Treball de Fi de Màster

Màster Universitari en Enginyeria Industrial (MUEI)

Leveraging Ground Penetrating Radar with Deep Reinforcement Learning for Subsurface-Informed Navigation in Autonomous Agricultural Robotics

MEMÒRIA

24 de juny de 2025

Autor: Riccardo Rettore

Director: Alba Pérez Gracia

Convocatòria: Juny 2025

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Resumen

El Reinforcement Learning (RL), impulsado por los avances en el Deep Learning, está logrando éxitos notables en resolver problemas complejos de toma de decisiones para una amplia variedad de dominios, como la robótica, la conducción autónoma y la modelación de lenguaje a gran escala. Estos sistemas llevan a cabo razonamientos de alto nivel, coordinación percepción-acción y planificación a largo plazo a partir de datos sensoriales sin procesar.

Inicialmente celebrados por dominar juegos como Go y StarCraft, el RL ha evolucionado hacia aplicaciones en el mundo real, siendo la robótica uno de los dominios más prometedores. Avances recientes de organizaciones como NVIDIA, Tesla y Google DeepMind han demostrado el potencial de combinar Aprendizaje por Refuerzo Profundo con simulación a gran escala y datos del mundo real para entrenar agentes capaces de realizar tareas complejas, desde manipulación diestra hasta locomoción ágil en entornos no estructurados.

Entre los dominios emergentes, la agricultura de precisión representa una oportunidad prometedora para los sistemas robóticos inteligentes. A pesar de su potencial, el Aprendizaje por Refuerzo Profundo ha tenido una aplicación limitada. Sin embargo, los sistemas inteligentes y adaptativos en agricultura se perfilan como necesarios para resolver problemas tales como la escasez de agua, la seguridad alimentaria y la sostenibilidad, a través del monitoreo y gestión de las condiciones del suelo y de los cultivos.

Esta tesis propone nuevos marcos de aprendizaje por refuerzo para la navegación autónoma en entornos agrícolas, considerando las condiciones subterráneas del terreno. Específicamente, se emplea el radar de subsuelo, o georradar (GPR) para detectar parámetros del subsuelo con los cuales tomar decisiones sobre el entorno. Este sensor se selecciona por su naturaleza no destructiva y su capacidad para revelar estructuras subterráneas, como capas del suelo y distribución de humedad, las cuales son importantes para la agricultura de precisión. El proyecto incluye la adquisición y procesamiento de datos reales de GPR, que luego se comprimen en representaciones latentes estructuradas utilizando autoencoders convolucionales. Estas codificaciones compactas sirven como entradas de observación para los agentes de RL, que operan en entornos de simulación construidos sobre el marco Gymnasium. Se exploran tanto espacios de acción discretos como continuos bajo condiciones ambientales variables. Los agentes se entrenan para una navegación dirigida a objetivos mientras se adaptan a condiciones subterráneas cambiantes. El proceso de entrenamiento utiliza algoritmos tales como la Optimización Proximal de Políticas y el Actor-Crítico Suave, lo que permite que los agentes aprendan normas efectivas en escenarios tanto discretos como continuos.

Los resultados experimentales demuestran que los agentes aprenden normas efectivas que aprovechan la información subterránea para una navegación informada. Este trabajo proporciona una integración novedosa entre aprendizaje por refuerzo profundo y sensores geofísicos para robótica agrícola, ofreciendo soluciones escalables para el monitoreo autónomo del suelo y la navegación sensible al subsuelo en entornos agrícolas. Al vincular la sensorización basada en GPR con RL profundo, este proyecto aporta una base para sistemas agrícolas de próxima generación capaces de exploración y toma de decisiones ambientales, no destructivas y basadas en datos.

Resum

El Reinforcement Learning (RL), impulsat pels avenços en Deep Learning, ha aconseguit recentment èxits remarcables en la resolució de problemes complexos de presa de decisions en una àmplia gamma de dominis, com la robòtica, la conducció autònoma i la modelització del llenguatge a gran escala. Aquests sistemes són ara capaços de realitzar raonaments d'alt nivell, tals com la coordinació percepció-acció i planificació a llarg termini directament a partir de dades primàries sensorials.

Inicialment reconegut per dominar jocs com el Go i el StarCraft, el RL ha evolucionat cap a aplicacions en el món real, essent la robòtica un dels dominis amb més potencial. Avenços recents d'organitzacions com NVIDIA, Tesla i Google DeepMind han demostrat el potencial de combinar l'Aprenentatge per Reforç Profund amb simulació a gran escala i dades del món real per entrenar agents capaços de realitzar tasques complexes, des de manipulació hàbil fins a locomoció àgil en entorns no estructurats.

Entre els dominis emergents, l'agricultura de precisió representa una oportunitat prometedora per als sistemes robòtics intel·ligents. Malgrat el seu potencial, aquest camp ha estat poc explorat dins el marc de l'Aprenentatge per Reforç Profund. Malgrat això, els sistemes intel·ligents i adaptatius en agricultura poden ajudar a resoldre reptes com l'escassetat d'aigua, la seguretat alimentària i la sostenibilitat, mitjançant la monitorització i gestió de les condicions del sòl i dels cultius.

Aquesta tesi proposa nous marcs d'aprenentatge per reforç per a la navegació autònoma en entorns agrícoles, tenint també en compte les condicions subterrànies del terreny. En concret, s'utilitza el radar de subsòl o georradar (GPR) per a la monitorització del sòl i la presa de decisions basades en l'entorn. Aquest sensor s'ha escollit per la seva naturalesa no destructiva i per la seva capacitat per revelar estructures subterrànies, com ara capes del sòl i distribució de la humitat, les quals són essencials per a l'agricultura de precisió.

El projecte comença amb l'adquisició i el processament de dades reals de GPR, que són comprimides en representacions latents estructurades mitjançant codificadors automàtics convolucionals personalitzats. Aquestes codificacions compactes serveixen com a entrades d'observació per als agents de RL, els quals operen en entorns de simulació construïts sobre el marc Gymnasium. Es posen a prova espais d'acció tant discrets com continus sota condicions ambientals de complexitat variable. Dins d'aquests entorns personalitzats, els agents s'entrenen per a una navegació orientada a objectius mentre s'adapten a condicions subterrànies canviants. El procés d'entrenament aprofita algoritmes d'última generació com Proximal Policy Optimization (PPO) i Soft Actor-Critic (SAC), permetent als agents aprendre polítiques efectives en escenaris discrets i continus.

Els resultats experimentals demostren que els agents aprenen polítiques efectives que aprofiten la informació subterrània per a una navegació informada. Aquest treball ofereix una integració innovadora entre aprenentatge per reforç profund i detecció geofísica per a la robòtica agrícola, proposant solucions escalables per al monitoratge autònom del sòl i la navegació sensible al subsòl en l'àmbit agrícola. En connectar la detecció mitjançant GPR amb el RL profund, aquest projecte aporta una base per a sistemes agrícoles de nova generació capaços d'exploració ambiental no destructiva i presa de decisions basada en dades.

Abstract

Reinforcement Learning (RL), driven by advances in Deep Learning, has recently achieved remarkable success in solving complex decision-making problems across a wide range of domains, such as robotics, autonomous driving, and large-scale language modeling. These systems can now perform high-level reasoning, perception-action coordination, and long-horizon planning directly from raw sensory data. From robotic arms mastering dexterous manipulation, to autonomous vehicles navigating complex environments, to conversational agents that understand and generate human-like language, modern RL systems are now solving challenges once thought out of reach.

Initially celebrated for mastering games like Go and StarCraft, RL has evolved toward real-world applications, with robotics emerging as one of the most promising domains. Recent breakthroughs by leading organizations such as NVIDIA, Tesla, and Google DeepMind have shown the promise of combining Deep Reinforcement Learning with large-scale simulation and real-word data to train agents capable of performing complex tasks, from dexterous manipulation to agile locomotion in unstructured environments.

Among emerging domains, precision agriculture presents a promising opportunity for intelligent robotic systems. Despite its potential, this field has seen limited exploration of Deep RL. However, as challenges related to water scarcity, food security, and sustainability intensify, the need for intelligent, adaptive systems in agriculture becomes increasingly critical. Robotic systems equipped with advanced sensing and learning capabilities holds the promise of revolutionizing the way soil and crop conditions are monitored and managed.

This thesis addresses this gap by proposing novel reinforcement learning frameworks for autonomous navigation in agricultural environments, driven also by subsurface field conditions. Specifically, Ground Penetrating Radar (GPR) is used as the primary sensing modality to enable both soil monitoring and environment-aware decision-making. This sensor is selected for its non-destructive nature and its unique ability to reveal underground structures, such as soil layers and moisture distribution, which are critical for precision agriculture. The project begins with the acquisition and processing of real GPR data, which are compressed into structured latent representations using custom convolutional autoencoders. These compact encodings serves as observation inputs for the RL agents which operate in simulation environments built on the Gymnasium framework. Both discrete and continuous space-action spaces are explored under varying environmental complexities. Within these custom environments, agents are trained for goal-directed navigation while adapting to varying subsurface conditions. The training pipeline leverages state-of-the-art algorithms such as Proximal Policy Optimization and Soft Actor-Critic, enabling the agents to learn effective policies in both discrete and continuous scenarios.

Experimental results demonstrate that the agents learn effective policies that leverage subsurface information for informed navigation. This work provides a novel integration of deep RL and geophysical sensing for agricultural robotics, offering scalable solutions for autonomous soil monitoring and subsurface-aware navigation in agriculture. By bridging GPR-based sensing with deep RL, this project contributes a foundation for next-generation agricultural systems capable of data-driven, non-destructive environmental exploration and decision-making.

Índex

Índex de figures

Índex de taules

1 Introduction

1.1 Motivation and Background

The increasing global demand for sustainable and efficient agricultural practices has accelerated the development of intelligent robotics systems capable of operating autonomously in complex, unstructured real-world environments. Among the critical challenges in this domain, the ability to perceive and adapt to heterogeneous dynamic soil conditions plays a fundamental role. In particular, soil moisture is one of the key environmental variables, since it directly affects crop growth, irrigation efficiency, and soil health. Accurate soil moisture estimation is essential for tasks such as precision irrigation, soil monitoring, and field planning.

These capabilities are especially relevant in the context of the United Nations Sustainable Development Goals (SDGs). Notably, SDG 2 (Zero Hunger) aims to end hunger, achieve food security, improve nutrition, and promote sustainable agriculture. Intelligent robotic systems that can perceive and respond to dynamic soil moisture levels contribute directly to these objectives through advanced decision-making, resource optimization, and improved agricultural productivity.

Traditional techniques for soil moisture measurement are either intrusive, limited in spatial coverage, or not scalable for real-time robotic deployment. Therefore, this project investigates a novel approach for soil-aware autonomous navigation using Ground Penetrating Radar (GPR) data-driven techniques. As detailed later in this work, GPR emerges as a promising, non-invasive solution to estimate subsurface properties. Mounted on a mobile platform, it enables the acquisition of continuous soil profiles while moving in the field.

However, integrating this sensor into the real-time decision-making process of autonomous systems is a complex task, especially when dealing with highly dimensional, noisy, or partially observable unstructured environments. To achieve this, Deep Reinforcement Learning techniques have been studied, with the goal of enabling mobile robots to navigate and act based on GPR-derived soil moisture information.

To improve reliability and environmental awareness, the system also incorporates sensor fusion with surface-level sensors such as LiDAR and cameras. This multimodal approach allows the robot to navigate more robustly by leveraging a unified representation of both subsurface and surface features.

1.2 Literature Review

This research is driven by the need to integrate raw subsurface GPR data into intelligent robotic decision-making processes, with a particular focus on navigation and path planning using deep reinforcement learning in agricultural settings.

Previous work has investigated GPR for subsurface analysis [4], soil moisture estimation [5], and inversion techniques using deep learning [6], but often isolated from autonomous navigation tasks. Separately, significant advances have been made in robotic control and navigation using reinforcement learning and multimodal sensor fusion [7, 8, 9], yet these systems typically rely on surface-level sensors like LiDAR or cameras, without incorporating underground information.

While studies like [4] and [10] demonstrate the potential of GPR in capturing valuable subsurface features, and [5] shows improved soil moisture estimation using hyperspectral imagery, their integration into closed-loop robot control remains unexplored. Similarly, GrASPE [7] and CropNav [8] present strong multimodal frameworks for surface navigation, but still do not incorporate subsurface awareness. Studies such as [11] demonstrate advancements in deep reinforcement learning-based navigation in uneven outdoor terrains, where elevation maps, along with the robot pose and goal position, are used to compute an attention mask representation of the environment.

This thesis builds on these insights by proposing a unified learning-based framework that uses GPR data for decision-making. By developing this framework in both offline and online learning settings, and extending it to include sensor fusion with surface sensors [9, 12], we aim to enable robust soil-aware navigation.

By proving the feasibility of this approach and evaluating its performance across various learning setup, this thesis aims to introduce the long-term vision of deploying adaptive, data-driven agricultural robots capable of making intelligent decisions in dynamic environments based on both subsurface and surface information.

1.3 Objectives

The primary goal of this thesis is to develop, train, and evaluate deep reinforcement learning-based navigation and path planning strategies that enable a omnidirectional mobile robot with offset differential wheels to make decisions by leveraging both subsurface information from GPR and surface-level data from sensors such ad LiDAR and cameras. The key objectives include:

- Acquiring and preprocessing subsurface soil data using ground penetrating radar, including data collection in simulated or real-world conditions.
- Analyzing and visualizing ground penetrating radar data to understand soil structure and characteristics, followed by extracting meaningful features relevant for robotic navigation.
- Designing simulation environments that model discrete and continuous soil moisture distributions to support different reinforcement learning frameworks.
- Developing both discrete and continuous deep reinforcement learning pipelines to study different abstraction levels of decision making in unstructured environments.
- Training deep reinforcement learning agents for both discrete and continuous scenarios, to learn navigation strategies that maximize coverage of dry regions and minimize travel distance.
- Enhancing navigation robustness through sensor fusion, incorporating surface-level data alongside GPR-derived features.

1.4 Scope of the project

Based on the objectives outlined before, the scope of the project is defined by the following key areas:

- Simulation frameworks: All experiments, training, and evaluation are conducted in simulated environments. This abstraction facilitates the development of methods that can be generalized and deployed to a wide range of robotic platforms. Physical hardware implementation, deployment, or field testing are beyond the scope of this thesis.

- Sensor Focus: The project focuses on subsurface sensing via ground penetrating radar, while surface-level data is abstracted and simulated rather than derived from specific sensors. Other sensor types, such as thermal or hyperspectral imaging, are not considered.

- Robotic Platform Assumptions: The robot is mainly modeled as a point agent operating in a 2D environment. This simplification is intended to support generalization, avoiding the need to define specific kinematic or dynamic models. By focusing on high-level decision-making, the agent can learn the fundamental structure of the tasks, which can later be adapted or fine-tuned for deployment on specific robotic platforms.

- Soil Modeling and Feature Extraction: Feature extraction from GPR data is limited to estimating soil moisture, which is then represented through discrete and continuous latent grid models. Other environmental parameters are not considered within the scope of this thesis.

- Reinforcement Learning Methods: The project focuses on model-free deep reinforcement learning algorithms. Specifically, Proximal Policy Optimization is used for the discrete scenario, while Soft Actor-Critic is employed for the continuous framework.

- Evaluation Criteria: The performance of the navigation strategies is evaluated based on trajectory efficiency, average cumulative water content encountered, preference for drier regions, and the agent's success rate in reaching predefined target positions.

1.5 Thesis Outline

The thesis is organized as follows:

- Chapter 2 presents the theoretical foundations of ground penetrating radar, including its operating principles, key parameters, techniques for interpreting and analyzing subsurface information, and tools for data visualization.

- Chapter 3 presents the principles of reinforcement learning, covering foundational concepts and the main algorithms up to the current state-of-the-art methods.

- Chapter 4 outlines the problem setup and introduces the dataset creation process along with the feature extraction architecture.

- Chapter 5 describes the discrete reinforcement learning framework, detailing the entire pipeline from environment modeling to agent training and performance evaluation.

- Chapter 6 presents the continuous reinforcement learning framework, following the same structure and methodologies as the discrete case, adapted for continuous control settings.

- Chapter 7 introduces the sensor fusion strategy and the velocity-based control approach, both integrated into the continuous reinforcement learning framework to improve navigation performance.

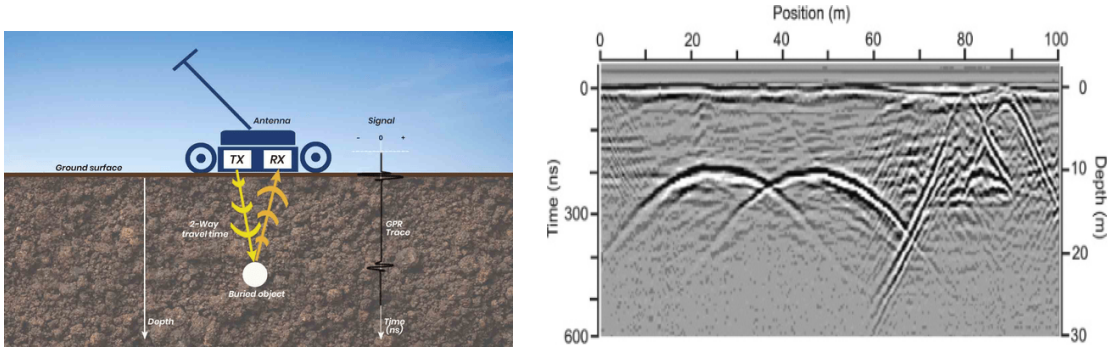
- Chapter 8 presents the overall planning and timeline of the project.
- Chapter 9 provides the economic assessment.
- Chapter 10 provides the environmental assessment.
- Chapter 11 presents the social and gender equality assessment.
- Chapter 12 concludes the thesis with key findings, limitations, and directions for future work.

2 Ground Penetrating Radar

2.1 Introduction

Ground Penetrating Radar (GPR) is a non-invasive, near-surface geophysical method of material analysis that produces a continuous, cross-sectional profile of subsurface features. It is based on the propagation of electromagnetic waves that respond to changes in the electromagnetic properties of the shallow subsurface, resulting in partial or total reflections at interfaces or point sources, which are then captured back by the GPR. The most common form of GPR measurements involves the use of a transmitter and a receiver in a fixed geometry, which are moved over the surface to detect reflections from the subsurface structures. This setup is presented in Figure 1a, while Figure 1b shows a possible GPR scan result.

Ground penetrating radar has found application across a variety of fields due to its ability to provide high-resolution subsurface imaging without excavation. In civil engineering, it is used to inspect concrete structures, locare rebar, and assess pavement conditions. In environmental studies, GPR facilitates mapping soil stratigraphy, detecting underground objects and storage tanks, and evaluating contamination plumes. Archaeologists use GPR to detect and map buried artifacts and architectural features with minimal site disturbance. In addition, in agriculture, GPR is increasingly used to analyze soil moisture content, root systems, and subsurface drainage patterns, allowing precision farming practices. These diverse applications underline the versatility of GPR as a powerful tool for non-destructive subsurface exploration.



(a) GPR setup, image retrieved online

(b) Example of GPR scan, adapted from [1]

Figure 1: Overview of the GPR system and example of GPR scan

2.2 Basic Principles

The foundations of ground penetrating radar are based on electromagnetic theory. At its core, GPR relies on Maxwell's equations, which govern the generation, propagation, and interaction of electromagnetic fields. These equations describe how electric and magnetic fields are interconnected and how they evolve in space and time, forming the mathematical backbone of wave propagation in different media. In addition to this, constitutive relations, such as permittivity, permeability, and electrical conductivity, define how materials respond to electromagnetic fields. Together, Maxwell's equations and the constitutive properties of subsurface materials allow rigorous modeling of electromagnetic wave behaviour, including reflection, refraction, and attenuation, as they encounter variations in the subsurface. This integration forms the basis for the simulation and interpretation of GPR signals in a quantitative way, thus supporting both theoretical and applied aspects of GPR analysis [1].

2.3 Maxwell's equations

In mathematical terms, electromagnetic (EM) fields and their interactions are governed by Maxwell's equations, as follows:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \tag{2}$$

$$\nabla \cdot \mathbf{D} = \rho \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{4}$$

where:

- $\mathbf{E}$ is the electric field strength vector (V/m)
- $\mathbf{H}$ is the magnetic field intensity (A/m)
- $\mathbf{B}$ is the magnetic flux density vector (T)
- $\mathbf{J}$ is the electric current density vector (A/m^2)
- $\mathbf{D}$ is the electric displacement vector (C/m^2)
- ρ is the electric charge density (C/m^3)
- t is time (s)

These differential equations capture both the dynamic and static behaviour of EM fields. In the context of ground penetrating radar, these equations are essential for modeling how electromagnetic pulses propagate through different geological layers, interact with buried objects, and generate reflections that are measured at the surface. The ability of GPR to detect variations in subsurface compositions is directly tied to how these equations govern field behaviour in heterogeneous media.

2.4 Constitutive equations

Constitutive relationships define how materials respond to applied electromagnetic (EM) fields and serve as a bridge between the theoretical field quantities described by Maxwell's equations 2.3 and the physical properties of real-world media. The material response is primarily characterized by its electrical permittivity, magnetic permeability, and electrical conductivity, parameters that govern how EM waves propagate, attenuate, and reflect within the subsurface. The constitutive equations (5)–(7) provide a macroscopic description of how electrons, atoms, and molecules respond to the application of an EM field.

$$\mathbf{J} = \sigma \mathbf{E} \tag{5}$$

$$\mathbf{D} = \varepsilon \mathbf{E} \tag{6}$$

$$\mathbf{B} = \mu \mathbf{H} \tag{7}$$

where:

- σ is the electrical conductivity that characterizes the movement of free charge when an electric field is present. Resistance to charge flow leads to energy dissipation.

- ϵ is the dielectric permittivity, which characterizes the displacement of charge constrained in a material structure due to the presence of an electric field. Charge displacement results in energy storage in the material.

- μ is the magnetic permeability, which describes how the intrinsic atomic and molecular magnetic moments respond to a magnetic field. For simple materials, distorting intrinsic magnetic moments stores energy in the material.

Moreover, σ, ε, and μ are tensor quantities and can also be nonlinear. For further details, see [1].

2.5 Material Properties

The subject of materials electrical properties, dielectric permittivity (ε), magnetic permeability (μ), and electrical conductivity (σ), is a wide and extensively studied field. In this project, the emphasis is placed on their relevance to the analysis and interpretation of ground penetrating radar data.

For the majority of the GPR applications, variations in ε and σ are the most influential factors, while changes in μ are generally negligible and rarely affect radar performance [1].

GPR is most effective in materials that exhibit low electrical losses, where signal attenuation is minimal and electromagnetic waves can propagate deeper into the subsurface. In the idealized case where the electrical conductivity σ is zero, GPR signals would suffer no conductive losses, allowing significantly higher penetration depths. However, such low-loss environments are rare in natural settings. In practice, many subsurface materials introduce substantial attenuation due to their electrical properties. For instance, clay-rich soils and regions containing saline groundwater are characterized by elevated conductivity, which causes strong signal damping. These conditions greatly limit the depth to which GPR signals can effectively penetrate, thus constraining the radar's utility in such environments. Moreover, the interpretation of GPR data becomes increasingly challenging due to the complex nature of subsurface materials, which are often heterogeneous mixtures rather than uniform substances. As a result, an accurate and quantitative analysis of GPR responses often requires additional information or modeling assumptions to account for these nonlinear interactions and composite behaviours.

This challenge becomes particularly relevant in the context of agriculture, where understanding the spatial distribution of soil moisture is essential for effective irrigation management, crop monitoring, and sustainable farming practices. GPR provides a non-invasive means of probing subsurface conditions, but its performance in agricultural fields is highly sensitive to factors such as soil texture, moisture content, and salinity, all of which directly affect σ and ε. Therefore, a careful and site-specific characterization of these electrical properties is critical to enable reliable and meaningful GPR-based sensing.

Although the electromagnetic behaviour of material mixtures can be complex, a simplified and practical perspective is often sufficient to understand their impact on GPR performance. Within the $10 - 1000$ MHz frequency range, typically employed in GPR applications, the presence or absence of water is the dominant factor that influences the behaviour of radar signals. The following general observations summarize the key principles.

Dry minerals and aggregates are typically excellent dielectric insulators, with relative permittivity (ε_r) values generally ranging from 3 to 8 depending on their mineral composition and

degree of compaction, and they exhibit negligible electrical conductivity.

Natural materials such as soils, rocks, and construction media are inherently porous, with void spaces between solid grains that can be filled with air, water, or other fluids. These pore spaces have a significant influence on the bulk electromagnetic properties.

Among fluids, water stands out as the most polarizable naturally occurring substance, having a high relative permittivity of approximately 80. Its presence substantially increases the effective permittivity of the material mixture. Furthermore, pore water typically contains dissolved ions, and the resulting ionic mobility contributes significantly to bulk electrical conductivity. In most soils and rocks, this conductivity falls within the range of 1 to 1000 mS/m and often represents the dominant source of signal attenuation at GPR frequencies. Since water is almost always present in the pore space of geologic materials, except in rare cases such as vacuum-dried samples, it exerts a significant influence on both dielectric and conductive properties, making it the principal factor affecting GPR signal behaviour in practical field conditions.

These practical insights are drawn from [1], which provides a more in-depth discussion.

2.6 Signal measurement

Ground Penetrating Radar (GPR) systems are essentially designed to measure the amplitude of electromagnetic fields as a function of time, following signal excitation. The timing unit is the core of a GPR system (see Figure 2). It governs both the generation and detection of radar pulses, ensuring precise control over the emission of electromagnetic pulses and the subsequent recording of reflected signals.

Most conventional GPR systems operate in the time domain, where short-duration pulses are transmitted into the ground, and the system records the time it takes for the reflected signals to return. This approach provides direct information about the subsurface structure based on the travel time of the pulses. Time-domain GPR is favored for its simplicity and the intuitive nature of its data representation, often resembling seismic reflection profiles.

Alternatively, some GPR systems utilize the frequency domain approach, transmitting continuous waves at varying frequencies and analyzing the frequency response of the subsurface materials. This method can offer advantages in specific applications, such as improved signal-to-noise ratio and the ability to synthesize time-domain responses through inverse Fourier transforms. However, frequency-domain systems often require more complex calibration and signal processing techniques.

Regardless of the domain of operation, the effectiveness of GPR is influenced by factors such as the dielectric properties of the subsurface materials, antenna design and system bandwidth. Understanding these factors is crucial for accurate data interpretation and successful application of the GPR.

Furthermore, several fundamental equations can be introduced to better describe signal propagation and measurement in GPR systems. The velocity v of electromagnetic wave propagation in a medium is governed by the relative permittivity ε_r of the susurface material, following the relation:

$$v = \frac{c}{\sqrt{\varepsilon_r}} \tag{8}$$

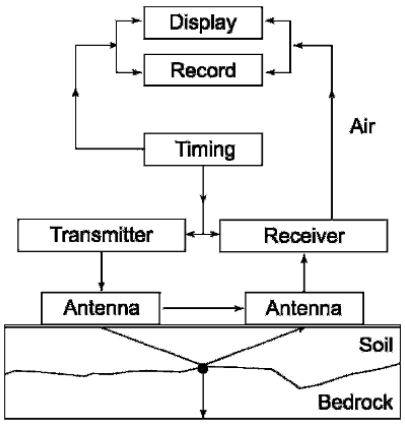


Figure 2: Block diagram of the GPR main components, adapted from [1]

where c is the speed of light in vacuum. The depth d of a reflecting target can then be estimated using the two-way travel time t as:

$$d = \frac{vt}{2} = \frac{ct}{2\sqrt{\varepsilon_r}} \tag{9}$$

The resolution of a GPR system is closely related to the bandwidth B of the transmitted signal. The vertical range resolution Δz can approximated as:

$$\Delta z \approx \frac{v}{2B} \tag{10}$$

indicating that higher bandwidth allows for finer resolution of subsurface features.

