UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUÍZA CAETANO GARAFFA

Evaluation of mapless navigation for unknown indoor environment exploration by single and multiple autonomous mobile robots using Deep Reinforcement Learning

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Edison Pignaton de Freitas

Porto Alegre
December 2022

CIP — CATALOGING-IN-PUBLICATION

Caetano Garaffa, Luíza

Evaluation of mapless navigation for unknown indoor environment exploration by single and multiple autonomous mobile robots using Deep Reinforcement Learning / Luíza Caetano Garaffa. – Porto Alegre: PPGC da UFRGS, 2022.

94 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Edison Pignaton de Freitas.

1. Mobile robotics. 2. Unknown environment exploration. 3. Deep Reinforcement Learning. 4. Single robot exploration. 5. Cooperative exploration. 6. Multi-Robot Systems. I. Pignaton de Freitas, Edison. II. Título.

AGRADECIMENTOS

Realizar o mestrado inteiro de forma remota e durante uma pandemia global foi, no mínimo, desafiador. Mas definitivamente foi uma experiência transformadora. E não posso deixar de agradecer a todos que fizeram parte dessa jornada, de perto ou de longe.

Ao meu orientador, Edison Pignaton de Freitas, por compartilhar sua experiência e por acreditar tanto nesse trabalho.

À pessoa que esteve ao meu lado todos os dias e que me apoiou nos momentos mais difíceis, Bruno Forlin. Sem ti, esse trabalho não seria possível.

Aos meus pais Ana e Adair, por nunca deixarem de acreditar em mim e por sempre me incetivarem a buscar novos desafios. À minha irmã, Juliana, por ser minha melhor amiga e fonte de alegria, mesmo à distância.

Aos amigos e família que foram fonte de calma e suporte em meio à insegurança.

Agradeço pela oportunidade de aprender e poder trabalhar em uma área com tanto poder de transformação como a Computação. Espero que eu possa, através de seu exercício, contribuir para uma sociedade melhor.

ABSTRACT

Efficient exploration of unknown environments is a fundamental requirement for modern autonomous mobile robot applications. The traditional exploration approach focuses on sensing the world to build a map and using the generated map to decide where to go next. In the specific case of collaborative exploration by multi-robot systems, map sharing and merging is often employed. Such methods tend to result in high computational costs, which can restrict their application in scenarios with limited memory and processing resources. An alternative to this is to employ mapless navigation when performing exploration. However, defining a resilient exploration strategy is not a straightforward task, especially in a mapless fashion. Concurrently, the employment of Deep Reinforcement Learning (DRL) has enabled optimal or near-optimal solutions for several complex problems with high-dimensional inputs. To the best of our knowledge, there are no works that investigate the application of DRL solutions for mapless exploration aiming at efficient area coverage, without pre-determined goal positions. In that context, this dissertation reviews recent research works that use RL to design unknown environment exploration strategies for single and multi-robots. Based on the gathered information, we propose an end-to-end mapless exploration framework based in DRL, suitable for single robots and teams of n robots. The exploration policy is trained and tested in different simulation environments. Our solution enabled exploration with efficiency comparable to DRL methods that use much more complex representations of the environment. The method also promoted cooperation between agents without the need of map merging algorithms, being able to generalize to different environments.

Keywords: Mobile robotics. Unknown environment exploration. Deep Reinforcement Learning. Single robot exploration. Cooperative exploration. Multi-Robot Systems.

Avaliação da navegação sem mapa para exploração de ambientes internos desconhecidos por robôs móveis autônomos usando Aprendizado por Reforço Profundo

RESUMO

A exploração eficiente de ambientes desconhecidos é uma condição fundamental para aplicações modernas de robôs móveis autônomos. A abordagem de exploração tradicional consiste em usar medidas de sensores para construir um mapa, e se basear no mapa gerado para decidir para onde ir. No caso específico da exploração colaborativa por sistemas multi-robôs, o compartilhamento e a fusão de mapas são frequentemente empregados. Tais métodos tendem a resultar em altos custos computacionais, o que pode restringir sua aplicação em cenários com recursos limitados de memória e processamento. Uma alternativa para isso é empregar a navegação sem mapa ao realizar a exploração. No entanto, definir o funcionamento de uma estratégia de exploração resiliente e apropriada não é uma tarefa simples, especialmente de forma sem geração de mapa. Ao mesmo tempo, o emprego do Aprendizado por Reforço Profundo (ARP) tem permitido soluções ótimas ou quase ótimas para vários problemas complexos com entradas de alta dimensionalidade. No entanto, até onde sabemos, não existem trabalhos que investiguem a aplicação de soluções DRL para exploração sem mapa visando a cobertura eficiente da área, sem posições alvo pré-determinadas. Nesse contexto, esta dissertação revisa pesquisas recentes que usam Aprendizado por Reforço para projetar estratégias de exploração de ambientes desconhecidos. Com base nas informações coletadas, propomos uma estrutura de exploração sem mapeamento de ponta a ponta baseada em ARP e adequada para n robôs. A política de exploração é treinada e testada em diferentes ambientes de simulação. Nossa solução permitiu a exploração com eficiência comparável aos métodos DRL que usam representações muito mais complexas do ambiente. O método também promoveu a cooperação entre os agentes sem a necessidade de algoritmos de fusão de mapas, podendo generalizar para diferentes ambientes.

Palavras-chave: Robótica móvel, Exploração de ambientes desconhecidos, Aprendizagem por Reforço Profundo, Exploração de um único robô, Exploração cooperativa, Sistemas Multi-Robôs.

LIST OF ABBREVIATIONS AND ACRONYMS

RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MRS	Multi-Robots System
PPO	Proximal Policy Optimization
LR	Learning Rate
UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
AUV	Autonomous Underwater Vehicle ML
SAR	Search and rescue
USAR	Urban search and rescue
ISR	Intelligence, surveillance and reconnaissance
AGV	Automated guided vehicles
SLAM	Simultaneous localization and mapping
GVG	Generalized Voronoi Graph
MDP	Markov decision process
POMDP	Partially observable Markov decision process
GPI	Generalized Policy Iteration
SAC	Soft Actor-Critic
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
TRPO	Trust Region Policy Optimization
SARSA	State-action-reward-state-action
DQN	Deep Q-Network
MC	Monte Carlo

NN	Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
OS-ELM	Online Sequential Extreme Learning Machine
D3QN	Deep double Q-network
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
TD3	Twin-delayed deep deterministic policy gradient
TD	Temporal Difference
CL	Curriculum Learning
GPS	Global Positioning System
EM	Exploration Module
CM	Communication Module
NM	Navigation Module
SS	Sensors Suite
DM	Decision-making module
IMU	Inertial Measurement Unit
EKF	Extended Kalman Filter
FIFO	First In First Out
AFCQN	Fully Convolutional Q-Network with Auxiliary task
BP	Back Propagation

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

In recent years, the advancement of new technologies applied to robotics has boosted the interest in academic research and practical application of mobile robots in different domains. Such applications include, for example, search and rescue (SAR) missions, intelligence, surveillance and reconnaissance (ISR), and planetary exploration, among others [Alatise e Hancke 2020]. Several situations require robotic autonomy when a map of the environment is not previously known. The employment of mobile robots in such complicated contexts depends on a robust and efficient exploration strategy. In the literature, mobile robotic exploration was described as the attempt to answer the question *"Given what you know about the world, where should you move to gain as much new information as possible?"* [Yamauchi 1997]. Exploratory behavior is a fundamental mobile robotic competence and represents a vast and complex research field. Several exploration methods were proposed in the past few decades, such as the Artificial Potential Fields [Krogh e Thorpe 1986] and the well-known Frontier-based exploration [Yamauchi 1997], enabling exploration for many applications.

1.1 Motivation

The traditional approach for autonomous mobile robotic exploration in unknown environments is to use sensor data to build a map of the agent's surrounding world. Then, the decision-making process regarding where to go next is based on the generated map. A limitation of this method is that the computational costs rapidly increase with the expansion of the explored area. Also, the efficiency of the exploration strategy heavily relies on precise maps [Juliá, Gil e Reinoso 2012]. Furthermore, when a multi-robot system performs a collaborative exploration, the maps generated by each agent are often shared between the team. The shared maps must usually be merged, which is not a trivial task, often associated with high computational costs [Velásquez Hernández e Prieto Ortiz 2020]. Such characteristics can restrict the application of these methods in scenarios where limited memory and processing resources are available.

Alternatively, if a robot could perform exploration without keeping an accurate map of the world, it would free memory and computational resources that could be used for faster decision making. However, developing an appropriate and resilient exploration strategy is not a straightforward task, especially in a mapless fashion. Concurrently, Reinforcement Learning techniques are based on letting the agent acquire skills through environment interaction instead of explicitly designing the desired behaviors.This Machine Learning (ML) paradigm tries to emulate the human learning process, which occurs through trial and error.

Recently, the employment of deep neural networks as powerful function approximators for RL algorithms (Deep Reinforcement Learning) has enabled optimal or near-optimal solutions for several complex problems with high-dimensional inputs [Wang et al. 2022]. Hence, RL and DRL are being highlighted as promising alternatives to develop solutions to robotics problems. It is no coincidence that the number of research works proposing solutions to the robotic exploration problem using RL algorithms has significantly increased in the past years [Garaffa et al. 2021]. In what concerns the mapless exploration problem, several works use DRL to navigate unknown environments aiming to reach a known target position. However, to the best of our knowledge, there are no works that investigate the application of DRL solutions for mapless exploration aiming at efficient area coverage, without pre-determined goal positions.

1.2 Research Goals and Contributions

The general objective of this dissertation is to investigate if Deep Reinforcement Learning can enable efficient and robust mapless exploration strategies for single and multiple robots. First, we review recent research works that use RL to design unknown environment exploration strategies for single and multi-robots. The review tries to compile the current state of research that links these two knowledge domains, aiming to comprehend how the two fields are being integrated. The study elaborated for this dissertation resulted in a survey published in the IEEE Transactions on Neural Networks and Learning Systems [Garaffa et al. 2021].

Based on the gathered information, we propose an end-to-end mapless exploration framework based in DRL and suitable for single robots and teams of n robots, which is the main contribution of this work. Through the proposed system, we evaluate if DRL is able to learn what actions result in an efficient environment coverage out of simplified information about the environment, such as laser measurements and odometry data. The proposed solution focuses on indoor environments and was designed to prioritize coverage efficiency and collision avoidance. Using a decentralized approach, each robot locally runs a system that determines the individual exploration policy. The idea is to obtain agents capable of exploring the environment alone, but able to optimize their decision-making process if information from other agents is received. The basis of the decision-making process is the Proximal Policy Optimization (PPO) algorithm, and deep artificial neural networks are employed as the function approximators. The exploration policy is trained and tested in different simulation environments, and the results are compared with other exploration methods.

1.3 Dissertation Overview

This dissertation is organized as follows: Chapter 2 contains a brief overview of the exploration problem in the field of mobile robotics. Chapter 3 presents background on Reinforcement Learning, including the definition of RL problem and the most traditional methods. It dives into more details into the Proximal Policy Optimization algorithm, used in this work. Chapter 4 reviews research works that combine mobile robotic exploration with Reinforcement Learning solutions. Then, the proposed architecture for mapless co-operative exploration is described in 5. The simulation environments and the experiments to train, validate and test the method in different conditions are formulated in 6. Chapter 7 presents the experiments' results and a discussion about the exploration strategy performance, strong points, and limitations. Finally, Chapter 8 presents our conclusions and the suggested future work.

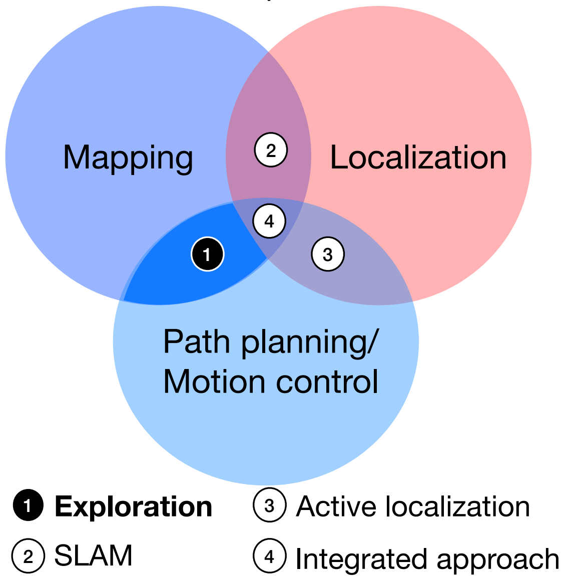
2 EXPLORATION IN MOBILE ROBOTICS

A mobile robot is a robotic platform that is able to move through an environment using locomotive elements (e.g. wheels, propellers, legs) [Tzafestas 2013]. In the 1950s, mobile robots were first introduced in industrial production processes, being mainly automated guided vehicles (AGVs) that followed a predefined trajectory in order to transport tools [Li, Yan e Li 2018]. Nowadays, however, mobile robots are operating in unstructured and dynamic environments, being employed in an increasing number of applications as medical care, personal services, planetary exploration, search and rescue operations, construction, subaquatical or aerial operations, entertainment, surveillance, among others [Garcia et al. 2007]. This kind of application demand autonomous systems capable of choosing appropriate actions from its perception and interaction with the environment. Hence, the robotics research field is increasingly focusing on the development of robust software solutions that enable robots to autonomously transpose the challenges arising from unknown and dynamic environments [Thrun, Burgard e Fox 2005].

One of the biggest challenges in developing autonomous robots is the *navigation problem* [Makarenko et al. 2002]. Navigation comprehends the robot's ability to select and perform actions based on its knowledge and sensor values, aiming to reach its goal positions reliably. In unknown environments, the robot has to learn how to navigate without an initial map or model. Ideally, an autonomous agent starts from an arbitrary position and explores the environment, while simultaneously collecting information about the surrounding world, building an appropriate map or model, and localizing itself on this map [Siegwart e Nourbakhsh 2004]. As illustrated in Figure 2.1, different elements compose a robot navigation strategy. The three basic navigation competences - mapping, localization, and path planning, compose the Simultaneous Localization and Mapping (SLAM) and active localization tasks, as well as the integrated approach. Each of these competences represents a vast and complex research area. However, this work focuses on an imperative aspect for mobile robot's autonomy: the exploratory behavior.

Autonomous exploration encompasses the ability of autonomously moving through an unknown environment, while collecting the necessary measurements and information in order to accomplish a pre-defined goal. In the literature, mobile robotic exploration was described as the attempt to answer the question *"Given what you know about the world, where should you move to gain as much new information as possible?"* [Yamauchi 1997]. The most common problems solved by the exploratory planning are map acqui-

Figure 2.1 – Competences for robotic navigation, with highlight for the exploration task.



Adapted from: [Makarenko et al. 2002].

sition (non-guided task) and single and multi targets identification (guided task). Is the exploratory behavior that makes human guiding or pre-planed trajectories unnecessary, being especially important in hostile or inaccessible environments (e.g Urban Search and Rescue (USAR) in post-disaster scenes, planetary exploration, Intelligence, Surveillance and Reconnaissance (ISR)) [Lluvia, Lazkano e Ansuategi 2021].

In a generic exploration algorithm, the agent collects information about the environment in its current state and uses this information to decide its next goal position or next movement. This process is repeated until the exploration goal is accomplished [Nehmzow 2012]. However, the strategies are widely dependent on the system application, and the techniques employed in each step of the process must be decided considering the application specifications. Some situations require a high quality map, some focus on robustness, while for others a reduced operation time is essential. Therefore, composing an exploration strategy is not a simple, straightforward process and there is not an ideal approach for every specific application. Instead, different methods combinations and approaches for composing a mobile robotic exploration strategy have been proposed and investigated over the years.

The exploration methods can compose robotic architectures classified according to the three paradigms for organizing intelligence in robotics: reactive, deliberative, and hybrid [Mohr et al. 2014]. **Reactive strategies** are behavior-based, which means that the robot exhibits behaviors as a reaction to events. Also, there is no planning stage, and the agent does not learn with experience: the defined event-reaction pairs remain fixed during the exploration [Arkin e Arkin 1998]. A simple example of a reactive approach that can be employed for exploration is *random search*. Statistics show that after some time, moving using random potential fields makes the robot cover an entire area [Chupeau, Bénichou e Voituriez 2015]. Other behaviors commonly employed by reactive strategies are *Avoid Obstacles* [De Silva e Ekanayake 2008], *Avoid Past* [Balch 1993] and *Follow Walls* [Ando e Yuta 1995]. More complex reactive architectures must be able to handle multiple concurrent behaviors. In this context, the two main examples of proposed and extensively investigated methods are the Subsumption architecture [Brooks 1987] and the Potential Fields Methodology [Krogh e Thorpe 1986]. The reactive paradigm enables an important range of behaviors while being easy to implement and presenting a fast execution time. However, the absence of learning and planning abilities, and problems like local Minima, make purely reactive methods usually inefficient for complex applications.

The exploration strategies can also be a part of control architectures based on a **Deliberative approach**. Deliberative strategies (also called hierarchical), are based on a fixed events sequence: the agent senses the world, plans its actions, and then acts. Through this process, the robot creates a global world model, which contains all the information employed in the planning phase. The deliberative is the oldest of the three paradigms and was vastly adopted between 1967 and 1990. However, the planning stage is very heavy in terms of processing and execution time, making it difficult to respond appropriately to unexpected changes in the the environment [Lazzeri et al. 2018].

Finally, there are the strategies that employ an **Hybrid approach**. The hybrid paradigm is the most recent one, proposed in 1990, and has been the main focus of current research [Mohr et al. 2014]. It combines the desirable characteristics of both deliberative and reactive approaches: planning capability and fast response time, respectively. Thus, the robot is able to perform long-term planning based on the world model (which is necessary for performing more elaborated tasks), while being able to deal with the environment unpredictability. The great majority of current exploration methods fit into the Hybrid paradigm.

Some successful exploration methods divide the world representation and rank the resulting regions in order to make logical decisions. One example is employing clustering algorithms such as K-means [Solanas e Garcia 2004] or spectral clustering [Kaleci et al. 2015] to divide the map into different regions. Other methods use Generalized Voronoi Graphs (GVG) [Park e Roh 2016] to represent the world, so that the agent attempts to stay equidistant to surrounding obstacles while moving. However, the most widespread method is the Frontier-based approach, proposed by Yamauchi *et al.* in 1997 [Yamauchi 1997]. Frontiers are the boundaries between open space that has been sensed and unexplored space, and the robot moves to new frontiers until the exploration process is over. The frontier-based approach has been widely studied, adapted, and combined with other techniques to perform exploration [Jain, Tiwari e Godfrey 2017].

A crucial part of exploration strategies that include a planning stage, whether it divides and rank regions or not, is deciding *where to go*. The autonomous exploration problem can be described as the *travelling salesman problem*, in which the agent must plan the order to visit the remaining unexplored regions while minimizing the total traveled distance [Adler e Karaman 2016]. Some frontier-based methods define interest attributes to decide the next target region (e.g., nearest, farthest, most extensive frontier). A common alternative is to use a cost-utility model to select destination positions that are not too costly in terms of time and distance while providing relevant information about the environment. One case is the Next-Best-View selection algorithms, which adopt scoring systems based on a utility function to select the appropriate waypoint candidate [Wang et al. 2020]. It is also possible to model the exploration problem using Markov Decision Process (MDP) or a Partially Observable Markov Decision Process (POMDP), that can be solved, for example, with dynamic programming algorithms, value iteration, or Reinforcement Learning (RL). Both (PO)MDP and RL are further discussed in the following sections. It is important to mention that these are not necessarily excluding methods, but instead can be combined to compose an exploration strategy.

3 REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a computational approach inspired by the biological learning process of animals. The nature of learning is to interact with the environment and gain knowledge about cause and effect from the sensory feedback received as a consequence of actions. Based on that, Reinforcement Learning consists in learning desired behaviors through interactions with the environment, without explicit examples or external instructors. Through trial-and-error, the agent seeks to learn an optimal policy (π^*) or, in other words, learn how to optimally map situations into actions, by maximizing a numerical reward signal [Puterman 2014]. Reinforcement Learning is considered the third Machine Learning paradigm, alongside supervised learning and unsupervised learning.

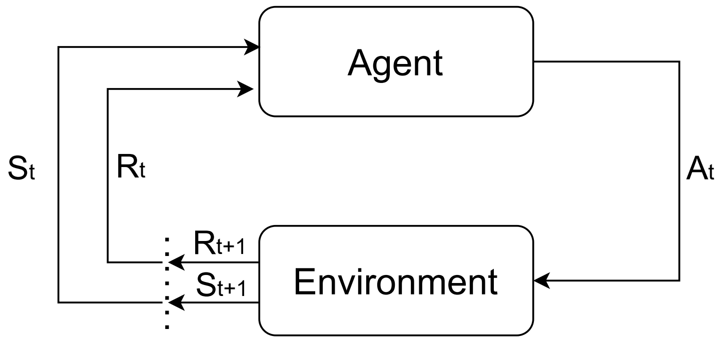
The term "Reinforcement Learning" refers simultaneously to a problem, to a class of methods that work as solutions to the problem, and to the field that studies the problem and its solutions [Sutton e Barto 2018]. With that in mind, this chapter tries to summarize the basic concepts of these different but overlapping descriptions. In Section 3.1, the mathematical formalization of the RL problem and its key elements are defined. Section 3.2 presents an overview of the main RL solutions categories, with a special focus on Policy Gradient and Actor-Critic methods. Finally, the Proximal Policy Optimization algorithm, which is the core of the robotic exploration architecture proposed in this work, is described in Section 3.3.

3.1 Reinforcement Learning Problem

The fundamental elements that compose a Reinforcement Learning problem setup are the *agent* and the *environment*. The agent is an entity that learns and makes decisions, while the environment is everything that surrounds the agent and that it interacts with. Figure 3.1 illustrates the interactions between agent and environment. At time step t, the agent uses the observation of its current *state* in the environment (S_t) to decide what *action* (A_t) to perform. Taking action A_t leads the agent to a new state S_{t+1} and results in a *reward* R_{t+1}, which depends on the impact action A_t had on the environment. By consecutively executing this closed loop, the agent can improve its strategy by trying to maximize the received reward.

The RL problem can be mathematically formalized as a *Markov Decision Process* (MDP). A tuple $\langle S, A, P, R \rangle$ defines a finite MDP, where:

Figure 3.1 – Agent–environment interaction. At every new time step $(t + 1)$, the agent receives a new state S_{t+1} and the reward R_{t+1} associated with performing action A_t from previous state S_t.



Source: [Sutton e Barto 2018].

- S represents a finite set of states ($s_t \in S$)
- A represents a finite set of actions ($a_t \in A$)
- P is the transition probability associated with the states ($P(s_{t+1} \| s_t, a)$)
- R is the reward function, where $R(s_t, a_t, s_{t+1})$ is the immediate reward received for going from state s_t to state s_{t+1} after action a_t.

In the RL framework, P and R are fixed and unknown. The system attends the Markov property: "The future is independent of the past given the present" [Feldman e Valdez-Flores 2010]. In other words, only the current state impacts the future state decision. In mathematical terms, a state St attends the Markov property if it captures all the information from the past, as expressed in Equation 3.1. An MDP also assumes that the environment is fully observable, with no uncertainty in the observations.

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_0, s_1, s_2, ..., s_t) \tag{3.1}$$

Some key elements compose this mathematical structure. As defined in Equation 3.2, a **trajectory** or **episode** (τ) is a sequence of states, actions and rewards that results from the agent-environment interaction illustrated at Figure 3.1. A **Policy** (π) is a function that maps the state space into the action space. It can be either *deterministic* ($\pi(s_t)$), returning a specific action given a specific state, or *stochastic* ($\pi(a_t|s_t)$), outputting an action probability distribution.

$$\tau_t = (s_0, a_0, r_1, s_1, a_1, r_2, ..., s_t, a_t, r_t + 1) \tag{3.2}$$

The goal of the agent is to maximize the expectation of the **cumulative episode return** ($R(\tau)$), which is the reward accumulated over a trajectory. It can be defined by

Equation 3.3, where T is the trajectory length, $\gamma \in [0, 1)$ is a discount factor, and r_t is the reward at time step t. Considering that both policy ($\pi(a_t|s_t)$) and state transitions (P) are stochastic, the probability of a trajectory with T steps is expressed in Equation 3.4. So, the **expected return** ($J(\pi)$) can be defined as the integral of the product between the probability of the trajectory and the cumulative return over that trajectory, as expressed in Equation 3.5. Finally, the optimal policy (π^*) is formulated as the one that maximizes the expected reward, according to Equation 3.6.

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t \tag{3.3}$$

$$P_\pi(\tau) = \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{3.4}$$

$$J(\pi) = \int_\tau P_\pi(\tau)R(\tau) = \mathbb{E}_\pi[R(\tau)] \tag{3.5}$$

$$\pi^* = argmax_\pi(J(\pi)) \tag{3.6}$$

The expected return of a trajectory that starts in a state s and follows a policy π is defined as the **state value**. The **state value function** ($V^\pi(S)$) measures "how good" a state is, and is defined according to Equation 3.7. Analogously, the **action-state value function** ($Q^\pi(a, s)$) estimates the value of a action-state pair, that is, the expected return after taking action a at state s while following policy π (Equation 3.8). The difference between the state-action value function and state function is called **advantage**, as expressed by Equation 3.9. As the name implies, it estimates the advantage of taking a specific action, depending on whether the state-action value is higher or lower than the current state value. The state value, action-state value, and advantage functions are useful in the process of searching for an optimal policy.

$$V^\pi(S) = \mathbb{E}_{\tau\ \pi}[R(\tau)|s_0 = s] \tag{3.7}$$

$$Q^\pi(a, s) = \mathbb{E}_{\tau\ \pi}[R(\tau)|s_0 = s, a_0 = a] \tag{3.8}$$

$$A^\pi(s, a) = Q^\pi(a, s) - V^\pi(S) \tag{3.9}$$

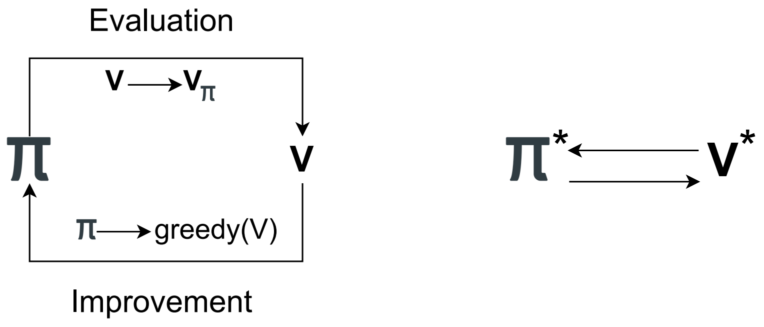
3.2 Methods and Algorithms

A large group of learning algorithms has been proposed to solve RL problems. These algorithms can be divided into two main classes: Model-based and Model-Free. **Model-based** methods use learning and a model of the environment's transitions to approximate a global value or policy function. The model can be learned, meaning that the agent learns both the model and a value or policy, or the model can be known, where the agent knows the model and uses planning to learn a global value or policy [Moerland, Broekens e Jonker 2020]. On the other hand, **Model-Free** methods do not rely on an environment model, and the agent learns the policy or value directly through trial-and-error with the physical system [Polydoros e Nalpantidis 2017]. Model-Free RL approaches are able to solve problems that can not be solved mathematically, but they return non-optimal solutions.

