralized Training with Decentralized Execution.

$$L(\phi) = \frac{1}{|D_k|T_\tau} \sum_{\tau \in D_k} \sum_{t=0}^{T_\tau} \left(V_\phi(O_t) - \hat{R}_t\right)^2 \tag{4}$$

Moreover, the neural networks used for the actor and critic (a feed-forward and CNN network, respectively) are updated using the Adam optimiser, since it is suitable for objectives that are non-stationary and for problems characterized by highly noisy and/or sparse gradients [18].
For reproducibility purposes, several clarifications can be made in Algorithm 1:

- All the hyperparameters (including the ones not listed here), can be found in Table 1.
- The batch size is defined to contain a maximum certain amount of experiences collected from the environment. The experiences τ collected by the agents are stored in the data buffer D_k. Once filled, D_k is used to update the actor and critic networks. After this, a new, empty buffer is initialised for the next k iteration.
- In $J(\theta)$ and $L(\phi)$ (the objective and loss of the actor and critic, respectively) the factor $|D_k|T_\tau$ refers to the size of D_k multiplied by the amount of maximum steps collected in a given environment episode. Since the environment can have a variable maximum number of steps T depending on the performance of the agents, this variability is indicated with T_τ.
- In this work, the advantage is calculated first with $\hat{A}_t = \hat{R}_t - V_\phi(O_t)$, and then z-score normalised with $\hat{A} \leftarrow \frac{\hat{A} - \mu_{\hat{A}}}{\sigma_{\hat{A}}}$, where the statistical properties of the advantages are calculated using the batch.
- In $J(\theta)$, g is the clip function.
- The perception (observations) of the actor and critic networks are differentiated with o and O. In the IPPO configuration, $O = o$.

Algorithm 1 MAPPO algorithm for swarm exploration.

Initialise policy parameters θ, and initial value function parameters ϕ ▷ Initialise actor and critic
Set learning rate α, entropy coefficient β, discount factor γ, ▷ Initialise hyperparameters
updates per batch n, etc
while $k < k_{max}$ **do** ▷ Train for a set amount of steps
    Set data buffer $D_k = \{\}$ ▷ Batch of all agents data
    while $t <$ timesteps per batch **do** ▷ Collect environment experiences
        while environment *not done* **do**
            for all agents a **do**
                Collect actor and critic observations (o_t^a, O_t^a), actions $a_t^a = \pi_{\theta_k}(o_t^a) + \mathcal{N}_t$,
                action log probabilities $\log \pi_{\theta_k}(a_t^a|o_t^a)$, and reward R_t^a
                $\tau_t^a = [o_t^a, O_t^a, a_t^a, \log \pi_{\theta_k}(a_t^a|o_t^a), R_t^a]$
                $D_k += \tau_t^a$ ▷ Store experience in data buffer
            end for
            Act on the environment
        end while
    end while
    Compute rewards-to-go $\hat{R}_t$ on all experiences in D_k
    Compute z-score normalised advantage estimates, $\hat{A}_t$ (using any method of
    advantage estimation) based on the current value function V_{ϕ_k}
    Calculate actor PPO-clip objective:

$$J(\theta) = \frac{1}{|D_k|T_\tau} \sum_{\tau \in D_k} \sum_{t=0}^{T_\tau} \min\left(\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)} \hat{A}^{\pi_{\theta_k}}(O_t, a_t), g(\epsilon, \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}) \hat{A}^{\pi_{\theta_k}}(O_t, a_t) \right) \tag{5}$$

    Calculate critic loss:

$$L(\phi) = \frac{1}{|D_k|T_\tau} \sum_{\tau \in D_k} \sum_{t=0}^{T_\tau} \left(V_\phi(O_t) - \hat{R}_t \right)^2 \tag{6}$$

    for n steps **do** ▷ Update actor and critic networks
        Adam update θ on $J(\theta)$ with data D_k
        Adam update ϕ on $L(\phi)$ with data D_k
    end for
end while

VI. Perception Models

The selection of the states perceived by the policy and value functions is of paramount importance, as they must allow solving the DEC-POMDP, and also apply to a future potential deployment in an exploratory robot (this last requirement is specific to the policy). In this manner, both the policy and value (actor and critic) networks must have sufficient information to learn successful control strategies and value functions.
It should be noted that in this study, when CTDE is used, the policy and value functions have access to different environment information, in other words, **they have different perceptions**; the critic receives some form of global information, whereas the actor is limited to perceive local information. This work uses the same nomenclature as [11]: the CTDE configuration for the multi-agent PPO is referred to as MAPPO (Multi-agent PPO), whereas not having CTDE (the critic perceives the same environment information as the actor) is referred to as IPPO (Individual PPO).

In both IPPO and MAPPO, the actor uses a LIDAR-like perception model (the critic only uses this configuration in IPPO). A schematic of the LIDAR model is shown in Figure 4. To simulate the LIDAR, beams evenly "irradiate" from the agent, reaching a defined perception radius. Each beam contains three channels of information (corresponding to identifying agents, targets, and obstacles), detecting the position of the **closest** agent, target, and obstacle, or having as output the perception radius when nothing is detected in the beam. For the ANNs, these distance measurements are

normalised. A feed-forward neural network architecture models the actor policy; mapping from the LIDAR perception input to the guidance actions.

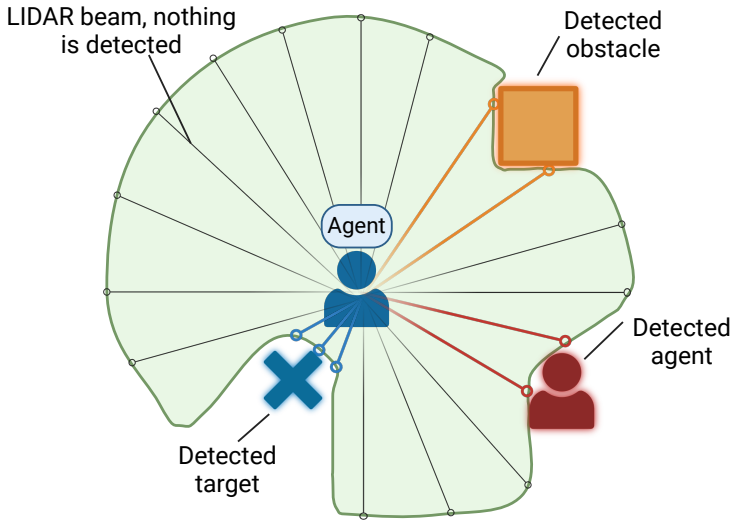


Fig. 4 Schematic of the LIDAR perception model used by the actor in both IPPO and MAPPO, and the critic in IPPO configuration.

In MAPPO configuration, the critic perceives a global environment tensor that also includes agent-specific information (see Figure 5). This tensor has three entries; entry i corresponds to the feature being extracted from the environment, and entries j and k to the horizontal and vertical coordinates of that feature space. In particular, four features are extracted from the environment: the position of all agents, targets, obstacles, and the self-position of the agent.
This way, the environment is discretized in a $M \times N$ grid where, for each environment feature, the grid cells in the tensor have a value of one if that feature is present, and zero otherwise. To process this input and model the critic, a deep convolutional neural network (CNN) is used, which estimates the value of the state defined by the input tensor.

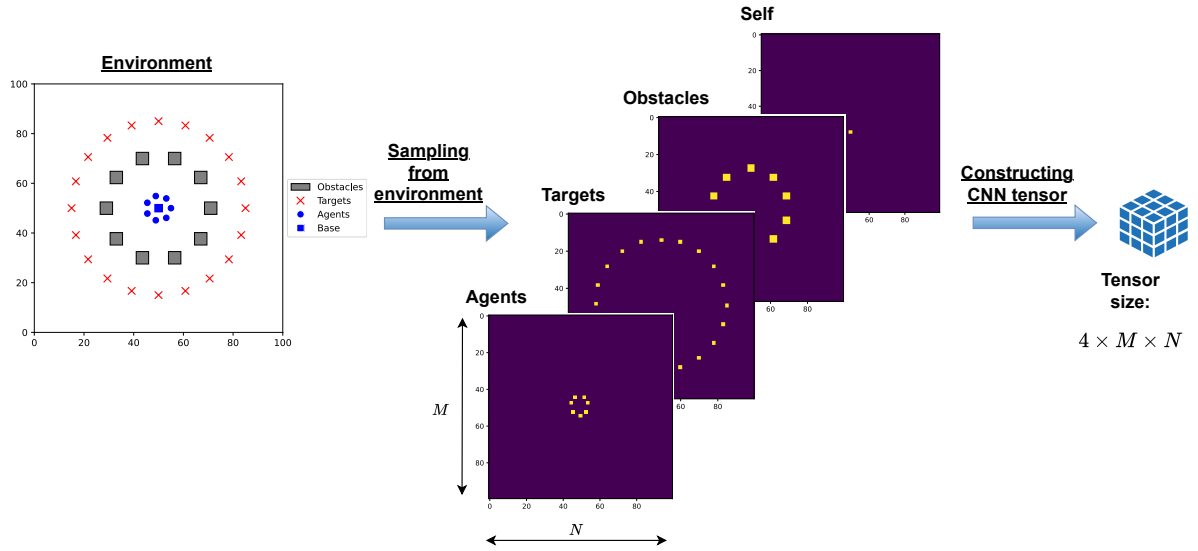


Fig. 5 Schematic of the generation of the MAPPO critic CNN perception tensor from the environment information. The tensor contains both global and agent-specific information (the self-position).

VII. Results and Discussion

This section presents the results of the tests carried out to investigate the performance of the proposed algorithm, both during training and execution. In particular, the experiments allow a comparison between IPPO and MAPPO in the following configurations:

- **Local, mixed, and global rewards**. This tests the extent to which IPPO and MAPPO can solve such learning problems and the consequences of using these types of reward functions in MARL.
- **Always training in the same environment versus changing the environment every learning episode**. This is done to test whether there is an improvement in the quality and robustness of the policy when training with rich environments, as suggested in [19], which could potentially help mitigate the adverse effects of using a sparse reward function, and might allow future development of reward functions that follow directly from the mission requirements, without the need to include lower level heuristics to make the reward functions dense.
- **3 and 7 agents environments using global reward functions**. This tests the scalability properties of the algorithms while learning.
- **Agent scalability of the learned policies**, comparing models trained with different reward and environment settings. This tests the generalisation capabilities of the learned policies.

All of these tests were performed using NVIDIA V100 GPUs, using the hyperparameters shown in Table 1. These parameters are obtained from [11] or else are experimentally found to be effective.
Moreover, these tests are computationally expensive, and due to computing resource constraints, one learning run has been performed per comparison setting. Hence, this work cannot fully address the statistical significance of the individual learning runs.
Additionally, other works in challenging MARL scenarios complete learning runs that are as much as **three thousand** times longer (episode-wise) [14]. The learning of complex emergent behaviors thus takes computational efforts beyond the resources available to produce these results, and although emergent behaviors are achieved and analysed, it is possible that more complex ones can emerge when training for longer.

Table 1 Default hyperparameters.

Parameter	Value	Parameter	Value
Action noise σ^2	0.5 [a.u.]^2	Clip value	0.2
Collision Radius	2.1 [a.u.]	CNN pixel spatial resolution	100×100
Discount Factor (γ)	0.99	LIDAR perception radius	20 [a.u.]
Learning Rate (α)	0.005	Map Size	100×100 [a.u.]
Max allowed action	± 2 [a.u.]	Max Gradient Norm	10.0
Max Time-steps per Episode	100	Number of Agents	7
Number of LIDAR beams	36	Number of Obstacles	15 (excluding map boundaries)
Number of Targets	140	Obstacle Height	5.0 [a.u.]
Obstacle Width	5.0 [a.u.]	Agent Spawn Radius (from map's centre)	10.0 [a.u.]
Time-steps per Batch	4800	Updates per batch (n)	5

A. Reward Function Types

In this experiment, three types of reward functions are compared; local, mixed, and global. This is shown in Equation 7. Here, R_i refers to the reward received by agent i, C_i is the collision status of the agent, and T refers to whether an agent has found a target in the current step. j refers to a specific agent within a swarm containing N agents. Hence, the local reward function is solely dependent on the individual performance of agent i. The mixed reward function is similar but also rewards when the swarm finds targets, although with lower relevance. Lastly, the global reward function sanctions agent-specific collisions and rewards the finding of targets by the overall swarm.
Moreover, different scaling for collision penalties were studied, however, the weighting of the collision penalty is smaller than the weight of finding a new target and is always linked to individual performance since it was found that otherwise, the agents fall in the local minima of not moving.

Importantly, these reward functions have different scaling (i.e. a local reward of 100 does not correspond with a mixed or global reward of the same value). This way, for a similar swarm performance, the local reward outputs the smallest value, followed by the mixed reward, and the global reward is the largest. This is caused by the summation terms in the mixed and global rewards. Hence, the learning curves of different reward functions can not be compared in terms of performance (this is done in subsection VII.D via policy analysis).

$$\begin{array}{lll} \textbf{Local Reward} & \textbf{Mixed Reward} & \textbf{Global Reward} \\ R_i = -\dfrac{C_i}{7} + T_i & R_i = -\dfrac{C_i}{7} + T_i + 0.2 \cdot \sum_{j=1}^{N} T_j & R_i = -\dfrac{C_i}{7} + \sum_{j=1}^{N} T_j \end{array} \tag{7}$$

The learning progress of IPPO and MAPPO is shown in Figure 6. Several observations can be made. Initially, all algorithms experience a rapid improvement in the rewards obtained, and the learning performance flattens afterward. This behavior agrees with similar results from other works [11].
Secondly, the training performance of IPPO and MAPPO is comparable in the local and mixed settings, while MAPPO doubles the performance of IPPO in global reward settings.
With local rewards, IPPO achieves better performance (close to episode $0.4 \cdot 10^5$), although it is possible that with longer run times or different initialisations, MAPPO reaches a similar performance.

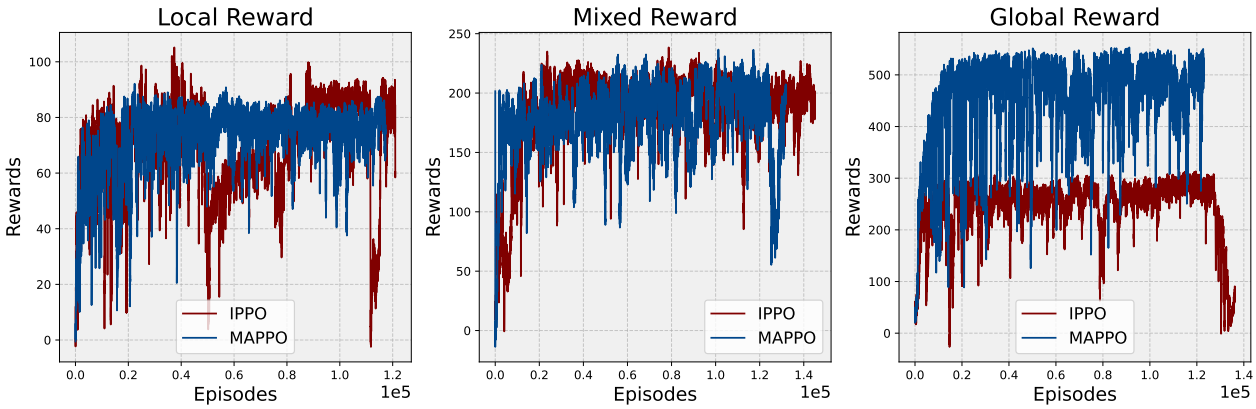


Fig. 6 Learning performance of the IPPO and MAPPO models with the respective type of reward function.

Moreover, when looking at the training stability, the variances of the learning curves are shown in Figure 7. Here, it is observed that for local rewards, the variance of MAPPO is always similar or significantly smaller than IPPO's. With mixed rewards, IPPO and MAPPO have comparable variances, with IPPO achieving marginal higher learning stability. Then, MAPPO has significantly higher variances than IPPO with global rewards (also occurring by the fact that it is achieving higher rewards), although after episode $1.2 \cdot 10^5$, IPPO has a catastrophic training instability; decreasing the obtained reward to be close to zero.
The exponential variance reduction behavior shown in all scenarios is attributed to taking the variance of a non-stationary value, reflecting the transition from rapid to slower reward increases.

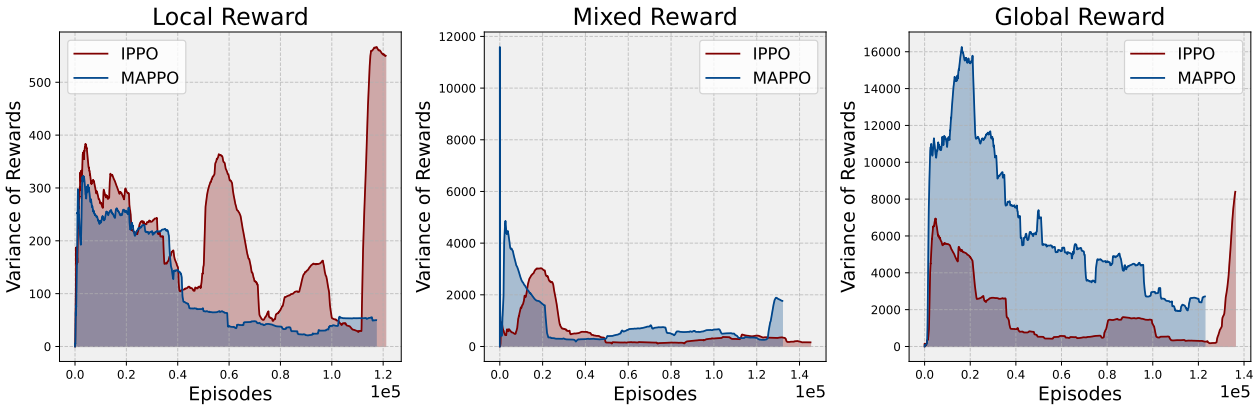


Fig. 7 Variance of the rewards obtained during training. The variance is calculated using a sliding window of 3,000 episodes.

From these results, it is observed that MAPPO and IPPO have similar learning performance when the rewards are not based (or weakly so) on global swarm performance and instead focus more on agent-specific metrics. Noticeably, MAPPO doubles the amount of rewards obtained by IPPO when this is not the case and the rewards are global.

This attests to the importance of the state representation perceived by the actor and critic networks. Compared to the CNN tensor, the LIDAR perceives richer information near the agent, detecting nearby environment features accurately, unlike the CNN tensor, which is limited in resolution by the selected pixel size. Thus, when considering IPPO with local or mixed rewards, the critic perceives information that is highly relevant for estimating the current value, whereas the CNN network is limited in local perception, although it has access to global information that can be relevant during further training progress (not considered here) where complex egocentric behaviors can emerge to increase the local reward (such as strategically stealing targets that could be captured first by other agents) .

However, when the rewards become global, the learning behavior of IPPO becomes critically hindered, unlike MAPPO's, which can still achieve good learning performance. In this scenario, knowing global environment information plays a crucial role in estimating the value function, since agents need to identify that they can receive rewards based on the behavior of other agents, and not just them, and crucially, that through cooperative behaviors, their rewards can be maximised. On the contrary, when using a LIDAR-critic in this scenario (IPPO), the agent receives rewards according to information they can not perceive and is not stationary, thus making the learning problem extremely challenging. These findings agree with the theory behind CTDE, and the usage of these techniques in multi-agent settings [11, 14].

As a remark, the different reward functions change the nature of the problem. Local reward functions result in adversarial training, as the different agents in the swarm compete for a limited amount of targets. On the contrary, the global reward function fosters the development of cooperative policies, which is the end goal of this study. From this perspective, MAPPO is the preferred algorithm to deploy.

B. Rich Environment Training

To test whether training in random environments can enhance the performance of the learned policy (following the findings from [19]), a rich environment training is used to also compare against always training in the same environment.

In the rich environment, the targets and obstacles are generated randomly every time the environment is initialised (although some obstacles near the map's center are always generated in the same place), and the total amount of obstacles can change. Moreover, rich environments contain the same number of targets as those used in subsection VII.A. Lastly, although the agents are always initialized evenly spaced from the map's center, in a circular formation, the agents' initial position within the circle is generated randomly.

The three algorithms showing better performance in subsection VII.A are considered: IPPO with local rewards, and MAPPO using mixed and global rewards (in the mixed reward setting, IPPO and MAPPO displayed similar performance, but given its CTDE nature which should favor stronger results [11], MAPPO is selected). Figure 8 shows the learning progress of the three algorithms.

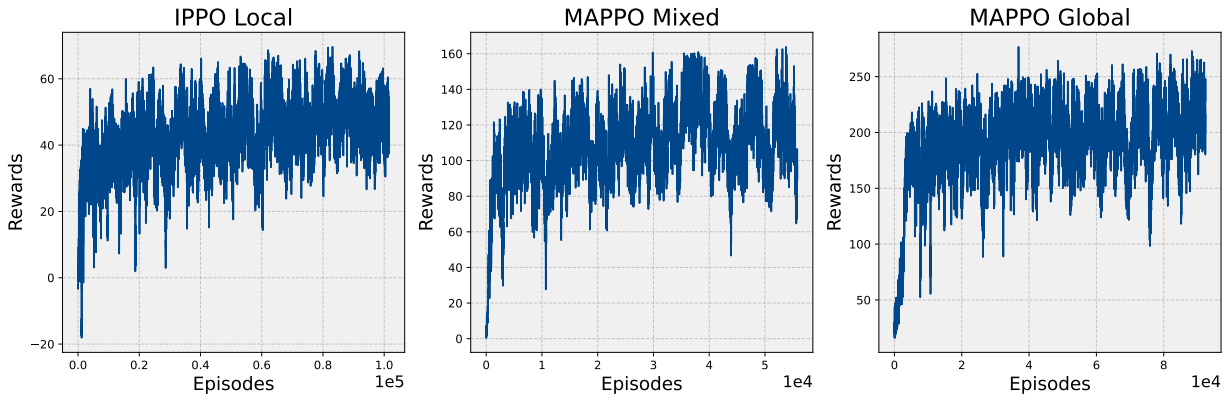


Fig. 8 Learning performance of the models in the rich environment. The models have been trained for different amounts of episodes due to constraints in computational resources.

Similar to the learning curves shown in subsection VII.A, the three algorithms experience a rapid increase of rewards at the beginning of training, slowly improving during the rest of the run.

Moreover, all three algorithms achieve lower rewards than the ones obtained always training in the same environment (see Figure 6 for a comparison). Although this difference in performance might also be caused by the shorter training times (the MAPPO algorithms are roughly trained for half the time of those shown in Figure 6, due to computational resources constraints), this reduction in performance might further be accentuated by the learning problem becoming more difficult, as over-fitting to explore targets or avoid obstacles that are always expected to be located in the same place is not possible. It is hypothesized that agreeing with the findings from [19], with longer learning times the algorithm can potentially find more generalizable policies leading to higher rewards, as the rich environment has fewer properties that can be exploited with simple heuristics.
However, as shown in subsubsection VII.D.2, the learned policies exhibit different behaviors from the ones obtained learning always in the same environment, and generalize better (see subsection VII.E).

C. Training With a Different Amount of Agents

Training with three agents has been tested to compare the agent scalability of the learning algorithms. Global rewards are used to ensure a fully cooperative setting, and both IPPO and MAPPO are examined. Furthermore, the training environment is identical to the one used with seven agents in subsection VII.A.
Figure 9 shows the learning progress of both IPPO and MAPPO. Here, it is observed that both algorithms gradually improve the obtained return, with IPPO achieving higher average rewards. This difference in ultimate reward is caused by the IPPO agents finding targets in the top left of the map, something the MAPPO agents fail to do (see subsubsection VII.D.3). It is hypothesized that with different training initializations, MAPPO could achieve similar average performance.

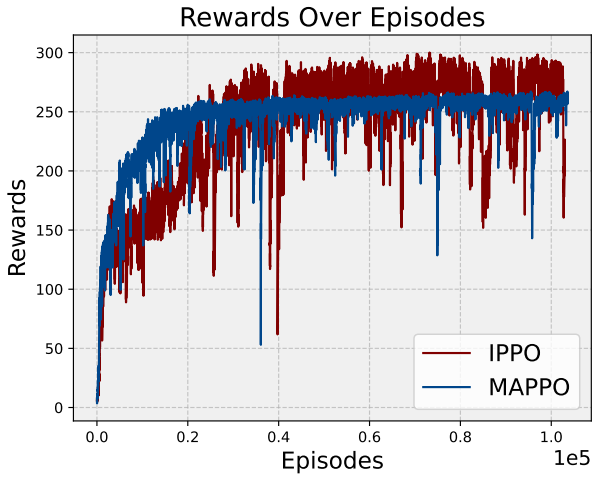


Fig. 9 Learning performance of IPPO and MAPPO in an environment with three agents, and global rewards.

A noticeable contrast lies in the training stability. The MAPPO training curve has a variance that is between two and three orders of magnitude smaller than IPPO's, as shown in Figure 10 (the initial high variances of both algorithms are caused by the rapid increase in rewards obtained at the beginning of the training phase, and due to its non-stationary behavior, the variances are **not** used to assess learning stability in this region). This finding further reflects the importance of the input to the critic in multi-agent settings, and how using CTDE stabilises the learning run.

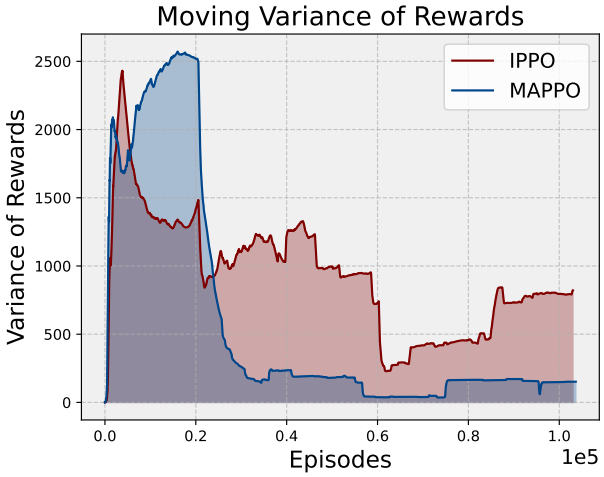


Fig. 10 Variances of the rewards obtained training with 3 agents (see Figure 9). The variance is calculated using a sliding window of 3,000 episodes.

Moreover, it is found that MAPPO has better agent scalability than IPPO when using global rewards. This is reflected by the fact that the MAPPO learning runs with seven agents achieve higher rewards than the ones with three agents (effectively doubling), while IPPO fails to improve (this is observed by comparing Figure 6 and Figure 9). Agreeing with the discussion in subsection VII.A, CTDE is thus likely to be an essential feature when developing MARL algorithms that can be used to learn policies in large swarms (when global reward functions are used).

Nevertheless, with both three and seven agents, the algorithms fall into suboptimal solutions (the maximum possible reward, related to finding all targets and not crashing, is never found). This suboptimal behavior is hypothesised to be caused by limited training episodes. However, agreeing with the theory in the field (see subsection II.C), this further

reflects that MARL algorithms are susceptible to falling into local minima that are difficult to get out of.

D. Policy Analysis

The behaviors of the learned policies are now studied; examining the policies obtained in subsection VII.A, subsection VII.B, and subsection VII.C.

1. Reward Functions

Inspecting the behaviors of IPPO and MAPPO with local rewards in Figure 11, IPPO explores more targets, with the pink agent exploring the bottom left targets too, something that the MAPPO agents fail to achieve. This could be caused by a more unfavorable training initialisation, and longer/more training runs would be needed to assess this.
However, it is observed that the behaviors of the agents are very different. The IPPO agents move in more erratic patterns, often following similar trajectories and hence stealing targets from each other, with MAPPO agents moving circularly to collect the targets near the center of the map (although it can be observed that the red MAPO agent in Figure 11 gets stuck between obstacles).

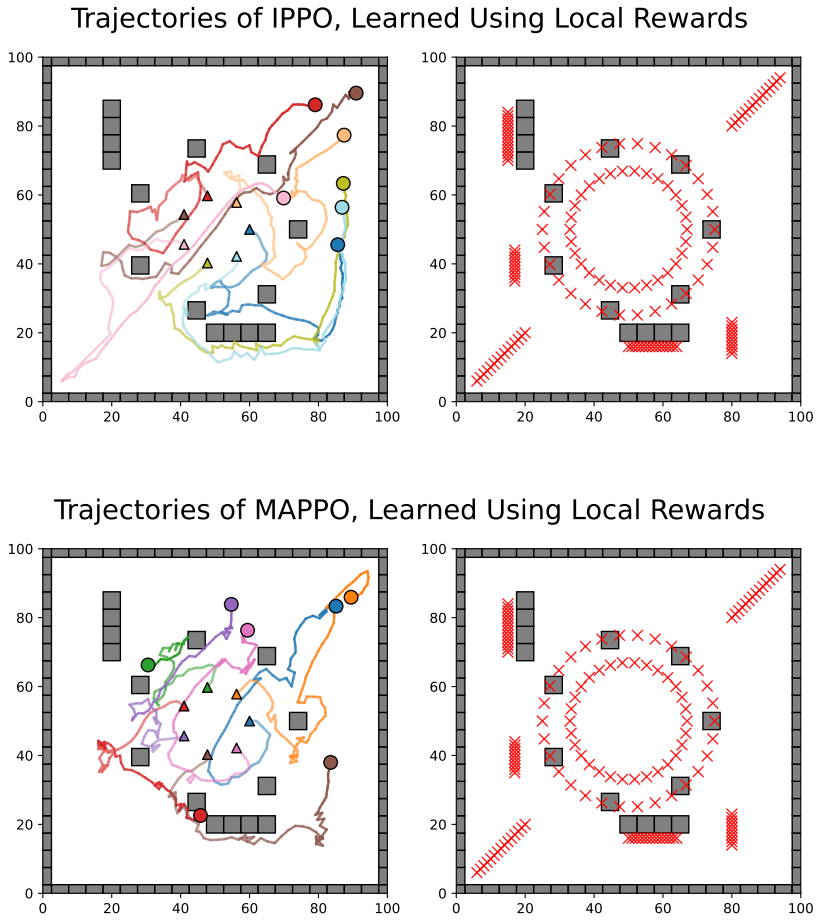


Fig. 11 IPPO (top) and MAPPO (bottom) policy behavior in the training environment, using local reward functions. See the Appendix for timesteps figures of the simulation. Left: Trajectory traces of the agents, the triangles indicate the starting position of the agents, and the circles, their final positions. Each color corresponds to a specific agent. Right: environment drawing, only containing the location of the obstacles and targets. Here IPPO and MAPPO explore 115 and 97 targets, respectively. The environment has a total of 141 targets.

When it comes to mixed rewards, similar behaviors are observed (see Figure 12). In this case, both algorithms achieve a similar exploration of the environment, with MAPPO having a more collaborative exploration of the inner circle of targets. Also, both algorithms explore the lower-left targets with the pink agent and have agents that crash against obstacles.

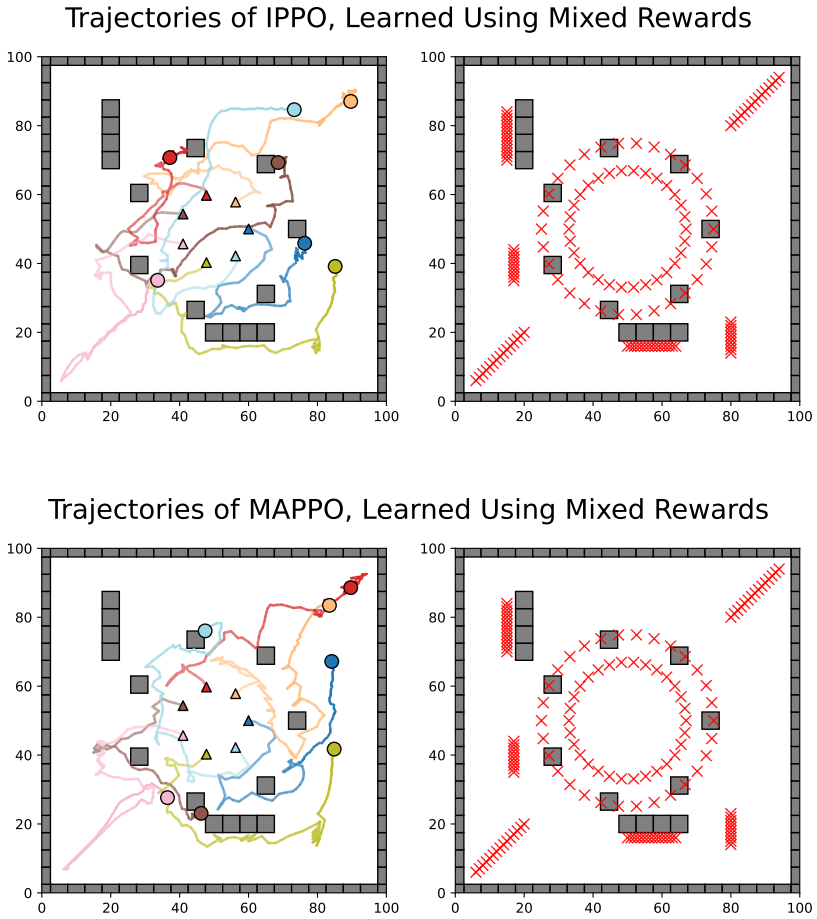


Fig. 12 IPPO (top) and MAPPO (bottom) policy behavior in the training environment, using mixed functions. Left: Trajectory traces of the agents, the triangles indicate the starting position of the agents, and the circles, their final positions. Each color corresponds to a specific agent. Right: environment drawing, only containing the location of the obstacles and targets. Here, IPPO and MAPPO explore 114 and 113 targets, respectively. The environment has a total of 141 targets.

Finally, Figure 13 shows the policy trajectories with global rewards. In both cases, the algorithms only find policies that explore a specific diagonal of the environment, with MAPPO having collaborative exploration of the majority of the inner targets' circle and better agent spread, also exploring the lower part of the map. The performance degradation of both algorithms when compared with the other reward settings is likely caused by the increased difficulty of using global rewards, and more training time might be needed to achieve performances similar to the local and mixed rewards counterparts.

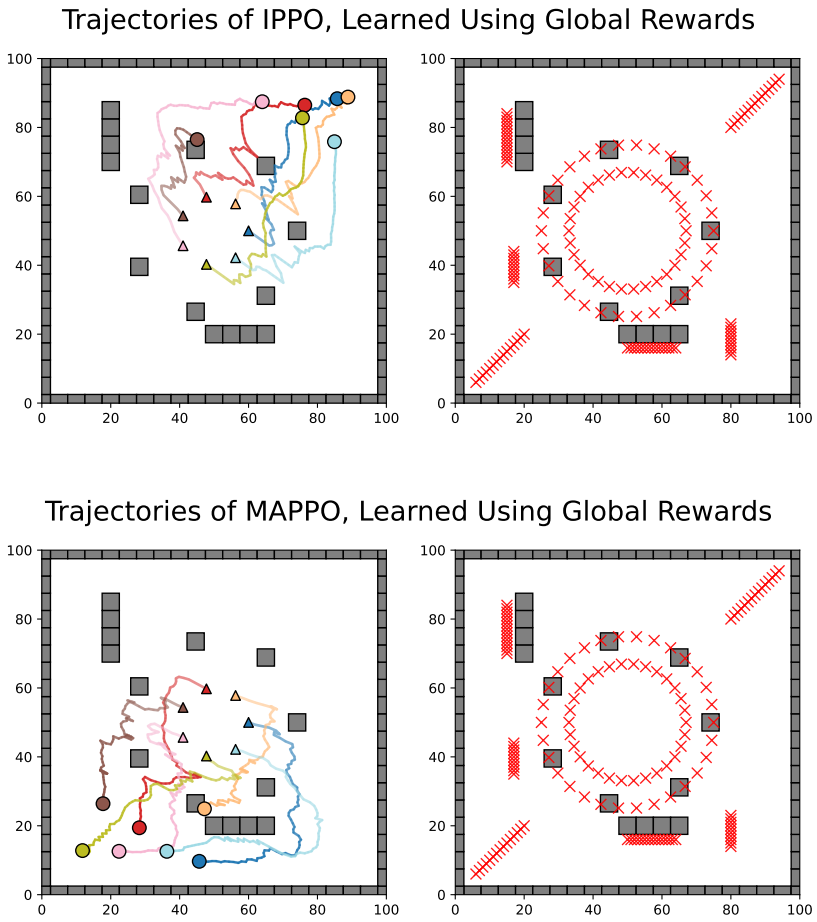


Fig. 13 IPPO (top) and MAPPO (bottom) policy behavior in the training environment, using global reward functions. Left: Trajectory traces of the agents, the triangles indicate the starting position of the agents, and the circles, their final positions. Each color corresponds to a specific agent. Right: environment drawing, only containing the location of the obstacles and targets. IPPO and MAPPO explore 47 and 82 targets, respectively. The environment has a total of 141 targets.

From these results, it can be observed that while there are no significant differences in performance between local and mixed rewards, the global reward scenario obtains policies that fail to explore as many targets. This performance difference is hypothesised to decrease with more training time, however, it highlights that for reduced training times, local or mixed rewards have the potential to learn higher performing policies, since the global rewards make the learning problem more complex.

2. Rich Environment

Inspecting the behaviors of IPPO and MAPPO learned in a rich environment (shown in Figure 14), it can be observed that the learned policies are very different from the ones obtained training in non-rich environments.

Firstly, both IPPO and MAPPO-mixed (using mixed rewards during training) agents learn a territorial behavior (see the top two plots of Figure 14), where they spread over the map and attempt to explore areas not covered by other agents, thus maximizing the probabilities of finding regions in the map with unexplored targets. This behavior might emerge because of the adversarial nature of the problem, imposed by the local and mixed reward functions. The policies also result in the agents sometimes colliding with obstacles. This could be caused by the limited training time and the scaling of the collision penalty; since the latter is much smaller than the one obtained when finding a target, it is plausible that the agents first prioritize the finding of targets, since this has a more salient effect on the reward function (for example, as shown in Equation 7, for the local reward function, finding a target has seven times more relevance than colliding).

Regarding MAPPO training with global rewards, the agents learn to collectively sweep the area (see Figure 14), keeping within a distance of each other, and exploring targets as they move towards the lower right part of the environment. This is a learned collaborative and cooperative behavior, where the swarm agents work together on the shared goal of collecting targets (established by the global reward function) and cooperating in agreeing to stay a "safe" distance from each other, each agent allowing others to explore targets if they are closer to them. A timestep plot of the sweeping behavior can be seen in the Appendix, Figure 21.

An important observation can be made from this experiment. Compared with the global reward, the local and mixed rewards achieve better exploration of the environment. This highlights the fact (shown analogously in the previous section) that even though MAPPO performs better than IPPO when using global rewards, depending on the learning problem, using IPPO or MAPPO with a different type of reward function can achieve better performance. The emergent patterns obtained using MAPPO with global rewards are more pronounced, but this does not necessarily lead to an increase in swarm performance, at least with this amount of training.

