UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Università degli studi di Modena
e Reggio Emilia

Dipartimento di Scienze e Metodi dell'Ingegneria

Corso di Laurea Magistrale in Digital Automation Engineering

Deep Reinforcement Learning in Frontier-Based Autonomous Exploration

Relatore:
Prof. Lorenzo Sabattini

Tesi di Laurea di:
Paolo Spallanzani

Correlatori:
Dott. Mattia Catellani
Dott. Mattia Mantovani

Anno Accademico 2024/2025

Abstract

Autonomous exploration of unknown environments is a fundamental problem in mobile robotics, requiring an agent to simultaneously build a map of its surroundings and decide where to move next in order to maximize the area discovered. Frontier-based exploration is one of the most established approaches to this task, in which the agent iteratively navigates toward the boundaries between known and unknown regions of the map. In classical frontier-based methods, the selection of the next target frontier is governed by deterministic heuristic policies, such as choosing the nearest frontier or the one with the highest information gain. While effective, these strategies are inherently rigid and may perform suboptimally in structurally complex or varied environments.

This work investigates the integration of a Deep Reinforcement Learning (DRL) decision-making module within a frontier-based exploration framework, replacing the heuristic selection policy with a learned one. A modular control architecture is adopted, in which path planning and mapping rely on established exact methods, while a Deep Q-Network (DQN) model is trained to select among available frontier cluster centroids at each step. A custom two-dimensional grid simulation environment was developed on the Gymnasium framework and used to systematically explore a range of observation representations, reward formulations, and training configurations.

A key finding of the work is the identification and diagnosis of a reward hacking phenomenon, in which the combination of distance-sorted observations and a truncation penalty inadvertently caused the agent to learn a degenerate policy replicating the greedy distance heuristic. Resolving this issue enabled the training of a genuinely

non-degenerate policy, which achieved a mean total reward of 15.55 ± 2.45 on the primary evaluation environment, compared to 16.48 ± 1.78 for the greedy distance baseline. The result was further validated on a larger grid geometry, confirming partial generalization of the learned policy. The simulation framework developed in this work is released as a flexible and extensible testbed for future research in DRL-based autonomous exploration.

Sommario

Sommario

Motivazioni

Uno degli obiettivi più ambiziosi della robotica moderna è la realizzazione di agenti autonomi capaci di operare in ambienti sconosciuti senza alcun intervento umano diretto. Le applicazioni di questa tecnologia sono molteplici e spaziano dalla ricerca e il salvataggio in ambienti pericolosi all'esplorazione di territori remoti o inaccessibili, dalla sorveglianza automatica al monitoraggio ambientale. In tutti questi scenari, la capacità di muoversi autonomamente in un luogo mai visitato prima — costruendo progressivamente una mappa dell'ambiente e decidendo dove spostarsi a ogni passo — rappresenta una competenza fondamentale.

Il problema dell'esplorazione autonoma è, nonostante la sua apparente semplicità, di grande complessità. Un agente che esplora un ambiente ignoto deve fare tutto contemporaneamente: percepire ciò che lo circonda, aggiornare la propria rappresentazione dello spazio, pianificare un percorso sicuro e decidere dove andare successivamente, il tutto sulla base di informazioni parziali e in continua evoluzione. Negli esseri umani queste capacità sono così naturali da sembrare banali; per un sistema artificiale, formalizzarle e implementarle in modo efficace è tutt'altro che scontato.

Tra i metodi sviluppati negli ultimi decenni per affrontare questo problema, l'esplorazione basata su frontiere si è affermata come uno degli approcci più solidi e diffusi. L'idea di fondo è intuitiva: le *frontiere* sono le zone di confine tra la parte dell'ambiente già esplorata e quella ancora sconosciuta. Navigando sistematicamente verso queste zone, l'agente espande progressivamente la propria conoscenza dell'ambiente, finché

non rimane più nessuna frontiera da raggiungere e l'esplorazione può considerarsi completa. Nella sua forma classica, la scelta di quale frontiera raggiungere è affidata a regole semplici e deterministiche, come dirigersi verso la più vicina oppure verso quella che promette di rivelare il maggior numero di nuove informazioni. Queste strategie sono efficienti e ben comprese, ma hanno alcuni limiti: presentano una certa rigidità, possono non addattarsi alle caratteristiche specifiche dell'ambiente o risultare subottimali in situazioni complesse o variate.

È in questo contesto che si inserisce il presente lavoro. Negli ultimi anni, i metodi basati sull'apprendimento automatico — e in particolare sul *Deep Reinforcement Learning* (DRL) — hanno dimostrato capacità straordinarie nel risolvere problemi decisionali complessi. Tra le altre strategie questi approcci sfruttano l'allenamento di reti neurali imparando direttamente dall'esperienza senza che le regole del comportamento ottimale vengano definite esplicitamente dal progettista. L'obiettivo di questa tesi è investigare se e in che misura un modello DRL possa essere integrato nell'approccio basato su frontiere sostituendo la politica euristica classica di selezione, aprendo così la strada a strategie di esplorazione più flessibili e adattive.

Metodi di indagine

Per affrontare questa domanda è stato sviluppato un sistema di controllo modulare, in cui i componenti di costruzione della mappa e pianificazione del percorso si basano su metodi esatti e consolidati, mentre il modulo decisionale — ovvero la scelta di quale frontiera raggiungere a ogni passo — è affidato a un agente DRL. Questo approccio ibrido consente di ridurre la complessità del problema di apprendimento, permettendo all'agente neurale di concentrarsi esclusivamente sulla decisione strategica ad alto livello senza dover imparare anche a muoversi o a costruire la mappa.

Il modello di apprendimento adottato è il *Deep Q-Network* (DQN), un algoritmo di reinforcement learning che utilizza una rete neurale per stimare il valore di ciascuna azione disponibile in un dato stato, e seleziona l'azione con il valore atteso più alto. L'agente interagisce con l'ambiente, riceve un segnale di rinforzo a ogni passo — positivo se l'azione ha contribuito all'esplorazione, negativo in caso contrario —

e aggiorna progressivamente la propria strategia per massimizzare la ricompensa cumulativa nel lungo periodo.

Un contributo centrale di questo lavoro è lo sviluppo di un ambiente di simulazione personalizzato, costruito sulla libreria Gymnasium e compatibile con Stable-Baselines3, appositamente progettato per lo studio e la sperimentazione del DRL applicato all'esplorazione basata su frontiere. L'ambiente modella un mondo a griglia bidimensionale di dimensioni configurabili, con ostacoli distribuiti casualmente e un agente dotato di un raggio di percezione limitato. Il sistema include un modulo di rilevamento e clustering delle frontiere, un algoritmo di navigazione con evitamento degli ostacoli basato su A*, e un'interfaccia flessibile per la definizione dello spazio delle osservazioni e della funzione di ricompensa. Questo framework ha permesso di condurre una campagna sperimentale sistematica, variando la rappresentazione delle osservazioni fornite al modello, la formulazione della ricompensa e la durata del training.

Le configurazioni di osservazione testate includono posizioni assolute e relative dei centroidi dei cluster di frontiera, l'aggiunta della posizione dell'agente, il guadagno informativo associato a ciascun cluster, e diverse combinazioni di questi elementi, con e senza ordinamento per distanza. Parallelamente, sono state sviluppate e confrontate due formulazioni della funzione di ricompensa, progressivamente raffinate a partire da tentativi iniziali più semplici che non producevano comportamenti stabili. Le prestazioni del modello sono state confrontate con due politiche euristiche di riferimento: la selezione della frontiera più vicina e la selezione di quella con il maggior guadagno informativo.

Risultati ottenuti

La campagna sperimentale ha prodotto risultati significativi sia dal punto di vista pratico che metodologico. Uno dei contributi più rilevanti è l'identificazione e la diagnosi di un fenomeno di *reward hacking*: in diverse configurazioni, il modello imparava una politica apparentemente efficace — con ricompense comparabili a quelle dell'euristica greedy — ma in realtà del tutto degenerata, selezionando quasi sempre la stessa azione indipendentemente dallo stato dell'ambiente. L'analisi ha rivelato che questo comportamento era causato dall'interazione tra l'ordinamento delle osser-

vazioni per distanza e una penalità di troncamento eccessivamente forte: il modello imparava a evitare la penalità selezionando sistematicamente la prima azione, che per effetto dell'ordinamento corrispondeva sempre alla frontiera più vicina, replicando di fatto l'euristica greedy senza aver appreso alcuna strategia genuina. La diagnosi è stata confermata rimuovendo l'ordinamento, il che ha causato il crollo immediato delle prestazioni.

La soluzione a questo problema — rimozione dell'ordinamento per distanza, utilizzo di posizioni assolute dei centroidi e sostituzione della penalità di troncamento con una formulazione più morbida — ha permesso, per la prima volta nel corso di tutti gli esperimenti, di addestrare una politica genuinamente non degenerata. Il modello risultante seleziona azioni diverse in risposta allo stato dell'ambiente, distribuendo le proprie scelte su più frontiere disponibili anziché concentrarsi su una sola, implementando quindi una strategia orihìginale differente da quella euristica. Valutato su 30 episodi in un ambiente statico 20×20, questo modello ottiene una ricompensa media di 15.55 ± 2.45, a fronte di 16.48 ± 1.78 per l'euristica greedy per distanza e 15.01 ± 1.12 per l'euristica greedy per guadagno informativo. Il modello si colloca quindi tra le due euristiche, dimostrando di aver appreso una strategia competitiva.

Una verifica di robustezza su una griglia più grande (30×30, con raggio di percezione maggiore) ha confermato che la politica appresa generalizza parzialmente a geometrie di ambiente diverse da quella di training, ottenendo una ricompensa media di 31.46 ± 4.75 rispetto a 36.39 ± 1.27 dell'euristica greedy per distanza. Il divario con le euristiche è più ampio rispetto al caso della griglia piccola, indicando che la complessità maggiore dell'ambiente — un numero più elevato di frontiere disponibili a ogni passo e sequenze decisionali più lunghe — richiede ulteriori sviluppi per essere gestita efficacemente da un modello appreso.

Nel complesso, questo lavoro dimostra la fattibilità dell'integrazione del DRL in un sistema di esplorazione basato su frontiere e fornisce evidenza sperimentale che un modello DQN può apprendere una politica decisionale competitiva con le euristiche classiche. Il framework di simulazione sviluppato costituisce uno strumento validato e flessibile per future investigazioni in questa direzione, e i risultati ottenuti — incluse le criticità identificate e risolte — offrono una base concreta per l'estensione

del lavoro verso ambienti più complessi, rappresentazioni delle osservazioni più ricche e architetture di apprendimento più espressive.