Both model-free and model-based methods aim at learning an optimal policy by maximizing the expected return (3.5). This learning process is based on the Generalized Policy Iteration (GPI), which is divided in to phases: policy evaluation and policy improvement. By repeatedly executing this two steps, an optimal policy can be learned. Depending on how the policy iteration process is formulated, the algorithms can be categorized as **value-based**, **policy-based**, or **actor-critic**.

In **Value-Based** algorithms, the agent's trial-and-error process results in a value function from which the policy is derived. This value function estimates how advantageous it is for the agent to be in a specific state or to perform a given action considering a specific state [Sutton e Barto 2018]. The general logic of value-based algorithms is illustrated in Figure 3.2. It demonstrates that the policy π evaluation depends on the value function V, while it follows a greedy behavior considering the value function. Therefore, the policy decides the action based on the optimum value of V. Among the most famous algorithms are the classic Q-Learning [Watkins 1989] and the Deep Q-Networks (DQN), which use deep learning to estimate the value function in a Q-Learning framework [Mnih et al. 2013]. Other important value-based algorithms the Monte Carlo (MC) [Bouzy e Chaslot 2006], State–action–reward–state–action (SARSA) [Rummery e Niranjan 1994] and its variations like the SARSA(λ). The advantage of value-based methods is the low variance in the expected return evaluation results in fast and stable policy evaluation steps. However, they only learn deterministic policies, and are only suitable to discrete action spaces.

Figure 3.2 – General policy iteration using value based approach.



Source: [Sutton e Barto 2018].

Unlike value-based approaches, **policy-based** methods learn the policy directly from the agent's interaction with the environment. This means that the search happens directly in the policy space, so the problem becomes a case of stochastic optimization. To optimize the policy parameters, the expected return J (Equation 3.5) is evaluated. According to [Sutton e Barto 2018], the gradient of the return can be calculated as:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_{i,t}|s_{i,t}) \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \tag{3.10}$$

where N represents the number of trajectories. Many policy search algorithms use descent ascent to optimize their parameters (θ) as described by Equation 3.11, where α represents the learning rate. Some advantages of policy-based over value-based methods is that they are able to learn stochastic policies, and are suitable to continuous action space. However, the policy convergence can be slow and unstable due to high variance in the expected return evaluation.

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{3.11}$$

Finally, **Actor-Critic** algorithms adopt a hybrid approach, aiming to combine the advantages of both value and policy-based methods. The *actor* is a parameterized policy that maps the state of the environment into the selected action. This policy is improved the direction suggested by the *critic*, which is a value function. At the end of the learning process, the actor correspond to the optimal policy, and the critic to the optimal value function. [Zeng 2019]. Some widespread actor-critic algorithms include the Soft Actor-Critic (SAC) [Haarnoja et al. 2018], Asynchronous Advantage Actor-Critic (A3C) [Mnih et al. 2016], and the Advantage Actor-Critic (A2C), and the Trust Region Policy Optimization (TRPO) [Schulman et al. 2015]. The Proximal Policy Optimization algorithm, used in this dissertation also fits into the actor-critic category, and is explained in further details in the following section.

3.3 Proximal Policy Optimization algorithm

The Proximal Policy Optimization is a Policy Gradient method proposed in 2017 by Schulman *et al.* [Schulman et al. 2017]. PPO was chosen as the algorithm used in this work because it has achieved good performance in state-of-the-art works in different fields, while being more robust to perturbations than other actor-critic algorithms. Usually, small changes in the underlying parameters of the neural network (NN) can cause large jumps in policy space of actor-critic methods. PPO addresses this problem by limiting the updates to the policy network. It bases the update on the ratio of new policy to old policy, constraining this ratio to be in a specific range. To avoid the loss function from growing too large, it is clipped and the lower bound is considered. PPO also provides a good balance between ease of implementation, parameters tuning, and sample complexity.

During training, a fixed length trajectory of memories is collected. This stage is called *rollout*, or *game*. When the rollout is over, the policy is updated adopting a minibatch stochastic gradient ascent. The update rule for the actor network is defined by Equation 3.12. It determines that the Loss of the Conservative Policy Iteration ($L_{CPI}(\theta)$) equals the Expectation value of the product of the ratio of the policy under the current parameter (π_θ) and the policy under the old parameters with the advantage ($\pi_{\theta old}$). This rate is represented by $r_t(\theta)$, a_t is the action taken, s_t is the state, and A_t is the advantage value at time t.

$$L_{CPI}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}A_t\right] = \mathbb{E}[r_t(\theta)A_t] \tag{3.12}$$

$$L_{CLIP}(\theta) = \mathbb{E}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{3.13}$$

To constrain the space of the loss function, the parameter ϵ is added to clip the loss function, resulting in Equation 3.13. The Critic loss function ($L_C(\theta)$) is represented by Equation 3.14, where MSE is the mean square error, V_{old} is the critic value from memory and V_t is the current critic value from the network. The total loss function ($L_t(\theta)$) is described by Equation 3.15, where c_1 and c_2 are positive constants, and S_{π_θ} is the entropy of the exploration policy at state. Algorithm 1 presents an overview of the PPO-Clip learning process implemented in this work.

$$L_C(\theta) = MSE(A_t + V_{old} - V_t) \tag{3.14}$$

$$L_t(\theta) = L_{CPI}(\theta) + c_1 L_C(\theta) - c_2 S_{\pi_\theta} \tag{3.15}$$

Algorithm 1 PPO-Clip. Adapted from [Schulman et al. 2017]

```
Input: Initial policy parameters $\theta_0$ and value function parameters $\phi_0$
1: for $k = 0, 1, 2, ...$ do
2:     Collect set of trajectories $T_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$
3:     Computes advantage estimates $A_t^\pi$ from current value function $V_{\phi k}$
4:     Using stochastic gradient ascent with Adam optimizer [Kingma e Ba 2014], update
       the policy by maximizing the PPO-Clip objective:
       $$\theta_{k+1} =_\theta \frac{1}{|T_k|N} \sum_{\tau \in T_k} \sum_{t=0}^{N} min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), clip(A^{\pi_{\theta k}}(s_t, a_t), i + \epsilon, i - \epsilon) \right)$$
5:     Using backpropagation, fit value function by regression on mean-squared error:
       $$\phi_{k+1} =_\phi \frac{1}{|T_k|N} \sum_{\tau \in T_k} \sum_{t=0}^{N} (V_\phi(s_t) - R_t)^2$$
6: end for=0
```

4 REINFORCEMENT LEARNING AND ROBOTICS

This chapter reviews how robotics and Reinforcement Learning are combined in modern applications, especially for mobile robot exploration. First, Section 4.1 discusses the motivation and challenges for integrating the two fields. Then, Section 4.2 presents the common approaches to employing Reinforcement Learning algorithms to address the mobile robotic exploration problem. Finally, recent academic works that propose solutions for single and multiple robot exploration strategies using RL algorithms are reviewed in Sections 4.3 and 4.4.

4.1 Reinforcement Learning in Robotics Applications

As remarkably described in [Kober, Bagnell e Peters 2013], the disciplines of reinforcement learning and robotics compose a promising relationship, given that RL makes hard-to-engineer behaviors feasible for robotics applications, while robotics challenges inspire and validate RL solutions. The intrinsic function of robots is to replicate animal behavior to assist or replace humans in different tasks. Therefore, the idea of integrating learning capability to robotic devices arises almost naturally. Simultaneously, reinforcement learning tries to emulate how humans and other animals learn through trial-and-error interactions with the environment, being even used to study the brain functioning in research fields like Neuroinformatics. Hence, the application of reinforcement learning techniques to robotics is increasingly being investigated by the academics, lead by the goal of letting the robot autonomously learn how to plan and control its actions, having generalization capability and becoming suitable for complex and dynamic tasks.

However, robotics differ in several essential aspects compared with domains where RL was already successfully employed, such as video games. Robotics applications take place in the real world, which means that the agent must cope with partially observable systems, measurement noise and delays, impossibility to speed up the training phase in the real environment, expensive hardware that requires safe exploration, among others. The most commonly adopted alternative to limit the interaction with the real world is to perform the training phase through simulation, which also presents issues. Transferring the behaviors learned through simulation to the real robot is not usually a straightforward task, given that errors in the simulated environment model can easily accumulate and cause the behavior to diverge from the expected [Dulac-Arnold, Mankowitz e Hester

2019]. Other RL problems that are highlighted in robotics are the reward shaping difficulty and the curse of dimensionality, that refers to the exponential raise of computational effort given the increase of spatial dimensions [Venkat 2018].

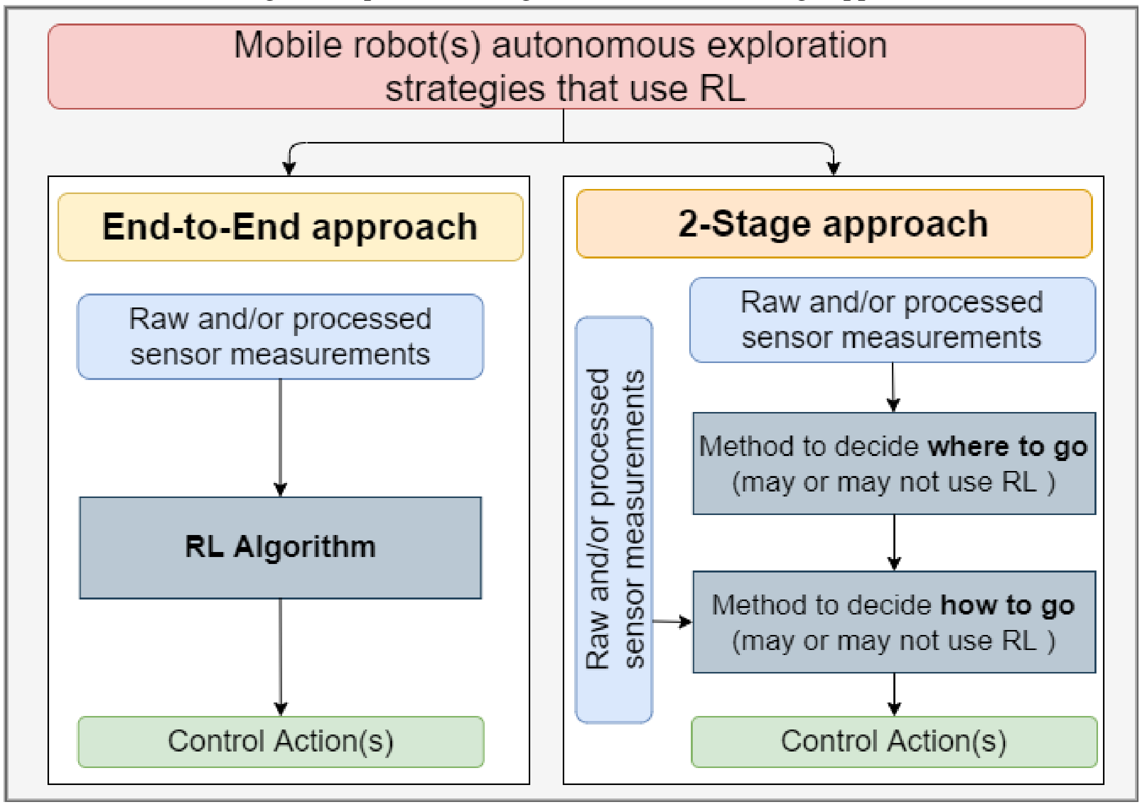
To solve the aforementioned problems and make the employment of RL in real robotic applications viable, several tools and strategies have been exploited. It is possible to reduce the dimensionality curse impact and enhance the RL algorithm convergence and generalization with smart approaches to discretize the state or action spaces [Akrour et al. 2018, Arai et al. 2018]. Function approximation can be used to help predict the reward functions [Lim, Ha e Choi 2020] and to make value based RL algorithms suitable for robotics [Yang, Juntao e Lingling 2020], using methods like Gaussian Regression and Neural Networks. An alternative to speed up training and increasing the convergence probability is transferring auxiliary information or knowledge to the agent before or during the learning phase. This can be accomplished by, for example, employing Transfer Learning techniques [Chalmers et al. 2018], using demonstration [Nair et al. 2018, Shimizu et al. 2015], learning forward environment models [Hirata, Iizuka e Yamamoto 2020, Le, Le e Nguyen 2017], incorporating human feedback during training [Pérez-Dattari et al. 2019] and decomposing a task into simpler components [Jain, Iscen e Caluwaerts 2019, Yang et al. 2018].

From a general perspective, it is possible to state that applying reinforcement learning to robotics remains a challenge. In the next sections, the aforementioned problems and solutions are evaluated in the state-of-the-art work in the specific context of mobile robotics exploration of unknown environments.

4.2 Exploring Unknown Environments using Reinforcement Learning

In the past decades, different methods to address the mobile-robots autonomous exploration problem have been proposed and successfully tested, including the famous frontier-based and the cost-utility approaches. So, what justifies using Reinforcement Learning to tackle autonomous exploration, considering its computational costs and already discussed limitations in robotic applications? The fact is that traditional exploration techniques usually make strong assumptions about the environments and the tasks, which may restrict their adaptability to dynamic complex environments and thus limit their application in real-world practices. The need for methods that provide robust and flexible solutions to robotic problems and the great advances in Machine Learning techniques have made the combination of the two research fields become a hot topic in recent years. In that context, the employment of Reinforcement Learning for robotic exploration tasks is being increasingly investigated, driven by the idea of letting the agents automatically learn skills from environment interaction, instead of receiving explicit instructions. Furthermore, RL does not require dataset labeling, which represents a large cost for Supervised Learning solutions.

Figure 4.1 – Classification of mobile-robot exploration strategies that employ Reinforcement Learning techniques, including End-to-End and 2-Stage approaches.


Mobile robot(s) autonomous exploration strategies that use RL
End-to-End approach
Raw and/or processed sensor measurements
RL Algorithm
Control Action(s)
2-Stage approach
Raw and/or processed sensor measurements
Raw and/or processed sensor measurements
Method to decide where to go (may or may not use RL)
Method to decide how to go (may or may not use RL)
Control Action(s)

Source: The Author.

The great majority of the recently proposed exploration techniques fit into the Hybrid paradigm, being able to use global environment information to plan the next actions, but also reacting to unseen events. To clearly synthesize how the RL algorithms usually compose the proposed exploration strategies, a new classification were developed, dividing the methods into two categories, as illustrated in Figure 4.1. In both classes, the inputs are composed by raw sensor measurements, such as camera images, sets of distances or velocities, by processed sensor measurements, like partial maps, robot trajectory and robot pose, or by a combination of both. In some less common cases, human feedback is also used as an input.

Strategies with an **End-to-End** approach accomplish robot exploration tasks as a black box. The inputs are fed to the RL or deep RL algorithm, that directly returns the robot control actions, like linear and angular velocities or the movement the robot must perform (e.g move forward, backwards, right or left). As the name states, the **2-Stage** strategies divide the robot decisions in two parts, integrating RL with non-learning-based approaches. First, the inputs are used by an algorithm that decides the next location the robot must move to. Then, the selected location, together with other sensory inputs, are used to perform the path planning and guide the agent from the current to the target position. RL algorithms can be employed in the first, second or both stages. In the next section, the proposed classification is taken as reference for evaluating the research works.

4.3 Single robot exploration

This section reviews the state-of-the-art exploration strategies designed for a single robotic agent using Reinforcement Learning techniques. The applications targeted by the academic works are divided into **goal-guided**, regarding tasks whose goal is to find or reach a specified target, and **non-guided**, regarding tasks whose main goal is to cover a whole area, usually with mapping purposes. The most common identified non-guided application is mapping unknown indoor environments [Tai e Liu 2016, Liu, Liu e Wang 2017, Tai e Liu 2016, Guerra et al. 2020]. For instance, in [Zhu et al. 2018] the problem of mapping an unknown office is tackled, and in [Craye, Filliat e Goudou 2016] the exploration is used for learning a saliency map of the environment.

Currently, goal-guided applications are being studied in a larger quantity and with a greater task diversity. Some examples include finding victims in post-disaster scenes [Niroui et al. 2019, Pham et al. 2018], finding the original location of a chemical leaking source in underwater environments [Hu, Song e Chen 2019], traversal of land vehicles in undiscovered tracks [Josef e Degani 2020], and goal-driven map-less navigation [Shi et al. 2019, Tai, Paolo e Liu 2017, Zhelo et al. 2018, Zhu et al. 2017]. The great majority of both kinds of applications include obstacle avoidance.

4.3.1 RL algorithms in the exploration strategies

Tables 4.1 and 4.2 summarize how the identified RL algorithms compose, respectively, the proposed 2-Stage and End-to-End single-agent exploration strategies. Two approaches were identified in the 2-stage strategies. The first and more common one is using the RL algorithm to decide the next location the robot should move to and then employing complementary methods to guide the robot towards the target point. In [Niroui et al. 2019], DRL is combined with the classic frontier-based exploration. An A3C network receives the known map, the robot location, and the frontiers locations, returning the coordinates of the next goal frontier. Similarly, an A3C network receives the current map, the agent's location and orientation in [Zhu et al. 2018], and returns the next visiting direction, given that the space around the agent is equally divided into six sectors.

Table 4.1 – How RL algorithms compose 2-Stage Single-Agent exploration strategies.

		Application Goal	**Research Work**	**RL Algorithm**	**DRL**	**State Space**	**Action Space**	**Reward**	**Complementary Method**
2-Stage Single Agent Exploration Strategies	RL algorithm to decide **where to go**	Goal-Guided	[Niroui et al. 2019]	A3C	Yes*	Discrete	Discrete	Sparse	A* to find path and ROS Move Base package to generate control movements
		Non-Guided	[Zhu et al. 2018]	A3C	Yes*	Discrete	Discrete	Dense	Next Best View with Bayesian Optimization to choose a point in the region selected by the DRL, and A* for path planning
			[Li, Zhang e Zhao 2019]	DQN	Yes*	Discrete	Discrete	Dense	A* for path planning and Timed-Elastic-Band (TEB) for setting the robot's velocities
			[Craye, Filliat e Goudou 2016]	Q-Learning	No	Discrete	Discrete	Sparse	Method to control robot movements not specified
	RL algorithm to decide **how to go**	Non-Guided	[Liu, Liu e Wang 2017]	OS-Q-ELM Network	Yes	Continuous	Discrete	Dense	Choose the furthest point measured by RPLIDAR

* Neural Netowrks with Convolutional Layers.

In [Li, Zhang e Zhao 2019], a DQN returns the goal points in the grid map, using as input the map, the current, and the historical robot positions. The three mentioned works employ the well-studied A* algorithm for path planning. In [Craye, Filliat e Goudou 2016], Q-learning is used to decide whether the agent must move to one of its four adjacent nodes or if it must learn from the current node by capturing useful images to create a saliency map of the environment. The reviewed works prove that this approach is a smart way to discretize and reduce the action space while using RL to solve the main exploration problem, which is to decide the next region the robot should visit. However, a possible drawback is that the methods focus on the trajectory generation from one point to another, not prioritizing the full coverage of explored areas in the path planning process.

Another approach in 2-Stage strategies is employing a non-learning method to decide the next goal location and then using Reinforcement Learning to guide the robot towards the selected point. For single agents, this approach is less common. In [Liu, Liu e Wang 2017], the goal point is randomly selected between the set of the furthest points measured by an RPLIDAR sensor with scanning range $360°$. An Online Sequential Extreme Learning Machine (OS-ELM) is used to estimate the Q-values and lead the robot towards the target with object avoidance. The network receives the sensors' measurements, the distance between robot and goal, the angle between robot orientation and goal, and returns the robot action, which can be moving 0.3m, turning left, or turning right. The downside of this approach is that it does not use the RL algorithm for the critical exploration decision-making process, but to plan the trajectory between two known points. At the same time, there are much more efficient and well-established path planning methods in the literature.

Table 4.2 reveals the most investigated approach for single robot exploration using RL: tackling goal-guided problems in an end-to-end fashion. Within this context, many works use neural networks with convolutional layers aiming to reproduce human behavior and directly translate pixels into actions. For example, the Rainbow network architecture was used to generate local paths toward goal positions in unknown rough terrain environments while maintaining the UGV's safety [Josef e Degani 2020]. Some works employ DQN algorithms' variations to improve their performance and avoid problems caused by estimation errors. Double Deep Q-Networks, proposed by H. V. Hasselt [Hasselt, Guez e Silver 2016] and based on decoupling action selection from evaluation, were used to define the next robot action between 5 possible movement controls [Issa et al. 2020,Çetin et al. 2019].

Table 4.2 – How RL algorithms compose End-to-End Single-Agent exploration strategies.

	Application Goal	**Research Work**	**RL Algorithm**	**DRL**	**State Space**	**Action Space**	**Reward**
End-to-End Single Agent Exploration Strategies	Non-Guided	[Cardona et al. 2019]	Q-Learning	No	Discrete	Discrete	Dense
		[Tai e Liu 2016]	DQN	Yes*	Discrete	Discrete	Dense
		[Tai e Liu 2016]	DQN	Yes*	Discrete	Discrete	Sparse
	Goal-Guided	[Issa et al. 2020]	Double DQN	Yes*	Discrete	Discrete	Dense
		[Zhou et al. 2018]	DQN	Yes	Discrete	Discrete	Dense
		[Yijing et al. 2017]	DQN	Yes*	Continuous	Discrete	Dense
		[Pham et al. 2018]	Q-Learning	No	Discrete	Discrete	Dense
		[Hu, Song e Chen 2019]	DPG	Yes	Continuous	Continuous	Sparse
		[Josef e Degani 2020]	DQN with Rainbow network	Yes*	Discrete	Discrete	Dense
		[Ruan et al. 2019]	Dueling Double DQN (D3QN)	Yes*	Discrete	Discrete	Dense
		[Shi et al. 2019]	A3C	Yes*	Discrete	Discrete	Sparse
		[Çetin et al. 2019]	Double DQN	Yes*	Discrete	Discrete	Dense
		[Fan et al. 2020]	SAC	Yes*	Continuous	Continuous	Dense
		[Tai, Paolo e Liu 2017]	Asynchronous DDPG	Yes	Continuous	Continuous	Dense
		[Zhelo et al. 2018]	A3C	Yes*	Continuous	Discrete	Dense
		[Jiang, Huang e Ding 2019]	DQN	Yes*	Continuous	Discrete	Sparse
		[Zhu et al. 2017]	Deep Siamese Actor-Critic Network	Yes*	Discrete	Discrete	Dense
		[Rana et al. 2020]	TD3	Yes	Continuous	Continuous	Sparse
	Non-Guided and Goal-Guided	[Guerra et al. 2020]	Q-Learning	No	Discrete	Discrete	Dense

* Neural Netowrks with Convolutional Layers.

A dueling architecture based deep double Q network (D3QN), which combines Double DQN and Dueling Network Architectures, was used with a similar action space [Ruan et al. 2019]. To tackle the common RL problem of lack of generalization capability to new goals, the authors of [Zhu et al. 2017] propose a new deep neural network called Siamese Actor-Critic network, which receives an RGB image of the current environment observation and an RGB image of the target, and returns the movement the agent should perform next. Other works also propose end-to-end exploration solutions combining convolutional layers for feature extraction with the A3C algorithm [Shi et al. 2019, Zhelo et al. 2018], and the Soft Actor-Critic (SAC) [Fan et al. 2020]. As previously mentioned, these methods take images as inputs. A limitation of all mentioned works is that the models are trained with synthetic, rendered scenarios, which generally do not generalize appropriately to real-world images.

Within the goal-guided end-to-end solutions, there are the ones that do not employ convolutional neural networks. To guide the robot to reach a specific point at an unknown environment with obstacle avoidance, a Back Propagation (BP) network is used in [Zhou et al. 2018] to estimate the Q-values. A continuous action space for the robot control is employed in [Hu, Song e Chen 2019] and in [Tai, Paolo e Liu 2017] by using, respectively, an actor-critic model with Deterministic Policy Gradient (DPG) and Asynchronous Deep DPG. End-to-end solutions for non-guided applications are not as numerous in the literature. In both [Tai e Liu 2016] and [Tai e Liu 2016], CNNs are used to extract features from raw RGB images and depth information from an RGB-D sensor, respectively, and DQN define the next agent's movement to explore an unknown environment entirely. With the same goal, classic Q-Learning is used in [Cardona et al. 2019].

Finally, few end-to-end solutions adapt for both goal-guided and non-guided applications. In [Guerra et al. 2020], a Q-Learning algorithm uses the UAV position, a binary parameter that indicates the presence or absence of a signal source (target) in the environment, and each grid cell's state to guide the robot to map the environment and detect targets. It is important to notice that the integrated approach of end-to-end solutions also has its limitations. The main downfall is the need to control the vehicle reactively through on-board real-time computing. Moreover, end-to-end methods still experience limited generalization capabilities and they are tightly dependent on the system (e.g., type of vehicle, sensors) [Guastella e Muscato 2021].

4.3.2 Approaching common problems

Exploration–Exploitation dilemma: The great majority of academic works about single mobile robot exploration using RL employ the classic ϵ-greedy or some simple variations of the algorithm to handle the Exploration-Exploitation dilemma. Considering ϵ a positive scalar between 0 and 1, the agent selects the action with maximum value with probability $1 - \epsilon$, and selects a random action with probability ϵ. If the ϵ value increases during training, the exploratory behavior is more stimulated in the early learning stages, and as time passes, the agent selects the optimal action more frequently. Furthermore, a fewer number of works employ different techniques, but with a similar approach. Stochastic Switching is used in [Rana et al. 2020], and in [Yijing et al. 2017] the Boltzmann distribution is employed to define the probability of choosing one action given the current state. In [Fan et al. 2020], the Temperature Decay training paradigm is proposed, working similarly to ϵ-greedy, but adapted to obtain an uncertainty-averse behavior.

Curse of dimensionality: To avoid the curse of dimensionality, the most common approach is to employ deep neural networks to perform function approximation. Another function approximation technique used in [Pham et al. 2018] is called Fixed Sparse Representation (FSR), and maps the original Q table to a parameter vector. The discretization of the states and action spaces is also a commonly adopted alternative, as can be observed in both Table 4.1 and Table 4.2. In the analysed works, the algorithms that directly return motion controls with a discrete set of actions used between 3 and 9 possible alternatives. It is important to observe that although adopting a small and discrete set of actions accelerates training convergence, in motion control applications it may limit the robot performance, resulting in tortuous paths.

Reward shaping: In general, the examined research works use heuristic strategies to define the reward functions for the robotic exploration applications. The application goal directly influences how the rewards are modeled: in a search and rescue mission, the reward usually encourages the agent to find the most quantity of information in the early stages of exploration [Pham et al. 2018], for example. For goal-driven navigation, the reward encourages the distance narrowing between the agent and the target [Zhou et al. 2018, Yijing et al. 2017]. Collisions with obstacles are associated with punishments in all strategies. The rewards can be either *dense*, which means they are assigned to the agent in many different states, or *sparse*, usually returning zero for most states and only rewarding the agent in a few states or events.