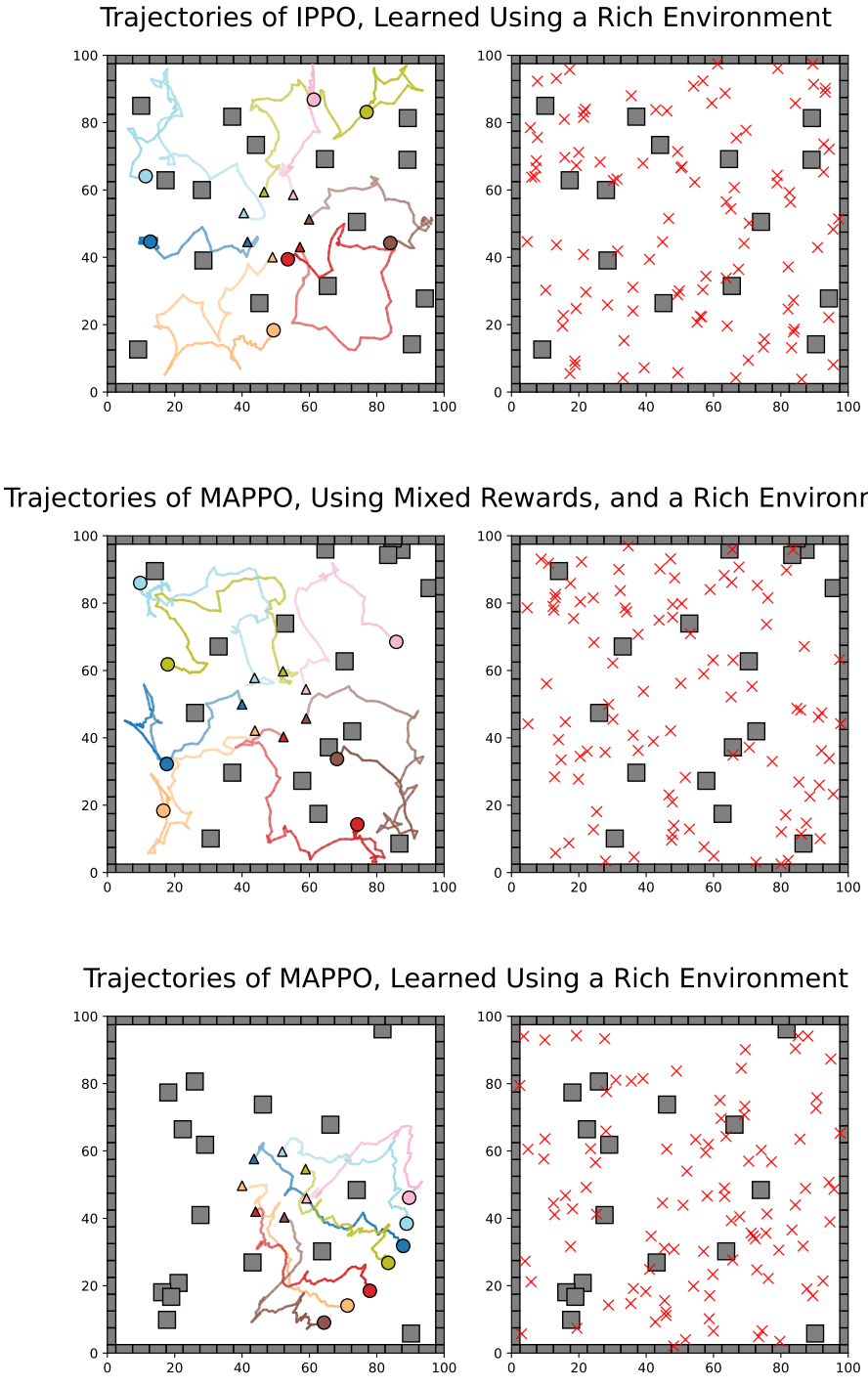


Fig. 14 IPPO-local (top), MAPPO-mixed (middle), and MAPPO-global (bottom) policy behavior in the rich training environment. Left: Trajectory traces of the agents, the triangles indicate the starting position of the agents, and the circles, their final positions. Each color corresponds to a specific agent. Right: environment drawing, only containing the location of the obstacles and targets. Both IPPO and MAPPO-mixed achieve a spreading behavior, while MAPPO agents learn to jointly sweep the map towards the bottom right part of it.

3. Training With 3 Agents

When training with 3 agents, two different patterns **emerge**, shown in Figure 15. The IPPO agents learn to explore the inner circle of targets collectively, each exploring one-third of it, and then proceed to explore targets in different areas of the environment, thus strategically not impeding each other.

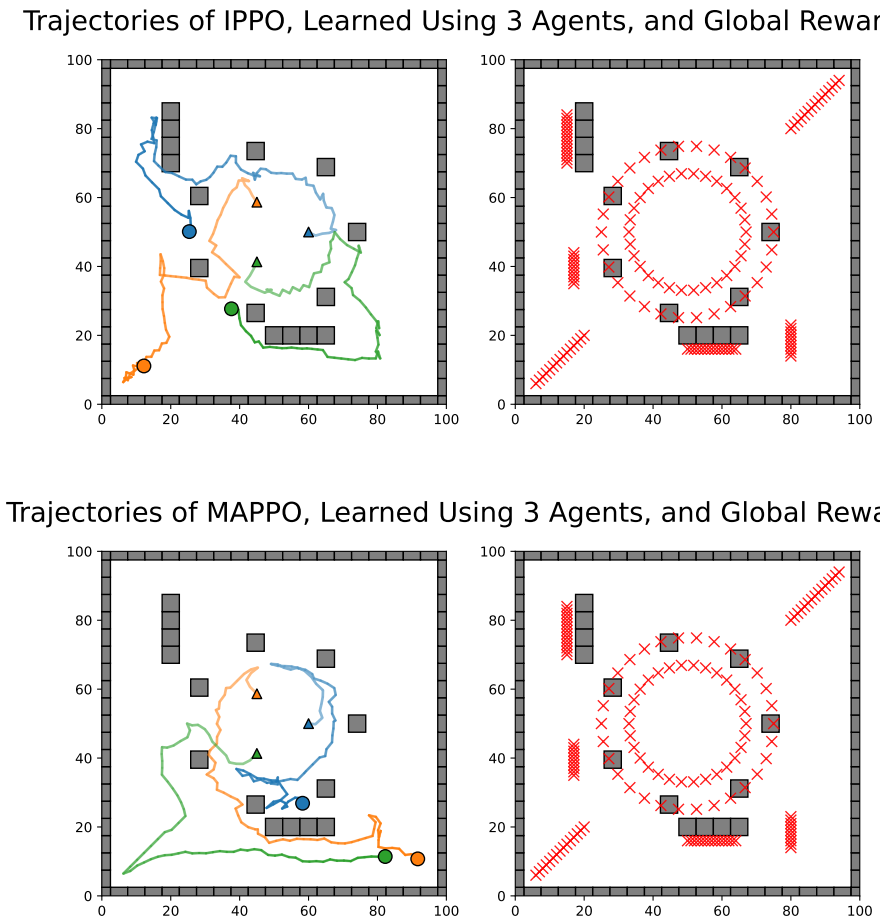


Fig. 15 IPPO (top) and MAPPO (bottom) policy behavior in the training environment, using global reward functions and 3 agents. Left: Trajectory traces of the agents, the triangles indicate the starting position of the agents, and the circles, their final positions. Each color corresponds to a specific agent. Right: environment drawing, only containing the location of the obstacles and targets. A timestep plot of the IPPO simulation can be seen in the Appendix, Figure 22. IPPO and MAPPO explore 107 and 91 targets, respectively. The environment has a total of 141 targets.

Moreover, the MAPPO agents learn a different exploration strategy. They also allocate tasks to different agents (see the lower plot in Figure 15), however, this is done differently. Inspecting the behavior of each MAPPO agent, shown in Figure 16, the blue agent gets assigned the exploration of the majority of targets in the inner circle, while the orange agent explores a small region of the inner circle and then focuses on exploring the lower part of the map, the green agent focusing on the cluster of targets in the lower-left part of the map.

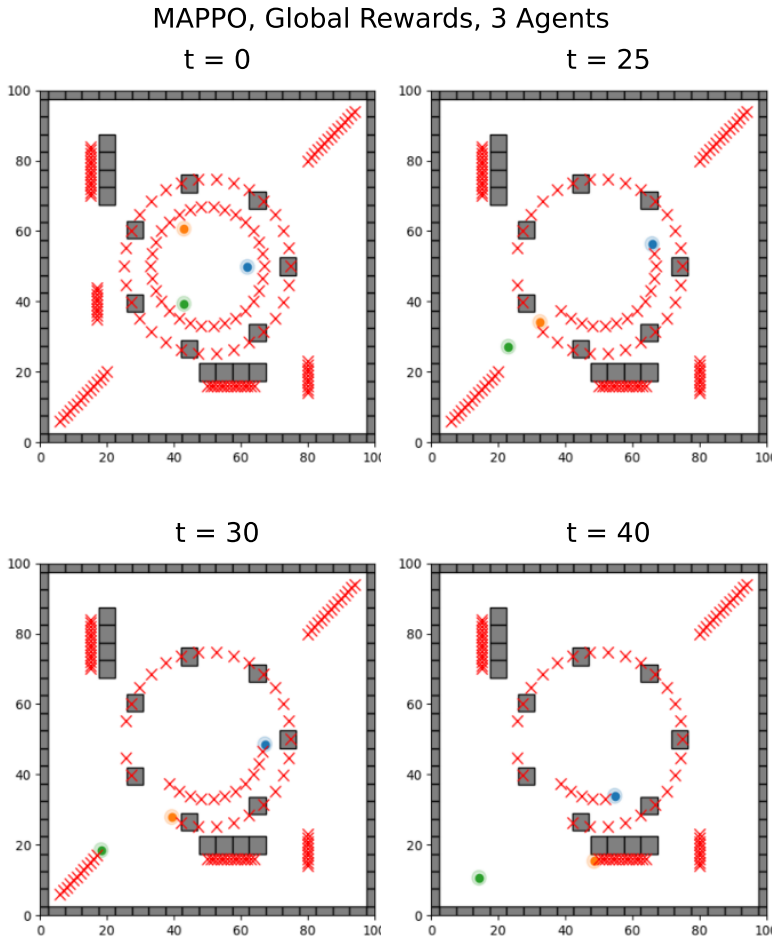


Fig. 16 Trajectory trace of the learned policy from MPPO, using local reward functions. The targets disappear from the environment as they are explored. It is observed that the agents autonomously distribute the task; the blue agent explores the inner target circle, the green one explores the targets in the bottom-left of the map, and the orange explores the targets in the lower part of the environment.

These policies show **cooperative motion planning strategies, and task recognition and allocation behaviors**. Compared with the previous environment configurations, the emerging strategies are clearly defined. This could be caused by the decrease in complexity of the learning problem, as it goes from 7 to 3 agents. It is thus possible that if the algorithms are allowed to train for longer times, such defined patterns can emerge for 7 or more agents too. Furthermore, both IPPO and MAPPO achieve these strategies. Regarding IPPO, there is a notorious performance gain when compared to when it learns using seven agents and global rewards (see Figure 6). This might occur because the variance in the rewards received reduces with fewer agents. Furthermore, in the discovered emergent strategy of collaborative inner circle exploration, all agents remain within LIDAR reach of each other, thus in this case, the critic of IPPO perceives the necessary information to determine the value of the current state and foster the emergence of the circular motion pattern.

E. Agent Scalability During Deployment

To further understand the applicability of the learned policy to swarm missions, the generalisation properties of the policy are studied. In particular, the policies of the algorithms trained in rich environments, and their non-rich environments counterparts, are studied.

In this experiment, the **learned** policies are deployed on swarms with different numbers of agents, to examine whether the behaviors learned training with seven agents can generalize to different swarm sizes. Each simulation run consists of 70 steps, and the environments generate 100 target positions. Moreover, three metrics are examined: the amount of explored targets, and the collisions with other agents and obstacles. Regarding the collision count, the total amount of collisions during the simulation is calculated by adding together all the individual collisions of the agents. In the case of the collisions between agents, when two agents collide, this is counted as a single collision. However, if an agent gets stuck in a crashing behavior against an obstacle or fellow agent, that collision is counted at each time step (for example, if an agent keeps colliding against the same obstacle during 100 time steps, that is counted as 100 collisions).

Lastly, the algorithms trained with a rich environment are referred to as "Rich Env".

Figure 17 shows the scalability properties of IPPO, trained with local rewards. Here it can be observed that as the swarm gets larger, more targets are explored. Also, IPPO Rich Env explores more targets than IPPO in all the different swarm sizes and has fewer agent-agent collisions. Regarding obstacle collisions, both algorithms have a similar performance; the number of collisions increases as the swarm gets larger. In the case of IPPO Rich Env, the number of explored targets starts to saturate after the swarm contains more than 11 agents because the environment has limited target resources: it generates 100 targets, some of which might not be reachable because they are generated on top of an obstacle, hence exploring close to 100 targets is the maximum possible performance in this regard.

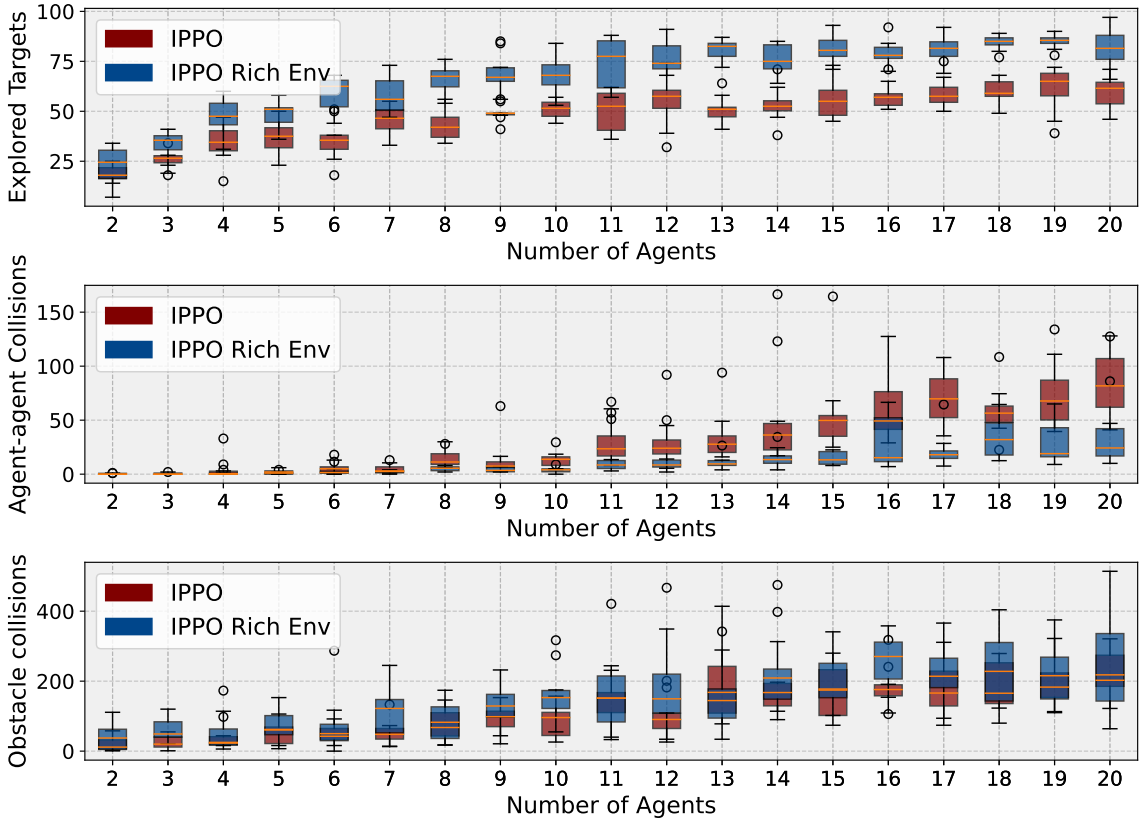


Fig. 17 Agent scalability plot of IPPO training with and without a rich environment. Local rewards are used during training. Lastly, the box plots are constructed using 10 runs per number of agents. The deployed policies are trained using 7 agents.

The scalability of MAPPO using mixed rewards is shown in Figure 18. Similar to the IPPO scenario, MAPPO Rich Env finds more targets than MAPPO for all numbers of agents and has fewer agent-agent collisions (although this last difference in performance is marginal). Moreover, both algorithms have a similar number of obstacle collisions, with MAPPO having marginally fewer collisions.

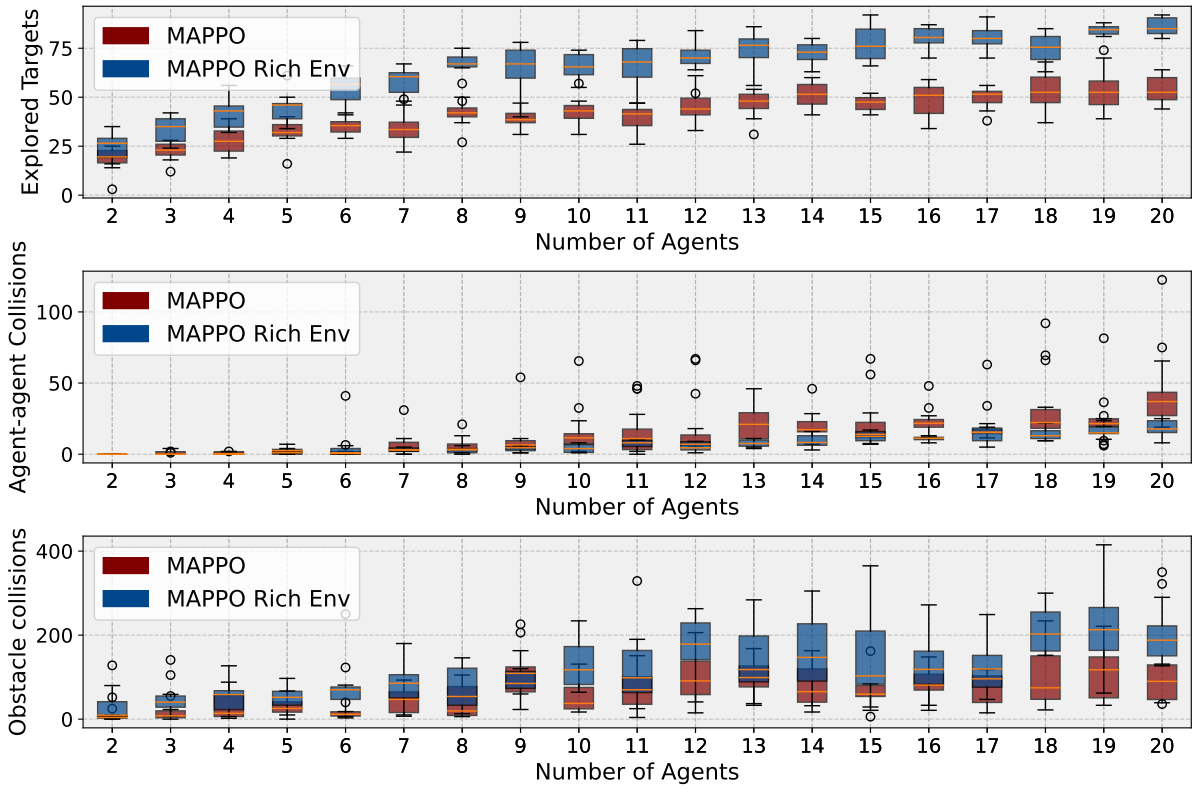


Fig. 18 Agent scalability plot of MAPPO training with and without a rich environment. Mixed rewards are used during training. Lastly, the box plots are constructed using 10 runs per number of agents. The deployed policies are trained using 7 agents.

When inspecting MAPPO with global rewards in Figure 19, both MAPPO and MAPPO Rich Env explore more targets as the swarm gets larger, but there is no significant performance difference between them. It should be noted that both algorithms have an asymptotic scalability behavior: exploring a maximum of 45 targets (approximately). Since the environments are generated with 100 targets, this behavior is not caused by exploring all available targets, but rather due to the policys' failure to generalize enough to use all the swarm agents effectively.
Regarding the number of agent-agent collisions, MAPPO Rich Env has fewer collisions, its collisions scaling approximately linearly with the number of agents, while MAPPO has a weak exponential behavior. Moreover, MAPPO has more obstacle collisions than MAPPO Rich Env.

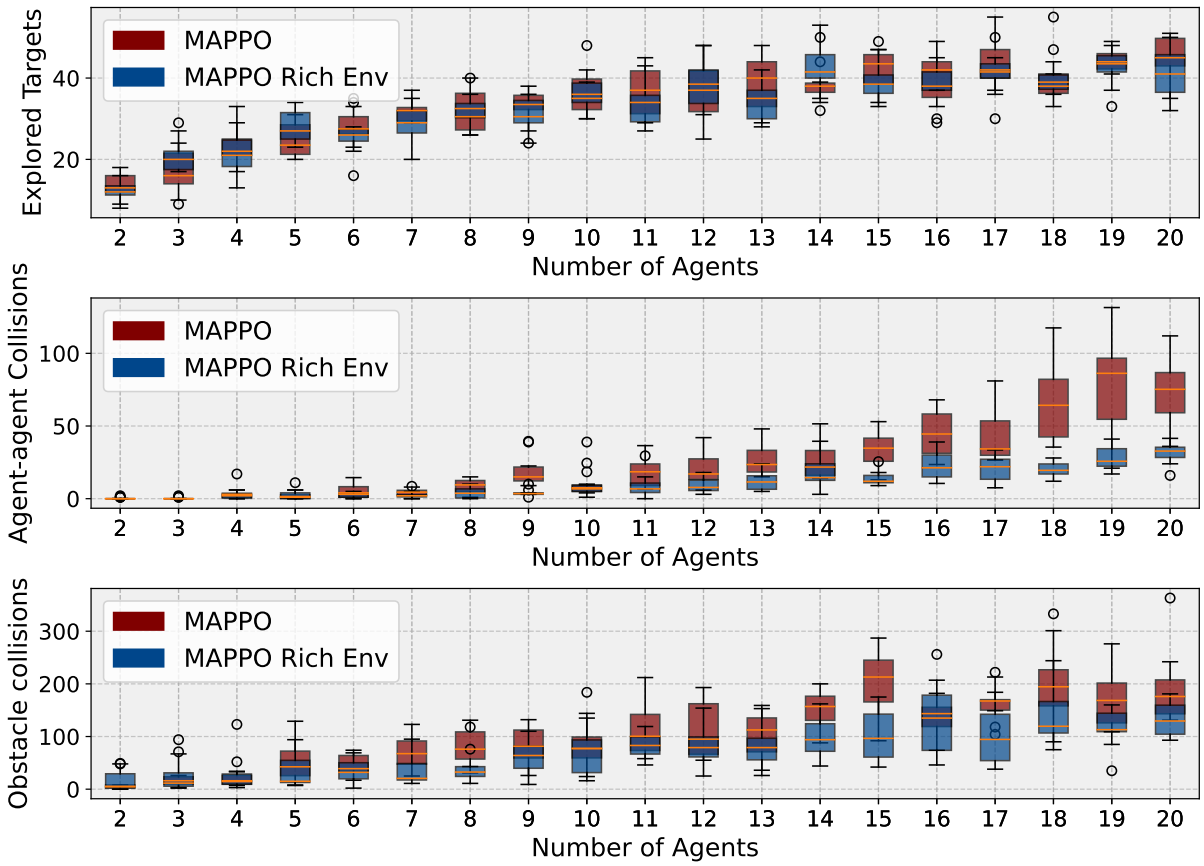


Fig. 19 Agent scalability plot of MAPPO training with and without a rich environment. Global rewards are used during training. Lastly, the box plots are constructed using 10 runs per number of agents. The deployed policies are trained using 7 agents.

Lastly, when comparing across types of reward functions (that is, Figures 17, 18, and 19), several observations can be made:

- All algorithms find more targets as the swarm gets larger. However, while IPPO and MAPPO-mixed (and their Rich Env versions) effectively explore the total amount of targets in the map, MAPPO-global fails to do so, having close to a 50% performance degradation. This is not expected and could be caused by the harder nature of using global rewards, thus needing more training time until efficient exploratory behaviors emerge.
- Regarding agent-agent collisions, the rich environment algorithms achieve similar performances, with MAPPO-mixed and MAPPO-global Rich Envs outperforming IPPO. Regarding the non-rich environment algorithms, IPPO and MAPPO-mixed show weak exponential scalability, thus being noticeably worse than MAPPO-mixed, which shows linear scalability. In this sense, MAPPO has better performance than IPPO. This is expected since using CTDE can foster the emergence of policies that result in coordinated behaviors.
- Concerning obstacle collisions, the rich environment algorithms collide less, with the exception of MAPPO-Mixed. This could occur because the training time in the rich environment is half that of the one done in the non-rich case, and with further training, it is theorized that the performances can be similar.
- In all scenarios, **training with a rich environment achieves similar or better performance than training without. This is a noticeable performance gain, given that the Rich Env algorithms trained for less time**, and aligns with the findings of [19], although this work extends it to a multi-agent scenario.
- From the aforementioned observations, **MAPPO-mixed Rich Env achieves the best scalability performance**.

F. Improving the TRL of Swarm Missions

After examining the learning performance of IPPO and MAPPO with different reward functions and environments, it is important to assess whether the policies contribute towards the NASA requirements listed in section III. In particular, the learned policies contribute to the following requirements:

Contributing to Req 1. Exploration, mapping, and sampling

1) The agents can explore autonomously, and the algorithms have no limitations in operating times.
2) Relative pose estimation is not addressed by these results.
3) The agents can keep formations and achieve collaboration behaviors.
4) The algorithm can be used as a first step to develop a distributed cooperation and mapping system, already showcasing adaptive sampling strategies.
5) The agents indirectly achieve inter-agent communication. However, a study of direct communication channels is not addressed in this work.

Contributing to Req 2. Cooperative Task and Task Allocation

1) The agents can recognize the task that needs to be performed based on LIDAR perception, hence considering environmental cues.
2) The agents dynamically allocate the exploration of targets based on the swarm state.

As an important remark, the algorithms fulfill these requirements to a certain extent, but **not fully**. However, it is hypothesized that with longer run times, bigger artificial neural networks, higher fidelity simulations, and thorough policy verification and validation, this framework can eventually be deployed in real swarm systems.

VIII. Conclusion

This work combines two fields; swarm space exploration, and multi-agent reinforcement learning. This is done by phrasing the swarm learning problem as a decentralized partially observable Markov Decision Process and considering planetary mission requirements. Moreover, the learned policies are placed within a NASA MaCMAS architecture, taking guidance actions. This is done within a simulated exploration environment where agents have target locations they need to explore, and obstacles to avoid. This environment was phrased to have similarities to the previous state-of-the-art [9], but severely increasing the complexity of the learning problem regarding environment features, the use of sparse reward functions that can be obtained from mission requirements, and the number of agents.

To solve the learning problem, a multi-agent extension of the PPO algorithm has been developed, which employs LIDAR perception for the policy, and a global environment tensor for the CNN of the critic when in CTDE (referred to as MAPPO). In this sense, both the environment and the algorithm provide a further step toward learning policies that can be used in a real exploration mission.

To further study the nature of the problem, the IPPO and MAPPO configurations are compared with reward functions that only account for agent-specific behavior, full swarm behavior, or a combination of the two. The IPPO configuration of the algorithm, using a LIDAR for the critic, achieves comparable performance to MAPPO when using local or mixed reward functions but, notoriously, MAPPO doubles IPPO's performance when using global reward settings, and achieves more stable learning in most scenarios.
Moreover, both IPPO and MAPPO can learn with more agents than the algorithm in [9] (7 vs 5). Also, the algorithms learn policies that achieve emergent cooperation behaviors and attain good learning scalability; tested training with both 3 and 7 agents.

Furthermore, **depending on the nature of the cooperative learning problem**, global reward functions are found to unnecessarily complicate the learning task, as there are scenarios where using local or mixed rewards also obtains high-performing cooperation policies (sometimes better than the ones obtained with global rewards).

Moreover, training in a rich environment achieves better generalization and scalability capabilities compared to always training in the same environment. This was achieved for all reward function types. This is a noticeable performance gain given that the rich environment algorithms trained for shorter times.

Additionally, for all tested algorithms, the learned behaviors scale to swarm sizes not seen during training, thus highlighting the potential of using MARL to learn policies that are flexible to different numbers of agents.

Finally, the learned policies improve the TRL of swarm planetary exploration missions, contributing to exploration, mapping and sampling, and cooperative task and task allocation, according to NASA requirements.

A. Significance of Contributions

To the best knowledge of the author, this is the first work that combines rigorous swarm planetary mission requirements and MARL. The flexibility of the assumptions used to construct the learning problem is a step forward in developing a learning algorithm that can be applied in a real swarm mission, as well as developing MARL algorithms that can be utilized in harder multi-agent problems (with more complex environments, number of agents, etc). This technology can potentially be applied in any field where several decentralized agents take actions that affect some desired outcome; such as the aviation field, logistics, terrestrial exploration, etc.

B. Recommendations

Further designing the swarm planetary mission: The proposed MaCMAS architecture is a simplified model of a full swarm mission. Having a more detailed design of such missions can further reveal the requirements that need to be satisfied, and the relevant metrics that need to be accounted for in the reward function.

Improving the environment: To bridge the reality gap between simulation and reality, more realistic environments can be used, where the low-level dynamics of the agents are more accurately simulated, as well as the inputs from the higher abstraction layers within the swarm mission architecture.

Improving the perception models: The actor and critic perceptions are critical for achieving successful learning behaviors. For the actor, more accurate perception models can be developed, such as using a more realistic LIDAR simulation and/or adding the reading of other sensors. Regarding the critic, other solutions can be explored, such as using other combinations of global and agent-specific environment information; using clustering techniques, modeling the actions of other agents, or similar heuristics.

Improving the actor and critic models: This study used a feed-forward neural network for the actor, and a CNN for the critic. These models offer poor flexibility regarding the size of the input, and in the case of the CNN, the generated input tensor is expensive in terms of memory, and rigid in terms of the spatial representation of the environment information. Learning performance can likely be enhanced by using models that have some form of time memory, such as the recurrent neural networks used in [11]. Moreover, in combination with improved perception models, having learning models that can selectively process the input information can be advantageous, using attention mechanisms to improve the scalability of the critic [20], or similar.

Improving the learning algorithm: Techniques such as using entropy or Kullback-Leibler divergence can potentially improve the training performance, offering a balance between exploration and exploitation. The usage of entropy will require using stochastic policies, however, and such a system might be harder to verify for planetary missions. Moreover, it is hypothesized that from a theoretical perspective, there might be structures within the parameter space of the learning models that reflect the emergence of patterns in behavioral space. Studying such behaviors can potentially lead to a better understanding of black-box models and the nature of multi-agent systems.

References

[1] Thai, Z. W., Balasubramani, P., Brand, C., Haines, A., and DeLaurentis, D. A., "Study of Swarm-based Planetary Exploration Architectures Using Agent-Based Modeling," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020. https://doi.org/10.2514/6.2020-0075, URL https://arc.aiaa.org/doi/10.2514/6.2020-0075.

[2] Rahmani, A., Bandyopadhyay, S., and Rossi, F., "Space Vehicle Swarm Exploration Missions: A Study of Key Enabling Technologies and Gaps," *70th International Astronautical Congress*, 2019.

[3] Vitug, E., "Cooperative Autonomous Distributed Robotic Exploration (CADRE)," *NASA*, 2021. URL http://www.nasa.gov/directorates/spacetech/game_changing_development/projects/CADRE.

[4] Staudinger, E., Shutin, D., Manß, C., Viseras, A., and Zhang, S., "Swarm Technologies For Future Space Exploration Missions," *14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-sairas)*, 2018.

[5] Rouff, C., "Intelligence in Future NASA Swarm-based Missions," *AAIA Fall Symposium*, 2007.

[6] Buşoniu, L., Babuška, R., and De Schutter, B., "Multi-agent Reinforcement Learning: An Overview," *Innovations in Multi-Agent Systems and Applications - 1*, Vol. 310, edited by D. Srinivasan and L. C. Jain, Springer Berlin Heidelberg, 2010, pp. 183–221. https://doi.org/10.1007/978-3-642-14435-6_7, URL http://link.springer.com/10.1007/978-3-642-14435-6_7, series Title: Studies in Computational Intelligence.

[7] Oliehoek, F., and Amato, C., "A Concise Introduction to Decentralized POMDPs," 2016. https://doi.org/10.1007/978-3-319-28929-8.

[8] Bernstein, D. S., Zilberstein, S., and Immerman, N., "The Complexity of Decentralized Control of Markov Decision Processes," *Conference on Uncertainty in Artificial Intelligence*, 2000. URL https://api.semanticscholar.org/CorpusID:1195261.

[9] Huang, Y., Wu, S., Mu, Z., Long, X., Chu, S., and Zhao, G., "A Multi-agent Reinforcement Learning Method for Swarm Robots in Space Collaborative Exploration," 2020, pp. 139–144. https://doi.org/10.1109/ICCAR49639.2020.9107997.

[10] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.

[11] Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y., "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games," *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[12] Zhang, K., Yang, Z., and Başar, T., "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," *ArXiv*, Vol. abs/1911.10635, 2019. URL https://api.semanticscholar.org/CorpusID:208268127.

[13] Dudukovich, R., Wagner, K., Kancharla, S., Fantl, J., and Fung, A., "Towards the Development of a Multi-Agent Cognitive Networking System for the Lunar Environment," *2021 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, IEEE, 2021, pp. 7–13. https://doi.org/10.1109/WiSEE50203.2021.9613839, URL https://ieeexplore.ieee.org/document/9613839/.

[14] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I., "Emergent Tool Use From Multi-Agent Autocurricula," *ArXiv*, Vol. abs/1909.07528, 2019. URL https://api.semanticscholar.org/CorpusID:202583612.

[15] Chen, Z., Subagdja, B., and Tan, A.-H., "End-to-end Deep Reinforcement Learning for Multi-agent Collaborative Exploration," *2019 IEEE International Conference on Agents (ICA)*, 2019, pp. 99–102. https://doi.org/10.1109/AGENTS.2019.8929192.

[16] Guan, Y., Zou, S., Peng, H., Ni, W., Yanglong, S., and Gao, H., "Cooperative UAV Trajectory Design for Disaster Area Emergency Communications: A Multiagent PPO Method," *IEEE Internet of Things Journal*, Vol. PP, 2023, pp. 1–1. https://doi.org/10.1109/JIOT.2023.3320796.

[17] Peña, J., Rouff, C., Hinchey, M., and Ruiz-Cortés, A., "Modeling NASA swarm-based systems: Using agent-oriented software engineering and formal methods," *Software and System Modeling*, Vol. 10, 2011, pp. 55–62. https://doi.org/10.1007/s10270-009-0135-2.

[18] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *CoRR*, Vol. abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

[19] Heess, N. M. O., Dhruva, T., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M. A., and Silver, D., "Emergence of Locomotion Behaviours in Rich Environments," *ArXiv*, Vol. abs/1707.02286, 2017. URL https://api.semanticscholar.org/CorpusID:30099687.

[20] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I., "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," , 2020-03-14. URL http://arxiv.org/abs/1706.02275.

IX. Appendix

A. Policy Trajectory Traces

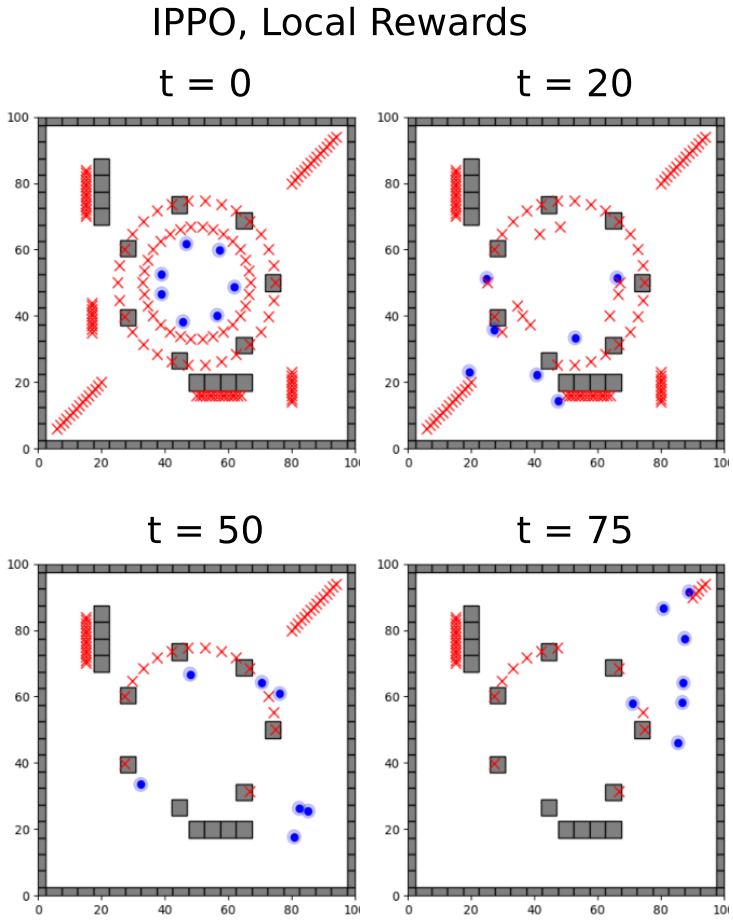


Fig. 20 Trajectory trace of the learned policy from IPPO, using local reward functions. The targets disappear from the environment as they are explored. It is observed that as time progresses, the agents collect more targets, starting from closer ones, and eventually reaching the furthest ones in the limits of the map.