Contents

Sommario **V**

1 Introduction **1**

2 Technical Background **5**

2.1 Autonomous Navigation and Exploration 5

2.1.1 The Autonomous Exploration Problem 5

2.1.2 Frontier-Based Exploration 6

2.2 Deep Reinforcement Learning 8

2.2.1 Reinforcement Learning 8

2.2.2 Deep Q-Network 10

3 System Design **13**

3.1 Overall Design 13

3.2 Modular Architecture 14

4 Agent-Environment System Simulator **17**

4.1 2D Occupancy grid 17

4.2 Frontier Detection and Navigation 19

4.3 Software Architecture 20

4.3.1 The Gymnasium Framework 21

4.3.2 Observation Space 22

5 The DRL decision-making module **25**

5.1 Problem Formulation as a Markov Decision Process 25

5.1.1 Reward Shaping 26
5.2 DQN Implementation and Training Setup 27

6 Experiments and Results **29**
6.1 Experimental Setup 30
6.1.1 Observation Space Configurations 31
6.1.2 Reward Configuration 31
6.1.3 Hyperparameter Optimization 33
6.2 Heuristic Baselines 33
6.3 Training Analysis 35
6.3.1 Degenerate Policy: Reward Hacking via Observation Sorting . 36
6.3.2 Non-Degenerate Policy: Absolute Positions Without Truncation 37
6.4 Evaluation and Comparison 39
6.4.1 Degenerate Configurations 40
6.4.2 Non-Degenerate Policy: Main Result 40
6.4.3 Robustness Verification 41

7 Conclusions **43**

List of Figures

2.1 Example of frontier-based navigation on a 2D world grid 7
2.2 The agent–environment interaction loop in reinforcement learning. . . 9
2.3 Schematic of a Deep Q-Network. 10

3.1 Modular control architecture of the proposed system 15

4.1 Examples of grid environments with different sizes and obstacle probabilities 18
4.2 Examples of frontier detection and clustering on partially explored maps 19
4.3 Example of frontier cells cluster splitting 20
4.4 Example of obstacle-free path from A* algorithm 21
4.5 Class hierarchy of the simulation software. 21
4.6 Block diagram of the `step()` function dynamics 22

6.1 Overview of the experimental process 30
6.2 Episode reward during training for configuration 4 (relative centroid positions, sorted) 36
6.3 TD loss ($\log_{10}$ scale) during training for configuration 4 37
6.4 Episode reward during training for configuration 1 (absolute centroid positions with agent position, unsorted, 10^6 steps) 38
6.5 TD loss ($\log_{10}$ scale) during training for configuration 1 39

List of Tables

4.1 Configurable observation space representations. 23
4.2 Example of a padded observation vector with three active clusters and three padding slots. 23

5.1 Comparison of the reward formulations used in the experimental campaign 26
5.2 Final DQN hyperparameter optimal configuration 28

6.1 Observation space configurations evaluated during the experimental campaign 31
6.2 Comparison of the two reward formulations used in the experimental campaign. Entries marked "= R1" are identical to Reward 1. 32
6.3 Hyperparameter search ranges and optimal values selected for the DRL decision-making module. 33
6.4 Final hyperparameter configuration used across all reported experiments. 34
6.5 Heuristic baselines evaluation results 35
6.6 Evaluation results for the selected configurations compared against the heuristic baselines 39

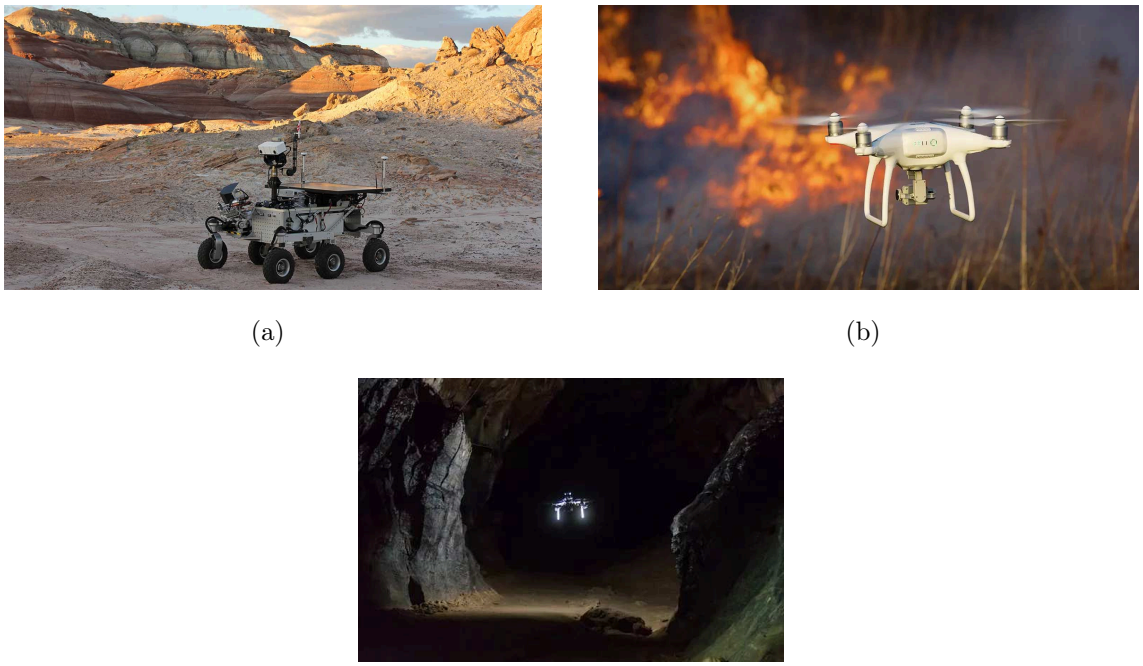
Chapter 1

Introduction

The deployment of autonomous agents in real-world scenarios has attracted growing interest across a wide range of fields, from industrial inspection and logistics to search-and-rescue operations and planetary exploration. The appeal of autonomous systems lies in their ability to operate without direct human presence or supervision, reducing exposure to hazardous conditions, lowering operational costs, providing increased effectiveness, and enabling deployment in environments that are difficult or impossible for humans [1, 2].

Among the capabilities that define a truly autonomous mobile agent, the ability to navigate and explore unknown environments stands out as both fundamental and challenging. Unlike tasks carried out in fully known settings, exploration of unmapped surroundings requires the agent to simultaneously build a representation of the environment and use that representation to make decisions about where to move next. What may seem an intuitive task for a human being reveals itself, upon closer inspection, to be a highly complex control problem: it involves continuous sensing, incremental map building, and an ongoing decision-making process with a potentially infinite number of degrees of freedom [1].

Several systematic approaches have been proposed to address autonomous exploration. Among these, frontier-based exploration has established itself as one of the most widely adopted methods [3, 4]. In this framework, the agent navigates



(a) (b)

(c)

Figure 1.1: Examples of autonomous exploration real world applications.

toward *frontier* regions, defined as the boundaries between the known and unknown portions of the map. By iteratively targeting these regions, the agent progressively extends its knowledge of the environment. The decision of which frontier to target at each step is typically governed by a deterministic heuristic policy, such as selecting the nearest frontier or the one promising the highest information gain. While these strategies are computationally efficient and well understood, their rigidity can limit performance in environments that are complex, dynamic, or structurally varied [4].

These limitations have motivated a growing body of research into learning-based approaches for autonomous exploration. These methods, which often include the use of trained neural networks, are well suited for solving complex problems as the one depicted here. In particular, Deep Reinforcement Learning (DRL) has emerged as a promising paradigm, owing to its ability to learn flexible control policies directly from interaction with an environment, without requiring an explicit analytical model [5, 6]. DRL methods have demonstrated remarkable results in complex sequential decision-making tasks, and their application to exploration is an active area of investigation.

This work proposes an approach that combines the strengths of both paradigms. Rather than replacing the entire exploration pipeline with a learned model, a modular control architecture is adopted in which the path planning and obstacle avoidance components rely on established exact methods, while the high-level decision-making process — namely, the selection of the next frontier region to navigate toward — is entrusted to a DRL agent. Specifically, a Deep Q-Network (DQN) model is trained to select among a set of frontier cluster centroids at each step of the exploration process.

To this end, a custom simulation environment has been developed, modelling a two-dimensional grid world explored by an autonomous agent equipped with a limited perception range. The environment, built on the Gymnasium framework and compatible with the Stable-Baselines3 training library, exposes configurable observation spaces and supports systematic variation of environment parameters. Several combinations of observation representations, reward formulations, and training configurations have been investigated. The trained models are evaluated against two heuristic baselines — greedy distance and greedy information gain — to assess the effectiveness of the learned policy.

The remainder of this thesis is structured as follows. Chapter 2 reviews the theoretical and technical background, covering frontier-based exploration, reinforcement learning, and the DQN algorithm. Chapter 3 describes the system design and the simulation environment, including the frontier detection pipeline and the path planning module, which describes the main logic of the system. Chapter 5 presents the formulation of the exploration task as a Markov Decision Process and details the design of the DRL agent, including the observation and action spaces and the reward shaping strategy. Chapter 6 reports the experimental setup, training analysis, and a comparative evaluation of the trained models against the heuristic baselines. Finally, Chapter 7 draws conclusions and outlines directions for future work.

Chapter 2

Technical Background

2.1 Autonomous Navigation and Exploration

Enabling a mobile agent to navigate autonomously in an unknown environment requires a heterogeneous set of capabilities to operate in tight coordination. Perception, map building, motion planning, and high-level decision-making must all work together, each feeding into the next. This section introduces the autonomous exploration problem, breaking it down into its main components and identifying the decision-making layer as the central focus of this work.

2.1.1 The Autonomous Exploration Problem

The task of autonomous exploration can be stated simply: an agent must move through an initially unknown environment, build a representation of it, and decide at each step where to go next to maximize the area discovered. In practice, every decision is made on the basis of partial, incrementally updated information, which makes the problem considerably harder than it appears.

A natural way to handle this complexity is to decompose the system into three functionally independent modules [6, 7], as illustrated in Figure 3.1.

The *mapping module* processes sensor data and maintains a representation of the

environment. In grid-based approaches, this is represented by an occupancy grid: a spatial discretization of the environment in which each cell is labeled as free, occupied, or unknown. As the agent moves, the known portion of the grid grows. Estimating the agent's position within this map is the localization problem, and the two tasks together constitute what is known as Simultaneous Localization and Mapping (SLAM) [1].

The *path planning module* computes a collision-free trajectory from the agent's current position to a target location selected by the decision module. The A* algorithm is a standard choice for this role in grid-based maps, providing the shortest-path in the known portion of the environment [6].

The *decision-making module* determines where the agent should go next. Given the current map and the agent's position, it selects the next target location:

$$g_t = f_{\text{decision}}(m_t), \tag{2.1}$$

where g_t is the goal point at time t, $l_{0:t}$ is the history of agent positions, and m_t is the map at time t [6]. This is the layer at which the exploration strategy is defined, and it is the focus of the present work.

2.1.2 Frontier-Based Exploration

The frontier-based (FB) approach to autonomous exploration was first introduced by Yamauchi [3] and has since become one of the most widely adopted strategies in the field. Its central idea is straightforward: the agent should navigate toward the boundaries between the known and unknown portions of the map, progressively extending its knowledge of the environment.

These boundaries are called *frontiers*. More precisely, a frontier cell is an unknown cell that is adjacent to at least one known cell [3]. In most FB algorithms frontiers cells from contiguous regions are grouped together into *frontier clusters*, the centroid of each of theme is computed and provided as a convenient navigation target. As the agent moves toward a frontier and perceives new cells, the map is updated, old frontiers may disappear, and new ones emerge. The process continues until no frontier cells remain, at which point the environment is considered fully explored. Figure 2.1 illustrates the key concepts.