Table 4.1 and Table 4.2 indicate the kind of reward used for each reviewed work. In all cases, the agent receives the most significant reward if it achieves the exploration goal and gets a considerable punishment if the mission fails (e.g., takes too much time, a collision happens). The difference is that dense strategies adopt intermediate rewards. Sparse rewards are easier to be defined and are adopted by many strategies [Niroui et al. 2019, Craye, Filliat e Goudou 2016, Tai e Liu 2016, Hu, Song e Chen 2019, Shi et al. 2019, Jiang, Huang e Ding 2019, Rana et al. 2020], but they can increase the learning convergence time. Therefore, despite the difficulty to properly determine dense rewards, they are the most adopted option among the single robot exploration works. Common approaches include: punishing the agent at every time-step to decrease exploration time [Pham et al. 2018, Çetin et al. 2019, Zhu et al. 2017]; small positive rewards if getting away from obstacles or closer to target; minor punishments otherwise [Cardona et al. 2019, Zhou et al. 2018, Yijing et al. 2017, Josef e Degani 2020, Tai, Paolo e Liu 2017].

Learning Convergence: Different strategies to accelerate learning convergence were identified in the robotic exploration works. The solutions used to tackle all previously mentioned aspects - exploration-exploitation dilemma, the curse of dimensionality, and reward shaping - directly impact the learning convergence. The discretization of the state and action spaces, widely adopted in the reviewed works, contributes to faster convergence. Curiosity-driven intrinsic rewards are used in [Shi et al. 2019] and [Zhelo et al. 2018] to stimulate the agent to explore new areas after many failures in familiar positions, improving data efficiency and avoiding possible deadlocks.

A method that is especially present is the Experience Replay (EP), which employs past experiences in the training process to reduce the correlation between successive samples, make the learning process smoother, and improve data efficiency [Tai e Liu 2016, Li, Zhang e Zhao 2019, Çetin et al. 2019, Fan et al. 2020, Tai e Liu 2016, Jiang, Huang e Ding 2019]. An attention mechanism that analyses the importance of the algorithm's inputs is employed in [Josef e Degani 2020], aiming to increase data efficiency. Transfer learning techniques were also employed in the single-robot exploration context. Dynamic programming was used to find approximate solutions to initialize the actor-critic networks [Hu, Song e Chen 2019]. Similarly, the weights of the CNN trained with RL were initialized using a supervised learning model trained with real-world data [Tai e Liu 2016]. Another identified strategy is to increase the training difficulty gradually and share knowledge between different agents [Zhu et al. 2017].

4.4 Multi-robot exploration

Multi-robot systems (MRS) can be defined as a group of two or more robots that are able to cooperate or compete with each other to achieve a specified goal [Gautam e Mohan 2012]. Research on MRS has attracted considerable attention in the past years, driven by its advantageous characteristics such as high levels of fault tolerance, increased efficiency in task accomplishment, situational awareness from multiple locations, greater flexibility in operations, and distributed payloads. This set of features make MRS suitable for several complex and important applications [Yan, Jouandeau e Cherif 2013], including unknown environment exploration. However, the multi-robot teams' success relies on the agents' autonomy skills and on the design of a proper cooperation strategy, which is a non-trivial task and represents the core challenge for multi-robot cooperation viability. In this context, Reinforcement Learning algorithms are being highlighted as a promising alternative to compose MRS coordination strategies. This section contains an overview of the most recent academic works that propose MRS cooperation strategies for unknown environment exploration using RL techniques.

The analysed research works were recently published and encompass UAVs, UGVs, and AUVs applications. Some commonly identified non-guided MRS goals are collectively exploring or covering entire unknown areas, and mapping a strange environment as soon as possible, which are suitable skills for applications such as exploring cluttered urban search and rescue (USAR) scenes and uncertain environment patrolling. On the other hand, usual goal-guided applications include goal-driven map-less navigation through unknown complex environments, and searching static or dynamic multi-targets in unknown environments. The target searching goal were investigated in contexts such as underwater environments, victims identification in USAR scenes, and pursuit-evasion game, where a group of predator agents are trained to capture the prey agents cooperatively. Collision avoidance, either with objects or between agents, is a highlighted concern of all non-guided and goal-guided applications. Again, to the best of our knowledge, there is no work approaching non-guided map-less exploration for MRS using RL.

4.4.1 RL algorithms in the exploration strategies

Tables 4.3 and 4.4 summarize how the identified RL algorithms compose, respectively, the so far proposed 2-Stage and End-to-End MRS exploration strategies. Three approaches were identified in 2-Stage strategies. First, there are the works that use RL to decide where each agent should move next, and employ a complementary method to perform the path planning between current position and goal destination. Both [Luo et al. 2019] and [Zhou et al. 2019] propose a coordination strategy for non-guided applications employing topological maps, in which the RL algorithm determines the next nodes the agents should move to. As the goal of the RL is to decide between a finite set of possible locations, the action spaces are inherently discrete. Again, a benefit of this approach is the employment of RL in the key decision-making process of exploration, while using a discrete action space. However, the path planning do not focus on efficient area coverage.

Table 4.3 – How RL algorithms compose 2-Stage MRS exploration strategies.

		Application Goal	**Research Work**	**RL Algorithm**	**DRL**	**State Space**	**Action Space**	**Reward**	**Complementary Method**
2-Stage Cooperation Strategies	RL algorithm to decide **where to go**	Non-Guided	[Luo et al. 2019]	DQN	Yes*	Discrete	Discrete	Not Specified	Any path planning algorithm that suits graph search
			[Zhou et al. 2019]	Monte Carlo (DGSMCP)	No	Discrete	Discrete	Sparse	
	RL algorithm to decide **how to go**	Non-Guided	[Hu et al. 2020]	DDPG	Yes	Continuous	Continuous	Dense	Voronoi-based target selection
		Goal-Guided	[Walker et al. 2020]	TRPO	Yes	Continuous	Discrete	Dense	POMD solver
			[Cai, Yang e Xu 2013]	MAXQ	No	Discrete	Discrete	Not Specified	Hungarian Method
	RL algorithm for **both decisions**	Goal-Guided	[Jin et al. 2019]	Where: DQN	Yes*	Discrete	Discrete	Dense	———
				How: DDPG	Yes	Continuous	Continuous		
			[Cao, Sun e Yan 2019]	Where: A3C	Yes*	Discrete	Discrete	Dense	———
				How: DQN	Yes*	Continuous	Discrete		

* Neural Netowrks with Convolutional Layers.

When it comes to RL usage for guiding the agents towards selected locations, the proposed strategies are more diverse. In [Hu et al. 2020], each robot is assigned a different target location based on dynamic Voronoi partitions. As DDPG can handle continuous action spaces, it is used to decide the linear and angular velocities of the robots, increasing behavior possibilities and enabling smoother movements. With focus on target searching, in [Walker et al. 2020] a POMDP solver defines if the agent should explore its current node or for which node it should move next, and in [Cai, Yang e Xu 2013] the Hungarian method is used to obtain the best arrangement of cooperative sub-tasks and distribute it between the robots. In both works, the RL algorithm receives the next goal location and a set of sensor measurements, and return the agents movement. Unlike [Hu et al. 2020], the action spaces are discrete and composed of 3 and 4 movements possibilities, respectively.

Finally, there are the works that use two different RL algorithms to assign where to go and how to reach the defined locations. As DQN is more suited to discrete action spaces, in [Jin et al. 2019] it is used to indicate the coordinates of the next location, and DDPG is used to guide the agents selecting rotation angles in a continuous action space. In [Cao, Sun e Yan 2019], the traditional frontier exploration method is combined with DRL, whre the A3C algorithm decides the next goal frontier and the DQN defines the movements to guide the agents. This approach results in smart path planning and cooperation strategies. However, the use of two deep neural networks can represent high computational costs that must be considered, especially in applications with energy, area and computational constrains.

As previously defined, end-to-end systems are a integrated approach, taking raw and/or processed sensors measurements as inputs and returning the robots' control actions. The state spaces of the MRS End-to-End exploration strategies are in general a combination of some or all of the following parameters: the robot's pose, the sensor measurements union, the location of the other agents and the known map. From Table 4.4, it is possible to notice that all methods with continuous state spaces use neural networks with convolutional layers. These layers are able to extract features from the inputs, and are connected to a fully-connected neural network that returns each agent actions. As for the action space, the great majority of research works employ a discrete space, defining, in average, between 3 and 9 possible moving directions or spinning angles. At the same time that this discretization makes the training time decrease and facilitates the algorithms convergence, it was repeatedly pointed as a limitation for the MRS performance, as it causes abrupt behaviors and tend to make the solutions less effective in realistic complex scenarios. Similar to single-robot end-to-end strategies, on-board real time computing to control the vehicles' movements must be considered as a project requirement, which can limit its application.

Table 4.4 – How RL algorithms compose End-to-End MRS exploration strategies.

	Application Goal	**Research Work**	**RL Algorithm**	**DRL**	**State Space**	**Action Space**	**Reward**
End-to-End Cooperation Strategies	Non-Guided	**[Chen, Subagdja e Tan 2019]**	PPO	Yes*	Continuous	Discrete	Dense
		[Geng et al. 2019]	PG	Yes	Discrete	Discrete	Dense
		[Cruz et al. 2020]	DQN	Yes*	Discrete	Discrete	Dense
		[Pham et al. 2018]	Q-Learning	No	Discrete	Discrete	Dense
	Goal-Guided	**[Yue, Guan e Xi 2019]**	Q-Learning	No	Discrete	Discrete	Dense
		[Venturini et al. 2020]	Double DQN	Yes*	Discrete	Discrete	Sparse
		[Yu et al. 2020]	DQN	Yes*	Discrete	Discrete	Sparse
		[Jun, Kim e Lee 2019]	TD Actor-Critic	Yes	Discrete	Discrete	Sparse
		[Lin et al. 2019]	PPO	Yes*	Continuous	Continuous	Dense

* Neural Netowrks with Convolutional Layers.

4.4.2 Approaching common problems

Exploration-Exploitation dilemma: Like the single-agent exploration works, ϵ-greedy is the most adopted method to balance exploration and exploitation during the learning process [Liu, Nejat e Vilela 2013, Venturini et al. 2020, Pham et al. 2018, Luo et al. 2019, Jin et al. 2019, Cao, Sun e Yan 2019]. In [Cao, Sun e Yan 2019] a switching strategy based on collision risk is also used to choose the actions, in combination with a self-decay probability to smooth the switch. Alternatively, the Boltzmann distribution mechanism is used in [Yue, Guan e Xi 2019] to determine the probability of choosing one action considering the current state. Given that many solutions to this dilemma have recently been proposed and proved successful in different applications [Hester, Lopes e Stone 2013, McFarlane 2018, Ecoffet et al. 2021], it is possible to conclude that there is space to investigate different methods in the robotic exploration context.

Curse of dimensionality: To avoid the curse of dimensionality, one approach is to employ RL, specially the value-based algorithms, only for tasks that inherently have a limited and discrete set of states and actions, such as choosing for which node the agent should move next [Jin et al. 2019, Luo et al. 2019, Zhou et al. 2019]. A widely used strategy is to efficiently reduce the space representation. In [Pham et al. 2018], Fixed Sparse Representation approximation maps the original Q-values to a low-dimension parameter vector. As already discussed, DRL solutions can be employed to learn the low-dimensional state features of the high-dimensional state from the sensory data, and provide robust function approximation. Due to its great performance improvements in recent years, CNNs are being increasingly adopted as feature extractors of raw images or other high-dimensional sensor data [Chen, Subagdja e Tan 2019, Luo et al. 2019, Cao, Sun e Yan 2019, Venturini et al. 2020, Yu et al. 2020, Lin et al. 2019].

Reward shaping: When it comes to reward shaping, the approach of the MRS works is very similar to the identified in single-agent research. Heuristc functions are by far the most adopted approach, being designed in accordance with the application objective. For example, in goal-guided tasks it is usual to give a positive reward when an agent finds a target, as well as giving a positive reward for finding new unexplored areas in non-guided tasks. For both kinds of multi-robot tasks, it is common to apply a negative reward in case of collision between agents or with objects, and to specify rewards for maintaining connectivity or learning to cooperate with other agents. Again, the most common approach is employing dense rewards, probably because it usually accelerates the agent's comprehension of how it should behave.

Learning convergence: To decrease the amount of time spent on training and to improve learning effectiveness, some MRS works make use of Transfer Learning techniques. One example are the *Curriculum Learning* (CL) techniques, in which the agents learn from increasingly difficult scenarios to progressively acquire complex skills. In [Chen, Subagdja e Tan 2019], CL is applied to simplify and direct the learning process in a teacher-student fashion, exposing the agents to four different environments with different complexity or difficulty. The strategy is tested with and without curriculum learning, and only succeed when the technique is applied. In [Geng et al. 2019], new obstacles are gradually introduced to the training environment, stimulating the agents to explore with the smallest number of collisions when faced with different dynamic environments.

A similar approach is used in [Jin et al. 2019], in which a target selection policy is pre-trained in obstacle-free environments, and then new obstacles are added during training. This technique makes the algorithm converge much faster than classic multi-agent

DDPG. In [Venturini et al. 2020], the agents trained in sparse environments are able to quickly adapt to clutter scenarios. Some methods use *Experience Replay* to improve data efficiency. Both [Venturini et al. 2020] and [Liu, Nejat e Vilela 2013] enable experience sharing between different robots, which allows faster convergence of the learning algorithms. Prioritized Experience Replay is proposed in [Hu et al. 2020], which samples important experience data more frequently by calculating the priority of each state transition.

From the performed academic research review and to the best of our knowledge, no works investigate the application of DRL solutions for mapless unknown environment exploration focusing on non-guided applications. In this work, mapless exploration refers to strategies that do not build an explicit map of the environment to decide where to go next. In that context, no mapless methods aim at efficient area coverage without pre-determined goal positions. As previously mentioned, some limitations arise from founding the exploration decisions on an online built map. First, the computational costs rapidly increase with the expansion of the explored area. Also, the efficiency of the exploration strategy heavily relies on precise maps [Juliá, Gil e Reinoso 2012]. Furthermore, when a multi-robot system performs a collaborative exploration, the maps generated by each agent are often shared between the team. The shared maps must usually be merged, which is not a trivial task, often associated with high computational costs [Velásquez Hernández e Prieto Ortiz 2020].

At the same time, there are applications where an entire area must be covered, but the final goal is not mapping the environment, as search and rescue or multi-target search. In situations with limited memory and processing resources, such applications could benefit from an exploration strategy that does not rely on an accurate world map. Therefore, using the information of how recent works are dealing with problems of RL applied to robotics, the following sections present the proposal of a mapless exploration framework suitable for n robots based on DRL.

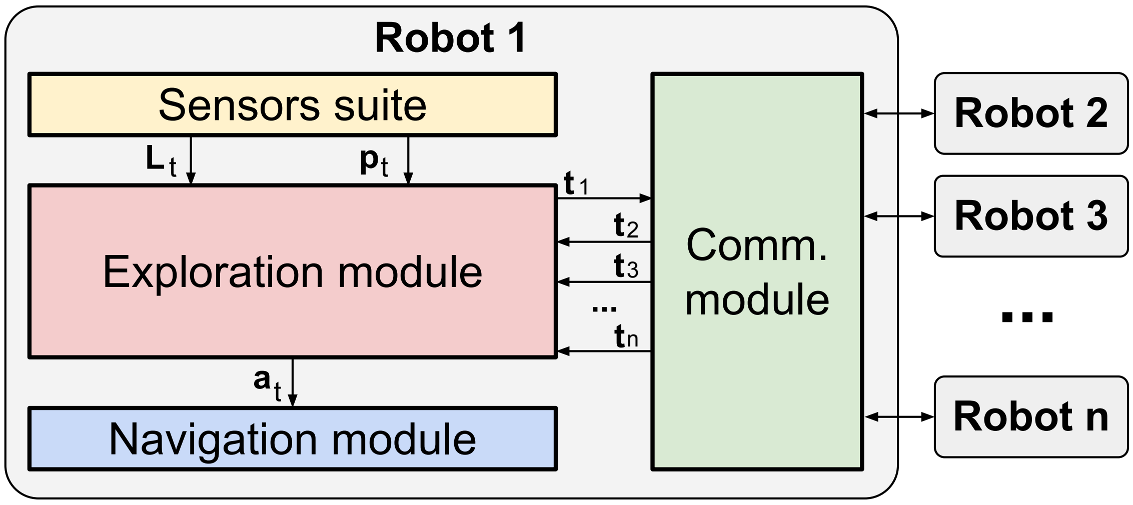
5 MAPLESS COOPERATIVE EXPLORATION USING DRL

The main problem tackled by this work is the mapless multi-robot collaborative exploration. In this thesis, mapless exploration refers to a strategy that does not build an explicit map of the environment. Instead, it keeps track of a fixed-size quantity of sensor information that is not heavily processed, such as laser measurements and the robot's trajectory. The work focuses on indoor environment exploration, where GPS signal is unavailable. We assume that the environment is completely unknown, which means the agents do not have prior access to the environment's map. Thus, the team's goal is to efficiently explore a 3D environment in a 2D fashion as fast as possible. In other words, the purpose of the exploration problem is to find the set of waypoints ((x, y)) that result in the fastest cooperative environment coverage. In the context of RL, as detailed in Section 3.1, this problem can be regarded as a sequential decision-making process, where the agent chooses a sequence of actions to maximize the accumulated discounted reward. So, in this work, the exploration problem is formulated as a Markov Decision Process.

To approach the collaborative exploration problem, a system was designed according to the architecture illustrated in Figure 5.1. A multi-robot system N_R formed of $n \in \mathbb{N}^*$ robots R_1, R_2,...R_n is considered. Each agent that composes the robot network comprises the same building blocks: sensors suite, communication, exploration, and navigation modules. The exploration strategy is decentralized, meaning each robot locally runs a system that determines the individual exploration policy. The idea is to obtain agents capable of exploring the environment alone, but that can optimize their decision-making process when information from other agents is received. As previously stated, in the proposed method, the exploration happens independently from any mapping process.

The system information flow happens as follows: at time t, the **Sensors Suite** (SS) sends laser measurements ($\mathbf{L}_t$) and the robot's current pose ($\mathbf{p}_t$) to the **Exploration Module** (EM). Using the updated pose, the EM assembles the robot trajectory ($\mathbf{t}_1$), and sends it to the **Communication Module** (CM), that tries to broadcast it to other robots. The CM also receives other robots trajectories ($\mathbf{t}_2$, $\mathbf{t}_3$,..., $\mathbf{t}_n$) if available, and sends them to the Exploration Module. Using the information from Sensors Suite and Communication Module, the Exploration Module defines the next action the robot should perform ($\mathbf{a}_t$). Finally, the **Navigation Module** (NM) receives the selected action and translates the high-level abstraction into electrical and mechanical variables that control the robot for performing the selected action.

Figure 5.1 – Simplified diagram of the proposed exploration system, suitable for **n** robots.



Source: The Author.

The main focus of this work is the exploration strategy. Therefore, the proposed system was designed to be modular, meaning that the blocks connecting to the exploration module are not limited to a specific method. Provided some determined assumptions are respected, the sensors, communication, and navigation can use different implementations. A considered premise is that at each time step, the agents try to communicate with each other. Data can be transmitted and received between the agents when the communication link is established. The mechanisms that the Communication Module employs to establish communication can vary. On the other hand, the Navigation Module is responsible for controlling the robot to perform the action selected by the exploration policy. The robot action space is discrete, and the robot moves in fixed steps. Therefore, the navigation system can use different techniques to control the robot, such as path planning methods, or simply using odometry data to rotate the robot in the target direction and to activate its propellers to drive a specific distance. It is valid to notice that the better the performance of the peripheral blocks, the better the performance of the exploration strategy.

The Sensor Suite and the Exploration Module are explained in further details in the next sections. Section 5.1 defines which sensors compose the sensors module and how odometry data are combined to estimate the robot's position and rotation in the environment. The main contribution of this work is documented in Section 5.2: the exploration module. The core of the exploration is an end-to-end decision-making module based on the deep reinforcement learning algorithm Proximal Policy Optimization. Section 5.2.1 presents how sensor data and information received from other agents are pre-processed and combined into the environment state. The set of states, rewards, and actions, as well as the training and inference processes, are characterized in Section 5.2.2.

5.1 Sensors suite

The Sensors Suite is responsible for sensing the world around the agent and for tracking its movements. The proposed exploration strategy assumes that the vehicle is equipped with an odometry module and a laser distance sensor. Because the proposed system is not limited to a specific kind of robot (e.g., UGV, UAV, AUV), the sensors that compose the odometry module can vary (e.g., wheel odometer, inertial measurement unit (IMU), camera). On the other hand, the exploration algorithm was designed to specifically receive data from a 2D laser scanner capable of sensing all directions around the robot.

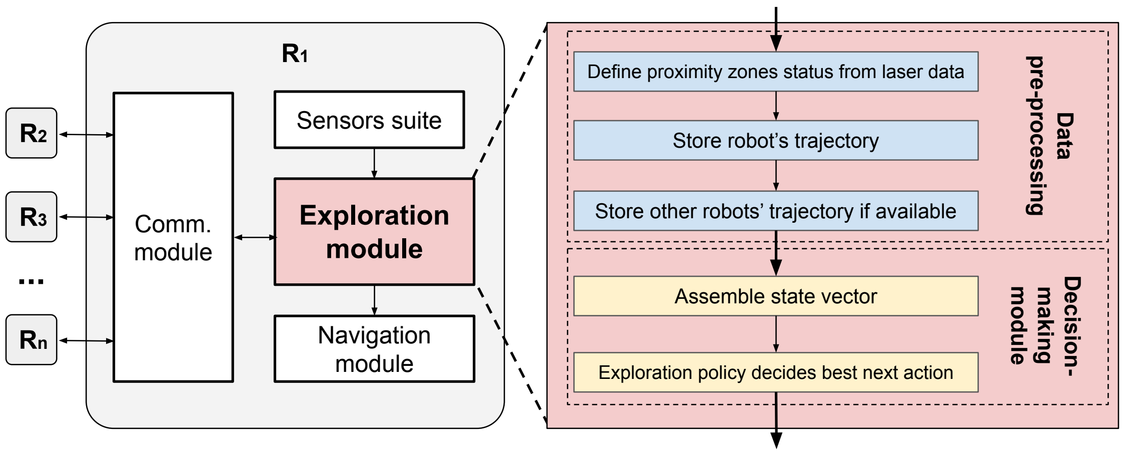
A mobile robot must be able to localize itself in the environment to perform any autonomous navigation task. This work focuses on indoor environments, so using GPS information to determine the agents' localization is not appropriate. Therefore, to approach the localization problem, the information from the odometry module is employed to perform position tracking, or dead reckoning. Considering that a robot's pose is the specification of its 2D position and its orientation ($\mathbf{p} = (x, y, \theta)$), dead reckoning is the process of estimating a robot's current pose by using a previously determined pose. Different localization methods could be employed without changing the underlying exploration architecture, such as raw odometry data, or the Extended Kalman Filter (EKF) [Fujii 2013].

In a system formed of $n \in \mathbb{N}^*$ robots, each robot is initialized in a random and distinct position. All robots are localized in the same coordinate frame. Robot r_1 is considered the coordinate frame reference, being initialized in a position defined as $(x = 0, y = 0)$. The other robots starting positions are defined considering their distance to robot r_1. Finally, the output of the Sensor Suite is the robot current pose ($\mathbf{p}_t$) and a vector $\mathbf{L}_t$ with the obtained laser measurements.

5.2 Exploration Module

As previously mentioned, the Exploration Module (EM) configures the main contribution of this work. Figure 5.2 expands the exploration module so its sub-processes can be visualized. The processes are divided into two main stages: data pre-processing and decision-making module. The following sections describe the two stages in detail.

Figure 5.2 – Exploration Module in details.



Source: The Author.

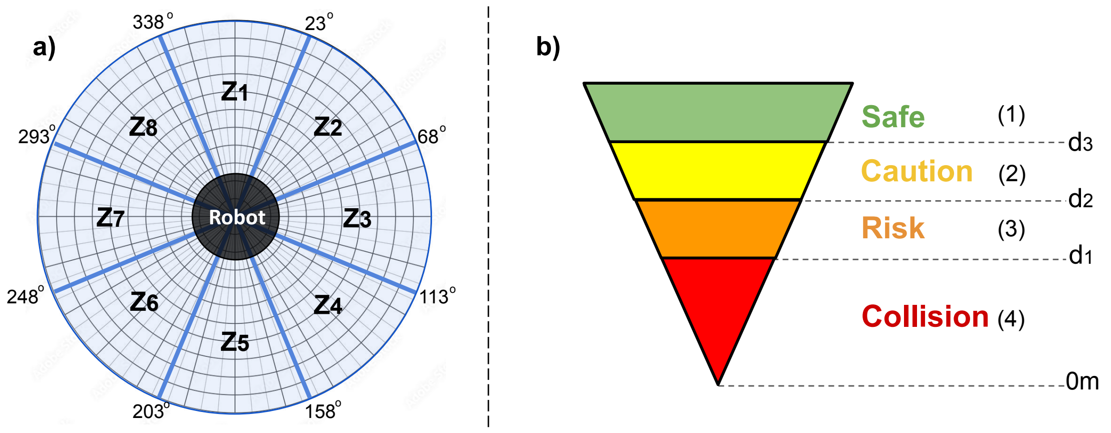
5.2.1 Data pre-processing

The data pre-processing stage transforms raw sensor measurements received from the Sensors Suite into data that is ready to be processed by the exploration policy. In this work, the exploration policy is the function approximated by a deep neural network (DNN). As any deep learning process, the DNN is trained to optimize its parameters and find patterns from a provided dataset. In the case of DRL, the dataset inputs are the environment states. Therefore, defining what kind of information comprises a state and how this information is presented to the policy model is one the most crucial factors for the success of the exploration strategy.

A fundamental precondition for autonomous navigation is collision avoidance. To be able to avoid collisions, the agent must receive information about the distance of obstacles in its surroundings. Furthermore, knowing which directions lead either to open areas or closer to obstacles can help determine the critical exploration question: where to go next. In this context, the proposed method uses the Sensors Suite's laser measurements to extract this information. A possible strategy would be simply using raw laser information to compose the environment state. However, this approach results in a large state space and is very susceptible to noisy measurements, which can make exploration policy convergence difficult. Therefore, the proposed solution employs an encoding method to compact the laser measurements, decreasing the state space dimensionality and reducing the impact of measurement noise. Figure 5.3 illustrates the schematic of the proposed encoding method.

The encoder idea is to divide the laser measurements into eight proximity zones

Figure 5.3 – a) Schematic representing how the 360 laser measurements spaced by one degree are equally divided into eight proximity zones. b) Scale that illustrate the proximity status each zone can assume depending on the measured distances.



Source: The Author.