The attenuation of the signal is another critical factor and is influenced by the electrical conductivity σ of the medium. The attenuation constant α in a low-loss dielectric can be approximated as:

$$\alpha \approx \frac{\pi\sigma\sqrt{\mu\varepsilon}}{f} \tag{11}$$

where μ is the magnetic permeability, ε is the absolute permittivity, and f is the frequency. Thus, higher frequencies tend to experience greater attenuation in conductive media, emphasizing the need for careful frequency selection in practical GPR applications.

2.7 Antennas

Antennas are the most critical component of a Ground Penetrating Radar system, as they directly determine both the depth of penetration and the resolution of the data acquired by a GPR system. Given their important role, it is essential to dedicate a focused discussion of their characteristics and principles. Therefore, this section presents key concepts and essential information to provide a general overview of the different types, configurations, and operational principles. Understanding these aspects is crucial for selecting the appropriate antenna setup based on the specific application, desired resolution, and subsurface conditions [1].

2.8 General Classification and Basic Parameters

Antennas can be classified into two general classes, omnidirectional and directional.

Omnidirectional antennas radiate energy in all directions simultaneously. This leads to the concept of an isotropic antenna, an idealized model that radiates energy equally in all directions. Although isotropic antennas do not exist in practice, they are used as reference for comparing the performance of real-world antennas.

In contrast, directional antennas radiate energy in patterns of lobes or beams that extend outwards from the antenna in one direction for a given antenna position. The radiation also produces weaker side lobes, also known as sidelobes, which have typically minimal influence on the primary radiation patter. In conventional radar systems, the main lobe may vary in angular width from 1° in some case to 15° in others. For GPR antennas, the main lobe is much wider and can be in the range of 90°. Directional antennas are characterized by two important characteristics, directivity and power gain.

Directivity of an antenna refers to the degree of sharpness of its beam. An antenna is said to have high directivity when its beam is narrow in either the azimuth or elevation plane, meaning it concentrates energy more sharply in that direction. Conversely, a wide beam indicates low directivity. Thus, for example, if an antenna has a narrow azimuthal beam and a wide elevation beam, the horizontal directivity is high and the vertical directivity is low. Increasing the antenna's directivity enables it to cover the same range with less power as the energy is more efficiently concentrated in a specific direction.

The power gain of an antenna is directly related to directivity and is the ratio of its radiated power to that of a reference dipole, under identical exciting conditions and from the same position. An antenna with high directivity has a high power gain and vice versa.

For more detailed information on this topic, refer to the corresponding chapter in [1].

While these theoretical concepts are fundamental for understanding antenna performance, they also have direct implications for real-world use. In practice, selecting a Ground Penetrating Radar system involves evaluating several key parameters that significantly influence field performance. The main factors include the choice between shielded and unshielded antennas, operating frequency, required penetration depth and resolution. A thorough understanding of these trade-offs is necessary to choose a GPR configuration that meets the specific demands of the intended application and performs effectively under operational constraints.

2.9 Shielded vs Unshielded

GPR typically uses either shielded or unshielded antennas, depending on the specific operational environments.

Shielded antennas are preferred in environments where external noise or surface reflections must be minimized, since they offer superior protection against electromagnetic interface (EMI). They employ a conductive material, like a metal casing, to block external signal from interfering with antenna's performance.

Unshielded antennas, on the other hand, do not have this protective barrier and are more vulnerable to EMI. However, they have the advantage of being more effective for velocity analysis and common midpoint surveys. In addition, unshielded antennas are less expensive and structurally simpler, as they require less materials and less complex manufacturing processes compared to shielded designs.

2.10 Operating Frequency

The operating frequency of the antenna plays a key role. Lower-frequency antennas (e.g., $100-400MHz$) can penetrate deeper into the ground but offer lower resolution, making them suitable for detecting large or deep targets. In contrast, higher-frequency antennas (e.g., $900Mhz$ and above) provide finer resolution but are limited in depth penetration, making them suitable for shallow, high-detail investigations. Therefore, selecting the appropriate antenna requires weighing these trade-offs to ensure optimal performance, aligning the antenna characteristics with the specific application and the predominant subsurface conditions.

2.11 Bandwidth

Another critical parameter is the bandwidth of the antenna, which determines the ability of the system to distinguish closely spaced reflectors. A wider bandwidth leads to shorter pulse durations and, consequently, better range resolution [13]. The effective bandwidth is usually a function of both the antenna design and the transmitted pulse waveform.

2.12 Antenna polarization

Antenna polarization also has important applications, as already introduced in 2.8. GPR antennas are typically linearly polarized, and the orientation of the transmitted polarization relative to subsurface features can strongly affect signal reflection. For example, horizontally aligned antennas are better suited to detect horizontal layers, while vertically oriented antennas can enhance the response of vertically aligned structures [14].

These parameters must be carefully selected and adapted based on the intended application, the target depth, and the geological environment to maximize the quality and interpretability of the GPR data collected.

2.13 Commercially Available GPR Systems and Antennas

To conclude this section on GPR system parameters, it is useful to provide a general overview of the commercial available ground penetrating radar systems. Several leading companies provide GPR systems, each offering unique solutions for different applications in multiple industries. The following is an overview of some of the most prominent GPR providers as of the time this thesis is being written.

2.13.1 GSSI (Geophysical Survey Systems, Inc.)

GSSI is one of the most well-known manufacturers of GPR systems. Their products, such as the *UtilityScan Pro*, are widely used in utility locating, environmental surveys, and forensic investigations. Their systems are known for their versatility and high-quality data collection capabilities, especially in detecting susurface objects and structures.

2.13.2 Leica Geosystems

Leica Geosystems is another important provider of this industry. Leica's *DS2000* system is primarily used for utility detection, geotechnical investigations, and concrete scanning. It provides high-resolution data for mapping underground utilities, which is critical for urban planning, construction, and environmental assessments. Leica systems are famous for their accuracy and ease of use.

2.13.3 IDS GeoRadar

IDS GeoRadar offers advanced multi-frequency systems, such as the *Stream UP*, which is designed for large-scale surveys, archaeological investigations, and geophysical mapping. The *Stream UP* can perform high-speed surveys and is suitable for both shallow and deeper subsurface investigations.

2.13.4 Sensors & Software

Sensors & Software provides a range of GPR systems, including the *LMX150 FINDAR*. This system is particularly popular for forensic applications, environmental investigations, and utility locating. The *LMX150* is known for its portability, ease of operation, and high-quality data output, making it ideal for both field and research applications.

2.13.5 Proceq

Proceq specializes in portable GPR systems like the *GS8000*, commonly used for concrete inspection, civil engineering, and structural assessments. The *GS8000* is designed to provide detailed images of concrete structures, helping engineers detect cracks, voids, and other structural anomalies.

2.13.6 MALA Geoscience

MALA Geoscience, a globally recognized provider of GPR technology, offers a comprehensive range of systems tailored for various subsurface investigation needs. Notable products include the *X3M* and the *Easy Locator* series, which are widely used in geological surveys, environmental monitoring, utility detection, and construction site assessments. These systems are particularly valued for their intuitive user interfaces, robust field performance, and compatibility with advanced data visualization tools. MALA's GPR solutions are also widely adopted in non-destructive testing (NDT) and geophysical mapping applications, where precision and ease of use are crucial. Equipment from MALA Geoscience will be used in the initial phases of this project to carry out high-resolution subsurface data collection and analysis.

In conclusion, when selecting a GPR system, it is important to consider the specific application requirements, including depth of penetration, resolution needs, terrain conditions, and data acquisition settings. Consultation with manufacturers or authorized distributors is highly recommended to ensure the appropriate selection of the system and the optimal performance of the investigation.

2.14 Data Analysis and Visualization

The transformation of GPR data into application-specific information can be achieved through two main strategies.

The first approach focuses on extracting quantifiable wave property variables such as propagation velocity, signal attenuation, or electromagnetic impedance. These physical properties are then translated into application-specific quantities.

The second approach, widely used across geophysical methods, involves visualizing the GPR responses through different spatial formats, such as two-dimensional profiles known as A-

scans, plan-view maps called B-scans, and three-dimensional volumetric representations referred to as C-scans. Figure 3 visually depicts these three different spatial formats.

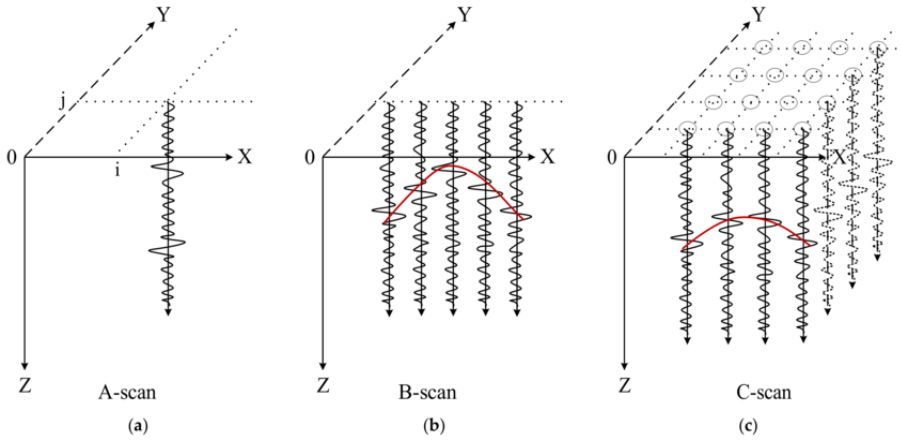


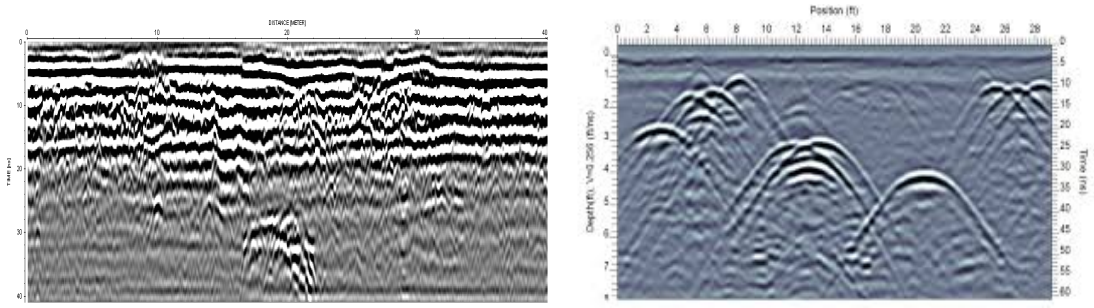
Figure 3: Types of GPR radargrams' visual presentations: (a) A-scan; (b) B-scan; (c) C-scan. Adapted from [2]

These visual formats facilitate the identification of subsurface anomalies or contrasts by highlighting variations in signal response. However, interpreting these representations still depends on human expertise or computer-aided pattern recognition to infer the presence and nature of features of interest. This is mainly done by analyzing the reflections of electromagnetic waves that bounce off different materials and geological structures beneath the surface. The strength, timing, and shape of these reflected signals provide valuable clues about subsurface conditions. The fundamental principles behind these interpretations can be summarized as follows.

Strong reflections typically arise at interfaces where there is a significant contrast in dielectric permittivity, such as, for example, the boundary between dry soil and a metal pipe, or between air-filled voids and surrounding earth. These reflections are often visualized as bright bands or sharp changes in signal amplitude on a radargram. In addition to amplitude-based features, the geometric pattern of the reflections is also crucial. For example, hyperbolic shapes typically indicate point or linear reflectors such as buried piped, cables, or rebar. These hyperbolic shapes arise because the reflected signal returns at different times as the antenna passes over the object, creating a distinct curved signature in the B-scan profile. Figures 4a–4b provide a visual intuition of these ideas. In particular, Figure 4a shows the strong flat reflections due to possible subsurface boundaries, Figure 4b, instead, shows the parabolic pattern due to some point reflectors.

Although ground penetrating radar is a powerful tool for subsurface imaging, it comes with its own limitations that can affect its performance and the interpretation of its results. One of the primary constraints is that GPR requires a significant contrast in dielectric properties, both between the subsurface layers and between the target and the surrounding material, for an effective detection.

Furthermore, GPR cannot directly identify the material of detected objects, it only identifies changes in electromagnetic properties, which manifest as contrast on the radargram. Hence, the identification of the material's nature relies on additional context, expert interpretation, or the use of complementary sensing technologies.



(a) B scan showing possible different sublevels

(b) B scan showing possible different refractors

Figure 4: Examples of different B scans scenarios, images retrieved online

In addition, the performance of the GPR is significantly affected by the conductivity of the soil. Highly conductive environments, such as those containing wet clay, saline content, or high moisture levels, tend to absorb radar signals rather than reflect them. This absorption leads to signal attenuation, resulting in reduced depth penetration and degraded image clarity.

Another important limitation is physical obstructions on the surface, such as rocks, dense vegetation, or uneven terrain, which can also interfere with the GPR's ability to collect accurate data.

Finally, the resolution of the GPR depends on multiple factors, including the frequency of the antenna, the size of the target, and its depth below the surface. Lower-frequency antennas penetrate deeper but offer lower resolution, while higher frequencies improve resolution but at the cost of shallower penetration. As a result, GPR may struggle to detect small or deeply buried objects under certain conditions.

2.15 GPR-SLICE Software

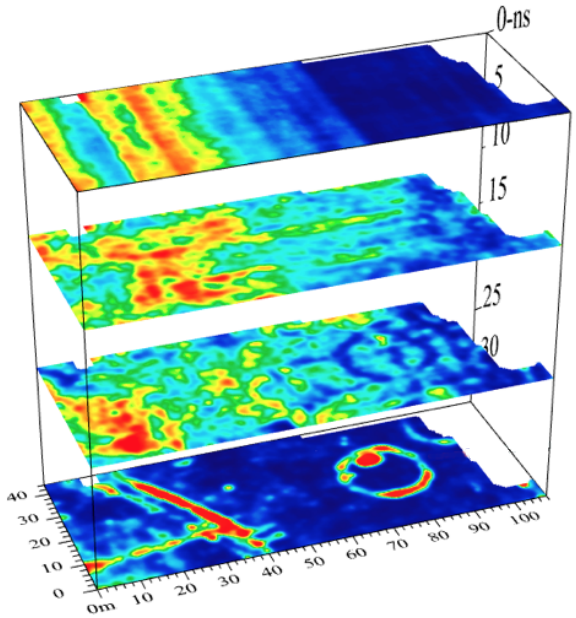
Among the various available tools for Ground Penetrating Radar data visualization, GPR-SLICE has been selected for this project due to its extensive capabilities in processing and rendering complex subsurface imagery. Developed by Dr. Dean Goodman and now part of the Screening Eagle Technologies suite, GPR-SLICE is one of the most advanced and widely adopted software platforms in the field of geophysical data interpretation [15].

The software excels in generating both two-dimensional and three-dimensional representations of GPR data, including time-slice images, amplitude maps, fence diagrams, and volumetric isosurface renderings. These visualization methods enable intuitive interpretation of subsurface features, which is particularly valuable in archaeological prospection, civil engineering assessments, and environmental studies. GPR-SLICE also supports a wide range of data processing tools, such as background removal, migration, filtering, and Hilbert transformation, providing a robust environment for enhancing data quality and extracting relevant features [16].

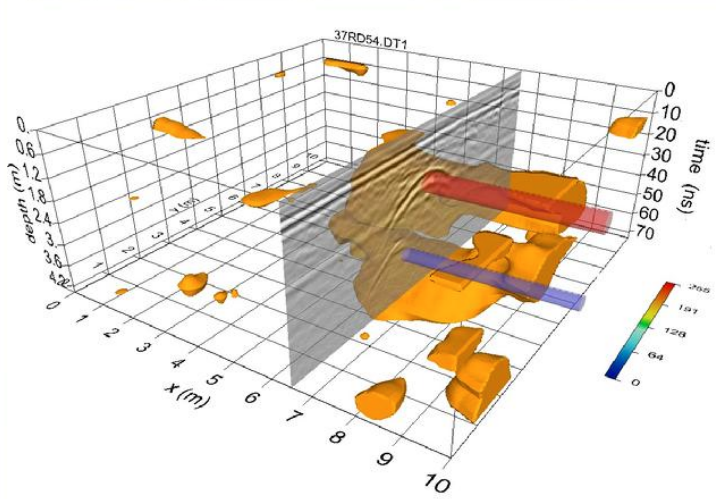
Moreover, its compatibility with various GPR hardware systems, including MALA Geoscience, GSSI, IDS, and Sensors & Software, ensures seamless integration of data collected during the survey phase. The ability to incorporate GPS and total station data also allows precise georeferencing and topographic corrections, which is essential when conducting surveys in uneven or complex terrain [17]. Given these strengths, GPR-SLICE is well suited to support the visualiza-

tion needs of this project, enabling the production of accurate and high-resolution subsurface maps.

Figure 5 shows some of the possible applications of GPR-SLICE.



(a) Depth slices at multiple depths, displayed side-by-side in GPR-SLICE [17].



(b) GPR-SLICE isosurfaces from ground-penetrating radar data with 3D drawn interpretations [17].

Figure 5: Overview of possible GPR-SLICE Software Features and Applications

3 Reinforcement Learning Overview

3.1 Introduction

Reinforcement Learning (RL) is a branch of machine learning where agents learn optimal behaviours by interacting with an environment through trial and error. Unlike supervised or unsupervised learning, RL does not rely on static datasets. Instead, an agent perceives the state of its environment, takes actions, receives feedback in the form of rewards or penalties, and gradually refines its policy to maximize cumulative long-term rewards.

Over the past decade, RL has made significant advancements across multiple domains:

- Autonomous Agents: RL forms the foundation of decision-making in self-driving cars, unmanned aerial vehicles (UAVs), and mobile robots, enabling them to navigate complex, uncertain environments.
- Games: RL powers agents like AlphaGo and AlphaStar, which have surpassed human performance in games such as Go, StarCraft II, and Dota 2.
- HVAC Optimization: RL is used to optimize heating, ventilation, and air conditioning systems by balancing energy consumption and user comfort.
- Finance: In trading and portfolio management, RL algorithms dynamically adapt strategies based on market feedback to maximize returns.
- Online Advertising and Recommendation Systems: Personalized content delivery, such as which news article to show or which product to recommend, is modeled as an RL problem, learning from user engagement.
- Healthcare and Computational Biology: RL supports applications such as personalized treatment planning, molecular drug discovery, and optimization of bioinformatics pipelines.
- Language Models and Human Feedback: RL is becoming increasingly important in fine-tuning large language models (LLMs), such as CHatGPT, through methods like Reinforcement Learning from Human Feedback (RLHF). This approach allows models to better align with human preferences and safety constraints, hence improving the quality of dialogue systems.

Among these domains, robotics stands out as a particularly compelling application area for reinforcement learning, due to its inherently interactive and goal-driven nature. Robots are often required to make decisions in real-time, adapt to dynamically changing environments, and learn from sparse or delayed feedback-scenarios that naturally align with the core principles of RL framework.

Notable robotic applications of RL include:

- Mobile Navigation: RL enables robot to learn optimal navigation policies in unstructured or unknown environments, avoiding obstacles and reaching goals without predefined maps.

- Manipulation and Grasping: Robotic arms employ RL to acquire dexterous skills like grasping, stacking, or tool use, often via simulation-to-real transfer.
- Multi-agent Collaboration: RL supports swarm robotics and collaborative tasks, such as coordinated exploration or assembly missions.
- Humanoid Locomotion and Dexterity: RL is increasingly used to train humanoid robots to perform complex, human-like daily tasks, such as walking, balancing, and manipulating objects, with the goal of enabling autonomous operation in real-world environments.

All these different applications highlight the versatility and growing importance of reinforcement learning in both research and real-world systems. From mastering games to enabling physical autonomy in robots, RL has proven to be a powerful framework for sequential decision-making.

In the following sections, we explore the fundamental principles of reinforcement learning. We begin by introducing the core concepts of the RL framework, then proceed to examine the main algorithms and learning paradigms. Finally, we present state-of-the-art reinforcement learning methods, where we integrate deep learning concepts within RL settings.

3.2 Elements of Reinforcement Learning

A Reinforcement Learning framework is composed of several fundamental components. The first important element is the agent, the actual entity that aims to solve a desired task. Then, the environment, with which the agent interacts. The environment has two main tasks, it provides the rewards at each time step and it moves the agent to a new state depending on the state-action pair. Indeed, to fully characterize the RL system, it is necessary to define a state space and an action space. The state space gives an exhaustive description of all possible configurations of the system the agent may encounter, capturing the full range of system's conditions. The action space provides a set of all possible moves or decisions the agent can make, which may depend on the current state. Both state and action spaces can be either discrete or continuous. A possible framework scheme is shown in Figure 6. The underlying principle of this scheme is as follows.

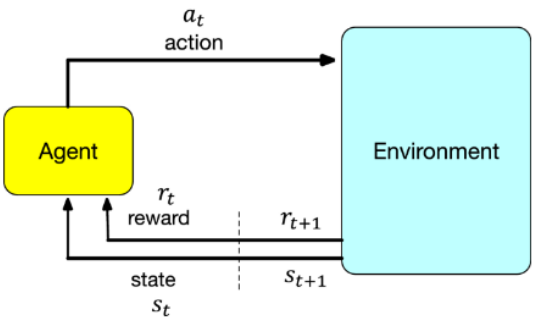


Figure 6: Reinforcement learning core structure, image retrieved online

At each time step, the agent is in a specific state and selects an action following its policy (see Section 3.3 for more details on the policy). After that, the agent receives a scalar reward from the environment and transitions to the next state according to the environment's dynamics. In an episodic environment, this process continues until the agent reaches a terminal state, at which

point the episode restarts. In contrast, in non-episodic environments, the process continues indefinitely without a terminal state. In some cases, the agent may also benefit from delayed rewards, where the consequences of an action are not immediately observable but accumulate over time. This further complicates the learning process, requiring the agent to use strategies such as discounting future rewards or leveraging temporal difference learning to properly credit actions.

3.3 More RL basics

Beyond the agent and the environment, four other main elements of a reinforcement learning system can be identified: a *reward signal*, a *value function*, a *policy*, and, optionally, a *model* of the environment [18].

The *reward signal*, partially introduced in the previous section, plays a fundamental role in defining the objective of a reinforcement learning problem. At each time step, the agent receives a scalar feedback value from the environment, known as the reward, which evaluates the immediate outcome of the agent's action within the current state. This reward is mainly used for shaping the agent's behaviour, or policy, over time. Depending on the environment or task, the reward signal can be either deterministic, providing the same outcome for the same action in a given state, or stochastic, where the reward may vary due to inherent randomness or uncertainty. Understanding the nature of the reward signal is crucial as it directly influences the learning dynamics and the strategies that the agent develops to maximize its cumulative reward.

A *policy* π is a mapping from state to action and defines the behaviour of the agent. It can be deterministic, so $A = \pi(S)$, or stochastic, thus $A \sim \pi(S)$. The goal of the RL agent is to find the optimal policy that maximizes cumulative rewards, the so called *return*. Formally, the *return* G_t represents the total accumulated reward an agent receives from time step t onward. It quantifies the long-term benefit of following a policy by taking specific actions in the environment, and is defined as the discounted sum of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{12}$$

Where the discount factor $\gamma \in [0, 1]$ is used to weight future rewards. It is the present value of future rewards and it values recent rewards more if it is close to zero ($\gamma = 0$ myopic case) or it gives to all the future rewards roughly the same importance when it gets closer to one ($\gamma = 1$ far-sighted case).

A key challenge at this stage, that exists only in the reinforcement learning framework and not in the pure forms of supervised and unsupervised learning, is to manage the trade-off between exploration and exploitation. To maximize rewards, a RL agent should favor actions that have previously yielded positive outcomes. But to discover such actions, it has to try actions that it has not selected before. This creates a fundamental tension: the agent must exploit known information to accumulate rewards, while also exploring new possibilities to improve future decision-making. Focusing only on exploration or exploitation leads to suboptimal performance. Hence, the agent must experiment with a diverse set of actions while gradually shifting preference toward those with higher estimated value. Furthermore, in stochastic environments, each action must be sampled multiple times to reliably estimate its expected reward. To ad-

dress this trade-off, several approaches have been developed, such as $\epsilon - greedy$, greedy with optimistic initialization, Upper Confidence Bound (UCB), gradient policy, and others.

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run [18]. It quantifies how good it is for an agent to be in a particular state, based on the expected cumulative reward it can obtain starting from that state and following a given policy. More formally:

$$v(s) = \mathbb{E}[G_t \mid S_t = s] \tag{13}$$

The expectation in the formula accounts for the inherent stochasticity of the environment, meaning that taking the same action in the same state can lead to different subsequent states.

Closely related to the value function is the action-value function, often called the Q-function. The action-value function estimates the expected return starting from a specific state, taking a particular action, and subsequently following the policy. Formally:

$$q(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \tag{14}$$

Both functions are critical for decision-making, as they guide the agent in selecting actions that maximize long-term rewards rather than immediate gains.

The fourth and final component of reinforcement learning systems is an environment model. This model simulates the behaviour of the environment or, more broadly, enables predictions about how the environment will respond to various actions [18]. However, not every framework requires such a model. In many cases, particularly in model-free approaches, the agent learns directly from trial-and-error interactions with the environment, without constructing a predefined model. Further details will be discussed in the following sections.

3.4 Markov Decision Process

A Markov Decision Process (MDP) is the mathematical framework used to formalize a reinforcement learning (RL) problem. It builds upon the concept of a Markov Reward Process (MRP), which itself is an extension of the simpler Markov Process (MP).

A Markov Process is a memoryless random process characterized by a set of Markov states $\mathcal{S}$ and a state transition probability $\mathcal{P}$. A state S_t is Markov if and only if

$$\mathcal{P}(S_{t+1}|S_t) = \mathcal{P}(S_{t+1}|S_1, \ldots, S_t) \tag{15}$$

The transition probability is defined as:

$$\mathcal{P}_{ss'} = \mathbb{P}\big(S_{t+1} = s' \mid S_t = s\big) \tag{16}$$

And it represents the stochastic transitioning behavior from one state s to its successor s'. The state transition matrix $\mathcal{P}$ defines all the transition probabilities.

$$\mathcal{P} = \begin{array}{c} \\ \text{from} \end{array} \begin{array}{c} \text{to} \\ \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \end{array} \tag{17}$$

The transition matrix $\mathcal{P}$ is said to be *row-stochastic* because each row of the matrix sums to 1. This reflects the fact that, for any given state s, the total probability of transitioning to all possible subsequent states s' is equal to 1. In other words, for each state s, we have:

$$\sum_{s'} \mathcal{P}(s'|s) = 1 \tag{18}$$

This property ensures that the system is probabilistic in nature.