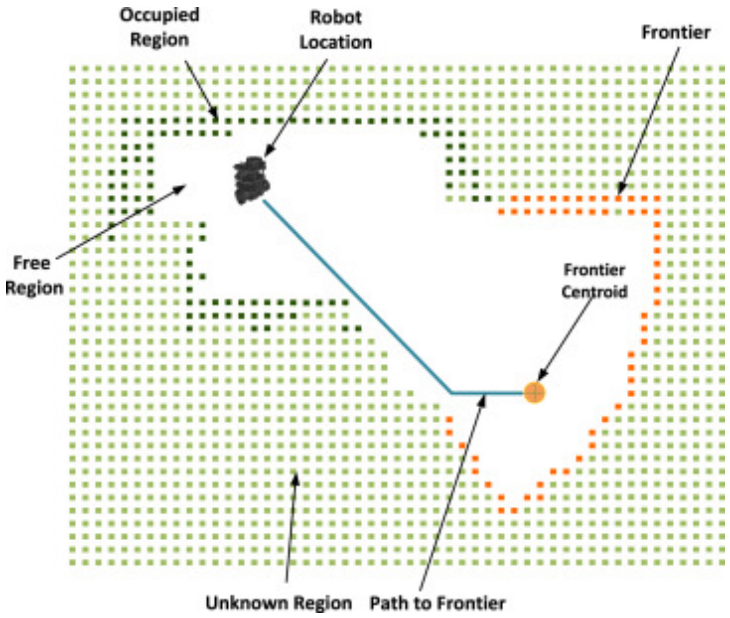


Figure 2.1: Example of frontier-based navigation on a 2D world grid. Frontier cells lie on the boundary between known free space and unknown space. The agent navigates toward the centroid of a frontier cluster to expand the explored area.

Once frontiers have been identified, the agent must decide which one to target. In classic frontier-based exploration this decision is made by a deterministic heuristic. The two most common choices are the *nearest frontier* and the *highest information gain* frontier. In the first one the cluster centroid located at the minimum distance is seeked as target. In the second one an estimation of the information gained by reaching the frontier, i.e. the number of new cells that would be revealed, is computed and the cluster presenting the highest value is selected as target [8, 4]. Both strategies are simple and computationally cheap, but they also have limits: they optimize a single step without accounting for the global structure of the unexplored space. In complex environments this can lead to inefficient trajectories, for instance repeatedly revisiting already-explored areas or ignoring large unknown regions that are slightly farther away [7].

This limitation is the primary motivation for replacing the heuristic policy with a learned one. Rather than committing to a fixed rule, a learning-based decision module can in principle discover strategies that are better suited to the structure of the environment. This is the approach explored in the present work, where a DRL agent is trained to select among the available frontier clusters at each step of the

exploration process.

2.2 Deep Reinforcement Learning

Machine learning methods have proven remarkably effective at solving tasks that resist explicit analytical formulation. Rather than encoding a solution by hand, these methods allow a model to extract patterns and develop strategies directly from data. Among them, deep learning — the use of multi-layer neural networks as function approximators — has produced particularly striking results across domains as diverse as image recognition, natural language processing, and game playing. The key advantage of neural networks in this context is their ability to learn high-level features from complex, high-dimensional inputs, without requiring the designer to specify in advance which are relevant.

Deep Reinforcement Learning (DRL) combines this potential with the reinforcement learning framework, in which an agent learns a behavioral policy by interacting with an environment and receiving a feedback in the form of scalar rewards. The result is a class of methods that can tackle sequential decision-making problems of considerable complexity, learning directly from experience rather than from a pre-existent dataset. It is precisely this combination of properties that makes DRL a natural candidate for the autonomous exploration problem.

2.2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework for sequential decision-making in which an agent interacts with an environment over a series of discrete time steps. Here the agent has the ability to gather an observation of either a portion or the entire environment, and based on that, chose an action to perform that will in turn may produce a change in the in the environment. At each step t, the agent observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, and receives a scalar reward $r_t = R(s_t, a_t)$ from the environment, which transitions to a new state s_{t+1}. This interaction loop is illustrated in Figure 2.2.

This interaction is formally modelled as a *Markov Decision Process* (MDP) [9]. An MDP is built on the *Markov property*: the next state s_{t+1} depends only on the

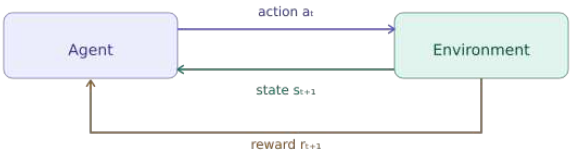


Figure 2.2: The agent–environment interaction loop in reinforcement learning. At each step the agent observes the state, selects an action, and receives a reward and the next state.

current state s_t and the action a_t, and not on any earlier history. This means that the current state is a sufficient statistic for decision-making, and that the agent does not need to remember the full sequence of past events to act optimally. Formally, an MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $T(s_{t+1} \mid s_t, a_t)$ is the transition probability distribution, R is the reward function, and $\gamma \in [0, 1)$ is a discount factor that controls the relative weight of immediate versus future rewards.

The agent's behaviour is described by a *policy* $\pi(a \mid s)$, which defines a probability distribution over actions given the current state. The objective is to find the policy that maximises the expected discounted cumulative reward, or *return*:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.2}$$

To evaluate how good it is to be in a given state, or to take a given action in a given state, RL methods make use of value functions. The *state-value function* $V^\pi(s)$ measures the expected return when starting from state s and following policy π. More directly useful for action selection is the *action-value function*, or *Q-function*:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t = s,\, a_t = a \right], \tag{2.3}$$

which gives the expected return of taking action a in state s and then following π. The optimal Q-function $Q^*(s, a)$ satisfies the *Bellman optimality equation*:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s', a') \;\middle|\; s, a \right]. \tag{2.4}$$

Once Q^* is known, the optimal policy follows immediately by selecting at each step the action with the highest Q-value: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

2.2.2 Deep Q-Network

In problems with small, discrete state spaces, the Q-function can be represented as a lookup table and updated directly using the *Q-learning* rule [10]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right], \tag{2.5}$$

where α is the learning rate. When the state space is large or continuous, however, tabular representation becomes intractable. The *Deep Q-Network* (DQN), introduced by Mnih et al. [5], addresses this by approximating the Q-function with a neural network $Q(s, a; \theta)$, parameterised by weights θ. The network takes the state as input and outputs a Q-value for each possible action, as illustrated in Figure 2.3.

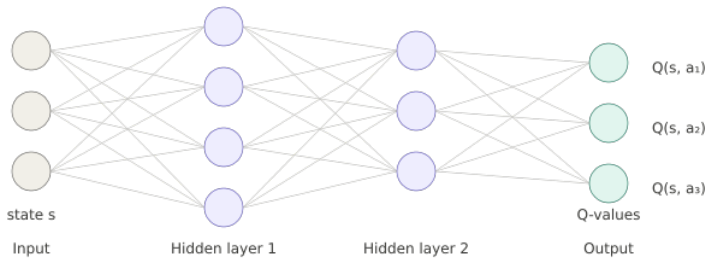


Figure 2.3: Schematic of a Deep Q-Network. The state is fed as input to a fully connected neural network, which outputs one Q-value per action. The action with the highest Q-value is selected.

Training the network naively by minimizing the mean squared Bellman error is known to be unstable, due to correlations between consecutive transitions and the non-stationarity of the target values. DQN addresses these issues with two key mechanisms.

Experience replay [5] stores the agent's transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ in a replay buffer $\mathcal{D}$ of fixed size. During training, mini-batches are sampled uniformly at random from $\mathcal{D}$, breaking the temporal correlations between consecutive updates and improving sample efficiency.

The *target network* is a periodically updated copy of the main network, with weights θ^-, used to compute the target Q-values during training. The loss minimized at each

step is:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a';\, \theta^-) - Q(s, a;\, \theta)\right)^2\right]. \tag{2.6}$$

Keeping the target fixed for a number of steps before synchronizing with the main network stabilizes training by decoupling the update target from the parameters being optimized.

Finally, action selection during training follows an ε-*greedy* policy: with probability ε the agent selects a random action (exploration), and with probability $1 - \varepsilon$ it selects the action with the highest Q-value (exploitation). The value of ε is typically annealed from 1 to a small minimum over the course of training, gradually shifting the balance from exploration towards exploitation.

Chapter 3

System Design

3.1 Overall Design

As discussed in the previous chapter, autonomous exploration can be effectively addressed through reinforcement learning-based algorithms. One possible formulation of this problem is to have the learning-based model entirely responsible for determining the final action — that is, low-level actuation — given the observation of the environment state — i.e., the output of the perception system (e.g., camera, LiDAR, etc.). However, in such a system the task to be learned is extremely general and involves different levels of analysis and decision-making. The literature shows that this problem formulation can often lead to training difficulties and yield solutions that behave inadequately in real-world applications [2].

For this reason, this work pursues a more conservative approach, investigating a hybrid solution that integrates a learning-based component into a well-established framework of deterministic navigation and exploration algorithms, with the aim of extending their capabilities. To implement this type of approach, it was necessary to develop a system with a modular architecture in which several components interact with one another.

3.2 Modular Architecture

As introduced in Section 2.1, the autonomous exploration problem lends itself naturally to being decomposed into a series of sub-problems, such as mapping, planning, and decision-making.

The approach presented here follows this modular structure and integrates a learning-based module into it. The following paragraphs describe how the system has been conceived in terms of its components.

The sensing and **mapping** module assumes that an agent moving through an unknown environment is able to observe a portion of it within its perception range, constructing a partial representation known as a *belief map*.

A **frontier detector** module then analyses these partial maps in order to discriminate frontier regions, organize them into clusters, and compute their centroids, as described in Section 2.1.2.

At this point, the **decision-making** process — which would normally be entrusted to an exact heuristic model — is instead delegated to a **deep reinforcement learning** algorithm, whose role is to select which cluster will be set as the agent's next desired position.

Finally, a path planning algorithm is responsible for determining a collision-free path that enables the agent to reach the selected target point.

Figure 3.1 illustrates the overall architecture of the system, showing the relationships between its components and the role of the learning-based module.

The following sections describe the modelling and implementation of each component in greater detail.

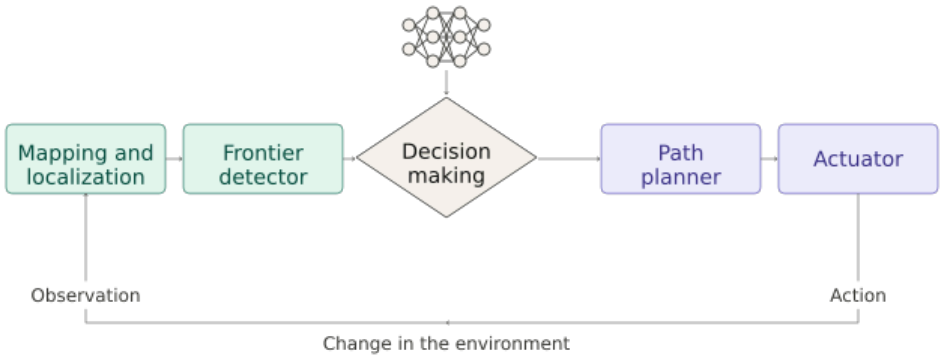


Figure 3.1: Modular control architecture of the proposed system. The agent is represented by a composition of modules. The Decision making module corresponds to the neural (leaning-based) component

Chapter 4

Agent-Environment System Simulator

As already mentioned, the fundamental framework of reinforcement learning methods is that of an Agent–Environment system, where the training process is carried out through a sequence of simulated trial-and-error experiences. To this end, a simulation environment has been developed.