($\mathbf{Z} = (Z_1, Z_2, ..., Z_8)$), as shown in Figure 5.3.a. Throughout the text, the proximity zones can also be referred to as proximity regions. Given that the $360°$ around the robots are equally divided, each zone represents the lasers contained in $45°$. A scale with four possible statuses is proposed to comprise the measurements into a single value, as represented in Figure 5.3.b. The possible statuses are *safe*, *caution*, *risk*, and *collision*, that are represented in the state vector by the values 1, 2, 3, and 4, respectively. A zone assumes a different status depending on the distances measured by its lasers. It is defined that at least k_l laser measurements must fit into a distance condition to make a region assume a worse status. For example, consider a proximity zone Z_M, with 45 measurements l_m. If at least k_l laser readings fit into the condition $0 \leq l_m < d_1$, the status of Z_M becomes *collision*, regardless of the other measurements. Otherwise, if k_l readings fit into condition $d_1 \leq l_m < d_2$, the status of Z_M becomes *risk*. If that is still not the case, when k_l laser readings fit into condition $d_2 \leq l_m < d_3$, the status become *caution*. Finally, if there are not k_l measurements that result in a distance smaller than d_3, then the zone is considered *safe*. The exact distances d_1, d_2, and d_3 depend on the dimensions of the employed robot, and the minimum number of laser readings k_l is empirically determined.

Besides the knowledge of the agent's surroundings, information about already visited regions can impact the exploration decision-making process. Traditional methods usually extract such information from the environment map built during exploration. Because the goal is to perform mapless exploration, another form of comprising visited regions is proposed. At each time step t, the agent has access to its current pose in the environment. That means it is possible to keep track of the robot's trajectory by saving its poses while the agent moves. Here, we are only interested in the robot positions, represented by a (x, y) point. As explained in Section 5.2.2, the robot action space is discrete, and the robot moves at fixed steps, which facilitates this trajectory representation. Similarly to the laser pre-processing reasoning, a possible approach would be to use the entire robot trajectory to compose the environment state. However, this strategy makes the state rapidly grow as the exploration happens, causing an increase in computational effort and making policy convergence challenging. Therefore, the proposed method represents the robot's trajectory as a vector ($\mathbf{t}_n$) with a fixed size k_{tr}.

Suppose the trajectory vector ($\mathbf{t}_n$) stores k robot's positions. Considering that each position is a (x, y) point, the total size of the trajectory vector is $k_{tr} = 2k$. When the robot is initialized in the environment, the two first vector elements correspond to the robot's starting coordinates, and the remaining vector values are set to zero. At each time step t, the robot takes a new step. While the number of steps is smaller than k, the points of the trajectory vector $\mathbf{t}_n(t-1)$ are shifted so that the new current robot position corresponds to the first vector elements. When the robot performs more than k steps, trajectory information is processed so that $\mathbf{t}_n$ size remains fixed. Three methods for processing the trajectory are proposed:

- **Drop Random Points**: Except for the first and last trajectory points (that represent the agent's current and starting position, respectively), random points are dropped from the trajectory vector.
- **FIFO**: As the name implies, the first position to fill the vector is the first one to be dropped. Only method that does not preserve the agent's starting position.
- **Merge Last Positions**: The first 20% of the vector elements are updated normally, and the last position contains the agent's starting position. The remaining points are sequentially merged, so that the resulting point is the average of two points. This method aims to provide more precise knowledge on the most recently visited positions without losing all data about past trajectories. So, although information about previously visited regions becomes more sparse, the vector maintains the information about where the robot started and does not completely neglect older positions.

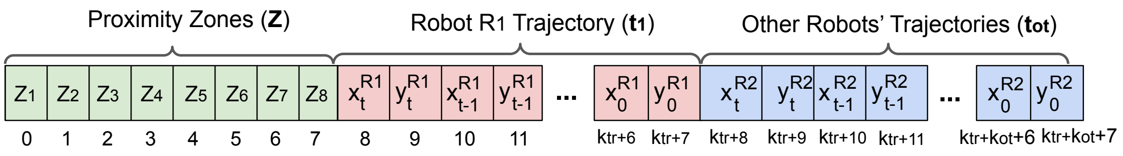
As previously described, the robots try to broadcast their trajectory vectors to the team. Considering that the exploration is collaborative, it is intuitive to affirm that an agent can benefit from knowing which regions were visited by other agents. That way, it can make more intelligent decisions and avoid visiting already explored areas. The same logic of avoiding rampant state growth must be considered when incorporating shared trajectories into the state. So, the proposed method also combines the other robots' trajectories in a fixed-sized vector ($\mathbf{t}_{ot}$). At time $t = 0$ all $\mathbf{t}_{ot}$ positions are initialized as zero. Each robot tries to send its trajectory vector to the other robots at every new step. For a system composed of n robots, each robot receives $n - 1$ trajectory vectors from the other robots. Then, each agent combines the received information into a single vector of fixed size k_{ot}. If only two robots are considered, both trajectories are simply concatenated. If more robots are collaborating, duplicate points are eliminated, the vectors are concatenated, and random points are dropped until the desirable vector size is achieved.

5.2.2 Decision-making Module

In general terms, the decision-making module combines the data processed by the former stage into the environment state, applies the state into the exploration policy, and returns the robot next action. The exploration problem was already formulated as a MDP, and the proposed strategy uses DRL to search for the optimal policy. The elements of the RL problem, which are the state space, the action space and the reward function, are defined as follows:

States: The environment state is the result of the concatenation of the proximity zones (**Z**), the robot trajectory ($\mathbf{t}_n$) and the other robots' trajectories ($\mathbf{t}_{ot}$), resulting in a vector with fixed size $8 + k_{tr} + k_{ot}$. Figure 5.4 illustrates the state vector representation of robot R_1 at time step t. To facilitate the comprehension of the vector structure, the other robots trajectory only contains the trajectory of robot R_2. However, the final state vector structure and size would remain the same, regardless of how many agents trajectories are received.

Figure 5.4 – State vector representation of robot R_1 at time t, considering that communication was established with robot R_2.


Proximity Zones (Z)
Robot R1 Trajectory (t1)
Other Robots' Trajectories (tot)
Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8
x_t^{R1} y_t^{R1} x_{t-1}^{R1} y_{t-1}^{R1} ... x_0^{R1} y_0^{R1}
x_t^{R2} y_t^{R2} x_{t-1}^{R2} y_{t-1}^{R2} ... x_0^{R2} y_0^{R2}
0 1 2 3 4 5 6 7 8 9 10 11
ktr+6 ktr+7 ktr+8 ktr+9 ktr+10 ktr+11 ktr+kot+6 ktr+kot+7

Source: The Author.

Action: The action space is a probability distribution of discrete actions including the possible directions the robot can move towards at a fixed step. Four possibilities are considered: move **forward**, **backwards**, **left**, or **right**. This set of actions makes the exploration model an end-to-end solution, because it directly translates the sensor measurements into movements.

Reward: In general, the approach for reward shaping of most research works that apply DRL to mobile robotics exploration is to elaborate heuristic strategies [Garaffa et al. 2021]. Also, in similar applications dense rewards usually perform better than sparse rewards [Mohtasib, Neumann e Cuayáhuitl 2021]. Dense systems adopt intermediate rewards which are assigned to the agent in many different states. With that in mind, the reward function was designed aiming to punish collisions and encourage the exploration of new regions. The basic structure for reward function is illustrated in Listing 5.1.

Listing 5.1 – Basic reward function structure.

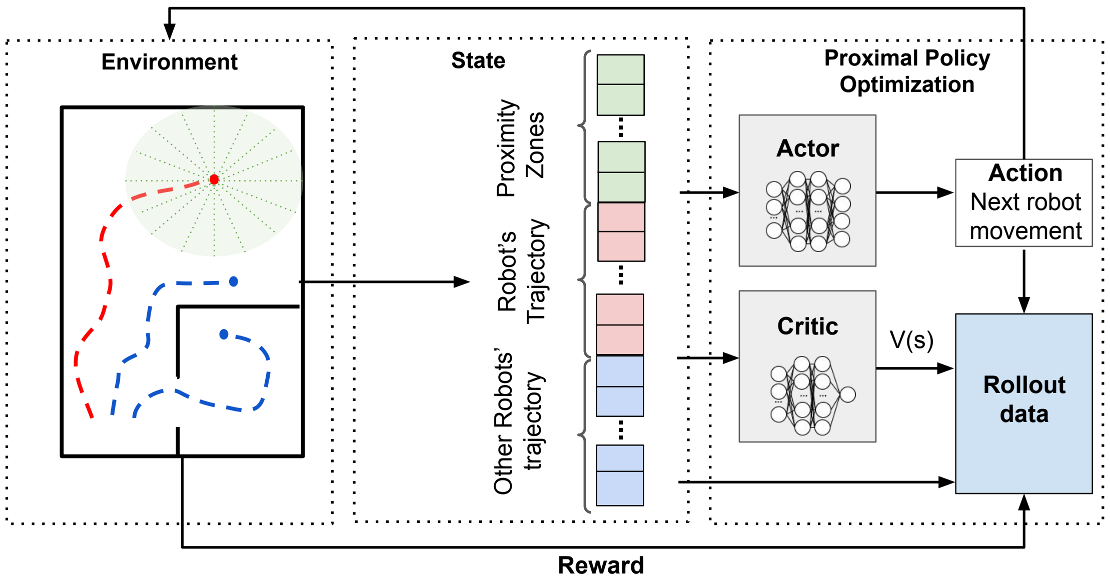
```
if collision:
    R(t) = max_penalty

else if (explored_rate >= max_explored_rate):
    R(t) = max_reward

else if (explored_rate (t) - explored_rate (t-1))>0:
    R(t) = R_1

else if (explored_rate (t) - explored_rate (t-1)) == 0
and new_position:
    R(t) = R_2

else if (explored_rate (t) - explored_rate (t-1)) == 0
and old_position:
    R(t) = R_3
```

If a robot collides, the reward assumes the value of *max_penalty*, which must be a negative value. If it does not collide and the next step results in a coverage equal or higher than a pre-defined value (*max_explored_rate*), the exploration is considered successful and the reward is *max_reward*, a positive value. Considering that the computation of the exploration rate is performed using the positions visited by all agents in the team, this reward should encourage cooperation between agents. If the robot next step does not result in a collision nor in a successful exploration, but the robot new position increased the explored region rate, the reward R_1 is proportional to the increase in the exploration.

If the robot does not explores new regions, but its current position has never been visited before, the reward is R_2. Finally, if none of the previous scenarios occur, it means the robot has already visited its current position. The reward R_2 is a negative value much smaller than *max_penalty*, that should encourage short paths.

The algorithm used to optimize the policy parameters is the Proximal Policy Optimization, whose equations and advantages were described in Section 3.3. The algorithm uses an actor-critic model, where actor and critic are fully-connected neural networks with two hidden layers of 64 neurons. The exploration is decentralized, meaning each robot has its own neural network. A simplified training framework of the proposed DRL actor-critic model is shown in Figure 5.5. Let us consider the training of a single robot. During the training phase, the agent performs several policy rollouts, or in other words, plays several games. Each rollout has a fixed number of steps (*steps_roll*). The agent and the neural networks' parameters are initialized when training starts.

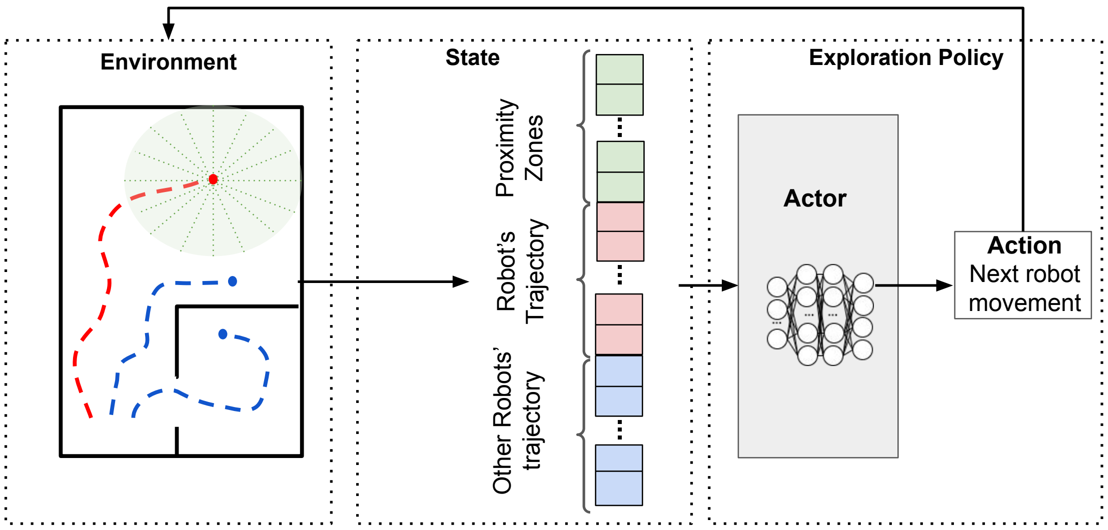
Figure 5.5 – Simplified schematic of how data is gathered at each rollout during the training phase.



Source: The Author.

The initialized agent collects data and assembles the state vector. The state is used as input to the critic, which outputs the state value ($V(s)$), and to the actor that returns the next action (a_t). The robot acts, resulting in a new environment state. Taking the selected action has consequences in the environment (e.g. robot collided, new region is visited, etc.). Such consequences are considered to determine the reward associated with the state-action pair. At each step, the state, action, state value, reward, and action probabilities are stored as rollout data. When a rollout is over, such information is used to update the actor and critic neural networks parameters according to the Equations presented in 3.3. One training process has a maximum number of steps (*max_steps*), which means that the number of NNs updates is equal to the quotient of *max_steps* and *steps_roll*. Figure 5.6 illustrates a simplified inference framework, where a trained robot performs exploration. The critic network and the reward computation is removed, and the resulting actor network configures the exploration policy.

Figure 5.6 – Simplified schematic of the inference framework after training is done.



Source: The Author.

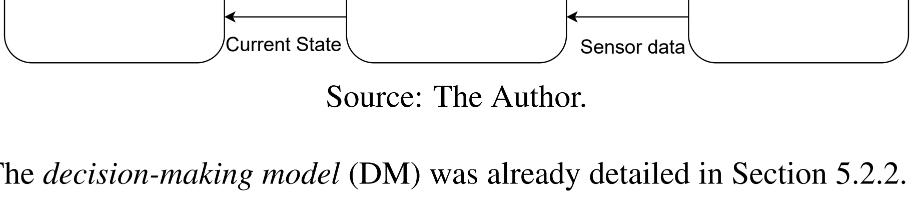
6 EXPERIMENTS

This chapter describes the experimental setup employed to implement, validate and test the proposed mapless collaborative exploration architecture. Section 6.1 presents the simulation framework built to train and test the agents in different environments. Section 6.2 describes the experiments elaborated to validate the proposed strategy in simple rooms and to define the parameters and training configurations that yield the best performances. Finally, Section 6.3 describes the experiments performed in more complex environments to compare the performance between different exploration methods.

6.1 Simulation

In order to train and test the proposed exploration architecture, a simulation framework was built. Figure 6.1 illustrates the three main communicating elements that compose the framework logic: the simulated world/environment, the robot controller, and the decision-making model. The simulated world and the robot controller were implemented using OpenAI Gym, a widespread environment for developing and testing learning agents [Brockman et al. 2016]. A customized environment was implemented to represent the evaluated maps as virtual grids. Considering that the robot moves with fixed steps of $20cm$, each grid cell presented an area of $20cm$x$20cm$. The simulator simplifies the robot representation as a circle that occupies a $20cm$x$20cm$ cell. It can move forwards, backward, right, or left without turning. The simulator also used vectorized environments, a method for stacking multiple independent environments into one. That way, the agent gets information of n environments per step. The developed simulation environment aims to increase data collection efficiency, make the training process faster, and facilitate parameters' tunning.

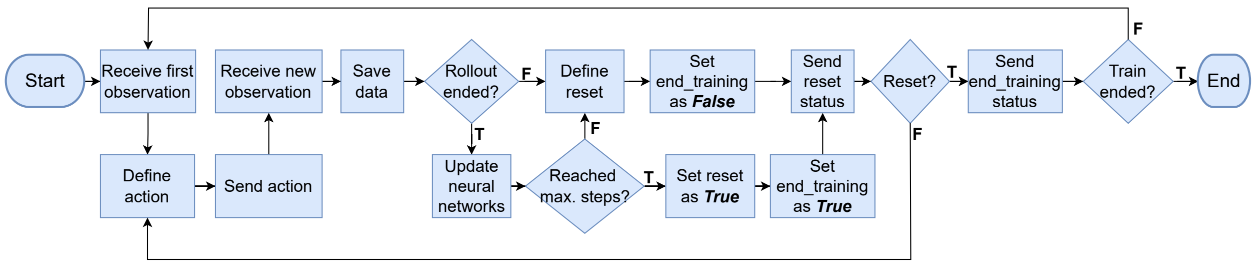
Figure 6.1 – Simulation framework elements and communication flow.



Source: The Author.

The *decision-making model* (DM) was already detailed in Section 5.2.2. It works as the "brain" of the robot, using processed sensor data as the input of the exploration policy neural network, and returning an appropriate next action. The flowchart in Figure 6.2 represents the communication with the robot controller from the decision-making model perspective. At the start of a new train, the DM receives the robot's current state from the controller and uses it to define the next action. Then, it communicates the action to the controller and waits until the next state is received. The data resulting from this interaction is saved. If there is no need to reset the training, the last received state is used to define the next action, and the process is repeated normally. This process is repeated until the rollout phase is over. When a rollout ends, the saved data is used to update the actor and critic neural networks. The reset is defined depending on the last state, and the end training is defined depending if the robot reached a pre-defined number of steps. Both statuses are sent to the controller, and the whole process is repeated until training is over.

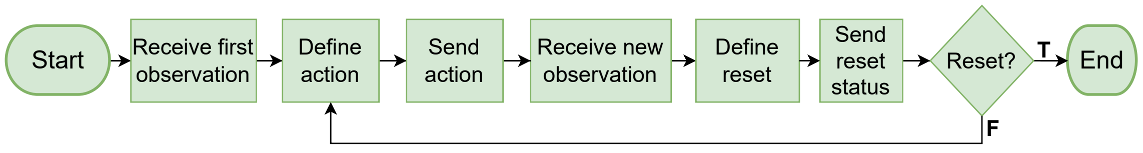
Figure 6.2 – Communication between decision-making model and robot controller from the decision-making model perspective during a training process.



Source: The Author.

Figure 6.2 represents the decision-making model communication with the robot controller when a trained agent is used for inference. All the steps related to saving data and updating the neural networks are removed. The DM receives the first state, defines the following action, and communicates the action to the controller. After a new state is received, it evaluates if the taken action resulted in a reset. If that is not the case, the inference continues. Otherwise, the DM thread is killed.

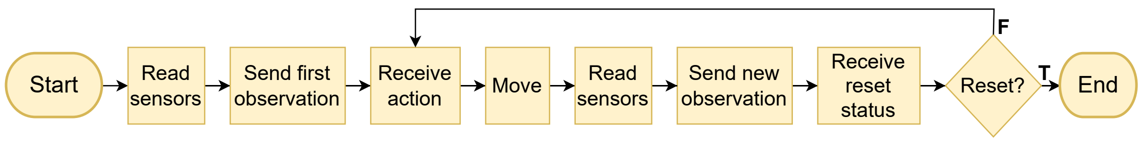
Figure 6.3 – Communication between decision-making model and robot controller from the decision-making model perspective during a inference process.



Source: The Author.

The *robot controller* is the interface between the decision model and the simulated world. It is responsible for receiving sensor measurements from the simulated environment, processing the received data, and transmitting the current state to the decision-making model. Then, it gets the action defined by the model and translates this high-level information into appropriate commands to move the robot in the simulated world. In our simulator, such command is just updating the robot's position in a discrete fashion. The communication with the decision model from the controller perspective is represented in Figure 6.4.

Figure 6.4 – Robot controller thread.



Source: The Author.

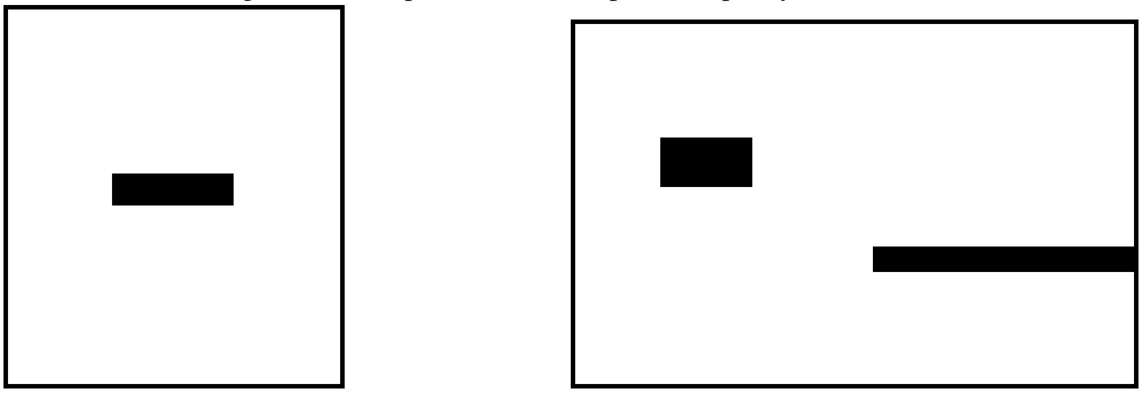
When the training of one robot is performed, a main process in the robot controller launches a single robot thread. If a reset happens, this thread is killed, and if the train is not over, it is launched again. At each reset, the robot is initialized in a random free position in the environment. For inference the process is the same, but the thread is not relaunched after reset. In both training and inference, the decision-making model and the simulated world continuously run throughout the process. The process is syncronous, so all robots in the simulation move one step per timestep. The Bresenham's line algorithm [Kuzmin 1995] is used to determine the simulated laser measurements and the explored region around the robot. The code for the simulation framework can be found in: https://github.com/luizagaraffa/mobile_robotic_exploration_PPO

6.2 Method validation

The method validation consists in verifying if the exploration algorithm converges when the proposed methodology is implemented. It also is used to evaluate what are the training parameters, reward functions, trajectory methods, exploration radius, and training configurations that facilitate the algorithm convergence. Different reward functions are tested based on the structure proposed in Section 5.2.2, depending on the experiment characteristics. The trajectory methods proposed in Section 5.2.1, including Drop Random Points, Merge Last Points, and FIFO, are evaluated. The exploration radius refers to the area around the robot that is considered as explored. Different values were defined according to the environment dimensions.

As for training configurations, the experiments can use curriculum learning or learn to explore from scratch. Fine-tuning is a way of utilizing transfer learning where a model that has already been trained for one given task is furthered tuned to make it perform a second task. Therefore, in the curriculum learning trains, the agent first learns how to avoid collisions based on the proximity zones status, and then the resulting neural networks are finetunned so that the agent learns how to optimize the exploration path. On the other hand, trains where the agent learns how to explore from scratch mean that the agent has no previous training and must learn how to avoid collisions and how to optimize exploration in the same training. To perform validation, a really simple set of maps is considered, as illustrated in Figure 6.5. The dimensions and resolutions of the maps are presented in Table 6.1.

Figure 6.5 – Maps used for the exploration policy validation.



(a) Simple maze map (b) Simple room map

Source: The Author.

Table 6.1 – Validation maps' dimensions and resolution.

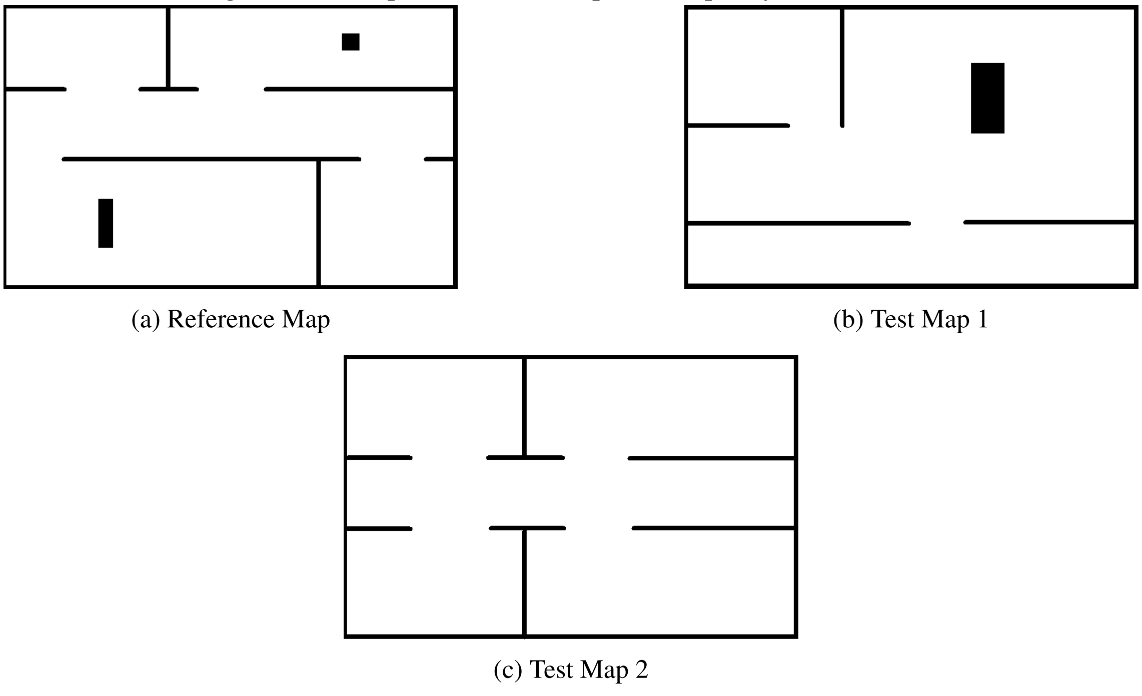
Map Name	**Resolution**	**Real Size (m)**
Simple Maze	8x9	1.6x1.8
Simple room	20x13	4.0x2.6

As previously described, during training, the number of data saved in a rollout phase and the number of updates in the actor-critic neural network are fixed and predefined. On the other hand, an episode can be composed of a different number of steps. An episode starts when the robot is initialized in the world and ends when it meets a reset condition. Depending on the experiment goal, the robot will reset if it collides, if the number of steps exceeds a maximum number of steps per episode, or if the robot has explored more than a percentage of the environment. The parameters that result in the highest ratio between explored environment rate and path length are used in the experiments in more complex maps.

6.3 Comparison with baselines

To evaluate the proposed exploration framework in more complex maps and compare its performance with other exploration methods, the research work published by Haoran Li *et al.* is used as a reference [Li, Zhang e Zhao 2019]. The work also proposes an exploration strategy based on Deep Reinforcement Learning, called Fully Convolutional Q-network with Auxiliary task (AFCQN). It compares their results with the classic Deep Q-network (DQN) method. The paper contains the used maps with dimension and resolution information. So, some of the maps were reproduced as illustrated in Figure 6.6. The idea is to train and test our method in the same (or at least very similar) maps, and compare the obtained exploration region rate and path length with the AFCQN and DQN performance. Another simple baseline that was implemented for comparison is the *random walk*. The agent chooses a random action between moving forward, backward, left, or right as long as this action does not result in a collision. The exploration ended every time the path length was equal to 60m, and the explored region rate was calculated.

Figure 6.6 – Maps used for the exploration policy train and test.