Markow Reward Process is a Markov Chain with reward values. It can be described as a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{P}$ the state-transition probability matrix, $\mathcal{R}$ the reward function, and γ the discount factor, with $\gamma \in [0, 1]$. The reward function $\mathcal{R}$ assigns to each state s the expected immediate reward as:

$$\mathcal{R}(s) = \mathbb{E}\big[R_{t+1} \mid S_t = s\big], \tag{19}$$

where R_{t+1} denotes the reward received at the next time step. With MRP, it is also now possible to define the concept of return G_t as the total discounted reward from time-step t, in particular:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{20}$$

See Section 3.3 for more details about discount element. The return G_t in Equation 20 represents the cumulative reward obtained along a single trajectory, so it is possible to define the value function $v(s)$ of a MRP as the expected return starting from state s:

$$v(s) = \mathbb{E}[G_t \mid S_t = s] \tag{21}$$

By exploiting the structure of G_t, it can be defined recursively as follows:

$$G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) = R_{t+1} + \gamma G_{t+1} \tag{22}$$

So, by using the law of iterated expectations, it is possible to obtain the Bellman equation for MRP, as follows:

$$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] \end{aligned} \tag{23}$$

Finally, Markov Decision Process introduces the concept of action $\mathcal{A}$, so it is described by the tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma \rangle$ where $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is now defined as:

$$\mathcal{P}^a_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \tag{24}$$

and the reward function as:

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \tag{25}$$

With the presence of actions, the agent is now able to take decisions and interact with the environment. Thus, it becomes necessary to define a distribution over the possible actions conditioned on a given state, the so called policy π, and defined as:

$$\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s) \tag{26}$$

A policy fully characterizes the behavior of an agent by specifying how it selects actions based on the current state of the environment, guiding the agent's decision-making process at every step. The previous definition of the value function V with MRP, now becomes:

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{27}$$

Now that there is the agency, the values of a state depends also on the policy. Furthermore, a new function Q_π must be introduced, which quantifies the expected quality of a specific state-action pair. Formally, the action-value function Q_π of an MDP is the expected return from state s if action a is taken and the policy π is then followed, hence:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{28}$$

As already done before with MRP, it is possible to exploit the iterative structure of G_t, obtaining the Bellman Expectation Equations for MDPs:

$$V_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s\right] \tag{29}$$

$$Q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right] \tag{30}$$

The relationship between Q_π and V_π can be derived by exploiting their definitions, given in (29), (30):

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q_\pi(s, a) \tag{31}$$

Computing these quantities is an integral part of the agent's learning process and, consequently, of the reinforcement learning problem. Various approaches will be examined in the next section (see Section 3.5). Furthermore, other quantities must be defined. The optimal state-value function $v_*(s)$ is the maximum state-value function over all the policies:

$$v_*(s) = \max_\pi v_\pi(s) \tag{32}$$

The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all the policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{33}$$

From a reinforcement learning perspective, an MDP is considered *solved* when the optimal action-value function is known, as this allows the agent to always select the best action from any given state. For example, knowing $Q(s, a)$ for each (state,action) pair, the agent could learn a possible optimal policy by acting greedily, hence picking $a_t = argmax_a Q(s_t, a)$. This leads to the well-known exploration–exploitation dilemma, already mentioned in Section 3.3. Finding the right balance between exploring new actions and exploiting known rewarding ones is a fundamental challenge in reinforcement learning. To address this, various algorithms have been developed, and a wide range of techniques have been proposed to ensure effective exploration while maximizing long-term rewards. The most well-known and used strategies have been presented in the previous section, Section 3.3 and are now explained in further detail. The $\epsilon - greedy$ policy involves selecting the greedy action with probability $1 - \epsilon$, and choosing a non greedy ones with probability ϵ. The hyperparameter ϵ controls the exploration/exploitation trade-off. The greedy with optimistic initialization is one of the techniques to encourage exploration in the early stage, but it has some limitations when the problem is non-stationary or when, in some applications, it is not possible to know optimistic values range in advance.

Upper confidence bound approach uses uncertainty in the value estimates for governing the trade-off. The more an action is taken and the lower the confidence interval is and the approach select the action that potentially could be the best. Formally:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{34}$$

Where the first term is the exploitation part, the second is the exploration, with $N_t(a)$ that stands for the number of times the action a has been taken. Finally, Gradient policy defines a preference value for each action, and actions are selected based on that. The larger the preference, the more often the action is taken. Formally:

$$\pi_t(a \mid s) \doteq \frac{e^{H_t(a|s)}}{\sum_{b=1}^{k} e^{H_t(b|s)}} \tag{35}$$

where $H_t(a \mid s) \in \mathbb{R}$ is the preference.

In conclusion, Markov decision processes can be solved from two distinct perspectives: by computing the value function and action-value function of a given policy, an approach known as policy evaluation, or by identifying the optimal policy itself, referred to as policy control.

3.5 Learning techniques

3.6 Dynamic Programming for MDPs

In simple scenarios, where the model of the environment's dynamics is completely known and state and action sets are relatively small, it is possible to apply Dynamic Programming algorithms for solving MDPs. Dynamic Programming (DP) refers to a class of methods that solves complex problems by breaking them down into subproblems, following a *divide and conquer* strategy. They are mainly used for solving problems that present an optimal substructure and an overlapping subproblems, so solutions of the subproblems can be efficiently reused. By systematically storing and reusing intermediate results, DP avoids redundant computations and significantly improves computational efficiency.

Recursive relationships in MDPs make the application of DP particularly effective. It is important to note that a complete and accurate model of the environment's dynamics is needed, hence DP is not suitable for full RL problems, where $\mathcal{P}$ is not known. DP methods, such as policy evaluation, policy improvement, policy iteration, and value iteration, rely on Bellman equations to compute value functions and find optimal policies. For instance, considering a policy evaluation problem with finite set of actions and states, it is possible to obtain the DP update rule by firstly rewriting the Bellman equation as follows.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned} \tag{36}$$

Now, instead of solving the Bellman equation directly, the update rule is derived from it (for all $s \in \mathcal{S}$):

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right] \end{aligned} \tag{37}$$

For a deep explanation and for all the details of the mentioned algorithms, see [18]. These algorithms operate in a tabular setting, where the value of each state (or state-action pair) is explicitly stored and updated. Since DP assumes full knowledge of transition probabilities and rewards, it is primarily a theoretical tool, providing important foundations and insights for more practical, model-free approaches. Furthermore, DP becomes impractical in environments with very large or continuous state spaces due to the curse of dimensionality, motivating the development of approximate methods and learning-based alternatives.

3.7 Methods for full RL scenarios

Moving towards the implementation of reinforcement learning in real-world scenarios, different approaches are required in order to address the complexity and dynamic nature of such environments. For now on, there won't be any prior knowledge of the transition probability $\mathcal{P}$. These approaches can be broadly classified into two categories, model-free methods and model-based methods. Model-free methods focus mainly on learning the value of actions through direct interaction with the environment, without building a model of the environment's dynamics. For this reason, they require many interactions with the environment to converge to an optimal solution, especially in complex or high-dimensional spaces. On the other hand, model-based methods involve constructing a model of the environment, which predicts the future state and reward for a given action. This model enables the agent to plan ahead by simulating potential outcomes, thus improving decision-making efficiency. However, they are more computationally demanding than the model-free approaches. In this thesis, model-free approaches will be the primary focus; however, further studies and improvements could be explored through model-based approaches.

3.7.1 Monte Carlo Methods

A first widely used model-free approach is the Monte Carlo estimation method, where the Monte Carlo technique is applied to estimate the value function or the action value function of a given policy π by averaging the returns obtained from a particular state or a state-action pair. Different versions are used, *first-visit MC* estimates V or Q as the average of the returns following only the first visit of s or (s, a), whereas the *every-visit MC* method averages the returns following all visits to s or $s(s, a)$. *First-visit MC* has been most widely studied, dating back to the 1940s, and a possible pseudocode for the estimation of v_π is presented in Algorithm 1. Every-visit MC extends more naturally to function approximation and eligibility traces, which will be discussed in the following sections. For more details, see [18].

Algorithm 1 First-Visit Monte Carlo Prediction (for estimating $V \approx v_\pi$)

Require: A policy π to be evaluated
Initialize $V(s) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}$
Initialize $Returns(s) \leftarrow$ empty list, for all $s \in \mathcal{S}$
while Loop forever (for each episode) **do**
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for each step of episode, $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if S_t does not appear in $S_0, S_1, \ldots, S_{t-1}$ **then**
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$
        end if
    end for
end while

Monte Carlo methods are also effective in addressing the control problem. The state-action space is much larger than the state space and MC methods need to collect data for each state-action pair to ensure optimality. As a result, a sufficient level of exploration becomes crucial to achieve this goal. Several techniques have been studied and developed, such as MC with Exploring Starts (ES) and $\epsilon - soft$ policies.

In MC with ES the initial state S_0 and action $A_0 \in \mathcal{A}(S_0)$ of the episode are randomly chosen, such that all pairs have probability greater than zero. However, this technique is not always feasible, since in many scenarios it is impossible to start an episode with all the possible state-action pairs. This is where the $\epsilon - greedy$ and $\epsilon - soft$ policies become useful. In general, soft policies are stochastic policies that have $\pi(a \mid s) > 0$, and $\epsilon - soft$ policies are soft policies that have $\pi(a \mid s) \geq \frac{\epsilon}{|A(s)|}$ for all a and s, with $\epsilon > 0$.$\epsilon - greedy$ is the *greediest* $\epsilon - soft$ policy.

All the reinforcement learning methods discussed so far fall under the category of *on-policy learning*. In on-policy methods, the learning agent improves the policy π based on the experience sampled by following the same policy π. In other words, the agent uses its current policy both to explore the environment and to update itself. An alternative approach is known as *off-policy learning*. In off-policy methods, the agent learns about a target policy π while using data generated by a different behavior policy b. This decoupling allows for greater flexibility, as the behavior policy can be more exploratory or even fixed, while the agent still learns and improves the target policy. Off-policy learning is particularly useful to leverage previously collected data (e.g., from demonstrations or from different strategies) or when exploration under the current policy is too costly or inefficient. Mathematically, in off-policy learning, the value of a target policy π is estimated using samples generated by a different policy b, and importance sampling techniques may be required to correct for the discrepancy between b and π. A well-known example of off-policy learning is Q-learning, where the behavior policy might be ϵ-greedy (to encourage exploration), but the updates are made towards the greedy policy derived from the Q-values, which is the target policy. More details will be presented in the next section.

3.7.2 Temporal Difference Learning

Temporal Difference (TD) learning is another approach to model-free RL, used for both value prediction and control tasks. Unlike Monte Carlo methods, which rely on waiting until the end of an episode to update value estimates, TD methods update estimates based on partially observed returns using the so called bootstrapping method (previously saw also with dynamic programming). Bootstrapping refers to the process of updating an estimate based on other existing estimates, rather than waiting for the final outcome or true return. In particular, TD learning combines ideas from both dynamic programming and Monte Carlo methods. It learns directly from experience, like Monte Carlo, and uses bootstrapping, like dynamic programming. Furthermore, TD learning could be seen as a general framework that also includes Monte Carlo methods.

The use of bootstrapping in Temporal Difference learning offers several key advantages. First, it enables online learning, allowing value estimates to be updated immediately after each time step, without requiring the final outcome of an episode. As a result, TD methods can learn effectively from incomplete sequences and are well-suited for continuing tasks. In contrast, Monte Carlo (MC) methods require complete episodes to compute actual returns, which limits their application to episodic environments. Moreover, bootstrapping generally leads to faster convergence compared to MC methods, as updates can be made more frequently and with lower variance, but with the cost of introducing some bias. This trade-off often results in more efficient learning in practical settings where immediate or incremental updates are desirable.

Temporal Difference learning plays a central role in solving control problems. TD methods form the foundation of some of the most widely used and significant control algorithms, such as SARSA (State–Action–Reward–State–Action) and Q-learning. SARSA is an on-policy algorithm that updates action-value estimates based on the action taken by the current policy. It gradually improves the policy by learning from sequences of state-action-reward-next state-next action tuples, making it suitable for applications where the agent must account for the actual behavior it follows, including exploratory actions. A possible pseudocode, extracted from [18], is outlined in Algorithm 2.

Algorithm 2 Sarsa (On-policy TD Control) for Estimating $Q \approx q_*$

Require: Step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, except $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    while S is not terminal **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$
        $S \leftarrow S'$
        $A \leftarrow A'$
    end while
end for

Q-learning, on the other hand, is an *off-policy* algorithm that learns the value of the optimal po-

licy independently of the agent's behavior. It updates Q-values using the maximum estimated future reward from the next state, regardless of the action taken, thus encouraging convergence to the optimal action-value function. A possible pseudocode, extracted from the same book [18] of the others, is shown in Algorithm 3.

Algorithm 3 Q-learning (Off-policy TD Control) for Estimating $\pi \approx \pi_*$

Require: Step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, except $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialize S
    while S is not terminal **do**
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
        $S \leftarrow S'$
    end while
end for

Due to their simplicity, effectiveness, and adaptability, SARSA and Q-learning have become standard tools in reinforcement learning research and applications.

Moreover, the core principles of Temporal Difference learning serve as the foundation for more advanced deep reinforcement learning approaches, which will be discussed in the next sections.

3.8 Value Function Approximation

Until now, value functions $V(s)$ and action-value functions $Q(s, a)$ have been represented using lookup tables, where each state s or state-action pair (s, a) has a corresponding entry. This approach assumes that there is sufficient memory to store the entire state or state-action space. However, in real-world scenarios, the state and action spaces are often extremely large or even continuous, making tabular methods impractical. In such cases, the number of possible states and actions is too great to store explicitly, and the learning process becomes prohibitively slow, as it requires estimating the value for each individual state or state-action pair. Moreover, in many real-world tasks, the agent is likely to encounter novel states that it has never seen before, rendering table-based representations ineffective. To overcome these problems, function approximation methods are employed. These methods generalize from seen to unseen states by approximating the value functions using parameterized functions, such as linear models or neural networks, enabling reinforcement learning to scale to high-dimensional and continuous spaces. Formally, it is possible to write:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s) \tag{38}$$

$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a) \tag{39}$$

where $\hat{v}(s, \boldsymbol{w})$ and $\hat{q}(s, a, \boldsymbol{w})$ are approximated value functions parameterized by $w \in \mathbb{R}^d$. This parameter vector $\boldsymbol{w}$ becomes the target of the learning process, which can still be guided by Monte Carlo (MC) or Temporal-Difference (TD) learning methods. It is important to note that, with function approximation, an update to the value estimate at a single state can influence the estimates of many other states due to shared parameters. As a result, it is no longer possible

to represent the value of every state exactly. Moreover, since the number of states typically far exceeds the number of weights, improving the estimate for one state may come at the cost of reducing the accuracy for others. This trade-off leads to the definition of a state distribution, which determines the relative importance of each state and guides the learning process to focus on the most relevant regions of the state space.

Function approximation methods can be broadly categorized into two classes: incremental methods and batch methods. Incremental methods, such as those based on stochastic gradient descent (SGD), update the parameters w after each individual experience. This allows the agent to learn online and adapt in real time as it interacts with the environment. In contrast, batch methods collect a set of experiences, often referred to as a data set or replay buffer of experiences, and perform updates using the entire batch at once. Examples include least-squares approaches like Least-Squares Temporal Difference (LSTD) learning. Although batch methods can lead to more stable and data-efficient updates, they typically require more memory and computational resources.

3.9 Policy Gradient Methods

In the previous section 3.8, it was discussed how the value function or the action value function can be approximated using a set of parameters denoted by $\boldsymbol{w}$. From this approximation, a policy π was subsequently derived, based on the estimated value function. In contrast, policy gradient methods [19] adopt a different approach. Rather than deriving the policy from the value function, the policy itself is explicitly parameterized and optimized. This means that the learning process focuses directly on improving the policy by adjusting its parameters to maximize expected performance, without necessarily computing or relying on an intermediate value function. So, it is possible to write:

$$\pi(a \mid s, \boldsymbol{\theta}) = \Pr\left\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\right\} \tag{40}$$

where $\theta \in \mathbb{R}^{d'}$ denotes the vector that parametrizes the policy.

Policy-based reinforcement learning methods often exhibit better convergence properties and are particularly well-suited for environments with high-dimensional or continuous action spaces, as they naturally support the learning of stochastic policies. However, a major drawback of these methods is the inefficiency and high variance often associated with policy evaluation. In these methods, the policy gradient is typically estimated by computing the gradient of a performance objective, such as the expected return from a fixed initial state of an episode, with respect to the policy parameters. Formally:

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) \tag{41}$$

where s_0 is the initial state. Hence, the update rule takes the form

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J \tag{42}$$

To apply this update effectively, the policy must be appropriately parameterized and the gradient of the performance must be estimated.

A typical policy parametrization is the exponential softmax distribution:

$$\pi(a \mid s, \boldsymbol{\theta}) = \frac{\exp(h(s, a, \boldsymbol{\theta}))}{\sum_b \exp(h(s, b, \boldsymbol{\theta}))} \tag{43}$$

where $h(s, a, \boldsymbol{\theta})$ is the parametrized numerical preferences. In this way $\pi(a \mid s, \boldsymbol{\theta})$ is differentiable with respect to its parameters, $\nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta})$ exists and is always finite, and exploration is ensured.

Computing the gradient of the performance objective to adjust the policy parameters in a way that ensures improvement is a challenging task. This difficulty arises because performance depends not only on the selected actions, directly governed by the policy, but also on the distribution of the states, which is also influenced by the environment, the dynamics of which are often unknown. For this reason, the policy gradient theorem plays a fundamental role, as it provides an analytic expression for the gradient of performance with respect to the policy parameter that does not involve the derivative of the state distribution. For the episodic case, it establishes that:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_{\boldsymbol{\theta}} \pi(a \mid s, \boldsymbol{\theta}) \tag{44}$$

Where the constant of proportionality is the average length of an episode. In the continuing case, the relationship is actually an equality.

The policy gradient theorem provides the theoretical foundation for computing the gradient of the expected return with respect to the policy parameters. However, in practice, the exact computation of this gradient is intractable due to the need to evaluate expectations over all possible states and actions. This is where the REINFORCE algorithm comes into play.

3.10 REINFORCE

REINFORCE is a fundamental policy gradient algorithm in the field of reinforcement learning. It leverages the policy gradient theorem by using Monte Carlo sampling to estimate the gradient based on trajectories collected from interaction with the environment. By sampling states, actions, and returns, REINFORCE provides an unbiased estimate of the gradient, enabling the agent to improve its policy using only data gathered through experience. The update rule of the algorithm is done via stochastic gradient ascent, where the policy parameters are adjusted in the direction of the estimated gradients. To obtain this, the policy gradient theorem formula 44 needs to be further elaborated.

$$\begin{aligned}
\nabla J &\propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a \mid s, \boldsymbol{\theta}) \\
&= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a \mid S_t, \boldsymbol{\theta}) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a \mid S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a \mid S_t, \boldsymbol{\theta})}{\pi(a \mid S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta})}{\pi(A_t \mid S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t \mid S_t, \boldsymbol{\theta})}{\pi(A_t \mid S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta}) \right]
\end{aligned} \tag{45}$$

In this way, the expression in the brackets of 45 is exactly what is needed, a quantity that can be sampled on each time step whose expectation is equal to the gradient. So, the update rule becomes:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t) \tag{46}$$

The pseudocode of the algorithm is outlined in Algorithm 4.

Algorithm 4 REINFORCE: Monte-Carlo Policy-Gradient Control (Episodic) for π_*

Require: A differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Require: Step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., $\boldsymbol{\theta} \leftarrow 0$)
while Loop forever (for each episode) **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$
  for each step of the episode $t = 0$ to $T - 1$ **do**
    $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$
  end for
end while

Although REINFORCE is a powerful reinforcement learning algorithm, it has certain limitations that can affect its performance. One of the primary challenges is the high variance in its gradient estimates, which often results in slow convergence and an unstable learning process. This variance arises because the gradient is based on individual trajectories, making the updates noisy and less reliable. Moreover, REINFORCE does not directly address the exploration-exploitation trade-off, but it relies on the policy's exploration behavior, which is typically influenced by hyperparameters. Additionally, REINFORCE is highly sensitive to the learning rate, and improper tuning of this parameter can lead to either slow learning or instability in the learning process.

3.11 REINFORCE with Baseline

To tackle the problem of high variance in the REINFORCE algorithm, the policy gradient theorem can be extended by incorporating a comparison of the action value to an arbitrary baseline $b(s)$. The update rule becomes:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t - b(S_t)\right) \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta}_t) \tag{47}$$

The introduction of the baseline still leads to an unbiased estimate as long as it depends just on the state and it does not vary with the action. A natural choice for the baseline could be the estimate of the state value function. In this way, two loss terms are involved, the regular REINFORCE loss, with the learned value as a baseline, and the mean squared error between the learned value and the observed discounted return G_t. Mathematically:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \left(G_t - \hat{v}(S_t, \boldsymbol{w})\right) \nabla \log \pi(A_t | S_t, \boldsymbol{\theta}) \tag{48}$$

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} + \beta \left(G_t - \hat{v}(S_t, \boldsymbol{w})\right) \nabla \hat{v}(S_t, \boldsymbol{w}) \tag{49}$$

3.12 Deep Reinforcement Learning

Deep reinforcement learning (DRL) combines reinforcement learning (RL) with deep neural networks to enable agents to learn complex decision-making policies from high-dimensional inputs. Using the representational capacity of deep learning, DRL overcomes the limitations of traditional RL in large or unstructured state spaces. By incorporating deep neural networks as function approximators of value functions, policies, or environment models, DRL enables agents to process raw sensory data such as images, audio, or text directly and to learn general representations of states or state-action pairs.

The integration of deep learning with reinforcement learning has led to breakthroughs in domains that demand both perception and planning, including autonomous robotics, medical diagnosis and treatment planning, algorithmic trading, adaptive energy systems, and personalized education. Focusing on robotics, DRL has numerous applications, such as locomotion, navigation, manipulation, and interaction.

DRL can be applied to coordinate the movement of multiple joints and limbs, allowing robots to perform complex locomotion tasks such as walking, running, or balancing on unstructured terrain. Recent advances in this area have been particularly prominent in the field of reinforcement learning for humanoid robots [20]. Among recent developments, NVIDIA has demonstrated significant advances in cutting-edge frameworks that leverage deep reinforcement learning to train humanoid agents capable of agile and adaptive locomotion in physically simulated environments. These agents can learn highly dynamic behaviors, such as climbing stairs or recovering from perturbations, by training in simulation with physics-based models and transferring the learned policies to real-world scenarios through simulation-to-real techniques. Furthermore, other major contributions come from DeepMind, whose MuJoCo-based research on motor control and proprioceptive policy learning has set new benchmarks in continuous control tasks [21], and Boston Dynamics, which continues to push real-world deployment with agile, sensor-rich legged robots like Atlas. OpenAI has also explored domain randomization and robotic dexterity through DRL, which allows a robotic hand to manipulate objects with remarkable precision [22].

Despite its success, deep reinforcement learning algorithms face several challenges. The high-dimensional and continuous action spaces present in many real-world problems can hinder efficient exploration and slow down convergence. Moreover, training deep neural networks within the RL framework is computationally intensive and often sensitive to the choice of hyperparameters. To mitigate these issues, researchers have developed a variety of enhancements to DRL algorithms, including prioritized experience replay, trust region policy optimization, and distributional reinforcement learning. These improvements aim to promote more effective exploration, increase training stability, and enhance sample efficiency, making DRL more viable for complex real-world applications.

3.13 Value-based methods for deep RL

3.14 Deep Q-Network

The Q-learning algorithm introduced in Section 3.8 can be extended to real-world scenarios by employing neural networks as function approximators. The first major advancement in this direction is the Deep Q-Network (DQN), which replaces the tabular Q-function with a deep neural network to estimate action values. The DQN algorithm, introduced by Mnih et al. [23], is able to obtain strong performance in an online setting for a variety of ATARI games, directly by learning from the pixels. DQN retains the core principles of Q-learning but introduces several crucial features to improve its stability and convergence performance. In particular, it incorporates experience replay, which stores past experiences in a memory buffer and samples batches for training, helping to break the temporal correlations between consecutive updates. Furthermore, DQN utilizes a target network, a separate copy of the Q-network, which is updated less frequently to stabilize training by reducing the moving target problem. Finally, techniques such as batch normalization are employed to standardize inputs and improve the convergence of the neural network, making the DQN more robust to varying data distributions during training. A sketch of the algorithm is shown in Algorithms 5.

Algorithm 5 Deep Q-Network (DQN) with Experience Replay and Target Network

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment
while not converged **do** ▷ — Sample Phase —
  $\varepsilon \leftarrow$ setting new epsilon using ε-decay
  Choose action a from state s using policy ε-greedy(Q)
  Agent takes action a, observes reward r, and next state s'
  Store transition $(s, a, r, s', \texttt{done})$ in replay memory D
  if enough experiences in D **then** ▷ — Learn Phase —
    Sample a random minibatch of N transitions from D
    for all $(s_i, a_i, r_i, s'_i, \texttt{done}_i)$ in minibatch **do**
      if $\texttt{done}_i$ **then**
        $y_i \leftarrow r_i$
      else
        $y_i \leftarrow r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
      end if
    end for
    Compute loss $\mathcal{L} = \frac{1}{N} \sum_{i=0}^{N-1} \left(Q(s_i, a_i) - y_i\right)^2$
    Update Q by minimizing $\mathcal{L}$ using SGD
    if step % $C == 0$ **then** ▷ — Every C steps —
      Copy weights from Q to $\hat{Q}$
    end if
  end if
end while

While DQN is a powerful algorithm, it also has several limitations that can affect its performance and its applicability in the real world. The use of the maximum operator during action selection can introduce an overestimation bias, especially in environments with high variance or sparse rewards. This bias often leads to suboptimal policies and slower convergence rates. Additionally, the potential correlation between consecutive samples in the replay buffer violates the independent and identically distributed (i.i.d.) assumption, a key requirement for standard deep learning algorithms, which can further destabilize training. The non-stationary target network and the algorithm's sensitivity to hyperparameters also contribute to instability and slow convergence. Furthermore, DQN is primarily designed for discrete action spaces, relying on a discrete action-value function to estimate Q-values. Adapting DQN to continuous action spaces requires the use of techniques such as discretization or function approximation, which introduce additional challenges and limitations.

3.15 Double DQN

As mentioned earlier, in Q-learning the maximum operation uses the same values for both selecting and evaluating an action. This can lead to the selection of overestimated values, particularly in the presence of inaccuracies or noise, resulting in overly optimistic value estimates. As a result, the DQN algorithm introduces an upward bias. The Double Q-learning method, proposed by Hasselt (2010) [24], addresses this issue by using two separate estimates for each variable. This decouples the process of action selection from the value evaluation, helping to mitigate

the positive bias in Q-value estimation, regardless of whether errors arise from environmental stochasticity, function approximation, non-stationarity, or other factors. The algorithm, taken directly from the paper [24], is outlined in Algorithm 6.