4.1 2D Occupancy grid

The environment is modelled as a 2D occupancy grid, in which a finite spatial domain is discretized into a number of cells. Each cell can be either free or occupied by an obstacle. The domain is only partially observable by the agent, which progressively builds its own *belief grid*: a representation of the environment that distinguishes between known and unknown cells, in addition to the free/occupied classification.

As shown in Figure 4.1, this simulation setup is configurable in terms of grid size and obstacle probability.

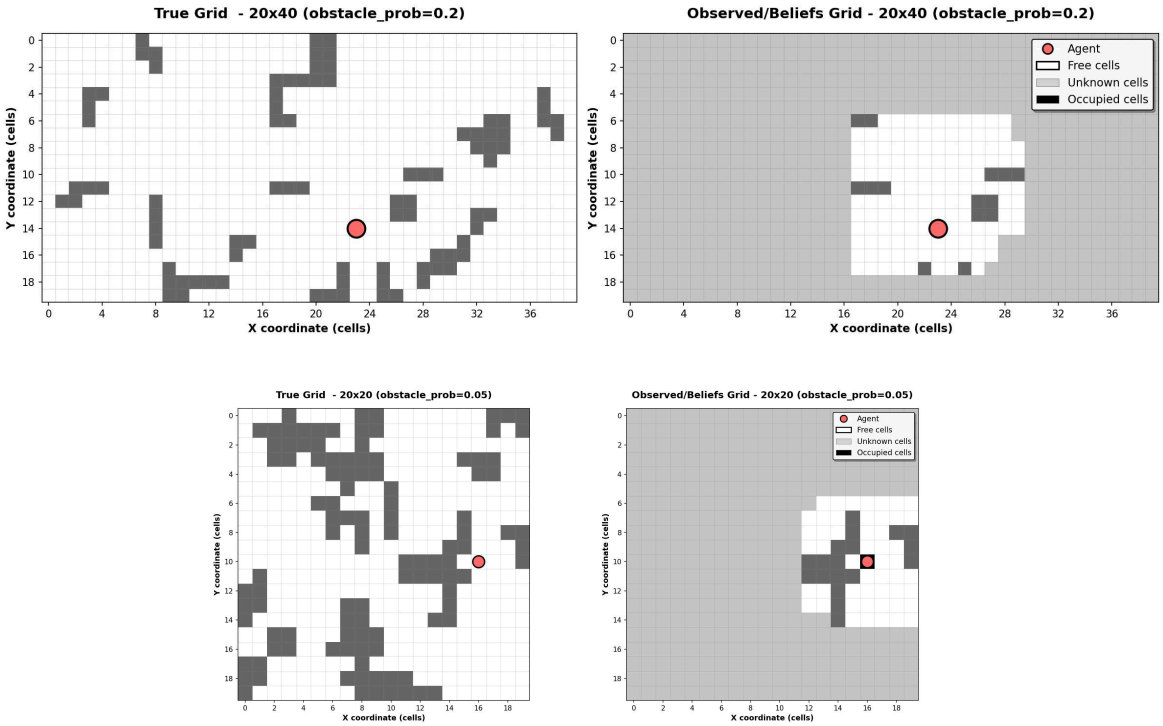
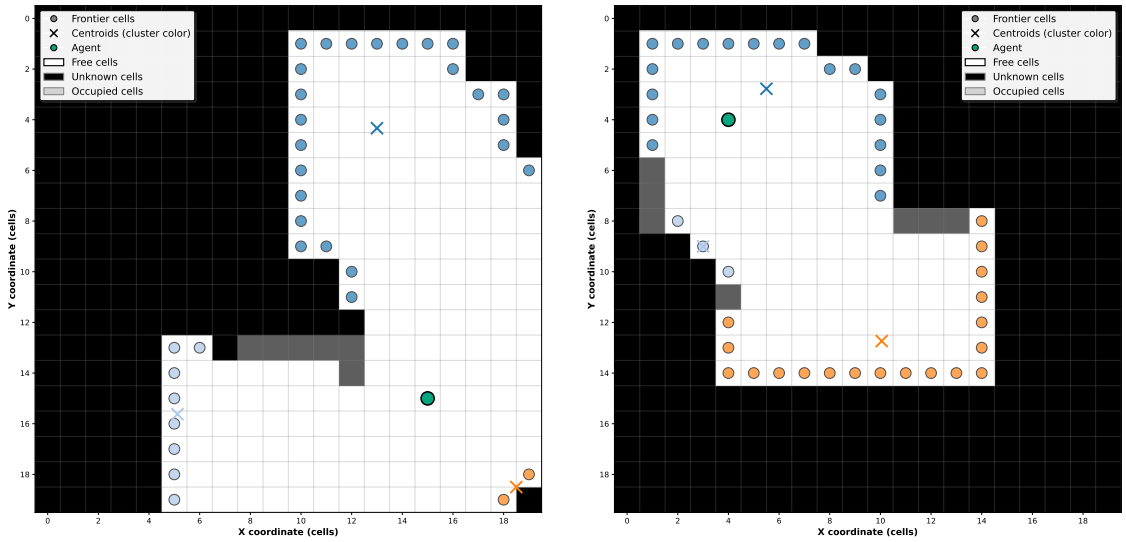


Figure 4.1: Examples of grid environments with different sizes and obstacle probabilities. For each configuration, the true grid (left) and the corresponding belief grid as observed by the agent (right) are shown.

4.2 Frontier Detection and Navigation

The exploration strategy adopted in this work relies on a frontier-based approach, in which the agent's decision-making is driven by the continuous tracking of the boundaries between explored and unexplored regions of the map. To support this, a frontier management algorithm has been integrated into the environment-agent system, operating as a pipeline of sequential processing stages every time the belief map is updated with new sensor information.

Frontier detection and clustering. A detection algorithm scans the entire belief map and identifies all frontier cells, defined as free cells adjacent to at least one unknown cell. These are then grouped by spatial proximity into clusters of contiguous cells, partitioning the frontier into a set of distinct regions of variable size and location. Each cluster represents a candidate exploration target: for each of them, the *centroid*, the distance from the agent, and an estimate of the *information gain* — approximated by the number of constituent cells — are computed. Graphical examples of the algorithm's output are shown in Figure 4.2.



(a)
b

(b)
b

Figure 4.2: Examples of frontier detection and clustering on partially explored maps. Frontier cells are shown as colored dots grouped by cluster, with centroids marked by crosses.

Cluster splitting. Since the agent's navigation target is selected among the cluster centroids, a well-known limitation of this approach — as noted by [6] — arises when

a centroid falls within or near the agent's current position. This situation is not only possible but becomes increasingly likely as cluster size grows. To mitigate it, a splitting mechanism is applied to clusters exceeding a maximum cell count: the cluster is recursively subdivided along the principal axes of its bounding box, yielding smaller and more spatially distributed sub-clusters. The effect of this decomposition can be appreciated in Figure 4.3.

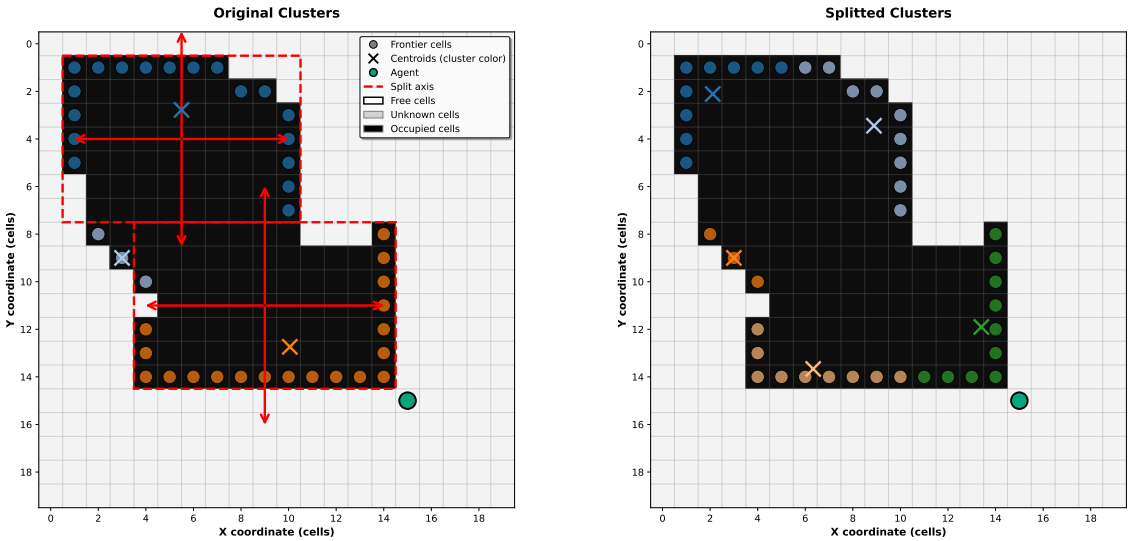


Figure 4.3: Cluster splitting along the bounding box axes. Left: original clusters with the split axes shown. Right: resulting sub-clusters after decomposition, with updated centroids.

Path planning and navigation. The frontier-based navigation pipeline operates as follows: at each decision step, the agent selects a target centroid from the set of detected clusters and delegates collision-free navigation to a path planning module. Given the occupancy grid representation of the environment, the A* algorithm [11] was adopted as the path planner of choice. It performs a heuristic search on the grid by expanding nodes in order of $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the start node and $h(n)$ is the Euclidean distance to the goal, guaranteeing an optimal and complete solution whenever one exists. An example of a computed path is shown in Figure 4.4.

4.3 Software Architecture

The simulator has been implemented as a hierarchy of Python classes, structured across multiple inheritance layers that progressively extend the capabilities of the

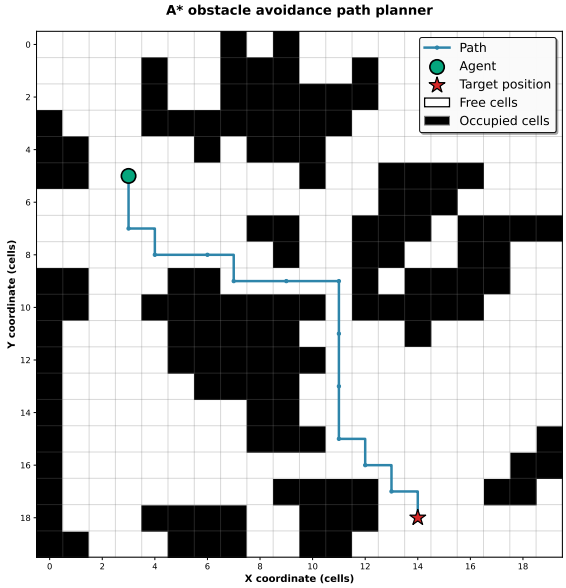


Figure 4.4: A path planned on the occupancy grid from the agent's current position to a target frontier centroid, navigating around occupied cells.*

system. As illustrated in Figure 4.5, each layer builds upon the previous one, adding frontier detection and navigation logic on top of the core grid-agent system.