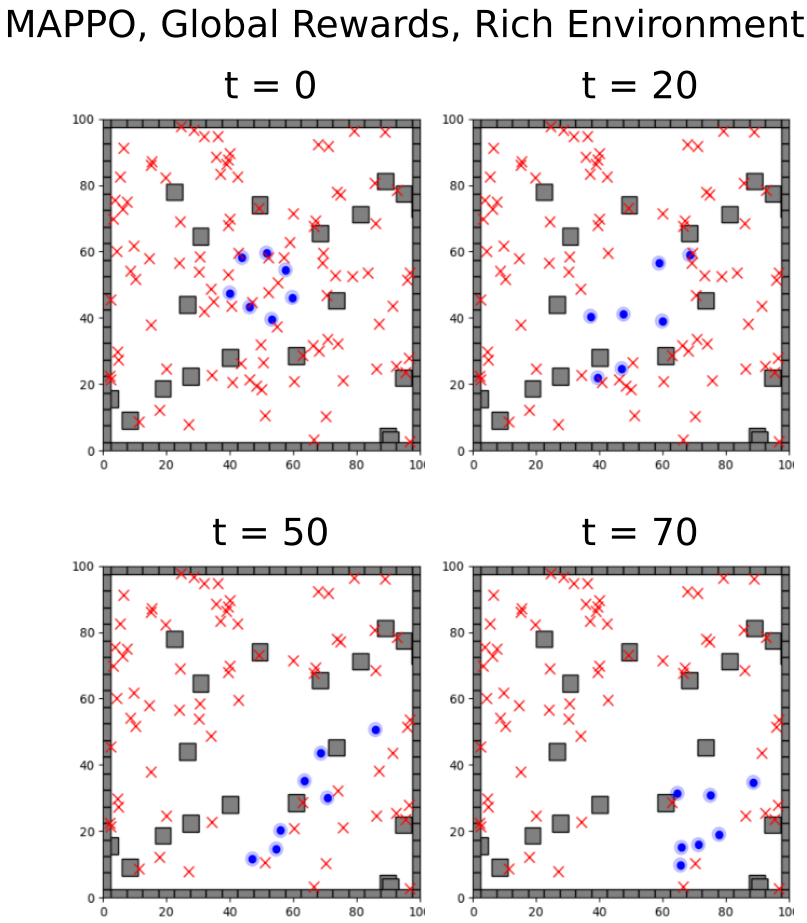


Fig. 21 **Trajectory trace of the learned policy from MAPPO, using local reward functions, and a rich environment. The targets disappear from the environment as they are explored. It is observed that as time progresses, the agents collect more targets, collaboratively sweeping the map towards the bottom-right corner.**

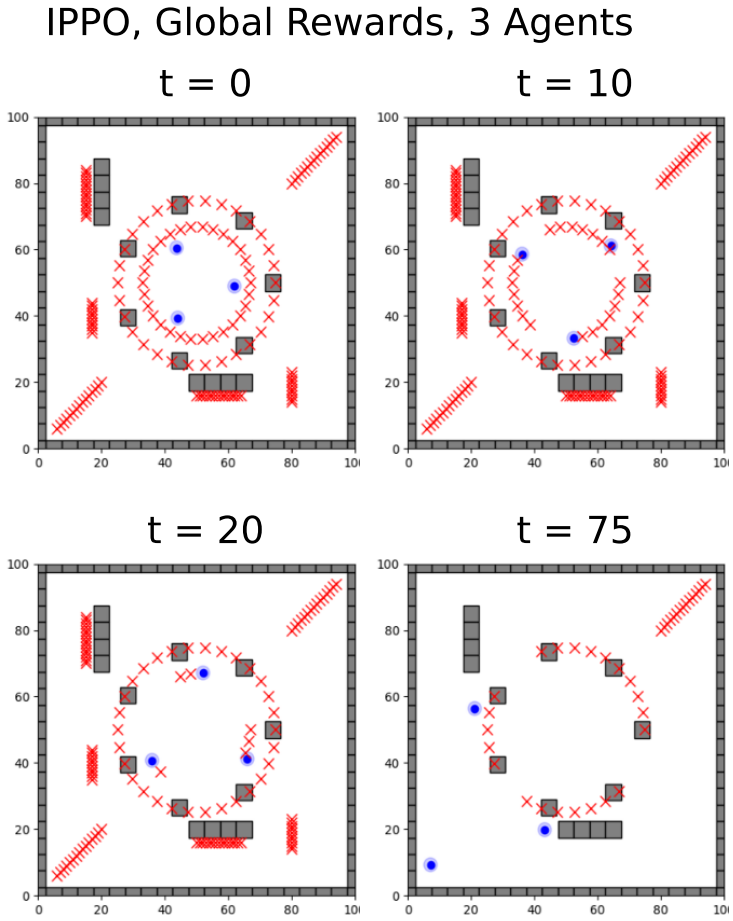


Fig. 22 Trajectory trace of the learned policy from IPPO, using local reward functions, and 3 agents during training. The targets disappear from the environment as they are explored. Also, the agents learn to collaborate, learning a circular movement pattern to explore the circle of targets near them. Afterwards, the agents explore most of the map's targets.

Part II

Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

3

Literature Study

This chapter investigates how can MARL be successfully applied to swarm space exploration. This involves three salient fields of knowledge:

1. Swarm space exploration.
2. Swarming.
3. Reinforcement learning.

These domains are investigated to guide the research from the wide field of swarm space exploration to a narrow scope where to develop a MARL algorithm.

Firstly, the field of swarm space exploration is studied in Section 3.1 to understand the trends in the space exploration field, as well as the enabling technologies that are needed to explore where can MARL be potentially applied. Also, relevant requirements are identified according to NASA, to further define the MARL problem.

After this, swarming is examined in Section 3.2; defining what is meant by ”swarm” in this investigation, examining swarm architectures that can later guide the design of a MARL algorithm, and describing an existing verification method system to consolidate the applied MARL algorithm.

Then, the field of reinforcement learning (RL), constituting the backbone of the algorithms in this study, is examined in depth (Section 3.3); first outlining the single-agent RL scenario (the foundation of the field) to finally delve into the multi-agent case; highlighting the different aspects that need to be accounted for to formulate a MARL problem, as well as overviewing state-of-the-art algorithms that can potentially be applied to space exploration. Finally, the existing MARL efforts applied to swarm space exploration are described and assessed, to provide an understanding of the current state of the field.

With the gathered information, three state-of-the-art single-agent algorithms (SAC, DDPG, and PPO) are tested on a simulated cooperative multi-robot environment in Section 3.4. This is done to find the limitations of naively extending single-agent RL to the multi-agent scenario, assessing their scalability capabilities, as well as allowing for the familiarisation of deploying RL systems.

Drawing upon insights learned from swarm space exploration, swarm system, and MARL understanding, and the empirical results from the RL tests, a preliminary swarm system is described in Section 3.5. The potential MARL implementation within this system is also specified, as well as the scope of its implementation for this MSc thesis, and the research plan to perform the aforementioned implementation.

Finally, Section 3.6 concludes by assessing the outcomes of the literature study. These efforts aim at producing research that is meaningful to the field of swarm space exploration. The degree to which these questions are answered can then help guide the following efforts in the MSc thesis.

3.1. Swarm Space Exploration

In this research, swarm space exploration plays a pivotal role, since the main goal is to contribute towards the technological development of the field. With this in mind, this chapter aims to provide an appropriate

background of the field; highlighting trends, discussing the technological development, as well as the top-level requirements that swarm space systems need to fulfill, and answering Research Questions 1 and 2.
Firstly, the historical overview of space exploration trends concerning swarm missions is discussed in Section 3.1.1. Secondly, Section 3.1.2 identifies the technology readiness level in the different areas of the field. Moreover, swarm space systems need to fulfill specific sets of requirements in order to successfully accomplish space missions. This is examined in Section 3.1.3. Lastly, the findings of the aforementioned subsections are summarised in Section 3.1.4.

3.1.1. Historical Overview and Current Trends

Until the mid-1980s, space missions were operated manually from ground control centres [7]. This led to high costs of satellite operation missions and prompted NASA and others the automation of as many functions as possible [7]. These automation efforts are being continued these days. Cost reductions can be achieved in a number of areas. Relevant to this study, spacecraft operations is one of such key areas [7]. Since then, more reliance has been put on intelligent systems, as stated by NASA in 2006 [7], which are also necessary to enable more complex missions, including those with swarms of spacecraft.

Currently, and within the last decade and vicenary, several efforts are being made to contribute to the possibility of having swarm planetary exploration; from specific robots, to control architectures. Some of those efforts are reflected in Table 3.1, where the shown robots are designed for Earth, Moon, Mars, asteroids, and icy moons missions. Figure 3.1 shows a conceptual mission of one of such robots; the PUFFER, where the **distributed exploration and beyond line-of-sight communications** (made possible by distributed communication strategies) are some of the key characteristics of such a swarm mission.

An important milestone contributing towards the field of swarm planetary exploration is the Mars technology demonstrator Ingenuity (see last row of Table 3.1); a small helicopter designed with the objective of completing a 30-seconds flight in Mars, and that due to its successful performance, it has now done more than fifty flights, shifting from a pure technology demonstration to an operations demonstration [8]. These successes have several implications for this research. Firstly, the Perseverance-Ingenuity mission proves the plausibility of having multi-agent planetary exploration, the Perseverance rover carrying the science experiments while Ingenuity, as quoted from NASA engineer Olivier Toupet, was eventually used to *"have that helicopter imagery to refine our strategic route and plan to avoid challenging terrain well before the rover can see it"* [8]. Secondly, it shows that this multi-agent cooperation can lead to faster, safer mission executions, with the Perseverance rover navigating the martian Séítah section more efficiently that it would have otherwise, in the words of NASA: *"having the helicopter, we were able to plan the* [Séítah] *route ahead of time, and make a* ***much better estimate of how long it would take, which helps the whole Perseverance rover team to plan more efficiently. That's pretty valuable****"* [8].
Lastly, it points out to the trends in the field that can also be observed from the other entries of Table 3.1 and outlined in *Intelligence in Future NASA Swarm-based Missions* [9], namely: the increasing possibility of using different, specialised robots that will allow performing missions that are currently not possible with monolithic architectures (in homogeneous or heterogeneous configurations), the increased complexity of such missions and the need for **autonomous** and autonomic systems, and crucially, the use of **intelligent** swarms that have the capability of **learning from their environment**. In the near future (2024) NASA is planning on sending the CADRE (see fourth row of Table 3.1) mission to the Moon, consisting of a network of shoe-box-sized mobile robots for Moon surface and lava caves exploration [3], thus further accentuating the swarm space exploration trend. As it will be discussed in Section 3.3, **reinforcement learning has the potential to contribute toward further advancing these trends.**

Table 3.1: Different architectures, and robots, with applications to swarm planetary exploration.

Name	Short Description	Status
A-PUFFER [10]	Foldable robot that can access tight spaces.	Completed (2020).
CAMPOUT [11]	Control architecture for multi-robot planetary missions.	Completed (2000).
Marsbee [12]	Bio-inspired Martian robotic flight vehicles.	Ongoing.
CADRE [3]	Shoe-box-sized mobile robots for autonomous robotic exploration.	Ongoing (2024 launch).
DUAXEL [13]	Rovers designed to access high-risk terrain on planetary surfaces.	Ongoing.
Ingenuity [14]	Martian helicopter technology demonstrator.	Deployed (2021).

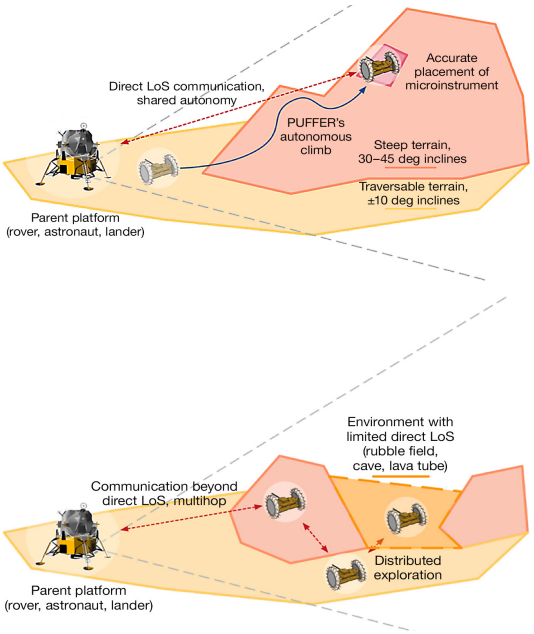


Figure 3.1: *"Conceptual missions illustrating a PUFFER climbing a steep incline to accurately place a microimager for stratigraphy, and multiple PUFFERs exploring a rubble field-like environment, while maintaining a communication network to the parent platform that is beyond direct line of sight of most PUFFERs".* Adapted from NASA [10, 2].

3.1.2. Enabling Technologies

In order to identify the technologies needed to progress in the field of space and planetary exploration, an assessment of the current state in the field is necessary. Figure 3.2 shows the different Technology Readiness Levels (TRL) for space, Earth, and planetary multi-spacecraft and swarm mission types [2]. Here, a few conventions and observations can be made. Firstly, in the field of swarm space exploration, two sub-fields arise; **space exploration**, which refers to explorations involving a swarm or constellation of spacecraft that either perform a formation flying mission, or a constellation mission, and **planetary exploration**, which entails exploring planetary/moon surfaces, atmosphere, and water bodies [2]. This research is focused on the later case, given that, as observed in Figure 3.2, planetary exploration has a much less mature TRL when compared to space exploration, and hence it is critical to contribute to the field to enable the possibility of such future missions. In addition, it is important to note that in this report, the terms space and planetary exploration are used interchangeably, and always refer to the aforementioned definition of planetary exploration.

Planetary exploration is of paramount importance to NASA's JPL mission [2]. Unlike Earth missions of multi-agent systems, space-based systems present two key differences, namely, the lack of existing infrastructure makes multi-agent robotics systems more attractive than their Earth counterparts (e.g. for communication relays), and secondly, the strict requirements on the **autonomy** and **resilience** of space

multi-agent systems, making many partially-supervised solutions unfeasible in space applications [2]. Figure 3.2 shows the different planetary mission types and the different elements of such missions. Here, it can be observed that for planetary exploration, the TRL is not very mature; with most elements having a TRL of 3-5, only the areas of relative pose estimation and formation keeping having a TRL of 6-8, all other TRLs having no development. Moreover, several technology gaps have bee identified, according to NASA's JPL [15, 2]:

1. Resource-aware and network-aware autonomous **task identification and task allocation for robot teams**.
2. Algorithms for optimising what and when to communicate among assets, given the costs of the communication and the benefits of coordination.
3. Relative localisation/team member pose estimation from on-board sensors and subject to computational and network conditions of small spacecraft.
4. **On-board risk awareness** and incorporation of risk into mission and motion planning.
5. **Mission planning and scheduling that accounts for multiple dynamic assets**; synchronisation and/or distribution of plans.
6. Autonomy and network software systems designed explicitly to coordinate multiple spacecraft.
7. Human interfaces and autonomy software designed for an updated operations paradigm; overall, great individual autonomy will be needed as **human sequencing for all agents is likely too cumbersome/impractical**.
8. Smaller and cheaper communications and sensor equipment, shifting the **focus from individual robustness to redundancy**.
9. Miniaturisation, with the development of small, low-cost platforms less demanding in terms of on-board resources, and **observation systems using several platforms**.

In particular, and relevant to this research, NASA's JPL proposes the *"Development of flight software designed from the ground up to allow multi-robot and multi-spacecraft coordination and operations"* [2].

Domain	Mission Type	Absolute Pose Estimation (metrology)	Relative Pose Estimation (metrology)	Time Synchronization	Formation Keeping	Distributed Inter-Vehicle Communication	Modular Space Systems	Cooperative Manipulation	Distributed Estimation and Cooperative Mapping	Cooperative Motion Planning	Cooperative Task Recognition and Task Allocation	Human-Space System Interface
Space	Satellite Navigation	●[1]		●		◑						
	Earth Observation	●	◑	●		◑			◕	◕	◕	
	Gravity Measurement	●	◕	◕	◕	◕			◕	◕		
	Distributed Aperture Telescopes	◕[2]	◑	◕	◑	◑			◑	◑	◑	
	Distributed Fractionated Spacecraft	●	◑	●		◑	◑			◑	◑	
	In-orbit Assembly and Servicing	●	◕	●	◕	◕	◑	◑		◑	◑	
	Solar Observation	◕	◑	◕	◑	◑			◕	◑		
	Planetary Exploration and Mapping	◕	◑	◕		◑			◑	◑	◑	
	Distributed Communication Array	◕	◑	◑		◑				◑		
	Interplanetary Missions	◔[4]	◔	◔		◔			◔	◔	◔	
Earth	Exploration, Mapping, and Sampling	◑[3]	◑	◕		◑			◑			
	Cooperative Lifting, and Assembly				◕			◑		◑	◑	
	Communication Infrastructure			◕		◑				◑	◑	
	Disaster Recovery/Search and Rescue								◑		◑	◑
	Reconnaissance, Patrolling and Tracking								◑	◕	◑	◑
	Urban Transportation/Delivery Systems									◑	◕	
	Entertainment		●		●							
Planetary	Exploration, Mapping and Sampling		◑	◕	◕	◑			◑			
	Cooperative Construction				◑			◑		◑	◑	
	Communication Infrastructure			◕		◑				◑	◑	
	Cooperative Computation					◑					◑	

[1] ● : the technology is mature (9 TRL)
[2] ◕ : the technology is currently under development, but quite mature (6–8 TRL)
[3] ◑ : the technology is currently under development, but not very mature (3–5 TRL)
[4] ◔ : the technology is currently not available or in conceptual stages of development (1–2 TRL)

Figure 3.2: Enabling technologies for multi-spacecraft and swarm mission types (as of 2019). Retrieved from [2].

3.1.3. Identified Requirements

Together with describing the enabling technologies needed for multi-spacecraft missions, NASA [2] has also identified requirements that such missions need to fulfil. Aligning this research with these requirements allows for developing intelligent exploratory swarm systems that have the potential to be incorporated in real space missions, as well as guiding the future design decisions of this work. In particular, requirements are set for the four domains of planetary exploration shown in Figure 3.2.

Req 1. Exploration, mapping, and sampling

1. **Autonomy**: the platforms should be able to operate for hours to days with no humans in the loop [2].
2. Relative pose estimation [2].
3. Formation keeping: the vehicles should be able to assess and, in many applications, control their relative location to a high degree of accuracy [2].
4. **Distributed estimation and cooperative mapping** to build real-time maps of the observed quantities and enable adaptive sampling strategies [2].
5. **Time synchronisation.** The vehicles must have access to synchronised clocks to time-stamp the collected data. Synchronisation accuracy can vary from minutes (for slow-changing phenomena) to sub-ns (for radio science, and in particular multi-static RADAR) [2].
6. Distributed inter-agent communication: in order to cooperate and relay data to Earth, the agents must establish and maintain a communication network. To support geographically-distributed agents, the communication mechanism may need to support reconfigurable multi-hop communications – a novel requirement for space applications, where the communication topology is generally single-hop or (in the case of rover-orbiter relays) well-defined in advance [2].

Req 2. Cooperative Construction

1. **Cooperative manipulation**, required to move parts and assemblies that are too large for a single robot to handle [2].
2. Cooperative motion planning [2].
3. **Formation keeping** to coordinate the motion of multiple vehicles grasping the same part or assembly [2].
4. **Cooperative task recognition and task allocation** to map a high-level assembly task to a set of grasping and motion tasks for the robots [2].

Req 3. Communication Infrastructure

1. **Cooperative task recognition and task allocation** to decide which agents should act as communication relays [2].
2. **Cooperative motion planning** to identify suitable locations where the relay agents should position themselves [2].
3. **Distributed inter-vehicle communication** to route packets appropriately [2].
4. **Time synchronisation** to enable time division multiple access control protocols [2].

Req 4. Cooperative Task and Task Allocation

1. **Recognise tasks** that should be performed based on environmental cues observed by the agents [2].
2. Assign such tasks according to the **agents' states and capabilities** [2].

Lastly, swarm-based missions must use **evolving systems** [16]. With this, the swarm must be able to adapt to the environment in the event of uncertainties, and when agents malfunction or fail.
Notice that in this research, the focus would be set on fulfilling certain requirements related to a specific part of a swarm mission, which is decided in Section 3.5 with the information gathered in the literature study.

3.1.4. Conclusion

Several conclusions can be reached from this section. Firstly, there is a clear trend in the space exploration field towards multi-agent intelligent systems. Although single-agent space exploration missions have been the norm in the past century, multi-robot research has been increasingly expanding, with new systems being investigated, and partially demonstrated (as in the case of the Ingenuity helicopter). An important milestone in the field is the CADRE robots that will be deployed in 2024 for a Moon exploration mission.
Furthermore, a significant knowledge gap has been identified in terms of the low technology maturity of space swarm exploration, where the planetary missions (chosen as the main focus of this research), have the lowest level of development, with no specific technology being fully mature.
Moreover, the top-level NASA requirements of the different planetary mission types have been recognised, and are of paramount importance for the successful design of swarm systems in this research.

3.2. Swarming

The concept of a swarm and swarming constitutes one of the building blocks of this research; since the collection of robots performing a space exploration mission needs to work as a system exhibiting some global behaviours (such as the successful completion of a specific task) while maintaining an individual identity. These ideas are grounded in the concept of a swarm. This section aims to first define what is meant by ”swarm” (Section 3.2.1), and later define swarm system architectures relevant to space missions (Section 3.2.2, as well as the way such swarm systems can be verified, in Section 3.2.3. Finally, the gathered findings are summarised in Section 3.2.4. These investigations will help answer Research Question 4.

3.2.1. Swarms

When looking at nature, it is not uncommon to find groups of living entities that seem to have a form of aggregate motion (referred to as swarming behaviour) or emergent behavioural patterns; flocks of birds, ants foraging for food, bees building a hive, etc. Such groups of living entities are referred to as swarms [17], and in the engineering field, there is an increasing interest in understanding the behaviours they exhibit, as they can be useful for optimisation problems, robotics, traffic control, etc [17], or in this context, achieve space exploratory tasks.

At a high level, the N agents in the swarm seem to be cooperating to achieve some specific global behaviour and achieve some goal [17]. This ”collective intelligence” emerges from relatively simple individuals who abide by some local rules that govern their actions [17], and via the interactions of the entire group, the swarm achieves its objectives [17].

With this in mind, **in this research a swarm is thus defined as a collection of individual agents that, when cooperating with each other, manage to achieve a common global goal**. The design of such swarms thus involves aspects such as hierarchical organisations, communication, and learning, with the final objective of achieving a swarm system that, in the context of this work, can successfully explore a planetary environment. Swarms can be composed of similar agents (homogeneous swarms) or different agents (heterogeneous swarms) [17], allow for global or partial communication between the agents, etc.

3.2.2. Architectures

When dealing with swarm systems, several architectures and information structures are possible. In this work, we consider NASA’s **Methodology fragment for analysing Complex Multiagent Systems** (MaCMAS) architecture due to its tailoring to space exploration, and several information structures that can later be implemented in reinforcement learning systems (see Section 3.3) and/or have been considered for space exploration missions. Lastly, a verification method for adaptive systems is investigated to provide a framework to later verify the swarm system(s) designed in this work.

MaCMAS

NASA’s MaCMAS architecture is tailored to model complex multi-agent systems [16], configuring agents into sub-organizations, groups, and teams. This is shown in Figure 3.3, where the system model is split into different layers that increase the abstraction level from the micro to the macro level. Such departmentalisation allows for the traceability of the system at different refinement levels, helping establish a traceable multi-agent system that follows both micro and macro requirements. Furthermore, in this configuration, **the roles of each agent can change over time** [16], as occurs in the proposed Autonomous NanoTechnology Swarm (ANTS) architecture.
In this architecture (see Figure 3.4), a model of the system at the micro-level of tasks can be linked with a model of the system at the macro-level [16] and can help ensure properties of the system at the micro, macro, and intermediate levels. This can be used to model the autonomous properties of the swarm system and create traceability models, such as the one done for ANTS, Figure 3.4, where horizontal lines represent abstraction levels and vertical lines separate autonomic and autonomous properties.

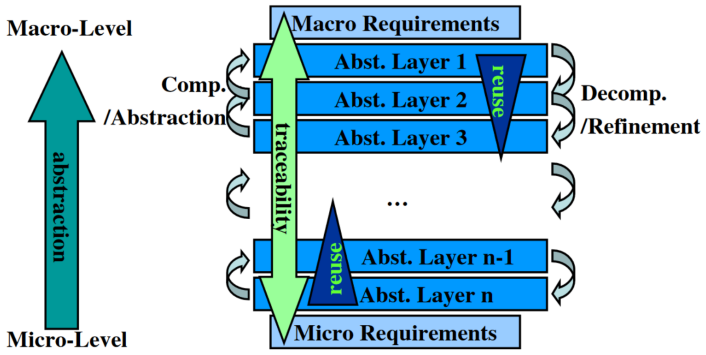


Figure 3.3: Overview of the structure of NASA's MaCMAS models. Retrieved from [16].

To represent swarm systems, MaCMAS proposes two kinds of models: **role models**; showing the relationships between roles, and **plan models**, showing how these roles evolve over time. Moreover, MaCMAS can model evolving systems, as required by swarm space missions (see Section 3.1.3), by representing each transition as the addition or elimination of a set of features [16].
Lastly, it should be noted, that in other works, the swarm hierarchy also distinguishes between levels of automation [18], ranging from fully autonomous agents to teleoperated systems.

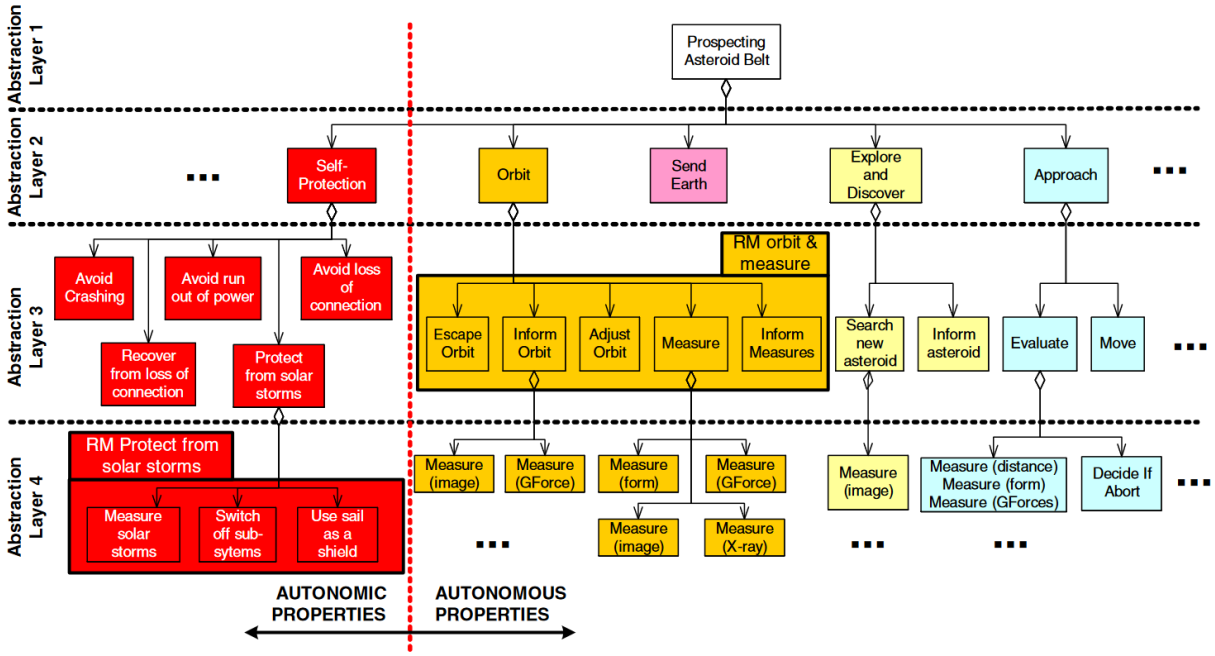
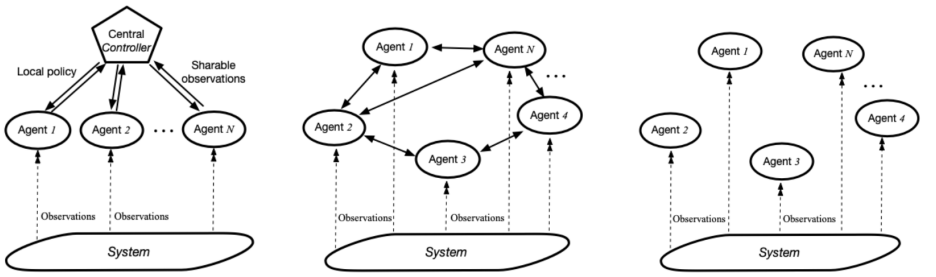


Figure 3.4: Traceability model of NASA's ANTS architecture. Retrieved from [16].

Information Structures
Information structures involve the *who knows what* within the swarm [19]. Due to its connection with multi-agent reinforcement learning (MARL) (discussed in Section 3.3.2), three information structures are considered, shown in Figure 3.5, discussed by Zhang et al [19].
The first structure concerns a centralised setting, where a **central** controller can collect information from the individual swarm agents, such as their actions, observations etc, and either give commands to the individual agents of the swarm or adapt the controllers of the individual agents (note that the way the controllers adapt in a reinforcement learning setting (RL) are discussed in Section 3.3). Such structure greatly simplifies the analysis of the swarm, and standard single-agent reinforcement learning analysis can be used [19]. This information structure, in the context of this work, can allow for more stable learning of the swarm agents in their environment (Section 3.3.2), but comes at the cost of requiring a stable connection

with the central controller, thus limiting the complexity of the missions. Furthermore, the agents don't interact with one another independently, limiting the adaptability of the swarm in deployment conditions.
Secondly, there can be a decentralised information structure where information is exchanged **locally** among the agents, much like in a biological swarm. In this scenario, the agents are connected via a (possibly) **time-varying** communication network [19]. This allows for a more adaptive swarm when compared to a centralised structure, although it aggravates the non-stationarity of the environment during the learning of the agents since the other agents' actions need to be predicted (as opposed to a central controller which knows all the actions taken by all the agents).
Finally, there can be fully decentralized structures, where the agents interact **individually** with the environment, with no explicit information exchange with one another. This limits the coordination capability of the swarm, but can be useful for game-theoretic learning algorithms [19], or in the case where the actions of other agents minimally change the environment, can allow the deployment of classic single-agent RL algorithms.



(a) Centralized setting

(b) Decentralized setting with networked agents

(c) Fully decentralized setting

Figure 3.5: Types of information structures in multi-agent reinforcement learning swarms. Retrieved from [19].

Moreover, the communication structure of NASA's CADRE mission [3] (discussed in Section 3.1.1) is also of relevance, given it is a multi-agent exploration mission that NASA is planning to send to the Moon in 2024. In such a mission, *"the robots would communicate with one another, and together they would share information and transmit that information back to a base station on the lander. The base station then redistributes all the collective data back out to the rovers. Frost calls this 'high brain' which entails the base station computer taking all the robots' collective information, such as maps, putting it together and sending it out to the robots with more detailed information and compiled maps"* [3]. Moreover, in scenarios with unstable communication between robots and the base, the communication can be established *"as long as there is good signal range between the multi-agents, robots and rovers, then data can be transmitted back to the ones on the lunar surface to communicate some never-before-seen data"* [3]. This suggests an information structure that combines both (a) and (b) configurations in Figure 3.5, with a centralised controller (the base), that can give high (and maybe low) level commands to the robots, as well as managing the information sharing among agents, and decentralised communication in scenarios where communication with the base is not possible, to then update the information again with the base when communication can be reestablished.

3.2.3. Verification of Swarm Systems

Adaptive systems such as swarms used for space exploration, critically need to be verified, given the costs of a space mission, and the importance of its fulfillment. To provide a complete approach to the verification and deployment of adaptive systems, Christopher Rouff et al [20] developed the *AdaptiV* tool chain and methodology. *AdaptiV* can be summarised as follows:

1. A stability analysis capability that identifies instabilities given a system model and partitions the system model into stable and unstable component models.
2. A state space reduction capability that prunes the state space of an unstable component model without loss of critical fidelity.

3. High-performance computing (HPC) simulations to explore component behavior over a wide range of an unstable component's reduced state space and produce a statistical verification for the component.
4. A compositional verification capability that integrates individual component verifications.
5. Operational monitors to detect and take action to correct undesired unstable behavior of the system during operation.

Christopher Rouff et al [20] argue that adaptive systems have large state spaces that must be reduced for verification, stating that *"state space reduction is achieved by aggregating state transitions into an abstract (coarser-grained) finite state model of the system"*. This requires a trade-off between the model's complexity and its fidelity, with the goal of achieving enough precision related to the specific interest of the verification.

Then, a stability analysis of the system (which, importantly for this research, can include artificial neural networks, of which methods have been developed [20, p. 2]) is done to identify the unstable parts of an adaptive system. This analysis needs to be expanded by performing HPC simulations to statistically compute a confidence level of the convergence ability of the system. Furthermore, *"while adaptive systems may be inherently unstable because of operational needs – e.g., the need to adapt in real time – this is not necessarily a reason for failing verification. An unstable system may still converge, even though* ***complete verification may not be possible****"* [20, p. 2].

Furthermore *"the above results will then be combined to yield a probabilistic measure of confidence in component behavior and provide state space convergence parameters that identify potential symptoms of unstable behavior. Where comprehensive verification is not possible, operational monitors can be deployed with the adaptive system. Monitors will be able to be automatically generated and deployed to detect non-convergence symptoms during operation and guide the adaptation towards stable behavior"* [20, p. 2].

A noticeable challenge with the verification of swarm systems is the compositional verification, since the adaptation of a swarm element (such as an agent for example) may in turn cause adaptations in other components [20], the mutual interaction among swarm elements affecting the overall system behavior. This non-stationarity in the system is further investigated in Section 3.3.2. In the *AdaptiV* approach, a combination of results from the verification of individual system components is used to produce an overall system-wide verification. In this context, several invariants are considered: mission goal invariants, behavior invariants, interaction invariants, and resource invariants [20, p. 4]. Behavior invariants relate to the reachability of safe states by the different system components, whereas interaction invariants are global constraints on the states of components involved in interactions [20, p. 4].
Selecting the appropriate set of invariants and determining the heuristics for computing invariants (such as interaction invariants) remains as one of the main challenges when designing a compositional verification technique, and is a topic of ongoing research [20]. Relevantly for swarm systems, *"while compositional verification alone cannot guarantee complete correctness of an adaptive system, it can prove such things as deadlock-freedom and* ***overall mission-goal reachability****"* [20, p, 4].

In this context, the *AdaptiV* method can potentially be a powerful tool for verifying swarm space exploration systems that use reinforcement learning (and hence can be adaptive), since the proposed analysis can be performed (even for systems consisting of ANNs) and can help monitor and assess the mission-reaching capabilities of the swarm, as well as identify the components of the system that are critical to achieving such mission.

3.2.4. Conclusion

This section defined what is meant in this research by "swarm", and the importance of having cooperation. Furthermore, NASA's architecture to model complex multi-agent systems has been examined, highlighting the division between abstraction tasks in the swarm. Moreover, different information structures are explained; spanning from centralized to fully decentralized settings, as well as the information structure in the CADRE mission. Importantly, NASA's *AdaptiV* method is discussed, which could further help verify the swarm system to be designed in this research.

3.3. Reinforcement Learning

For the purpose of this research, reinforcement learning (RL) is the third main building block (space exploration and swarming have been the other two) to contribute towards the field of swarm space exploration. RL will serve as the main tool for learning or discovering swarm behaviors and/or making the swarm adaptive to its environment. To know where and how to implement RL into a swarm space exploration scenario, several steps are taken.

Firstly, the single-agent RL scenario is discussed in Section 3.3.1, providing a historical context, as well as describing the fundamentals of RL by mainly using explanations from the book *"Reinforcement Learning: An Introduction"* by Richard S. Sutton and Andrew G. Barto [6], to provide a solid foundation. Then, this classic definition is extended into the multi-agent RL (MARL) scenario in Section 3.3.2; investigating the motivation of using MARL, as well as its top-level configurations and challenges, and providing a historical overview combined with the investigation of state-of-the-art MARL elements and algorithms relevant for this research. Moreover, Section 3.3.5 focuses on MARL specifically applied to swarm space exploration, highlighting the research history in this field and the challenges that lie ahead. Finally, the findings of this section are discussed in Section 3.3.7, contributing to Research Questions 1 and 3.

3.3.1. Single-Agent Reinforcement Learning

Single-agent reinforcement learning (RL) stands as a cornerstone in the evolution of artificial intelligence and autonomous systems (such as space robot swarms). Its historical journey spans several decades, witnessing remarkable advancements and milestones. This section provides a exploration of single-agent RL, dissecting its key components and methodologies. Firstly Section 3.3.1 traces the historical trajectory of RL, from its early roots to its current prominence in solving complex decision-making tasks. This subsection then delves into the essential elements of RL in Section 3.3.1; including the concept of policies, rewards, and value functions. Afterward, Section 3.3.1 underlies the fundamental algorithms that empower agents to learn, adapt, and optimize their actions. This will help pave the way for a deeper understanding of not only single-agent RL but also its role as a foundational framework for the domain of multi-agent reinforcement learning.

Historical Overview

Over the past half-century, RL has been evolving from being applied to simple control tasks, which were limited to discrete state and action spaces, to complex control scenarios. The RL timeline is shown in Figure 3.6, where it is observed that in the past decade, RL has greatly evolved in terms of developed algorithms, and benefited from innovation in supervised learning methods, etc; enhancing its applicability from games to robotics, etc.

In this way, RL is becoming a widespread method of solving very challenging problems; from achieving super-human performance in board games such as GO to solving the protein folding problem [1]. As will be shown in this section, RL can help find solutions to problems without necessarily needing human input about the problem except for a reward signal based on the desired result. This makes RL to be a powerful tool for solving complex problems. As will be shown in the following subsections, it can be applied to swarms.

[1] www.deepmind.com/research/highlighted-research/alphafold

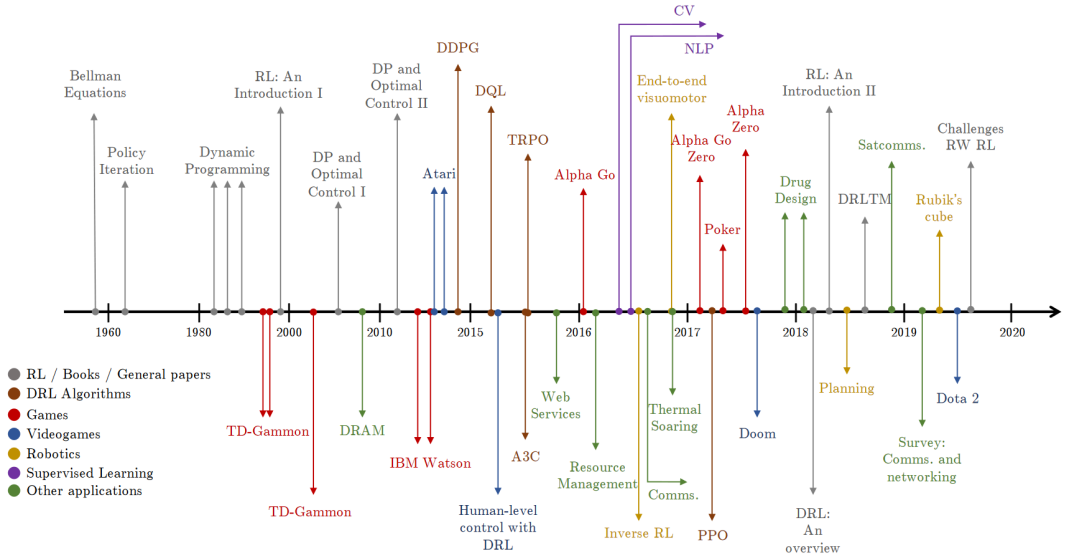


Figure 3.6: Timeline of deep reinforcement learning research up to 2020 [21].