Algorithm 6 Double Q-learning

Initialize Q^A, Q^B, and state s
repeat
  Choose action a based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$ (e.g., ε-greedy), observe r, s'
  Randomly choose either `UPDATE(A)` or `UPDATE(B)`
  if `UPDATE(A)` **then**
    Define $a^* \leftarrow \arg\max_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  else if `UPDATE(B)` **then**
    Define $b^* \leftarrow \arg\max_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\left(r + \gamma Q^A(s',b^*) - Q^B(s,a)\right)$
  end if
  $s \leftarrow s'$
until termination condition met

Although Double DQN improves upon the vanilla DQN, it still has its own limitations. It does not completely eliminate overestimation bias, requires more memory and computational resources, and can have slower learning. Additionally, it is still sensitive to hyperparameters and can be more difficult to tune.

3.16 Dueling Network Architecture

The dueling network architecture introduces a key modification to traditional Q-networks by separately estimating the state value function $V(s)$ and the advantage function $A(s,a)$. In standard Q-learning, the Q-value is computed by combining these two functions, which can lead to inefficient learning, particularly when many actions in a given state yield similar outcomes. The dueling architecture addresses this by splitting the network into two streams, one for estimating the state value and the other for estimating the action advantage. These two streams are then combined to compute the Q-value for each action, as shown in the following equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha)\right) \tag{50}$$

Where θ represents the shared parameters of the network, while α and β are the parameters for the advantage and value streams, respectively. This separation allows the agent to assess the value of a state more efficiently, regardless of the actions available. The dueling architecture improves learning efficiency by allowing the value stream to generalize across actions in states where the action choice has little impact, leading to faster convergence and more stable training, particularly in environments with large or redundant action spaces [25].

3.17 Distributional DQN

The methods discussed so far in this chapter focus on directly approximating the expected return through a value function. However, a more expressive alternative is to model the entire

distribution of possible cumulative returns, rather than just their expectation. This approach is known as distributional reinforcement learning, and it captures the variability and uncertainty inherent in the return values.

One of the earliest contributions to this field is by Morimura et al.(2010) [26], who proposed a non-parametric method for estimating return distributions. By maintaining and updating a full return distribution, agents can gain a deeper understanding of risk and uncertainty, which may be particularly beneficial in particular environments.

Although distributional methods, such as Distributional DQN, have shown promising results, they are not the focus of this work. For a more comprehensive treatment of these techniques, refer to [26] and related literature.

3.18 Policy gradient methods for deep RL

3.19 Actor-Critic

Actor-Critic methods are a class of reinforcement learning algorithms that combine the advantages of both policy-based and value-based approaches. In these methods, the agent consists of two key components: the actor and the critic. The actor is responsible for selecting actions based on the current policy and directly interacting with the environment. It aims to maximize the expected cumulative reward by learning a parametrized policy, usually a neural network, that maps states to actions. The critic, on the other hand, evaluates the quality of the actions chosen by the actor. It estimates the expected cumulative reward, or the value function, for a given state or state-action pair. Using value function approximation methods, such as a neural network, the critic predicts these values and provides feedback to the actor. This feedback helps guide the actor toward more optimal actions by providing feedback on the expected rewards. The interaction between the actor's policy updates and the critic's value estimation enables the agent to achieve better long-term performance.

Formally, as previously discussed in Section 3.10 the policy gradient is given by:

$$\nabla J = \mathbb{E}_\pi \left[G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t \mid S_t, \boldsymbol{\theta}) \right] \tag{51}$$

In the REINFORCE algorithm q_π was given by the complete return G_t. In this case, with Actor-Critic methods, q_π is now estimated using the critic network, which addresses the well-known policy evaluation problem. As already discussed before, to further reduce variance and enhance learning stability, the advantage function is often used. In particular, the temporal difference (TD) error, defined as:

$$\delta_{\pi_\theta} = r + \gamma v_{\pi_\theta}(s') - v_{\pi_\theta}(s) \tag{52}$$

serves as an unbiased estimator of the advantage function $A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$. Hence, the policy gradient can be computed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[G_t \nabla_{\boldsymbol{\theta}} \ln \pi(a \mid s, \theta) \delta_{\pi_\theta} \right] \tag{53}$$

In practice, the critic network is trained to approximate the value function $v_{\pi_\theta}(s)$, and the TD error δ_{π_θ} is used to estimate the advantage, which drives the policy update. A possible implementation is presented in Algorithm 7.

Algorithm 7 One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Require: Differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Require: Differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Require: Step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to 0)
while Loop forever (for each episode) **do**
    Initialize S (first state of the episode)
    $I \leftarrow 1$
    while S is not terminal (for each time step) **do**
        Sample action $A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ ▷ If S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \, \delta \, \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \, \delta \, \nabla_{\boldsymbol{\theta}} \log \pi(A \mid S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
    end while
end while

Furthermore, both the actor and the critic can estimate their respective functions using targets derived from various time scales, such as Monte Carlo returns, TD(0), or TD(λ) methods.

Actor-Critic methods also come with several notable limitations. One of the primary challenges is the high variance in the policy gradient updates. Since the actor relies on the critic's estimated value functions to guide its policy updates, any inaccuracy or instability in the critic's estimates can propagate to the actor. This dependence often results in significant fluctuations during learning, potentially leading to unstable convergence. Moreover, the use of function approximation to estimate the value function or the advantage function can introduce bias into the policy gradient computation. If the approximation is inaccurate, the gradient direction may deviate from the true optimal update. Therefore, it is crucial to design or select value function approximators that achieve a balance between bias and variance. In addition, Actor-Critic algorithms can be highly sensitive to the choice of hyperparameters, such as learning rates for both the actor and critic, the discount factor γ, and any parameters governing exploration. Finally, these methods may require a large number of environment interactions to learn good policies, especially in complex scenarios.

In the following sections, several widely used algorithms that build upon the Actor-Critic framework are presented, such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC). These methods maintain the core actor-critic architecture but they also incorporate additional strategies to improve learning stability, performance, and efficiency.

3.20 Trust Region Policy Optimization - TRPO

Trust Region Policy Optimization (TRPO) [27] is a policy optimization algorithm designed to improve the stability and sample efficiency of reinforcement learning and it can be applied to environments with discrete action spaces and, with specific adjustments, can also be used for continuous action spaces. Unlike traditional policy gradient methods, which can lead to large

updates and unstable performance, TRPO introduces a constraint on the size of policy updates to maintain stable improvement. This is achieved by enforcing a trust region using the Kullback-Leibler (KL) divergence between the old and new policies.

The key idea behind TRPO is the optimization of a surrogate objective function under the constraint that the KL divergence between the old and new policies does not exceed a threshold. This ensures that the policy update is small enough to avoid performance degradation. Keeping policy updates within a trust region that ensures a certain level of improvement leads to more stable and controlled policy optimization.

The surrogate objective function $L_{\text{PG}}(\theta)$ is given by:

$$L_{\text{PG}}(\theta) = \mathbb{E}_t \left[\frac{\pi(\cdot|s_t;\theta)}{\pi_{\text{old}}(\cdot|s_t)} A^{\pi_{\text{old}}}(s_t, a_t) \right] \tag{54}$$

where $A^{\pi_{\text{old}}}(s_t, a_t)$ is the advantage function, defined as $r_t + \gamma V(s') - V(s)$, π_{old} is the policy with old parameters, $\pi(\cdot \mid s;\theta)$ is the policy with updated parameters, δ is the trust region radius, and α is the step size for the update.

The update step for TRPO is constrained by the KL divergence:

$$\text{maximize } L_{\text{PG}}(\theta) \quad \text{subject to} \quad \text{KL}[\pi_{\text{old}}(\cdot|s) \parallel \pi(\cdot|s;\theta)] \leq \delta \tag{55}$$

Where KL is the Kullback-Leibler divergence. So θ update results:

$$\theta_{\text{new}} = \theta + \alpha \cdot \arg\max_{\delta\theta} L_{\text{PG}}(\theta + \delta\theta) \tag{56}$$

This constraint ensures that the new policy π does not deviate too far from the old policy π_{old}, maintaining stability while optimizing performance. TRPO has shown strong empirical results in various domains, such as robotic locomotion and Atari game playing, with minimal hyperparameter tuning. Furthermore, it can be asserted that the objective will improve monotonically, as long as the KL divergence constraint is adhered to.

However, TRPO comes with some drawbacks. One of the main challenges is the computationally expensive and complex task of estimating the Kullback-Leibler (KL) divergence between the old and updated policies. Estimating this divergence between two probability distributions typically requires either sampling or numerical integration methods, both of which can be computationally intensive and may not scale well in environments with large state and action spaces. Accurate estimation of the KL divergence is critical because incorrect estimates can result in too conservative or too aggressive policy updates.

3.21 Proximal Policy Optimization - PPO

Proximal Policy Optimization (PPO) [3] is an improvement of Trust Region Policy Optimization that addresses some of its limitations and offers several advantages. Unlike TRPO, which relies on a strict KL divergence constraint to limit policy updates, PPO simplifies this by using a clipped surrogate objective function. This function directly constrains the policy update within a pre-defined threshold, making it easier to implement and computationally more efficient.

Formally:

$$L_{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{57}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the updated policy and the old policy, $\hat{A}_t$ is an estimator of the advantage function at timestep t, and ϵ is a hyperparameter that controls the clipping range. The clipping mechanism ensures that the policy update remains within a trust region, thus preventing too large updates that can destabilize learning. Figure 7 provides another intuition of the surrogate objective L_{CLIP}.

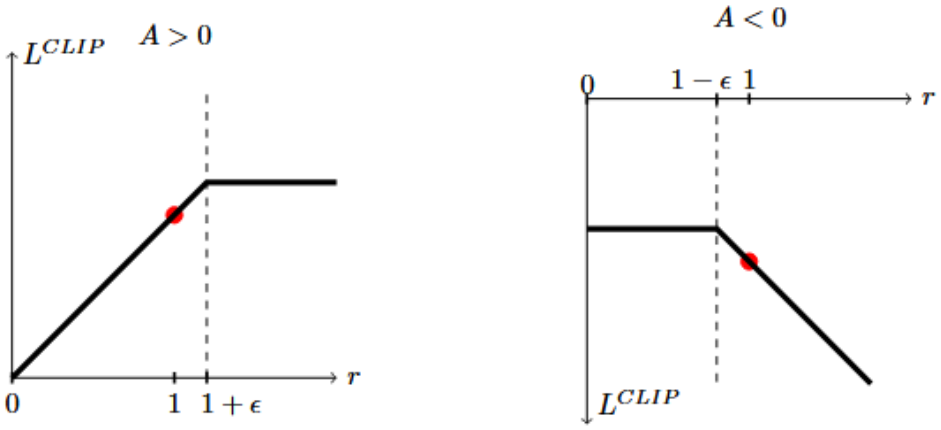


Figure 7: Plots showing a single timestep of the surrogate function L_{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization [3].

Additionally, PPO improves sample efficiency by reusing collected data across multiple iterations, allowing the policy to be updated several times per iteration and enabling it to learn more efficiently with fewer interactions with the environment. PPO also improves stability by addressing the high variance in policy updates that may arise in TRPO with the clipped surrogate objective function. This prevents large and destabilizing policy changes. The increased stability makes PPO less sensitive to hyperparameters and easier to tune. Moreover, PPO offers more flexibility in step size choices, avoiding the computationally expensive line search used in TRPO. The implementation of the PPO-clip algorithm is presented in Algorithm 8.

Algorithm 8 PPO-Clip

Require: Initial policy parameters $\boldsymbol{\theta}_0$, initial value function parameters $\boldsymbol{\phi}_0$

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: Collect a set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\boldsymbol{\theta}_k)$ in the environment

3: Compute rewards-to-go $\hat{R}_t$

4: Compute advantage estimates $\hat{A}_t$ using any suitable method, based on the current value function $V_{\boldsymbol{\phi}_k}$

5: Update the policy by maximizing the PPO-Clip objective:

$$\boldsymbol{\theta}_{k+1} = \arg\max_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}_k}(a_t \mid s_t)} A^{\pi_{\boldsymbol{\theta}_k}}(s_t, a_t),\ g(\epsilon, A^{\pi_{\boldsymbol{\theta}_k}}(s_t, a_t)) \right)$$

▷ typically optimized via stochastic gradient ascent (e.g., Adam)

6: Fit value function by regression on mean-squared error:

$$\boldsymbol{\phi}_{k+1} = \arg\min_{\boldsymbol{\phi}} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_{\boldsymbol{\phi}}(s_t) - \hat{R}_t \right)^2$$

▷ typically optimized via gradient descent

7: **end for**

3.22 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [28] is a model-free, off-policy reinforcement learning algorithm specifically designed for environments with continuous action spaces. It follows an actor-critic architecture and extends the Deterministic Policy Gradient (DPG) method by integrating key ideas from Deep Q-Networks (DQN). While DQN performs well in high-dimensional observation spaces, it is limited to discrete and low-dimensional action spaces. This is because DQN relies on selecting the action that maximizes the action-value function, a process that becomes computationally impractical in continuous domains, where it would require solving an iterative optimization problem at every step. However, DQN is able to learn value functions using large, non-linear function approximators in a stable and robust way due to two particular innovations, the network is trained off-policy with samples from a replay buffer to minimize correlations between samples, and the network is trained with a target Q network to give consistent targets during temporal difference backups. For other details, see previous section 3.14. DDPG makes use of the same ideas along with batch normalization.

Diving deeper in the core structure, DDPG computes the gradient of the expected return with respect to the deterministic policy [29]:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \right] \tag{58}$$

The actor network thus learns a deterministic policy $\mu(s|\theta^\mu)$ (using μ instead of π to remain consistent with the original paper's notation). The deterministic gradient is preferred over a stochastic one because it requires integration over the state space only, rather than over both the state and action spaces.

To stabilize learning, DDPG employs soft updates for target networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{59}$$

where $\tau \ll 1$ is a small constant. These slow updates ensure that the targets used in the critic's temporal difference (TD) learning are relatively stable over time. Furthermore, DDPG uses batch normalization to normalize inputs across mini-batches, which helps the networks handle input features with varying scales. For exploration in continuous spaces, noise is added to the actor's output:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \tag{60}$$

This temporally correlated noise facilitates efficient exploration in environments with momentum and inertia. The implementation of the DDPG algorithm is shown in Algorithm 9.

Algorithm 9 DDPG Algorithm

Randomly initialize critic network $Q(s, a \mid \boldsymbol{\theta}^Q)$ and actor $\mu(s \mid \boldsymbol{\theta}^\mu)$ with weights $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\mu$
Initialize target networks Q' and μ' with weights $\boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^Q$, $\boldsymbol{\theta}^{\mu'} \leftarrow \boldsymbol{\theta}^\mu$
Initialize replay buffer $\mathcal{R}$
for episode = 1 to M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for $t = 1$ to T **do**
        Select action $a_t = \mu(s_t \mid \boldsymbol{\theta}^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute a_t, observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
        $y_i \leftarrow r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \boldsymbol{\theta}^{\mu'}) \mid \boldsymbol{\theta}^{Q'})$
        Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i \Big(y_i - Q(s_i, a_i \mid \boldsymbol{\theta}^Q)\Big)^2$$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\boldsymbol{\theta}^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a \mid \boldsymbol{\theta}^Q)\big|_{s=s_i, a=\mu(s_i)} \nabla_{\boldsymbol{\theta}^\mu}\mu(s \mid \boldsymbol{\theta}^\mu)\big|_{s_i}$$

        Update the target networks:

$$\boldsymbol{\theta}^{Q'} \leftarrow \tau\boldsymbol{\theta}^Q + (1-\tau)\boldsymbol{\theta}^{Q'}$$

$$\boldsymbol{\theta}^{\mu'} \leftarrow \tau\boldsymbol{\theta}^\mu + (1-\tau)\boldsymbol{\theta}^{\mu'}$$

    end for
end for

While Deep Deterministic Policy Gradient (DDPG) is a powerful algorithm for continuous control, it is not without significant limitations and challenges. First, DDPG critically relies on a well-learned estimate of the action-value function $Q(s, a)$. Unlike REINFORCE-based methods

that use sampled trajectories from the environment to estimate the gradient of the expected return with respect to policy parameters, DDPG computes this gradient through backpropagation via the critic through 58. This implies that the quality of the policy updates is entirely dependent on how accurately the critic estimates $Q(s, a)$. If the critic is poorly trained or unstable, the actor may follow misleading gradients, which can significantly degrade performance or destabilize training.

Furthermore, in deterministic actor-critic settings such as DDPG, the overestimation of $Q(s, a)$ becomes a critical issue. Unlike DQN, where the relative ranking of Q-values is more important than their absolute values, DDPG uses the actual magnitude and gradient of $Q(s, a)$ with respect to actions. Hence, if $Q(s, a)$ is overestimated in regions of the action space, it can lead the policy to exploit erroneous regions, resulting in degraded or diverging performance.

Another key challenge lies in the non-convex nature of the learned action-value function $Q(s, a)$ with respect to actions, that can be highly non-linear and contain multiple local optima, saddle points, or regions with flat gradients. This non-convexity makes gradient-based updates of the actor prone to getting stuck in local optima.

Finally, because DDPG learns a deterministic policy, it requires external noise to drive exploration. This approach is often sensitive to hyperparameter tuning. Thus, DPG can easily converge prematurely to suboptimal deterministic strategies if exploration is inadequate. These limitations have led to the development of more robust alternatives, such as Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) 3.14, which addresses several of DDPG's shortcomings by introducing entropy-regularized policy learning.

3.23 Soft Actor Critic - SAC

Soft Actor-Critic (SAC) [30] is an off-policy actor-critic algorithm built upon the maximum entropy reinforcement learning framework. In this framework, the objective is not only to maximize the expected cumulative reward but also to maximize the entropy of the policy, encouraging exploration by favoring stochastic policies that remain uncertain when multiple actions appear similarly advantageous. SAC improves the deterministic actor-critic paradigm of DDPG by incorporating architectural enhancements introduced in Twin Delayed Deep Deterministic Policy Gradient (TD3), such as employing two Q-networks and using the minimum of their outputs to form a more conservative and robust target value estimate. Additionally, noise is injected into the target action of the Q-learning update to smooth the value estimates and further prevent overestimation. The key innovation in SAC is the addition of an entropy bonus to the policy objective, resulting in a soft Q-function that guides both exploration and exploitation. Formally:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)) \right] \tag{61}$$

where $\mathcal{H}(\pi(\cdot \mid s_t))$ denotes the entropy of the policy at state s_t and α is a temperature parameter that governs the balance between reward maximization and entropy, thus controlling the stochasticity of the optimal policy. This leads to the modeling of both the mean $\mu(s)$ and standard deviation $\sigma(s)$ of a Gaussian policy. To do so, the reparameterization trick is employed, in order to enable efficient and differentiable sampling from this distribution, allowing the agent to learn both the mean and variance of the action distribution through backpropagation.

The reparameterization trick enables sampling from the stochastic policy in a way that is diffe-

rentiable with respect to the policy parameters. Instead of sampling the action a_t directly from the distribution $\pi(a_t \mid s_t) = \mathcal{N}(\mu(s_t), \sigma(s_t)^2)$, the action is expressed as a deterministic function of a noise variable $\epsilon \sim \mathcal{N}(0, I)$:

$$a_t = \mu(s_t) + \sigma(s_t) \odot \epsilon, \tag{62}$$

where $\odot$ denotes element-wise multiplication. This formulation allows gradients to propagate through the sampling operation, enabling the actor network to be trained using standard gradient descent techniques, thus reducing the variance of gradient estimates and improving training stability. Algorithm 10 shows a possible implementation of the Soft Actor-Critic method. However, for a comprehensive description of the details, the reader is referred to the original work by Haarnoja et al. [30].

Algorithm 10 Soft Actor-Critic (SAC)

Require: Initial policy parameters $\boldsymbol{\theta}$, Q-function parameters $\boldsymbol{\phi}_1$, $\boldsymbol{\phi}_2$, empty replay buffer $\mathcal{D}$

Set target Q-function parameters: $\boldsymbol{\phi}_{\text{target},1} \leftarrow \boldsymbol{\phi}_1$, $\boldsymbol{\phi}_{\text{target},2} \leftarrow \boldsymbol{\phi}_2$
repeat
  Observe state s and select action $a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)$
  Execute action a in the environment
  Observe next state s', reward r, and done signal d indicating if s' is terminal
  Store transition (s, a, r, s', d) in replay buffer $\mathcal{D}$
  if s' is terminal **then**
    Reset the environment
  end if
  if it's time to update **then**
    for j in range(number of updates) **do**
      Randomly sample a batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for Q-functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\boldsymbol{\phi}_{\text{target},i}}(s', a') - \alpha \log \pi_{\boldsymbol{\theta}}(a' \mid s')\right), \quad a' \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s')$$

      for $i = 1, 2$ **do**
        Update Q-function by minimizing:

$$\nabla_{\boldsymbol{\phi}_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\boldsymbol{\phi}_i}(s, a) - y(r, s', d)\right)^2$$

      end for
      Update policy by maximizing:

$$\nabla_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\boldsymbol{\phi}_i}(s, \tilde{a}_{\boldsymbol{\theta}}(s)) - \alpha \log \pi_{\boldsymbol{\theta}}(\tilde{a}_{\boldsymbol{\theta}}(s) \mid s)\right)$$

      ▷ $\tilde{a}_{\boldsymbol{\theta}}(s)$ is a sample from $\pi_{\boldsymbol{\theta}}(\cdot \mid s)$ which is differentiable wrt $\boldsymbol{\theta}$ using the reparameterization trick
      for $i = 1, 2$ **do**
        Update target Q-networks:

$$\boldsymbol{\phi}_{\text{target},i} \leftarrow \rho \boldsymbol{\phi}_i + (1 - \rho)\boldsymbol{\phi}_{\text{target},i}$$

      end for
    end for
  end if
until convergence

4 Project Setup and Data Preparation

4.1 Introduction

In the following chapters, we will transition from the theoretical foundations to the practical implementation of the project. The initial phase will focus on the dataset collection and preprocessing stages 4.2, where we will present the methodologies and tools used to prepare the data for subsequent analysis. In this section, we will also introduce the contributions of Adriano Liso, a PhD student who collaborated significantly in this aspect of the project 4.3.

After setting up the data, we will define the essential components of the reinforcement learning frameworks used throughout this research. Hence, the following chapters will be organized into two primary learning scenarios: the discrete framework 5 and the continuous frameworks 6. In the discrete learning scenario, both the environment and the agent operate within a discrete state and action space. We will present the algorithms specifically designed and optimized for this setting, highlighting the main features, strengths, and limitations.

In contrast, the continuous learning scenario deals with environments and agents that operate in continuous spaces. This chapter will introduce algorithms specifically designed to manage the complexities of continuous spaces, highlighting the unique advantages they offer, as well as the challenges introduced by the increased flexibility and complexity of such environments.

Together, these two learning scenarios enables the exploration of both offline and online training paradigms. In the discrete scenario, the RL agent is trained offline using previously collected field data, where the environment is modeled as a grid map with each cell containing GPR-derived features. This allows for safe and rapid policy development. Following this, the project transitions to an online continuous learning framework, where the agent still interacts with a simulated environment, but now adapts to new, unseen soil conditions in real-time.

Finally, we will extend the continuous learning problem by considering potential real-world applications 7. In this chapter, we will discuss practical considerations of applying the framework to real-life scenarios, particularly in environments where obstacles are present. We will consider challenges related to sensor fusion, integrating both subsurface and surface-level information, and explore the transition from position-based control to velocity-based control. We will then conclude by highlighting opportunities for future research and development in this area.

4.2 Dataset

In order to effectively train reinforcement learning agents, it is crucial to construct a large and representative dataset. Deep reinforcement learning algorithms, in particular, require large volumes of samples and experience in order to properly generalize and converge to optimal policies. Furthermore, given the dynamic and heterogeneous nature of agricultural environments, characterized by varying soil types, moisture levels, and seasonal changes, a comprehensive set of measurements is needed.

To capture this variability, an extensive data collection was performed, using a ground penetrating radar system. In particular, two types of shielded antennas were used during the data acquisition process, both developed by MALA [31], the MALA Shielded $500MHz$ antenna and the MALA Shielded $800Mhz$ antenna (see Figure 8). The former offers greater penetration depth, which is suitable for detecting deeper soil features, the latter, instead, provides higher re-

solution data, useful for identifying finer surface level variations. By combining measurements from both antennas, it is possible to achieve a proper balance between depth and resolution.

(a) MALA Shielded $500Mhz$ Antenna

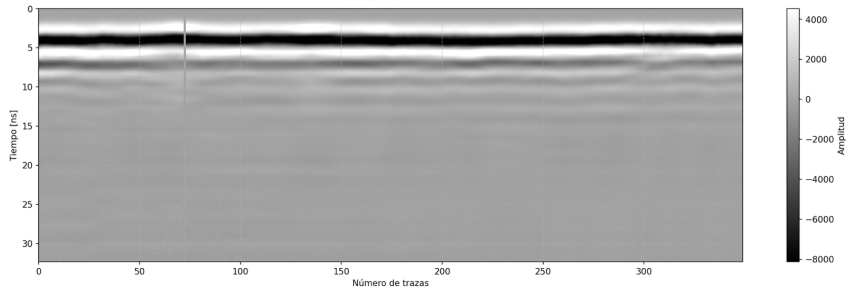
(b) MALA Shielded $800Mhz$ Antenna

Figure 8: MALA Schielded Antennas Overview

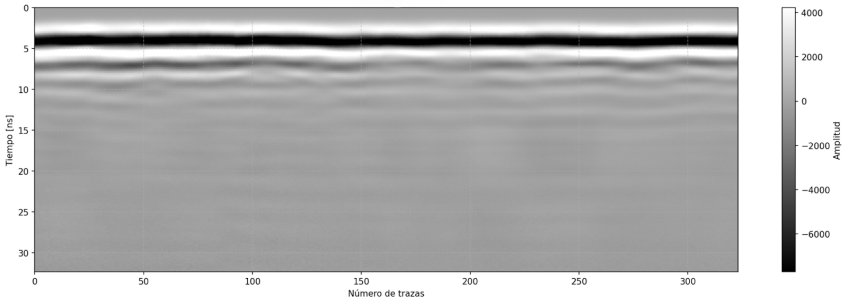
It is also important to define the time window, which defines the amount of time the receiver antenna listens for and records the returning signal after a pulse is transmitted. In other words, it corresponds to the recording duration of a single A-scan. Defining this parameter is important, as it directly affects the length of time the receiver is capturing data , hence the depth of penetration achievable during the scan. A longer time window allows the system to capture the signal reflected from deeper targets, while a shorter window is more suitable for detecting shallow features with higher temporal resolution. For our measurements, the chosen time window is set to $t = 43.117741$ ns.

Multiple scans were conducted with both antennas under different environmental conditions and across a range of field locations, trying to catch a wide range of different soil types and surface properties under different environmental and weather conditions. Both A-scan and B-scan data were collected during this acquisition process. The A-scan measurements were obtained by keeping the antenna fixed at a single location, thus capturing a recurring precise time-domain response from the subsurface at that specific point. To be precise, the resulting radargram is still a B-scan, but it is composed of repeated A-scan responses from the same spatial location, since the antenna is fixed. In contrast, B-scan data were acquired by continuously moving the antenna along a predefined path, hence generating a two-dimensional radargram that provides spatially continuous information over a local area. This combined acquisition approach augments the dataset, which now includes both highly localized reflections and broader subsurface profiles, leading to effective feature learning and improved model performance during training.