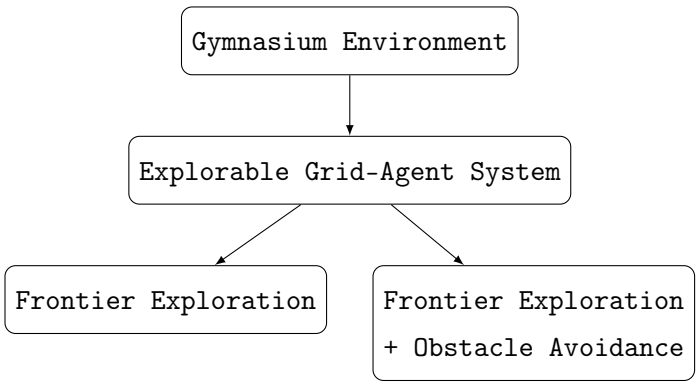


Figure 4.5: Class hierarchy of the simulation software.

4.3.1 The Gymnasium Framework

The simulator has been developed on top of Gymnasium, a widely adopted open-source framework for developing and comparing reinforcement learning environments. Gymnasium defines a standard interface that any environment must expose to interact with an RL training loop, ensuring compatibility with most existing RL libraries.

The entire class hierarchy inherits from the base `Environment` class provided by Gymnasium. All system dynamics are encapsulated in its standard methods: `step()`, `reset()`, and `get_obs()`.

The `step()` method is the core of the simulation: at each invocation it receives the action selected by the policy, advances the state of the environment, and returns the current observation, the reward accumulated during the step, and additional diagnostic information. A schematization of its logic is reported in Figure 4.6

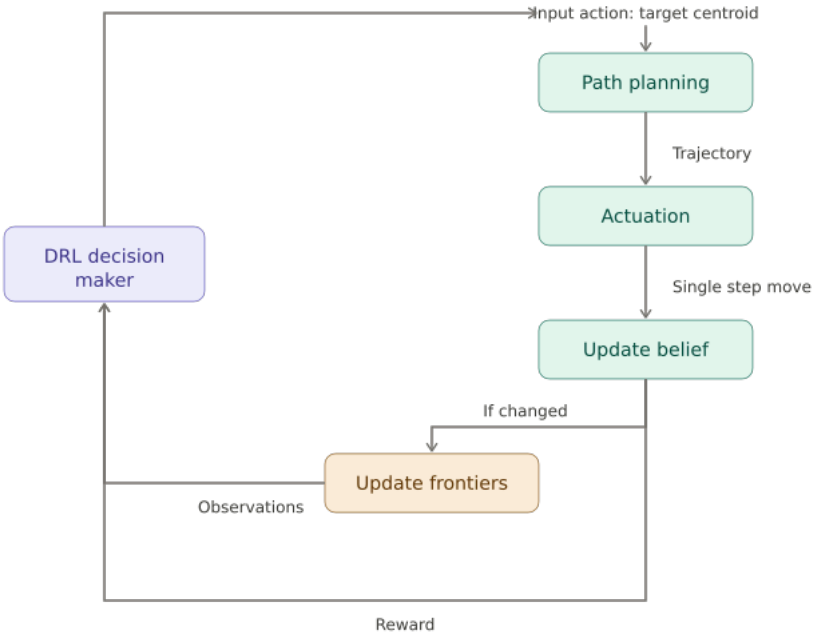


Figure 4.6: Block diagram of sequential information processing of the `step()` function dynamics.

Every Gymnasium environment must declare the structure of its observation space and action space. In the present case, the observations consist of arrays of information relative to the detected frontier clusters, while the action corresponds to the selection of which frontier cluster to navigate towards.

4.3.2 Observation Space

Several observation representations have been implemented, and the environment is designed to be highly configurable in this regard, so that different combinations

can be tested and compared during training. The configurable components are the centroid position encoding, the inclusion of the agent's own position, and the inclusion of an information gain estimate for each cluster. Table 4.1 summarizes all possible combinations.

Table 4.1: Configurable observation space representations.

Centroid Position	**Agent Position**	**Information Gain**
Absolute position		
Relative position	optional	optional
Distance		

Since the number of frontier clusters can vary from episode to episode, while the observation array must have a fixed size for a given model, a padding scheme has been adopted. Inactive slots in the observation vector are filled with a conventional null value, keeping the array dimension constant regardless of the number of clusters present at any given time. An example of a padded observation is shown in Table 4.2.

Table 4.2: Example of a padded observation vector with three active clusters and three padding slots.

Cluster 1	Cluster 2	Cluster 3	Pad	Pad	Pad
$[x_1, y_1]$	$[x_2, y_2]$	$[x_3, y_3]$	$[0, 0]$	$[0, 0]$	$[0, 0]$

Chapter 5

The DRL decision-making module

5.1 Problem Formulation as a Markov Decision Process

The autonomous exploration task is formulated as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{R}$ the reward function, and $\mathcal{T}$ the transition dynamics. At each decision step, the agent observes the current state of the environment and selects a frontier cluster to navigate towards, with the objective of maximizing the total explored area over the course of an episode.

State - Frontier Clusters Information

The state $s \in \mathcal{S}$ is represented by the observation vector described in Section 4.3.2, encoding the positions of the detected frontier clusters along with optional information on the agent's own position and the estimated information gain associated with each cluster.

Action - Frontier Clusters Selection

The action $a \in \mathcal{A}$ corresponds to the selection of a target frontier cluster. The action space is discrete, with a fixed maximum number of clusters $N_{\max}$ to keep the

space bounded. Selecting a cluster triggers the path planning and actuation pipeline described in Chapter 4, which navigates the agent towards the chosen target.

5.1.1 Reward Shaping

The design of the reward function is one of the most critical aspects of training the DRL decision-making module. An initial set of simpler reward formulations was explored before arriving at the two configurations presented here. Early attempts, based on sparse rewards tied solely to exploration progress, did not produce stable or meaningful learning behavior, often leading to degenerate or non-converging policies. The reward signal was progressively refined to provide denser feedback and to discourage specific undesirable behaviors, such as selecting padding slots or remaining stationary.

The two final reward formulations, referred to as *Reward 1* and *Reward 2*, share the same base structure and differ in two respects: the treatment of padding slot selection and the addition of a penalty for oscillating behavior in Reward 2. Table 5.1 summarizes the complete specification of both formulations.

Table 5.1: Comparison of the two reward formulations used in the experimental campaign. Entries marked "= R1" are identical to Reward 1.

Condition	**Reward 1**	**Reward 2**	**Episode end**	
			R1	**R2**
Action outcome				
Action selects padding slot	large negative	small negative	truncated	continues
No new cells explored	small negative	= R1	—	—
Some cells explored	intermediate positive	= R1	—	—
Episode termination				
Exploration $\geq$ threshold	large positive	= R1	terminated	
Max steps exceeded	large negative	= R1	truncated	
No progress for ≥ 20 steps	intermediate negative	= R1	truncated	
Additional signal — Reward 2 only				
Oscillating behavior detected	—	small negative	—	

The key difference between the two formulations lies in how padding slot selection is penalized. In Reward 1, selecting a padding slot incurs a large penalty and immediately truncates the episode. This strong signal was intended to discourage invalid

actions, but as discussed in Section 6.3, it inadvertently introduced a bias in the learned policy when combined with distance-sorted observations. In Reward 2, the penalty is reduced to a small negative value and the episode is allowed to continue, removing the truncation incentive that had distorted the policy. An additional small penalty for oscillating behavior was also introduced to discourage the agent from repeatedly cycling between the same frontier targets.

5.2 DQN Implementation and Training Setup

The DQN agent has been implemented using Stable Baselines3 (SB3), an open-source library providing reliable, well-tested implementations of standard deep reinforcement learning algorithms. SB3 was chosen for its clean interface, native compatibility with Gymnasium environments, and built-in support for TensorBoard logging, which was used throughout training to monitor the evolution of the reward, the loss, and the exploration rate.

The SB3 implementation of DQN corresponds to the vanilla formulation of the original algorithm [5]: it maintains an online Q-network and a periodically synchronized target network, and uses a uniform experience replay buffer to decorrelate training samples. No extensions such as Double-DQN, Dueling-DQN, or Prioritized Experience Replay are included. The policy is represented by a multi-layer perceptron (`MlpPolicy`), with ReLU activations and the Adam optimizer, both set to their default SB3 configuration.

Hyperparameters

The DQN algorithm exposes a number of hyperparameters that govern both the structure of the learning process and the exploration strategy. Table 5.2 reports the final configuration adopted after an initial tuning phase.

The *replay buffer* (`buffer_size`) stores past transitions and allows the agent to learn from decorrelated samples; its size represents a tradeoff between memory usage and sample diversity. The `batch_size` controls how many transitions are sampled at each gradient update, while `train_freq` determines how often these updates occur relative to environment steps.

The exploration schedule is governed by `exploration_fraction` and `exploration_final_eps`: the agent follows an ε-greedy policy in which ε decays linearly from 1 to `exploration_final_eps` over the first `exploration_fraction` fraction of total training steps, after which it remains constant. A relatively high `exploration_fraction` of 0.7 was chosen to ensure sufficient coverage of the observation space before the policy begins to consolidate.

The `target_update_interval` controls how frequently the target network is synchronized with the online network, a key mechanism for stabilizing training in DQN. The `learning_rate` determines the step size of the Adam optimizer used to update the network weights.

Table 5.2: Final DQN hyperparameter configuration. The total number of training time steps was varied across experiments and is not reported here.

Hyperparameter	**Value**
`policy`	`MlpPolicy`
`batch_size`	64
`buffer_size`	50 000
`learning_rate`	5×10^{-4}
`train_freq`	4
`target_update_interval`	500
`exploration_fraction`	0.7
`exploration_final_eps`	0.05

Experimental setup

Training experiments were organized along two main axes of variation: the observation space configuration and the reward function design. All combinations reported in Table 4.1 were initially evaluated, and a subset was subsequently discarded based on training performance. Similarly, several reward configurations were tested before converging on the formulation described in Section 6.1.2.

Each experimental run was monitored via TensorBoard, tracking the episode reward, the TD loss, and the ε schedule. These metrics guided the iterative refinement of both the observation representation and the reward signal.

Chapter 6

Experiments and Results

This chapter presents the experimental evaluation of the DRL-based exploration system. Section 6.1 describes the experimental setup, detailing the observation configurations, reward formulations, and training parameters explored throughout the work. Section 6.2 introduces the heuristic baselines used for comparison. Section 6.3 analyses the training dynamics of the selected configurations. Finally, Section 6.4 presents the evaluation results and a comparative analysis against the baselines.

The experimental campaign was conducted in three successive phases, summarized in Figure 6.1. In the first phase, the full set of observation configurations was evaluated under both reward formulations, with the goal of identifying which combination of observation representation and reward shape produced the most promising training behavior. Each reward formulation was treated independently, yielding a best-performing configuration for Reward 1 and one for Reward 2. In the second phase, hyperparameter optimization was carried out on these two selected configurations, varying the most influential DQN training parameters — learning rate, exploration fraction, batch size, buffer size, target update interval, and training frequency — to determine a shared optimal parameter set. In the third phase, the two best configurations, equipped with the optimal hyperparameters, were trained across multiple training budgets to assess the effect of training duration and to identify the models that would be carried forward to final evaluation. This structured process, while necessarily a rationalization of what was in practice a more iterative and exploratory

activity, reflects the overall logic that guided the experimental decisions reported in this chapter.