Moreover, RL is also been increasingly used to solve real-world problems (such as in this research), where a vast amount of investigation is been done, as it can be seen in Figure 3.7.

Critically salient are the advances in multi-agent settings, now using concepts such as centralized critic, decentralized actor methods, hybrid mechanisms, etc [21]. The multi-agent challenges can also use the innovation from the other approaches, for example, population-based methods can benefit multi-agent RL, and the RL field as a whole can benefit from improvements in embeddings, masking, neural network architectures etc.

Importantly, RL's ability to obtain control policies for complex environments is also been leveraged in the aerospace sector, such as using it to find control policies to explore planetary bodies [22], and develop multi-agent cognitive networking systems for the lunar environment [23], to name but a few applications.

Approach	Description	Examples
AutoML	An outer loop learner changes meta parameters to better adapt to the challenge	Meta learning for offline DRL, meta learning for multi-objective reward, Meta RL for generalizability, replacing action maximization by neural network search in high-dimensional spaces
Mathematical guarantees	Derive equations and theorems that support the challenge fulfillment	Lyapunov functions and primal-dual methods for safe DRL, assume uncertainty matrices to address non-stationarity
Neural network architectures	Rely on Deep Learning advances to increase robustness against the challenge	RNNs for partial observability, attention mechanisms for credit assignment, network ensembles for offline DRL
RL theory	Adapt theoretical RL frameworks to DRL settings and the use of neural networks	Off-policy algorithms, POMDPs, Delay-Aware MDPs, Constrained MDPs, Maximum-entropy RL
Embeddings and latent spaces	Address challenge problems by relying on robust intermediate embeddings	High- to low-dimensional embeddings, latent variables for non-stationarity, multimodal and contrastive learning-based representations, unsupervised learning
Reward estimation or modification	Try to overcome the challenge by directly modifying the reward structure and/or function	Reward shaping in long trajectories, reward shaping for safe DRL, reward redistribution, distributional RL against stochasticity
Deriving a model	Instead of learning a policy, learn models of the environment and use them to plan	Model-based RL, imitation learning, inverse RL
Pruning and masking	The learning process involves deciding, among different learning signals, how important each of them is and eliminating the unnecessary ones	Batch-constrained methods for offline DRL, distributed training in large-scale settings, action elimination in high-dimensional spaces, divide-and-conquer methods in stochastic environments
Use auxiliary tasks	Provide the agent with auxiliary tasks that, combined, increase robustness against the challenge	Multitask learning and online learning for data efficiency, self-supervised learning for unspecified reward, hierarchical RL in long trajectories
Data augmentation	Rely on different data augmentation and data wrangling techniques	Randomization to transfer better, data augmentation to address non-stationarity and stochasticity
Heuristics	Use human-crafted rules or processes to address the challenge	Scalarization of multiple objectives, hyperparameter tuning
Population-based methods	Have multiple agents with slightly different parameters/objectives and search for the best ones	Multi-agent populations in partially observable environments, multi-objective populations
Multi-agent specific	Solutions specific to the multi-agent challenge that can not be mapped to other challenges	Independent Q-learning, decentraliced actors and centralized critic, hybrid mechanisms

Figure 3.7: Summary of the different approaches that have been considered by the RL community to address the specific challenges of real-world RL. Each approach has been considered for different challenges independently. Retrieved from [21, p. 47].

The Reinforcement Learning Problem

Learning by interacting with an environment can be arguably regarded as one of the most natural ways of learning [6, p. 2]. It can be observed in nature, for example, when an infant is trying to learn how to walk it may not have an explicit teacher, but by **interacting** with the environment (with which it has a sensorimotor connection) it learns how to stand up, and eventually move. This occurs in an iterated fashion, with the infant trying to stand up and balance itself in different ways, and **observing** how its actions influence how well it can reach the goal of walking.
In this fashion, RL is the *computational* approach to mimic such learning by interaction that is often found in nature [6, p. 2].

RL is a subfield of machine learning, that does not fully lie in the supervised or unsupervised learning category. Supervised learning requires a set of labeled data examples that are used for training as well as assessing the extrapolation/generalization capabilities of the ML model. In interactive scenarios, it is often impractical to have such labeled data [6, p. 2]. On the other hand, unsupervised learning usually

deals with finding structures that are hidden in data that is unlabelled [6, p. 2]. This is also different from RL, because, even though *"uncovering structure in an agent's experience can certainly be useful in reinforcement learning* (this learning paradigm) *by itself does not address the reinforcement learning agent's problem of maximizing a reward signal"*[6, p. 3].

As stated in the previous paragraphs, in RL, an agent interacts with an environment with the objective of maximising a numerical reward function. In this sense, the agent is able to observe the environment, and take actions that will change the state of this environment. The state of the environment affects the value of the reward function.

Moreover, the agent needs to do a trade-off between exploiting what is already known and exploring in order to discover better actions or to make better action selections in the future [6, p. 3].

Elements of Reinforcement Learning
With this conceptual introduction, the different building blocks of RL become more apparent. This sub-subsection provides a more in-depth look at the different RL elements. Apart from the agent and the environment, generally, there are four main sub-elements of an RL system:

1. **Policy**.
2. **Reward signal**.
3. **Value function**.
4. **Model of the environment**.

The **policy** is the mapping from the environment states to the actions to be taken in those states [6, p. 7]. *"In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic"* [6, p. 7]. Specifically, a policy π is a mapping from each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(a|s)$ of taking action a when in state s [6, p. 70].

The **reward signal** defines the goal in the RL problem. The environment sends the agent a reward signal depending on the state achieved by the agent and the current action that is taken. The goal of the agent is to maximize the total reward. In other words *"The reward function defines what an agent should do, and a reinforcement learning algorithm determines how to do it"* [24]. The reward function can be sparse, meaning that it is only given at the end of a training episode or only after achieving a certain state and action, it can also be sent at each time-step and state, or something in between these two approaches.

The **value function** indicates the total amount of reward an agent can expect in the future [6, p. 7]. *"Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states"* [6, p. 7]. With the value function, the idea is that the agent pursues actions that would achieve the highest value, not the highest immediate reward. As stated by Sutton and Barto, *"The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades"* [6, p. 8]. Particularly, for Markov Decision Processes (MDPs) the value of a state under a policy π is defined in Equation 3.1, where the discount factor γ is a number between 0 and 1 used to give more or less importance to future rewards R [6, p. 70].

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{3.1}$$

Furthermore, the value of taking an action a on a certain state s and following a given policy π can also be quantified and is defined by the **action-value function** of π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{3.2}$$

Both v_π and q_π can be estimated through experience, and are Monte Carlo estimation methods since they involve averaging over random samples of returns [6]. Crucially for this work, if there are many states or actions (or both spaces become continuous), it starts becoming unfeasible to keep a specific value for each state or state-action pair (due to memory constraints), and thus **function approximators** are investigated to model the value and policy functions (swarm systems have a large number of states and the action spaces, and in the context of planetary exploration, these spaces tend to be continuous).

Finally, a **model of the environment** allows for inferring how the environment will evolve in the future and thus is used for planning. As an example; for a given state and action, the model of the environment can predict the next state and reward [6, p. 8]. RL algorithms that use models of the environment are called *model-based*, whereas RL algorithms not using such models are referred to as *model-free* [6, p. 8].

Markov Decision Processes

With the identified elements of RL, it is important to understand the framework in which the algorithms function, and importantly, what are the required assumptions. This will be essential, since for the multi-agent case, several of these assumptions are violated, thus severely increasing the difficulty of the RL problem (see Section 3.3.2).

As previously discussed (Section 3.3.1), in the RL framework, the agent makes decisions as a function of the *state* it is in, which can be seen as a signal from the environment [6, p. 62]. It is thus crucial to understand *how* this state affects learning.

Although the state signal should include immediate sensations such as sensory input, there is no reason to restrict the signal to immediate sensor readings; rich state representations are also possible [6, p. 63]. Moreover, *"the state signal should not be expected to inform the agent of everything about the environment, or even everything that would be useful to it in making decisions"* [6, p. 63], having hidden state information in the environment is thus possible.

Ideally, the state signals should summarise **past** sensations compactly, but in such a way that **all** relevant information is retained [6, p. 63]. This usually **requires more** than the immediate sensations, but never more than the whole history of past sensations [6, p. 63]. Such as a signal is defined as having ***the Markov property***. Here, the Markov property is defined for a finite number of states and rewards, but this definition can be extended to the infinite scenario replacing the sum operations with integrals, and the probabilities with probability densities. In the most general environment, the dynamics are fully described by the complete probability distribution from the beginning time until the present:

$$\mathbb{P}\left[\{\right] R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \tag{3.3}$$

If the state signal has a Markov property, the environment's dynamics can be simplified to:

$$p(s', r \mid s, a) = \mathbb{P}\left[\{\right] R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\} \tag{3.4}$$

Here, the next-step prediction and expected reward can be calculated having only the **current** state and action [6, p. 64]. Having such a Markov property allows for proving certain successful problem-learning criteria, and the fundamental theory of RL is based on this assumption [6]. As it will be shown in Section 3.3.2, the algorithms relevant to solving the swarm space exploration problem tend to violate this assumption.

The Learning Process

As stated at the beginning of this section, RL algorithms learn by interacting with their environment. Learning-by-interaction allows for **adaptability** and thus can have potential benefits in an exploratory context. This subsection gives a formal definition of learning in RL.

For the purpose of this study, RL algorithms generally have to work in continuous state and action spaces. As mentioned in Section 3.3.1, this makes the usage of tabular methods to store the policy and value functions' information unfeasible due to memory constraints. Instead, the policy and value functions are approximated through function approximators; functions consisting of a fixed number of parameters or constituent parts that model the policy or value functions in a continuous (non-discrete) manner. There are different types of function approximators, such as fuzzy functions, splines, multivariate regressors,

interpolated tabular functions, artificial neural networks (ANN) etc that can be updated on their parameters or constituent parts to model the policy or value functions. It is important to realize that these functions need to be able to learn from incrementally acquired data, and that most sophisticated ANNs and statistical methods rely on the assumption of static training, and can be hindered by non-stationary environments, which are present in multi-agent RL [6, p. 227], as described in Section 3.3.2.

Before defining the function update algorithm(s), it is important to define the different possibilities of **when** to update; such as at the end of a learning episode, at each taken step in that episode, or something in between.
In the first category lies Monte Carlo (MC) methods for episodic tasks, which require the completion of a learning episode to update the policy and value functions [6, p. 113]. Temporal difference (TD or TD(0)) learning, on the contrary, can be updated at each step in the learning episode. Equation 3.5 shows the every-visit MC update, (suitable for nonstationary environments) where V is the approximate value function which can be updated with the actual return G_t following time t, and α is a constant step-size parameter [6]. Crucially, G_t is calculated at the **end** of the episode, since only then is the actual return known.

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left[G_t - V\left(S_t\right)\right] \tag{3.5}$$

G_t is the difference then between MC and TD(0), since instead of waiting for the completion of the learning episode to obtain the actual return and update the value estimate, TD(0) updates the value estimate at the next time-step $t+1$ by using the observed reward R_{t+1} and the value **estimate** $V(S_{t+1})$ (thus making TD model-free). This is shown in Equation 3.6, where γ is the discount factor. Because TD methods use estimates of the value instead of the actual one, they are called **bootstrapping** methods. It should be noted that, in general, TD methods are more efficient than their Monte Carlo counterparts [6], and have the advantage of not having to wait until the end of the training episode, something that might become crucial for planetary exploration when learning online during the mission. Furthermore, *"for any fixed policy π, the TD algorithm described above has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to usual stochastic approximation conditions"* [6, p. 148], moreover *"on tasks with large state spaces, TD methods may be the only feasible way of approximating the certainty equivalence solution"* [6, p. 154], however, they also make the values of training examples to become non-stationary [6, p. 227].

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left[R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right] \tag{3.6}$$

In between the MC and TD(0) methods lies TD(λ). Such methods use the so-called eligibility traces to obtain a more general method that learns more efficiently [6]. These methods can be updated requiring a different number of steps during the training episode. The spectrum of all the aforementioned methods is schematized in Figure 3.8.

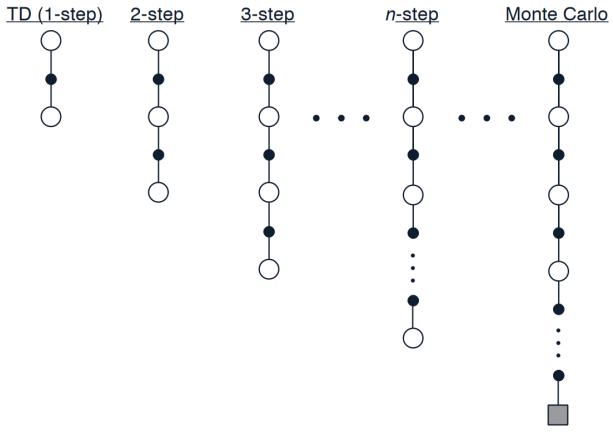


Figure 3.8: Schematic of different value update schemes, ranging from TD(0) all the way to MC, which needs the completion of the learning episode. Retrieved from [6, p. 169].

Lastly, it must be mentioned that dynamic programming (DP) approaches are also possible, however, are not considered for this work due to their imperative need to have a perfect model of the environment as an MDP (an assumption that is violated for multi-agent systems, as discussed in Section 3.3.2) and their great computational expense [6, p. 89].

Among the learning methods not requiring a model of the environment, there are different approaches when updating the policy function; on-policy and off-policy algorithms. **On-policy** algorithms update on the **current** policy used during the training episode, whereas **off-policy** algorithms can update on a policy function different from the one being used during the learning episode. In this manner, on-policy methods learn the value of the current policy used on the environment, whereas off-policy methods attempt to learn the value of the optimal policy.

Furthermore, when having estimates of the action values (with a q function), then, during learning, there is an action whose estimated value is greater than the others. Selecting this action is defined as taking a **greedy** action, and is understood as **exploiting** the current knowledge learned about the estimation of the action values. On the contrary, if an action is selected such that it doesn't have the greatest estimated value, it is called an **exploration** action. Exploitation allows for maximizing the expected reward in one learning step, whereas exploration can produce a greater total reward as learning progresses [6, p. 32]. In practice, exploration and exploitation can be alternated, for instance, by having a small probability ϵ where a random action is selected for exploration, and a probability $1 - \epsilon$ of selecting the (greedy) action with the maximum estimated value [6, p. 34]. Such an approach is referred to as ϵ**-greedy**.

When discussing how to train the RL algorithm, this can be done on-line and off-line. During **on-line** learning, the agent learns by directly interacting with the environment, whereas in the **off-line** scenario the agent uses data previously collected from other agents and does not directly interact with the environment (the dataset is collected once, and not altered during training) [24]. In this work, only on-line RL is considered, given the lack of an explicit dataset for swarm planetary exploration, and the fact that when training on a simulated environment (in the case where such an environment can also be used to generate the dataset), on-line learning on such a simulation and then transferring the learned policy to the real environment tends to be a *"pragmatic and often effective"* approach [24].

Finally, there are myriad possible RL architectures to guide the learning of a successful control policy. Here, **actor-critic** is described, given its popularity and relevance for this research, as the majority of MARL algorithms discussed in Section 3.3.4 and Section 3.4 make use of it in some fundamental form. Actor-critic methods are TD methods where the policy and value functions are represented independently [6, p. 257]. The policy is known as the *actor* since it takes actions that are then criticized by the value function (which is usually a state-value function); the *critic*. The learning is always **on-policy** since the critic learns from the policy that is currently being followed by the actor [6, p. 257]. The critic takes the form of a TD error, outputting a scalar signal used to update the actor:

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V(S_t) \tag{3.7}$$

Where V_t is the value function implemented by the critic at a given time t [6, p. 258]. *"This TD error can be used to evaluate the action just selected, the action A_t taken in state S_t. If the TD error is positive, it suggests that the tendency to select A_t should be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened"* [6, p. 258]. Moreover, two strengths of actor-critic are the minimal computational requirements needed to select actions, as well as its ability to learn stochastic policies (selecting the optimal probabilities of selecting various actions) [6, p. 259], something that is useful on non-Markov cases (as is the case of multi-agent RL, see Section 3.3.2). The top-level actor-critic architecture is shown in Figure 3.9.

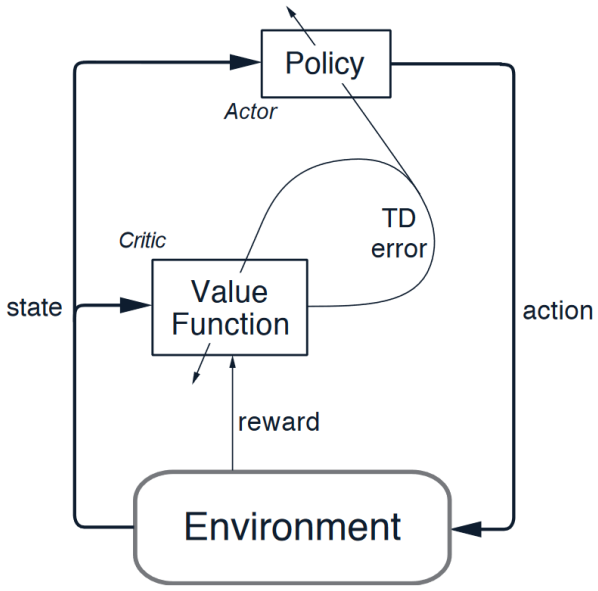


Figure 3.9: The actor-critic architecture. Retrieved from [6, p. 258].

3.3.2. Multi-Agent Reinforcement Learning (MARL)

Multi-agent reinforcement learning (MARL) applies the aforementioned schemes of RL to systems with several agents. Multi-agent systems consist of a group of autonomous, interactive entities that all share a common environment, which they perceive with sensors and in which they can act through actuators. *"Although the agents in a multi-agent system can be endowed with behaviors designed in advance, they often need to learn new behaviors online, such that the performance of the agent or of the whole multi-agent system gradually improves. This is usually because the complexity of the environment makes the a priori design of good agent behaviors difficult or even impossible. Moreover, in an environment that changes over time, a hardwired behavior may become unappropriate"* [25, p. 2]. In this context, RL has the potential to allow for a multi-agent system that can adapt to a changing environment. This work focuses on multi-agent **cooperative** tasks since they are relevant for space exploration, as shown by the different TRLs discussed in Section 3.1.2.
Additionally, there can be different goals in the RL scheme; such as the stability of the agent's learned dynamics (i.e. each agent converging to a stationary policy), and their adaptability to the changes in the environment (also induced by the changing behavior over time of the other agents, as learning progresses) [25, p. 2, 3]. It is important to realize that in the multi-agent case, the environment becomes **non-stationary** due to the aforementioned conditions, and *"each agent is therefore faced with a moving-target learning problem: the best policy changes as the other agents' policies change"* [25, p. 9].

Indeed it is this balance between **stability** and **adaptability** that becomes crucial for MARL, and is investigated in the following paragraphs.
Convergence to equilibria during learning is a basic stability requirement, meaning that the agent's strategies converge to a coordinated equilibrium [25, p. 10]. In this context, Nash equilibrium has been used in the field. However, Nash equilibrium is not always practical if the agents are not fully rational and convergent, and the performance in dynamic stochastic games is unclear [25, p. 10]. An agent is defined as **rational** if it can learn models of other agents and maximize its expected return given its models of the other agents [25, p. 11]. Having a MARL scheme with such agents results in **opponent-aware learning**, related to adaptation; the agents reacting in some form of best-response to the behavior of other agents. On the contrary, if agents do not learn models of other agents, this results in **opponent-independent learning**, related to stability. Opponent-independent learning converges to an equilibrium strategy (that is part of an equilibrium solution) regardless of what the other agents are doing [25, p. 9]. During learning, it is important to achieve both stability and adaptation within some acceptable bounds.

Moreover, there can be several MARL configurations, discussed in the following paragraphs. Regarding value functions, there can be centralized, decentralized, and mixed approaches. **Centralised value**

functions can have global state and action space information, whereas **decentralized value functions** only have access to local state and action spaces. Using these approaches has several consequences. Firstly, if a centralized value function is combined with a centralized policy, *"the task would reduce to a Markov decision process, the action space of which would be the joint action space of the stochastic game"* [25] (although if agents use greedy actions, this can lead to suboptimal joint actions, as discussed in [26]). However, this comes at the cost of having an input space that grows linearly with the number of agents, thus poorly scaling to systems with a large number of agents [27]. This is referred to as *"the curse of dimensionality"* [26], one of the main problems of MARL, and several approaches are suggested to combat this issue; such as using mean field theory (see Section 3.3.2), embedding the large global state space into a smaller latent space, etc [26, 27]. Decentralized value functions thus can help solve this issue, however, they can result in poor (unstable) learning; *"one issue is that each agent's policy changes during training, resulting in a non-stationary environment and preventing the naïve application of experience replay"* [28].

These different MARL designs, and configurations discussed in the rest of the section, need to be evaluated to satisfy the needs of swarm space exploration, as discussed in Section 3.5.

Historical Overview

MARL has been evolving over the last decades. The fundamental design of MARL algorithms considers aspects from TD methods, Game Theory, and Direct Policy Search [25, p. 13]. In the past, MARL has been focused on simple games in discrete state and action spaces. As time has progressed, other breakthroughs in the field of artificial intelligence, such as the use of deep learning techniques have been incorporated in the field. Relevant was the year 2019 (see Figure 3.10), where in a single year MARL showed several successful results; either achieving super-human performance, or grand master-level skills in games as complex as StarCraftII (10^{26} possible choices per move, with limited information about the game state), playing hide and seek strategies, etc. All these scenarios dealt with complicated problems that were thought to be near impossible with artificial intelligence [29], but that have been solved with MARL. Moreover, in several of these scenarios, the emergent behaviors induced by the MARL methods could be understood by humans and are grounded in physical theory [29]. This suggests that MARL can help also understand the solutions to complex problems.

There are still several problems to be solved in the MARL field, such as dealing with poor agent scalability, non-stationarity in the environment, robustness etc. However, techniques in the field of artificial intelligence are helping solve these problems, such as using deep learning architectures, where *"function approximation is vital in MARL, which necessitates the development of policy-based algorithms"* [19].

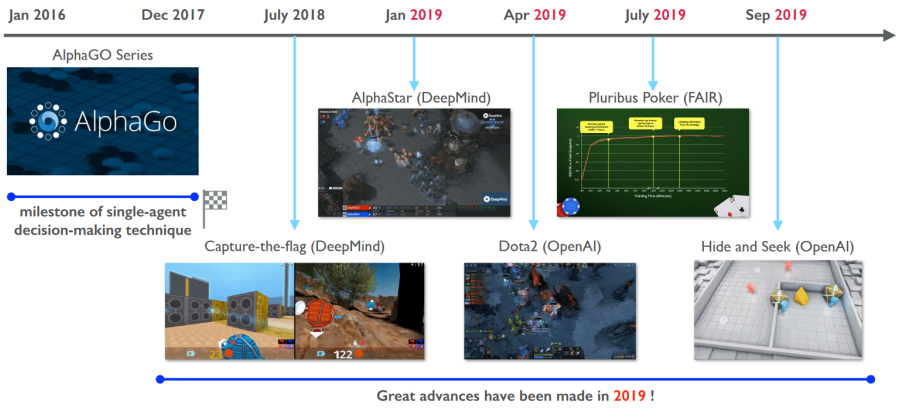


Figure 3.10: Timeline of the booming 2019 year for MARL. Retrieved from [29].

Markov Decision Processes in MARL

When previously discussing the basic building blocks of RL, Section 3.3.1 discussed that having Markov properties is the underlying assumption to develop the fundamentals of RL theory. One of the crucial elements of Markov systems is the state signal. The state signal should include immediate measurements such as sensory readings, however, state representations can also be highly processed versions of the sensory readings, which can be combined with other additional information, as well as past information,

embedding strategies, etc. This state representation is of crucial importance for MARL algorithms, and it is currently been investigated in the field through different state-of-the-art approaches (Section 3.3.4).

Having different state representations results in different Markov games in MARL. Here the most relevant cases are discussed. Notice that in this research, only cooperative settings are considered, as specified in Section 3.2.

One of the most simple scenarios involves having homogeneous agents (also having similar rewards), where convergence to an optimal/equilibrium Q-function has already been established in past research [19, p. 16]. This is done by *"performing the standard Q-learning update at each agent, but taking the max over the joint action space* $a' \in A$*"* [19, p. 16]. However, the convergence of the Q-function does not necessarily signify that an equilibrium policy is found in the Markov setting, since if the equilibrium policies are non-unique, the agents can fail to agree on which policy to select, hence a successful equilibrium policy (convergence to the Nash Equilibrium policy) is only guaranteed if the equilibrium is assumed to be unique [19, p. 16], or if the agents are coordinated for equilibrium selection.

Furthermore, MARL settings can also consider the so-called *mean field* regime; where there is an extremely large number of homogeneous agents, each having a very small impact in the overall multi-agent system [19, p. 18]. This results in all agents being indistinguishable from each other (this can, in a simplified manner, be compared with a swarm of ants, where each ant has little influence over the general state of the swarm). In this setting, the interaction with other agents can be represented by some mean-field metric [19, p. 18], some average state, or empirical distribution of states. **This considerably simplifies the analysis, since each agent needs to find the best response to the mean-field** [19, p. 18]. MARL in these systems has not been explored until recently, but mostly in non-cooperative settings [19, p. 18].

Another approach entails having heterogeneous agents which may have different rewards, and that form a team that tries to maximize a *team-average* reward [19, p. 19]. This approach applies to more realistic scenarios, for example, it would allow different robots with different local objectives to achieve the common goal of satisfying a space exploration mission according to some criteria that can be quantified as a reward. One solution to this scenario is to use some form of centralized controller that can collect the average rewards, and distribute information to all agents [19, p. 19], which can then guarantee a Markov decision process. However, such approaches can not usually be applied in realistic scenarios due to poor cost, robustness, and scalability properties [19, p. 18] (the poor scalability properties of this method are experimentally shown in Section 3.4).

MARL for decentralized networked agents (see image b in Figure 3.5) has also been investigated [19, p. 21], where each agent can be modeled to deal with an *independent* MDP not affected by other agents. Distributed actor-critic algorithms have been considered for this purpose, and convergence can be guaranteed for linear function approximators [19, p. 21] (which are generally not applicable to this research, see Section 3.3.3).

Lastly, and extremely relevant for this research, there is the coordination scenario where agents have only partial observability of the state. Although they can be common in practice, ***"theoretical analysis of MARL algorithms in this setting is still relatively scarce, in contrast to the aforementioned fully observed settings"*** [19, p. 24]. These systems can be modeled by decentralized partially-observable Markov decision processes (Dec-POMDPs), and are notoriously difficult to solve. Notably, ***"Dec-POMDPs have been known to be NEXP-complete, requiring super-exponential time to solve in the worst case"*** [30, 19]. For Dec-POMDPs there is the trend for using *centralised-learning-decentralised-execution* [19], and thus the majority of the state-of-the-art MARL algorithms described in Section 3.3.4 have this configuration. The overview of the different Markov decision processes that can be applied to systems relevant to this research are shown in Figure 3.11.

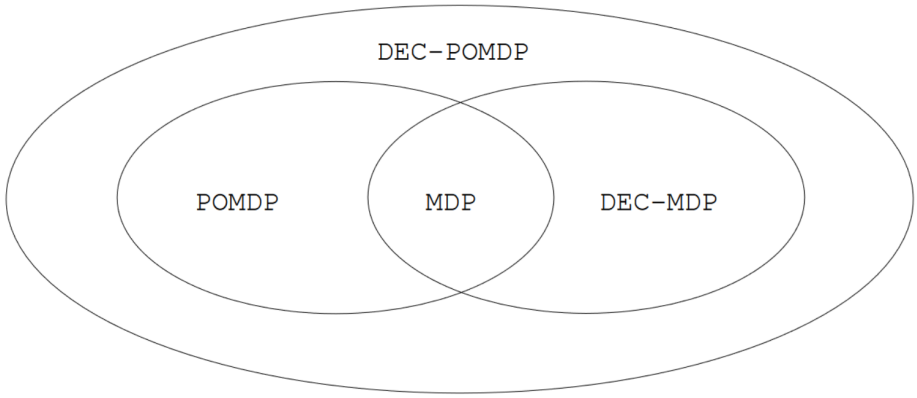


Figure 3.11: Relationship among the different relevant Markov models. Retrieved from [30].

3.3.3. MARL Function Approximators

As discussed in Section 3.3.1, the swarm space exploration scenario deals with continuous action and observation spaces, hence, instead of estimating the policy and value functions with tabular methods, continuous function approximators are used instead. *"Function approximation is vital in MARL, which necessitates the development of policy-based algorithms"* [19]. **In particular, deep learning and fuzzy logic methods are investigated** given the successes in the field of ML related to the successful approximation of complex functions by the former, and the resilient characteristics of the latter.

Deep Reinforcement Learning for Multi-Agent Systems

Deep MARL methods have been successfully applied to multi-agent systems [27]. Such methods typically involve the usage of neural networks consisting of several layers to approximate both the policy and value functions. Here, the neural network function approximators are discussed.

Artificial Neural Networks

Neural networks (ANNs) can have different architectures; ranging from direct input-output mapping to local feature extraction, time-dependencies, etc. Here, Feed-Forward, Convolutional, and Recurrent neural networks are discussed in particular, given their practical application to MARL problems, embeddings, and variable input characteristics (for some configurations). Moreover, the explanations are retrieved from the book *"Deep Learning with Python"* by F. Chollet [31], and the author of this research in [32, p. 10-12]. The following definitions are thus taken from [32].

Feed forward neural networks (FF) (see Figure 3.12) are used to map an input to an output. This mapping is achieved by using three main elements; weights w and biases b (the trainable parameters of the network), and the so-called activation functions (which need to be specified beforehand, and can be non-linear), shown as dots in Figure 3.12. Furthermore, a FF network can consist of several layers; each with its own sets of weights, biases, and activation functions. Each layer maps an input vector x into an output vector y by performing a dot product between the weights tensor W and the input x, and then adding the bias term: $y = W \cdot x + b$. This in itself induces a linear transformation of the input. In this context, the activation functions are used as operators that, if desired, can add non-linearities, or modify the mapping $x \rightarrow y$ in a certain way. Thus, for example, if the `ReLu` activation function is used, the mapping is as follows $y = \texttt{ReLu}(W \cdot x + b)$.
Different layers can be stacked (the output of one layer becomes the input of the next) in the network to achieve increasingly complex mappings. Furthermore, the "learning" procedure consists on finding the correct weights and biases to realize a certain mapping [31]. This is achieved by using the gradient of the loss (error), whose metric needs to be defined (mean squared error, for example). This loss gradient is then related to each of the trainable parameters (weights and biases) by using the derivatives chain rule [31].

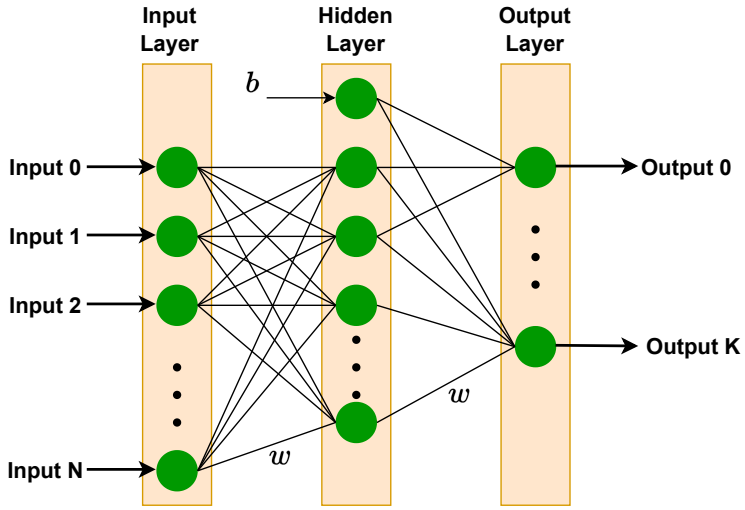


Figure 3.12: Feed forward neural network with one hidden layer, with example bias b and weights w shown. Inspired from [32].

While the FF network learns global patterns in the input spectrum, the **Convolutional neural network** (CNN) learns local patterns [31]. These learned patterns are translation invariant [31]. Furthermore, the spatial hierarchies of patterns are also learned [31]. The different CNN elements are shown in Figure 12. For a 3D input image, a convolution operation is done over the data. Here, a sliding window (kernel) of predefined size stops at every possible location in the data and extracts a 2D patch (window length, input depth) of the surrounding features [31]. The number of features (the filters in Figure 3.13) is also predefined. After this, a max-pooling operation is used *"to aggressively down sample feature maps, much like strided convolutions. Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel. It's conceptually similar to convolution, except that instead of transforming local patches via a learned linear transformation (the convolution kernel), they're transformed via a hard-coded max tensor operation"* [31, p. 209]. Convolution and max-pooling layers can be stacked to extract higher-level features from the data. Finally, the data from the last max-pooling layer is flattened into a 1D array which can then be connected to dense layers (much like the FF model layers) to finally generate the outputs.

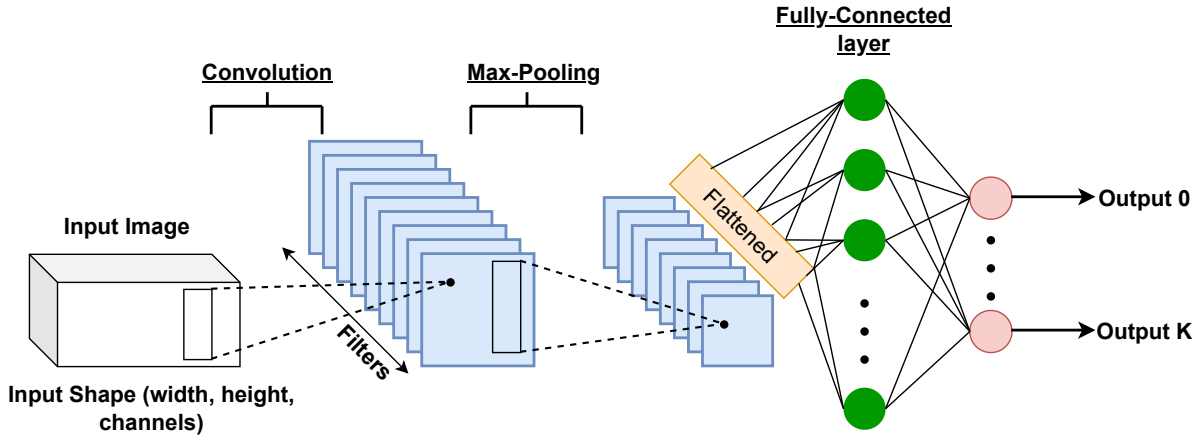


Figure 3.13: Example Convolutional neural network architecture, with an image as an input. Inspired from [32].

Furthermore, when there is valuable information on the temporal order of the input data (for example, when reconstructing a trajectory of the state-space), **Recurrent neural networks** (RNNs) can be used. These networks can be adapted to variable temporal input sizes and can output at every time step, or at

the last time step, for example (these configurations are referred to as many-to-many, and many-to-one, respectively). ”Moreover, classic RNNs have the problem of catastrophically forgetting information from the past, as sometimes the gradient used to update the network (i.e. learning) vanishes or increases to a very large number, over time” [32, p. 11-12]. With this consideration, this research considers Long Short Term Memory cells (LSTM), a specific RNN architecture that improves on the forgetting shortcoming. *”LSTM cells work by recursively transmitting two channels of information over the input; the cell state C and the self state H, shown in Figure 3.14. The cell state C can be seen as the long-term memory component of the cell, while H, the self-state, is the short-term memory component. The cell state C transmits information down the entire time-series input chain, having only some minor linear interactions, and short-term memory modifications are possible, but controlled by information gates who forget, store, and update information. Furthermore, the H cells carry the more recent information, and directly concatenate with the input at each time step”* [32, p. 12].

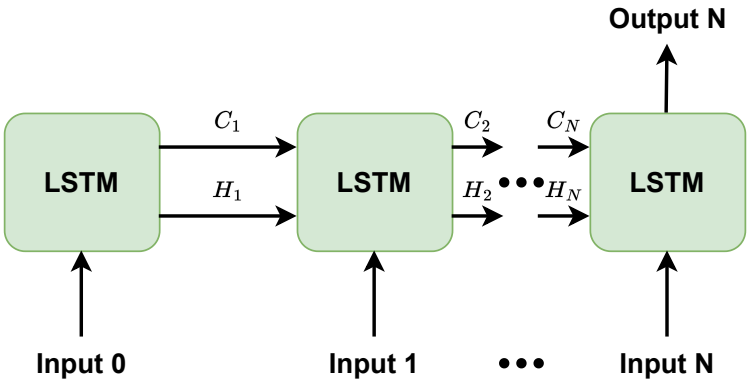


Figure 3.14: LSTM neural network architecture, in a many-to-one configuration. Inspired from [32].

Fuzzy Logic Reinforcement Learning for Multi-Agent Systems

Fuzzy logic is a mathematical framework that allows for the representation and processing of uncertainty and imprecision in decision-making and control systems. Unlike traditional binary logic, which deals with crisp true/false values, fuzzy logic deals with degrees of truth or membership in a set. It employs linguistic variables and membership functions to capture and quantify vague, qualitative information. Fuzzy logic provides a way to model and handle uncertainty, making it possible to create systems that can make approximate, nuanced decisions based on imprecise data.

Fuzzy logic MARL has also been investigated and might be useful for this research in case of a need to pivot from the aforementioned ANNs. Here, fuzzy-logic RL is briefly described. Fuzzy logic has gained widespread acceptance due to its capacity to integrate human-like expertise for managing intricate systems without relying on a predefined model. This is achieved by describing the acquired understanding of the system’s behavior through linguistic relationships. Moreover, fuzzy logic has been applied for the identification and control of nonlinear dynamic systems [33].

Importantly, fuzzy strategies are applied to guide mobile robots following a leader and preventing collisions with obstacles [33]. Additionally, a collision avoidance method utilizing an extended Takagi-Sugeno-Kang inference system has been implemented for a flock system [33]. In this implementation, the individual agent separation of the flock is realized through *”a zero-order Tagaki-Sugeno (TS) fuzzy logic inference engine with an online adaptation capability based on an actor-critic scheme”* [33]. The value of this adaptive fuzzy-RL scheme lies in its ability to approximate highly nonlinear systems [33].