(a) Reference Map

(b) Test Map 1

(c) Test Map 2

Adapted from: [Li, Zhang e Zhao 2019]

The experiments using the baseline maps consist in training the agent(s) in the Reference Map, and testing the resulting exploration policies in the Reference Map itself, Test Map 1, and Test Map 2.The experiments in the test maps are used to evaluate the method's generalization capacity or, in other words, to evaluate how well it performs

Table 6.2 – Baselines maps dimensions and resolution.

Map Name	**Resolution**	**Real Size (m)**
Reference Map	40x25	8.0x5.0
Test map 1	40x25	8.0x5.0
Test map 2	40x25	8.0x5.0

in an environment that is different from the training one. The experiments are perform for one and two agents, and the aspects evaluated are path length, exploration rate, and generalization capability. The details of the tests and training parameters are described in detail in the following section.

7 RESULTS AND DISCUSSION

This chapter presents and discusses the results obtained from the experiments proposed in Chapter 6. Section 7.1 defines the parameters used in training and presents the validation results for one and two robots in the simple maze and the simple room environments. Section 7.2 presents the results for a single robot and for two robots in more complex environments, comparing the achieved exploration efficiency and generalization capability with different exploration methods.

7.1 Method validation

The proposed mapless exploration framework was validated in the simple maps illustrated in Figure 6.5. Different configurations were employed for training the agents, and the next sections presents which reward functions, model parameters, trajectory logic, and exploration radius resulted in the most efficient exploration. As previously mentioned, the configurations that achieve the best performance in the validation phase are employed in the more complex baseline maps.

7.1.1 Training parameters

As described in Section 3.3, one of the Proximal Policy Optimization algorithm advantages is that it provides a good balance between ease of implementation and parameters tuning. At the same time, proper parameter initialization can be decisive in improving policy performance. Different works that use PPO were consulted [Schulman et al. 2017] [Chen, Subagdja e Tan 2019], and the most commonly used parameters were selected as described in Table 7.1. Learning Rate (LR) annealing was used as described by Equation (7.1), where U represents the total number of policy parameters updates during training, u_t represents how many updates have been performed at time t, and $LR_{t=0}$ represents the initial learning rate value. The total number of updates (U) can be defined as the quotient of the total number of training steps and the number of steps per rollout. This way, the LR decreases linearly as the number of policy updates increases.

$$LR_t = \frac{1.0 - (u_t - 1)}{U} LR_{t=0} \tag{7.1}$$

Table 7.1 – Set of PPO parameters.

Parameter	Best Value
Clip Coefficient (ϵ)	0.2
Learning rate initial value	0.0025
Value Function Coefficient (c_1)	0.5
Entropy Coefficient (c_2)	0.01
Number of mini batches	4
Steps per Rollout	128
Number of vectorized environments	4

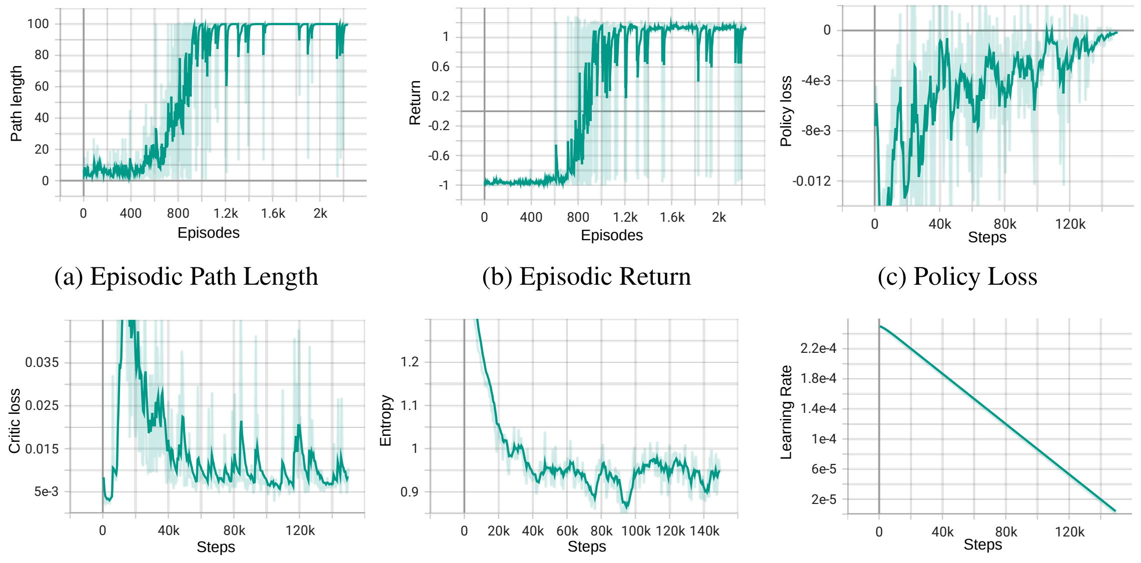
7.1.2 Single Robot

7.1.2.1 Simple maze

The Simple Maze experiment with a single robot is the most basic environment used to confirm the method convergence. The first experiment uses curriculum learning so that the robot first learns how to avoid collisions and then learns how to optimize the exploration path. Thus, the state vector only updates the proximity regions' status. The positions that would receive the trajectory remain filled with zeros. The reward is shaped differently than for experiments that aim at optimizing exploration. Considering the goal of collision avoidance, the agent receives a punishment of -1 when it collides. If the robot moves 100 steps without colliding, it receives a reward of $+1$. A small reward of 0.01 is defined every time a new position is visited to encourage the agent to move to different places instead of repeating the same movements (e.g., back and forward), as this behavior would also successfully avoid collisions. If none of the previous conditions is met, the reward is 0. An episode ends if there is a collision or the robot moves 100 steps. The agent is initialized in a random free position at the start of every episode. Training ends when the robot completes a total of $150k$ steps.

Figure 7.1 illustrates the results of the collision avoidance training for a single robot in the Simple Maze environment. The graphs were generated using the TensorBoard, a visualization tool developed by TensorFlow [Abadi et al. 2016]. To facilitate visualization, the curves were smoothed with a feature of Tensorboad that applies an exponential moving average to the metrics. The absolute values are represented by the fading colors. In Figure 7.1a, it is possible to observe that the path length is small in the first 600 episodes, when the agent has not learned yet to avoid collisions. As the training progresses, the path length converges to 100 steps per episode, which was the defined stop criteria. Figure 7.1b confirms that logic since the episodic return starts at -1, which is the punishment for collisions and then converges to a value slightly higher than 1, representing the sum of the 0.01 reward per new positions and the $+1$ reward for completing 100 steps.

Figure 7.1 – Training metrics of single agent in Simple Maze learning collision avoidance.



(a) Episodic Path Length (b) Episodic Return (c) Policy Loss

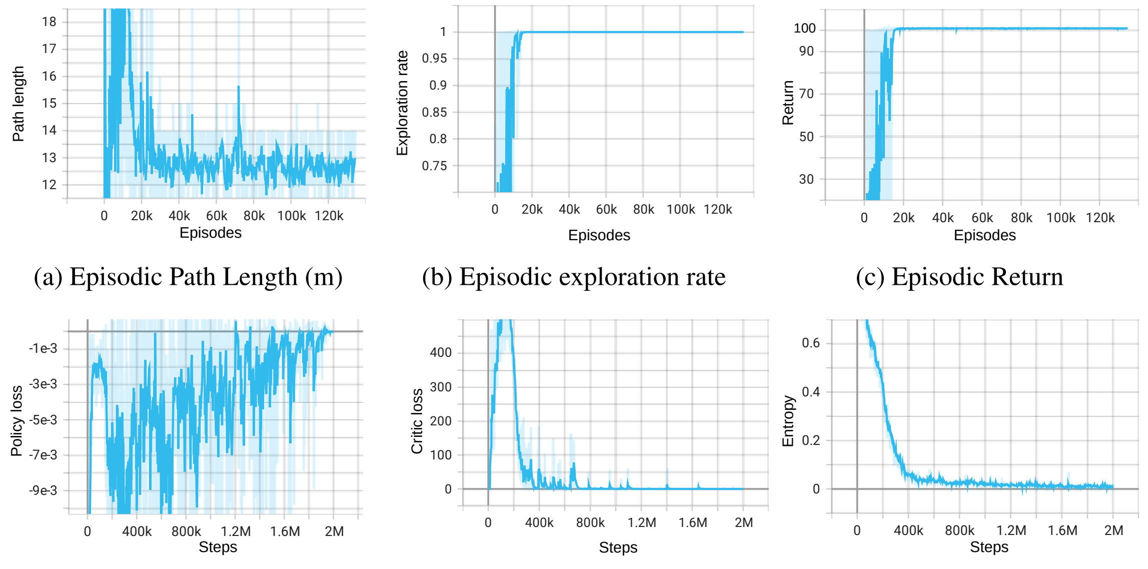
(d) Critic Loss (e) Entropy (f) Learning Rate

The Learning Rate is demonstrated in Figure 7.1f, linearly decreasing as the number of policy updates increases. That means that at the last episodes, the variation in the neural networks' weights updates become smaller. By that logic, in successful trains it is expected that the policy loss, critic loss, and the entropy adapt their behaviors to the number of steps, presenting higher variation in early episodes and gradually converging. As the behavior of the learning rate is the same for all trains performed in this work, only adapting the angular coefficient depending on the total number of steps, learning rate graphs are omitted in the following experiments.

The remaining graphs reinforce that the learning process was successful. Policy loss correlates to how much the policy changes during training. As mentioned in Section 3.3, the policy parameters are updated using gradient ascent. In Figure 7.1c, it is possible to observe that the policy loss increases over time, converging to a value close to zero. Figure 7.1d shows the critic loss, which correlates to how well the model can predict the value of each state. The critic parameters are updated using backpropagation and gradient descent. Its values should increase while the agent is learning and then decrease once the reward stabilizes, which is precisely what happens in the training process. Finally, entropy represents how random are the policy decisions. It should slowly decrease during a successful training process, as shown in Figure 7.1e. The variance in path length observed in the final episodes shows that the agent can still collide in certain situations. However, the overall results demonstrate that the resulting policy sufficiently learns how to avoid collisions, being suitable for the base neural network for exploration finetuning.

The next train goal is to teach the robot how to explore the environment with the smallest possible number of steps. The agent is initialized with the collision avoidance policy, whose parameters are updated during training. Considering that this experiment is supposed to validate the method and the that the maze presents a small size, it was considered that the agent only explores an area of $20cm$ around its position. Furthermore, 30 trajectory points form the state vector. The *drop random points* trajectory logic, presented in Section 5.2.2, is employed, meaning that when more than 30 positions are visited, random points are discarded. The reward shape changed accordingly to the new goal. The base for the reward function was presented in Section 5.2.2. If a collision happens, the reward is -1. If the exploration rate is 100%, the reward is 100. If the exploration delta, or in other words, the number of new explored cells, is higher than zero, the reward is the number of new explored cells. If no new cell is explored, the reward is -0.1. An episode ends if the agent collides or the exploration rate is 100%. Again, the agent is initialized in a random free position at the beginning of each episode. Training ends when the agent completes $2M$ steps. A high number of steps is chosen to evaluate how long the policy takes to optimize exploration, and to compare to the results of learning exploration from scratch.

Figure 7.2 – Training metrics of single agent in Simple Maze learning exploration by finetuning the collision avoidance policy.

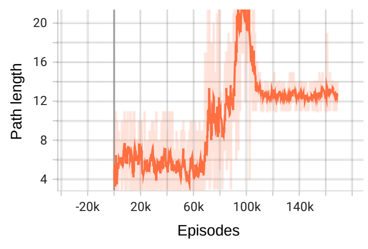


(a) Episodic Path Length (m)
(b) Episodic exploration rate
(c) Episodic Return
(d) Policy Loss
(e) Critic Loss
(f) Entropy

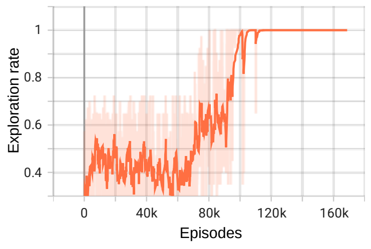
The results presented in Figure 7.2 demonstrate that the agent successfully learns to explore the environment efficiently. The episodic path length starts with a high number of steps, as the agent already knows how to avoid collisions. Around episode $20k$, the path length decreases and converges to a value close to 13 steps. Given that each step represents $20cm$, the average path length is $2.6m$. Given that the exploration radius was defined as $20cm$ around the agent, $2.6m$ is the approximate length to cover the maze optimally. Also around episode $20k$, the exploration rate converges to 100% and the accumulated reward to 100. The policy loss, critic loss, and entropy behaviors confirm the training convergence.

The final experiment in the Simple Maze employs the same configuration used for the exploration finetuning. However, the actor and critic weights are randomly initialized instead of using pre-trained neural networks. Figure 7.3 illustrates the obtained results. The agent takes more steps to learn how to optimize exploration compared to the finetuning results. This behavior makes sense, considering that when training starts the agent does not know how to avoid collisions, which is demonstrated by the small path lengths obtained in the first episodes. Around episode $60k$, path length significantly increases, which can be interpreted as the policy understanding that more rewards can be accumulated if collisions are avoided. Finally, around episode $120k$, the agent learns how to optimize the exploration, reducing the path length to around 13 steps, achieving 100% of exploration rate, and presenting an episodic return of 100. Policy loss decreases, and critic loss and entropy increase when the agent starts learning how to explore, to later converge to values close to zero.

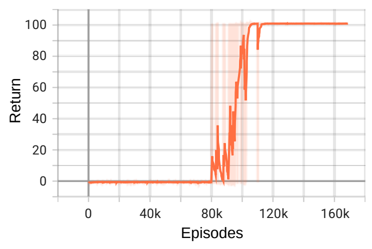
Figure 7.3 – Training metrics of single agent in Simple Maze learning exploration from scratch.



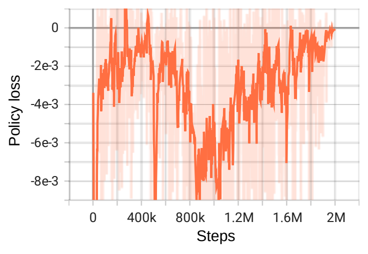
(a) Episodic Path Length



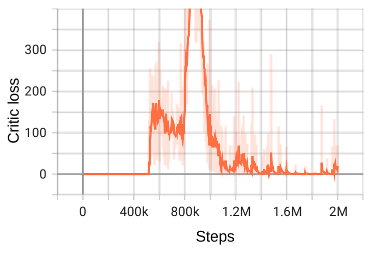
(b) Episodic exploration rate



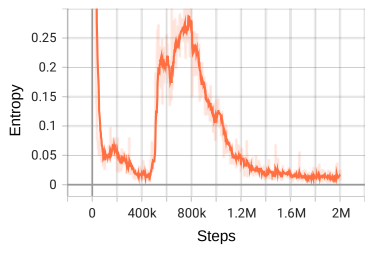
(c) Episodic Return



(d) Policy Loss



(e) Critic Loss



(f) Entropy

To test the performance of the resulting models, the achieved policies were used to perform inference in the Simple Maze map 50 times, and the path length and exploration rate were collected. The mean and standard deviation results are demonstrated in Table 7.2. The model trained using curriculum learning, referred in the Table as Finetunne, presented slightly better results, with an smaller average path length ($2.49m$) for a higher average explored rate (99%). However, the Simple Maze experiments demonstrated that the proposed mapless framework was able to achieve an efficient exploration with or without curriculum learning.

Table 7.2 – Inference results of exploration policies for a single robot in the Simple Maze environment.

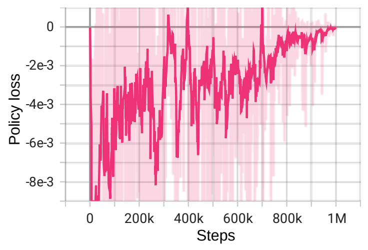
Map	**Train**	**Path Length (m)**		**Exploration Rate**	
		Mean	**Std**	**Mean**	**Std**
Simple Maze	Finetunne	2.49	0.03	0.99	0.01
	From scratch	2.73	0.01	0.98	0.02

7.1.2.2 Simple room

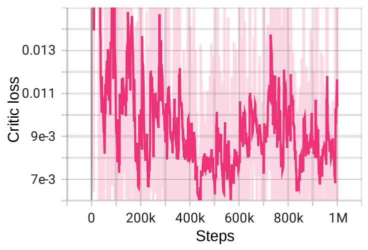
Similar to the Simple Maze experiments, the first set of trains in the Simple Room uses curriculum learning. Therefore, the first train's goal is to learn how to avoid collisions based only on the proximity regions' status. The reward function is the same as the one employed for the Simple Maze, except for the number of steps used as a success criterion. Because the Simple Room presents higher dimensions than the maze, the maximum number of steps per episode was considered 200 instead of 100. Thus, the reward was -1 for collision, $+1$ for completing 200 steps, $+0.01$ for discovering new positions, and 0 otherwise. The number of steps for ending the training process was increased to $1M$.

Figure 7.4 demonstrates that the agent can learn to avoid collisions, given that the path length converges to an average of 200 steps. The episodic return starts close to -1, the penalty for collisions, and ends with an average value slightly higher than 1. Interestingly, performing inference with the resulting model showed that the agent starts moving to different positions but ultimately ends up repeating the same limited movements. This behavior is consistent with the experiment design, given that the robot must perform 200 steps and there are less than 200 free positions in the map. Eventually, the robot visits all unseen cells, and the reward for performing any movement that does not end up in collision is the same.

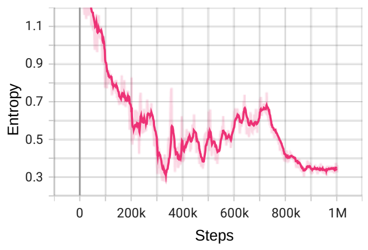
Figure 7.4 – Training metrics of single agent in Simple Room learning collision avoidance.



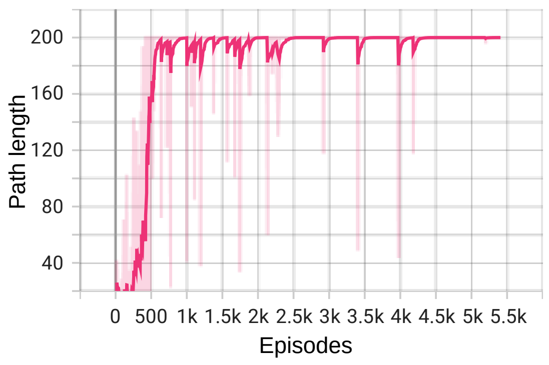
(a) Policy Loss



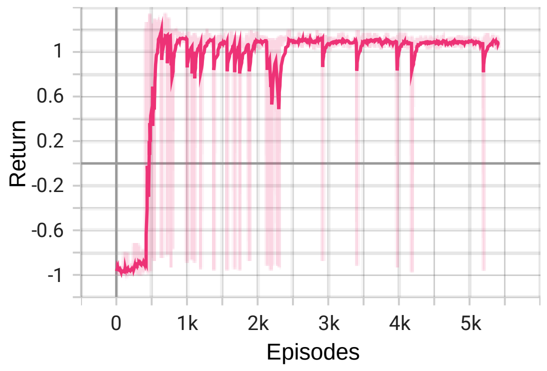
(b) Critic Loss



(c) Entropy



(d) Episodic Path Length

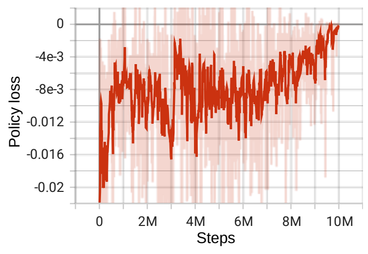


(e) Episodic Return

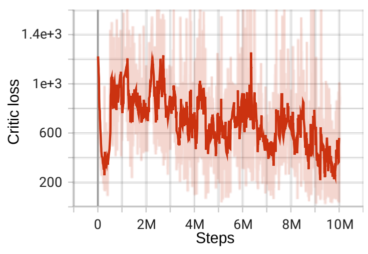
The resulting collision avoidance actor and critic neural networks were finetuned so that the agent learns how to optimize exploration. 50 positions form the trajectory vector, and the *drop random points* trajectory logic is employed. Considering the size of the map, the exploration radius was defined as $80cm$. The same radius is employed in all Simple Room experiments. To reinforce collision avoidance, a reward of -100 was determined for every collision. If the exploration rate is 98%, the reward is $+100$. If the agent explores previously unseen cells, the reward becomes the quantity of newly explored cells. If no new cells are explored, but the robot is in a position that was not visited before, the reward is $+0.5$. Finally, if no new cells are explored and the robot has already visited its current position, a penalty of -0.5 is applied. An episode ends when a collision happens or when the exploration rate is 98%. Training ends when the agent performs $10M$ steps.

The training metrics illustrated in Figure 7.5 demonstrate that the agent learns how to explore the environment, since the average exploration rate per episode converges to 98%. It is possible to observe that the path length starts with high values, as the agent already knows how to avoid collisions, decreases because of the penalty for visiting repeated positions, and then grows again to achieve the 98% exploration rate. The path length is expected to present a higher variation in the Simple Room experiments considering that, differently from the maze, the optimal trajectory can be significantly distinct depending on the robot's starting position.

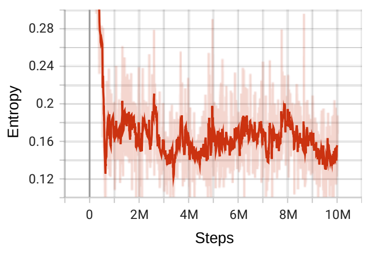
Figure 7.5 – Training metrics of single agent in Simple Room learning exploration by finetuning the collision avoidance policy.



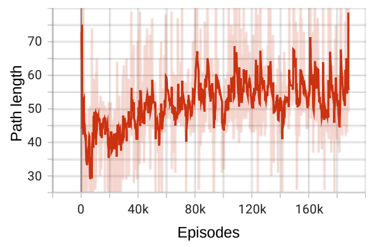
(a) Policy Loss



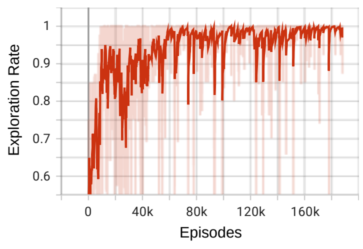
(b) Critic Loss



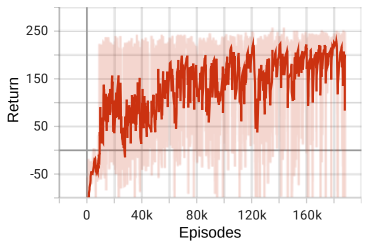
(c) Entropy



(d) Episodic Path Length



(e) Episodic Exploration Rate



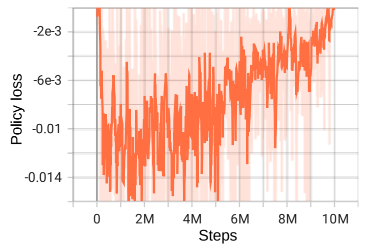
(f) Episodic Return

In order to teach the robot to explore the environment without curriculum learning, the same configuration used for the exploration finetuning was employed, except that the neural networks' weights are randomly initialized. Figure 7.6 illustrates the resulting training metrics. It is apparent that the agent first learns how to avoid collisions and later learns how to optimize the path for exploration. Unlike the Simple Maze experiments, the metrics suggest that learning exploration from scratch resulted in a more efficient exploration strategy in the Simple Room map, considering the tested configurations. In other words, the average path length converged to smaller values to cover the same exploration rate as the finetuned policy.

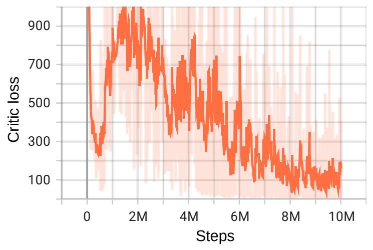
The goal of the next experiment was to test different reward functions aiming to improve exploration efficiency. For that, the configuration of learning exploration from scratch was employed, given that it resulted in the best performance for the Simple Room. The only changes are referent to the reward function, which is reshaped to further encourage the agent to move to positions that increase the exploration rate. A penalty of -5 is defined for moving to already visited positions. When the agent visits a new position, but the exploration rate does not change (exploration delta is zero), three rewards are tested: 0.0, -1.0, and -3.0. The goal is to reduce the agent average path length while maintaining the exploration rate.

Figure 7.7 compares the training metrics of the three proposed reward functions. The reward function that produced the worst results was the one that defined a reward of

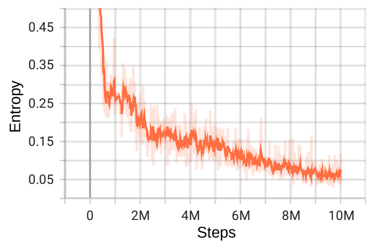
Figure 7.6 – Training metrics of single agent in Simple Room learning exploration from scratch.



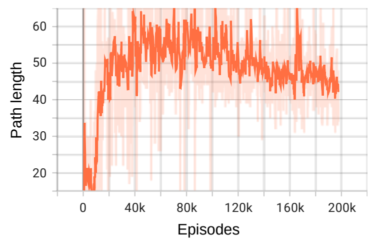
(a) Policy Loss



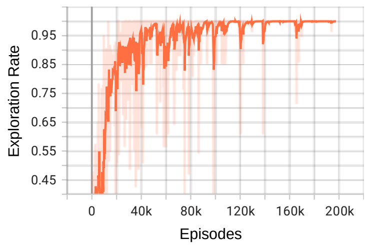
(b) Critic Loss



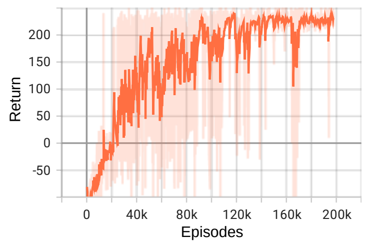
(c) Entropy



(d) Episodic Path Length



(e) Episodic Exploration Rate



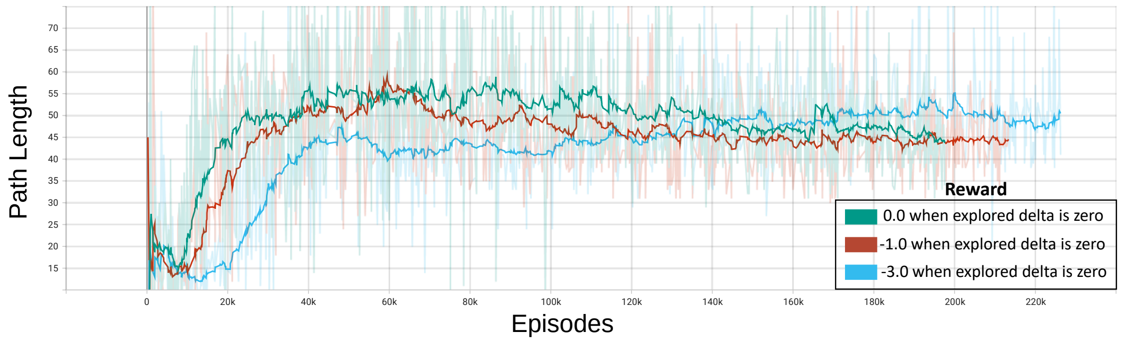
(f) Episodic Return

-3 for exploration delta zero. Because this penalty is more aggressive, the agent takes longer to understand that it can accumulate more reward visiting regions that result in no newly explored cells than colliding, as demonstrated in the path length graph. However, after the agent learns to avoid collisions, the episodic path length keeps increasing, ending up with higher values than the other configurations. One hypothesis is that this reward function would require more training steps to optimize the path length. The exploration rate converges to around 97%, but presents a higher variation compared to the other experiments. Furthermore, the policy and critic loss also culminate in worse values. The final entropy is higher than the other methods, demonstrating that the resulting policy decisions have a more elevated randomness level.