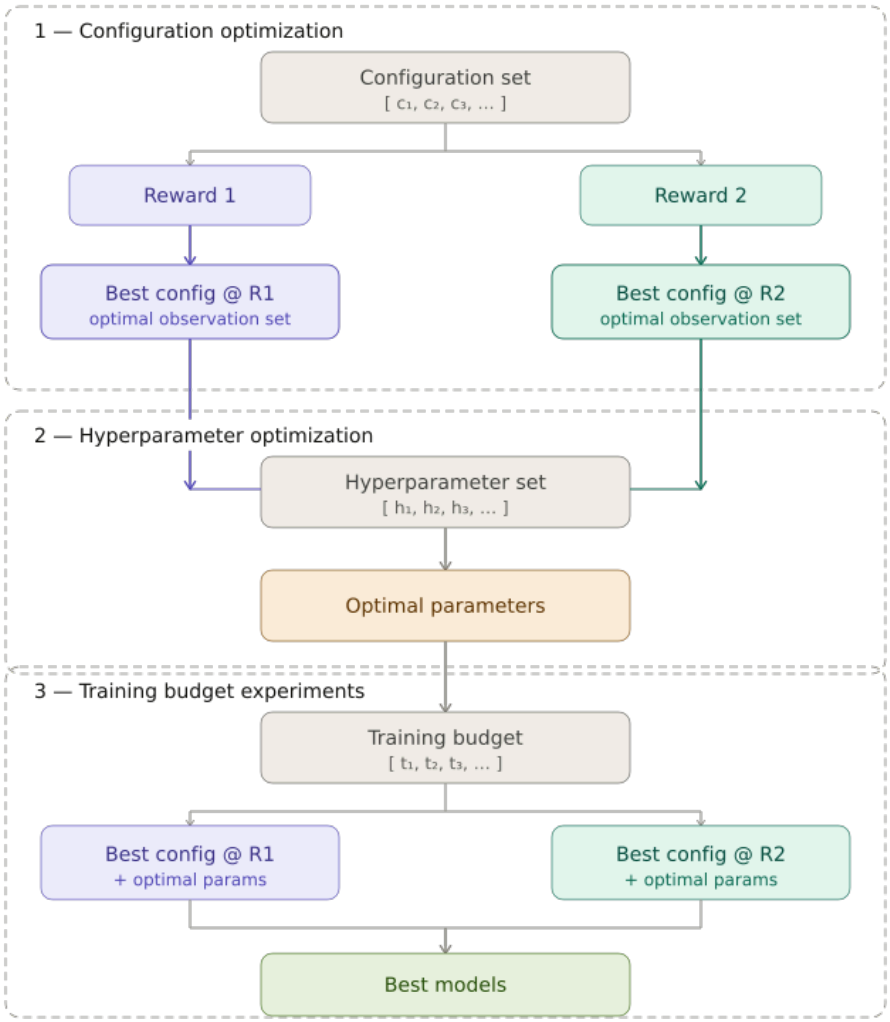


Figure 6.1: Overview of the experimental process followed in the present work

6.1 Experimental Setup

The experimental campaign was structured as a systematic exploration of the design space, varying three main dimensions: the observation representation provided to the DRL agent, the reward formulation, and the training duration. All experiments were conducted in the custom Gymnasium environment described in Chapter 3, using the DQN implementation provided by Stable-Baselines3. Unless stated otherwise, the default environment geometry consists of a 20×20 grid with a perception range of 3 cells.

Table 6.1: Observation space configurations evaluated during the experimental campaign. Configurations marked with ⋆ are discussed in detail in Sections 6.3 and 6.4.

#	Observation components	Sorting	Environment	Notes
Absolute centroid positions				
1	Abs. centroids + agent pos.	unsorted	20×20, r=3	⋆ main result
2	Abs. centroids + agent pos. + info gain	sorted	20×20, r=3	
3	Abs. centroids + agent pos. + info gain	unsorted	20×20, r=3	
Relative centroid positions				
4	Rel. centroids only	sorted	20×20, r=3	⋆ degenerate policy
5	Rel. centroids only	unsorted	20×20, r=3	⋆ policy collapse
6	Rel. centroids + agent pos.	sorted	20×20, r=3	
7	Rel. centroids + agent pos.	unsorted	20×20, r=3	
8	Rel. centroids + agent pos. + info gain	sorted	20×20, r=3	
9	Rel. centroids + agent pos. + info gain	unsorted	20×20, r=3	
Information gain only				
10	Info gain only	sorted	20×20, r=3	
11	Info gain only	unsorted	20×20, r=3	
Robustness verification				
12	Abs. centroids + agent pos.	unsorted	30×30, r=5	⋆ robustness

6.1.1 Observation Space Configurations

A central question in the design of the DRL agent is what information should be included in the observation vector. Three types of centroid representation were considered: *absolute* positions, expressed as (x, y) coordinates in the grid frame; *relative* positions, expressed as offsets from the agent's current position; and *information gain* only, using the number of unknown cells associated with each cluster as the sole observation. Each representation was tested with and without the inclusion of the agent's position and the per-cluster information gain as additional observation components. All configurations were evaluated both with centroids sorted in increasing order of distance from the agent and in unsorted order. Table 6.1 summarizes the full set of observation configurations tested.

6.1.2 Reward Configuration

The design of the reward function is one of the most critical aspects of training the DRL decision-making module. An initial set of simpler reward formulations was explored before arriving at the two configurations presented here. Early attempts,

based on sparse rewards tied solely to exploration progress, did not produce stable or meaningful learning behavior, often leading to degenerate or non-converging policies. The reward signal was progressively refined to provide denser feedback and to discourage specific undesirable behaviors, such as selecting padding slots or remaining stationary.

The two final reward formulations, referred to as *Reward 1* and *Reward 2*, share the same base structure and differ in two respects: the treatment of padding slot selection and the addition of a penalty for oscillating behavior in Reward 2. Table 6.2 summarizes the complete specification of both formulations.

Table 6.2: Comparison of the two reward formulations used in the experimental campaign. Entries marked "= R1" are identical to Reward 1.

Condition	**Reward 1**	**Reward 2**	**Episode end**
Action outcome			
Action selects padding slot	-1.0	-0.1	R1: truncated R2: continues
No new cells explored	-0.05	= R1	—
Some cells explored	$+0.05 \times \Delta_{\text{cells}}$	= R1	—
Episode termination			
Exploration $\geq$ threshold	$+1.0$	= R1	terminated
Max steps exceeded	-1.0	= R1	truncated
No progress for ≥ 20 steps	-0.5	= R1	truncated
Additional signal — Reward 2 only			
Oscillating behavior detected	—	-0.05	—

The key difference between the two formulations lies in how padding slot selection is penalized. In Reward 1, selecting a padding slot incurs a large penalty of -1.0 and immediately truncates the episode. This strong signal was intended to discourage invalid actions, but as discussed in Section 6.3, it inadvertently introduced a bias in the learned policy when combined with distance-sorted observations. In Reward 2, the penalty is reduced to -0.1 and the episode is allowed to continue, removing the truncation incentive that had distorted the policy. An additional small penalty for oscillating behavior was also introduced to discourage the agent from repeatedly cycling between the same frontier targets.

6.1.3 Hyperparameter Optimization

The performance of the DRL decision-making module is sensitive to the choice of training hyperparameters. A systematic search was conducted over the most influential parameters of the DQN algorithm, varying each over a range of values while keeping the remaining parameters fixed. The primary criteria for evaluation were training stability, convergence speed, and the mean total reward achieved at the end of training. Table 6.3 reports the range explored for each parameter and the optimal value selected. The resulting configuration, used as the reference for all experiments reported in this chapter, is summarized in Table 6.4.

Among the parameters explored, the learning rate had the most significant impact on training stability. High values led to loss explosion and divergence, while excessively low values resulted in slow or incomplete convergence. A value of 5×10^{-5} was identified as the most stable choice across training runs. The exploration fraction was set to 0.7, meaning that ε is annealed from 1.0 to its final value over 70% of the total training steps, allowing the agent sufficient time to explore the environment before committing to exploitation.

Table 6.3: Hyperparameter search ranges and optimal values selected for the DRL decision-making module.

Parameter	**Range tested**	**Optimal value**
Learning rate	1×10^{-3} – 1×10^{-5}	5×10^{-5}
Exploration fraction	0.1 – 0.9	0.7
Batch size	32 – 128	64
Buffer size	10,000 – 100,000	50,000
Target update interval	200 – 500	500
Training frequency	1 – 8	4

6.2 Heuristic Baselines

Before evaluating the DRL decision-making module, it is necessary to establish a reference performance level against which the learned policy can be meaningfully compared. To this end, two classical heuristic strategies were implemented and evaluated under the same conditions used for training and testing the DRL decision-making module. Computing the mean total reward achieved by these heuristics

Table 6.4: Final hyperparameter configuration used across all reported experiments.

Parameter	**Value**	**Notes**
Learning rate	5×10^{-5}	most stable long-term
Exploration fraction	0.7	70% of training steps
Final epsilon	0.05	
Batch size	64	
Buffer size	50,000	
Target update interval	500	steps
Training frequency	4	steps between updates
Policy (centroids only)	`MlpPolicy`	
Policy (centroids + agent pos.)	`MultiInputPolicy`	

provides a concrete benchmark: if the learned policy reaches or approaches this level, it can be considered competitive with the best simple analytical strategies available for the task.

The two heuristics considered are the *greedy distance* policy, which at each step selects the frontier cluster whose centroid is closest to the agent's current position, and the *greedy information gain* policy, which selects the cluster associated with the largest number of unknown cells. Both strategies are deterministic and require no training.

Each heuristic was evaluated across four environment configurations, combining two grid geometries — 20×20 with perception range $r = 3$, and 30×30 with perception range $r = 5$ — with two obstacle layout modes. In the *static* mode the obstacle configuration is fixed across all episodes, while in the *dynamic* mode the grid is regenerated at the start of each episode with a different random obstacle layout. The dynamic setting is more challenging, as the agent cannot rely on any episode-to-episode consistency in the environment structure. Results are reported as mean total reward and standard deviation over a set of evaluation episodes, and are summarized in Table 6.5.

Several observations can be drawn from these results. The greedy distance heuristic consistently outperforms greedy information gain across all configurations, suggest-

Table 6.5: Mean total reward of the two heuristic baselines across all environment configurations. Results are reported as mean ± standard deviation over 30 evaluation episodes.

Environment	**Heuristic**	**Mean reward**	**Std**
20×20 grid, perception range $r = 3$			
Static	Greedy distance	16.48	1.78
Static	Greedy info gain	15.01	1.12
Dynamic	Greedy distance	13.89	3.29
Dynamic	Greedy info gain	13.19	2.67
30×30 grid, perception range $r = 5$			
Static	Greedy distance	36.39	1.27
Static	Greedy info gain	34.32	2.58
Dynamic	Greedy distance	35.15	3.57
Dynamic	Greedy info gain	31.70	4.73

ing that minimizing travel distance is a more effective criterion than maximizing immediate information gain in the environments considered. The dynamic setting produces lower mean rewards and higher variance in both geometries, as expected given the increased variability of the environment from episode to episode. The larger 30×30 grid yields substantially higher absolute reward values, reflecting the greater number of frontier clusters available and the correspondingly larger exploration horizon.