To achieve agent separation [33], a set of fuzzy membership functions needs to be optimized. To optimize and adjust the consequent membership functions for each rule in a fuzzy logic system combined with RL, the effectiveness of the chosen actions using a value function are evaluated. This value function is estimated through a critic neural network, **where the output is determined by summing the products of certain parameters ($\Psi_{ij,\zeta}(p)$) and critic weights ($\Phi_{ij,\zeta}(p)$), each corresponding to a specific rule** [33]. The resulting control method offers each agent an immediate collision avoidance mechanism, eliminating the necessity for extensive offline training sessions before making the correct decisions [33].

3.3.4. Overview of Recent MARL Algorithms

This section discusses three state-of-the-art MARL algorithms with potential applications or characteristics that can be used for swarm space exploration. Firstly, Section 3.3.4 describes the MADDPG algorithm, one of the important breakthroughs in combining MARL with deep learning, then, the MAAC algorithm, which modifies the action-value function of MADDPG, is described in Section 3.3.4, followed by the MAPPO algorithm in Section 3.3.4. Afterward, a different approach is considered utilising hierarchical RL described in Section 3.3.4, followed by FLDDPG, which aims at achieving learning robustness in scenarios with unstable communications. The suitability of these algorithms is also discussed at the end of each subsection.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm deals with a centralized action-value function (critic) and a decentralized policy function (actor), where both functions work in continuous state and action spaces, and can be approximated with ANNs. In this way, the policies always take their local observations as input, meaning that no modifications are needed between training and execution. In contrast, each agent uses a critic function $Q_i^\pi(x, a_1, \ldots, a_i, \ldots, a_N)$ that takes as input the actions of all agents, as well as some state information, and outputs the Q-value for that specific agent [28]. This is illustrated in Figure 3.15.

Since each Q_i^π is learned separately, each agent can have its own reward structure, thus allowing for competitive settings with mixed rewards, cooperation settings with aligned rewards, and, relevant for planetary exploration, it allows for heterogeneous agents (using such an algorithm on a swarm with, for example, both rovers and drones, is possible).

A key feature of MADDPG is that, if the actions of all other agents are known, the environment becomes **stationary**, since $P(s'|s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s'|s, a_1, \ldots, a_N) = P(s'|s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N)$ for any $\pi_i \neq \pi'_i$ [28], and this centralized-critic with deterministic policies set-up works very well in practice [28].

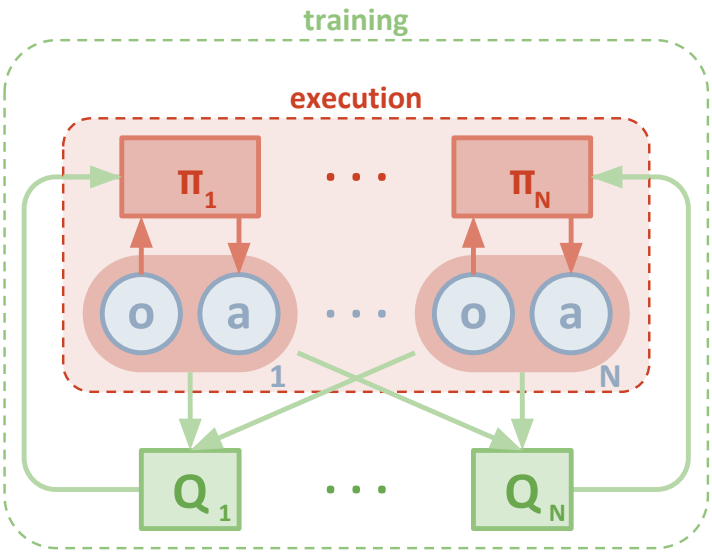


Figure 3.15: MADDPG algorithm schematic, with action-value Q functions that have access to all observations and actions. The policies π use local observations only. Retrieved from [28, p. 4].

One feature of MADDPG is that, as shown in Algorithm 1, line 12, the policies of other agents are required to update the critic. This does not pose a problem for in-simulation learning, however, for in situ learning during planetary exploration, this information might not be available. The authors of MADDPG [28] relax this requirement by allowing the learning of the policies of other agents from observations. This is done as follows; *"each agent i can additionally maintain an approximation $\hat{\mu}_{\phi_i^j}$ (where ϕ are the parameters of the approximation; henceforth $\hat{\mu}_i^j$) to the true policy of agent j, μ_j. This approximate policy is learned by*

maximizing the log probability of agent j's actions, with an entropy regulariser" [28, p. 5]:

$$\mathcal{L}(\phi_j^i) = -E_{o_j,a_j}\left[\log \hat{\mu}_i^j(a_j|o_j) + \lambda H(\hat{\mu}_i^j)\right] \tag{3.8}$$

"Where H is the entropy of the policy distribution. With the approximate policies, y in Algorithm 1, line 11, can be replaced by an approximate value $\hat{y}$ calculated as follows" [28, p. 5]:

$$\hat{y} = r_i + \gamma Q_{\mu i}(x', \hat{\mu}_i^1(o_1), \ldots, \mu_i'(o_i), \ldots, \hat{\mu}_i'^N(o_N)) \tag{3.9}$$

"Where $\hat{\mu}_i'^j$ denotes the target network for the approximate policy $\hat{\mu}_i^j$. Note that Equation 3.8 can be optimized in a completely online fashion: before updating Q_i^μ, the centralized Q function, the latest samples of each agent j from the replay buffer are used to perform a single gradient step to update ϕ_i^j. Note also that, in Equation 3.9, the action log probabilities of each agent are inputted directly into Q, rather than sampling" [28, p. 5].

Although contributing towards stable learning, it should be noted that a drawback of this algorithm is that it **scales poorly** as the number of agents increases since the input space of the action-value function grows quadratically as more agents are included during learning, together with the memory requirements of the replay buffer $\mathcal{D}$, shown in Algorithm 1.
This might be solved if emergent exploration behaviors can be achieved during learning, and then the found policies are embedded into a larger amount of agents during execution (real deployment), still exhibiting the swarm behavior (much like an ant colony can maintain its exploration behavior when the number of ants changes).

Algorithm 1: MADDPG, retrieved from [28].

Input: Number of episodes M, max-episode-length
for $episode = 1$ *to* M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state x
  for $t = 1$ *to max-episode-length* **do**
    For each agent i, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $\mathbf{a} = (a_1, \ldots, a_N)$ and observe reward r and new state x'
    Store (x, a, r, x') in replay buffer $\mathcal{D}$
    $x \leftarrow x'$
    for *agent* $i = 1$ *to* N **do**
      Sample a random minibatch of S samples (x^j, a^j, r^j, x'^j) from D
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a_1', \ldots, a_N')\big|_{a_k' = \mu_k'(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j (y^j - Q_i^\mu(x^j, a_1^j, \ldots, a_N^j))^2$
      Update actor using the sampled policy gradient:
      $\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^\mu(x^j, a_1^j, \ldots, a_i^j, \ldots, a_N^j)\big|_{a_i = \mu_i(o_i^j)}$
    end
    Update target network parameters for agent i: $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
  end
end

Actor-Attention-Critic for Multi-Agent Reinforcement Learning (MAAC)
A different approach to MARL involves also using a centralized action-value function (critic) such as with MADDPG (Section 3.3.4), but including an attention mechanism in the critic of each agent [34]. This attention mechanism, shown in Figure 3.16, can intelligently help reduce information overload by allowing the critic to dynamically select which agents it should pay attention to at each time step during training [34]. This is essential for improving the scalability of the MARL algorithm since the critic input space **grows linearly with respect to the number of agents**, as opposed to the quadratic growth of the aforementioned MADDPG, Section 3.3.4.
Specifically, the action-value function $Q_i^\psi(o, a)$, shown in Equation 3.10, consists of an FF network f_i

(upper MLP in Figure 3.16) which takes as an input the output of an embedding FF network g_i (lower MLP) and the contribution from other agents x_i, calculated through the multi-headed attention mechanism.

$$Q_i^{\psi}(o,a) = f_i(gi(o_i,a_i),x_i) \tag{3.10}$$

Note that in this set-up, the model weights for the critic are shared across all agents, with the idea of encouraging a common embedding space [34]. This has the advantage of allowing for effective learning in scenarios where different agents have different reward functions but share common features [34], as it could be in a planetary exploration scenario where different robots have different tasks with certain similarities.

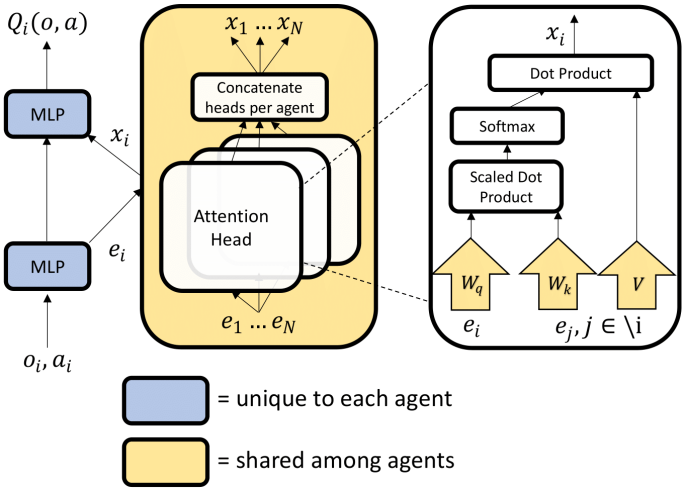


Figure 3.16: Schematic of the calculation of the action-value of agent i $Q_i^{\psi}(o,a)$ with a multi-headed attention mechanism. Note that the attention block is shared among agents. Retrieved from [34].

As previously mentioned, the MAAC algorithm **scales better** than MADDPG as agents are added since the attention mechanism compresses the information from other agents x_i into a **constant**-sized vector. This is empirically shown in Figure 3.17, where, for an example cooperation task in [34] the MAAC performance stays constant as more agents are increased, whereas the mean episode rewards of MADDPG severely deteriorate.
Lastly, this algorithm uses all actions from the agents' **current** policies to calculate the gradient estimate, as opposed to MADDPG, which requires a memory buffer that can potentially overgeneralize such that agents fail to coordinate based on their current policies [34].
Relevantly, both MADDPG and MAAC rely on a centralized critic with perfect communication with the agents, which can be available during simulation, but not during deployment. Hence, its potential might be expanded by performing some algorithm modifications in that regard.

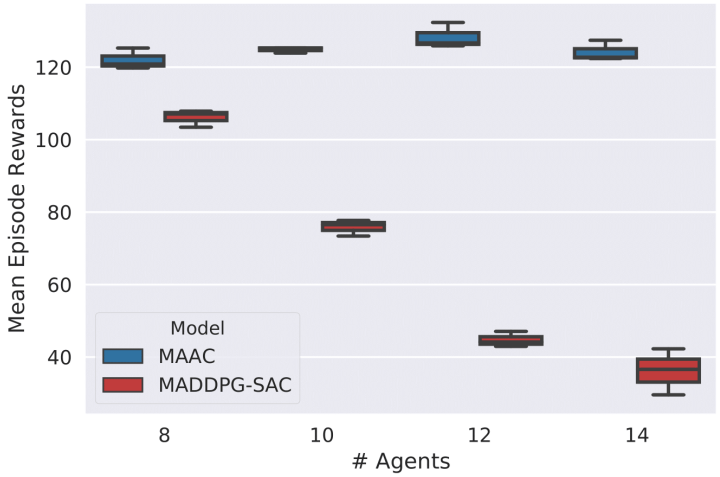


Figure 3.17: Agent scalability of the MAAC and MADDPG algorithm. The performance of MAAC doesn't deteriorate as more agents are added. Retrieved from [34].

Multi-Agent Proximal Policy Optimisation (MAPPO)
The third considered algorithm is the multi-agent extension of Proximal Policy Optimisation; MAPPO (see Algorithm 2). This algorithm is investigated to offer a comparison point against MADDPG, since it is also actor-critic, and helps compare off-policy and on-policy MARL performance.
It also often achieves superior results than off-policy algorithms regarding final returns and sample efficiency [35], as shown in Figure 3.18.

The family of PPO algorithms addresses the challenge of maximizing policy improvement while avoiding sudden performance collapse. Standard policy gradient algorithms (such as MADDPG) perform small updates in the parameter of the actor and critic networks. However, it can occur that even though the difference between the old policy and the new, updated one is small in parameter space, it can result in large differences in performance. With this in mind, PPO utilizes first-order techniques supplemented with additional strategies to ensure that new policies remain close to the old ones. PPO algorithms can offer competitive performance in this manner.
In particular, the clip version of the algorithm is implemented in [35], which uses a specialized clipping in the objective function such that the new policy can't get far from the old one (in performance space).

Thus, multi-agent PPO can potentially be used for swarm planetary exploration, since implementing CTDE or similar techniques used by MADDPG or MAAC is possible.

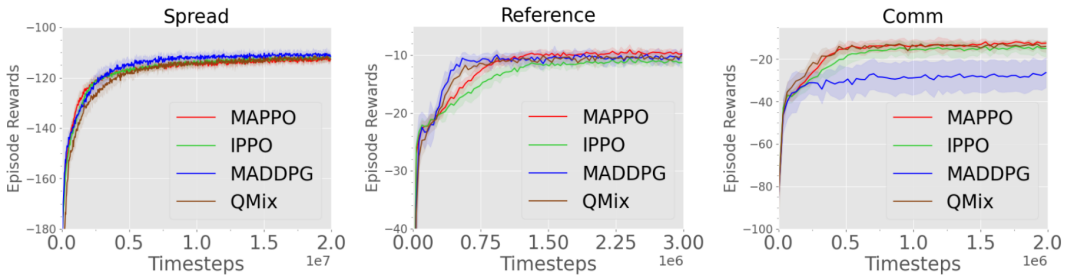


Figure 3.18: Performance of different MARL algorithms in simple multi-agent environments. Retrieved from [35].

Algorithm 2: Recurrent-MAPPO, retrieved from [35]

Input: Number of steps stepmax, batch size batch_size, learning rate α
Initialize : θ: Parameters for policy π, ϕ: Parameters for critic V using Orthogonal initialization (Hu et al., 2020)
while *step* $\leq$ *stepmax* **do**
  Set data buffer $D = \{\}$
  for $i = 1$ *to batch_size* **do**
    $\tau = []$ (empty list)
    Initialize $h(1)_0, \pi, \ldots, h(n)_0, \pi$ (actor RNN states)
    Initialize $h(1)_0, V, \ldots, h(n)_0, V$ (critic RNN states)
    for $t = 1$ *to* T **do**
      for *all agents* a **do**
        $p(a)_t, h(a)_t^{\pi} = \pi(o(a)_t, h(a)_{t-1}^{\pi}; \theta)$
        $u(a)_t \sim p(a)_t$
        $v(a)_t, h(a)_t^{V} = V(s(a)_t, h(a)_{t-1}^{V}; \phi)$
      end
      Execute actions u_t, observe r_t, s_{t+1}, o_{t+1}
      $\tau += [s_t, o_t, h_t^{\pi}, h_t^{V}, u_t, r_t, s_{t+1}, o_{t+1}]$
    end
    Compute advantage estimate $\hat{A}$ via GAE on τ, using PopArt
    Compute reward-to-go $\hat{R}$ on τ and normalize with PopArt
    Split trajectory τ into chunks of length L
    for $l = 0, 1, \ldots, T//L$ **do**
      $D = D \cup (\tau[l : l + T], \hat{A}[l : l + L], \hat{R}[l : l + L])$
    end
  end
  for *mini-batch* $k = 1, \ldots, K$ **do**
    $b \leftarrow$ random mini-batch from D with all agent data
    for *each data chunk* c *in the mini-batch* b **do**
      Update RNN hidden states for π and V from first hidden state in data chunk
    end
    Adam update θ on $L(\theta)$ with data b
    Adam update ϕ on $L(\phi)$ with data b
  end
end

Hierarchical Reinforcement Learning

Hierarchical MARL is also investigated due to its potential applicability to swarm space exploration, because of its suitability to be included in the design of a NASA MaCMAS model (see Section 3.2.2). In particular, there is a focus on hierarchical, fully cooperative MARL. The method proposed at [36] is investigated, and can be summarised as follows:

1. A two-level hierarchical agent structure is established, comprising high-level policies that select latent variables for extended periods and low-level policies that use observations and selected latents for basic actions.
2. The training strategy combines centralized training for high-level policies with an extrinsic team reward and decentralized training for low-level policies using intrinsic rewards and team rewards, enabling the utilization of robust algorithms for cooperative MARL and single-agent RL.
3. Intrinsic rewards are defined based on a decoder's performance in predicting latent variables from trajectories produced by low-level policies. Dynamic weighting between intrinsic and extrinsic rewards guides low-level policies to balance decodability and utility, avoiding the need for manually designed skill-specific rewards.
4. Hierarchical agents outperform flat methods in ad-hoc cooperation scenarios, showcasing adaptability when collaborating with teammates using previously unseen policies. This approach provides a

promising foundation for cooperative MARL with autonomous skill discovery.

This can be seen as a bi-level optimization problem, where *"at the high level (left of figure Figure 3.19), the extrinsic team reward is used to train a centralized action-value function $Q_{tot}(s, z)$ that decomposes into individual utility functions $Q^n(o_n, z_n)$ for decentralized selection of latent skill variables z. At the low level (right of figure Figure 3.19), skill-conditioned action-value functions $Q^n(o_n, z_n, a_n)$ take primitive actions independently. Trajectories τ generated under each z are collected into a data-set $D = \{(z, \tau)\}$, which is used to train a skill decoder $p(z|\tau)$ to predict z from τ. The probability of selected skills under $p(z|\tau)$ is the intrinsic reward for low-level Q^n"* [36]. The complete algorithm is shown in Algorithm 3.

This algorithm has the potential to contribute to swarm space exploration since it has been shown to develop quantifiable, distinct, and useful skills in stochastic multi-agent environments [36]. This could be used to discover or construct high-level strategies in space scenarios.

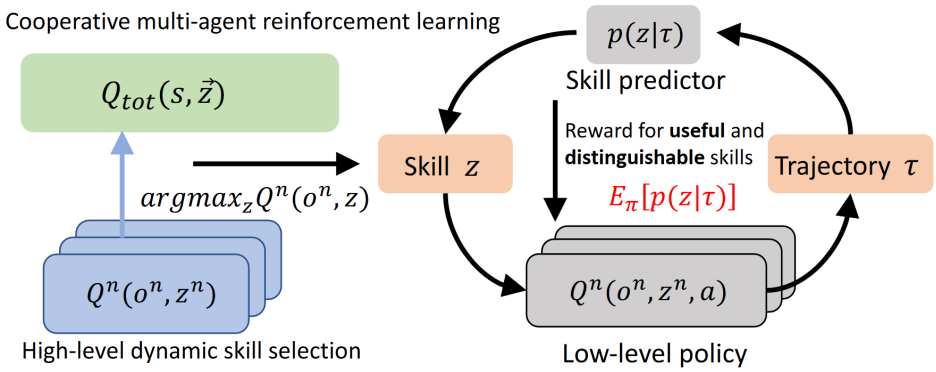


Figure 3.19: Hierarchical MARL with unsupervised skill discovery. Retrieved from [36].

Algorithm 3: Hierarchical MARL with unsupervised skill discovery. Retrieved from [36].

Input: Hyperparameters: γ, t_{seg}, t_{train}, N_{batch}, $\alpha_{\text{threshold}}$, α_{step}, α_{end}
Initialize : Q_ϕ: High-level Q-function, Q_θ: Low-level Q-function, p_ψ: Decoder, B_H: High-level replay buffer, B_L: Low-level replay buffer, D: Trajectory-skill dataset
for *each episode* **do**
  $s_t, o_t =$ env.reset()
  Initialize trajectory storage $\{\tau^n\}_{n=1}^N$ of max length t_{seg}
  for *each step* $t = 1, \ldots, T$ *in episode* **do**
    if $t \mod t_{seg} = 0$ **then**
      if $t > 1$ **then**
        Compute $\tilde{R}_t := \gamma^{t_{\text{seg}}} \sum_{k=0}^{t_{\text{seg}}} R_{t-k}$
        Store $(s_{t-t_{\text{seg}}}, o_{t-t_{\text{seg}}}, z, \tilde{R}_t, s_t, o_t)$ into B_H
        for *each agent* n **do**
          Store (z_n, τ^n) into D
          Compute intrinsic reward $R_{I_n} := p_\psi(z_n|\tau^n)$
        end
      end
      Select new z_n by ϵ-greedy($Q_n^\phi(o_n, z)$), $\forall n \in [N]$
      if #(*high level steps*) $\mod t_{train} = 0$ **then**
        Update $Q_\phi(s, z)$ using B_H and $L(\phi) := \mathbb{E}_{\mu,\pi}\left[\frac{1}{2}(y_k - Q_\phi^{\text{tot}}(s_k, z_k))^2\right]$
      end
    end
    Get a_t^n from ϵ-greedy($Q(o_t^n, z_t^n, a)$) for each agent
    $s_{t+1}, o_{t+1}, R_t =$ env.step(a_t)
    for *each agent* n **do**
      Compute $R_{nL} := \alpha R_t + (1 - \alpha) R_{I_n}$
      For all agents, store $(o_t^n, a_t^n, R_{nL}, o_{t+1}^n, z_t^n)$ into low-level replay buffer B_L, and append o_t^n to trajectory τ^n
      if #(*low-level steps*) $\mod t_{train} = 0$ **then**
        Update $Q_\theta(o_t^n, z_t^n, a_t^n)$ using B_L and $L(\theta) := \mathbb{E}_{\mu,\pi}\left[\frac{1}{2}(y_t^n - Q_\theta^n(o_t^n, z^n, a_t^n))^2\right]$
      end
    end
    if *size of* $D \geq N_{batch}$ **then**
      Update decoder $p_\psi(z|\tau)$ using D, then empty D
    end
    if *evaluation win rate exceeds* $\alpha_{threshold}$ **then**
      $\alpha \leftarrow \max($ $\alpha_{\text{end}}, \alpha - \alpha_{\text{step}})$
    end
  end
end

Federal Learning Deep Deterministic Policy Gradient (FLDDPG)
When looking at DDPG communication-architecture variants of multi-agent systems, several variants can be distinguished, as shown in Figure 3.20: IDDPG, SNDDPG, SEDDPG, and FLDDPG. IDDPG, Independent DDPG, deals with a fully decentralized training scheme, where all agents have their own ANN and local memory. Because it is fully decentralized, this method results in unstable learning, as discussed in Section 3.3.2, and usually fails to achieve collective behaviors [37]. Shared Experience DDPG (SEDDPG) is a state-of-the-art algorithm [37] that involves agents having individual neural networks and a shared memory [37], and that *"despite the improvement in the training speed and performance, the robots still share the collected data with the central server, which requires significant communication bandwidth"* [37]. Shared Network DDPG (SNDDPG) shares both the network and the memory across agents. *"In every training period, the network update is performed by the central server, and the model is distributed to the individual agents"* [37]. This has the benefit that the data can be collected from different agents in

different environments [37]. This signifies a more generalizable policy compared to IDDPG, but comes at the cost of a high reliance on communication with the central server (relatively similar in this sense to MADDPG and MAAC), thus making this algorithm vulnerable in situations with unstable communication [37].

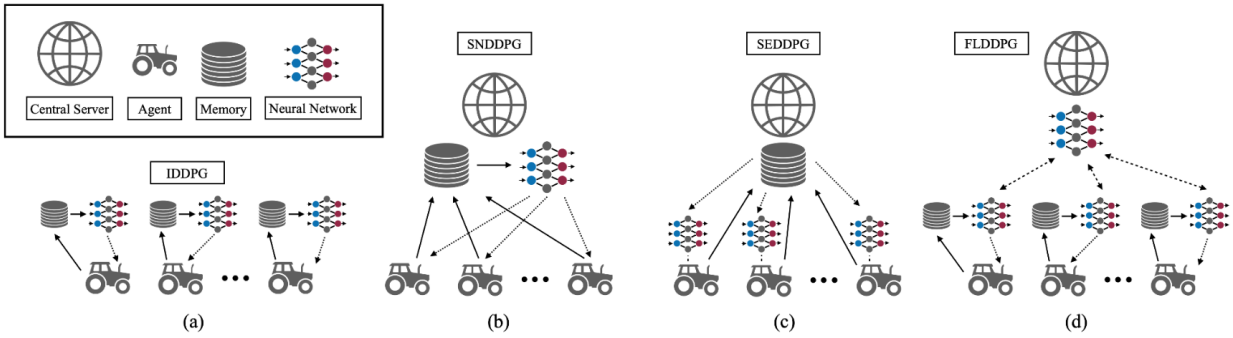


Figure 3.20: Different DDPG learning architectures for swarm robotic systems. Noticeable are the differences in communication structure. Retrieved from [37].

The fourth option proposed by Seongin Na et al [37] deals with centrally sharing the ANN **parameters** only (the agents don't share their collected data). The complete scheme is described in Algorithm 4. Here, the ANN update is performed **individually** for each agent every designed time t_{train} (see Algorithm 4). Then, the update of the target networks is done after a designed t_{target}, much like in other deep reinforcement learning schemes.
Importantly, in Algorithm 4 the parameters of the individual ANNs of the agents are updated **centrally** every update period T_{wa}, where instead of performing a hard update as suggested in [38] (in which each ANN parameter θ of the agent is updated to be the average of that parameter θ_{wa} across all agents, as shown in Equation 3.11), a *soft weight update* is done by fractionally updating the agent's ANN weights through a parameter $\tau \in (0, 1)$. *"When τ is close to zero, the local neural network weights are completely replaced with the averaged ones. In contrast, when τ is close to 1, it becomes analogous to IDDPG"* [37]. Performing a soft update method can overcome the problem of the hard weight update method dealing with the fact that when *"the local weights are updated with the averaged weights, adverse change in the neural networks can occur, decreasing the efficiency of individual controllers in their corresponding environments and tasks after the update"* [37].

$$\theta_{wa} = \frac{1}{N}\sum_{k=1}^{N}\theta_k; \quad \theta_1, \ldots, \theta_N = \theta_{wa} \tag{3.11}$$

Algorithm 4: FLDDPG, retrieved from [37].

Input: Initialise N critic neural networks: $Q_1, \ldots, Q_N\ (s, a|\theta_{Q_1}, \ldots, \theta_{Q_N})$ and N actor neural networks: $\pi_1, \ldots, \pi_N\ (s|\theta_{\pi_1}, \ldots, \theta_{\pi_N})$ with weights $\theta_{Q_1}, \ldots, \theta_{Q_N}$ and $\theta_{\pi_1}, \ldots, \theta_{\pi_N}$
Input: Initialise N target networks: $Q'_1, \ldots, Q'_N$ and $\pi'_1, \ldots, \pi'_N$ with weights $\theta_{Q'_1} \leftarrow \theta_{Q_1}, \ldots, \theta_{\pi'_1} \leftarrow \theta_{\pi_1}$
Input: Initialise replay buffers $R_1, \ldots, R_N$
for *episode* $= 1, \ldots, M$ **do**
  Initialise the states $s_t = s_1$
  for $t = 1, \ldots, T$ **do**
    Run N actors and collect transition samples $D_t = (s_t, a_t, r_t, s_{t+1})$ into $R_1, \ldots, R_N$
    if $t = 0 \mod t_{train}$ **then**
      Sample l transitions (s_i, a_i, r_i, s_{i+1}) from local replay buffer memories $R_1, \ldots, R_N$
      Set $y_i = r_i + \gamma Q'(s_i, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
      Update critic networks by minimising the loss:
      $L_Q = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
      Update actor networks using the sampled policy gradient:
      $\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s_i|\theta^\pi)|_{s=s_i}$
    end
    if $t = 0 \mod t_{target}$ **then**
      Update the target networks:
      $\theta^{Q'} \leftarrow \theta^Q, \theta^{\pi'} \leftarrow \theta^\pi$
    end
  end
  if *episode* $= 0 \mod T_{wa}$ **then**
    Perform *Soft Weight Update* for actor and critic networks for all robots
    $\theta_{wa} = \frac{1}{N}\sum_{k=1}^{N} \theta_k$
    for $i = 1, \ldots, N$ **do**
      $\theta_i = \tau\theta_i + (1 - \tau)\theta_{wa}$
    end
  end
end

As quoted from Seongin Na et al, FLDDPG's robustness and learning in unstable communication scenarios *"can benefit swarm robotic systems operating in environments with limited communication bandwidth, e.g., in high-radiation, underwater, or subterranean environments"* [37], and thus, it can potentially be applied in swarm space exploration scenarios where direct communications with a centralised entity are not possible, such as when exploring a lava cave for example. This type of performance can be observed in Figure 3.21, where FLDDPG achieves both the highest learning reward, and crucially for space exploration, the least amount of catastrophic events and failed agents.

An important aspect of FLDDPG is that it **does not consider different types of agents** and that in the training of the robots described by Seongin Na et al [37], **the robots do not communicate with each other to obtain a collaborative behaviour, but only use the collective behaviour to better update their ANNs**. This algorithm thus still requires certain modifications entailing local communication models, network updates, and heterogeneous agent characteristics.

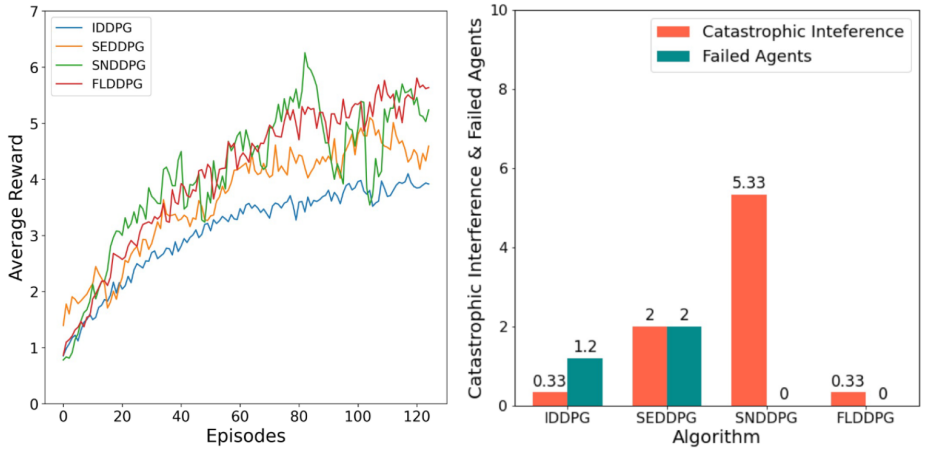


Figure 3.21: Average training of IDDPG, SEDDPG, SNDDPG, and FLDDPG (left image), and failure characteristics of these algorithms (right image), in a robot exploratory environment. Noticeable are the rewards obtained by SNDDPG and FLDDPG, and the high robustness of FLDDPG. Retrieved from [37].

3.3.5. MARL in the Context of Swarm Space Exploration

Regarding MARL space exploration, only a few approaches have been tried so far. First, an extension to the MADDPG algorithm is described in Section 3.3.5, and its applicability to space exploration is also discussed. Then, NASA's MARL approach is also examined in Section 3.3.5.

MADDPG+E

Yixin Huang et al [22] propose a MADDPG algorithm approach where the MADDPG is modified in the way the policies get updated, by using a policy-ensemble method, in which each agent has several sub-policies and where the objective is to maximise the policy **ensemble** reward. Moreover, each sub-policy K has its own replay buffer $\mathcal{D}$ storing the policy in different episodes. The gradient of the **ensemble objective** is defined as follows [22]:

$$\nabla_{\theta_i^{(k)}} J_e(\boldsymbol{\mu}_i) = \frac{1}{K}\mathbb{E}_{x,a\sim\mathcal{D}_i^{(k)}}[\nabla_{\theta_i^{(k)}}\boldsymbol{\mu}_i^{(k)}(a_i \mid s_i)\nabla_{a_i}Q^{\boldsymbol{\mu}_i}(\mathbf{x}, a_1, \ldots, a_n)|_{a_i=\boldsymbol{\mu}_i^{(k)}(s_i)}] \tag{3.12}$$

This algorithm, which will be referred to as MADDPG+E, uses an experience sample optimiser that gives priority to recent experiences by having a probability $Pr(i)$ (for agent i) of sampling the current experience:

$$\text{Pr}(i) = \frac{p_i^\alpha}{\sum_{j=1}^{M} p_j^\alpha} + \beta \tag{3.13}$$

Where α is the amplification factor of priority, $\beta \in (0, 1)$ is the bias factor of probability, and p_i (p_j can be defined in a similar way) is the ranking of the target network loss function of agent i [22]. The use of the experience sample optimiser has limited advantages on simple scenarios but improves training results as the environment's complexity grows [22].

Although improving on the standard MADDPG algorithm during training, this method in itself **does not** address the requirements linked to swarm space exploration, specifically during mission execution, such as the ones outlined in Section 3.1.3, due to its high dependency on stable communications. Moreover, as discussed in Section 3.3.4, MADDPG has scalability issues that have not been addressed in this method.

NASA's MARL Approach

In the paper *"Towards the Development of a Multi-Agent Cognitive Networking System for the Lunar Environment"*, NASA's Rachel Dudukovich et al [23] use a MARL approach to dynamically account for *"the various decision-making elements involved in link quality assessment, data selection, and routing,*

as well as how decisions of one node affect another" [23] within a network of an exploratory multi-agent system where interrelated decisions affect network congestion and memory capacity in multiple areas of the network.
They investigate MARL schemes that can learn their environment in real-time, also stating that these models might need to do initial training off-line from a data set. Particularly, they consider Deep Q-learning and Advantage Actor-Critic, where they propose using a centralised critic and decentralised actors. Moreover, they identify routing and link selection, data selection, and network parameter tuning as the potential tasks the algorithms can do. In their work, Rachel Dudukovich et al [23] implement Deep Q-learning in identifying routing and link selection in a network consisting of 100 nodes, reducing the latency by approximately 40%.

3.3.6. MARL Implementation Plan

From an algorithmic perspective, several state-of-the-art MARL algorithms have been considered. All the discussed architectures can be applied to a swarm exploration scenario, given that it consists of homogeneous agents (FLDDPG would need to be modified in this regard). With this in mind, the implementation strategy is based on first implementing the standard algorithm in the field, MADDPG, and then pivot based on performance. As it will be shown in Section 3.4, DDPG's convergence capabilities tend to be worst than other methods like PPO, which might suggest that the multi-agent version has similar issues in this regard. Also, as discussed in Section 3.3.4, MADDPG scales poorly with the number of agents, hence even though it is selected as the baseline, the following implementation order is considered in case MADDPG is experimentally found to under-perform in test environments:

1. MADDPG.
2. MAPPO.
3. MAAC.
4. FLDDPG.
5. Hierarchical MARL.

These pivots are based on the complexity of the algorithms, providing for an iterative design direction.

3.3.7. Conclusion

In this section, the foundations of reinforcement learning have been introduced, highlighting the important main RL elements: the policy, reward signal, and value functions. Furthermore, Markov decision processes have been explained and identified as the way to formulate the classic RL problem. Then, the RL learning process has been explained, highlighting the importance of temporal difference methods, and how they fit within the set of Monte Carlo methods. Additionally, the different RL design choices have been explained; online vs off-line learning, on-policy vs off-policy updates, exploration vs exploitation, and the concept of actor-critic methods, which provide the foundation for some state-of-the-art MARL algorithms.

Regarding MARL, there is a trade-off to be made between stability of learning and adaptability to other agents. Moreover, the MARL learning problem gets exacerbated because it deals with Markov decision processes that can be decentralised, and/or partially observable, which can lead to super-exponential times to solve. Function approximators are essential for deploying MARL in complex scenarios, and both ANNs and fuzzy logic have been discussed.
State-of-the-art algorithms such as MADDPG, MAPPO, MAAC, and MADDPG+E use an actor-critic configuration where there is a centralised critic with global information during training (either by considering the actions and observations of all agents, containing additional global information from the environment, or embedding that information through an attention mechanism) and a decentralised actor that can be deployed after training, and who's learning can be strengthened via experience sampling from a replay buffer (not applicable in MAPPO and MAAC), as well as using the ensemble of policies or collected experiences for gradient update.
Together with the aforementioned algorithms; based on the central-critic, decentralised actor, hierarchical RL has also been investigated, using a two-level hierarchy that selects latent variables for extended periods of time, and then uses low-level policies to perform the basic actions in the environment. Federal learning is also explored; where the function approximator parameters can be updated together with other agents. This can allow for swarm teams with unstable communication, but has not been applied to heterogeneous agents, and does not consider cooperation strategies, but collective behaviours to better update the agent's ANNs.

Moreover, the specific field of MARL applied to swarm space exploration has been studied; assessing that it is in a non-mature stage, where (to the best knowledge of the author) only a variant version of MADDPG has been proposed, as well as a NASA development approach for a multi-agent networking system for the Moon, where centralised critic, decentralised actor methods are again proposed, as well as using Deep Q-learning for optimising networks.

Lastly, an implementation plan is envisioned based on an incrementally more complex approach to the swarm planetary exploration problem, first implementing MADDPG, and later allowing to pivot (based on experimental performance) to MAPPO, MAAC, FLDDPG, and hierarchical MARL, in that order.

3.4. Preliminary Analysis

This section implements state-of-the-art RL algorithms in a simplified cooperative environment. Firstly, Section 3.4.1 explains the motivation for doing this experiment, then, Section 3.4.2 describes the environment, and subsections 3.4.3 and 3.4.4 describe the experiment results. Finally, Section 3.4.5 summarises the findings of this section, where the experimental results will contribute towards answering Research Question 3.

3.4.1. Motivation for a Preliminary Analysis

MARL algorithms, with the continuous action and state spaces considered in this research, have a significant level of complexity (see subsections 3.3.2, 3.3.3, and 3.3.4). Moreover, a high-fidelity environment of a space mission is also time-consuming to simulate or pipeline (in the case of using existing software) for a preliminary study. Here, the objective is to perform early RL tests on simplified environments to empirically assess the factors that can help select the MARL algorithm to later implement in this research, as well as familiarise with RL algorithms and theory discussed in this report. Furthermore, this preliminary analysis is also done for several reasons:

1. **Benchmarking and baseline performance:** Single-agent RL algorithms provide a baseline for performance evaluation. Later in this research, the results obtained by the single-agent RL algorithms can potentially be compared against their multi-agent counterpart (to be developed). This can help understand the relative advantages of MARL.
2. **Identification of challenges:** Comparing single-agent algorithms to MARL algorithms can help identify the unique challenges and complexities introduced by multi-agent interactions in later research.
3. **Algorithm selection:** Understanding the strengths and weaknesses of single-agent algorithms can help inform the decision of which MARL algorithm to develop in this research (see Section 3.5).
4. **Hyper-parameter tuning:** By tuning hyper-parameters for single-agent algorithms, insights can be gained about which settings might be beneficial in the multi-agent scenario.
5. **Training stability:** Observing how single-agent algorithms handle training stability can inform the design of strategies to stabilise training in MARL.
6. **Transferable techniques:** Some techniques and insights from single-agent RL might be transferable to MARL, such as reward shaping, or policy gradient methods, for example.
7. **Prototyping and rapid development:** Single-agent RL algorithms are often simpler and quicker to prototype and experiment with. This allows experimenting more rapidly before investing extensive resources developing the MARL algorithm.