Some representative figures illustrating examples of the A-scan and B-scan measurements collected during these tests are presented here to provide a better understanding of the acquired data. By visualizing these scans, we aim to highlight the variability and complexity of the subsurface responses, which is crucial for understanding the structure of the data and its relevance to the model's learning process. Figures 9a–9b show examples of A-scans, while Figures 10a–10b present examples of B-scans. As mentioned above, A-scans are presented in a B-scan format since the same GPR settings are used, but the antenna is kept fixed at a specific location. The results is a B-scan radargram composed of repetitive A-scan measurements take from the same position over time.

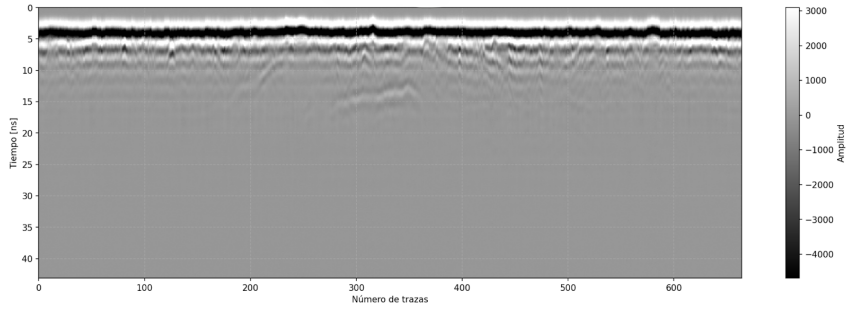


(a)

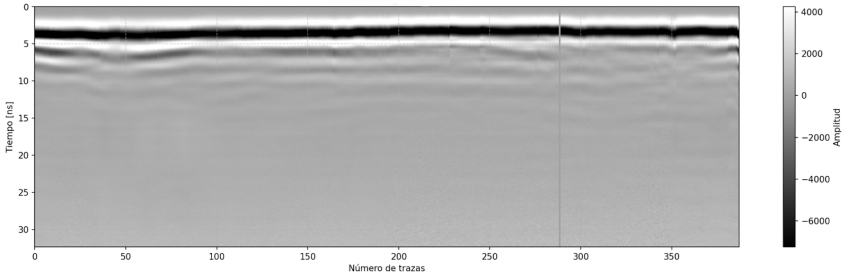


(b)

Figure 9: A-scan examples



(a)



(b)

Figure 10: B-scan examples

To establish ground truth data for the training and evaluation of the neural network specifically developed to extract soil moisture content from raw GPR scans (see Section 4.3), several field samples were collected immediately after each radar acquisition and subsequently subjected to a drying process under controlled conditions. In this way, the water content was accurately measured, allowing for a direct comparison with the predicted values of the neural network. Figure 11 shows an example of samples collection of different field types, Figure 12 shows the drying process under controlled conditions.

Figure 11: Example of different field samples

Figure 12: Drying process under controlled conditions

In conclusion, the resulting dataset includes numerous GPR B-scan radargrams obtained with two different shielded antennas to achieve a proper balance between penetration depth and spatial resolution. Before using this dataset to train the RL agent, a dedicated preprocessing pipeline must be applied. In particular, soil moisture features are extracted, using a dedicated neural network 4.3. This preprocessing and feature extraction step is crucial, as it significantly reduces the complexity of the environment in which the agent must operate and learn. Indeed, instead of learning directly from raw two-dimensional radar images, the reinforcement learning agent is provided with normalized soil moisture values in the range $[0, 1]$. This compact and semantically meaningful representation not only simplifies the input space, but also contributes

to a more stable and efficient learning process, as the handling of noisy and high-dimensional image data is delegated to a separate neural network specifically developed for this task 4.3.

4.3 Features Extraction

After the dataset has been constructed, the next step involves extracting meaningful features from the raw scan data. To do so, a tailored neural network architecture was designed and implemented. It consists of a Convolutional Autoencoder (CAE), a particular type of autoencoder that leverages convolutional neural networks (CNNs) for feature extraction and reconstruction, particularly well suited for image data [32, 33].

The network is composed of two primary components, an encoder and a decoder. The encoder progressively compresses the high-dimensional input scans into a lower-dimensional latent space representation by applying convolutional filters and downsampling operations. This latent space captures the most important features of the input data, leading to an effective dimensionality reduction while still preserving the essential structural information [34]. Following this, the decoder attempts to reconstruct the original input from this latent representation by using convolutional transpose layers, or upsampling, to restore the spatial dimensions. Figure 13 gives a general overview of this architecture.

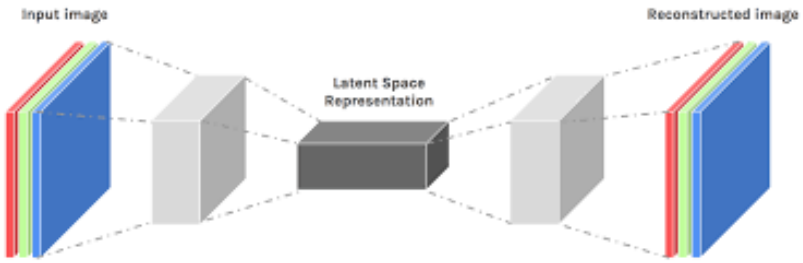


Figure 13: General Structure of the Convolutional Autoencoder, image retrieved online

This architecture not only enables efficient feature extraction, but also enforces a meaningful embedding of the raw input signals, which can be used as a new input for many other different tasks [32, 35]. The ability of the CAE network to learn hierarchical and spatially invariant meaningful features makes it an ideal choice for our application, where GPR scans must be compressed into a lower-dimensional latent space to extract representative and robust features.

To guide the network in learning the desired features from the input data, a composite loss function was employed, combining three distinct components. Each loss compares the reconstructed output of the decoder with the original input, scan but at different scales within the network, to ensure that the latent space representation preserves essential information. The first component is content loss, implemented as a mean squared error (MSE) between the encoded latent vector and a set of ground truth parameters. This loss directly supervises the encoder, encouraging the latent space to encode physically meaningful features. The second component is the reconstruction loss, also based on MSE, which penalizes pixel-wise differences between the input and the reconstructed output, ensuring the network learns faithful representation of the GPR scans given in input. The third component is an intermediate feature consistency loss, which compares the intermediate activations of selected encoder and decoder layers using MSE. This loss

enforces a symmetric information flow and preserves hierarchical structures at multiple scales. By jointly optimizing these three losses, the convolutional autoencoder is trained not only to reconstruct the input data, but also to produce a structured and interpretable low-dimensional embedding well suited for the following tasks. In particular, the latent representation learned by this network is incorporated into the input space of the Reinforcement Learning frameworks, providing a compact and informative feature set that allows agents to learn more efficiently and focus on the most relevant aspects of the environment.

This is a brief introduction to the principles underlying the network. For further details, see the forthcoming work by Liso, who has primarily led the development of this architecture [36].

4.4 Reinforcement Learning Framework Formalization

We now formally introduce and define the reinforcement learning frameworks that will be the foundation for the development of the algorithms. As outlined in previous chapters, the primary objective is to design Deep Reinforcement Learning agents capable of performing autonomous path planning and navigation by leveraging both surface and subsurface information. Specifically, the agent must not only reach a predefined target location but also adapt its path based on the soil moisture characteristics of the terrain it navigates. This dual objective allows us to integrate traditional mobile robot navigation with agricultural goals. This becomes particularly useful for precision agriculture applications, where navigation decisions based on environmental features can significantly improve efficiency and sustainability.

Diving deeper, different soil-aware navigation behaviours can be introduced. A potential learning setup could prioritize following paths with higher soil moisture, which can be valuable for tasks like targeted irrigation or planting where water availability is critical. However, it is equally important to develop agents capable of avoiding moist or water-saturated regions, especially in cases where the robot's physical constraints, such as limited torque, make it difficult to escape from these areas once entered. Traversing such wet zones may require excessive effort, increasing energy consumption, or even risking getting stuck.

Both scenarios are highly relevant in real-world applications. Thanks to the flexibility of RL settings, it is possible to address either case with minimal modifications. In this work, we mainly focus on the latter challenge, training agents to prefer drier paths and avoid high-moisture areas. This choice is motivated by the physical constraints of the differential-drive mobile robot platform we are working with, and, in particular, its restricted wheel torque capabilities, which limit its ability to effectively navigate through wet or water-saturated areas. Therefore, our priority is to ensure that the agent can reliably complete its agricultural tasks without getting stuck in the field, establishing a foundation that can later be extended to other objectives such as moisture-seeking behaviours.

5 Discrete Reinforcement Learning Framework

In this chapter, we formalize the discrete reinforcement learning framework, where both the state and the action spaces are discrete. The environment is modeled as a grid of predefined size, and the agent acts on it using a finite set of possible actions. Although this setup may appear simpler than the real-world applications, it provides a solid foundation for developing and testing the core properties of the algorithms. Furthermore, it allows us to identify key challenges and advantages associated with the general agricultural framework, while also serving as a starting point for the more complex continuous framework.

5.1 State Space

In a Reinforcement Learning problem, the state represents the set of information available to the agent at any given time. When the state provides all the necessary information for the agent to make an optimal decision, the environment is said to be fully observable, and the problem satisfies the Markov property. In this case, the decision-making process can be modeled as a Markov Decision Process 3.4, where the next state depends only on the current state and action, independently of the history of previous steps.

In contrast, if the state does not give all the information that the agent needs to evaluate the action, the problem is considered partially observable. These scenarios are typically modeled as Partially Observable Markov Decision Processes. In such cases, the agent may require a memory of past observations or actions to infer a better understanding of the current situation.

However, in some cases, it is reasonable to simplify the problem by assuming that the order in which past events occurred is not important. This allows us to reduce the history from a time-dependent sequence to an unordered set of relevant past observations or actions, potentially simplifying the learning problem from a partially observable MDP to a fully observable MDP, thus making the training more efficient and effective while still preserving essential contextual information.

The state representation in the proposed framework is composed of several key elements, each providing essential information for the agent's decision-making process. Specifically, the state includes the current coordinates position (x_r, y_r) of the agent on the grid, the coordinates (x_g, y_g) of the goal position, and a partial observation of the environment's soil moisture map, encoded as a latent grid.

To be precise, all the latent grid of the environment is encoded in the state space representation, but just a mask grid is actually presented to the agent. So, it perceives only a localized region defined by this mask, which limits its observation to the surrounding area within a certain range. This masked view allows the agent to reason about the soil moisture content in its neighborhood without having global knowledge of the terrain, thus resembling real-world robotic navigation scenarios, where onboard sensors provide only limited local environmental data. Furthermore, such partial observability encourages better generalization in the learning process, as the agent is primarily focused on following the underlying moisture features rather than just relying on the shortest path to the goal. This full and masked environment duality comes from our choice to operate in a static-offline setting, where the soil moisture values of the field have been extracted beforehand and remain constant during the robot's navigation. While this is a simplification of real-world dynamics, where moisture levels may vary over time, it provides a practical and

controlled starting point for developing and evaluating the agent's behaviour.

In summary, the state space consists of two coordinate values for the absolute robot position in the grid, two coordinate values for the target goal position, and of a ($\texttt{MASK SIZE} \times \texttt{MASK SIZE}$) flattened array with water content values of the local mask centered around the agent. The mask size is an important hyperparameter, as it directly influences the agent's perception and its ability to generalize across different terrains. This parameter will be subject to careful tuning during the training process.

5.2 Latent Grid

As introduced in the previous section 5.1, the latent grid representation of the field is one of the key components of the reinforcement learning framework, effectively forming part of the underlying environment within which the agent operates. In this section, we provide a more detailed explanation of how these latent grids are constructed and embedded within the custom environment.

The process begins with the latent space embeddings produced by the Convolutional Autoencoder, as discussed in Section 4.3. These embeddings capture essential features of the subsurface profiles derived raw B-scan measurements. Among the features encoded in the latent representation, water content values are decoded from the latent vectors and then exported as scalar real values, normalized in the range $[0, 1]$. Then , to construct a spatially meaningful representation of the environment, the entire field discretized using a configurable resolution parameter, and each decoded water content value is assigned to its corresponding spatial cell, based on the relative position. The result is a two-dimensional matrix $\mathcal{G} \in \mathbb{R}^{(\text{SIZE}\times\text{SIZE})}$, where each entry $\mathcal{G}_{i,j} \in [0, 1]$ denotes the water quantity level in the specific cell located at position (i, j). Values closer to 1 indicate higher levels of water content.

The hyperparameter `SIZE` defines the resolution of the field representation and it varies across different learning processes in order to evaluate the robustness and generalization capabilities of the agents. Indeed, by training and evaluating the same agent on environments with different grid sized, we can systematically test the scalability and adaptability of the learned policies to different spatial complexities.

It is important to note that, since this framework is still in its foundational stage, several adjustments have been introduced to the water content fields in order to refine the learning setup. These modifications are designed to enhance potential gaps in soil moisture values and simplify some particular complex scenarios. The goal is to ensure that any failure in agent performance is due to learning limitations rather than excessive environmental complexity. By idealizing certain aspects of the field structure, we create more interpretable environments.

Specifically, additional dry paths have been added on top of the original water content fields to prevent the agent from getting entirely stuck in saturated areas all the time. Furthermore, dry and wet zones have been randomly incorporated to simulate more realistic and diverse soil conditions while still preserving the original field composition.

Figure 14 presents representative examples of the original latent grids extracted from the original dataset. Values closer to 1 indicate higher levels of soil moisture, while values approaching 0 represent drier regions with lower water content. Furthermore, Figure 15 showcases enhanced

latent grids, where variations in spatial composition and water distribution can be observed. These enhancements give a clearer understanding of moisture patterns and their localized differences.

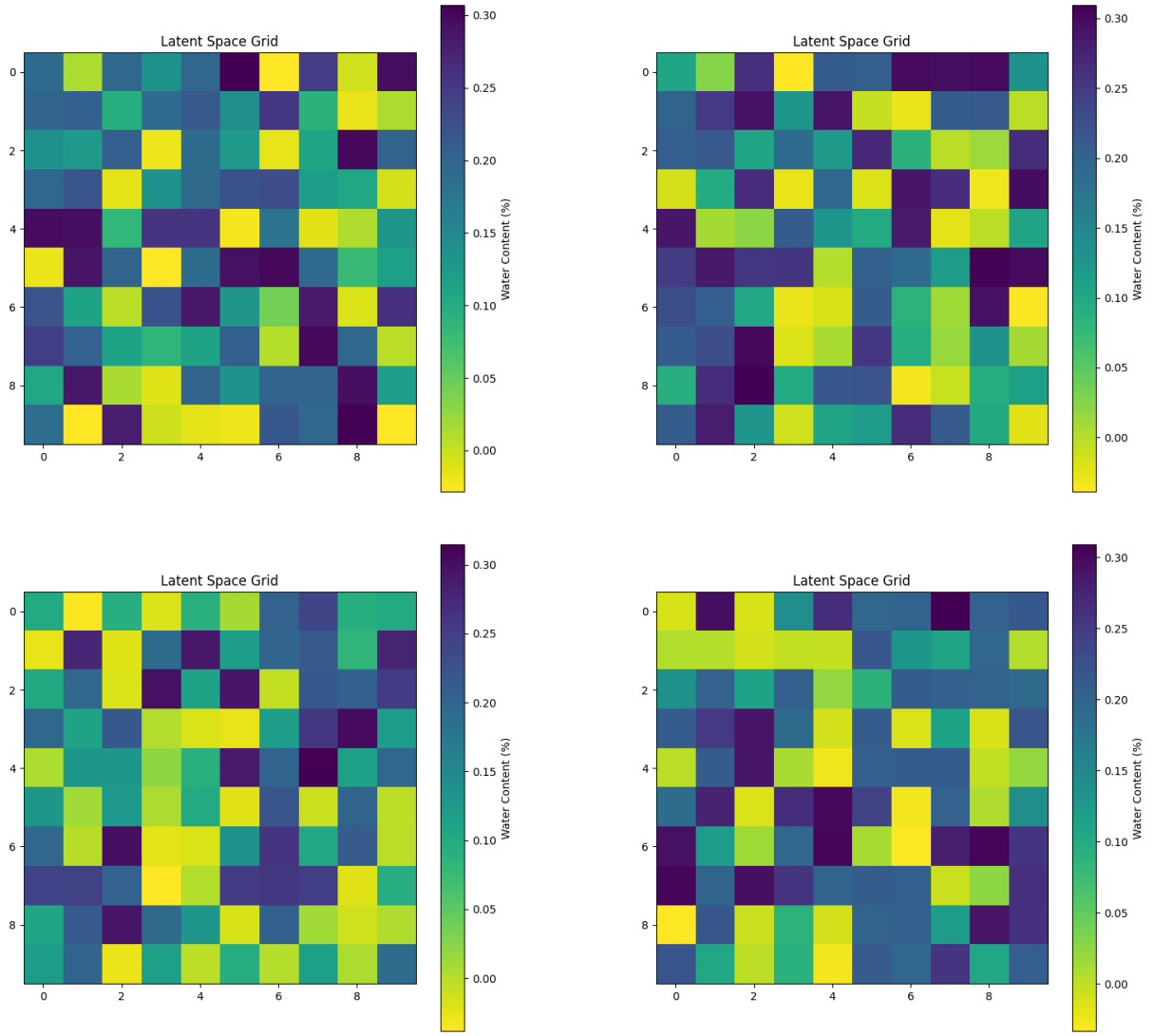


Figure 14: Representative examples of the original latent grids.

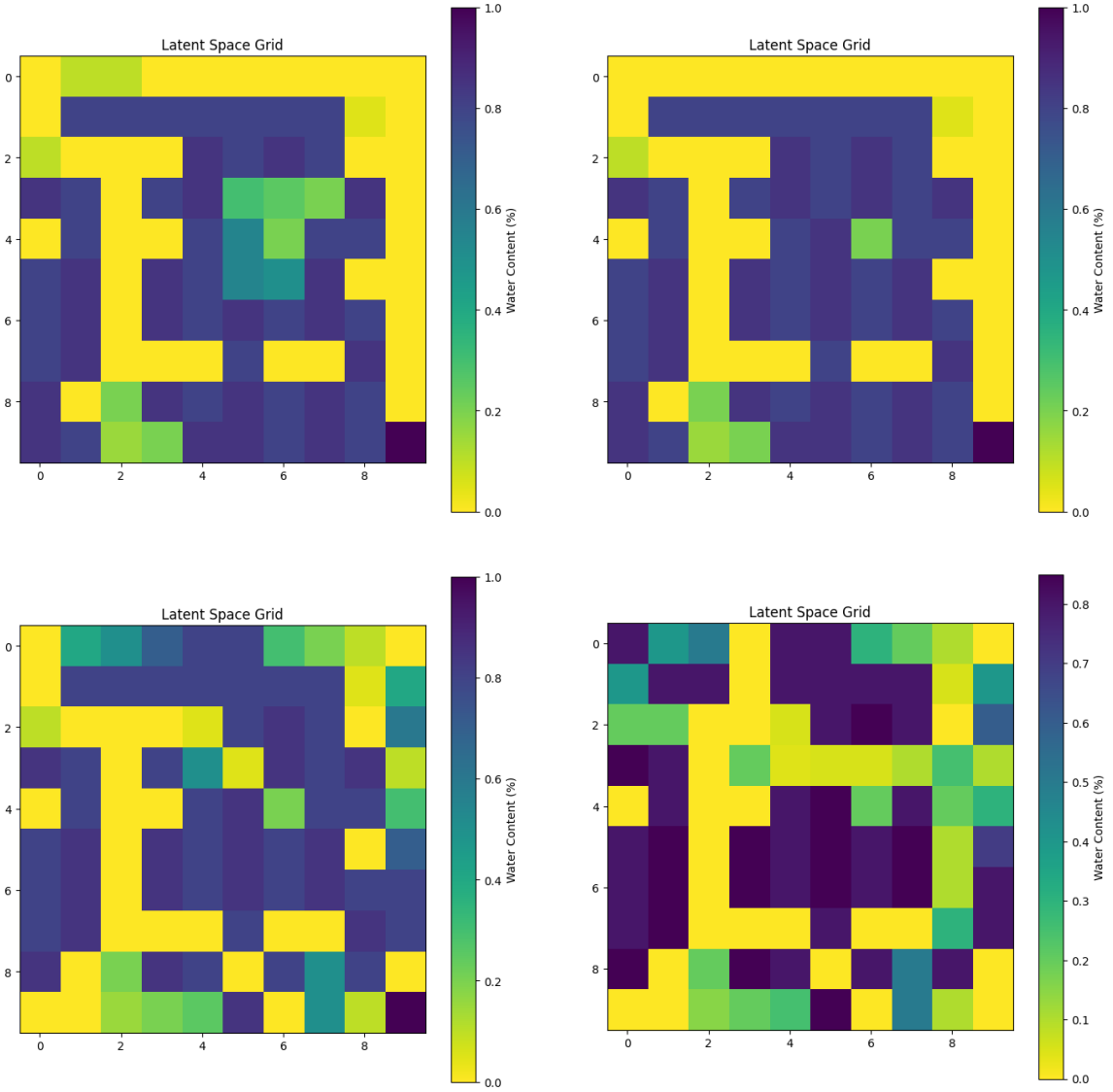


Figure 15: Representative examples of the enhanced latent grids.

5.3 Action Space

The action space of the agent is defined as a discrete set of actions, each corresponding to one of the four cardinal directions of movement on the grid. In particular, the agent can choose to move `up`, `down`, `left`, or `right` at each decision step. These actions are encoded as integer values in a discrete space represented as $\{0, 1, 2, 3\}$, where each number maps to a specific directional movement.

To ensure realistic behaviour and prevent invalid transitions, the environment presents strict boundaries at the edges of the grid. Hence, if the agent attempts to move beyond the grid limit, the environment blocks the movement, and the agent remains in its current position.

5.4 Training and Evaluation Pipeline

After setting up the water content fields and defining the agent's interactions with the environment, we wrapped the resulting setup using the OpenAi Gym-compatible interface to ensure compatibility with standard reinforcement learning tools. The environments were then handled with the Stable-Baseline3 training pipeline, which provides a robust framework for implementing and evaluating state-of-the-art RL algorithms.

To enable scalable and efficient training, we use vectorized environments using the `SubprocVecEnv` utility from Stable-Baseline3, which allows multiple parallel instances of environments to run simultaneously. This accelerates data collection and policy updates. Furthermore, we also used the `VecMonitor` wrapper from Stable-Baseline3 to log key statistics such as episodic reward, episode length, and goal achievement across all environments, enabling both live monitoring and post-training analysis.

The training pipeline is built around the Proximal Policy Optimization (PPO) algorithm 3.21 provided by the `Stable-Baseline3` library. PPO is a widely-used and stable on-policy reinforcement learning method which balance between performance and implementation simplicity by using a clipped surrogate objective function that prevents large, unstable policy updates.

PPO operates using multiple epochs of stochastic gradient ascent over mini-batches of experience collected through environment rollouts. In our setup, we use a multithreaded training loop with $n\_steps = 2048$, a batch size of 4096, and a moderate entropy coefficient $ent\_coef f = 0.01$ to encourage exploration while maintaining learning stability.

Stable-Baseline3 provides flexible and modular policy definitions for reinforcement learning agent through its `policy` parameter. For our PPO implementation, we employ the built-in `MlpPolicy`, which defines both the actor and the critic netowrks as multi-layer perceptrons (MLPs). This architecture is well-suited for environments with low-dimensional or structured vector observations, such as our grid-based setup with flattened local masks.

By default, `MlpPolicy` consists of two fully connected layers with 64 hidden units each and `tanh` activations, but the architecture can be easily modified using the `policy_kwargs` argument for more advanced customization.

For higher-dimensional or image-based inputs, such as raw B-scans or camera frames, Stable-Baseline3 also provides the `CnnPolicy`, which leverages convolutional layers to extract spatial features before applying fully connected layers. This modularity allows for easy switching between architectures depending on the nature of the observations, enabling the same training

pipeline to adapt to a wide range of state representations.

The choice of PPO algorithm is mainly due to allow agents trainings that are both sample-efficient and robust to the sparse and noisy reward structure commonly observed in exploration-based tasks such as moisture-aware navigation (see next section for more details about reward structure).

Furthermore, it is important to highlight other key hyperparameters ussed during the training process. In our setup, the agent is trained over a total of $1{,}500{,}000$ timesteps using 20 parallel environments, which enables efficient sampling and faster policy updates. Each environment employs a local observation mask of size 3×3. These hyperparameters can be easily modified and adapted to suit specific experimental conditions or reward formulations. In particular, the mask size plays a critical role in shaping the agent's perception of its surroundings and will be systematically modified in later sections to evaluate its impact on policy performance and generalization.

The framework then supports optional curriculum learning and early stopping condition logics. Evaluation is performed in a separate vectorized environment with deterministic actions, and evaluation callbacks track the number of successful goal-reaching episodes. Videos of the agent behaviour are recorded periodically using a custom `VideoEvalCallback` function, providing visual insights into the policy's learning dynamics.

At the end of training, performance plots are generated, including the all-reward learning curve, a smoothed reward graph, a per-cell reward heatmap, and a visit frequency heatmap. Additionally, all trajectories are saved for post-analysis, and a textual summary is generated, containing statistics such as the best and final rewards, most visited cell, and the goal-reaching success rate.

To quantitatively compare the different reward function designs developed, as well as the associated hyperparameters, we evaluated the trained agents across a batch of test environments, most of which were not encountered during training. This evaluation involved testing various training checkpoints and best-performing models, saved during the agents training, in order to track the learning progress over time. Agent performance was measured by two key metrics: the success rate of reaching the goal and the average cumulative water content encountered throughout the episodes. By plotting both success rates and average cumulative water levels over the training time, we can assess how each reward function influences the agent's learning progress and ability to generalize.

Overall, this pipeline supports systematic benchmarking of policy performance across different environmental setups, grid sizes, and, especially, reward functions.

5.5 Reward Function Design and Evolution

In reinforcement learning, the reward function plays a central role in shaping the agent behaviour, as it defines the objectives that the policy must learn to optimize. In the context of moisture-aware navigation through partially observable latent grids, designing an effective reward function is particularly critical. The reward must guide the agent not only toward reaching the goal, but also toward minimizing exposure to high water content areas and avoiding redundant exploration. Hence, in this section we present the different reward function variants developed, each designed to address specific performance goals. Before diving into these variants, it is important to clarify some aspects of the setup. The goal position is consistently places at the

bottom-right corner of the gird, hence at coordinates $(\texttt{grid\_size} - 1, \texttt{grid\_size} - 1)$, while the agent's initial position is randomly sampled from anywhere within the grid domain.

Furthermore, for consistency in notation throughout the following sections, we will refer to each individual element of the reward function as a reward component. Further distinctions or explanations will be provided where necessary.

5.6 Initial Baseline and Limitations

Our initial reward function is simple and sparse, designed to establish a behavioral baseline for the agent. The agent receives a fixed negative penalty at every timestep to encourage shorter trajectories, and a large positive reward when reaching the goal, while small rewards and penalties are assigned for visiting new cells and revisiting previously explored ones, respectively. This setup is used to evaluate the agent's ability to learn basic navigation behaviour across environments of different grid sizes, without any bias toward water avoidance.

In addition, episodes are limited in time, with a maximum duration of 100 timesteps, in order to prevent excessively long or never ending behaviours. Furthermore, a penalty is imposed when the agent attempts to move beyond the grid boundaries.