These results serve as the reference baseline for the evaluation of the DRL decision-making module presented in Section 6.4.

6.3 Training Analysis

This section presents a diagnostic analysis of the training dynamics observed across the experimental campaign. Rather than reporting all configurations exhaustively, the discussion focuses on two runs that were most informative for understanding the behavior of the DRL decision-making module: the configuration that produced a degenerate policy (relative centroid positions, sorted, configuration 4 in Table 6.1) and the configuration that yielded the first genuinely non-degenerate learned policy

(absolute centroid positions with agent position, unsorted, configuration 1).

6.3.1 Degenerate Policy: Reward Hacking via Observation Sorting

The training run for configuration 4 exhibits a pattern that initially appears promising. As shown in Figure 6.2, the episode reward increases steadily and converges to a stable level comparable to that of the greedy distance heuristic. Taken in isolation, this would suggest successful learning. However, inspection of the training loss curve reveals a fundamentally different picture: as shown in Figure 6.3, the TD loss increases monotonically throughout training, reaching values on the order of 10^6 by the end of the run. A diverging loss of this magnitude is inconsistent with genuine Q-value learning and indicates that the network is not converging to a meaningful value function.

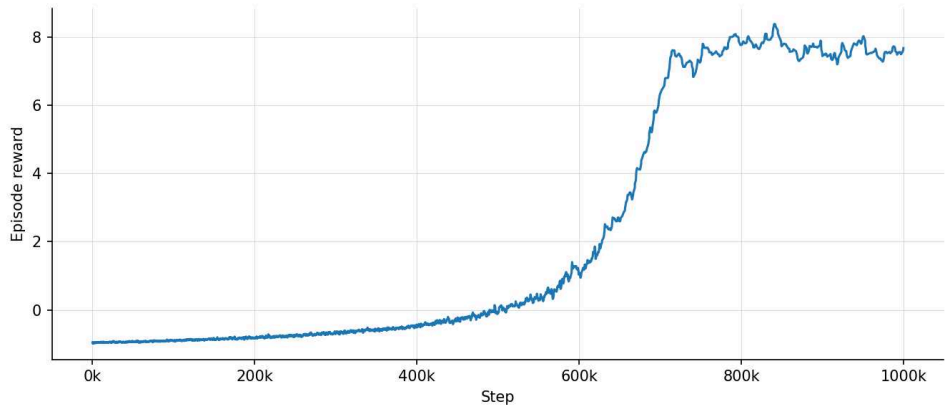


Figure 6.2: Episode reward during training for configuration 4 (relative centroid positions, sorted). The reward converges to a level comparable to the greedy baseline, suggesting a well-performing policy.

The explanation for this discrepancy lies in the interaction between the observation sorting and the truncation penalty for padding slot selection introduced in Reward 1. Since frontier centroids were sorted in increasing order of distance from the agent, the first element of the observation vector always corresponded to the nearest cluster — the same target selected by the greedy distance heuristic. The strong truncation penalty for selecting padding slots, which were placed after the actual observations, created an additional incentive to always select the first action. The network therefore learned a degenerate policy of always choosing action 0, which coincidentally

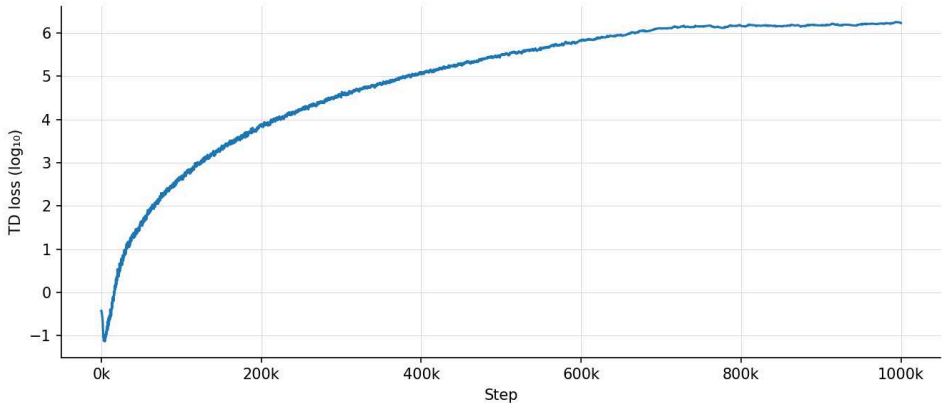


Figure 6.3: TD loss (log_{10} scale) during training for configuration 4. The monotonic divergence to values on the order of 10^6 is inconsistent with genuine Q-value convergence and indicates a degenerate training dynamic.

replicated the greedy distance strategy and achieved competitive reward without ever reasoning about the content of the observation.

This diagnosis was confirmed by removing the distance sorting from the same configuration. Without the ordering that made action 0 always correspond to the nearest frontier, the policy collapsed immediately, producing near-zero reward. This result ruled out any alternative explanation and established reward hacking as the cause of the apparently good performance.

6.3.2 Non-Degenerate Policy: Absolute Positions Without Truncation

The key modifications introduced in configuration 1 — switching to absolute centroid positions, removing the distance sorting, and replacing the truncation penalty for padding with the softer formulation of Reward 2 — were sufficient to break the reward hacking dynamic. For the first time across all configurations tested, the DRL decision-making module learned a policy that selects genuinely different actions depending on the state of the environment.

Figure 6.4 shows the episode reward curve for this configuration over 10^6 training steps. The reward starts at approximately -2.5 and increases steadily, crossing zero around 200k steps and converging to a stable range of approximately 12.5–13.5

between 700k and 1M steps. The shape of the curve — a smooth, monotonic increase followed by a plateau — is characteristic of a well-behaved learning process.

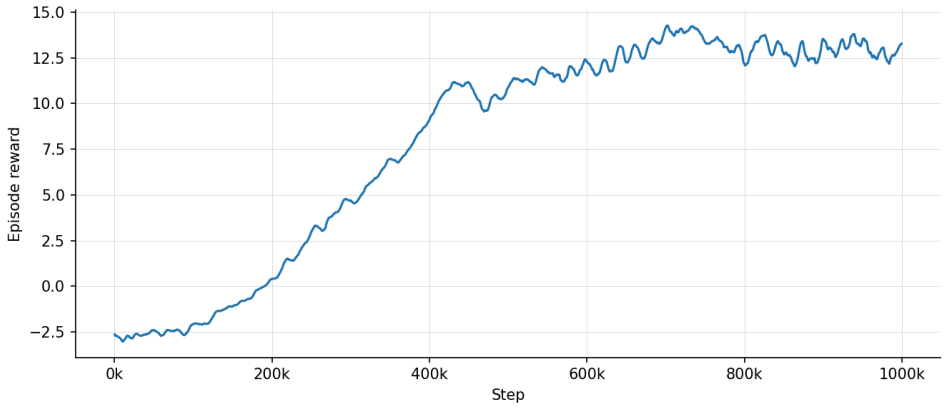


Figure 6.4: Episode reward during training for configuration 1 (absolute centroid positions with agent position, unsorted, 10^6 *steps). The reward increases steadily and converges to a stable range of approximately* 12.5–13.5.

The corresponding TD loss curve is shown in Figure 6.5. After an initial sharp drop in the first few thousand steps, the loss settles into an oscillatory regime with values between approximately 10^{-2} and $10^{-0.75}$, corresponding to a mean of roughly 0.1 in linear scale. The amplitude of the oscillations increases gradually from around 400k steps onward, peaking near 750k steps, which coincides with the final stages of the ε decay and the transition to a predominantly exploitative policy. This behavior is consistent with the reduced diversity of transitions sampled from the replay buffer as exploration decreases. Crucially, the loss remains bounded throughout training — in stark contrast to the diverging loss observed in the degenerate configuration — confirming that the network is engaged in genuine value function approximation.

Training runs of 5×10^5 and 2×10^6 steps were also evaluated for this configuration. Both produced slightly lower evaluation rewards than the 10^6 step run, suggesting that 10^6 steps represents the optimal training duration for this setup: the shorter run does not allow the policy to fully converge, while the longer run shows mild signs of performance degradation, likely due to overfitting on a narrow distribution of replay buffer experiences accumulated during the exploitation phase.

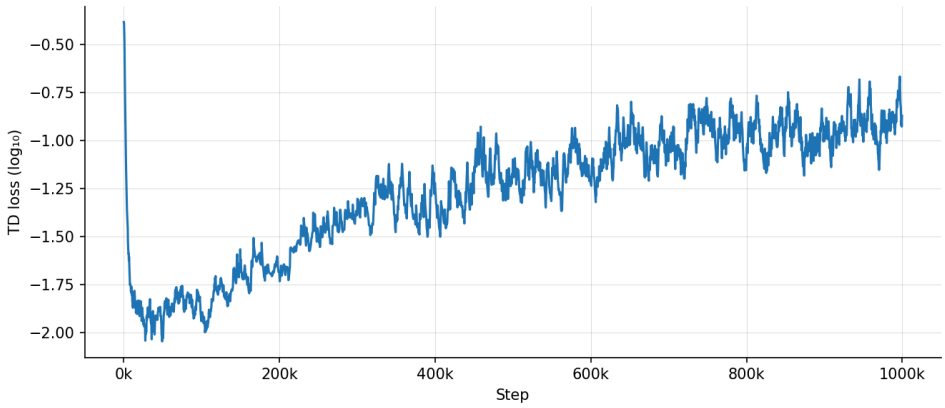


Figure 6.5: TD loss ($\log_{10}$ scale) during training for configuration 1. The loss remains bounded throughout, oscillating between 10^{-2} and $10^{-0.75}$ with increasing amplitude toward the end of the ε decay phase.

6.4 Evaluation and Comparison

The trained models were evaluated using `evaluate_policy()` `StableBaselines3` function over 30 episodes on a static environment instance, with $\varepsilon = 0$ to ensure purely exploitative behavior. Results are compared against the heuristic baselines reported in Section 6.2. Table 6.6 summarizes the evaluation metrics for all configurations discussed in this chapter.

Table 6.6: Evaluation results for the selected configurations compared against the heuristic baselines. All results refer to the static environment. Mean reward and standard deviation are computed over 30 evaluation episodes.

Configuration	**Policy type**	**Mean reward**	**Std**	**Terminations**
Heuristic baselines — 20×20, $r = 3$				
Greedy distance	heuristic	16.48	1.78	—
Greedy info gain	heuristic	15.01	1.12	—
DRL configurations — 20×20, $r = 3$				
Relative centroids, sorted (config. 4)	degenerate	16.91	0.57	—
Info gain only, sorted	degenerate	16.66	0.75	28/30
Abs. centroids + agent pos., unsorted (config. 1)	non-degenerate	15.55	2.45	25/30
Robustness verification — 30×30, $r = 5$				
Greedy distance	heuristic	36.39	1.27	—
Greedy info gain	heuristic	34.32	2.58	—
Abs. centroids + agent pos., unsorted (config. 1)	non-degenerate	31.46	4.75	14/30

6.4.1 Degenerate Configurations

The two degenerate configurations achieve mean rewards of 16.91 ± 0.57 and 16.66 ± 0.75 respectively, both nominally comparable to the greedy distance baseline. However, as established in Section 6.3, these results are not the product of genuine learned behavior. The action distribution of configuration 4 is overwhelmingly concentrated on action 0, which under distance sorting always corresponds to the nearest frontier — an exact replication of the greedy distance heuristic. The low standard deviation of 0.57 reflects the rigidity of this degenerate policy rather than robustness.