Moreover, several, **single-agent** RL algorithms are considered, namely; Deep Deterministic Policy Gradient (DDPG), Sof-Actor-Critic (SAC), and Proximal Policy Optimisation (PPO). This is done for several reasons. Firstly, most of the state-of-the-art MARL algorithms discussed in Section 3.3.4 use some form of multi-agent DDPG, hence making it a good baseline to compare, in the first place, whether DDPG offers the correct single-agent platform to then modify to accommodate for several agents. Secondly, in this early research, where fast prototyping is a priority, and hyper-parameter tuning of the RL models is limited, SAC is a state-of-the-art algorithm that has certain robustness that allows for its off-the-self deployment with successful results. Lastly, while DDPG and SAC are off-policy algorithms (although SAC explores in an on-policy way), PPO offers an on-policy comparison. All agents are implemented with `Stable Baselines 3 2.0.0` [39] on its default parameters, and with FF ANN policies. Furthermore, the DDPG actor network is updated after every time-step in the environment.

This analysis serves to answer two main questions, related to answering Research Question 3:

- What is the effect of having sparse vs dense reward functions for learning a cooperative multi-agent task?
- How well do state-of-the-art single-agent RL algorithms scale with the number of agents in a cooperative environment?

3.4.2. Simulated environment

In order to assess the performance of the different RL algorithms used in this preliminary research, a cooperative object transportation task is implemented, which is an adaptation of the *"coordinated multi-agent object transportation"* environment described by L. Buşoniu, R. Babuvska, and B. De Schutter

in [25, p. 28], but modified to accommodate continuous state and action spaces instead, in order to resemble a more representative scenario that might be encountered in swarm space exploration. A cooperative-manipulation task is selected since it is one of the least developed technologies for swarm space exploration, as described in Section 3.1.2, and thus its research is of interest. Furthermore, the environment can be adapted such that a variable number of agents can be used.

An image of the environment is shown in Figure 3.22, where there is a $10m$ x $10m$ two-dimensional continuous grid in which the agents (shown in blue) can navigate. The task is thus to push an object (shown as a red square) to the target location (the green cross). The box can only be moved if **all** the agents interact with it simultaneously. Notice that this is in fact a challenging environment, with complexity $\mathcal{O}(> n)$ [18].

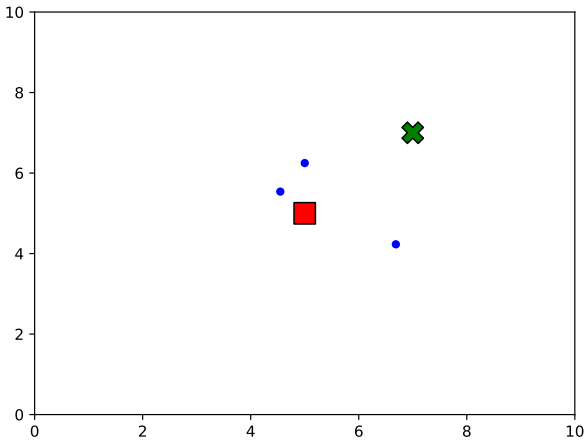


Figure 3.22: Simulated swarm environment. The red square is the object to transport, the green cross is the target location where to transport the object, and the three blue dots are the agents.

The global observation space (when applying a centralised RL algorithm on all N agents) comprises the x and y locations of each agent, and the locations of the object and the target, as shown in Equation 3.14. Each agent can move in the horizontal and vertical (x and y) axes by modifying their velocities. Hence, the global action space is the velocities of the agents, shown in Equation 3.15. The state-space of the system is $S = [\mathcal{O}, A]^T$.

$$\mathcal{O} = [x^1, y^1 ... \, x^N, y^N, x_{obj}, y_{obj}, x_{targ}, y_{targ}]^T \tag{3.14}$$

$$A = [v_x^1, v_y^1 ... \, v_x^N, v_y^N]^T \tag{3.15}$$

Lastly, the pushing action on the box is applied by first ensuring that all the agents are within a specific norm on the object, and then doing a simulated momentum transfer, scaled by the number of agents and their velocity vectors, as shown in Equation 3.16.

$$P_x^{obj} = \frac{\sum_{n=1}^{N} v_x^n}{N}, \; P_y^{obj} = \frac{\sum_{n=1}^{N} v_y^n}{N} \tag{3.16}$$

3.4.3. Experiment I: Reward Functions

Two reward functions are compared to assess how they affect learning; sparse and dense reward functions. The sparse reward function is inversely proportional to the norm between the object and the target and is scaled to restrain its maximum value, shown in Equation 3.17, while the dense reward function also includes a component relating the distance between the robots and the object to be pushed, as shown

in Equation 3.18. Here, the cubic exponential was found experimentally, and the factors Q and K need to be adjusted depending on the number of agents to ensure that the distance-to-target reward has also influence (a large number of agents combined with a low K and a large Q results in a second element of the reward function that is severely more meaningful than the first). In this experiment $K = 30$ and $Q = 1$.

$$R = -\frac{\sqrt{(x_{targ} - x_{obj})^2 + (y_{targ} - y_{obj})^2}}{10} \tag{3.17}$$

$$R = -\frac{\sqrt{(x_{targ} - x_{obj})^2 + (y_{targ} - y_{obj})^2}}{Q} - \sum_{n=1}^{N} \frac{[\sqrt{(x_n - x_{obj})^2 + (y_n - y_{obj})^2}]^3}{K} \tag{3.18}$$

Moreover, **three robots** are controlled by each RL agent, and the maximum duration of a learning episode is 16 steps. The results of the experiment are shown in Figure 3.23 and Figure 3.24, where several observations can be made. Firstly, except for SAC, dense reward functions foster learning, with sparse rewards leading to slower learning or no successful learning at all (successful learning is achieved when low episode lengths, high rewards, and low actor and critic losses occur). Furthermore, PPO has a noticeable convergence rate for the dense reward, achieving the fastest learning of all models (almost four times faster than the next model, SAC sparse).
During training, it was also noticed that DDPG is quite sensitive to the random starting conditions (especially for the sparse reward case), in cases where the actor was updated either after one or three time-steps, showing poor convergence capabilities.

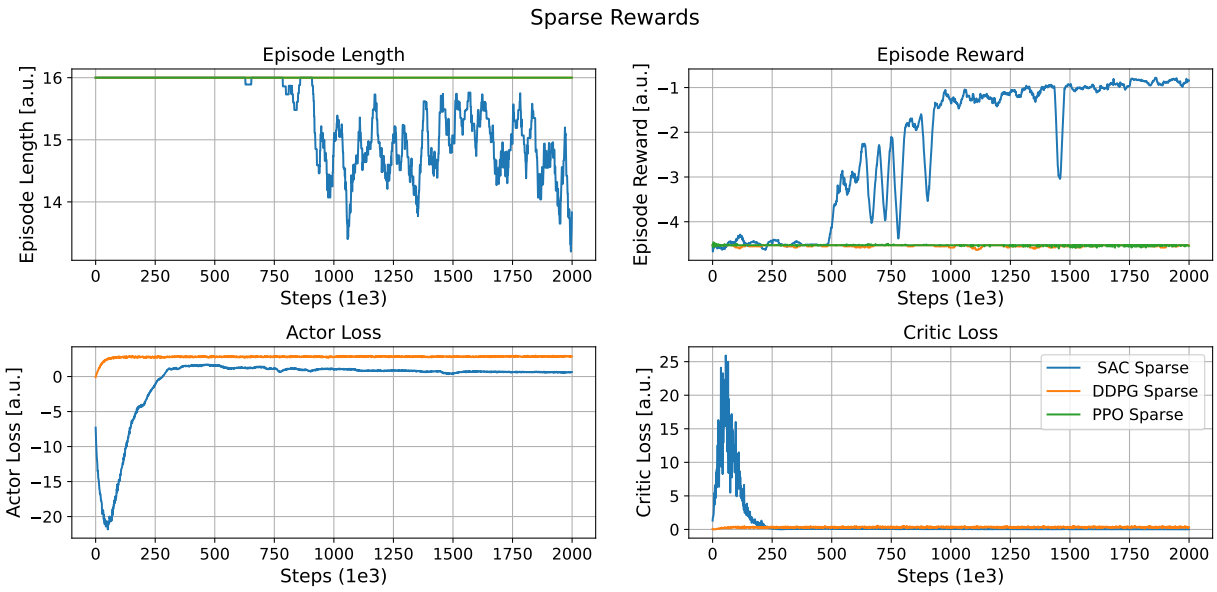


Figure 3.23: Results from Experiment I, sparse rewards. Here it is observed that except for SAC, sparse rewards hindered the learning performance.

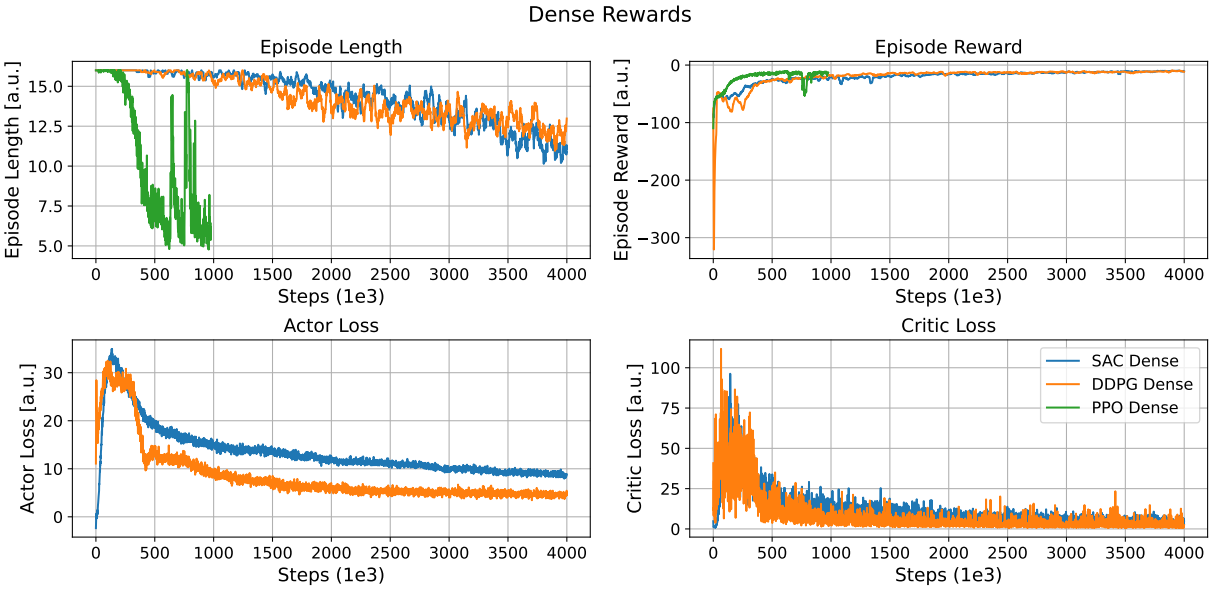


Figure 3.24: Results from Experiment I, dense rewards. Here it is observed that PPO rapidly learns, while SAC and DDPG require more steps to converge.

Moreover, dense reward functions increase the variance of the actor and critic losses. This could be due to the fact that the dense reward has significantly bigger numerical values at each time-step, as well as being more sensitive to states in which the robots wander without touching the box, thus making more abrupt changes in the value of a certain state, and consequently also leading to a more sudden change in the actor (policy) loss. Finally, it was also observed that the success of the learning, as expected, depends on how ”lucky” the agents were with having randomly generated initial states, with SAC and PPO being noticeably more robust than DDPG. Nonetheless, this sensitivity was reduced with the dense reward function, since it also helps provide values for states in which the robots are not interacting with the box, as opposed to the sparse reward, in which the agent needs to explore the state-space having no feedback signal in states where the robots are not in contact with the box. Having a sparse reward thus makes the exploratory search of the states in which all robots touch the box more unlikely. Similar results have been found in [40], as shown in Figure 3.25, in which a two-robot push-box scenario leads to unsuccessful learning of most agents with a sparse reward function, and the opposite with a dense reward. These results suggest that dense reward functions can help foster learning in multi-agent scenarios.

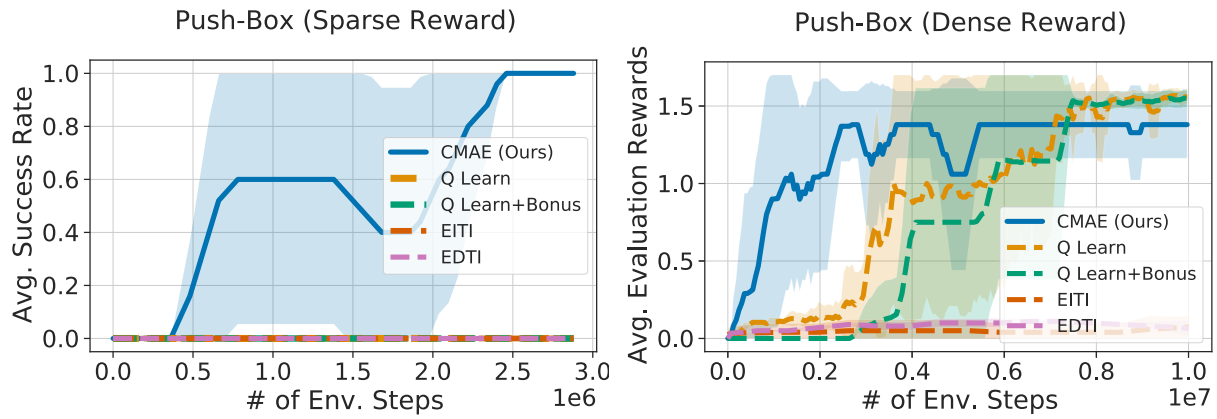


Figure 3.25: Reward density performance comparison for a push-box multi-agent (2 agents) environment. Retrieved from [40].

3.4.4. Experiment II: Scalability

The second experiment aims at analysing the robot scalability of the different RL algorithms. For this, the environment was modified, simulating 1 to 4 robots to observe the differences in learning (please note that the reward function is the same as Equation 3.18, but also including a $+100$ in case the learning episode is successfully finished, as this was found to foster learning). Extending the experiment to more robots was considered, but the algorithms showed extremely poor (or lacking) convergence capabilities from 5 robots onward (the training times are already on the million steps, sometimes going as far as 4 million steps until convergence), and this lack of convergence is already a meaningful result for the intent of this study.

Looking at the experiment result, firstly, Figure 3.26 displays SAC's scalability. Here, the convergence time significantly deteriorates as more agents are added. This is reflected in the form of a slower convergence of the reward function (top right image), taking more time to learn how to successfully move the box to the target (and thus finish the episode earlier, top left image), and greater values for the actor loss. Furthermore, the episode length duration shows that finding a successful pushing strategy for 3 and 4 robots is significantly more challenging. Moreover (although a more thorough analysis would be needed), having more robots decreased the variance of the critic loss (bottom right plot).

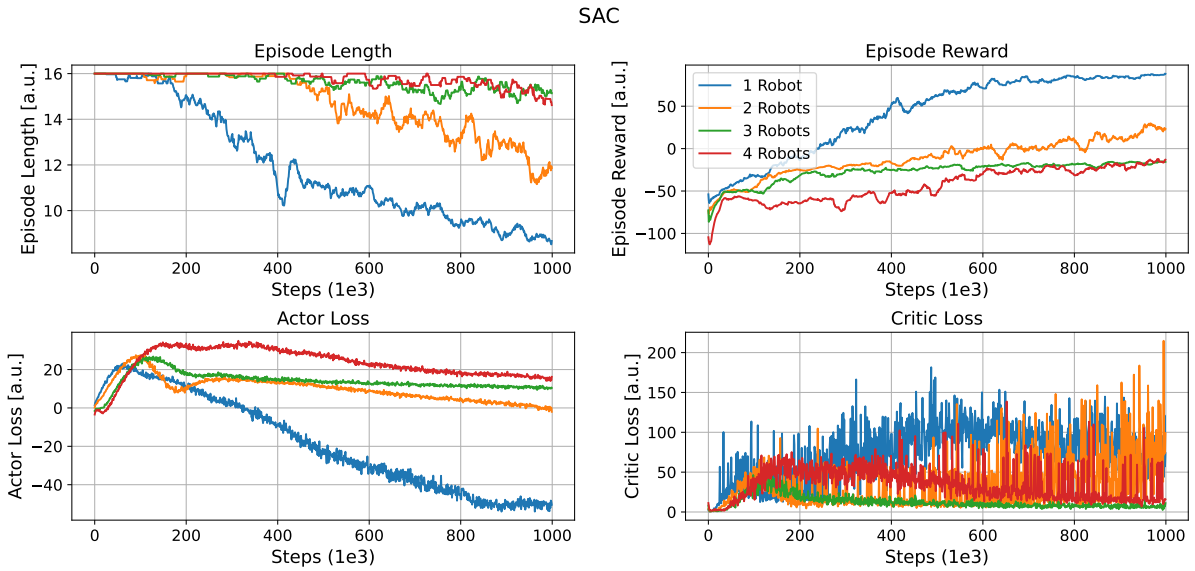


Figure 3.26: SAC scalability. Learning performance is shown for 1, 2, 3, and 4 robots.

To further investigate the time it takes to achieve convergence, the SAC learning run for 1 and 4 robots was extended to 4 million steps, see Figure 3.27. Here, it is noticeable that for 1 robot, convergence to a final maximum episode reward is achieved around 4 times faster than the 4-Robots scenario; 1-Robot needing close to 1 million steps, and 4-Robots taking around 4 million steps (top right plot). Noticeably, the 4-Robots scenario allowed for finding faster ways to push the box to the target (see final values of top left plot), this suggests that there is a richer set of possible strategies that the four robots can follow, which increases the complexity of learning, but that when learned, makes the 4-Robots scenario less sensitive to the random starting conditions of the training episode.

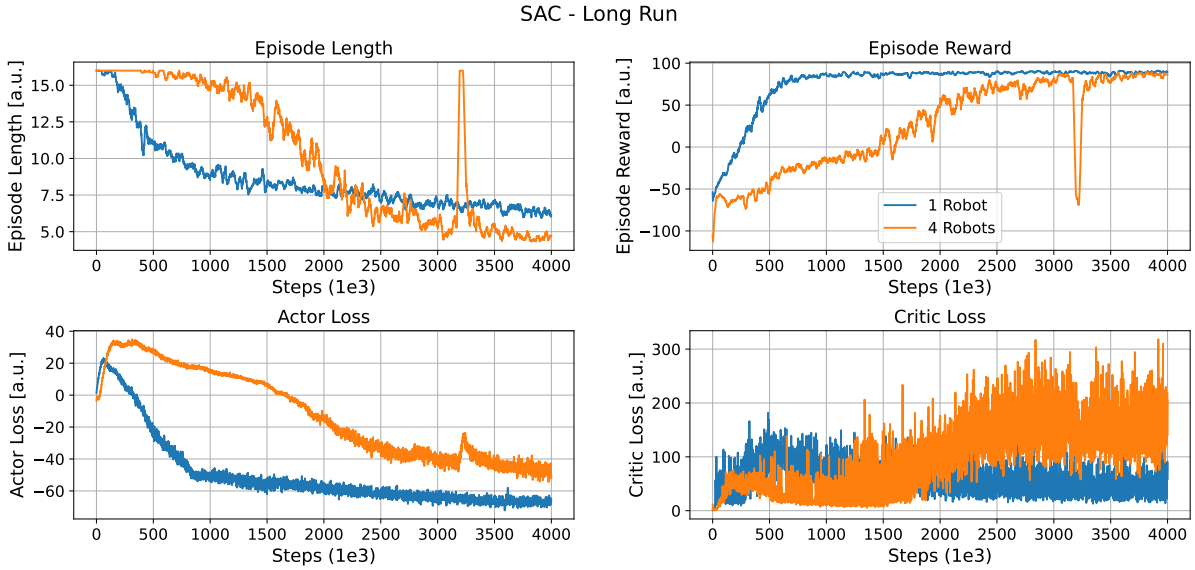


Figure 3.27: SAC long run (4 million steps) comparison between 1, and 4 robots.

DDPG shows a similar pattern to SAC (Figure 3.28). However, it should be noted that DDPG is very sensitive to the starting conditions, and it took several attempts to achieve DDPG scenarios where the algorithm successfully learns. It can also be observed that for 4 robots, DDPG is unable to find a successful control strategy, falling into a local minimum where it is capable of getting the robots close to the box, but not pushing the box successfully.

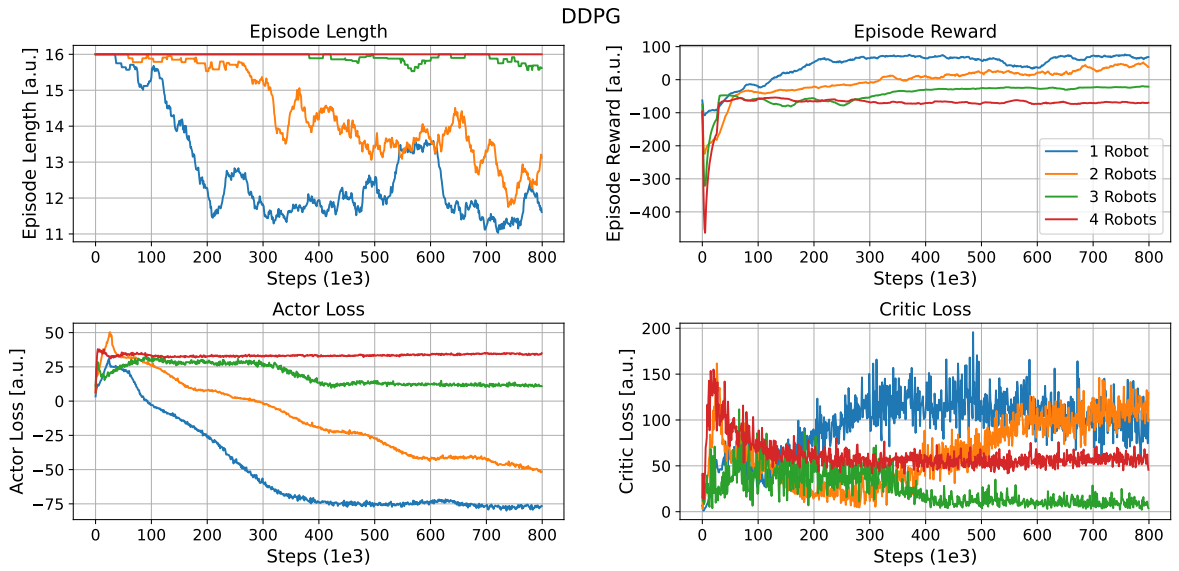


Figure 3.28: DDPG scalability. Learning performance is shown for 1, 2, 3, and 4 robots.

Lastly, PPO also shows deteriorating learning convergence as the number of robots increases (Figure 3.29). Also, as it was observed in Experiment I (Figure 3.24); even though PPO was found to be robust against the starting conditions of the learning episodes, and finds successful control strategies faster than the other algorithms, it has a tendency to catastrophically deteriorate in performance (see sudden ”jumps” in Figure 3.29).

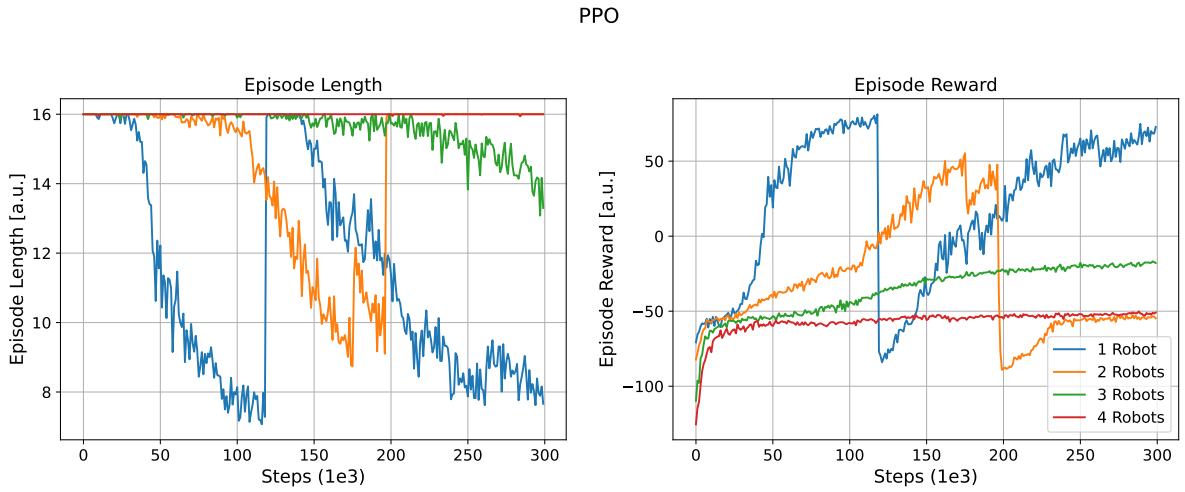


Figure 3.29: PPO scalability. Learning performance is shown for 1, 2, 3, and 4 robots.

Gathering the results from this experiment, it is clear that the push-box environment is challenging to solve because the robots need to learn the cooperation dynamics to push the box together. Although this might be relatively simple for 1 robot, adding more robots accrescents the difficulty of discovering a policy where the robots both get close to the box together, but they also coordinate to push the box where they want to.

This increase in complexity is reflected in the form of slower learning convergence, where the algorithms struggle to achieve successful policies for large numbers of robots. The centralised control architecture used in this experiment thus shows scalability limits that need to be improved for swarm space exploration, which further motivates the investigation of different MARL strategies.

3.4.5. Conclusion

This section implemented SAC, DDPG, and PPO on a collaborative push-box environment, were agents need to simultaneously push the box to a target location. Two experiments were made; changing the sparsity of the reward function, and scaling the number of agents in the environment.

It has been shown that dense rewards tend to help the learning, allowing for faster episode reward convergence.

Moreover, all three algorithms couldn't solve the environment with 5 agents, and the learning performance severely deteriorted as more agents were added, highlighting the poor scalability of this algorithms when applied with no modifications to accommodate for several agents, and the need for an algorithm which is specific for the multi-agent case.
From the three algorithms, DDPG showed the worst robustness, been highly sensitive to starting conditions. As it was described in Section 3.3.4, state-of-the-art MARL algorithms tend to use DDPG in their scheme, and thus, there might be potential for improving them using an equivalent scheme but with SAC, PPO or other algorithm instead [35].

Lastly, PPO was the fastest learning algorithm. However, it also has a tendency to catastrophically forget what was learned so far during the episode.

3.5. Preliminary Swarm System Selection

With the information gathered from Sections 3.1, 3.2, and 3.3, a preliminary swarm system can be designed. Since (to the best knowledge of the author) there doesn't seem to be a universally applicable swarm control type [18] with incorporated MARL, it is essential to determine which swarm system is suitable for planetary exploration, given the found requirements. With the studied literature and the performed analysis in Section 3.4 in the aforementioned referenced fields of space exploration, swarming, and reinforcement learning, this section showcases the swarm system designed for space exploration (Section 3.5.1), as well as the scope of its implementation for the remaining of this thesis work, in Section 3.5.2. Then, a top-level research plan is discussed in accordance with the aforementioned subsections, in Section 3.5.3. Lastly, the content of this section is summarised in Section 3.5.4, where Research Question 2 is further answered.

3.5.1. Proposed Swarm System

The proposed swarm system is shown in Figure 3.30. Here, the system is divided into abstraction levels, in the same format as the MaCMAS architecture (see Section 3.2.2), so that it is more traceable, and where requirements can be checked at different levels. **Notice that the different elements at each abstraction level can be moved with iterations in the design**, and that the blocks constituting each abstraction layer might not fully cover the required elements for all kinds of missions. The swarm system is also defined in this way because the different NASA requirements (see Section 3.1.3) can be mapped to each of the abstraction layers. It is important to notice that feedback iterations are also possible across layers. Also, this structure can be verified with the *AdaptiV* approach (Section 3.2.3). Lastly, having different layers in the system can help determine where to deploy MARL.
Although for academic purposes MARL can be deployed virtually everywhere in the system, this study tries to find the options where MARL can begin to be incorporated in such a swarm system with the idea that future real-life systems can be deployed with this technology.

The different levels of the proposed swarm system depicted in Figure 3.30 are explained using a simplified example of a swarm space exploration mission. Consider a swarm Martian mission concerning the exploration of a cave, a rocky region, and a vast flat region, where both terrestrial rovers and flying drones are available. When looking at the designed swarm system from the top layer (highest abstraction layer) to the bottom, several abstraction layers can be identified:

1. **Mission planning:** This entails assessing which parts of the main exploratory mission need to be completed, and together with the assessment of available resources (for example, the amount of available healthy rovers and drones), to establish a mission plan (exploring the flat region first might be less risky than going directly to the cave exploration, thus trying to minimise the overall mission risk, for instance). Such a system could be designed to function off-line (with predefined heuristics and algorithms established before launch) or online, actively learning environment characteristics that could not be accounted for beforehand.
2. **Task identification and allocation**: With a mission plan established, and with the available resources identified, the different steps in the mission plan can be identified and tasks be allocated (for example, if the flat region needs to be mapped, a team of drones might fly first through different defined targets to examine how rough the terrain is, and then this information can be used to generate a set of attainable points where a team of rovers can go and perform measurements). Then, the exploration task might need a certain construction or object manipulation, which needs to be assessed on whether a single agent or a combination of them is needed to achieve the task. This layer can be designed beforehand, but it is also possible to have some degree of adaptability that is learned during the mission. Moreover, it should specifically abide by **Reqs 4.1** and **4.2**.
3. **Cooperation structure:** Once the different tasks are identified and allocated, this set of activities might need the gathering of a combination of agents, or having a swarm of similar (homogeneous) agents perform the mission. This is the abstraction layer where robot teams are formed to achieve a specific goal. In the Martian example, a team of 5 drones might be chosen because they are likely to explore the target locations of the flat region faster than, say, a team of 3 drones. Moreover, specific robot agents might be chosen depending on their actual proximity to the target, and other parameters. Also, notice that hierarchical structures are possible. Furthermore, the maximum team size might be limited to a specific number to limit risks, for instance. This layer might be designed both on or

off-line, although learning from the success rates of different robot combinations might allow for a flexible assembly of robot teams.

4. **Communication structure:** Once the specific task is defined, together with the robot team that needs to do it, it is necessary to establish the communication structure of the team. This structure can be dependent on all the above layers (for example, the flat region exploration mission requires 5 drones that are organised hierarchically), and decides whether to use local communication between the team, communicate with the base, combine both strategies or in the case the mission falls beyond the communication range to the base (and such communication channel is needed), to establish a relay communication network. This layer is subject, among others, to **Reqs 3.1, 3.2, 3.3,** and **3.4**.
5. **Cooperation strategies:** With the established communication network and team structure, the robots need to find control strategies that allow them to perform the task. This is where the set of control strategies must find an emergent swarm behaviour. The guidance strategies can be included at this abstraction level, as well as establishing required attitude changes. In the example, the team of drones might find a benefit in achieving a specific fly formation that maximises stable communication, or that makes the team integrity more robust against unplanned gusts. Finding these control strategies is non-trivial, and although some actions, like flocking formation, are well understood, the discovery of policies that lead to more complex swarm behaviours is substantially difficult (see Section 3.3.2). This might require on-line adaptability of the swarm. This layer is subject, among others, to **Reqs 2.1, 1.4,** and **3.2**.
6. **Agent-level control:** Once each agent has a defined set of actions, it needs to execute them. This can be done at the local agent level, often involving classic control theory or well-known control methods; executing attitude control, collision avoidance, perception, navigation, measuring, etc. In the example, the drones need to adjust their attitude through the rotational speed of each of their rotors, follow a target location, perform a measurement, etc.

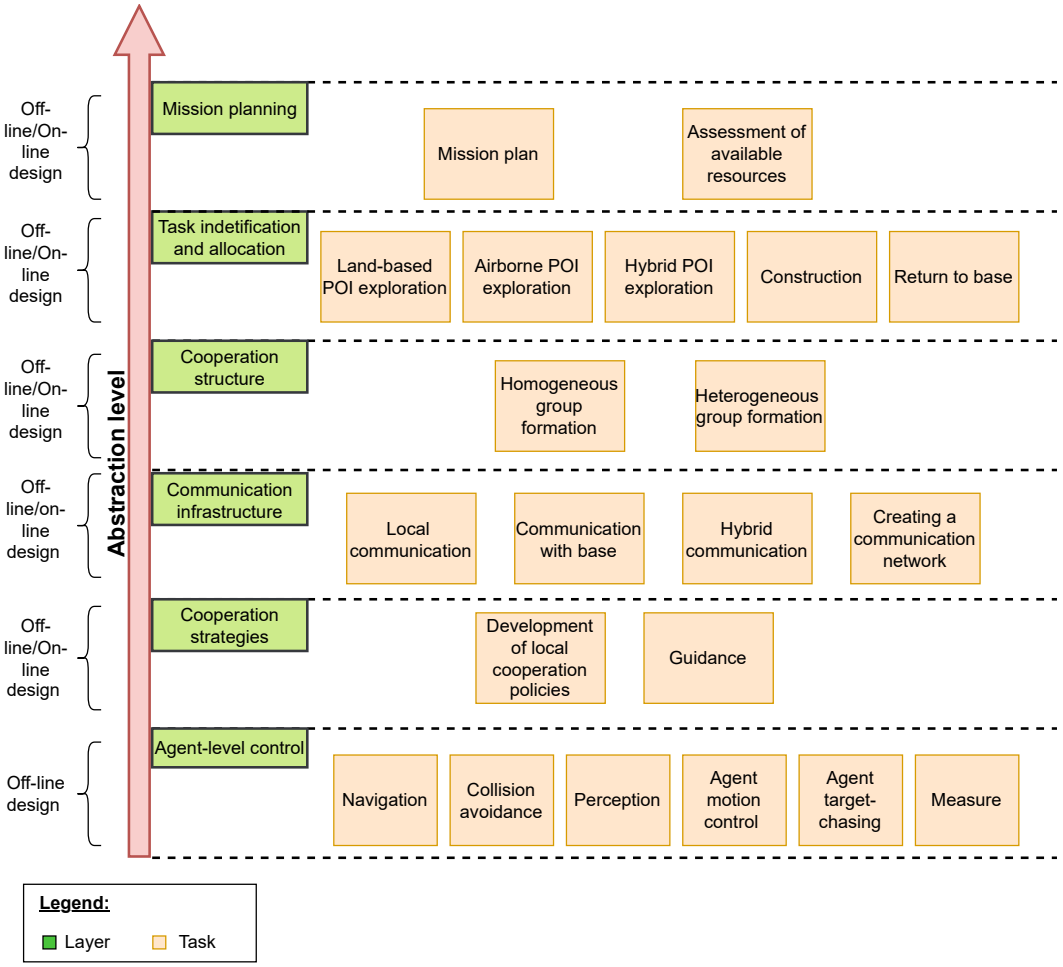


Figure 3.30: Proposed swarm system architecture.

3.5.2. Scope of Implementation

As previously mentioned in Section 3.5.1, MARL could potentially be applied to all the layers of the swarm system shown in Figure 3.30. However, this research aims at finding the first sensible place where MARL can be implemented. Of the six levels shown in the swarm system, **cooperation strategies** is identified as having great potential for MARL, given the hard problem of developing cooperation policies (Section 3.3.2), and, fittingly, the fact that MARL tries to find policies that can solve a certain multi-agent problem.
In this way, a specific mission task, with an identified multi-agent team and defined communication infrastructure (an input coming from the upper abstraction layers of the swarm system), can lead to a control problem that a MARL algorithm can solve.
This also allows for an algorithm implementation where design pivots are allowed based on performance and resources.

3.5.3. Research Plan

With the gathered information in this study, a research plan is established for the rest of the thesis:

1. Simulate a simplified multi-agent cooperative exploration scenario, where agents need to collectively explore a certain area. Developing the environment will provide the ground to further answer the Research Questions in the following steps.
2. Implement a MARL algorithm from Section 3.3.4, starting from a simplified implementation with a small number of homogeneous agents, to a more complicated environment with more agents. Testing the algorithm in the environment will further contribute to answering Research Questions 1 and 3.

3. Pivot the MARL algorithm or parts of it according to the findings of the previous step. If needed, the environment can be made simpler or more complex. Pivoting the algorithm design will further contribute to answering Research Questions 1, 2 and 3.
4. Once the algorithm is working, test its generalisation capabilities (injecting noise into the system, slightly modifying the dynamics of the agents, etc). This will further contribute to answering Research Questions 1, 2, and 3.
5. Verify the implemented algorithm, contributing to Research Question 4.
6. If the algorithm generalises well, simulate a more realistic environment, where the dynamics of each agent are more realistic, and observe whether the algorithm can generalise to that environment too. This can show that the algorithm might be able to generalise enough to be implemented in a real scenario (beyond the scope of this research), and serve as a form of validation. This will further help answer Research Questions 1, 2, and 3.

This plan is intended to be flexible, given that the feasibility of successfully implementing each of the steps in the allowable thesis time needs to be found experimentally.

Moreover, there are several ramifications of the research plan that can increase the quality of the research. If homogeneous agents scenarios are successfully learned, learning exploration with different agents can increase the contribution of the research towards swarm space exploration, given that different robot types could be combined. And (depending on the selected algorithm), implement other techniques to add robustness, such as self-play etc, and deepen the analysis of the function approximators in the MARL algorithm. Also, different communication architectures can be investigated, shifting from more centralised to fully decentralised scenarios. Finally, structural modifications to the selected MARL algorithm can lead to a novel method in the field.

3.5.4. Conclusion

With the gatherings of the literature research, this section proposes a swarm space exploration system constituted of different abstraction levels, similar to the NASA MaCMAS architecture, and which can later be verified through the *AdaptiV* method. Furthermore, the different levels of the system have been linked to the relevant NASA requirements.
From the different elements of this swarm system, the ”cooperation strategies” layer is selected to be fitting to implement a MARL algorithm to perform the required layer functions. Then, a top-level research plan is established, which identifies the main steps needed to complete the scope of this research, also having possible ramifications that can improve the research quality if resources allow.

3.6. Conclusion

This literature research aims to guide the MSc thesis efforts for positively contributing to the field of swarm space exploration. This is a broad field; shown to be constituted by the space exploration, swarming, and MARL disciplines.