Formally, the reward r_t at time t is defined as:

$$r_t = \begin{cases} +10 & \text{if the agent reaches the goal position} \\ -2 & \text{if the agent attempts to move outside the grid boundaries} \\ +0.5 & \text{if the agent explores a new cell} \\ -0.1 & \text{if the agent revisits a cell} \\ -1 & \text{if the episode exceeds the 100-timestep limit} \\ -0.2 & \text{for each step} \end{cases} \tag{63}$$

This formulation motivates the agent to reach the goal in the fewest steps possible, while penalizing aimless wandering. However, the absence of any environment-specific feedback related to water content levels means that the agent has no incentive to avoid undesirable regions or explore strategic, drier paths.

As expected, the agent quickly converges on a specific, fixed trajectory toward the goal, repeating the same path across all environments. This behavior occurs because the only optimization objective is to minimize the number of steps to reach the goal, without any consideration for the water content encountered along the route.

Figure 16 presents the performance of the agent trained for $1,500,000$ timesteps using the baseline reward function, evaluated on a batch of randomly selected environments from the evaluation dataset. The objective is to monitor the agent's learning progress by tracking two key metrics: the success rate and the total average water content encountered during episodes. The success rate is computed by counting how many environments in the batch the agent successfully completes by reaching the goal. Simultaneously, we compute the total water content encountered along the path to the goal for each environment, and then average this value across the batch. Although water content is originally expressed as a percentage within the range $[0, 1]$, in this context we treat it as a raw numerical value so we can actually sum the values encountered at

each step to estimate the total amount of water. This evaluation procedure is performed at every checkpoint of the training process to track the agent's performance over time.

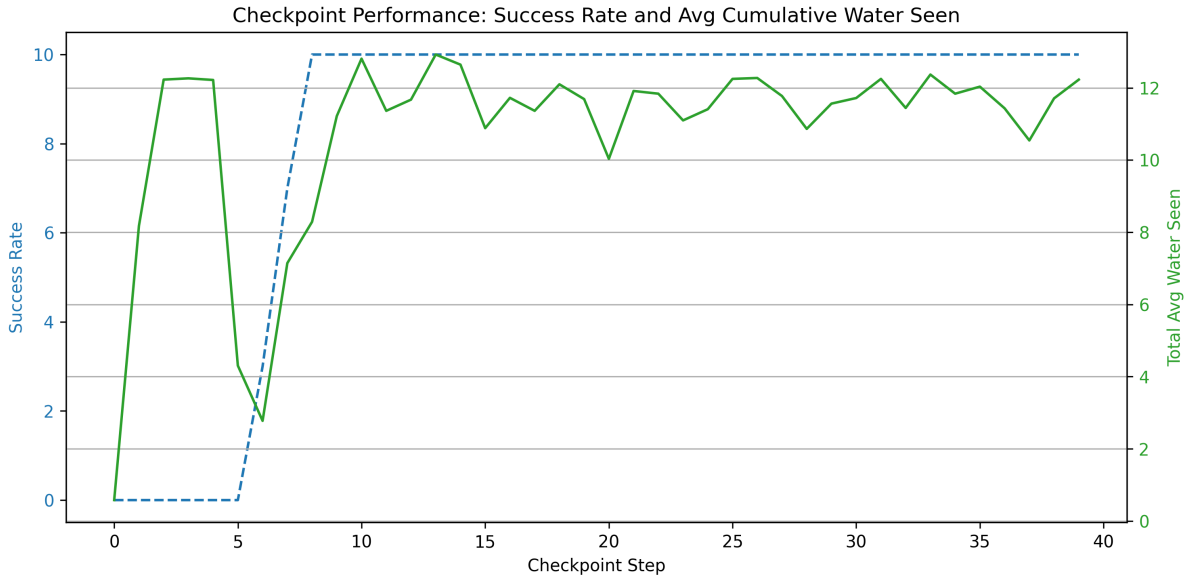


Figure 16: Performance evaluation of the baseline agent over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

Figure 16 proves that the agent successfully learns a policy capable of reaching the goal in every tested scenario. However, because the baseline reward function does not incorporate any feedback related to environmental conditions, the agent does not consider the water content encountered along its path. As a result, the obtained average cumulative reward remains high, and it can be shown that it increases proportionally with the size of the environments.

While this baseline reward structure provides a useful starting point, it does not account for any feedback related to the environmental properties. To address this, several improvements were progressively introduced to enrich the reward signal with additional information. These enhancements provide the agent with better environmental awareness behaviours, such as preferring drier paths, avoiding saturated zones, and exploring the grid more effectively. These are progressively presented in the following subsections.

5.7 Water-Aware Local Evaluation

To better align the learning agent's behaviour with both the physical properties of the field and the dual objectives of this work, we introduce a water-sensitive reward term. This term penalizes the agent in proportion to the water content it encounters during navigation, encouraging it to follow the driest paths in the field. Two different strategies have been implemented and compared. The first imposes a penalty based on the cumulative water content encountered by the agent from the beginning of the episode up to its current position, while the second applies a penalty just on the instantaneous water level at each time step.

Let us denote w_t the water content in the agent's position at time step t. The first approach can

be formalized as follows. The cumulative water content level encountered by the agent is:

$$W_{\text{tot}} = \sum_{t=0}^{T} w_t \tag{64}$$

The corresponding cumulative penalty applied at each time step is:

$$R = -\alpha \cdot W_{\text{tot}} = -\alpha \cdot \sum_{t=0}^{T} w_t \tag{65}$$

where $\alpha > 0$ is a scaling factor that controls the magnitude of the penalty. This formulation ensures that maximizing the cumulative reward corresponds to minimizing the agent's overall exposure to water.

Even though this approach captures the cumulative nature of water exposure, it presents several important drawbacks. In particular, agents may learn to exploit the reward structure by staying in regions with low water content, repeatedly looping within these areas. As a result, they can effectively minimize the cumulative penalty but without making any meaningful progress toward the actual final target goal.

To address this issue, we introduce a second strategy in which only the immediate water content at each time step is penalized, encouraging consistent progress toward the goal. Formally, the agent receives a penalty at each time step is:

$$r_t = -\alpha \cdot w_t \tag{66}$$

This method achieves the same overall objective while avoiding unnecessary memory usage and preventing degenerate behaviours such as local looping. Indeed, minimizing the immediate penalty at each time step implicitly leads to the minimzation of the cumulative penalty. Thus, minimizing the immediate penalty $r_t = -\alpha \cdot w_t$ at each step effectively minimizes the cumulative penalty. This stepwise formulation is both computationally efficient and better aligned with the agent's forward progression toward the goal.

Hence, by using the second approach for the water content consideration, the overall reward structure at each step t is defined as follows:

$$r_t = r_t^{\text{water}} + r_t^{\text{distance}} + r_t^{\text{explore}} + r_t^{\text{goal}} + r_t^{\text{boundaries}} + r_t^{\text{time}} + r_t^{\text{step}} \tag{67}$$

Where:

- r_t^{water} is the reward based on the water content level
- r_t^{distance} is the reward component related to progress toward the goal
- r_t^{explore} is the reward component for visiting new cells
- r_t^{goal} is the reward for reaching the goal

- $r_t^{\text{boundaries}}$ is the penalty for trying to go beyond the boundaries of the grid
- r_t^{time} is the penalty for exceeding the time limit
- r_t^{step} is the penalty for each step

The water-related reward component penalizes the agent based on the water content at its current location. As previously described, it is defined as

$$r_t^{\text{water}} = -\alpha \cdot w_t \tag{68}$$

where $w_t \in [0, 1]$ denotes the water content at time step t, and α is the scaling factor. In this setup, the selected values of α is:

$$\alpha = 2 \tag{69}$$

This choice is motivated by the fact that all w_t values are within the range $[0, 1]$, and higher scaling factor amplifies the difference between drier and wetter regions. This encourages the agent to prefer paths with lower water content.

The goal-oriented reward component, r_t^{distance}, is computed using the Manhattan distance, which is appropriate in a discrete grid-based environment. The Manhattan distance between the agent's current position (x_t, y_t) and the goal position (x_g, y_g) is given by:

$$d_{\text{Manhattan}}(t) = |x_t - x_g| + |y_t - y_g| \tag{70}$$

This distance is used to compute the goal-oriented reward component r_t^{distance}, which, in practice, acts as a penalty since it is a negative reward proportional to the distance from the goal. As a result, the agent is incentivized to reduce this penalty by moving towards the goal, thus reducing the Manhattan distance. Formally, this component is defined as:

$$r_t^{\text{distance}} = -\beta \cdot d_{\text{Manhattan}}(t) \tag{71}$$

where $\beta = 0.1$ is a scaling coefficient. This formulation encourages the agent to minimize its distance to the goal at each time step, promoting efficient and goal-directed navigation.

The exploration reward component is conceptually similar to the one introduced in the baseline reward function, see Section 5.6. Its purpose it to encourage the agent to explore unvisited regions of the environment and to achieve this, the agent maintains an internal tracking of the grid cells it has already visited during the current episode.

Hence, at each time step, if the agent's current position lies within a cell it has not previously visited, the exploration reward component assigns a positive reward, while, if the agent revisits a previously explored cell, it is penalized with a negative reward. This mechanism promotes spatial exploration and discourages redundant behaviour. However, it must be carefully tuned to avoid excessive exploration that could lead the agent away from the goal.

Formally, the exploration reward component $r_t^{\text{exploration}}$ is defined as:

$$r_t = \begin{cases} +0.8 & \text{if the agent explores a new cell} \\ -0.3 & \text{if the agent revisits a cell} \end{cases} \tag{72}$$

Note that the reward values used here differ from those presented in the baseline function, as the current reward formulation includes additional components. Hence, a different tuning process is required. The values presented yield the best observed performance.

The goal reward component provides a positive bonus to the agent when it reaches the predefined goal location in the environment. Formally, the goal reward at time step t is defined as:

$$r_t^{\text{goal}} = +5 \quad \text{if the agent reaches the goal position} \tag{73}$$

The boundaries reward component penalizes the agent whenever it attempts to move beyond the defined boundaries of the environment. Specifically, if the agent selects an action that would place it outside the valid grid area, its position remains the same of its previous location, and a fixed penalty is applied. Formally the penalty is defined as:

$$r_t^{\text{boundaries}} = -1 \quad \text{if grid boundaries are violated} \tag{74}$$

The time reward component is designed to encourage the agent to reach the goal within a predefined time limit. In this setup, each episode is limited to a maximum of 100 timesteps. If the agent fails to reach the goal within this limit, it is penalized and the episode is truncated and reset. This discourages inefficient navigation, such as aimless wandering or looping behaviour, and promotes efficient decision-making.

Formally, the penalty is defined as:

$$r_t^{\text{time}} = -3 \quad \text{if the episode exceeds the 100 timestep limit} \tag{75}$$

The step reward component applies a constant negative reward at each time step of the episode. Its purpose is to encourage the agent to complete its task as quickly as possible. By penalizing each step, the agent learns to minimize the length of its path to the goal.

Formally, the step penalty is defined as:

$$r_t^{\text{step}} = -0.2 \quad \text{for each step} \tag{76}$$

So, the overall structure of the reward function can be represented numerically as:

$$r_t = \begin{cases} -2 \cdot w_t & \text{water reward} \\ -0.1 \cdot d_{\text{Manhattan}}(t) & \text{distance reward} \\ +0.8 \text{ (new cell)} & \text{exploration reward} \\ -0.3 \text{ (revisited cell)} & \\ +5 & \text{goal reward} \\ -1 & \text{boundaries reward} \\ -3 & \text{time limit reward} \\ -0.2 & \text{step reward} \end{cases} \tag{77}$$

The presence of many different numerical values across the component of the reward function results in a high degree of variability in the overall reward signal. This variability introduces

complexity in tuning the parameters, as different combinations of reward components can produce varying effects on the learning dynamics, which could slow down the convergence to an optimal policy.

Figure 17 presents the performance metrics of the agent trained with this new reward function design defined in Equation 67. As observed in the plot, the agent continues to successfully reach the goal across different scenarios. However, compared to the baseline case, where the average cumulative water content was around 12, the new reward design leads to a significant improvement, with the water content converging around a value of 4. This demonstrates that the introduction of the water-aware component in the reward function effectively enables agents to interpret the field conditions and to learn behaviours that actively avoid high-moisture regions. The comparison of the cumulative water content between the two reward function designs is shown in Figure 18.

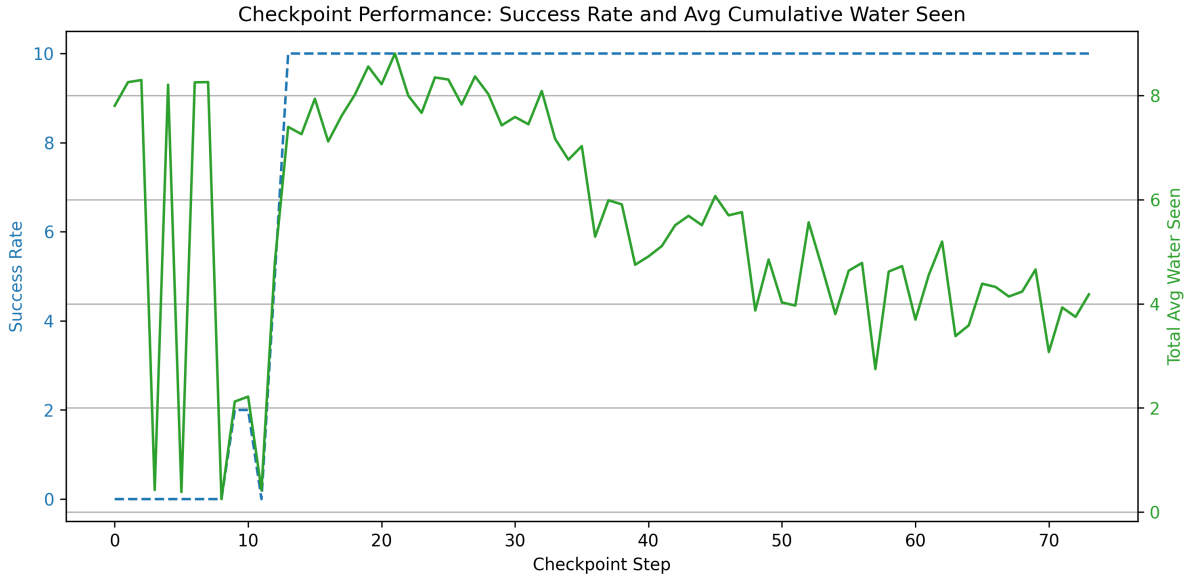


Figure 17: Performance evaluation of the agent with the new reward function design over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

From these plots, it is possible to notice that high variability in the average water content is still present even with the new reward function. Therefore, careful balancing is required to ensure stable and effective learning. The complexity and the fluctuations of the current reward formulation highlights the need for a smoothly varying reward structure. For this reason, improved versions of the reward function are introduced and analyzed in the following sections.

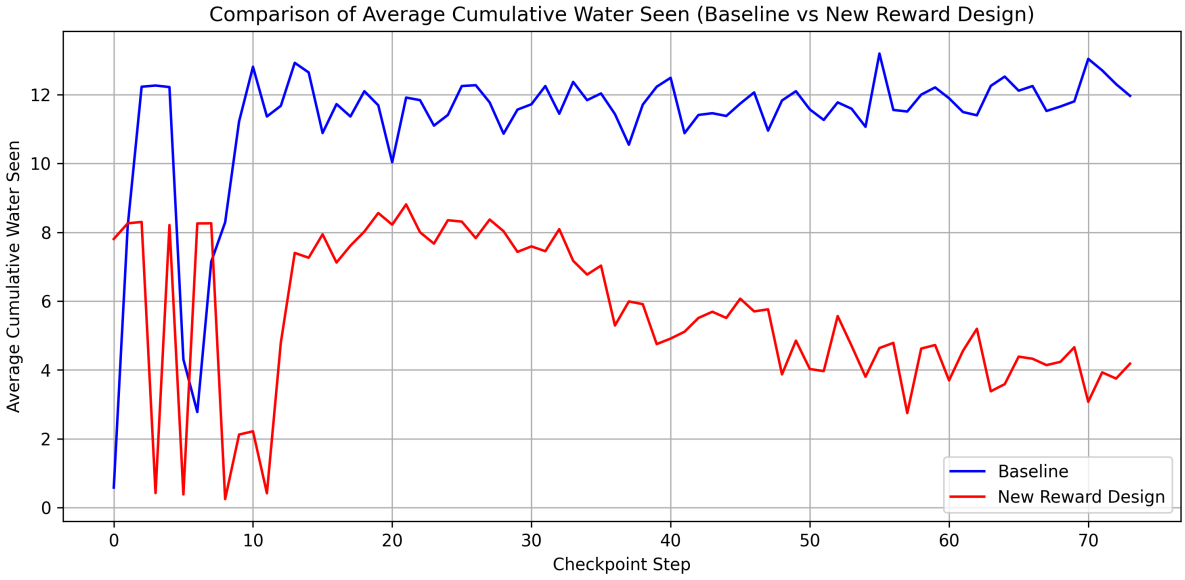


Figure 18: Average cumulative water content comparison between the baseline reward function in blue, and the new reward function design in red.

5.8 Water-Based Improvements

The reward function previously described in Section 5.7 provides a robust and efficient mechanism to integrate the water sensitivity into the distance-based shaping objective. By aligning immediate behaviour with long-term goals, it helps the agent avoid suboptimal strategies such as local looping and inefficient wandering.

However, this formulation also introduces high variability in the rewards the agent receives across different episodes, which can lead to unstable learning and slow convergence. In addition, the integration of water sensitivity creates new challenges. For instance, the agent may get trapped in dry regions with limited mobility or in areas where no dry paths are available to reach the goal while avoiding highly wet zones. These issues highlight the need for further refinement of the reward structure to ensure both robustness and stability during training.

To mitigate the instability caused by high reward variance across episodes and improve convergence during training, some improvements have been introduced in the water-aware reward logic. Rather than relying only on the instantaneous water content at the agent's current position w_{current}, the agent is now equipped with a local perceptual field defined by a square observation mask of size $M \times M$, where $M = \texttt{MASK\_SIZE}$. This mask enables the agent to perceive the water content of local regions, thus broadening its spatial awareness and supporting more informed, context-sensitive decision-making.

Formally, at each time step, the environment computes a local water mask:

$$\mathbf{W}_{\text{local}}[i,j] = \begin{cases} w(x+i-c, y+j-c) & \text{if } 0 \le x+i-c < G \\ & \text{and } 0 \le y+j-c < G \quad \text{for } i,j = 0,\ldots,M-1 \\ -1.0 & \text{otherwise} \end{cases} \tag{78}$$

where:

- G is the grid size (width and height of the latent grid),
- $c = \left\lfloor \frac{M}{2} \right\rfloor$ is the center index of the mask,
- $w(i, j)$ is the water content at position (i, j),
- values outside the grid boundaries are filled with a default value of -1.0, which is a clear invalid value of water content level.

This discrete sampling over a centered window enables the agent to observe the immediate neighborhood's water content and plan its movement while avoiding wet or invalid regions.

From this local observation mask, several algorithms can be derived. In what follows, we present the primary and most effective methods we developed. In subsequent sections, we will incorporate some of them to construct the complete and final reward function.

5.8.1 Local Water Gradient-Based Reward Formulation

A possible use of the mask previously computed is to extract the minimum local water content in the mask, defined as:

$$w_{\text{min}} = \min(\mathbf{W}_{\text{local}}) \tag{79}$$

From this, we compute the local water gap as the difference between the current water content at the agent's position and the minimum in the local mask:

$$\Delta w = w_{\text{current}} - w_{\text{min}} \tag{80}$$

The reward component associated with this water gradient is then determined based on two thresholds, both introduced as hyperparameters of the system:

- τ_g: maximum acceptable gap to still consider the current position relatively optimal,
- τ_d: absolute dryness threshold, below which a region is considered sufficiently dry.

Using these thresholds, the water-related reward r_{water} is defined as:

$$r_{\text{water}} = \begin{cases} \alpha & \text{if } \Delta w \leq \tau_g \text{ and } w_{\text{current}} \leq \tau_d \\ \beta & \text{if } w_{\text{current}} \leq \tau_d \\ \gamma & \text{otherwise} \end{cases} \tag{81}$$

Where:

- α is the highest reward, assigned when the current position is not only sufficiently dry but also among the driest in the local neighborhood,
- β is a moderate reward, given when the position is dry but not the most optimal in its local area,
- γ is a negative reward ($\gamma < 0$) applied when the agent is in a wet region

This formulation encourages the agent to move toward locally optimal dry regions while avoiding wet areas along the way. If the agent is already in one of the driest positions, it receives a strong positive reward (α), while, if it is in a dry zone but not the best in its local area, it receives a smaller but still positive reward (β). Conversely, positions that are too wet relative to the local minimum are penalized (with γ), discouraging movements in these areas.

It is important to note that the mask size M now becomes a critical hyperparameter. A smaller mask may not capture enough environmental features, while a larger one may attenuate relevance characteristics or introduce too much noise. Thus, M must be carefully tuned to balance this trade-off.

While the water-based reward component effectively encourages the agent to avoid wet regions, replying only on this criterion leads to suboptimal behaviour, where the agent tends to settle in the nearest dry areas relative to its starting position. Therefore, to ensure directed exploration and movement, it is necessary to still consider the reward component based on the distance to the goal. This term is built upon the distance-based reward component r_t^{distance} introduced in Section 5.7, with specific considerations to ensure compatibility with the new water-aware reward formulation just introduced. For further details and implementation considerations, see Section 5.10.

5.8.2 Average Local Water-Based Reward Formulation

Another possible use of the mask previously computed is to calculate the average water content in the agent's local neighborhood and compare it with the water content at the agent's current position. This allows the agent to evaluate whether its current location is relatively wet or dry compared to its local area.

Formally, the average local water is given by:

$$\bar{w}_{\text{local}} = \frac{1}{|\mathcal{V}|} \sum_{(i,j)\in\mathcal{V}} \mathbf{W}_{\text{local}}[i,j] \tag{82}$$

where $\mathcal{V} \subseteq [1, M] \times [1, M]$ is the set of valid indices in the mask, and $|\mathcal{V}|$ is the number of such valid entries.

Let w_{current} denote the water content at the agent's current position. The reward component r_t^{avg} is defined by comparing w_{current} and $\bar{w}_{\text{local}}$:

$$r_t^{\text{avg}} = \begin{cases} \alpha & \text{if } w_{\text{current}} < \bar{w}_{\text{local}} \\ \beta & \text{if } w_{\text{current}} = \bar{w}_{\text{local}} \\ \gamma & \text{if } w_{\text{current}} > \bar{w}_{\text{local}} \end{cases} \tag{83}$$

Here, the symbolic constants are defined as:

- α: a positive reward for being in a drier-than-average local position,
- β: a neutral or small reward if the agent's water level matches the local average,
- γ: a penalty for being in a wetter-than-average location.

This formulation allows the agent to know its relative condition within the local patch, encouraging movement toward locally drier regions even when no absolute dryness threshold is satisfied. It is especially useful in smooth environments where differences in water content are less evident.

Compared to the local gradient-based reward formulation presented in the previous Section 5.8.1, the average-based reward offers several advantages:

- Robustness to Outliers: the minimum-based approach is sensitive to isolated dry cells, outliers, potentially encouraging the agent to follow suboptimal directions. On the other hand, the average water content provides a more stable and representative measure of the local environment.
- Smoothness of the Reward Function: using the mean value creates a smoother reward signal across different positions, which is beneficial for gradient-based policy updates and generally leads to more stable learning dynamics.
- Better Local Awareness: while the minimum-based method focuses on identifying and moving toward a single optimal cell, the average-based approach considers the overall values of the local area. This encourages the agent to prefer globally drier regions, even if no clear local minimum exists.

Despite its advantages, the average-based reward formulation also presents several limitations.

- Reduced Sensitivity to Fine Gradients: by averaging the local water values, meaningful variations within the neighborhood may be smoothed out. This can lea to weaker incentives for the agent to seek the driest specific path.
- Weaker Directionality: the average-based approach lacks a natural directional signal. This may result in more indecisive behaviour in locally heterogeneous regions.
- No Explicit Target Preference: the agent might remain in relatively dry areas if they are just below the average, rather than actively pursuing the direst possible regions.

These limitations suggest that the average-based reward may be most effective when used in combination with other components, such as local minimum-aware incentives, to compensate for its lower directional precision. Hence, in the future algorithms we will integrate both formulations to exploit their complementary strengths and achieve more robust and adaptive behaviour.

5.8.3 Directional Reward Toward Drier Regions

Another possible water-based enhancement consists of encouraging the agent not only to remain in dry regions but also to move in the direction of drier areas detected within its local observation mask. This strategy promotes local exploration in low-moisture zones while maintaining alignment with global navigation objectives.

Let us take the same mask as previously defined $\mathbf{W}_{\text{local}} \in \mathbb{R}^{M \times M}$. From this mask, we identify the coordinates of the driest cell:

$$(i^*, j^*) = \arg\min_{i,j} \mathbf{W}_{\text{local}}[i, j] \tag{84}$$

The reward then considers the direction of movement relative to this driest cell.

Let d_{current} be the Manhattan distance between the agent and the driest cell before the action, and d_{new} be the distance after the action. The reward component r_t^{drier} is defined as:

$$r_t^{\text{drier}} = \begin{cases} \delta & \text{if } d_{\text{new}} < d_{\text{current}} \\ -\delta & \text{otherwise} \end{cases} \tag{85}$$

where $\delta \in \mathbb{R}^+$ is a small positive constant controlling the reward magnitude. This mechanism effectively encourages the agent to shift its position toward locally optimal directions while penalizing movement away from such regions. However, since it operates mainly on local observations, it does not inherently guarantee progress toward the global objective. Therefore, this strategy must be supported with a customized distance-based reward component to ensure that the agent also continues to make meaningful progress toward the final goal position.

5.9 Distance-Based Improvements

Following the improvements proposed for water-driven behaviour presented in Section 5.7, we now enhance the reward terms related to goal-oriented navigation. In the current reward design (see Equation 67), two key components encode the distance-based logic, r_t^{distance} and r_t^{goal}. The r_t^{distance} component represents a progress-based reward, which encourages the agent to reduce the distance to the goal at each time step. The r_t^{goal} component, instead, is a goal-reaching bonus, which gives a large positive reward when the agent reaches the desired target location.

Starting from these components, multiple variations can be introduced. For instance, the agent can be strongly or weakly guided toward the goal by adjusting the magnitude and balance of these rewards. This introduces a critical trade-off between the two main competing objectives in our project, following drier path by minimizing water exposure but also reaching the goal efficiently by optimizing the lengths of the paths.

To handle this dual objective, the design of the reward function must carefully balance the contributions of both the water content and the distance-based components. This is typically achieved by tuning the weights of each term in the total reward function.

5.9.1 Normalized Manhattan distance reward

One of the possible variants of the progress reward component r_t^{distance} is based on its normalized version. In this approach, the agent is still rewarded based on its relative progress toward the goal but the raw distance is scaled to a $[0, 1]$ range.