The same pathology is present in the information gain only configuration, where 95.1% of actions select action 0. In this case, the observation vector contains only per-cluster information gain values sorted by distance, making action 0 a consistently reliable choice regardless of the actual gain distribution. The learning of the same policy despite the different content of the observations is the proof that the result is again an artifact of the sorting and the reward structure rather than a learned strategy.

6.4.2 Non-Degenerate Policy: Main Result

Configuration 1 (absolute centroid positions with agent position, unsorted) is the only configuration that produced a genuinely non-degenerate learned policy with considerable reward performances. The action distribution across the 30 evaluation episodes covers all available actions, with action 0 selected in approximately 50.6% of steps, followed by actions 1 (25.4%), 2 (21.2%), and 3 (2.8%). The mean action of 0.76 and standard deviation of 0.88 confirm that the policy is responsive to the observation rather than fixed on a single output.

The model achieves a mean total reward of 15.55 ± 2.45 over 30 episodes, with 25 out of 30 episodes reaching the exploration threshold and terminating successfully. This result is slightly below the greedy distance baseline of 16.48 ± 1.78, but above the greedy information gain baseline of 15.01 ± 1.12. The higher standard deviation of 2.45 compared to the heuristics reflects the variability inherent in a stochastic learned policy, but also suggests that the model has not yet fully converged to a consistent strategy.

It is worth noting that the comparison with the heuristic baselines is not entirely symmetric. The heuristics are deterministic and operate with complete knowledge of the distance or information gain metric at every step, while the DRL decision-making module must infer an appropriate action from a fixed observation vector without direct access to either metric. That the learned policy approaches the greedy distance baseline under these conditions is a meaningful result.

6.4.3 Robustness Verification

To assess whether the learned policy generalizes beyond the training geometry, configuration 1 was also trained and evaluated on the larger 30×30 grid with perception range $r = 5$. The model achieves a mean total reward of 31.46 ± 4.75, compared to the greedy distance baseline of 36.39 ± 1.27 and greedy information gain of 34.32 ± 2.58.

The gap between the DRL decision-making module and the heuristics is wider on the larger grid than on the 20×20 environment, and the termination rate drops to 14 out of 30 episodes. This is consistent with the increased complexity of the larger environment: a greater number of frontier clusters are available at each step, the action space is correspondingly larger, and the exploration task requires a longer sequence of decisions. Nonetheless, the model maintains a qualitatively similar behavior to the smaller grid case, confirming that the learned policy is not entirely specific to the training geometry and scales to at least moderately larger environments.

The higher standard deviation of 4.75 compared to 2.45 on the 20×20 grid further reflects this increased difficulty, and suggests that additional training steps or a more expressive observation representation may be required to close the gap with the heuristic baselines on larger environments.

Chapter 7

Conclusions

This work investigated the applicability of Deep Reinforcement Learning to the task of autonomous exploration, with a specific focus on the integration of a DRL decision-making module within the framework of frontier-based exploration. Rather than replacing the full control pipeline with an end-to-end learned system, a modular architecture was adopted in which the path planning and mapping components rely on established exact methods, while the high-level decision of which frontier cluster to navigate toward is delegated to a trained DQN model. This separation allowed the complexity of the learning problem to be substantially reduced, and made the system more interpretable and easier to diagnose.

A central contribution of this work is the development of a flexible simulation framework specifically designed for studying and experimenting with DRL-based approaches to frontier exploration. The framework, built on Gymnasium and Stable-Baselines3, supports configurable grid geometries, obstacle densities, perception ranges, and observation representations, and was validated through systematic experimentation. It provides a reproducible and extensible testbed for future investigations in this direction.

The experimental campaign confirmed that integrating a DRL decision-making module within a frontier-based exploration system is feasible. A key finding of the work was the identification and diagnosis of a reward hacking phenomenon that caused

several configurations to learn degenerate policies: by combining distance-sorted observations with a strong truncation penalty for padding slot selection, the agent learned to always select action 0 — a behavior that coincidentally replicated the greedy distance heuristic without engaging in any genuine reasoning about the observation. This pathology was systematically diagnosed and resolved by removing the distance sorting and replacing the truncation penalty with a softer formulation, which allowed a genuinely non-degenerate policy to emerge for the first time.

The best-performing configuration — absolute centroid positions with agent position, unsorted observations, and 10^6 training steps — achieved a mean total reward of 15.55 ± 2.45 on the 20×20 static environment, compared to 16.48 ± 1.78 for the greedy distance heuristic and 15.01 ± 1.12 for the greedy information gain heuristic. The learned policy selects actions across the full range of available frontier clusters, demonstrating that it responds to the content of the observation rather than defaulting to a fixed output. A robustness verification on the larger 30×30 grid confirmed that the policy generalizes to a different environment geometry, achieving 31.46 ± 4.75 against the greedy distance baseline of 36.39 ± 1.27.

Several limitations of the current work should be acknowledged. The heuristic baselines remain more performant and considerably more flexible across environment configurations: they require no training, are deterministic, and degrade gracefully on larger or structurally different environments. The trained DRL decision-making module, by contrast, shows a widening gap with the heuristics on the larger grid and a higher variance in performance, reflecting the difficulty of generalizing a learned policy beyond the training conditions. Furthermore, the simulation environment is deliberately simplified — a two-dimensional grid with static obstacle layouts, idealized perception, and discrete four-directional motion — and the model was trained exclusively on static environments, limiting the validity of the conclusions to this constrained setting.

Several directions for future work emerge naturally from these findings. Training and evaluating on dynamic environments, in which obstacle layouts vary across episodes, would be a direct and important extension, as it would test whether the DRL decision-making module can develop policies robust to environment variability. Within the same vector-observation framework, richer information could be incorpo-

rated into the observation space — for instance, weighting the reward signal by the informational content of the explored regions, so as to incentivize the agent to prioritize areas likely to contain obstacles or structurally complex features. Augmenting the observation with a memory component, such as frame stacking or a recurrent network architecture, could allow the agent to maintain a more complete picture of the exploration history and develop more refined long-horizon strategies.

On the system side, moving toward a less simplified environment model — one that more closely reflects real-world conditions, including continuous state spaces, noisy sensing, and three-dimensional geometry — would be a necessary step toward practical applicability. Finally, investigating DRL in a more complex decision-making framework, such as an end-to-end architecture in which the agent operates directly on a partial map representation rather than a pre-processed vector of frontier observations, would allow a more direct comparison with the broader literature on learning-based autonomous exploration and could unlock richer representational capacity for the learned policy.

Ringraziamenti

Intraprendere e portare a termine questo percorso di studio in questo momento della mia vita, è stato tutto meno che una passeggiata. E' stata un'esperienza intensa, ricca di sfide e gratificazioni che mi ha permesso di conoscere meglio tanto le mie risorse personali, quanto i miei limiti. Se sono riuscito a portarla a termine, raccogliendo anche una certa dose di soddisfazione, lo devo sicuramente in buona misura alle persone da cui ho la fortuna di essere circondato, a cui va tutta la mia gratitudine.

Il primo pensiero va alla mia famiglia, a cui dedico questo risultato, per il coinvolgimento e la generosità che hanno dimostrato nel supportarmi. Perchè so che la loro felicità a riguardo non è eguagliata da quella di nessun altro.

Un garnde ringraziamento va al mio relatore, il Professor Lorenzo Sabbatini, per la fiducia dimostrata nei miei confronti e per la preziosa opportunità di svolgere il lavoro che oggi presento. Vorrei ringraziare inoltre i miei correlatori Mattia Catellani e Mattia Mantovani, che hanno messo a disposizione senza riserve la loro competenza e la loro intelligenza nell'accompagnarmi, per il rispetto e l'onestà con cui hanno condiviso con me pensieri ed opinioni.

Vorrei ringraziare inoltre i compagni e le compagne del DAE, che ho avuto la fortuna di conoscere ed apprezzare in questi anni. In particolare a quelli e quelle con cui ho condiviso progetti e sfide, che ho avuto l'opportunità di conoscere più da vicino, rivelandosi ogni volta persone belle e interessanti. In particolare un pensiero affettuoso va a quella cerchia con la quale le sedute al Pub, dopo essere stata la sede della stesura del progetto di Azionamenti Elettrici, sono diventate un rituale

ricorrente.

Infine, un ringraziamento speciale va alle mie amicizie, risorsa che ho scoperto essere, anche in questi anni, di fondamentale importanza. Che mi hanno motivato e incoraggiato, mostrando stima ed interesse per il mio percorso, anche, e soprattutto, quando da parte mia questa vacillava. A chi ha avuto la pazienza di tollerare i miei ritiri dalla vita sociale pre esame, e chi non l'ha avuta convincendomi ad uscire. A chi, tra questi, dopo aver per sbaglio assistito a un video degli scarsi risultati dei miei esperimenti, hanno preso a cuore l'apprendimento dell'agente autonomo soggetto della mia tesi, e con cui ho potuto in ultimo condividere: "Ha imparato!!".

9 Aprile, Paolo Spallanzani.

Bibliography

[1] **Sebastian Thrun, Wolfram Burgard, and Dieter Fox**. *Probabilistic Robotics*. MIT Press, 2005.

[2] **Chuqi Wang et al.** "Multi-Robot System for Cooperative Exploration in Unknown Environments: A Survey". In: *arXiv preprint arXiv:2503.07278* (2025).

[3] **Brian Yamauchi**. "A Frontier-Based Approach for Autonomous Exploration". In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. 1997, pp. 146–151.

[4] **M. Juliá, A. Gil, and O. Reinoso**. "A Comparison of Path Planning Strategies for Autonomous Exploration and Mapping of Unknown Environments". In: *Autonomous Robots* 33.4 (2012), pp. 427–444.

[5] **Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al.** "Human-level Control through Deep Reinforcement Learning". In: *Nature* 518 (2015), pp. 529–533.

[6] **Haoran Li, Qichao Zhang, and Dongbin Zhao**. "Deep Reinforcement Learning-based Automatic Exploration for Navigation in Unknown Environments". In: *arXiv preprint arXiv:2007.11808* (2020).

[7] **Boyu Zhou et al.** "FUEL: Fast UAV Exploration using Incremental Frontier Structure and Hierarchical Planning". In: *IEEE Robotics and Automation Letters* (2020).

[8] **H. H. González-Baños and J.-C. Latombe**. "Navigation Strategies for Exploring Indoor Environments". In: *The International Journal of Robotics Research* 21.10–11 (2002), pp. 829–848.

[9] **Richard S. Sutton and Andrew G. Barto**. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[10] **Christopher J. C. H. Watkins and Peter Dayan**. "Q-learning". In: *Machine Learning* 8 (1992), pp. 279–292.

[11] **Peter E. Hart, Nils J. Nilsson, and Bertram Raphael**. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.