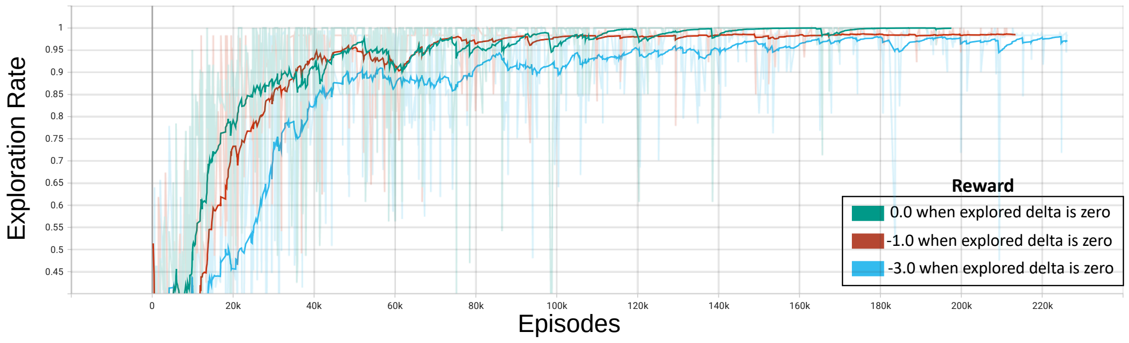
On the other hand, the reward functions that used 0.0 and -1.0 for exploration delta zero presented very similar results. Along the training process, the path length increases before converging to comparable values, close to 45 steps. Both exploration rates converge to values close to 98%. The policy loss, critic loss, and entropy also presented similar behaviors. However, the metrics of the reward function that used a penalty of -1 were more stable, and the path length took fewer training steps to converge. Therefore, this reward shaping is adopted in all subsequent experiments.

The last single robot experiment in the Simple Room uses the best training configuration identified so far to test different trajectory logics. As described in Section 5.2.1, three methods to assemble the agent trajectory with a fixed number of positions are proposed: Merge last points, *drop random points* and *FIFO*. For each training, the reward

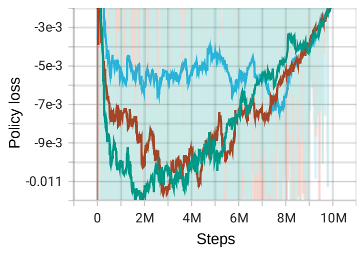
Figure 7.7 – Training metrics of single agent in Simple Room learning exploration from scratch, using different reward functions.



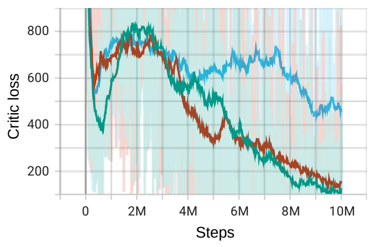
(a) Episodic path length



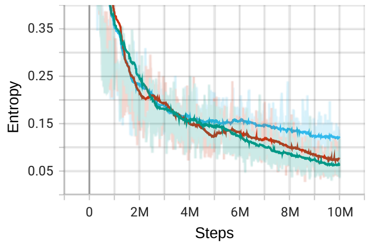
(b) Episodic exploration rate



(c) Policy Loss



(d) Critic Loss

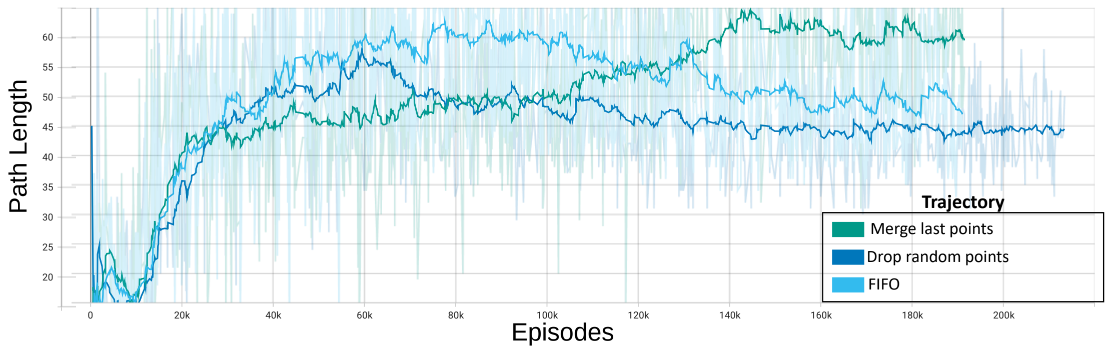


(e) Entropy

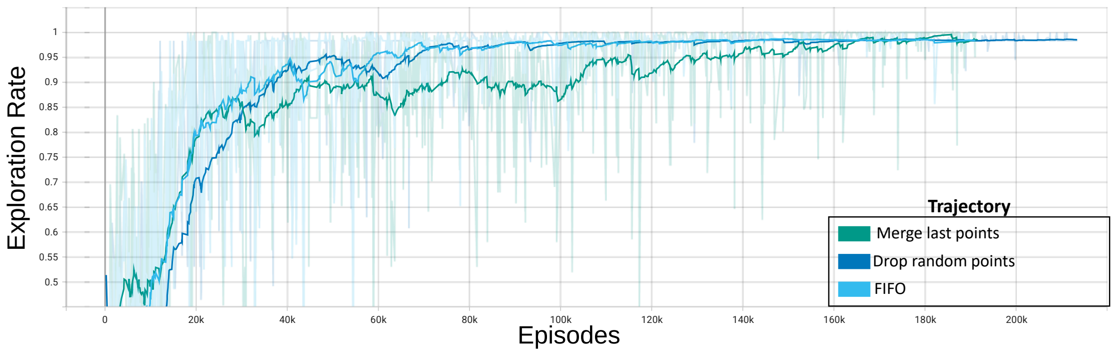
was -100 for collision and $+100$ for exploration rate 98%. If the agent explores unseen cells, the number of new cells is the reward. If it does not find new cells, but its current position was not visited before, the penalty is -1. Finally, a penalty of -5 is defined if the agent previously visited its current position. The agents learn how to explore from scratch. Figure 7.8 illustrates the resulting training metrics.

Critic loss, policy loss, and entropy converge to similar values for the three configurations. However, the trajectory method that resulted in the worst exploration efficiency was the *merge last positions*. Besides the exploration rate presenting an increased variance, the path length converged to an average of 10 steps higher than the other methods. It is possible to conclude that the model has more difficulty identifying patterns in this trajectory logic to map the states into proper actions. On the other hand, the trajectory logic with the best performance was the *drop random points*. The path length presented the lower final values, and the exploration rate converged stably to around 99%. Therefore, the *drop random points* trajectory is employed in all following experiments.

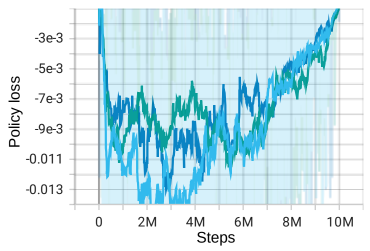
Figure 7.8 – Training metrics of single agent in Simple Room learning exploration from scratch, using different trajectories.



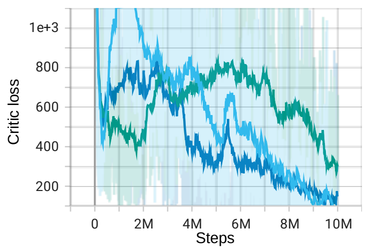
(a) Episodic path length



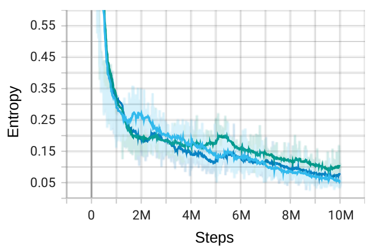
(b) Episodic exploration rate



(c) Policy Loss



(d) Critic Loss



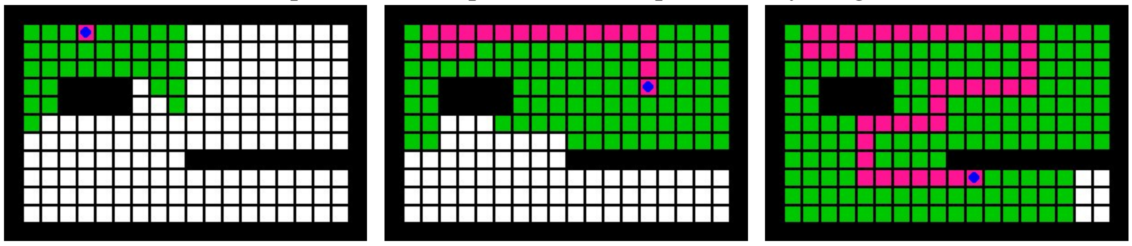
(e) Entropy

Finally, inference tests were performed using the resulting finetuned exploration policy and the best exploration policy learned without curriculum learning. The models were tested 50 times in the Simple Room map, and in each test the agent was initialized in a random free position. Table 7.3 demonstrate the path length and exploration rate's resulting mean and standard deviation. The model trained from scratch resulted in a more efficient exploration, covering the same area (98%) while navigating a shorter average path. Figure 7.9 illustrates the path and the explored area using the most efficient policy and initializing the agent in a random position. Although presenting different levels of efficiency, the Simple Room experiments demonstrated that the proposed mapless framework was able to learn how to explore with or without curriculum learning using a single robot.

Table 7.3 – Inference results of exploration policies for a single robot in the Simple Room environment.

Map	Train	Path Length (m)		Exploration Rate	
		Mean	Std	Mean	Std
Simple Room	Finetunne	9.54	2.07	0.98	0.01
	From scratch	8.12	1.19	0.98	0.05

Figure 7.9 – Three stages of a single agent exploring the Simple Room map. The taken path is illustrated in pink and the explored area is represented by the green cells.



7.1.3 Two robots

7.1.3.1 Simple room

The main goal of the two robots' validation experiments is to verify if the proposed framework can enable collaborative exploration between the agents. In the first experiment, both agents were initialized with the exploration policy that, in Section 7.1.2, resulted in the best performance for a single agent in the same map. Only one of the agents (referred to as *robot 1*) receives the trajectory of the other agent (referred to as *robot 2*), updating its actor and critic weights. The goal is that *robot 1* learns to cooperate and optimize exploration by finetuning the already trained single robot policy. In the second training logic, *robot 2* was initialized with the single agent policy, while the weights of *robot 1* neural networks were randomly initialized. The idea is to test if *robot 1* is able to learn how to cooperate from scratch.

Another critical aspect for enabling collaboration between agents is reward shaping. The basic reward structure used in the single agent experiments was adapted to encourage cooperation. To test different approaches, two reward functions are proposed. Both attribute a high penalty for colliding with obstacles and with each other. Besides optimizing collaborative exploration, *robot 1* must learn how to avoid colliding with *robot 2*, which represents a dynamic obstacle. Also, a high reward is defined when a specific exploration rate is achieved. It is important to notice that the exploration rate is now calculated considering the area explored by the two agents.

The difference between the reward functions is the definition of when to reward agent 1 if the exploration rate increases. Reward function 1 is represented in Listing 7.1. It uses the individual performance of the agent being trained as a condition to determine the reward. Therefore, *robot 1* will only receive a positive reward if it explores new cells. When that happens, the reward is the sum of the number of new cells discovered by both robots at step t. The goal of including the cells discovered by *robot 2* is to reward cooperation, but only when *robot 1* contributes to the exploration. If it does not discover new cells but is in a new position, it receives a penalty of -1. If it is in an already visited position, the penalty is -5. Reward function 2 rewards *robot 1* considering the performance of both robots at each step. Its logic is demonstrated in Listing 7.2. If any of the two robots improves the exploration rate, the reward becomes the sum of all newly explored cells.

Listing 7.1 – Reward function 1.

```
if collision:
    R(t) = -100
else if (explored_rate >= 98%):
    R(t) = +100
else if (robot_1_new_cells)>0:
    R(t) = robot_1_new_cells + robot_2_new_cells
else if (robot_1_new_cells) == 0 and new_position:
    R(t) = -1
else if (robot_1_new_cells) == 0 and old_position:
    R(t) = -5
```

Four experiments were performed considering the proposed training configurations (Finetunne and From scratch) and reward functions (Reward function 1 and Reward function 2). The *Drop random points* trajectory logic was employed for both robots, and each trajectory was composed of 50 points. Hence, the size of the state vector of *robot 2* remains 108, and of *robot 1* changes to 208 to include both trajectories. The robots' exploration radius was $80cm$. At each episode, the robots were initialized at different random free positions. The reference for localization was *robot 2*, meaning that it starting position is locally represented as $(0, 0)$, and *robot 1* position is defined considering its distance from robot 2. The resulting training metrics are depicted in Figure 7.10.

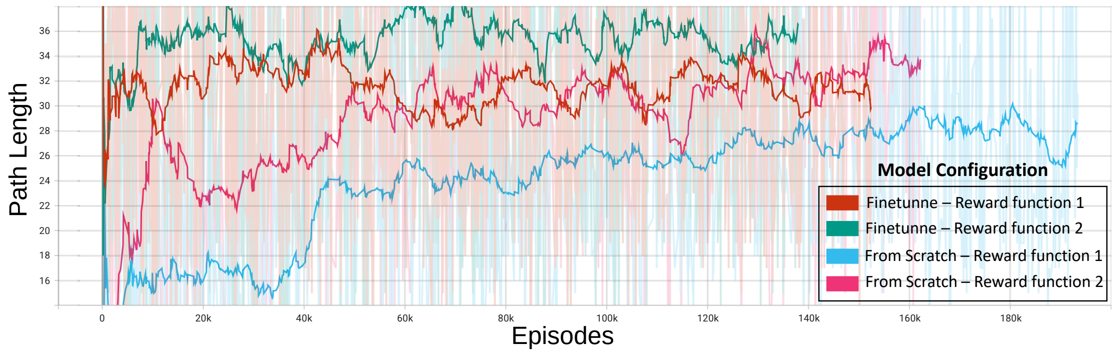
Listing 7.2 – Reward function 2.

```
if collision:
    R(t) = -100
else if (explored_rate >= 98%):
    R(t) = +100
else if (robot_1_new_cells)>0 or (robot_2_new_cells)>0:
    R(t) = robot_1_new_cells + robot_2_new_cells
else if (robot_1_new_cells) == 0 and (robot_2_new_cells) == 0
and new_position:
    R(t) = -1
else if (robot_1_new_cells) == 0 and (robot_2_new_cells) == 0
and old_position:
    R(t) = -5
```

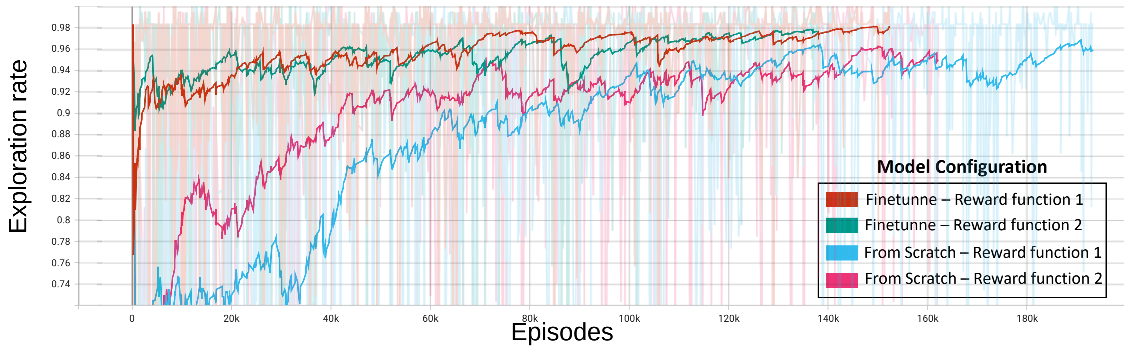
Results show that the experiments where *robot 1* has to learn to explore and cooperate from scratch converged to lower exploration rates. Both configurations *From Scratch* also ended the training with the higher entropy and critic loss values. One hypothesis that justifies the results is that because *robot 2* already knows how to explore the environment individually, *robot 1* does not have enough time to learn to avoid collisions and optimize exploration before the episode ends. However, the experiment *From Scratch* using Reward function 1 culminated in exploration rates close to 96% with path lengths around 24 steps ($4.8m$), which is almost half of the average path length achieved in single agent experiments. Therefore, these metrics demonstrate that cooperation was achieved.

The Finetunne experiments presented a better performance, given that exploration rate converged to 98%. The path length results present considerable variance through episodes. This behavior can be justified by the fact that robot 1's optimum path varies depending on its starting position, on robot 2's starting position and chosen path. Comparing the two reward functions in the finetunne configuration, it is possible to conclude that Reward function 1 resulted in more efficient cooperation, given that path length culminated in lower values for similar exploration rates. Its final entropy was also smaller. Results demonstrate that a reward function that is more strict with individual performance results in a more efficient exploration. Finally, the validation experiments demonstrated that the proposed framework can promote cooperation between two agents in a simple room environment.

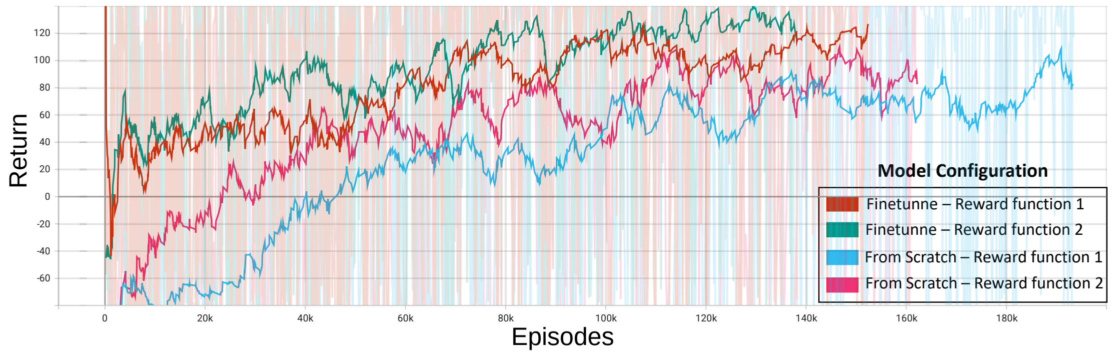
Figure 7.10 – Training metrics of one agent in Simple room learning cooperative exploration between two robots.



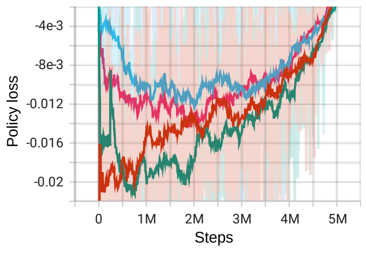
(a) Episodic path length



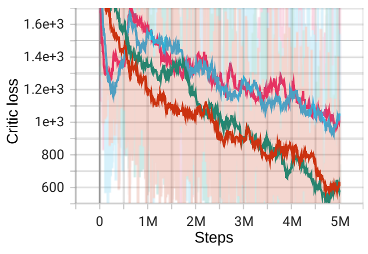
(b) Episodic exploration rate



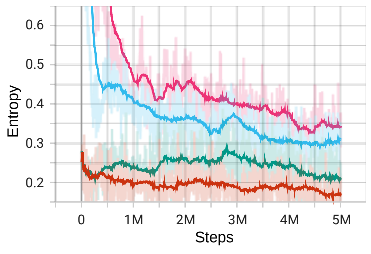
(c) Episodic Return



(d) Policy Loss



(e) Critic Loss



(f) Entropy

7.2 Comparison with baselines

In this section, the proposed exploration framework is evaluated in more complex environments. Section 7.2.1 presents the results of training a single robot using different laser ranges with and without curriculum learning. The best performance is compared with different baselines, including the Random Walk method, the DQN and the AFCNQ, proposed by [Li, Zhang e Zhao 2019]. Section 7.2.2 presents the exploration efficiency, the generalization and the collaboration capability for two robots tested in different configurations.

7.2.1 Single robot

Before training new policies, the exploration policy that presented the best performance in the Simple Room experiments was tested in the Reference map. Considering that the Reference Map is larger than the Simple Room, different exploration ranges were employed: $80cm$, which is the same radius used in the Simple Room experiments, $2m$, and $3m$. 50 tests were performed for each configuration, and the path length and exploration rate were recorded. As demonstrated in Table 7.4, regardless of the exploration radius, the agent is not able to explore the environment successfully. In all tests, the agent collides after covering an area similar to the Simple Room. This behavior is more clearly observable in the experiment using the $80cm$ radius: the Simple Room represents 26% of the area of the reference room, and the average exploration rate in the Reference Map was 28%. On the other hand, the policies trained in the Reference Map, which will be detailed in this section, can explore the Simple Room. The results yield the conclusion that the proposed method can only explore environments with a maximum size similar to the map used for training the policy.

Table 7.4 – Results of testing the Simple Room exploration policy in the Reference Map using different exploration radius.

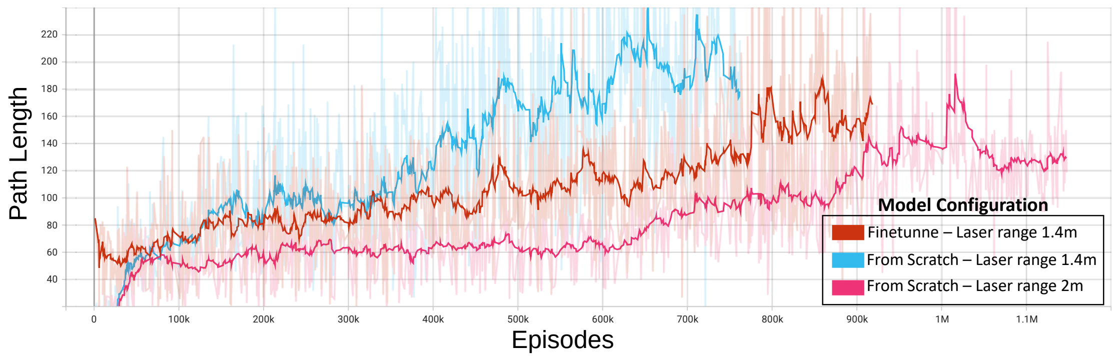
Maps		**Exploration Radius (m)**		**Path Length (m)**		**Exploration Rate**	
Train	**Test**	**Train**	**Test**	**Mean**	**Std**	**Mean**	**Std**
Simple Room	Reference Map	0.8	0.8	13.96	8.08	0.28	0.09
		0.8	2.0	16.92	6.75	0.53	0.17
		0.8	3.0	14.0	6.23	0.51	0.18

The configurations that resulted in the best performance in the Simple Room validation phase were used to train the single agent in the Reference Map. Thus, the utilized trajectory method was the *drop random points*, and 50 points were considered. The reward was 100 for collision and $+100$ for exploration rate 93%. Compared to previous experiments, the exploration rate was reduced to 93% because the Reference Map presents bigger dimensions and higher complexity. Using a value of 98%, for example, could result in too many episodes before achieving the necessary exploration rate or even in complete trains where the agent never gets the maximum reward. Continuing with the reward function, if the agent explores unseen cells, the number of new cells is the reward. If it does not find new cells, but its current position was not visited before, the penalty is 1. A penalty of 5 is defined if the agent previously visited its current localization. Like previous experiments, an episode ends if a collision happens or the exploration rate is higher than 93%, and the agent is initialized in a random free position. Training ends when the agent performs $100M$ steps.

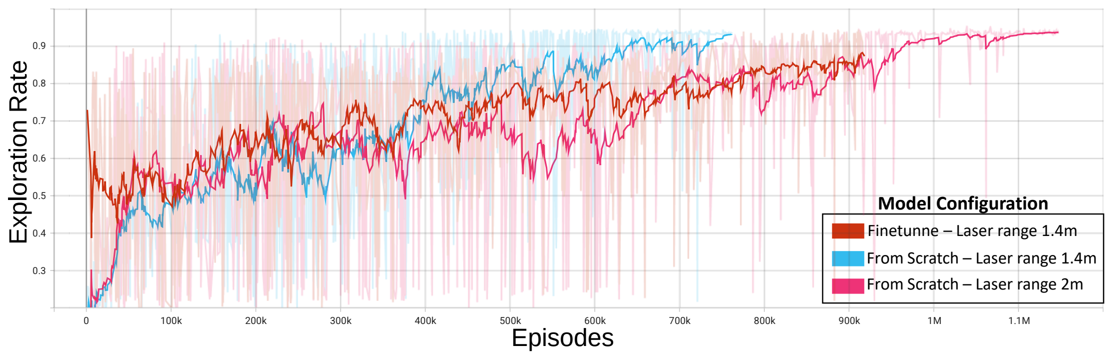
The research work from which the baseline maps are extracted does not specify what area around the robot is considered explored [Li, Zhang e Zhao 2019]. Therefore, different exploration radii are tested. Three models are trained in the Reference Map with a single agent. The first initializes the actor and critic with the neural networks resulting from the Simple Room experiment so that the agent learns how to explore a more complex map by finetuning. The second initializes the weights with random values, so the agent learns to explore the Reference Map from scratch. Both experiments determine an exploration radius of $1.4m$. The third experiment also aims at learning to explore from scratch, but uses an exploration radius of $2m$. As a reference, the laser range of a Turtlebot3 robot is $3.5m$, so it is possible to affirm that the robot could cover a $1.4m$ or a $2m$ exploration radius with sufficient precision.

Figure 7.11 compares the metrics of the three proposed training structures. The figure legend refers to the exploration radius as *Laser range*. As explained in Section 7.1.2.1, an exponential moving average is used to smooth the curves and facilitate visualization. However, observing the absolute values of each episode, it is possible to notice a higher variance in all metrics compared to the experiments with smaller maps. This behavior can be justified by the fact that with higher map dimensions, there are more possible interactions with the environment that the agent can explore. In the last episodes, path length variance can still be significant even in successful training, given that there are different possibilities of routes that result in an effective exploration. Path length can also vary depending on the agent's starting position.