Let (x_t, y_t) be the agent's position at time step t, and (x_g, y_g) the goal position, the Manhattan distance to the goal is defined as:

$$d_{\text{Manhattan}}(t) = |x_t - x_g| + |y_t - y_g| \tag{86}$$

To normalize this distance, we divide it by the maximum possible Manhattan distance in the environment. Assuming a grid of size $G \times G$, where $G =$ `GRID_SIZE`, the maximum distance occurs when the agent starts at the corner opposite the goal, leading to:

$$d_{\text{Manhattan}}^{\text{max}} = (G-1) + (G-1) = 2(G-1) \tag{87}$$

The normalized distance is thus:

$$\tilde{d}_{\text{Manhattan}}(t) = \frac{d_{\text{Manhattan}}(t)}{2(G-1)} \tag{88}$$

The reward component can now be defines as the negative of this normalized distance:

$$r_t^{\text{distance}} = -\sigma \cdot \tilde{d}_{\text{Manhattan}}(t) \tag{89}$$

where σ is a optional scaling factor to control the magnitude of the reward, based on the different scenarios.

This variant provides a smooth and consistent signal throughout the episode, effectively avoiding sparse rewards and leading to more stable training dynamics.

5.9.2 Linear and Exponential Attractors

Another potential enhancement to the distance-based reward design is the introduction of attractor terms, either linear or exponential, which can be added on top of the previous distance rewards. These attractors are particularly useful in scenarios where the goal is surrounded by regions with high water content levels. In these cases, the agent may hesitate to approach the goal due to water penalties. The attractor-based reward helps to address this problem by progressively increasing the incentive to approach the goal. The closer the agent gets to the target position, the stronger the reward signal becomes, thus encouraging reaching the goal even when it is surrounded by unfavorable regions.

Formally, a possible linear attractor reward that increases linearly as the agent approaches the goal could be the follow:

$$r_t^{\text{lin}} = \eta \cdot (1 - d_t) \tag{90}$$

where $\eta > 0$ is a tunable gain and d_t is the distance from the goal. This formulation provides a smooth and gradually increasing reward signal as the distance decreases. It can be used with both the Manhattan distance and its normalized version.

An alternative is to use an exponential shaping function to increase the sharpness of the incentive near the goal:

$$r_t^{\text{exp}} = \eta \cdot e^{-k \cdot d_t} \tag{91}$$

where $\eta > 0$ is the scaling factor, and $k > 0$ controls the steepness of the exponential curve. Higher values of k produce a sharper gradient near the goal, encouraging final-phase optimization.

The linear attractor is simple and provides a constant reward gradient as the agent moves closer to the goal. This uniform increase makes it easy to tune and particularly effective when a consistent incentive is desired across the entire trajectory. However, since the reward grows uniformly with proximity, it may implicitly encourage movement toward the goal even through wet regions. Therefore, careful tuning is required to prevent the agent from prioritizing distance over environmental constraints.

On the other hand, the exponential attractor concentrates the reward signal near the goal, making it highly effective in scenarios where the final convergence is challenging. However, this approach provides little to no reward signal when the agent is far from the goal, especially for large

values of the steepness parameter k. Hence, this approach is best used in combination with other distance-based reward components that incentivize early-stage progress toward the goal position.

Figure 19 compares the linear attractor with an exponential attractor using $k = 3$. The x-axis starts from 1, representing the maximum normalized distance from the goal, and gradually decreases to 0 as the agent approaches the target. As shown, the exponential function increases more rapidly as d_t decreases toward zero.

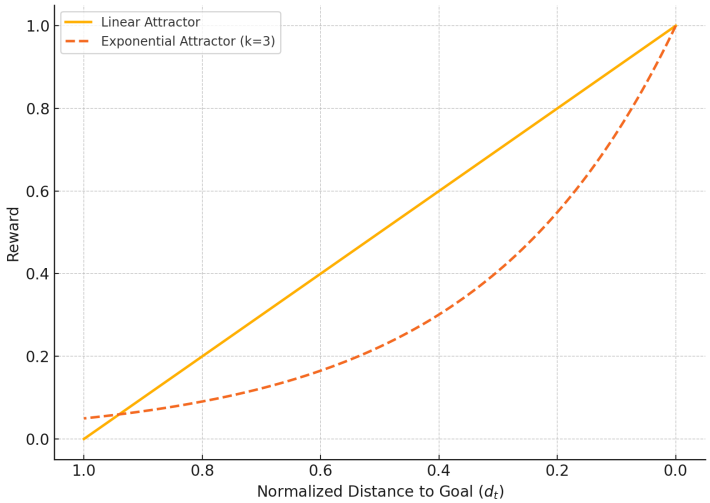


Figure 19: Comparison between linear and exponential attractor reward functions $(k = 3)$

In Figure 20, we plot several exponential attractor curves for different values of k to highlight the effects of steepness tuning. Larger k values result in sharper gradients near the target.

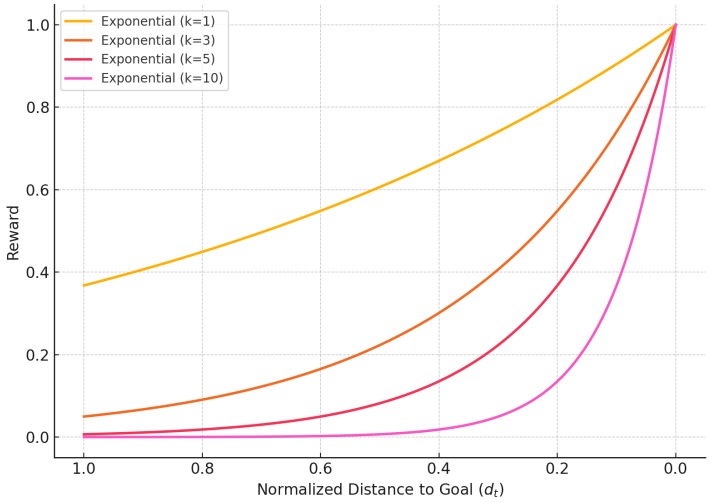


Figure 20: Exponential attractor reward functions for different k values.

5.9.3 Generalizing the Target Concept under Partial Observability

In the previous sections, we relied on the explicit computation of the Manhattan distance between the agent and the goal. This provided a direct and fully observable reward signal that encouraged the agent to optimize for the shortest path.

Hence, we now propose a generalization of the goal-reaching reward, where we still leverage the Manhattan distance and a metric of spatial progress, but we use them in a more implicit and comparative way. Rather than giving the agent access to the absolute distance at each step, we reward it based on whether its distance to the goal has decreased relative to the previous time step. This shift not only aligns better with partial observability settings, but also promotes different behaviours, enabling the agent to discover alternative and potentially better paths.

Although this approach typically requires longer training, and may presents higher reward variability and slower convergence, it ultimately leads to more generalizable policies.

Formally, we can still compute the Manhattan distances as presented in previous sections, and then define the agent's reward at each timestep based on the change in distance as follows:

$$r_t^{\text{progress}} = \begin{cases} +\lambda & \text{if } d_t < d_{t-1} \\ -\rho & \text{otherwise} \end{cases} \tag{92}$$

where $\lambda > 0$ is a progress reward and $\rho > 0$ is a penalty for regressing or stagnating.

In this approach, the agent no longer observes the exact target position directly but instead relies only on relative progress signals provided by the environment. While such a formulation may slow the convergence due to reduced observability, it leads to the development of policies that are more adaptable to changes in the environment or variations in the goal location.

5.10 Final Comprehensive Reward Function

After introducing improvements to both the water-related (see Section 5.8) and the distance-related reward terms (see Section 5.9), we now present the final reward function by combining these components into a unified formulation. The objective of this structure is to jointly encourage the agent to follow drier trajectories, approach the goal effectively, and avoid undesirable behaviours such as stagnation, looping, or boundary violation.

Different combinations of the reward components described in the previous sections have been tested and evaluated. These experiments involved merging various subsets of components and tuning their hyperparameters accordingly. Based on empirical performance and behavioural observations, the following formulation has been selected as the final reward function.

The reward received at time t is computed as:

$$\begin{aligned} r_t = \; & r_t^{\text{water\_gradient}} + r_t^{\text{distance}} + r_t^{\text{exp\_attractor}} + r_t^{\text{goal}} \\ & + r_t^{\text{explore}} + r_t^{\text{boundaries}} + r_t^{\text{time}} + r_t^{\text{step}} \end{aligned} \tag{93}$$

Each component is detailed below.

The $r_t^{\text{water\_gradient}}$ term is the one presented in Section 5.8.1, with the following structure:

$$r_t^{\text{water\_gradient}} = \begin{cases} \alpha & \text{if } \Delta w \leq \tau_g \text{ and } w_{\text{current}} \leq \tau_d \\ \beta & \text{if } w_{\text{current}} \leq \tau_d \\ \gamma & \text{otherwise} \end{cases}$$
$$= \begin{cases} +2 & \text{if } \Delta w \leq 0.2 \text{ and } w_{\text{current}} \leq 0.15 \\ +1 & \text{if } w_{\text{current}} \leq 0.15 \\ -0.8 & \text{otherwise} \end{cases} \tag{94}$$

The specific values used in $r_t^{\text{water\_gradient}}$ are chosen to capture the different scenarios while preserving a smooth and informative reward signal. These values have been empirically determined to ensure a balanced trade-off between the two main objectives, minimizing water exposure and reaching the goal efficiently. Moreover, their selection takes into account the influence of other hyperparameters within the full reward structure. As a result, any change in the environment conditions or in the tuning of other reward components may require corresponding adjustments to these values.

The r_t^{distance} component follows the structure introduced in Section 5.9.3, where rewards are based on the relative progress of the agent rather than absolute distance to the goal. Formally, the reward is defined as:

$$r_t^{\text{distance}} = \begin{cases} +\delta & \text{if } d_t < d_{t-1} \\ -\sigma & \text{otherwise} \end{cases} = \begin{cases} +1.5 & \text{if } d_t < d_{t-1} \\ -1.0 & \text{otherwise} \end{cases} \tag{95}$$

where d_t denotes the normalized Manhattan distance between the agent's current position and the goal position at time step t. The values of $\delta = 1.5$ and $\sigma = 1.0$ were selected to provide directional incentives while still allowing water-based navigation and possible exploration.

The $r_t^{\text{exp\_attractor}}$ reward component corresponds to the exponential attractor introduced in Section 5.9.2. Its mathematical structure is defined as:

$$r_t^{\text{exp}} = \eta \cdot e^{-k \cdot d_t} = 1.5 \cdot e^{-10 \cdot d_t} \tag{96}$$

where d_t denotes the normalized Manhattan distance between the agent's current position and the goal position at time step t. This attractor enables the agent to be strongly attracted to the goal in the final stages of an episode, even when the goal is surrounded by wet or suboptimal regions. For a visual representation of this behaviour, see Figure 20, which illustrates the exponential attractor function for different values of k, including the one we selected here ($k = 10$).

The r_t^{goal} is the reward bonus when the agent reaches the target cell and it can be formally written as:

$$r_t^{\text{goal}} = \begin{cases} \lambda_{\text{goal}} = +5.0 & \text{if the agent reaches the goal position} \\ 0 & \text{otherwise} \end{cases} \tag{97}$$

The r_t^{explore} component encourages exploration of the environment. Its structure is defined as:

$$r_t^{\text{explore}} = \begin{cases} \lambda_{\text{new}} = +1.0 & \text{if the agent explores a new cell} \\ \lambda_{\text{old}} = -0.5 & \text{if the agent revisits a cell} \end{cases} \tag{98}$$

The $r_t^{\text{boundaries}}$ component is used to ensure spatial validity by penalizing the agent whenever it attempts to move outside the predefined grid. Formally, it is defined as:

$$r_t^{\text{boundaries}} = \lambda_{\text{b}} = -0.5 \quad \text{if grid boundaries are violated} \tag{99}$$

This penalty is applied when detecting an out-of-bounds action, after keeping the agent's position to the last valid location before the invalid move.

The r_t^{time} component acts as a time-based penalty applied when the agent fails to complete the task within a predefined step limit, here $timestep\_limit = 100$. Its purpose is to encourage the agent to reach the goal before this limit, avoiding unnecessarily long trajectories or never ending situations. Formally, this component is defined as:

$$r_t^{\text{time}} = \lambda_{\text{t}} = -3.0 \quad \text{if } t \geq 100 \tag{100}$$

Finally, the r_t^{step} component accounts for the number of steps taken by the agent during an episode. It introduces a small penalty at each time step to encourage shorter and more efficient trajectories toward the goal. This term complements other components presented earlier, reinforcing path optimality and discouraging unnecessary movements. Formally, it is defined as:

$$r_t^{\text{step}} = \lambda_{\text{step}} = -0.1 \quad \text{for each step} \tag{101}$$

In conclusion, this modular reward structure offers a flexible framework for fine-tuning and guiding the agent's behaviour in a multi-objective setting. Each component plays a distinct role in shaping policies that count both for the water content levels and the goal achieving. Moreover, the structure is designed to remain robust under partial observability, allowing the agent to properly generalize across different scenarios. The complete set of hyperparameter values used in the final reward function is summarized in Table 1.

Table 1: Reward components with corresponding hyperparameters and their values for the discrete framework

Reward Component	**Hyperparameters to Tune**	**Values Used**
$r_t^{\text{water\_gradient}}$	$\alpha, \beta, \gamma, \tau_g, \tau_d$	$\alpha = +2.0, \beta = +1.0, \gamma = -0.8,$ $\tau_g = 0.2, \tau_d = 0.15$
r_t^{distance}	δ, σ	$\delta = +1.5, \sigma = -1.0$
$r_t^{\text{exp\_attractor}}$	η, k	$\eta = 1.5, k = 10$
r_t^{goal}	λ_{goal}	$\lambda_{\text{goal}} = +5.0$
r_t^{explore}	$\lambda_{\text{new}}, \lambda_{\text{old}}$	$\lambda_{\text{new}} = +1.0, \lambda_{\text{old}} = -0.5$
$r_t^{\text{boundaries}}$	λ_{b}	$\lambda_{\text{b}} = -0.5$
r_t^{time}	$\lambda_{\text{t}}, T_{\text{thr}}$	$\lambda_{\text{t}} = -0.1, T_{\text{thr}} = 100$
r_t^{step}	λ_{step}	$\lambda_{\text{step}} = -0.1$

As in previous sections, the agent's performance is evaluated by testing it across batches of environments. Figure 21 presents the performance metrics obtained using the final comprehensive reward function now presented (see Equation 93). It can be observed that the average cumulative water content is now significantly more stable and less variable than before, converging

asymptotically to a value around 4. Additionally, Figure 22 compares the average cumulative water content across all three reward function designs introduced so far. In particular, while the final reward function and the previous design, introduced in Section 5.7 yield similar asymptotic water levels, the final formulation exhibits significantly lower variability. This increased stability is particularly important as the size and complexity of the environments increase, where the performance gap between these two reward functions becomes more evident.

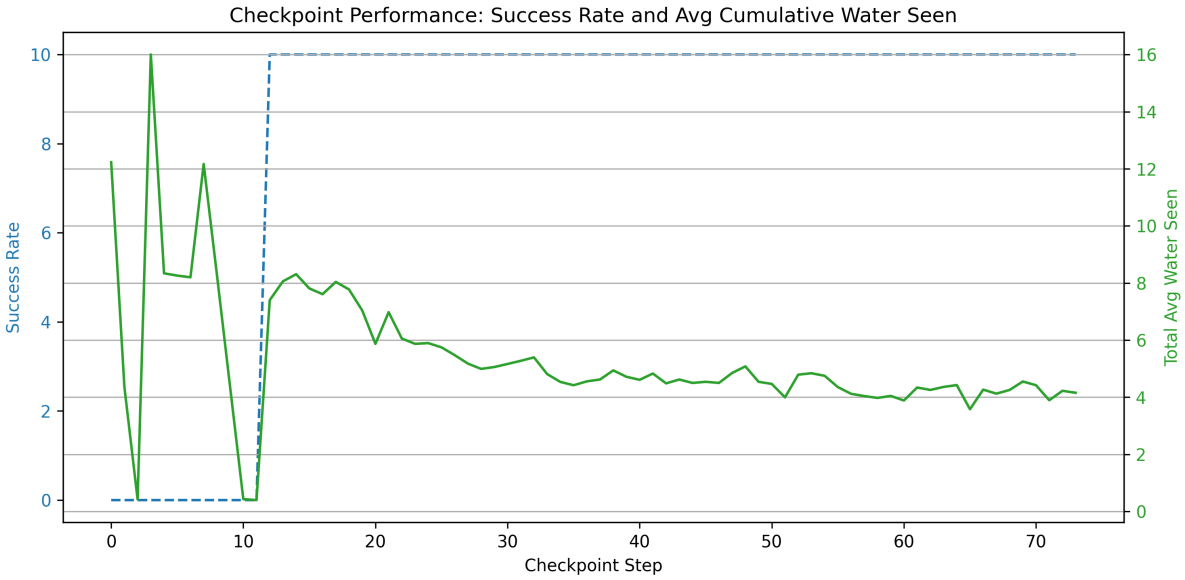


Figure 21: Performance evaluation of the agent with the final comprehensive reward function design over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

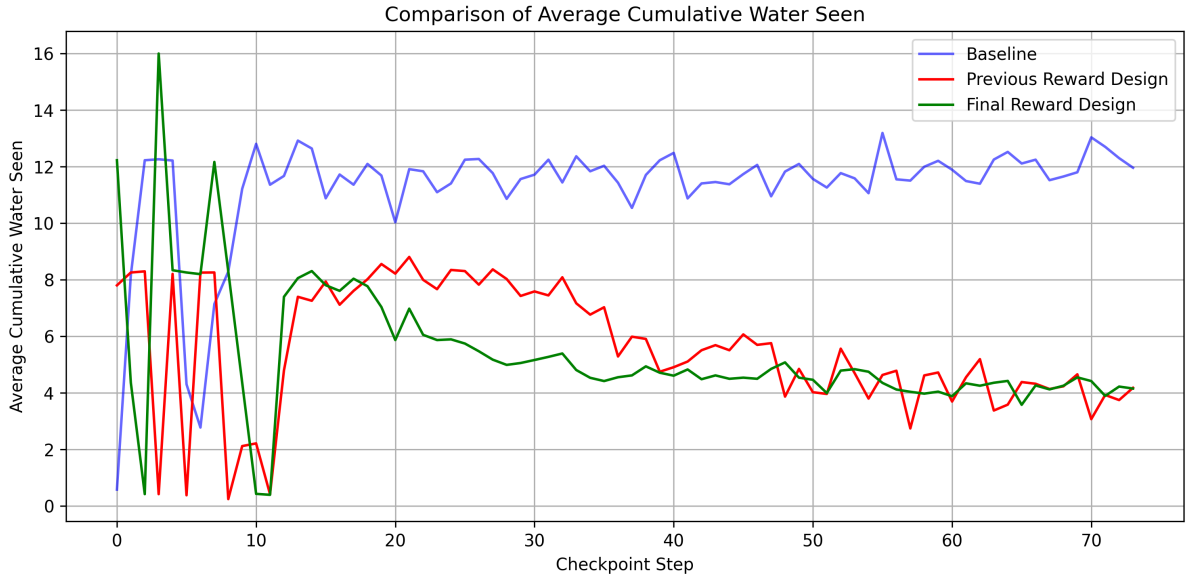


Figure 22: Average cumulative water content comparison between the baseline reward function in blue, the previous reward function design in red and the final reward function in green.

5.11 Results

In this section, we provide a comprehensive summary and interpretation of the plots and key performance indicators presented earlier. The goal is to consolidate insights across all experimental configurations to facilitate a global understanding of the agent's behaviour and learning dynamics.

As previously discussed, Figure 23 illustrates the average cumulative water content encountered by the agent across episodes in different training environments during its learning phase. This metric is used to evaluate the agent's effectiveness in identifying and navigating through dry regions, which is one of the two main learning objectives in this work.

The second objective, reaching the designated goal position, is evaluated by tracking the agent's success rate across different environments. As shown in earlier sections, all tested algorithms consistently achieved a 100% success rate in reaching the goal, confirming the reliability of the learned navigation policies.

Given this consistent performance, the subsequent analysis primarily focuses on the agent's water-awareness capability, which represents the more discriminative objective.

Hence, by examining the plot of Figure 23, several meaningful observations can be drawn.

The baseline reward function, represented in blue, converges asymptotically to an average cumulative water value of approximately 12. This relatively high value indicates limited awareness of the underlying water distribution and suboptimal navigation strategy.

The intermediate reward function, shown in red, introduces initial improvements by incorporating water-awareness shaping mechanisms and distance-based components. These improvements result in a significant reduction in the asymptotic water value, which stabilizes around 4.

This change indicates a substantial improvement in policy effectiveness, particularly notable in larger and more complex environments where effective exploration and water awareness becomes more challenging. However, both the baseline and intermediate reward strategies exhibit noticeable variability and lack robustness, as reflected in the fluctuations present in the plotted curves.

To address this, a third and final reward structure was proposed, designed to reduce instability while preserving the low water values achieved. This formulation introduced more refined water-awareness components that leverage information from the agent's local surroundings, as perceived through its observation mask. In addition, the distance-based components were redesigned to enhance generalization under partial observability, promoting consistent and efficient navigation across different environments.

As illustrated by the green curve in Figure 23, the final reward formulation maintains an asymptotic water value around 4, similar to the intermediate approach, but with significantly reduced variance across episodes and environments. This enhanced consistency indicates that the agent has learned a more stable and generalizable policy.

The selection of hyperparameters for each reward formulation was guided by a combination of quantitative metrics and qualitative analysis. Quantitatively, the success rate and the average cumulative water content were used to evaluate agents trained in different configurations. Complementing this, the qualitative assessment involved visual inspection of agent trajectories to determine whether their behaviour reflected the intended objectives. By integrating these two perspectives, we were able to fine-tune the hyperparameters, leading to policies that not only achieved strong numerical performance, but also demonstrated clear, objectives-aligned behavioural patterns.

Overall, these results validate the efficacy of the progressive reward shaping strategy, resulting in a robust policy that balances water minimization with spatial awareness and goal-oriented behaviour.

For a more detailed discussion of each individual reward formulation, including the specific components introduced and their respective impacts on agent behaviour, refer to the earlier sections where each reward design is described and analyzed in depth.

To conclude, Figures 24a–24b provide qualitative insight into the agent's behaviour under the final reward formulation. The two examples illustrate successful navigation across different field configurations, where the agent consistently starts from the top-right corner and moves toward the bottom-left goal. These trajectories are the result of the training process, showcasing the agent's ability to generalize its policy across different scenarios. The plots represent the final trajectory, as it was impractical to present a video showcasing each individual step and decision made during the episode. Nevertheless, each red dot in the visualization corresponds to a single step take by the agent, reflecting a specific decision made during the navigation. These visual outcomes emphasize the effectiveness of the final reward design. The agent consistently avoids high water content areas while maintaining a reliable path to the goal.

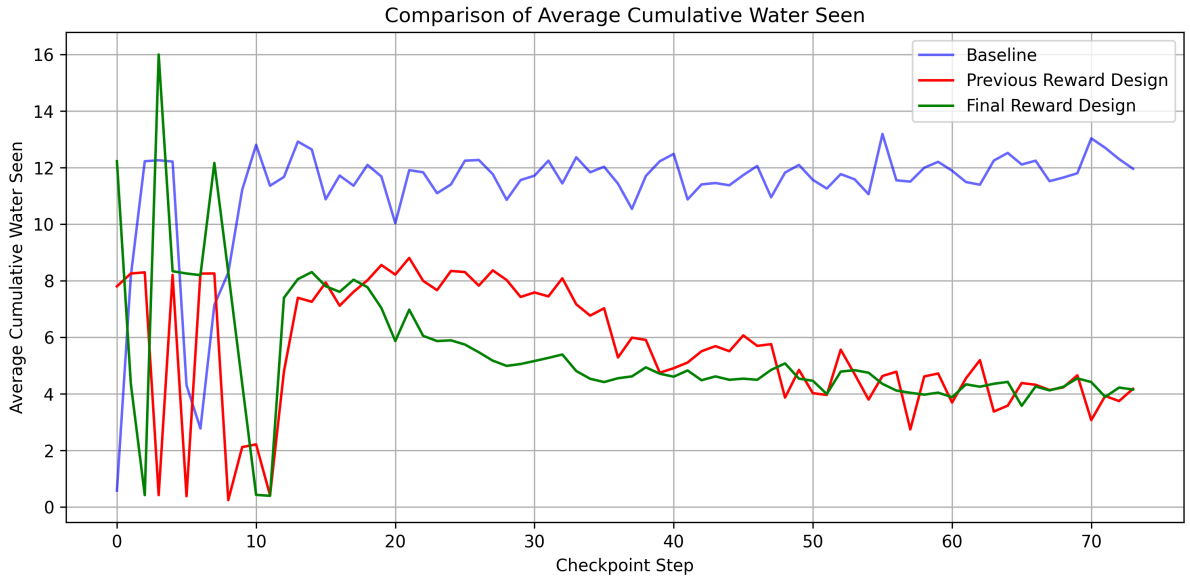
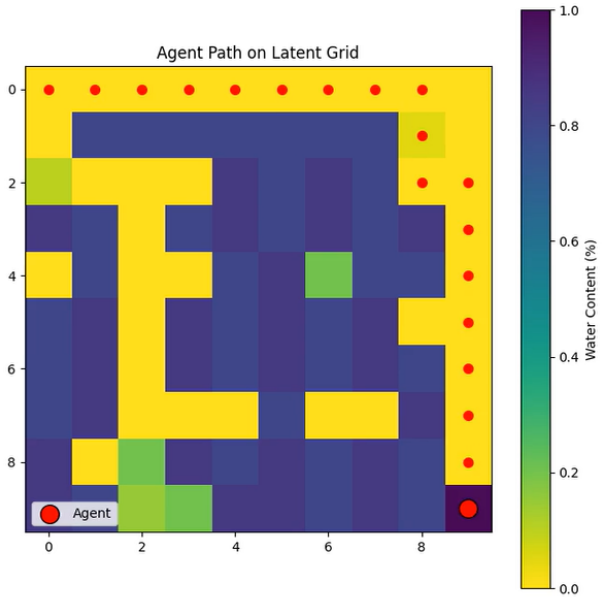
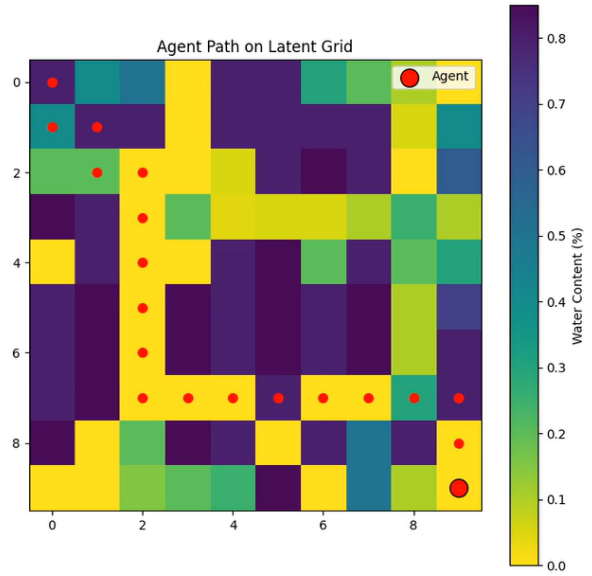


Figure 23: Average cumulative water content comparison between the baseline reward function in blue, the intermediate reward function design in red and the final reward function in green.