Regarding space exploration, it has been found that there is a strong need in the field to achieve swarm technologies, since they can lead to unprecedented science exploration. This is highlighted by testaments from the Ingenuity mission and the future CADRE mission, among others.
Moreover, the technology readiness level of swarm systems is still very mature, especially regarding planetary exploration, chosen as the focus of this research. Additionally, the needed enabling technologies and requirements are identified to guide the design of a MARL algorithm later in the thesis.

To improve the understanding of the MARL traits that are needed, swarm architectures have been investigated, with MaCMAS being a structure suggested by NASA that can be used for the MARL algorithm to be designed. Also, the *AdaptiV* verification method has been investigated and chosen to verify the future MARL algorithm.

Then, reinforcement learning has been investigated, from the single-agent case to the relevant multi-agent scenario. Successful MARL methods must overcome the non-stationarity that occurs in the environment, have good scalability, and find an appropriate balance between stability of learning and adaptability. Also, the complexity of the MARL problem severely increases when going from standard Markov decision processes to partially observable ones that can be decentralised too.

To overcome these challenges, it has been observed that state-of-the-art MARL algorithms tend to use a centralised critic, decentralised actor approach, where the value function has access to the global state of the environment. Moreover, the scalability of these algorithms can be improved using embeddings of the global state vector with, for instance, attention techniques, or collective parameter updates of the individual actors and critics.
Although MARL specifically envisioned for space exploration has been investigated, there is still a large knowledge gap; where problems such as scalability and applicability need to be addressed.

From the experiments carried on a cooperative multi-agent environment, it was empirically found that dense reward functions can foster learning and that current state-of-the-art RL algorithms envisioned for single-agent RL have poor scalability. Moreover, DDPG was significantly less robust than SAC and PPO, and since it is the backbone of multiple modern MARL algorithms, there might be opportunities for improving the performance of MARL algorithms in this aspect.

Lastly, the information learned throughout the literature study has been combined to propose a swarm system architecture, in the fashion of MaCMAS, where different abstraction layers are identified, and where the aforementioned space requirements can be traced to.
Of all the layers, the remaining of this MSc thesis will focus on using MARL to find cooperation strategies of the swarm. A top-level research plan is created to accomplish this implementation; focusing on successfully applying the MARL algorithm to a simplified environment first, and then pivot towards more complex scenarios based on the empirical limitations, bottlenecks, and potential that will be found when implementing the algorithm.

From the initial broadness of the field of swarm space exploration, this literature study has narrowed down the scope of research such that meaningful results can be achieved in the remainder of the thesis.

Part III

Additional Results

4

Implementation Strategy

This thesis has followed the iterative design approach suggested in the research plan outlined at the end of the Literature Study (Section 3.5.3). The plan was intended to be flexible and pivot in favor of meaningful results, and trying to avoid bottlenecks or unnecessary delays. Additionally, all the environments and algorithms produced in this study are developed from the ground up and hence consume significant time resources to develop and verify, highlighting the need to carefully select how to properly pivot. For convenience, the research plan steps are listed below:

1. Simulate a simplified multi-agent cooperative exploration scenario, where agents need to collectively explore a certain area. Developing the environment will provide the ground to further answer the Research Questions in the following steps.
2. Implement a MARL algorithm from Section 3.3.4, starting from a simplified implementation with a small number of homogeneous agents, to a more complicated environment with more agents. Testing the algorithm in the environment will further contribute to answering Research Questions 1 and 3.
3. Pivot the MARL algorithm or parts of it according to the findings of the previous step. If needed, the environment can be made simpler or more complex. Pivoting the algorithm design will further contribute to answering Research Questions 1, 2, and 3.
4. Once the algorithm is working, test its generalization capabilities (injecting noise into the system, slightly modifying the dynamics of the agents, etc). This will further contribute to answering Research Questions 1, 2, and 3.
5. Verify the implemented algorithm, contributing to Research Question 4.
6. If the algorithm generalizes well, simulate a more realistic environment, where the dynamics of each agent are more realistic, and observe whether the algorithm can generalize to that environment too. This can show that the algorithm might be able to generalize enough to be implemented in a real scenario (beyond the scope of this research), and serve as a form of validation. This will further help answer Research Questions 1, 2, and 3.

In **step 1**, the environment development, no bottlenecks were found, as it was implemented and verified (see Section 7.1) having no specific delays.

In **step 2**, and according to the plan outlined after the literature research on MARL algorithms (see Section 3.3.6), MADDPG was implemented. Despite achieving good results in some works (see Section 3.3.4) the algorithm was found to have poor performance in the developed environment. MADDPG was first tested using only two agents and targets located close to the initial positions of the agents. Figure 4.1 shows the learning run on the environment. In this learning curve, the rewards achieved correspond to the behavior of having both agents running away from each other on the map, collecting targets that happened to be in the way. Despite learning for $40,000$ episodes, the algorithm fails to find more complex behaviors. For comparison, the final algorithm (MAPPO, see Part I) achieves the exploration of more than 50 times more targets using seven agents, which severely complicates the task (also when training for an equivalent number of steps). Other tests were attempted, where MADDPG also showed poor learning performance.

It is hypothesized that such poor performance is attributed to the nature of the used reward functions, since in this study, sparse reward functions are implemented (unlike the literature, where dense reward functions tend to be used). During the preliminary implementation of RL algorithms in the Literature Study (see Section 3.4), it was found that DDPG has brittle performance in scenarios with sparse rewards. Thus, it is hypothesized that MADDPG can potentially suffer from this lack of robustness in multi-agent settings too.

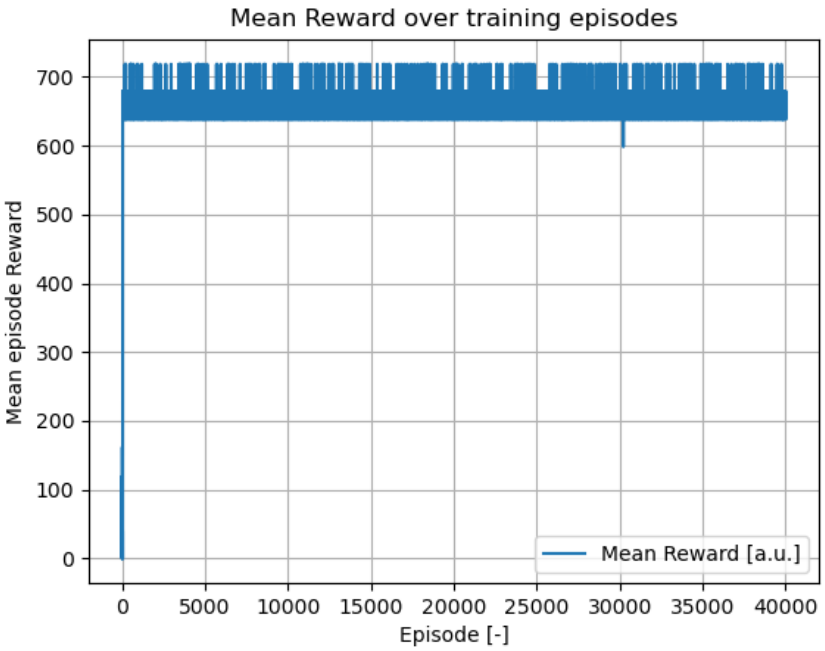


Figure 4.1: Learning performance of MADDPG in the developed environment. Two agents were used for this test.

After this finding, it was decided to pivot to the implementation of MAPPO. This agrees with **steps 2** and **3** in the research plan above. MAPPO was found to learn using simplified parameters in the algorithm (see Section 7.2).
From this, **step 4** in the research plan was executed, leading to the results shown in Part I. Also, **step 5** (verification), was executed during the development of the MAPPO algorithm.

In this way, the research plan was executed until its last point, **step 6**, which is ultimately considered beyond the scope of this work, since further improving the fidelity of the simulations will take time resources which will delay the thesis completion time.
Nevertheless, the research plan was executed successfully regarding its intended purpose; answering the Research Questions (see Chapter 8), and achieving state-of-the-art results through the development of a multi-agent PPO algorithm including the combination of novel techniques (see Part I).

5

Perception

Correctly sensing the environment is a critical aspect of MARL; both the policy and value (actor and critic) networks must have sufficient information to learn successful control strategies and value functions, as discussed in Chapter 3. It should be noted that in this study, when CTDE is used, the policy and value functions have access to different environment information, in other words, **they sense differently**; the critic receives some form of global information, whereas the actor is limited to perceive local information. This CTDE configuration for the multi-agent PPO is referred to as MAPPO (Multi-agent PPO), whereas not having CTDE (the actor and critic perceive the same environment information) is referred to as IPPO (Individual PPO).

A challenging aspect of perception when using artificial neural networks, is that the input provided to the ANNs needs to be of **constant size** when dealing with FF networks and CNNs (RNNs and other model structures such as transformers allow for variable input sizes, but are not treated in this study), and thus when the number of agents, targets and/or obstacles varies in the environment, the perception model needs to represent such changes as a tensor of constant size (for a given learning run).
Other approaches such as masking the input are possible but not robust when the number of features is larger than the designed size of the input tensor (for example, if the ANN is tasked to detect targets, and its input tensor is of size 100 x, y coordinates (thus allowing the detection of 100 targets), it can not cope with an environment containing more than 100 targets).

With this in mind, this section describes the naive approach to perception first tried in the environment in Section 5.1, and the improved actor and critic perceptions in Sections 5.2 and 5.3, respectively.

5.1. Naive Approach

The first iteration of the perception models uses the heuristic that information in the vicinity of the agent is more valuable than information coming from further away.
In this manner, the actor has access to the closest agent, target, and obstacle information, the input to the policy being:

$$\text{Actor} = [\Delta x_{\text{ closest agent}}, \Delta y_{\text{closest agent}}, \Delta x_{\text{closest target}}, \Delta y_{\text{closest target}}, \Delta x_{\text{closest obstacle}}, \Delta y_{\text{closest obstacle}}]^T \quad (5.1)$$

While the critic perceives the nearest target and obstacles, and, differently from the actor, has also access to the location of all agents in the environment. This results in the following input to the critic:

$$\text{Critic} = [\Delta x_{\text{all agents}}, \Delta y_{\text{all agents}}, \Delta x_{\text{closest target}}, \Delta y_{\text{closest target}}, \Delta x_{\text{closest obstacle}}, \Delta y_{\text{closest obstacle}}]^T \quad (5.2)$$

The perception tensors of Equations 5.1 and 5.2 are compact, and allow for using ANNs containing a reduced number of parameters. One noticeable issue (common in MARL) is the poor scalability of the critic's input as the number of agents increases, since for each new agent added to the training environment, two more elements must be added to the perception tensor (the 2D position of the agent).
Figure 5.1 shows a visual example of the information perceived by the actor and the critic. Noticeable is

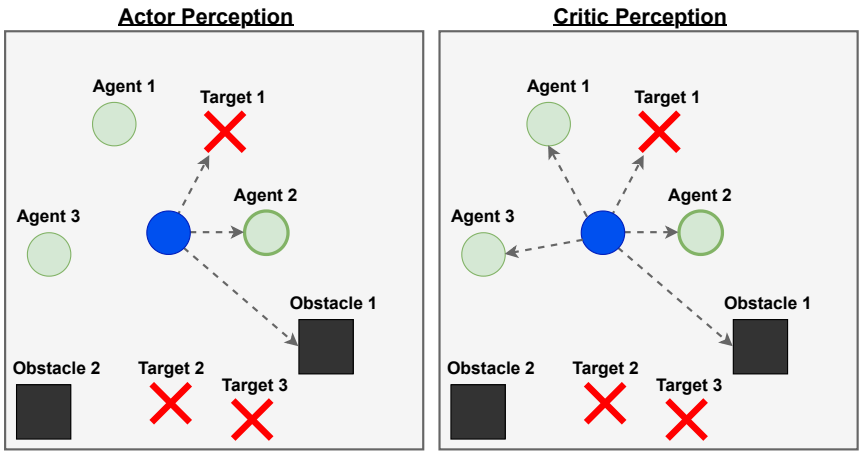


Figure 5.1: Example scenario where the environment elements detected by the main agent's actor and critic (in blue) are represented by arrows.

the actor and critic's inability to detect two targets close to one another that would result in higher rewards, and that the actor fails to perceive two other agents (1 and 2) that are also close by.

Nonetheless, this perception model allows for learning successful control policies, although it has several drawbacks:

- For the actor perception, there must **always** be a target, fellow agent, and obstacle in the environment, otherwise, this approach would need input masking.
- Similarly, the critic also needs the presence of the number of agents specified in the input tensor, and at least one target and one obstacle.
- Only detecting the closest environment features leads to situations in which the policy's perception oscillates between obstacles, agents, and/or targets; often leading to jittery control outputs when the agent is between two (or more) of such environment features. This can also lead to the agent getting stuck in a perpetual motion pattern.

All considered, this approach offers a simple solution to the perception problem, albeit being susceptible to robustness and awareness limitations. With this in mind, other perception models are investigated and ultimately used.

5.2. Reviewed Actor's Perception: LIDAR

To offer a more realistic perception model for the actor; one that can potentially be applied in real hardware and be more robust than the naive approach, a LIDAR-like perception model is developed.

To simulate the LIDAR, beams evenly "irradiate" from the agent, reaching a defined perception radius. Each beam contains three channels of information (corresponding to identifying agents, targets, and obstacles), detecting the position of the **closest** agent, target, and obstacle, or having as output the perception radius when nothing is detected in the beam. For the ANNs, these distance measurements are normalized. Figure 5.2 shows an example perception of the LIDAR in the environment, for one agent.

The number of beams in the LIDAR affects the perception quality. The minimum distance X of which an agent, target, or obstacle of radius R can be and have a chance of **not** being detected by the LIDAR (that is, it lies in between two beams) is defined by Equation 5.3. This way, the more beams the LIDAR has, the better the perception gets.
However, having more LIDAR beams comes at the cost of increasing the input tensor size to the ANNs with a factor $3 \times$ beams, and thus a trade-off needs to be made between perception quality and tensor size.

$$X = \frac{R}{tan(\frac{\pi}{\text{beams}})} \tag{5.3}$$

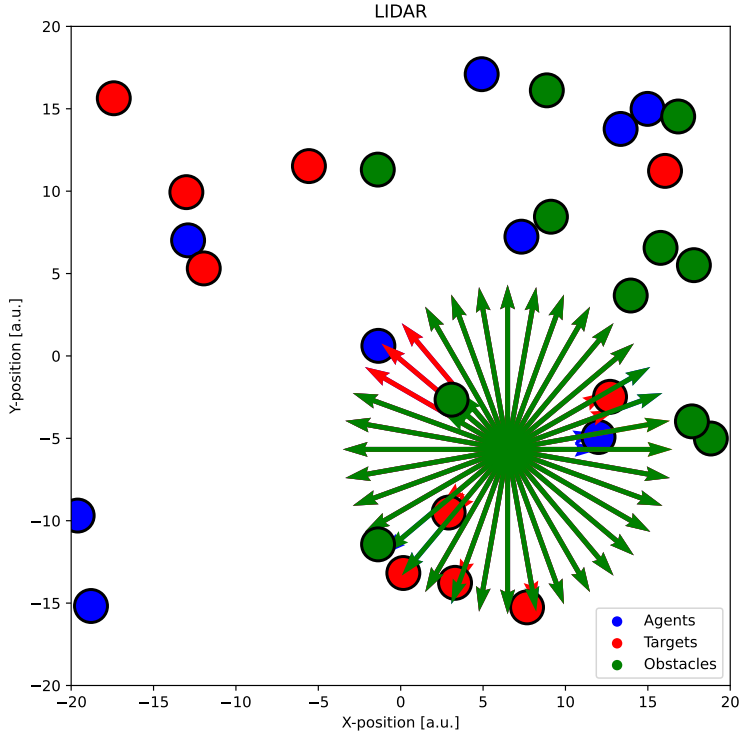


Figure 5.2: LIDAR perception of one agent in an example environment. The colored arrows represent the LIDAR beams, and their lengths (for each beam, the three arrows correspond to the detection of an agent, target, and/or obstacle). In this figure, 36 LIDAR beams are used, having a perception reach of 10 length units.

Having a LIDAR perception model successfully results in perceiving the environment such that successful policies can be learned (see Part I). In IPPO configuration, both the actor and the critic sense using the LIDAR, and a new approach using more information for the critic is needed for the MAPPO configuration. As an important remark, a feed-forward ANN is used to process this information and output the required actions.

5.3. Reviewed Critic's Perception: CNN Tensor

Regarding the critic perception when in CTDE, there are several options, such as using a concatenation of local agent observations, an environment-provided global state, using agent-specific information, etc. In [35] these configurations are studied, and their performance is analyzed, concluding that including both agent-specific features and global features in the value function input results in the best learning behavior. Considering this, the critic is designed to perceive a global environment tensor that also includes agent-specific information (see Figure 5.3). This tensor has three entries; entry i corresponds to the feature being extracted from the environment, and entries j and k to the horizontal and vertical coordinates of that feature space. In particular, four features are extracted from the environment: the position of all agents, targets, obstacles, and the self-position of the agent.

This way, the environment is discretized in a $M \times N$ grid where, for each environment feature, the grid cells in the tensor have a value of one if that feature is present, and zero otherwise.
To process this input and model the critic, a deep convolutional neural network (CNN) is used, which estimates the value of the state defined by the input tensor. Specifically, a CNN is chosen for this end since these models can extract features from spatial data (as has been verified in Section 7.2.1).

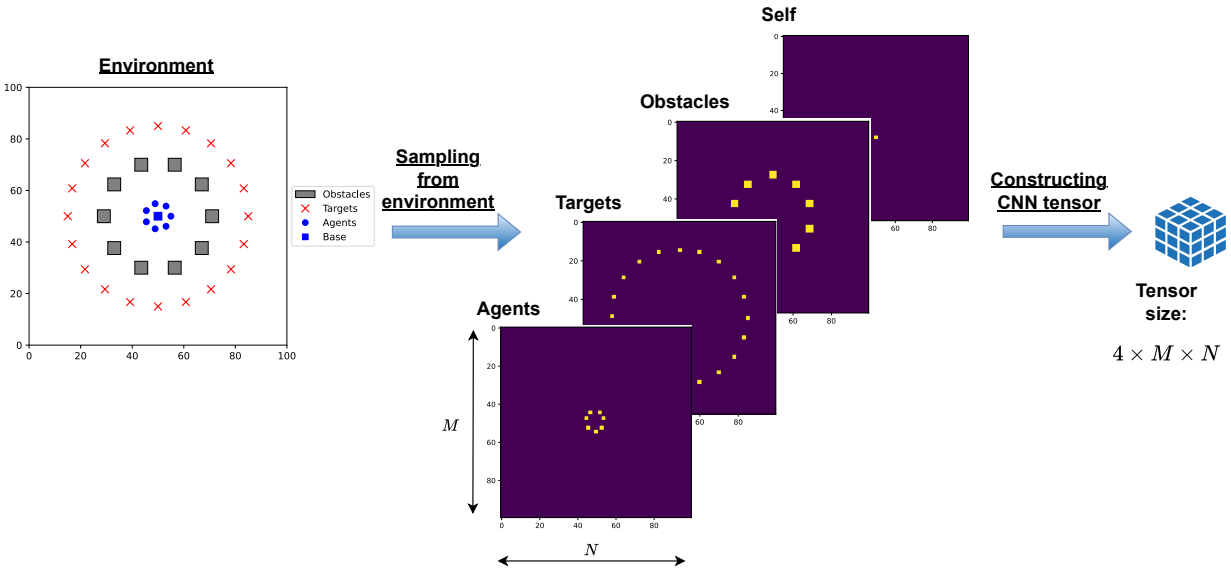


Figure 5.3: Schematic of the generation of the critic CNN perception tensor from the environment information.

6

Hyperparameter Tuning

A hyperparameter study is done to better understand the MARL problem and provide insights into how to further improve learning performance. It is important to realize that this algorithm has a great number of parameters; ranging from the ANN architectures and the PPO update scheme to the environment settings. Hence, due to computational resource constraints, the focus is to reduce the search space by studying parameters identified as highly relevant in multi-agent PPO algorithms [35] and explore some additional ones hypothesized to be important in this particular learning problem.
The findings from [35] are listed below:

- **Utilise value normalization to stabilize value learning**. In this sense, this research always normalizes the advantage calculations.
- **When available, include both local, agent-specific features and global features in the value function input. Also, check that these features do not unnecessarily increase the input dimension**. This is the salient difference between IPPO and MAPPO in this work; the former uses local information while the latter utilises a combination of agent-specific and global features.
- **Use at most 10 training epochs on difficult environments and 15 training epochs on easy environments. Additionally, avoid splitting data into mini-batches**.
- **For the best PPO performance, maintain a clipping ratio under 0.2; within this range, tune it as a trade-off between training stability and fast convergence**.
- **Utilize a large batch size to achieve best task performance with MAPPO. Then, tune the batch size to optimize for sample-efficiency**.

From these parameters, a random search optimization is performed using values in the proximity of the recommended settings from [35]; calculating the possible parameter permutations and sampling from them without replacement. Moreover, several parameter combinations are tried in parallel in the `DelftBlue`. Additionally, the variance of the exploration noise during training is also hypothesized to be of relevance and is included in this study, as is the learning rate of the ANNs. The different parameter values are shown in Table 6.1. It should be noted that the limitation on the upper range of the batch size is the memory constraint of the used GPUs (set by MAPPO) and that larger batch sizes could also be analyzed in different hardware.

Table 6.1: Investigated parameters.

Parameter	Values
Value function input	[IPPO, MAPPO]
Batch size	[3000, 3500, 4000, 4800]
Noise σ^2	[0.25, 0.5, 0.7]
Updates per batch	[5, 10, 15]
Clip value ϵ	[0.1, 0.2, 0.3]
Learning rate	[0.001, 0.003, 0.005]

A learning run is performed for each parameter combination, and the highest achieved reward during training is used as the performance metric (global rewards are used in this test, described in Part I). Moreover, the learning runs are simulated until reaching $14 \cdot 10^6$ experience tuple acquisitions (equivalent to training during $2 \cdot 10^4$ episodes), as this was found to surpass the initial transient behavior of the learning curve (see example plots in Part I), where the reward greatly increases at the beginning of the learning run (even though much longer training times are needed to achieve complex emergent behaviors). Moreover, 7 agents are used in the environment (the same number of agents used in Part I).

The test is split into the IPPO and MAPPO configurations. Figure 6.1 shows the parallel coordinate plot obtained with IPPO. Here, none of the parameters was found to have a strong correlation with the obtained rewards, except for the number of updates per batch, which has a weak correlation coefficient of 0.54 with the obtained rewards. Nevertheless, this metric is linear, and hence does not fully give insights into this analysis. In this sense, further parameter search is needed.
However, although patterns in the parameter configurations are difficult to obtain with this amount of exploration, it is salient that IPPO's performance greatly varies, going from rewards of 58 (barely exploring any targets) to 632 (exploring the majority of targets).

MAPPO has a different behavior compared with the observations above, shown in Figure 6.2. Although finding patterns in the parameter space is still challenging with this amount of search, the variance on the obtained reward is much smaller than IPPO's. Particularly, MAPPO has a variance of 5,483, while IPPO's is 48,966, hence differing by a factor close to 9 (which translates in MAPPO finding policies achieving strong target finding performance in 7 out of the 8 tried parameter combinations). When comparing the bounds in the reward values, MAPPO's smallest reward is 7 times larger than that of IPPO's, and its highest reward is also larger than IPPO's (672 vs 632).
Hence, the most salient gain in this analysis comes from using a combination of agent-specific and global features for the value function input, instead of using local information, further strengthening the suggested use of CTDE in multi-agent settings (see Part II).

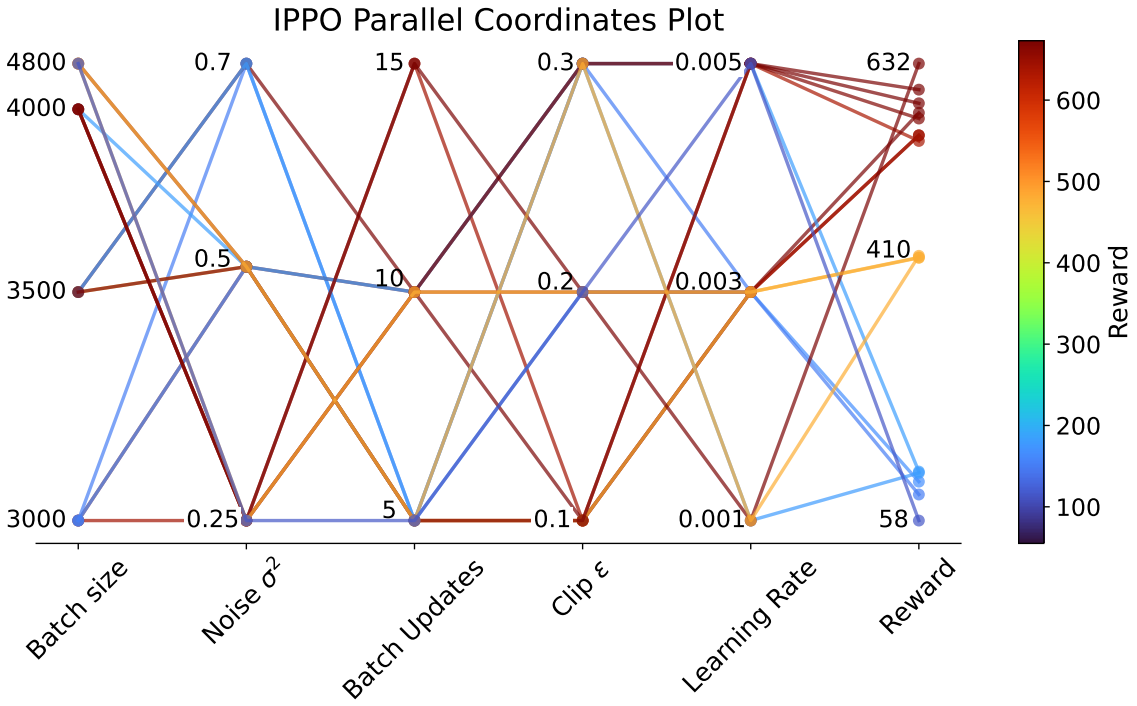


Figure 6.1: IPPO parallel coordinates plot obtained from the sweep of the selected hyperparameters. For each combination, $2 \cdot 10^4$ episodes are simulated (equivalent to collecting $14 \cdot 10^6$ experiences from the environment).

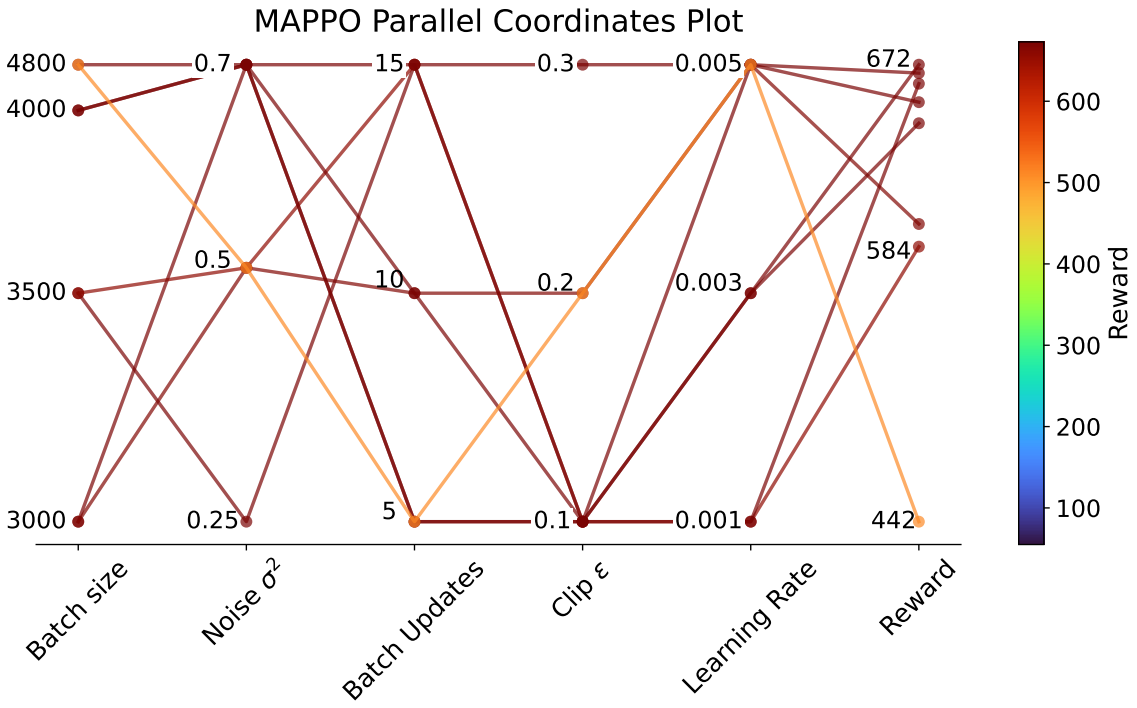


Figure 6.2: MAPPO parallel coordinates plot obtained from the sweep of the selected hyperparameters. For each combination, $2 \cdot 10^4$ episodes are simulated (equivalent to collecting $14 \cdot 10^6$ experiences from the environment).

7

Verification and Validation

Both the methodology and the software used in this thesis have been verified. Regarding the first, a gradual implementation approach has been adopted; first testing the learning algorithms in simplified settings to ensure that they can learn properly and not have fundamental limitations, and then using them to solve more complex problems (see Chapter 4).

Concerning the software, the thesis code is principally developed in Python, from scratch, to have a thorough understanding of the MARL problem, and the capacity to create bespoke algorithms for this research. Particularly, all the algorithms and simulations developed in this thesis only use the Python libraries of `Pytorch` and `NumPy`, and the visualizations use `Matplotlib`. Moreover, the code is flexible to run both on CPUs and GPUs. Specifically, the code has been primarily run in the `DelftBlue` supercomputer, using NVIDIA V100 GPUs.

Furthermore, the implemented code is modular; where top-level functionalities are defined and assigned to modules and/or scripts, followed by lower-level functionalities located within top-level ones. Also when the scripts are run as a standalone script (sometimes needing to import information from other scripts) a verification check is run and printed in the terminal, or shown graphically, allowing for sanity checks and unit testing.
While training, the code consists of two main modules; the environment, and the learning algorithm (see Figure 7.1). This top-level structure is similar to the classic phrasing of the RL learning scheme, and it easily allows for swift interface connections between the learning algorithms and the environment, permitting fast pivots between MADDPG, IPPO, and MAPPO. Moreover, the code is split into these two modules for unit and module testing and is further examined in Sections 7.1 and 7.2.

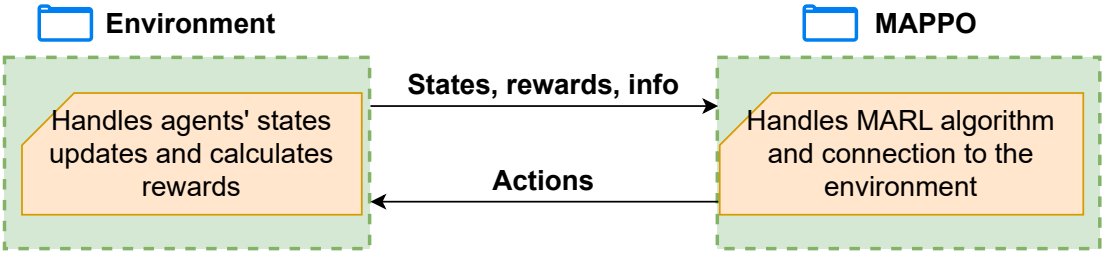


Figure 7.1: Top level structure of the implemented code.

7.1. Environment

The environment module is constituted by sub-modules implemented via Python scripts with clearly defined interfaces, as shown in Figure 7.2. In this way, each submodule is first verified before it is integrated with the rest of the module, and later, the module can be verified too.

In particular, the following checks have been carried out at a submodule level:

- `perception.py`: Testing the correct implementation of the LIDAR system: running numerical and visual inspections of the system with simple perception scenarios, and later testing more complicated

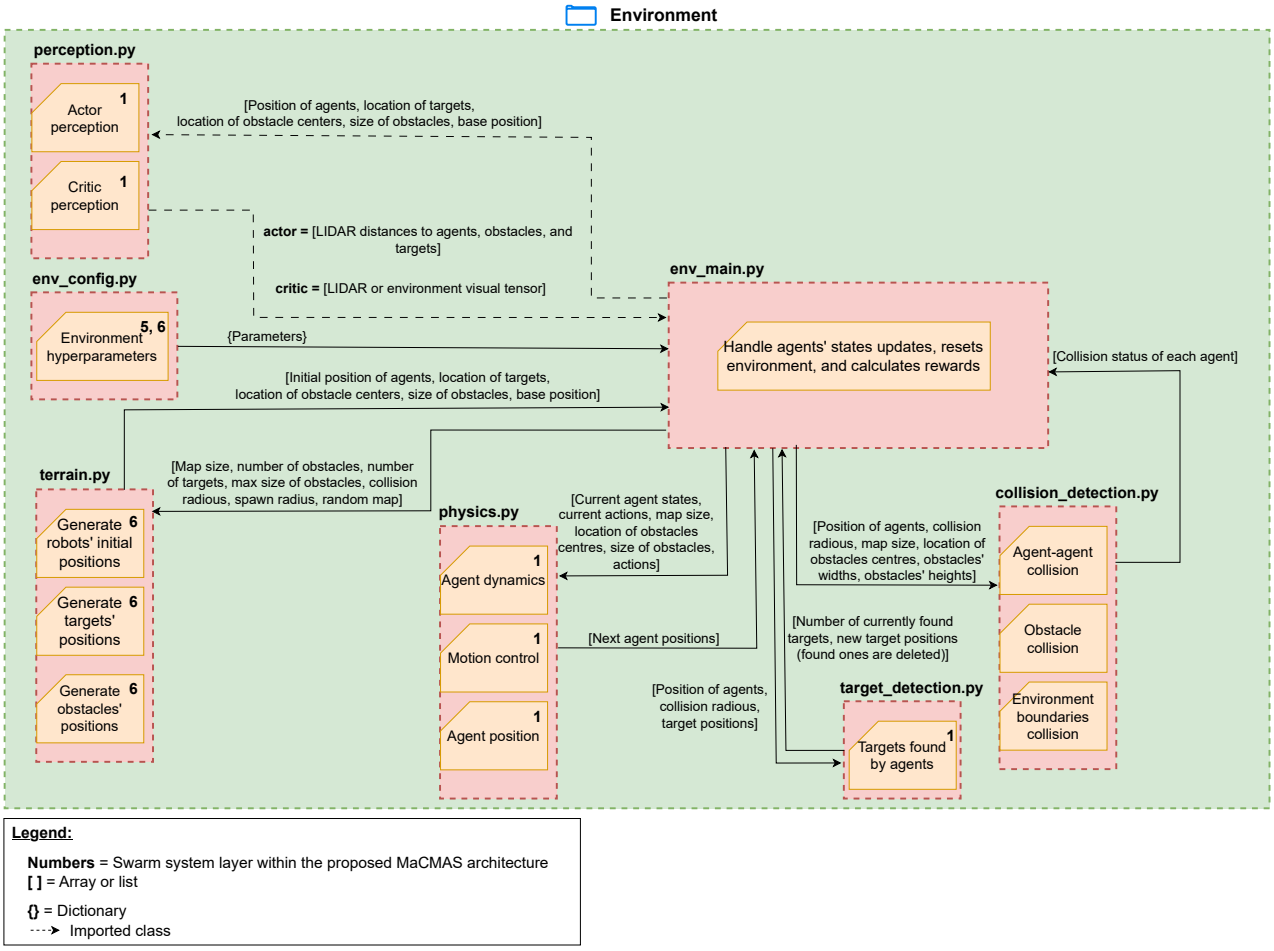


Figure 7.2: Modular code structure of the implemented environment code.

perception events. The CTDE critic tensor has been verified by checking the different entries of the generated tensor; ensuring that the selected resolution used in the thesis (100×100 pixels) has enough information such that the individual environment features can be identified, and that the spatial mapping between the environment and the tensor is correct.

- `env_config.py`: This script only contains a Python dictionary and the only needed test is the correct extraction of dictionary keys.
- `terrain.py`: Testing that the environment correctly generates positions and sizes from the environment parameters by numerical and visual inspection. Internal checks within the script are also tested, such as the requirement to generate at least two agents, and generating the agents' initial positions with sufficient spacing between them, as well as with the obstacles. There are also checks to ensure that the environment features are generated within the map bounds.
- `physics.py`: Testing the correct implementation of the dynamics by numerically and visually inspecting that the agents move according to the input action commands and that collisions with obstacles and map boundaries are correctly implemented.
- `target_detection.py`: Testing numerically that targets are removed from the map when they are inspected by an agent, that is, when the distance between the target and the agent is smaller than the detection radius specified in the environment parameters. The counting of the found targets with the correct assignation of the agent who found it has also been tested numerically.
- `collision_detection.py`: Similar to the previous script, numerical testing of this script has been carried out to ensure that whenever an agent collides with another agent or obstacle (when the proximity between the agent and the other agent, obstacle or map boundary is smaller than the

collision radius specified in the environment) this collision is correctly detected and assigned to the agent who collided.

After these verification tests were carried out at a submodule level, a module-level verification was done by assembling all the submodules in `env_main.py` and testing numerically and visually that full environment simulations are correct: testing the perception inputs, ensuring that the environment is designed according to its parameters, that the terrain layout is implemented correctly, and that the agents move according to their input commands; having correct dynamics and interaction with targets and obstacles. The correctness of the reward function was also tested numerically, inspecting the local, mixed, and global specifications. Lastly, the environment's capacity to reset after an episode is complete was also verified.

7.1.1. Runtime Analysis

A runtime test of the environment was analyzed to test the computational sensitivity of the environment with respect to its more salient parameters: the number of agents, obstacles, and targets. This is shown in Figure 7.3. The environment shows $\mathcal{O}(n^2)$, $\mathcal{O}(n)$, and $\mathcal{O}(log(n))$ complexity concerning the number of agents, obstacles, and targets, respectively. This resulted in acceptable runtimes during the learning runs done during the thesis (a bigger limiting factor is the data handling of the CTDE critic input, which is expensive in terms of memory). However, the time efficiency can be further optimized by improving the logic of the `physics.py`, `target_detection.py` and `collision_detection.py` submodules. Particularly, it is hypothesized that the runtime complexity of the targets is better than the obstacles one because it uses `NumPy` arrays logic instead of for loops. `NumPy` arrays outperform for loops due to their utilization of optimized `C` code, facilitating vectorized operations on entire arrays simultaneously. This approach minimizes the computational overhead associated with Python looping, leading to substantial performance enhancements. Hence future improvements could be achieved by modifying the for loops in the submodules mentioned above and replacing them with `NumPy` logic. The computational complexity concerning the number of agents is higher than that of the targets and obstacles because increasing the number of agents affects both the `target_detection.py` and `collision_detection.py` submodules (as well as `physics.py`).