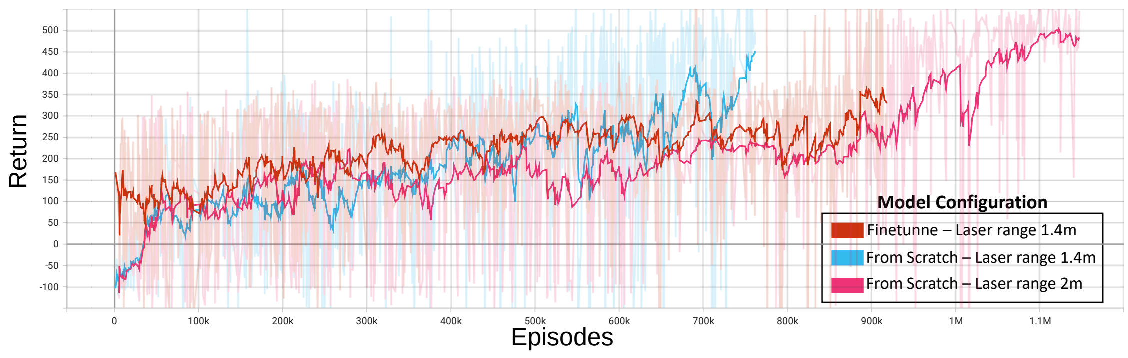
Figure 7.11 – Metrics of single agent in Reference Map using different training configurations.



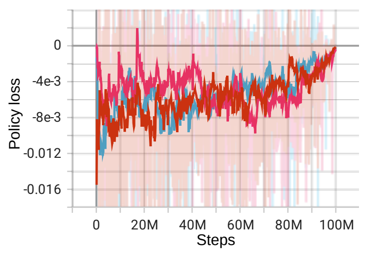
(a) Episodic path length



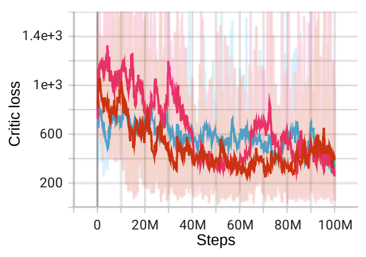
(b) Episodic exploration rate



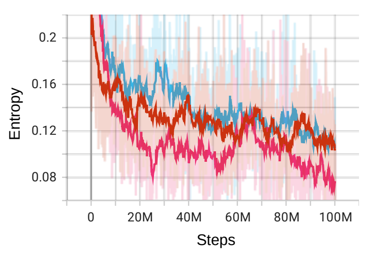
(c) Episodic Return



(d) Policy Loss



(e) Critic Loss



(f) Entropy

Analyzing the two experiments that employed a $1.4m$ range, it is possible to conclude that learning exploration from scratch presented better results. The last episodes obtained higher exploration rates for similar path lengths, resulting in higher episodic returns. However, the experiment that used $2m$ exploration radius surpassed the performance of both other experiments. It would be expected that a higher coverage radius would result in a more efficient exploration, and the metrics confirm that prediction. In the last episodes, the exploration rate stably converged to 93%, and the path length to approximately 120 steps ($24m$). Naturally, it also culminated in the highest episodic returns. The final entropy values were the lowest of the three experiments, demonstrating that the resulting policy presents a lower randomness level in its decision-making process.

To test the performance of the three resulting policies, inference was performed in the Reference Map, in Test Map 1, and in Test Map 2, presented in Section 6.3. Each model was tested 50 times on every map, and the path length and exploration rate were recorded. Table 7.5 shows the results of the tests. All policies' best performance resulted from the Reference Map tests. The metrics mean and standard deviation confirm the performance expected from the observation of the graphs in Figure 7.11. The model that resulted in the most efficient exploration used $2m$ as the exploration radius. Naturally, the models with a lower coverage range take longer paths to achieve similar exploration rates.

Table 7.5 – Results of the three models trained in the Reference Map and tested in the Reference Map, test map 1 and test map 2.

Maps		**Model Configuration**		**Exploration Rate**		**Path Length (m)**	
Train	**Test**	**Train**	**Exploration Radius (m)**	**Mean**	**Std**	**Mean**	**Std**
Reference Map	Reference Map	Finetunne from Simple Room	1.4	0.92	0.04	36.82	4.22
		From scratch	1.4	0.92	0.06	34.06	6.38
		From scratch	2	0.94	0.004	24.88	3.13
Reference Map	Test Map 1	Finetunne from Simple Room	1.4	0.73	0.16	38.21	3.65
		From scratch	1.4	0.75	0.09	39.57	1.96
		From scratch	2	0.82	0.17	30.4	7.38
Reference Map	Test Map 2	Finetunne from Simple Room	1.4	0.66	0.19	39.28	3.05
		From scratch	1.4	0.74	0.13	37.32	5.44
		From scratch	2	0.81	0.14	28.88	7.32

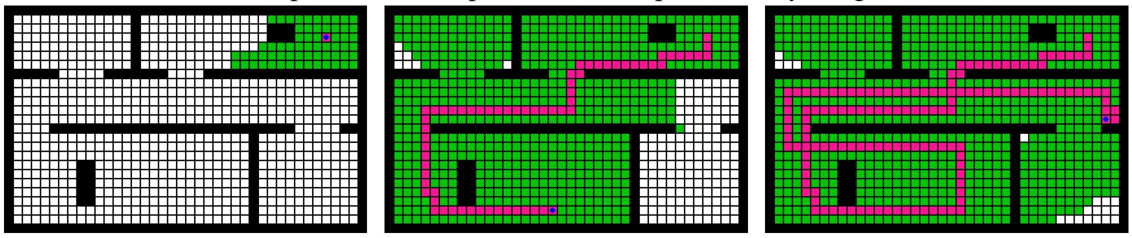
The goal of testing the policies in Test Maps 1 and 2 was to evaluate the generalization capability of the exploration strategies. In other words, analyze the policies' performance when a previously unseen map is presented. It is possible to notice that, for all methods, the exploration efficiency decreases compared to the tests in the Reference Map. The agent takes longer paths to cover a smaller area. These results indicate that some level of overfitting happens during training. The agent fits too closely to the training map, and is not able to maintain the same efficiency in new environments. However, to more objectively analyze the performance in the different maps, the metrics of the model with exploration radius $2m$ are compared to the baselines presented in Section 6.3: Random Walk, AFCQN, and DQN.

Tables 7.6 contains the comparison of test performed in the Reference Map using our method and the selected baselines. Our method performance surpassed the classic DQN in terms of exploration efficiency, covering in average 10% more of the environment with similar path lengths. The obtained results are comparable with the AFCNQ. Although our average path length was higher, its standard deviation was lower, and the average exploration rate was 94% in comparison with 91%. Figure 7.12 illustrates the path and the explored area running our method in the Reference Map and initializing the agent in a random position.

Table 7.6 – Comparing different exploration methods for single robot in the Reference Map.

Method	**Exploration Rate**		**Path Length (m)**	
	Mean	Std	Mean	Std
Random Walk	0.36	0.12	60.0	0.0
AFCNQ	0.91	0.0028	24.39	6.61
DQN	0.84	0.21	23.19	5.9
Our method	**0.94**	**0.004**	**24.88**	**3.13**

Figure 7.12 – Three stages of a single agent exploring the Reference Map. The taken path is illustrated in pink and the explored area is represented by the green cells.

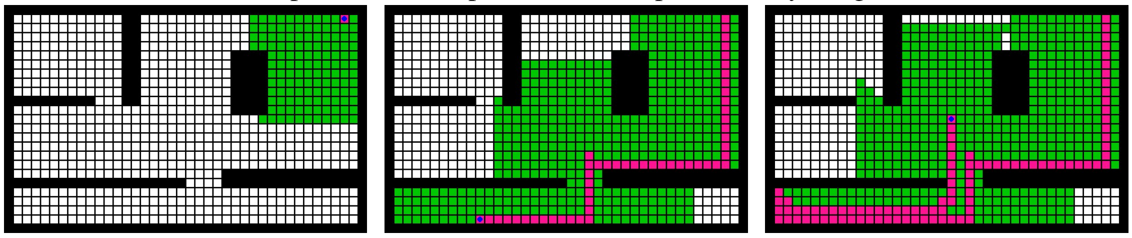


For the experiments with Test Maps 1 and 2, the reference research work did not present the metrics' absolute values, but depicted them in a graphic format [Li, Zhang e Zhao 2019]. Therefore, the values included in Tables 7.7 and 7.8 are approximations based on the graphs observation. In Test Map 1, once more our method presented an average exploration rate close to 10% higher than the DQN baseline. Although the average path length was longer, the standard deviation of the DQN is elevated ($16.56m$). So, it is possible to affirm that our method presented better generalization capabilities in Test Map 1 and is generally more efficient than DQN. In Test Map 2, DQN results were objectively better than ours. Figures 7.13 and 7.14 illustrate the path and the explored area running our method in the Test Map 1 and in the Test Map 2, respectively. Although there is some redundancy in the taken paths, it is possible to observe that the exploration coverage presented a consistent behavior in both maps. Therefore, although there was a performance degradation from tests in Reference Map, the exploration is still comparable with baseline methods, and it is consistent for maps of the same size with different obstacles.

Table 7.7 – Comparing different exploration methods for single robot in the Test Map 1.

Method	**Exploration Rate**		**Path Length (m)**	
	Mean	Std	Mean	Std
Random Walk	0.63	0.19	60.0	0.0
AFCNQ	0.96	0.011	20.01	8.3
DQN	0.73	0.26	23.12	16.56
Our method	**0.82**	**0.17**	**30.4**	**7.38**

Figure 7.13 – Three stages of a single agent exploring the Test Map 1. The taken path is illustrated in pink and the explored area is represented by the green cells.

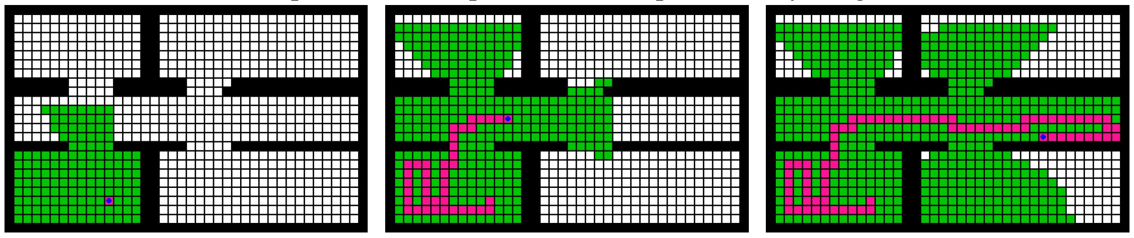


The AFCNQ method presented the most efficient exploration for the test maps. The original publication does not include the number of layers and neurons used for exploration. Thus, it is impossible to objectively affirm that AFCNQ would present higher computational costs than our method. However, it does use information about the environment with higher degrees of data processing, including the environment occupancy grid map, a frontier map, and a Q-value map. In the image that depicts the employed CNN, at least 14 layers are used. The resolution of the input image is 161x201, and the action space is composed of all grid map sampling points. Therefore, the performance achieved by our method can be considered a competitive result, since the single agent tests used an input with size 108, two hidden layers with 64 neurons each, an action space of 4 possible actions, and only basic information about the robot localization. Furthermore, AFCNQ defines a localization the robot should go to, and a complementary path planning method must be used to safely guide the robot to the position. Our exploration strategy is end-to-end, directly mapping the states into movements and providing collision avoidance together with the exploration plan. Finally, both methods present different advantages that should be evaluated according to each application constrains and goals.

Table 7.8 – Comparing different exploration methods for single robot in the Test Map 2.

Method	**Exploration Rate**		**Path Length (m)**	
	Mean	Std	Mean	Std
Random Walk	0.52	0.19	60.0	0.0
AFCNQ	0.94	0.5	21.0	5.0
DQN	0.9	0.5	25.4	8.5
Our method	**0.81**	**0.14**	**28.88**	**7.32**

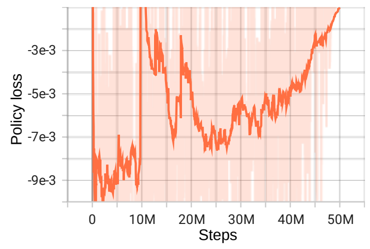
Figure 7.14 – Three stages of a single agent exploring the Test Map 2. The taken path is illustrated in pink and the explored area is represented by the green cells.



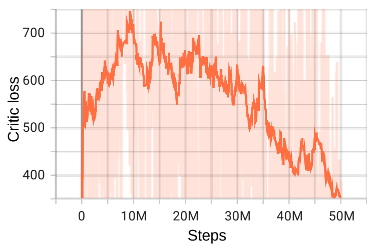
7.2.2 Two robots

As previously stated, the training configuration that resulted in the best performance for two robots in the validation experiments was reproduced in the baseline maps. That means both robots are initialized with the single agent exploration strategy for the Reference Map, from Section 7.2.1. Because this environment presents a higher complexity, the total number of steps for training is increased to $50M$, and the maximum exploration rate is decreased to 93%. The reward function and all training configurations are described in detail in Section 7.1.3. Figure 7.15 presents the resulting metrics.

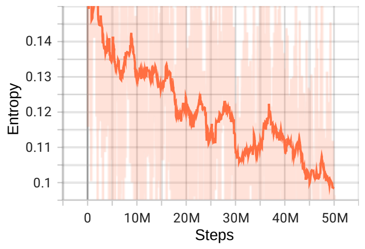
Figure 7.15 – Training metrics of one agent in Reference Map learning cooperative exploration between two robots.



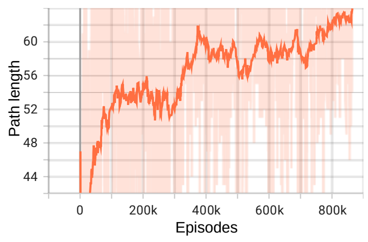
(a) Policy Loss



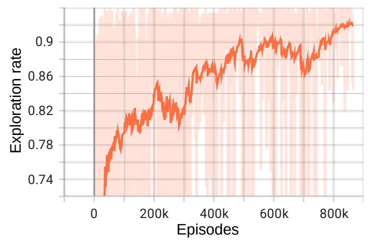
(b) Critic Loss



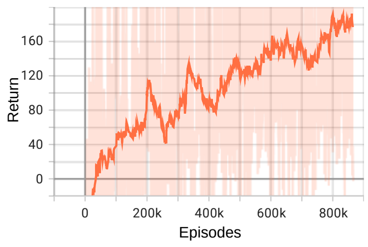
(c) Entropy



(d) Episodic Path Length



(e) Episodic Exploration Rate



(f) Episodic Return

The resulting metrics indicate that the agent learns how to cooperate to explore the environment. Each robot's path length converges to an average value close to 65 steps, or $13m$, which is mostly half of the length obtained by a single agent in the Reference Map. At the same time, the exploration rate culminated in 93% in the last episodes, increasing the episodic return. However, to objectively verify the resulting policy performance, tests in the Reference Map, Test Map 1, and Test Map 2 are performed. In each map, four experiments are compared. The first one is the single agent performance in the target map. In the second experiment, both agents run the single agent policy so that they explore the environment individually, without sharing information. In the third experiment, *robot 1* runs the policy that resulted from the two robots train, while *robot 2* runs the single agent policy. Only *robot 1* receives the other agent trajectory. Finally, both agents run the policy trained with two agents, sharing trajectory information. The robots use an exploration radius of $2m$, and each experiment is repeated 50 times.

Table 7.9 contains the results of the tests in the Reference Map. The experiments where each robot explored the environment independently resulted in an average exploration rate of 87% covering a path length of $16.6m$. The rate is lower than the single robot results because the agents do not know how to avoid colliding with each other, resulting in shorter tests. On the other hand, running the two-agent policy in *robot 1* and the single-agent policy in *robot 2* resulted in the most efficient exploration. This result could be expected because this is the same configuration used for training the agents. Although the exploration rate decreased to 89% compared to the single agent 94%, the path length traveled by each agent was more than 2 times smaller.

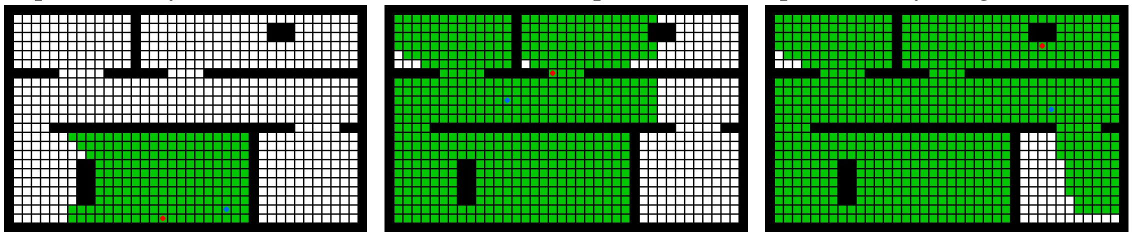
Table 7.9 – Tests of one and two robots in the Reference Map

Map	**Number of Robots**	**Policy Robot 1**	**Policy Robot 2**	**Exploration Rate**		**Path Length (m)**	
				Mean	**Std**	**Mean**	**Std**
Reference Map	1	Single agent trained in Reference Map	-	0.94	0.004	24.88	3.13
	2	Single agent trained in Reference Map	Single agent trained in Reference Map	0.87	0.08	16.6	10.07
	2	Two agents trained in Reference Map	Single agent trained in Reference Map	0.89	0.09	10.95	2.33
	2	Two agents trained in Reference Map	Two agents trained in Reference Map	0.81	0.11	11.65	2.64

Running the two-agent policy on both robots resulted in a similar average path length but a lower exploration rate. A possible justification is that for one of the robots, the trajectory order in the state vector is the inverse of what was presented in training, which can mislead the policy. An alternative to improve the shared network would be employing other multi-agent training techniques instead of training the networks separately. Figure 7.16 illustrates the explored area initializing the two robots in random positions. It is possible to notice that in this example the robots start in similar positions, but are able to split paths to cover opposite rooms.

The results of the experiments performed in Test Map 1 are shown in Table 7.10. For all tests, the exploration rate presented similar values. However, running the two-

Figure 7.16 – Three stages of two agents exploring the Reference Map. The agents are represented by the red and blue circles, and the explored area is represented by the green cells.



robot policy in one agent and the single agent policy in the other resulted in the most efficient exploration. The robots covered the same area as the single robot (82%) using only one-third of its path length. It is possible to interpret that in this configuration, *robot 1* successfully learns how to avoid the positions that *robot 2* already visited. Although losing some efficiency, the results demonstrate that the two-robot policy could generalize to unseen maps. The two robots achieved the highest exploration rate with individual exploration policies. However, the path length was almost 2x higher than the cooperative methods. Because each robot does not know which areas have already been explored by the other agent, it travels redundant paths. Figure 7.17 illustrates the explored area initializing the two robots in random positions.

Table 7.10 – Tests of one and two robots in Test Map 1

Map	**Number of Robots**	**Policy Robot 1**	**Policy Robot 2**	**Exploration Rate**		**Path Length (m)**	
				Mean	**Std**	**Mean**	**Std**
Test Map 1	1	Single agent trained in Reference Map	-	0.82	0.17	30.4	7.38
	2	Single agent trained in Reference Map	Single agent trained in Reference Map	0.85	0.1	21.83	15.62
	2	Two agents trained in Reference Map	Single agent trained in Reference Map	0.82	0.15	10.14	2.61
	2	Two agents trained in Reference Map	Two agents trained in Reference Map	0.815	0.09	12.03	3.74

Figure 7.17 – Three stages of two agents exploring the Test Map 1. The agents are represented by the red and blue circles, and the explored area is represented by the green cells.

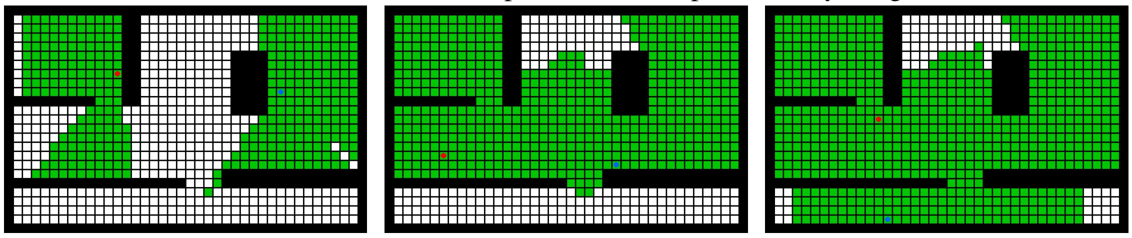


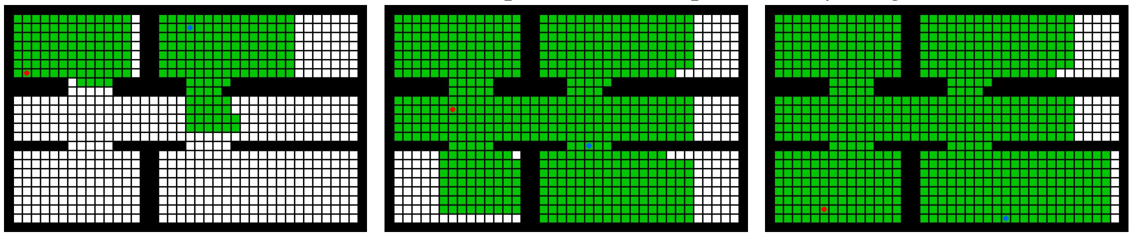
Table 7.11 contains the results of the experiments performed in Test Map 2. The metrics reinforce the conclusions of the other maps' tests. Using the two-robot policy in one agent and the single-robot policy in the other was, once more, the most efficient method. It resulted in 79% of exploration rate compared to 81% of the single agent. How-

ever, the path length covered by each robot was almost four times smaller. Figure 7.18 illustrates the explored area initializing the two robots in random positions. To conclude, the experiments demonstrated that cooperation between two agents was effectively established without the need of complex map merging algorithms. However, this conclusion is true in the controlled environments and for the tested maps' dimensions and complexity levels. More experiments are needed to verify the method's efficiency in larger maps, with uncertainties in sensor readings and in the communication between robots.

Table 7.11 – Tests of one and two robots in Test Map 2

Map	**Number of Robots**	**Policy Robot 1**	**Policy Robot 2**	**Exploration Rate**		**Path Length (m)**	
				Mean	**Std**	**Mean**	**Std**
Test Map 2	1	Single agent trained in Reference Map	-	0.81	0.14	28.88	7.32
	2	Single agent trained in Reference Map	Single agent trained in Reference Map	0.82	0.08	14.62	11.49
	2	Two agents trained in Reference Map	Single agent trained in Reference Map	0.79	0.11	7.45	2.66
	2	Two agents trained in Reference Map	Two agents trained in Reference Map	0.71	0.14	7.84	3.42

Figure 7.18 – Three stages of two agents exploring the Test Map 2. The agents are represented by the red and blue circles, and the explored area is represented by the green cells.



8 CONCLUSION

This dissertation aimed to investigate whether Deep Reinforcement Learning can enable efficient and robust mapless exploration strategies for single and multiple mobile robots. An extensive literature review was performed and revealed that the employment of DRL for mobile robotic exploration has significantly increased in the last few years. This can be justified by the fact that several modern applications require robotic autonomy when a map of the environment is unknown. The employment of mobile robots in such complicated contexts depends on a robust and efficient exploration strategy. At the same time, employing deep neural networks as powerful function approximators for RL algorithms has enabled optimal or near-optimal solutions for several complex problems with high-dimensional inputs. Therefore, multiple works have successfully applied DRL algorithms to compose exploration strategies for single and multiple agents. The tackled applications include finding or reaching a specified target and covering a whole area to map the environment.

However, no works that investigate DRL for mapless exploration aiming at efficient area coverage were identified. Mapless exploration refers to strategies that do not build an explicit map of the environment to decide where to go next. It is known that some limitations can arise from basing the exploration decisions on an online built map. The computational costs tend to increase with the expansion of the explored area, and the efficiency of the exploration strategy heavily depends on accurate maps. For multi-robot cooperative exploration, the maps generated by each agent are often shared between the team and must be correctly merged, which is a challenging task. Concurrently, there are applications where the agents must cover an entire area but the final goal is not mapping the environment, as search and rescue or multi-target search. In situations with limited memory and processing resources, such applications can benefit from an exploration strategy that does not rely on a precise world map.

Therefore, based on the information gathered from the performed academic literature review, we proposed an end-to-end mapless exploration framework based on DRL and suitable for single robots and teams of n robots. The agent keeps track of a fixed-size quantity of sensor information that is not heavily processed, such as laser measurements and the robot's trajectory. The team's goal is to efficiently explore a 3D environment in a 2D fashion as fast as possible. The idea is to obtain agents capable of exploring the environment alone, but that can optimize their decision-making process when information from other agents is received.

The proposed exploration architecture was trained and tested using one and two robots in different simulation environments. The achieved exploration efficiency surpassed the random walk baseline and was comparable to the classic DQN method, which relied on much more complex representations of the environment. The performance of our approach was also more consistent than the DQN, presenting lower path length and exploration rate variance. The method that raised the most efficient exploration for the evaluated unseen maps was the AFCNQ, proposed in [Li, Zhang e Zhao 2019]. However, AFCNQ uses information about the environment with higher degrees of data processing, including the environment occupancy grid map, a frontier map, and a Q-value map. Furthermore, unlike AFCNQ, our exploration strategy is end-to-end, directly mapping the states into movements and providing collision avoidance with the exploration plan. Finally, both methods present different advantages that should be evaluated according to each application's constraints and goals.

The experiments using two robots demonstrated that the proposed architecture can promote cooperation between agents without needing map merging algorithms. It increased the explored region rate with smaller path lengths per individual robot. The strategy also generalized between different environments, presenting small performance drops for unseen maps. The experiments showed that proper reward shaping is essential to learn the desired behavior successfully. The method to record the robot trajectory and the decision of when to use curriculum learning also significantly impacted the exploration policy's success. Finally, it is crucial to notice that all results were obtained in controlled environments and the presented conclusions are valid for the tested maps' dimensions and complexity levels.

In future works, we aim to evaluate the exploration framework performance for more than two robots. Different reward shaping can be tested to increase coverage efficiency. The comparison with other MRS collaborative exploration strategies is also an essential future analysis. Furthermore, the exploration strategy must be tested in a non-ideal environment, with uncertainties in the sensor measurements. The evaluation of the collaborative method will be tested considering unstable communication, where the robots cannot share information at each new step. These experiments pave the way for the final research goal: testing the exploration strategy in real robots.

REFERENCES

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

Adler, A.; Karaman, S. The stochastic traveling salesman problem and orienteering for kinodynamic vehicles. In: **2016 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.: s.n.], 2016. p. 2788–2795.

Akrour, R.; Veiga, F.; Peters, J.; Neumann, G. Regularizing reinforcement learning with state abstraction. In: **IEEE International Conf. on Intelligent Robots and Systems (IROS)**. [S.l.: s.n.], 2018. p. 534–539.

ALATISE, M. B.; HANCKE, G. P. A review on challenges of autonomous mobile robot and sensor fusion methods. **IEEE Access**, IEEE, v. 8, p. 39830–39846, 2020.

Ando, Y.; Yuta, S. Following a wall by an autonomous mobile robot with a sonar-ring. In: **Proceedings of 1995 IEEE International Conf on Robotics and Automation**. [S.l.: s.n.], 1995. v. 3, p. 2599–2606 vol.3.

Arai, T.; Toda, Y.; Mutsumi, I.; Shao, S.; Tonomura, R.; Kubota, N. Reinforcement learning based on state space model using growing neural gas for a mobile robot. In: **Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems**. [S.l.: s.n.], 2018. p. 1410–1413.

ARKIN, R. C.; ARKIN, R. C. **Behavior-based robotics**[S.l.]: MIT press, 1998.

Balch, T. Avoiding the past: a simple but effective strategy for reactive navigation. In: **[1993] Proceedings IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 1993. p. 678–685 vol.1.

BOUZY, B.; CHASLOT, G. Monte-carlo go reinforcement learning experiments. In: IEEE. **2006 IEEE symposium on computational intelligence and games**. [S.l.], 2006. p. 187–194.

BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. Openai gym. **arXiv preprint arXiv:1606.01540**, 2016.

Brooks, R. A hardware retargetable distributed layered architecture for mobile robot control. In: **Proceedings. 1987 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 1987. v. 4, p. 106–110.

CAI, Y.; YANG, S. X.; XU, X. A combined hierarchical reinforcement learning based approach for multi-robot cooperative target searching in complex unknown environments. In: IEEE. **2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)**. [S.l.], 2013. p. 52–59.

CAO, X.; SUN, C.; YAN, M. Target search control of auv in underwater environment with deep reinforcement learning. **IEEE Access**, IEEE, v. 7, p. 96549–96559, 2019.

CARDONA, G.; BRAVO, C.; QUESADA, W.; RUIZ, D.; OBENG, M.; WU, X.; CALDERON, J. Autonomous navigation for exploration of unknown environments and collision avoidance in mobile robots using reinforcement learning. In: IEEE. **2019 SoutheastCon**. [S.l.], 2019. p. 1–7.

ÇETIN, E.; BARRADO, C.; MUÑOZ, G.; MACIAS, M.; PASTOR, E. Drone navigation and avoidance of obstacles through deep reinforcement learning. In: IEEE. **2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)**. [S.l.], 2019. p. 1–7.

Chalmers, E.; Contreras, E. B.; Robertson, B.; Luczak, A.; Gruber, A. Learning to predict consequences as a method of knowledge transfer in reinforcement learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 29, n. 6, p. 2259–2270, 2018.

CHEN, Z.; SUBAGDJA, B.; TAN, A.-H. End-to-end deep reinforcement learning for multi-agent collaborative exploration. In: IEEE. **2019 IEEE International Conference on Agents (ICA)**. [S.l.], 2019. p. 99–102.

CHUPEAU, M.; BéNICHOU, O.; VOITURIEZ, R. Cover times of random searches. **Nature Physics**, v. 11, 08 2015.

CRAYE, C.; FILLIAT, D.; GOUDOU, J.-F. Rl-iac: An exploration policy for online saliency learning on an autonomous mobile robot. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2016. p. 4877–4884.

CRUZ, J. A.; CARDOSO, H. L.; REIS, L. P.; SOUSA, A. Reinforcement learning in navigation and cooperative mapping. In: IEEE. **2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)**. [S.l.], 2020. p. 200–205.

De Silva, L.; Ekanayake, H. Behavior-based robotics and the reactive paradigm a survey. In: **2008 11th International Conference on Computer and Information Technology**. [S.l.: s.n.], 2008. p. 36–43.

DULAC-ARNOLD, G.; MANKOWITZ, D.; HESTER, T. Challenges of real-world reinforcement learning. **arXiv preprint arXiv:1904.12901**, 2019.

ECOFFET, A.; HUIZINGA, J.; LEHMAN, J.; STANLEY, K. O.; CLUNE, J. First return, then explore. **Nature**, Nature Publishing Group, v. 590, n. 7847, p. 580–586, 2021.

FAN, T.; LONG, P.; LIU, W.; PAN, J.; YANG, R.; MANOCHA, D. Learning resilient behaviors for navigation under uncertainty. In: IEEE. **2020 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.], 2020. p. 5299–5305.

FELDMAN, R. M.; VALDEZ-FLORES, C. Markov processes. In: ____. **Applied Probability and Stochastic Processes**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 181–199. ISBN 978-3-642-05158-6. Disponível em: <https://doi.org/10.1007/978-3-642-05158-6_6>.

FUJII, K. Extended kalman filter. **Refernce Manual**, p. 14–22, 2013.

GARAFFA, L. C.; BASSO, M.; KONZEN, A. A.; FREITAS, E. P. de. Reinforcement learning for mobile robotics exploration: A survey. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, 2021.

GARCIA, E.; JIMENEZ, M. A.; SANTOS, P. G. D.; ARMADA, M. The evolution of robotics research. **IEEE Robotics & Automation Magazine**, IEEE, v. 14, n. 1, p. 90–103, 2007.

GAUTAM, A.; MOHAN, S. A review of research in multi-robot systems. In: IEEE. **2012 IEEE 7th international conference on industrial and information systems (ICIIS)**. [S.l.], 2012. p. 1–5.

GENG, M.; XU, K.; ZHOU, X.; DING, B.; WANG et al. Learning to cooperate via an attention-based communication neural network in decentralized multi-robot exploration. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 21, n. 3, p. 294, 2019.

GUASTELLA, D. C.; MUSCATO, G. Learning-based methods of perception and navigation for ground vehicles in unstructured environments: a review. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 21, n. 1, p. 73, 2021.

GUERRA, A.; GUIDI, F.; DARDARI, D.; DJURIC, P. M. Reinforcement learning for uav autonomous navigation, mapping and target detection. In: **2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)**. [S.l.: s.n.], 2020. p. 1004–1013.

HAARNOJA, T.; ZHOU, A.; ABBEEL, P.; LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2018. p. 1861–1870.

HASSELT, H. V.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2016. v. 30, n. 1.

Learning Exploration Strategies in Model-Based Reinforcement Learning, v. 2. 1069-1076 p.

Hirata, K.; Iizuka, H.; Yamamoto, M. Reinforcement learning method with internal world model training. In: **2020 IEEE/SICE International Symp on System Integration (SII)**. [S.l.: s.n.], 2020. p. 201–204.

HU, H.; SONG, S.; CHEN, C. P. Plume tracing via model-free reinforcement learning method. **IEEE transactions on neural networks and learning systems**, IEEE, v. 30, n. 8, p. 2515–2527, 2019.

HU, J.; NIU, H.; CARRASCO, J.; LENNOX, B.; ARVIN, F. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. **IEEE Transactions on Vehicular Technology**, IEEE, v. 69, n. 12, p. 14413–14423, 2020.

ISSA, R. B.; RAHMAN, M. S.; DAS, M.; BARUA, M.; ALAM, M. G. R. Reinforcement learning based autonomous vehicle for exploration and exploitation of undiscovered track. In: IEEE. **2020 International Conference on Information Networking (ICOIN)**. [S.l.], 2020. p. 276–281.

Jain, D.; Iscen, A.; Caluwaerts, K. Hierarchical reinforcement learning for quadruped locomotion. In: **2019 IEEE/RSJ International Conf on Intelligent Robots and Systems (IROS)**. [S.l.: s.n.], 2019. p. 7551–7557.

Jain, U.; Tiwari, R.; Godfrey, W. W. Comparative study of frontier based exploration methods. In: **2017 Conference on Information and Communication Technology (CICT)**. [S.l.: s.n.], 2017. p. 1–5.

JIANG, L.; HUANG, H.; DING, Z. Path planning for intelligent robots based on deep q-learning with experience replay and heuristic knowledge. **IEEE/CAA Journal of Automatica Sinica**, IEEE, v. 7, n. 4, p. 1179–1189, 2019.

JIN, Y.; ZHANG, Y.; YUAN, J.; ZHANG, X. Efficient multi-agent cooperative navigation in unknown environments with interlaced deep reinforcement learning. In: IEEE. **ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [S.l.], 2019. p. 2897–2901.

JOSEF, S.; DEGANI, A. Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain. **IEEE Robotics and Automation Letters**, IEEE, v. 5, n. 4, p. 6748–6755, 2020.

JULIÁ, M.; GIL, A.; REINOSO, O. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. **Autonomous Robots**, Springer, v. 33, n. 4, p. 427–444, 2012.

JUN, H.; KIM, H.; LEE, B. Goal-driven navigation for non-holonomic multi-robot system by learning collision. In: IEEE. **2019 International Conference on Robotics and Automation (ICRA)**. [S.l.], 2019. p. 1758–1764.

KALECI, B.; SENLER, C. M.; PARLAKTUNA, O.; GÜREL, U. Constructing topological map from metric map using spectral clustering. In: IEEE. **2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)**. [S.l.], 2015. p. 139–145.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KOBER, J.; BAGNELL, J.; PETERS, J. Reinforcement learning in robotics: A survey. **The International Journal of Robotics Research**, v. 32, p. 1238–1274, 09 2013.

KROGH, B.; THORPE, C. Integrated path planning and dynamic steering control for autonomous vehicles. In: IEEE. **IEEE International Conference on Robotics and Automation**. [S.l.], 1986. v. 3, p. 1664–1669.

KUZMIN, Y. P. Bresenham's line generation algorithm with built-in clipping. In: WILEY ONLINE LIBRARY. **Computer graphics forum**. [S.l.], 1995. v. 14, n. 5, p. 275–280.

LAZZERI, N.; MAZZEI, D.; COMINELLI, L.; CISTERNINO, A.; ROSSI, D. D. Designing the mind of a social robot. **Applied Sciences**, v. 8, p. 302, 02 2018.

Le, T. D.; Le, A. T.; Nguyen, D. T. Model-based q-learning for humanoid robots. In: **2017 18th International Conference on Advanced Robotics (ICAR)**. [S.l.: s.n.], 2017. p. 608–613.

LI, H.; ZHANG, Q.; ZHAO, D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. **IEEE transactions on neural networks and learning systems**, IEEE, v. 31, n. 6, p. 2064–2076, 2019.

Li, S.; Yan, J.; Li, L. Automated guided vehicle: the direction of intelligent logistics. In: **2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)**. [S.l.: s.n.], 2018. p. 250–255.

Lim, J.; Ha, S.; Choi, J. Prediction of reward functions for deep reinforcement learning via gaussian process regression. **IEEE/ASME Transactions on Mechatronics**, v. 25, n. 4, p. 1739–1746, 2020.

LIN, J.; YANG, X.; ZHENG, P.; CHENG, H. End-to-end decentralized multi-robot navigation in unknown complex environments via deep reinforcement learning. In: IEEE. **2019 IEEE International Conference on Mechatronics and Automation (ICMA)**. [S.l.], 2019. p. 2493–2500.

LIU, Y.; LIU, H.; WANG, B. Autonomous exploration for mobile robot using q-learning. In: IEEE. **2017 2nd International Conference on Advanced Robotics and Mechatronics (ICARM)**. [S.l.], 2017. p. 614–619.

LIU, Y.; NEJAT, G.; VILELA, J. Learning to cooperate together: A semi-autonomous control architecture for multi-robot teams in urban search and rescue. In: IEEE. **2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)**. [S.l.], 2013. p. 1–6.

LLUVIA, I.; LAZKANO, E.; ANSUATEGI, A. Active mapping and robot exploration: A survey. **Sensors**, MDPI, v. 21, n. 7, p. 2445, 2021.

LUO, T.; SUBAGDJA, B.; WANG, D.; TAN, A.-H. Multi-agent collaborative exploration through graph-based deep reinforcement learning. In: IEEE. **IEEE International Conference on Agents (ICA)**[S.l.], 2019. p. 2–7.

Makarenko, A. A.; Williams, S. B.; Bourgault, F.; Durrant-Whyte, H. F. An experiment in integrated exploration. In: **IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.: s.n.], 2002. v. 1, p. 534–539 vol.1.

MCFARLANE, R. A survey of exploration strategies in reinforcement learning. **McGill University**, 2018.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In: PMLR. **International conference on machine learning**. [S.l.], 2016. p. 1928–1937.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013.

MOERLAND, T. M.; BROEKENS, J.; JONKER, C. M. Model-based reinforcement learning: A survey. **arXiv preprint arXiv:2006.16712**, 2020.

MOHR, D.; SCHUELLER, S.; MONTAGUE, E.; BURNS, M.; RASHIDI, P. The behavioral intervention technology model: An integrated conceptual and technological framework for ehealth and mhealth interventions. **Journal of medical Internet research**, v. 16, p. e146, 06 2014.

MOHTASIB, A.; NEUMANN, G.; CUAYÁHUITL, H. A study on dense and sparse (visual) rewards in robot policy learning. In: SPRINGER. **Annual Conference Towards Autonomous Robotic Systems**. [S.l.], 2021. p. 3–13.

Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In: **2018 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.: s.n.], 2018. p. 6292–6299.

NEHMZOW, U. **Mobile robotics: a practical introduction**. [S.l.]: Springer Science & Business Media, 2012.

NIROUI, F.; ZHANG, K.; KASHINO, Z.; NEJAT, G. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. **IEEE Robotics and Automation Letters**, IEEE, v. 4, n. 2, p. 610–617, 2019.

Park, S.; Roh, K. S. Coarse-to-fine localization for a mobile robot based on place learning with a 2-d range scan. **IEEE Transactions on Robotics**, v. 32, n. 3, p. 528–544, 2016.

PHAM, H. X.; LA, H. M.; FEIL-SEIFER, D.; NEFIAN, A. Cooperative and distributed reinforcement learning of drones for field coverage. **arXiv preprint arXiv:1803.07250**, 2018.

PHAM, H. X.; LA, H. M.; FEIL-SEIFER, D.; NGUYEN, L. V. Reinforcement learning for autonomous uav navigation using function approximation. In: IEEE. **2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)**. [S.l.], 2018. p. 1–6.

POLYDOROS, A. S.; NALPANTIDIS, L. Survey of model-based reinforcement learning: Applications on robotics. **Journal of Intelligent & Robotic Systems**, Springer, v. 86, n. 2, p. 153–173, 2017.

PUTERMAN, M. L. **Markov decision processes: discrete stochastic dynamic programming**. [S.l.]: John Wiley & Sons, 2014.

Pérez-Dattari, R.; Celemin, C.; Ruiz-del-Solar, J.; Kober, J. Continuous control for high-dimensional state spaces: An interactive learning approach. In: **2019 International Conference on Robotics and Automation (ICRA)**. [S.l.: s.n.], 2019. p. 7611–7617.

RANA, K.; TALBOT, B.; DASAGI, V.; MILFORD, M.; SÜNDERHAUF, N. Residual reactive navigation: Combining classical and learned navigation strategies for deployment in unknown environments. In: IEEE. **2020 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.], 2020. p. 11493–11499.

RUAN, X.; REN, D.; ZHU, X.; HUANG, J. Mobile robot navigation based on deep reinforcement learning. In: IEEE. **2019 Chinese control and decision conference (CCDC)**. [S.l.], 2019. p. 6174–6178.

RUMMERY, G. A.; NIRANJAN, M. **On-Line Q-Learning Using Connectionist Systems**. [S.l.], 1994.

SCHULMAN, J.; LEVINE, S.; ABBEEL, P.; JORDAN, M.; MORITZ, P. Trust region policy optimization. In: PMLR. **International conference on machine learning**. [S.l.], 2015. p. 1889–1897.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **arXiv:1707.06347**, 2017.

SHI, H.; SHI, L.; XU et al. End-to-end navigation strategy with deep reinforcement learning for mobile robots. **IEEE Transactions on Industrial Informatics**, IEEE, v. 16, n. 4, p. 2393–2402, 2019.

Shimizu, T.; Saegusa, R.; Ikemoto, S.; Ishiguro, H.; Metta, G. Robust sensorimotor representation to physical interaction changes in humanoid motion learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 26, n. 5, p. 1035–1047, 2015.

SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots**. USA: Bradford Company, 2004. ISBN 026219502X.

Solanas, A.; Garcia, M. A. Coordinated multi-robot exploration through unsupervised clustering of unknown space. In: **2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)**. [S.l.: s.n.], 2004. v. 1, p. 717–721 vol.1.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

TAI, L.; LIU, M. Mobile robots exploration through cnn-based reinforcement learning. **Robotics and biomimetics**, Springer, v. 3, n. 1, p. 1–8, 2016.

TAI, L.; LIU, M. Towards cognitive exploration through deep reinforcement learning for mobile robots. **arXiv preprint arXiv:1610.01733**, 2016.

TAI, L.; PAOLO, G.; LIU, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2017. p. 31–36.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)**. [S.l.]: The MIT Press, 2005. ISBN 0262201623.

TZAFESTAS, S. **Introduction to Mobile Robot Control**. [S.l.]: Elsevier, 2013.

Velásquez Hernández, C. A.; Prieto Ortiz, F. A. A real-time map merging strategy for robust collaborative reconstruction of unknown environments. **Expert Systems with Applications**, v. 145, p. 113109, 2020. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417419308267>.

VENKAT, N. **The curse of dimensionality: Inside out**. Tese (Doutorado) — PhD thesis, Dept. of CSIS, BITS Pilani, 2018.

VENTURINI, F.; MASON, F.; PASE, F.; CHIARIOTTI, F.; TESTOLIN, A.; ZANELLA, A.; ZORZI, M. Distributed reinforcement learning for flexible uav swarm control with transfer learning capabilities. In: **Proceedings of the 6th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications**. [S.l.: s.n.], 2020. p. 1–6.

WALKER, O.; VANEGAS, F.; GONZALEZ, F.; KOENIG, S. Multi-uav target-finding in simulated indoor environments using deep reinforcement learning. In: IEEE. **IEEE Aerospace Conference**. [S.l.], 2020. p. 1–9.

Wang, C.; Ma, H.; Chenc, W.; Liu, L.; Meng, M. Q. . Efficient autonomous exploration with incrementally built topological map in 3d environments. **IEEE Transactions on Instrumentation and Measurement**, p. 1–1, 2020.

WANG, X.; WANG, S.; LIANG, X.; ZHAO, D.; HUANG, J.; XU, X.; DAI, B.; MIAO, Q. Deep reinforcement learning: A survey. **IEEE Transactions on Neural Networks and Learning Systems**, p. 1–15, 2022.

WATKINS, C. J. C. H. **Learning from delayed rewards**. Tese (Doutorado) — King's College, Cambridge United Kingdom, 1989.

YAMAUCHI, B. A frontier-based approach for autonomous exploration. In: IEEE. **IEEE International Symp. on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'**. [S.l.], 1997. p. 146–151.

YAN, Z.; JOUANDEAU, N.; CHERIF, A. A. A survey and analysis of multi-robot coordination. **International Journal of Advanced Robotic Systems**, v. 10, 2013. ISSN 17298806.

Yang, Y.; Juntao, L.; Lingling, P. Multi-robot path planning based on a deep reinforcement learning dqn algorithm. **CAAI Transactions on Intelligence Technology**, v. 5, n. 3, p. 177–183, 2020.

Yang, Z.; Merrick, K.; Jin, L.; Abbass, H. A. Hierarchical deep reinforcement learning for continuous action control. **IEEE Transactions on Neural Networks and Learning Systems**, v. 29, n. 11, p. 5174–5184, 2018.

YIJING, Z.; ZHENG, Z.; XIAOYI, Z.; YANG, L. Q learning algorithm based uav path learning and obstacle avoidence approach. In: IEEE. **2017 36th Chinese Control Conference (CCC)**. [S.l.], 2017. p. 3397–3402.

YU, C.; DONG, Y.; LI, Y.; CHEN, Y. Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit. **The Journal of Engineering**, IET, v. 2020, n. 13, p. 499–504, 2020.

YUE, W.; GUAN, X.; XI, Y. Reinforcement learning based approach for multi-uav cooperative searching in unknown environments. In: IEEE. **2019 Chinese Automation Congress (CAC)**. [S.l.], 2019. p. 2018–2023.

Zeng, X. **Reinforcement learning based approach for the navigation of a pipe-inspection robot at sharp pipe corners**. 2019. Disponível em: <http://essay.utwente.nl/79790/>.

ZHELO, O.; ZHANG, J.; TAI, L.; LIU, M.; BURGARD, W. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. **arXiv preprint arXiv:1804.00456**, 2018.

ZHOU, B.; WANG, W.; WANG, Z.; DING, B. Neural q learning algorithm based uav obstacle avoidance. In: IEEE. **2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)**. [S.l.], 2018. p. 1–6.

ZHOU, X.; WANG, W.; WANG, T.; LEI, Y.; ZHONG, F. Bayesian reinforcement learning for multi-robot decentralized patrolling in uncertain environments. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 12, p. 11691–11703, 2019.

ZHU, D.; LI, T.; HO, D.; WANG, C.; MENG, M. Q.-H. Deep reinforcement learning supervised autonomous exploration in office environments. In: IEEE. **2018 IEEE international conference on robotics and automation (ICRA)**. [S.l.], 2018. p. 7548–7555.

ZHU, Y.; MOTTAGHI, R.; KOLVE, E.; LIM, J. J.; GUPTA, A.; FEI-FEI, L.; FARHADI, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: IEEE. **2017 IEEE international conference on robotics and automation (ICRA)**. [S.l.], 2017. p. 3357–3364.

APPENDIX A — RESUMO EXPANDIDO

Nos últimos anos, o avanço de novas tecnologias aplicadas à robótica tem impulsionado o interesse pela pesquisa acadêmica e pela aplicação prática de robôs móveis em diversos domínios. Tais aplicações incluem, por exemplo, missões de busca e resgate, inteligência, vigilância e reconhecimento, e exploração planetária, entre outras [Alatise e Hancke 2020]. Várias situações requerem autonomia robótica quando um mapa do ambiente não é previamente conhecido. O emprego de robôs móveis em contextos tão complicados depende de uma estratégia de exploração robusta e eficiente. Na literatura, a exploração robótica móvel foi descrita como a tentativa de responder à pergunta *"Dado o que você sabe sobre o mundo, para onde você deve se mover para obter o máximo de novas informações possível?"* [Yamauchi 1997]. O comportamento exploratório é uma competência robótica fundamental e representa um vasto e complexo campo de pesquisa. Vários métodos de exploração foram propostos nas últimas décadas, como os Campos de Potencial Artificial [Krogh e Thorpe 1986] e a conhecida exploração baseada em Fronteiras [Yamauchi 1997], permitindo a exploração em muitas aplicações.

A abordagem tradicional para exploração autônoma de robôs móveis em ambientes desconhecidos é usar dados de sensores para construir um mapa do mundo ao redor do agente. Então, o processo de tomada de decisão sobre onde ir a seguir é baseado no mapa gerado. Uma limitação deste método é que os custos computacionais aumentam rapidamente com a expansão da área explorada. Além disso, a eficiência da estratégia de exploração depende fortemente de mapas precisos [Juliá, Gil e Reinoso 2012]. Além disso, quando um sistema multi-robô realiza uma exploração colaborativa, os mapas gerados por cada agente são frequentemente compartilhados entre a equipe. Os mapas compartilhados devem ser combinados corretamente, o que não é uma tarefa trivial, muitas vezes associada a altos custos computacionais [Velásquez Hernández e Prieto Ortiz 2020]. Tais características podem restringir a aplicação desses métodos em cenários onde a disponibilidade de memória e recursos de processamento são limitados.

Alternativamente, se um robô pudesse realizar exploração sem construir e armazenar um mapa preciso do mundo, seria possível liberar memória e recursos computacionais que poderiam ser empregados em uma tomada de decisão mais rápida. No entanto, desenvolver uma estratégia de exploração adequada e resiliente não é uma tarefa simples, especialmente sem a utilização de mapas. Ao mesmo tempo, as técnicas de Aprendizagem por Reforço baseiam-se em permitir que o agente adquira habilidades por meio da interação com o ambiente, em vez de projetar explicitamente os comportamentos desejados. Esse paradigma de Aprendizado de Máquina tenta emular o processo de aprendizagem humana, que ocorre por tentativa e erro.

Recentemente, o emprego de redes neurais profundas como aproximadores de função poderosos em algoritmos de Aprendizado por Reforço tem viabilizado soluções ótimas ou quase ótimas para vários problemas complexos, com entradas de alta dimensionalidade [Wang et al. 2022]. Assim, técnicas de Aprendizado por Reforço Profundo (ARP) vêm se destacando como alternativas promissoras para desenvolver soluções para problemas de robótica. Não é por acaso que o número de trabalhos de pesquisa propondo soluções para o problema de exploração robótica usando esses algoritmos aumentou significativamente nos últimos anos [Garaffa et al. 2021]. No que diz respeito ao problema de exploração sem geração de mapa, vários trabalhos usam ARP para navegar em ambientes desconhecidos com o objetivo de atingir uma posição de destino conhecida. No entanto, até onde sabemos, não existem trabalhos que investiguem a aplicação de soluções ARP para exploração sem mapa visando uma cobertura eficiência de área, sem posições-alvo pré-determinadas.

A.1 Contribuições e Objetivos Alcançados

O objetivo geral desta dissertação foi investigar se técnicas de Aprendizado por Reforço Profundo podem viabilizar estratégias de exploração sem mapa eficientes e robustas para um ou mais robôs móveis. Primeiro, revisamos trabalhos de pesquisa recentes que usam ARP para projetar estratégias de exploração de ambientes desconhecidos. A revisão buscou compilar o estado atual da pesquisa que relaciona esses dois domínios de conhecimento, com o objetivo de compreender como os dois campos estão sendo integrados. O estudo elaborado para esta dissertação foi publicado na revista IEEE Transactions on Neural Networks and Learning Systems [Garaffa et al. 2021].

Com base nas informações coletadas, propusemos uma arquitetura de exploração sem mapa baseada em ARP e adequado para robôs individuais e equipes de n robôs, sendo esta a principal contribuição deste trabalho. Por meio do sistema proposto, avaliamos se o ARP é capaz de aprender quais ações resultam em uma cobertura eficiente do ambiente a partir de informações simplificadas sobre o ambiente, como medições de laser e dados de odometria. A solução proposta se concentra em ambientes internos e foi projetada para priorizar a eficiência de cobertura e evitar colisões. Usando uma abordagem descentralizada, cada robô executa localmente um sistema que determina a política de exploração individual. A ideia foi obter agentes capazes de explorar o ambiente sozinhos, mas que podem otimizar seu processo de tomada de decisão caso recebam informações de outros agentes. A base do processo de tomada de decisão é o algoritmo Proximal Policy Optimization (PPO), e redes neurais artificiais profundas são empregadas como aproximadores de função. A política de exploração é treinada e testada em diferentes ambientes de simulação usando um e dois robôs. Nossa solução viabilizou exploração com eficiência comparável a métodos que usam representações muito mais complexas do ambiente. O método também promoveu a cooperação entre agentes sem a necessidade de algoritmos de fusão de mapas, sendo capaz de generalizar para diferentes ambientes.

A.2 Trabalho Futuro

Todos os resultados obtidos neste trabalho foram coletados em ambientes controlados e as conclusões apresentadas são válidas considerando as dimensões e os níveis de complexidade dos mapas testados. Em trabalhos futuros, pretendemos avaliar o desempenho da arquitetura de exploração em ambientes não ideais, com incertezas nas medições dos sensores. A avaliação do método colaborativo será realizada considerando uma comunicação instável, onde os robôs não são capazes de compartilhar informações a cada passo. Além disso, o método deve ser testado com mais de dois robôs. Ambientes mais complexos e dinâmicos também devem ser considerados. Diferentes formatos de recompensa podem ser testados para aumentar a eficiência da cobertura. A comparação com outras estratégias de exploração colaborativa também é uma análise futura essencial. Esses experimentos abrem caminho para o objetivo final da pesquisa: testar a estratégia de exploração em robôs reais.