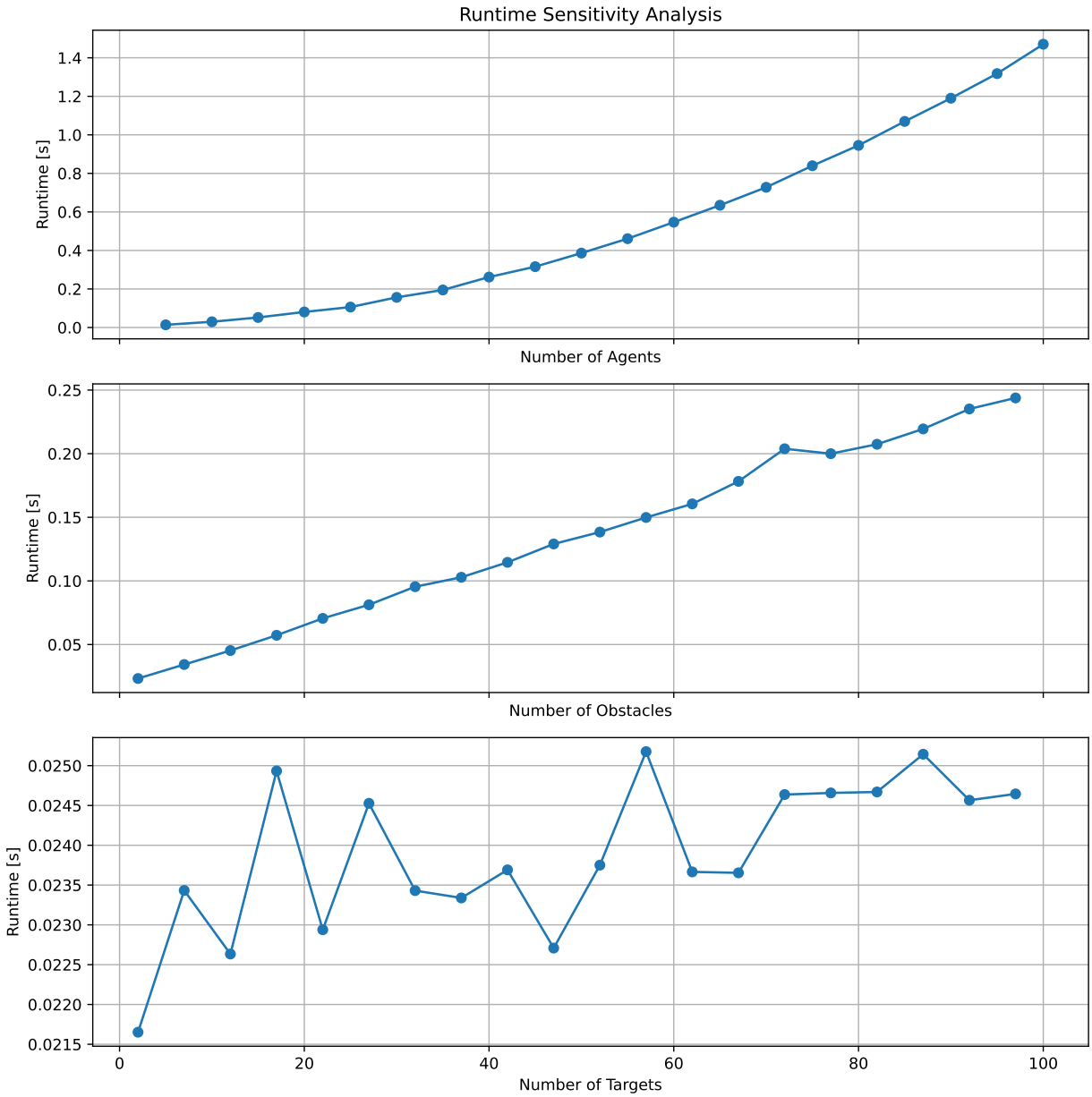


Figure 7.3: Runtime analysis of the environment (100 environment steps simulated per data point). The default parameters are 7 agents, 2 obstacles, and 2 targets. This test was run on a laptop CPU.

7.2. MAPPO/IPPO Learning Algorithm

Similar to the environment module, the learning algorithm module is constituted by submodules with clearly defined interfaces, as shown in Figure 7.4, and which are verified at a submodule level first, and then at a module level:

- `networks.py`: Testing the correct implementation of the artificial neural networks by performing input-output numerical tests; ensuring that the tensor shapes are correct and that the networks can run both on CPU and GPU. The CNN-specific test is discussed in Section 7.2.1.
- `logging` and `models`: These submodules store the saved models and learning progress. The submodules were verified by testing that the models were successfully saved during training in the specified folders.
- `GPU_job.sh`: The correct running of the code using the GPU's in `DelftBlue` was tested by submitting

simple jobs to the supercomputer and inspecting that the correct amount of resources (number of GPUs, CPUs, maximum runtime, etc) was assigned to the job.

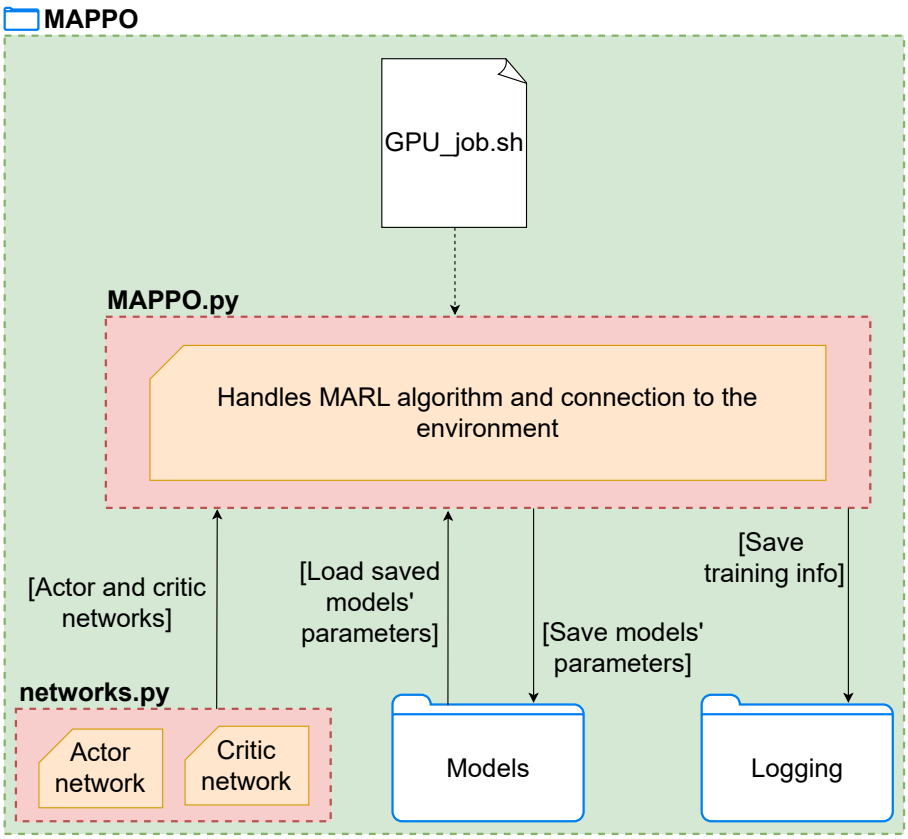


Figure 7.4: Modular code structure of the implemented MARL algorithm.

After the verification tests were done at a submodule level, a module-level verification was completed in `MAPPO.py`. Regarding integrating the different submodules, the correct import of the artificial neural networks and saving the learning progress were tested numerically, ensuring that the saving checkpoints worked as intended. From an algorithm standpoint, first, the algorithm was implemented as a single-agent PPO, and tested in the Gym Pendulum-v1 environment[1], as shown in Figure 7.5 to verify that the algorithm shows appropriate learning behavior.
Then, the algorithm was extended to its final MAPPO configuration, and the handling of the data specific to each agent in the environment (the correct assignment of observations, rewards, generated actions, etc) was tested numerically.

[1] https://www.gymlibrary.dev/environments/classic_control/pendulum/ [Visited on 14/04/2024]

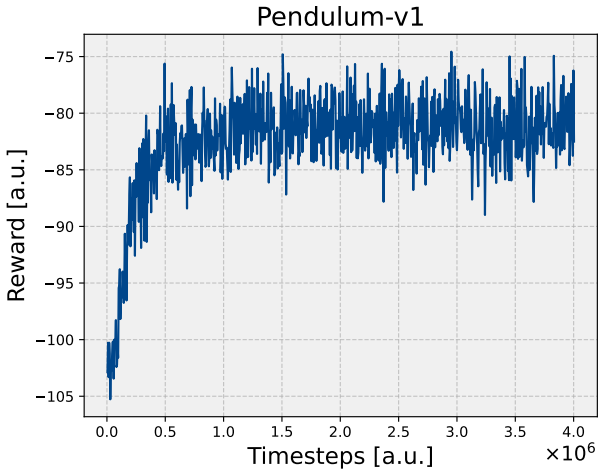


Figure 7.5: Learning run of the implemented algorithm in single-agent PPO format in the Gym Pendulum-v1 environment.

7.2.1. Critic's CNN, Supervised Learning Test

The critic's 2D CNN (see detailed architecture in Table A.2) is tested on the simpler task of detecting the location of a target within the environment to assess whether its architecture can successfully extract global environment information from the input tensor to determine position features of the agents, targets, and obstacles; useful for modeling the value function. 100 validation and 1,000 training samples are used for this test, where the target locations are generated with uniform distributions in the x, y axes, and are normalized. Notice that instead of having one output corresponding to the value of a specific global state (as is the case when implemented in the MARL algorithm) the CNN was modified to have two outputs for this test; corresponding to the x, y location of the target. The loss criterion is MSE between the real target location and the one generated by the CNN model.

As shown in Figure 7.6, the CNN successfully learns to identify the targets; rapidly decreasing the loss over training, and showing good performance on the validation data. This shows that this model can successfully extract environment position information (generalizing beyond the training data), and was used as the CNN critic model in this research.

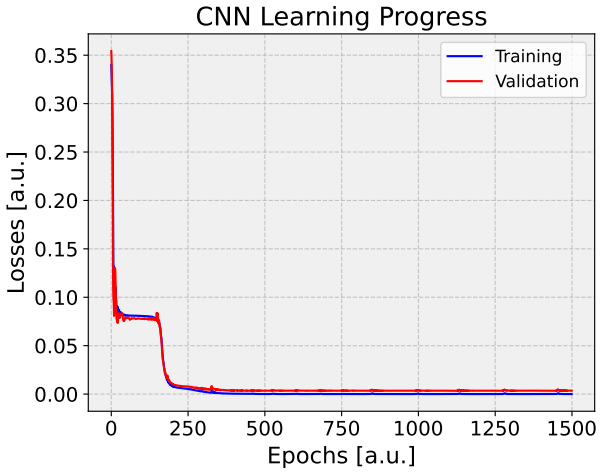


Figure 7.6: CNN learning progress; simple target detection.

7.3. Full Integration Verification

Once the environment and the learning algorithms were verified, a full system verification was performed by connecting the environment and MAPPO modules and performing a learning run, using environment parameters to simplify the learning problem (placing the targets close to the agents, using a small number of obstacles, etc).
The learned policy is simulated in Figure 7.7, which leads to the agents successfully exploring the targets. This test verified that the interface between the environment and algorithm modules is correct and that the MAPPO algorithm was correctly implemented to successfully learn multi-agent environments, as suggested in the literature (see Section 3.3.4).

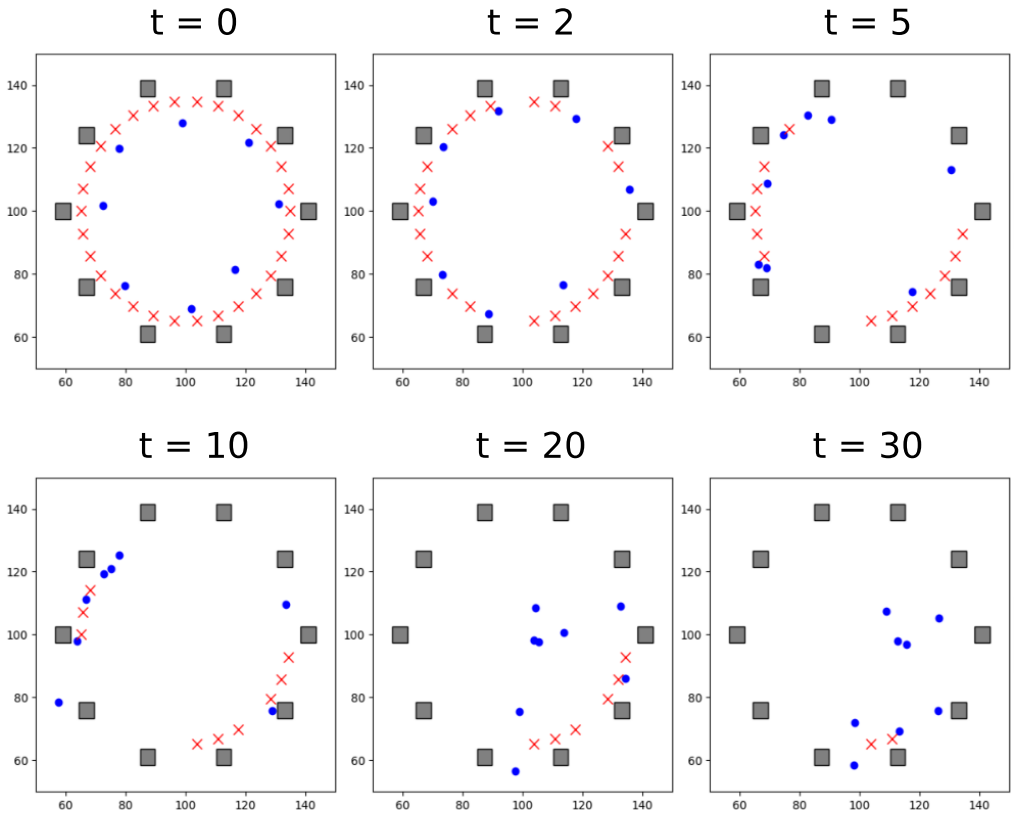


Figure 7.7: Trajectory trace of the learned policy for an environment with 8 agents.

7.4. Validation

Ultimately validating MARL for planetary exploration entails thoroughly testing the learned policies and proving they can successfully apply to real multi-agent systems; fulfilling the mission requirements. With this in mind, this work contributes by demonstrating that the policies obtained via MARL can (to some extent at this phase in the research) fulfill NASA requirements for swarm exploration missions (see Part I). However, although this is a significant step towards applying MARL in real exploration, it is not sufficient to fully validate the technology. Considering this, several recommendations are outlined for future validation efforts:

- Examining whether the approach to the learning problem is correct (whether it allows finding policies that are mission-applicable), including its underlying assumptions; using decentralized, partially observable MDPs, instant transition dynamics of the agents, no failures or performance degradation on the perception systems and the robots, squared shaped obstacles, no 3D terrain features, accurately identified targets, and accurate pose estimation of the agent with respect the other features in the environment.
- Deploying the learned policies in real robots, to investigate whether the policies learned in simulation can overcome the reality gap and achieve the behaviors obtained in simulation. This also serves to validate the environment simulation.
- Testing that the policies are robust to uncertainties that occur in real life, but may not be captured by the simulations.
- Confirm whether the real-life testing scenarios accurately represent the conditions found on other planets. This might need to be tested by gradually using the policies in real technology demonstration missions (similar to CADRE [3]) and/or using validated testing facilities (such as vacuum chambers, environments that mimic the conditions in the planetary body, etc).

These steps are necessary to make the technology mission-ready, but require significant resources involved with high-fidelity simulations, testing facilities, and overall hardware and human power. Hence they are considered out of the scope of this research but are strongly recommended for future investigation.

Part IV

Closure

8

Conclusion

This Chapter delves into the conclusions of this study, first discussing the closing remarks in Section 8.1, addressing the research questions in Section 8.2, and finally summarising the general contribution of this work.

8.1. Closing Remarks

This thesis has combined multi-agent reinforcement learning and swarm planetary exploration. In doing so, a state-of-the-art multi-agent extension of the Proximal Policy Optimisation algorithm has been developed, where the agents use a simulated LIDAR model to perceive their environment, and the value function can use a CNN tensor containing global and agent-specific environment information.

The thesis empirically shows that the proposed IPPO and MAPPO algorithms achieve better learning performance than the previous state-of-the-art in the field, which used MADDPG [22] and dense reward functions. Particularly, the algorithm can train with more agents simultaneously (7 vs 5, although it is possible that a higher agent number can be used), and achieve generalization capabilities: allowing for large swarms to be deployed after training (tested until reaching 20 agents), and use problem-specific perception models for the actor and the critic. Moreover, the reinforcement learning problem has been aligned with the field of planetary exploration, considering the requirements that can contribute to improving its technology readiness level.

Consequently, the IPPO and MAPPO configurations have been compared with reward functions that only account for agent-specific behavior, full swarm behavior, or a combination of the two. The IPPO configuration of the algorithm, using a LIDAR for the critic, achieves comparable performance to MAPPO when using local or mixed reward functions but, notoriously, MAPPO doubles IPPO's performance when using global reward settings, and achieves more stable learning in most scenarios.

Furthermore, **depending on the nature of the cooperative learning problem**, global reward functions are found to unnecessarily complicate the learning task, as there are scenarios where using local or mixed rewards also obtains high-performing cooperation policies (sometimes better than the ones obtained with global rewards).

Additionally, training in a rich environment achieves better generalization and scalability capabilities compared to always training in the same environment. This was achieved for all reward function types. This is a noticeable performance gain given that the rich environment algorithms trained for shorter times.

With the aforementioned findings, this thesis provides a framework to improve the technology readiness level of swarm missions, discovering policies that allow swarms to autonomously explore unknown areas, and contribute to NASA requirements. Hence, this work contributes toward making reinforcement learning an applicable technology in real swarm planetary exploration missions.

8.2. Research Questions

This research aimed at improving the technology readiness level of swarm space missions by developing a multi-agent reinforcement learning framework to discover swarm control policies. In particular, a multi-agent PPO algorithm has been developed to find swarm guidance policies. Ultimately proposing a swarm mission framework and developing this algorithm inside it has required extensive research in MARL and space exploration, and the culmination of all the mentioned efforts has helped answer the research questions within the scope of this work.

Answering Research Question 1

How can reinforcement learning be used to contribute to the autonomy of swarm space exploration missions?

Reinforcement learning offers a promising approach for enhancing the autonomy of swarm space exploration missions, given the low technology readiness levels (between 1 and 5) associated with such planetary missions, as depicted in Figure 3.10 and discussed in Section 3.1.2. This presents an opportune moment for RL to significantly contribute to the field. Section 3.1.3 has identified the specific technological requirements, exposing a notable technology gap where reinforcement learning can play a pivotal role. Particularly, reinforcement learning can be used to address the following mission types:

- Exploration, mapping, and sampling.
- Cooperative construction.
- Robust communication infrastructure.
- Cooperative computation.

Successfully applying reinforcement learning in such areas has the potential to increase the autonomy of swarm space exploration missions.

From a reinforcement learning perspective, RL can be used for policy discovery, a problem that can be phrased differently depending on the desired contribution to the swarm mission. Furthermore, a variety of algorithms have been developed in the field (MADDPG, MAPPO, MAAC, etc) to tackle multi-agent problems. Selecting among these algorithms offers a repertoire to find policies regarding different aspects of the swarm mission. This applicability is highlighted by the fact that NASA has already used MARL for Communication Infrastructure purposes [23], as described in Section 3.3.5.

Moreover, a multi-agent PPO algorithm has been developed in this research, demonstrating that reinforcement learning can help find complex swarm control strategies at a guidance level. These policies allow the swarm to autonomously explore by defining the coordinates of points of interest. In doing so, the learned policies improve the TRL of swarm planetary exploration missions, contributing to exploration, mapping and sampling, and cooperative task and task allocation, according to NASA requirements [2].

Answering Research Question 2

What are the main requirements of swarm space exploration missions?
The main requirements in the areas where reinforcement learning can be applied to contribute towards the technology development of swarm exploration missions have been identified in Section 3.1.3. These main requirements come from exploration, mapping and sampling, cooperative construction, communication infrastructure, and cooperative task and task allocation.

1. **Which swarm space exploration requirements can-not be satisfied with reinforcement learning given the available resources?**
   Among the considered requirements, the developed algorithm and environment did not address relative pose estimation and verbal inter-agent communication. If the problem needed to be solved to satisfy the mission requirements can be phrased as a DEC-POMDP, the algorithms studied in this work can potentially be applied to it.
2. **Among the different tasks of a space mission, upon which task(s) should this research be focused, given the suitability of reinforcement learning and the available resources?**
   The research is focused on Exploration, Mapping, and Sampling missions. Particularly in the development of local cooperation strategies, as discussed in Subsections 3.5.1 and 3.5.2.

Answering Research Question 3

Which reinforcement learning setup and algorithm(s) satisfy [swarm planetary exploration mission] requirements?
As discussed in Section 3.3.4, multi-agent reinforcement learning algorithms can be applied to swarm planetary missions, and the variety of available architectures offers flexibility regarding where to deploy MARL within the mission. In this sense, the algorithm selection can be specific to satisfy a given requirement, such as the need to have centralized or decentralized communication, etc.

1. **Which of the identified swarm-mission requirements are satisfied by existing MARL algorithms applied in this field?**
   The MARL algorithms and learning problem can be phrased to potentially satisfy Exploration, Mapping and Sampling, Cooperative Construction, Communication Infrastructure, and Cooperative Task and Task Allocation. In this research the developed MAPPO and IPPO algorithms have shown to contribute to Exploration, Mapping and Sampling, and Cooperative Task and Task Allocation.
2. **Which architecture/hierarchy is more suitable for swarm space exploration?**
   Several MARL structures are possible based on the state perception of the actor and critic, and the nature of the learning problem; centralized MARL, mean-field regime, team-average reward, decentralized networked agents, partial state observability scenarios, etc (discussed in Section 3.3.2). Decentralized policies allow for swarms with limited communication between the agents and the base, but finding such policies results in phrasing the problem as a DEC-POMDP, which is notoriously difficult to solve, and often needs specialized techniques such as CTDE, as implemented in the MAPPO algorithm developed in this work.
3. **What are the current state-of-the-art reinforcement learning algorithms applied to swarms?**
   There is a variety of MARL algorithms. One trend in the field is the use of actor-critic architectures. In this study, among the many different algorithm options, MADDPG, MAPPO, MAAC, FLDDPG, and hierarchical MARL can have the potential to be applied to a planetary exploration mission, and MADDPG has already been applied to swarm planetary exploration missions [22]. This research has shown that the MAPPO and IPPO algorithms can achieve better performance than MADDPG, using sparse reward functions.
4. **How should the value and policy functions be approximated?**
   There is a variety of possible function approximations. In particular fuzzy logic and deep artificial neural networks are studied (see Section 3.3.3). Among both, the latter is the standard option used in state-of-the-art MARL algorithms (see Section 3.3.4) and is thus considered in this research, where they have shown to be able to approximate the actor and critic functions such that emergent swarming behaviors are obtained.
5. **What are the limitations of using single-agent reinforcement learning algorithms such as SAC and PPO directly on swarms?**
   Single-agent reinforcement learning agents (DDPG, PPO, and SAC are experimentally tested in Section 3.4) have been found to achieve successful learning performance in a multi-agent push-box environment. However, their scalability to different amounts of agents is severely limited, with no algorithm achieving successful learning with more than four agents. Notoriously, DDPG achieves the worst learning performance, with PPO achieving faster learning, and SAC better agent scalability. Moreover, using single-agent reinforcement learning limits the problem to only have a centralised swarm policy, thus not allowing for situations with limited communication with a base, or similar.
6. **Which type of reward function promotes better learning (in terms of sample efficiency, or convergence capabilities)?**
   Dense reward functions have resulted in better learning according to the experiments carried out in Section 3.4.3 (in terms of sample efficiency, and convergence capabilities) for DDPG, PPO, and SAC. However, as discussed in Part I, sparse reward functions limit the amount of heuristics that need to be assumed about the solution to the multi-agent problem, and can be directly derived from mission requirements. Thus a trade-off can be made between the easiness of the learning problem, and the usefulness of the learned policy.

Answering Research Question 4

How can the selected reinforcement learning algorithm be verified?

1. **How can adaptive and swarm systems be verified?**
   Adaptive and swarm systems can be verified by the *AdaptiV* method [20], in particular, stability analysis, state space reduction, statistical verification, compositional verification, and monitorisation are suggested for this end. In this research, the implementation of the algorithms and simulated environment have been verified. A thorough implementation of the *AdaptiV* method has not been implemented.
2. **How can reinforcement learning systems for swarm exploration be verified?**
   Similar to the previous answer, reinforcement learning systems can be verified via the *AdaptiV* method, even when containing ANNs. Moreover, this research has verified the developed algorithms through submodule testing, module integration testing, testing the algorithm in a single-agent setting, and finally testing the algorithm in a learning problem within the developed multi-agent environment.

8.3. General Contribution

To the best knowledge of the author, this is the first work that combines rigorous swarm planetary mission requirements and MARL, paving the way for improving the technology's readiness. The flexibility of the assumptions used to construct the learning problem is a step forward in developing a learning algorithm that can be applied in a real swarm mission and MARL algorithms that can be utilized in harder multi-agent problems: having large swarm sizes, heterogeneous agents, and objectives other than obtaining guidance policies. Furthermore, this framework can be extended to other multi-agent systems due to the aforementioned flexibility in assumptions.

9

Recommendations

Although the present work has achieved notable successes, it is not without its limitations. This chapter offers a succinct overview of the key recommendations for the future advancement of this research project.

Further designing the swarm planetary mission
The proposed MaCMAS architecture is a simplified model of a full swarm mission. Having a more detailed design of such missions can further reveal the requirements that need to be satisfied, and the relevant metrics that need to be accounted for in the reward function.

Improving the environment
To bridge the reality gap between simulation and reality, more realistic environments can be used, where the low-level dynamics of the agents are more accurately simulated, as well as the inputs from the higher abstraction layers within the swarm mission architecture.

Improving the perception models
The actor and critic perceptions are critical for achieving successful learning behaviors. For the actor, more accurate perception models can be developed, such as using a more realistic LIDAR simulation and/or adding the reading of other sensors. Regarding the critic, other solutions can be explored, such as employing a combination of global and agent-specific environment information; using clustering techniques, modeling the actions of other agents, or similar heuristics.

Improving the actor and critic models
This study used a feed-forward neural network for the actor, and a CNN for the critic. These models offer poor flexibility regarding the size of the input, and in the case of the CNN, the generated input tensor is expensive in terms of memory, and rigid in terms of the spatial representation of the environment information. Learning performance can likely be enhanced by using models that have some form of time memory, such as the recurrent neural networks used in [35]. Moreover, in combination with improved perception models, having learning models that can selectively process the input information can be advantageous, using attention mechanisms to improve the scalability of the critic [28], or similar.

Improving the learning algorithm
Techniques such as using entropy or Kullback-Leibler divergence can potentially improve the training performance, offering a balance between exploration and exploitation. The usage of entropy will require using stochastic policies, however, and such a system might be harder to verify for planetary missions. Moreover it is hypothised that from a theoretical perspective, there might be structures within the parameter space of the learning models that reflect the emergence of patterns in behavioral space. Studying such behaviors can potentially lead to better understanding black-box models, and the nature of multi-agent systems.

References

[1] Zhong W. Thai et al. “Study of Swarm-based Planetary Exploration Architectures Using Agent-Based Modeling”. In: *AIAA Scitech 2020 Forum*. AIAA Scitech 2020 Forum. Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 6, 2020. DOI: 10.2514/6.2020-0075. URL: https://arc.aiaa.org/doi/10.2514/6.2020-0075 (visited on 05/29/2023).

[2] Amir Rahmani et al. “Space Vehicle Swarm Exploration Missions: A Study of Key Enabling Technologies and Gaps”. In: *70th International Astronautical Congress* (2019).

[3] Eric Vitug. *Cooperative Autonomous Distributed Robotic Exploration (CADRE)*. NASA. Feb. 5, 2021. URL: http://www.nasa.gov/directorates/spacetech/game_changing_development/projects/CADRE (visited on 07/26/2023).

[4] Emanuel Staudinger et al. “Swarm Technologies For Future Space Exploration Missions”. In: *14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-sairas)* (June 2018).

[5] NASA Jet Propulsion Laboratory. *CADRE Test Rovers in the Mars Yard*. Online. Mar. 2024. URL: https://www.jpl.nasa.gov/images/pia26168-cadre-test-rovers-in-the-mars-yard.

[6] Richard S. Sutton et al. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[7] W.F. Truszkowski et al. “Autonomous and autonomic systems: a paradigm for future space exploration missions”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 36.3 (May 2006). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), pp. 279–291. DOI: 10.1109/TSMCC.2006.871600.

[8] https://www.facebook.com/48576411181. *Mars Helicopter Is Much More Than a Tech Demo - IEEE Spectrum*. URL: https://spectrum.ieee.org/mars-perseverance (visited on 06/05/2023).

[9] Christopher Rouff. “Intelligence in Future NASA Swarm-based Missions”. In: *Papers from the 2007 AAAI Fall Symposium*. fall-2007-06. AAAI. Contents, Fall 2007.

[10] https://www.jpl.nasa.gov. *A-PUFFER*. NASA Jet Propulsion Laboratory (JPL). URL: https://www.jpl.nasa.gov/robotics-at-jpl/a-puffer (visited on 07/26/2023).

[11] Paolo Pirjanian et al. “CAMPOUT: A control architecture for multi-robot planetary outposts”. In: *Proceedings of SPIE - The International Society for Optical Engineering* 4196 (Oct. 16, 2000), pp. 221–230. DOI: 10.1117/12.403721.

[12] Loura Hall. *Marsbee - Swarm of Flapping Wing Flyers for Enhanced Mars Exploration*. NASA. Mar. 27, 2018. URL: http://www.nasa.gov/directorates/spacetech/niac/2018_Phase_I_Phase_II/Marsbee_Swarm_of_Flapping_Wing_Flyers_for_Enhanced_Mars_Exploration (visited on 07/26/2023).

[13] https://www.jpl.nasa.gov. *DuAxel*. NASA Jet Propulsion Laboratory (JPL). URL: https://www.jpl.nasa.gov/robotics-at-jpl/duaxel (visited on 07/26/2023).

[14] Bob Balaram et al. “Mars Helicopter Technology Demonstrator”. In: *NASA.gov* (Jan. 8, 2018). DOI: 10.2514/6.2018-0023.

[15] Manuel Grande et al. “Planetary Exploration Horizon 2061 – Report Chapter 5: Enabling technologies for planetary exploration”. In: *14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-sairas)* ().

[16] Joaquin Peña et al. “Modeling NASA swarm-based systems: using agent-oriented software engineering and formal methods”. In: *Software & Systems Modeling* 10.1 (Feb. 1, 2011), pp. 55–62. DOI:

10.1007/s10270-009-0135-2. URL: https://doi.org/10.1007/s10270-009-0135-2 (visited on 07/27/2023).

[17] Yang Liu et al. “Swarm Intelligence: Literature Overview”. In: *Dept. of Electrical Engineering, The Ohio State University* (Mar. 2000). Tel: (614)292-5716, fax: (614)292-7596, Email: liuya@ee.eng.ohio-state.edu, passino@ee.eng.ohio-state.edu.

[18] Todd Wareham et al. “Swarm Control for Distributed Construction: A Computational Complexity Perspective”. In: *ACM Transactions on Human-Robot Interaction* 12.1 (Mar. 31, 2023), pp. 1–45. DOI: 10.1145/3555078. URL: https://dl.acm.org/doi/10.1145/3555078 (visited on 05/30/2023).

[19] Kaiqing Zhang et al. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. Apr. 28, 2021. arXiv: 1911.10635[cs,stat]. URL: http://arxiv.org/abs/1911.10635 (visited on 08/28/2023).

[20] Christopher Rouff et al. “The *AdaptiV* approach to verification of adaptive systems”. In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. C3S2E ’12: Fifth International C* Conference on Computer Science & Software Engineering. Montreal Quebec Canada: ACM, June 27, 2012, pp. 118–122. DOI: 10.1145/2347583.2347600. URL: https://dl.acm.org/doi/10.1145/2347583.2347600 (visited on 07/28/2023).

[21] Juan José Garau Luis. “Robustness of Reinforcement Learning Systems in Real-World Environments”. PhD thesis. MIT Department of Aeronautics and Astronautics, July 2023.

[22] Yixin Huang et al. “A Multi-agent Reinforcement Learning Method for Swarm Robots in Space Collaborative Exploration”. In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. 2020 6th International Conference on Control, Automation and Robotics (ICCAR). ISSN: 2251-2446. Apr. 2020, pp. 139–144. DOI: 10.1109/ICCAR49639.2020.9107997.

[23] Rachel Dudukovich et al. “Towards the Development of a Multi-Agent Cognitive Networking System for the Lunar Environment”. In: *2021 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*. 2021 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE). Cleveland, OH, USA: IEEE, Oct. 12, 2021, pp. 7–13. DOI: 10.1109/WiSEE50203.2021.9613839. URL: https://ieeexplore.ieee.org/document/9613839/ (visited on 09/13/2023).

[24] Sergey Levine et al. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. Nov. 1, 2020. arXiv: 2005.01643[cs,stat]. URL: http://arxiv.org/abs/2005.01643 (visited on 08/09/2023).

[25] Lucian Buşoniu et al. “Multi-agent Reinforcement Learning: An Overview”. In: *Innovations in Multi-Agent Systems and Applications - 1*. Ed. by Dipti Srinivasan et al. Red. by Janusz Kacprzyk. Vol. 310. Series Title: Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–221. DOI: 10.1007/978-3-642-14435-6_7. URL: http://link.springer.com/10.1007/978-3-642-14435-6_7 (visited on 05/20/2023).

[26] Lucian Busoniu et al. “A Comprehensive Survey of Multiagent Reinforcement Learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (Mar. 2008), pp. 156–172. DOI: 10.1109/TSMCC.2007.913919. URL: https://ieeexplore.ieee.org/document/4445757/ (visited on 06/14/2023).

[27] Maximilian Hüttenrauch et al. “Deep Reinforcement Learning for Swarm Systems”. In: *CoRR* abs/1807.06613 (2018). arXiv: 1807.06613. URL: http://arxiv.org/abs/1807.06613.

[28] Ryan Lowe et al. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. Mar. 14, 2020. arXiv: 1706.02275[cs]. URL: http://arxiv.org/abs/1706.02275 (visited on 05/24/2023).

[29] Yaodong Yang et al. *An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective*. Mar. 17, 2021. arXiv: 2011.00583[cs]. URL: http://arxiv.org/abs/2011.00583 (visited on 05/26/2023).

[30] Daniel Bernstein et al. “The Complexity of Decentralized Control of Markov Decision Processes”. In: *Mathematics of Operations Research* 27 (Dec. 14, 2002). DOI: 10.1287/moor.27.4.819.297.

[31] Francois Chollet. *Deep Learning with Python*. 1st Edition. USA: Manning Publications Co., 2017.

[32] Adrián Menor de Oñate et al. *Algorithms for Tune Estimation and Damper Control*. CERN-ACC-NOTE 2023-0007. Submitted by paulina.samcova@cern.ch. Geneva: CERN, May 2023, p. 27.

[33] Shuzheng Qu et al. "An Adaptive Fuzzy Reinforcement Learning Cooperative Approach for the Autonomous Control of Flock Systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. May 30, 2021, pp. 8927–8933. DOI: 10.1109/ICRA48506.2021.9561204. arXiv: 2303.09946[cs,eess]. URL: http://arxiv.org/abs/2303.09946 (visited on 08/15/2023).

[34] Shariq Iqbal et al. "Actor-Attention-Critic for Multi-Agent Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, May 24, 2019, pp. 2961–2970. URL: https://proceedings.mlr.press/v97/iqbal19a.html (visited on 09/07/2023).

[35] Chao Yu et al. "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games". In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. Sept 2022. URL: https://openreview.net/forum?id=YVXaxB6L2Pl.

[36] Jiachen Yang et al. *Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery*. May 7, 2020. DOI: 10.48550/arXiv.1912.03558. arXiv: 1912.03558[cs,stat]. URL: http://arxiv.org/abs/1912.03558 (visited on 08/29/2023).

[37] Seongin Na et al. *Federated Reinforcement Learning for Collective Navigation of Robotic Swarms*. Sept. 11, 2022. arXiv: 2202.01141[cs]. URL: http://arxiv.org/abs/2202.01141 (visited on 09/11/2023).

[38] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. ISSN: 2640-3498. PMLR, Apr. 10, 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html (visited on 09/12/2023).

[39] Ashley Hill et al. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[40] Iou-Jen Liu et al. "Cooperative Exploration for Multi-Agent Deep Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 1, 2021, pp. 6826–6836. (Visited on 09/08/2023).

A

Artificial Neural Networks

For the actor and critic models in the algorithms used in this work, two kinds of ANNs are used; a feed-forward ANN to process the LIDAR data, and a CNN to process the environment tensor. The architectures of both ANNs are described in Tables A.1 and Table A.2, shown below.

Table A.1: Feed-forward ANN policy architecture.

Layer	Size	Number of Parameters
layer1.weight	64×108	$6,912$
layer1.bias	64	64
layer2.weight	64×64	$4,096$
layer2.bias	64	64
layer3.weight	2×64	128
layer3.bias	2	2
Total		$11,266$

Table A.2: 2D CNN critic network architecture. The displayed parameter sizes correspond to an input tensor of size $4 \times 100 \times 100$. Max pooling performed with kernel size 1×1 and stride 2. The convolution layers have stride 1, and 0 padding.

Layer	Output Size	Operation	Parameters
Conv1	16 channels	3×3 Conv, ReLU, Max Pooling	592
Conv2	32 channels	3×3 Conv, ReLU, Max Pooling	$4,640$
Conv3	64 channels	3×3 Conv, ReLU, Max Pooling	$18,496$
Conv4	128 channels	3×3 Conv, ReLU, Max Pooling	$73,856$
Conv5	256 channels	3×3 Conv, ReLU, Max Pooling	$295,168$
Flatten			
FC1	100 neurons	ReLU	$102,500$
FC2	30 neurons	ReLU	$3,030$
FC3	1 neuron		31
Total			$498,313$