(a)



(b)

Figure 24: Examples of final trajectories of the agent

6 Continuous Reinforcement Learning Framework

After having fully described and formalized the discrete reinforcement learning framework in Chapter 5, we now shift our focus to the continuous scenario, where both the state and the action spaces are continuous rather than discrete.

In this chapter, we follow a structure similar to that of the previous one in order to maintain conceptual continuity. We will systematically highlight the key differences between the discrete and continuous settings and introduce the new challenges that arise when moving to a continuous domain, laying the groundwork for effective learning in more complex and realistic environments. To address these challenges, we present new techniques and algorithms specifically designed for the continuous reinforcement learning framework.

These developments will form the foundation for the final frameworks, closely aligned with real-world applications, presented in Chapter 7, where we further build upon and extend these algorithms to address more complex and practical scenarios.

6.1 State Space

Following the discrete state space formulation previously outlined, the transition to a continuous state space preserves the essential goal of providing the agent with localized and goal-relevant information. However, several key changes are introduced to follow the increased representational flexibility and complexity of a continuous environment.

As will be discussed in Section 6.2, the discrete latent grid is no longer sufficient to capture the variability and complexity required by the new framework. This motivates a shift toward a continuous, real-valued latent plane which better represents spatial patterns in water distribution. In this revised formulation, the agent moves in a continuous domain, where the coordinates (x, y) are real-valued and lie within the bounded range $[0, \texttt{plane\_size}] \times [0, \texttt{plane\_size}]$. At each timestep, the agent samples the interpolated water content at its current position and collects localized observations through a continuous mask centered on (x, y). This enables precise interactions with the environment and more realistic decision-making mechanisms.

Hence, while the structure of the state space proposed in this continuous framework resembles that of the discrete counterpart, the nature and semantics of the values are different. Specifically, the state includes a flattened version of a $(\texttt{mask\_size} \times \texttt{mask\_size})$ local observation window, where each element corresponds to the interpolated water content sampled from the continuous latent plane around the agent's position.

In addition to this mask, the environment internally encodes both the agent's current position (x, y) and the fixed goal position (x_g, y_g). However, in strict terms of the agent's observable state, only the local mask is exposed to the policy, in alignment with the principles of partially observable environments. This design choice enforces the partially observable Markov decision process (POMDP) formulation, forcing the agent to infer spatial context and goal proximity purely mainly from its local perceptual field.

As a consequence, the choice of `mask_size` becomes a critical design parameter in this framework. A small mask may limit the agent's ability to detect environmental gradients and identify good movement directions. On the other hand, a larger mask provides better spatial context but increases the state dimensionality and may attenuate the importance of local featu-

res. Therefore, selecting an appropriate mask size involves a trade-off between local detail and global awareness, and directly impacts the agent's capacity to generalize effective navigation strategies within the continuous domain. For these reasons, this parameter will be studied in the relative section.

6.2 Continuous Latent Plane

As introduced in the previous section, a more expressive environmental representation is required to model the continuous scenario where the agent operates. We now present in detail the construction of the continuous latent plane and how the agent moves within it.

6.3 Continuous Latent Space Representation

The environment is modeled as a continuous two-dimensional latent plane $\Omega \subset \mathbb{R}^2$, with spatial extent

$$\Omega = [0, L_x] \times [0, L_y], \tag{102}$$

where L_x and L_y represent the size of the plane along the x and y axes, respectively. In our studies, we consider a square domain with $L_x = L_y = \texttt{grid\_size}$ for simplicity.

Each point $(x, y) \in \Omega$ is associated with a real value of the scalar water content

$$w(x, y) \in [0, 1] \tag{103}$$

which varies smoothly across the domain, capturing spatial heterogeneity and environmental gradients relevant to the agent's task. These values are derived from the GPR B-scan measurements and the subsequent features extraction process described in Sections 4.2–4.3.

6.4 Perlin Noise-Based Augmentation

Moving to a continuous state space significantly increases the volume and complexity of data required to effectively represent the environment. To represent this expanded domain and facilitate robust learning, it becomes necessary to employ data augmentation techniques. In this context, we generate and augment the continuous, spatially correlated dataset of water content fields $w(x, y)$ using Perlin noise [37], a widely used procedural noise function that produces natural-looking textures and landscapes. The water function $w(x, y)$ is hence defined as a normalized multi-octave Perlin noise:

$$w(x, y) = \left(\frac{pnoise2\left(\frac{x}{s}, \frac{y}{s}; \text{octaves} = O, \text{persistence} = P, \text{lacunarity} = L, \text{base} = \text{seed}\right) + 1}{2} \right)^3 \tag{104}$$

where:

- $pnoise2$ is the two-dimensional Perlin noise function,
- s is the scale parameter controlling the noise frequency,
- O is the number of octaves controlling the level of detail,
- P is the persistence controlling the amplitude decay between octaves,
- L is the lacunarity controlling the frequency increases between octaves,

- `seed` is a random or fixed integer to ensure reproducibility.

The cubic exponentiation $(\cdot)^3$ biases the distribution toward lower water content values, emphasizing dry regions that are crucial for the agent's navigation.

6.5 Online Sampling

To better capture the dynamic changes of the environment and incorporate realistic, real-world aspects, we adopt an online sampling approach. In this setting, the latent plane is no longer precomputed and stored statically within the environmnet implementation, as we did with the discrete framework, see Section 5.2. Instead, the agent samples directly from the continuous plane at each step during training.

This dynamic sampling enables the environment to evolve over time, allowing for varying conditions and richer scenarios. Such an approach significantly enhances the representational power and flexibility of the training framework, improving the agent's capacity to generalize and adapt to non-stationary environments.

6.6 Rendering

To facilitate visualization and analysis, the continuous latent plane is discretized at a sufficiently high resolution R along each axis. This discretizatoin produces a grid of size $(L_x \cdot R) \times (L_y \cdot R)$, where each grid cell corresponds to a sampled point in the continuous domain.

Formally, the water content values on the discretized grid are given by:

$$W_{ij} = w\left(x_i, y_j\right), \quad x_i = \frac{i}{R}, \quad y_j = \frac{j}{R}, \quad i = 0, \ldots, L_x R - 1, \quad j = 0, \ldots, L_y R - 1, \tag{105}$$

where $w(x, y)$ is the continuous water content sampling function defined over the domain.

This high-resolution grid enables detailed rendering of the latent space, allowing clear visualization of spatial variations and patterns in water content. The choice of resolution R balances between rendering details and computational efficiency.

6.7 Agent Interaction with the Latent Plane

As previously mentioned, at each timestep the agent dynamically samples the local water content values around its current position $\mathbf{p}_t = (x_t, y_t) \in \Omega$. This sampling is performed by extracting a continuous localized mask of size `mask_size` $\times$ `mask_size` centered on $\mathbf{p}_t$, where each mask element corresponds to

$$m_{uv} = w(x_t + \Delta x_u, y_t + \Delta y_v), \quad u, v = -\frac{\texttt{mask\_size}}{2}, \ldots, \frac{\texttt{mask\_size}}{2} \tag{106}$$

with offsets Δx_u, Δy_v representing relative displacements in the continuous domain.

Sampling outside the domain bounds returns an invalid value, allowing the agent to detect and handle such cases.

6.8 Implementation Details

The `ContinuousPerlinLatentPlane` class (see Listing 1) implements the concepts described above, using the `noise` Python package for Perlin noise generation and NumPy for efficient numerical operations.

Listing 1: Excerpt of `ContinuousPerlinLatentPlane` class implementing Perlin noise latent plane and obstacle generation

```
class ContinuousPerlinLatentPlane:
    def __init__(self, grid_size=(10,10), resolution=100,
       seed=None):
        self.grid_size = grid_size
        self.resolution = resolution
        self.seed = seed or random.randint(1, 100)
        self.scale = 5.0
        self.octaves = 4
        self.persistence = 0.5
        self.lacunarity = 2.0

    def f(self, x, y):
        val = pnoise2(x/self.scale, y/self.scale,
                      octaves=self.octaves,
                         persistence=self.persistence,
                      lacunarity=self.lacunarity, base=self.seed)
        return ((val + 1)/2) ** 3

    def sample(self, x, y):
        if (x > self.grid_size[0] - 1.0) and (y >
           self.grid_size[1] - 1.0):
            return -1.0
        return self.f(x, y)
```

6.9 Examples of Continuous Latent Planes

To better illustrate the structure and variability of the generated continuous environments, we now present a series of visual examples. These plots depict representative samples of the latent water content fields, measured using ground penetrating radar and augmented using Perlin noise-based technique. By visualizing the spatially correlated patterns and gradients in water content, we aim to provide an intuitive understanding of the underlying landscape that the agent perceives and explores during training. Figures 25-32 showcase such examples, each corresponding to a different possible field configuration. Together, they highlight the variety of spatial features and environmental challenges the agent must face. This variety reinforces the need for precise and adaptive algorithms capable of achieving robust agent learning and generalization across heterogeneous conditions.

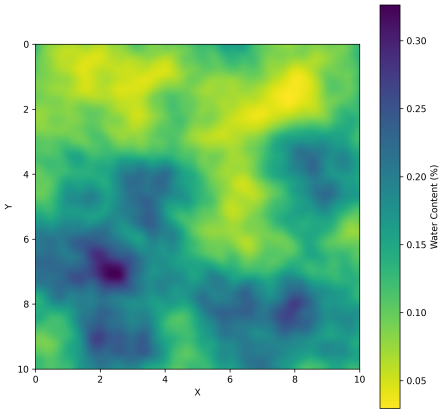


Figure 25

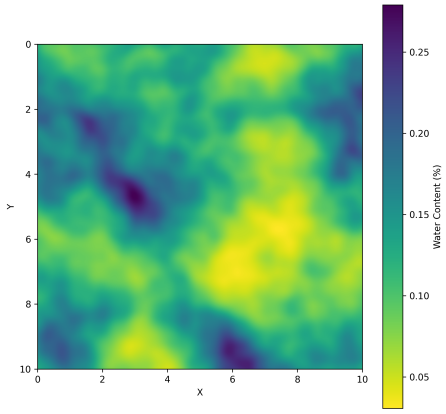


Figure 26

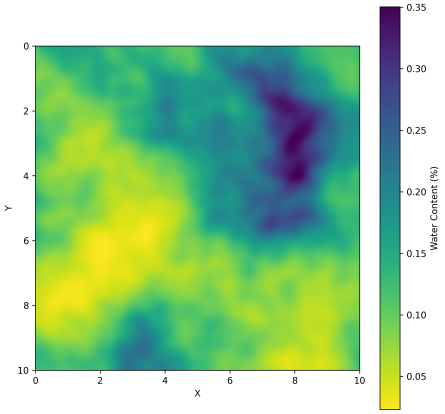


Figure 27

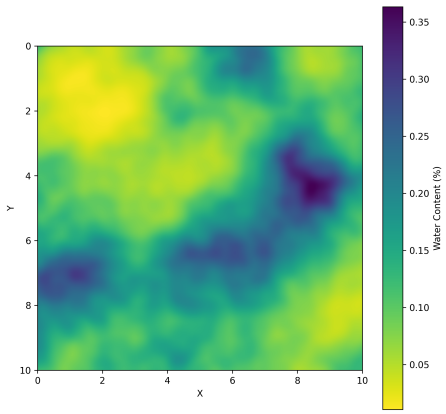


Figure 28

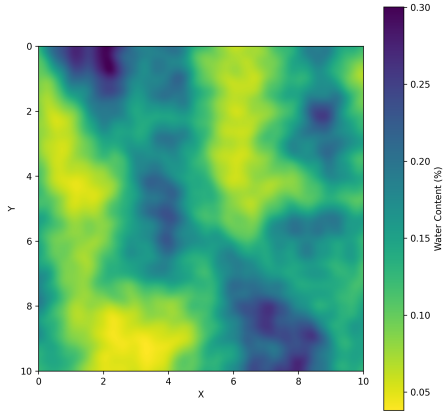


Figure 29

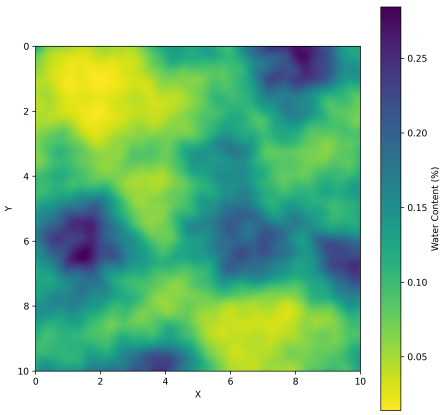


Figure 30

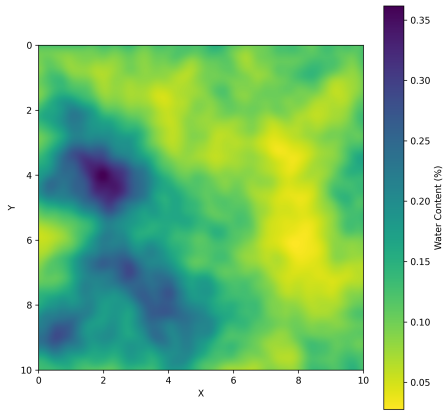


Figure 31

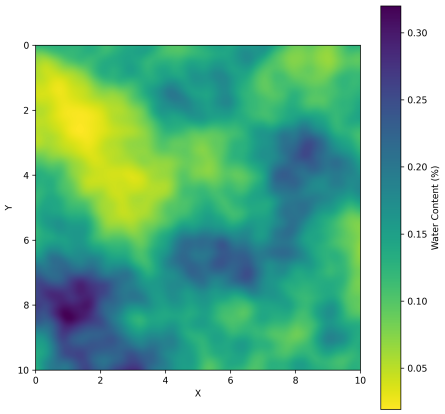


Figure 32

Figure 33: Overview of representative samples of latent water content fields augmented using Perlin noise. Each sample illustrates a particular configuration presented to the agent during training.

6.10 Final Remarks

This continuous latent plane formulation provides a realistic and accurate representation of dynamic real-world landscapes, enabling agents to learn and make decisions in partial observability settings.

The continuous nature of the state space, combined with online sampling of local observations, allows the agent to adapt to changing environmental conditions in a flexible and scalable manner.

Furthermore, it is important to note that, unlike in the discrete framework, the water content levels in this continuous setting are not manually enhanced. As a result, the contrast between dry and wet regions is often less pronounced, making it more difficult for the agent to distinguish favorable areas for navigation. This design choice preserves the natural variability of real-world conditions, but it also creates a more challenging learning environment that demands higher precision from the learning algorithms.

Overall, this framework lays a strong foundation for robust learning in continuous, dynamic environments, enabling future developments and extensions such as adaptive mask sizing or the integration of additional sensory modalities. Some of these extensions will be addressed in Chapter 7.

6.11 Action Space

Unlike the discrete setting where the agent selected from a fixed set of four possible moves (up, down, left, right), the current framework employs a continuous action spae. The agent's actions are now defined as continuous displacements along the x and y coordinates, represented by the vector

$$\mathbf{a}_t = (\Delta x_t, \Delta y_t) \in \mathbb{R}^2 \tag{107}$$

At each timestep t, the agent decides the displacement values Δx_t and Δy_t, which determine how much the agent moves in each spatial dimension:

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{a}_t = (x_t + \Delta x_t,\ y_t + \Delta y_t) \tag{108}$$

Furthermore, to more accurately simulate real-world robotic actuation constraints, we apply action clipping to the displacement values Δx_t and Δy_t. This clipping ensures that the agent's actions remain within a fixed, predefined range, reflecting the physical limitations of real robot actuators. As a result, the agent must learn to optimize its policy within these bounded action intervals.

Multiple displacement ranges were empirically tested to evaluate their influence on learning dynamics and will be presented in the next sections. Narrower ranges tend to encourage more caution and locally optimized behaviors, while broader ranges allow for fast and aggressive exploration at the risk of instability.

This continuous framework formulation allows the agent to learn precise control policies in order to follow drier paths and avoid wet areas, while effectively progressing toward the goal position. By directly learning position control policies in the continuous space, the agent dynamically adapts its trajectories in response to environmental variations. This capability is fun-

damental for real-world navigation and path planning, especially in complex and unstructured terrains where flexibility and precise adjustments are essential to achieve the tasks.

Furthermore, in Chapter 7 we will extend the state and action spaces to incorporate velocity-based controls. This extension will allow the agent to learn to command linear and angular velocities directly, providing a better implementation of realistic robotic dynamics and improving the applicability of the framework to real-world robotic systems.

6.12 Training and Evaluation Pipeline

The overall training and evaluation pipeline for the continuous scenario largely follows the setup described for the discrete case, with adaptations to handle the continuous state and action spaces.

A key difference is the choice of the reinforcement learning algorithm used to train the agents. Instead of Proximal Policy Optimization, mostly suited for discrete scenarios, we adopt the Soft Actor-Critic algorithm, an off-policy method well-suited for continuous action spaces, known for its sample efficiency, stable convergence properties and effective balance between exploration and exploitation through entropy maximization. See Chapter 3.23 for further details and properties of the algorithm.

Similarly to the discrete setting, vectorized environments are employed using `SubprocVecEnv` to enable parallel data collection and efficient training. The `VecMonitor` wrapper is maintained for logging episodic statistics such as reward, episode length, and success rate.

The agent's policy is parameterized by multi-layer perceptrons (`MlpPolicy`), which are well-suited to process continuous observation vectors such as the flattened local observation mask and the agent's continuous positional information. This architecture allows the policy to capture complex relationships within the continuous state space and outputs precise continuous actions.

Furthermore, the entropy coefficient is set to *auto*, allowing the algorithm to adaptively balance exploration and exploitation automatically. No adjustments are made to the other main hyperparameters, such as learning rate, batch size, or buffer size, since the default settings are sufficient for the given tasks.

Due to the increased complexity of the continuous state and action spaces compared to the discrete framework, more extensive samples collection is required to ensure adequate exploration and learning. Consequently, the number of parallel environments is increased significantly, from 20 in the discrete case to 80 or more, enabling faster experience collecting and more stable gradient updates. Similarly, the total number of training timesteps is significantly extended, exceeding 5 million iterations in particular cases, to allow the agent sufficient interaction time to converge to effective policies under these more challenging conditions.

The local mask size remains one of the most critical hyperparameters that influences the agent's perceptual resolution and performance. Its impact is systematically studied through a series of empirical experiments.

Agent evaluation is conducted in a separate vectorized environment to ensure isolation from the training process and to provide unbiased assessment of the agent's performance. The evaluation

setup is almost identical to that of the discrete framework, employing the same set of evaluation metrics, including cumulative rewards, episode lengths, and success rate.

Furthermore, we apply the same evaluation metrics used in the discrete framework. Specifically, agent performance is assessed based on the success rate of reaching the goal within the episode, and the average cumulative water content levels encountered throughout the agent's navigation. In this way, we can track the impact of the different reward functions on the agent's learning progress.

6.13 Reward Function Design and Evolution

The reward design for the continuous reinforcement learning framework extends the design principles already introduced in the discrete setting in Section 5.5. Although the core components are similar, their deployment in a continuous action space introduces several complexities and critical behavioral patterns.

A major challenge encountered during this transition is the agent's inclination to get stuck in local minima, often looping within locally dry areas or failing to escape particular zones. Unlike in discrete domains, where actions are bounded and structured, continuous agents have a high degree of freedom, increasing the probability of prematurely converging to suboptimal behaviours. To mitigate this, the reward design has to be very precise, encouraging both short-term gains and broader objectives such as exploration and goal-reaching.

Another issue that emerged during the design and testing process is the environmental overfitting in large-scale parallel training. With many environments running concurrently, agents occasionally developed a preference for specific configurations, achieving high performance in a limited subset while neglecting broader generalization. This phenomenon highlights the need for careful tuning of reward weights, particularly those related to spatial guidance and local dryness, to ensure robust learning across different scenarios.

For these reasons, in the following sections we first present a new baseline reward function, which serves as a reference point for all the subsequent designs. We then proceed through a sequence of increasingly refined reward functions, highlighting the key design elements, motivations, and potential problems encountered in each case. This iterative process allows us to understand the trade-offs and limitations that emerged during empirical testing.

Finally, we describe the final comprehensive reward function selected for the continuous learning tasks. This reward formulation integrates all the essential components required to ensure robust learning and generalizable learning across different environments.

In this final design, each reward term is discussed in detail, with a focus on specific mechanisms by which it influences agent behaviour, Additionally, we provide the associated hyperparameters, explaining how they were selected and tuned based on empirical performance and stability considerations during training.

6.14 Baseline Reward Function

The baseline reward function adopted in the continuous learning framework is inspired by the one previously introduced in the discrete setting (see Section 5.6). However, in this case, both core objectives, water avoidance and goal reaching, are considered from the beginning.

The reward at each timestep consists of two principal components, water content penalty and goal distance incentive.

The water content penalty term penalizes the agent based on the level of water content at its current position. The intuition is to discourage navigation on moist or saturated regions. The penalty is proportional to the samples water content value w_t such that higher moisture results in a larger negative reward.

The goal distance incentive term encourages the agent to move towards a predefined goal position. The reward is inversely related to the Euclidean distance d_t between the agent's current location and the goal. As the agent gets close to the goal, this component provides increasing positive rewards.

Formally, the baseline reward r_t at timestep t can be expressed as:

$$r_t = -\alpha \cdot w_t - \beta \cdot d_t \tag{109}$$

where:

- w_t is the water content at the agent's current position,
- $d_t = \|\mathbf{p}_t - \mathbf{p}_{\text{goal}}\|$ is the Euclidean distance to the goal,
- $\alpha > 0$ and $\beta > 0$ are scalar coefficients that balance the relative importance of water minimization versus goal-directed movement.

This reward formulation ensures that the agent is simultaneously encouraged to avoid high-moisture regions and to make consistent progress toward the goal. The exploration aspect is handled by the SAC algorithm employed, optionally supported by custom reward components. However, while conceptually simple, this baseline reward function presents several limitations and areas for improvement.

One major challenge lies in the careful tuning of the scalar coefficients α and β, which respectively control the emphasis on water minimization and goal-reaching behaviour. Selecting appropriate values for these weights is challenging. If either component dominates, the agent may prioritize just one of the two objectives. Moreover, identifying a pair of values that generalizes well across different environments is difficult, as the optimal balance between the two objectives may vary depending on the moisture distribution and environmental structure. For our specific application and field conditions, the optimal values were found to be:

$$\alpha = 2 \quad \text{and} \quad \beta = 0.1 \tag{110}$$

Figure 34 presents the performance of the agent trained for $2,500,000$ timesteps using the baseline reward function, evaluated on a batch of randomly selected environments from the evaluation dataset. Both the success rate and the average cumulative water content encountered during episodes were tracked, following the same methodology as in the discrete scenario (see Chapter 5).

Empirical observations revealed that, in many training scenarios, agents trained with this baseline reward function were inclined to learn direct, nearly linear paths from the starting point

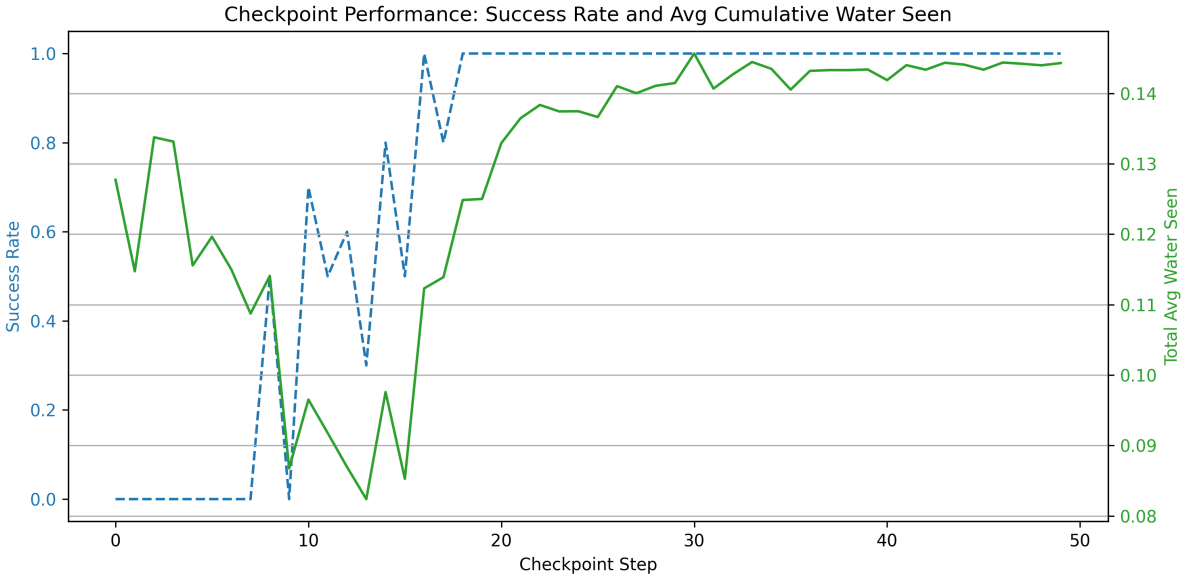


Figure 34: Performance evaluation of the baseline agent over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

to the goal. While they successfully avoided high-moisture regions along these trajectories, the agents neglected alternative paths that could have offered a lower cumulative water exposure. This behaviour indicates a lack of contextual awareness and limited adaptability to more complex environmental configurations.

To address these limitations, subsequent reward designs introduce additional components and constraints specifically designed to improve generalization, exploration, and context-aware navigation strategies.

6.15 Improved Reward Function

To overcome some of the limitations identified in the baseline design, an improved reward function is introduced. This revised formulation retains the original objectives, encouraging progress toward the goal and avoidance of wet regions, but introduces a series of modifications that enhance the agent's ability to generalize and behave robustly in more complex scenarios.

The key improvements include:

1. Local Moisture Awareness: instead of relying only on the water content w_t at the agent's current position, the improved function evaluates the water levels within the local observation mask centered around the agent. This encourages the agent to make decisions based on its immediate surroundings rather than single-point estimates.

2. Water Content Term: the reward compares the agent's current water content value w_t with the minimum water value within the local observation mask $\mathcal{M}_t$ of size $N \times N$, where $N = \texttt{mask\_size}$. This encourages the agent to follow not only relatively drier spots in its proximity but also absolutely dry regions.

$$r_{\text{water}} = \begin{cases} +\lambda_1 & \text{if } w_t \leq T_{\text{dry}} \text{ and } w_t \approx \min(\mathcal{M}_t) \\ +\lambda_2 & \text{if } w_t \leq T_{\text{dry}} \\ -\lambda_3 & \text{otherwise} \end{cases} \tag{111}$$

Here, w_t is also compared with a water threshold that ensures absolute dryness, rather than only relative dryness with respect to the local minimum in the mask. This threshold requires tuning depending on the overall moisture conditions of the environments.

3. Distance Progress Incentive: similar to the baseline, the agent is rewarded for decreasing its Euclidean distance to the goal. Let $\mathbf{p}_t$ denote the agent's position at time t, and $\mathbf{g}$ the goal position. The progress reward is given by:

$$r_{\text{progress}} = -\lambda_4 \cdot d_t = -\lambda_4 \cdot \|\mathbf{p}_t - \mathbf{g}\| \tag{112}$$

where $\lambda_4 > 0$ is a scaling factor that controls how strongly the Euclidean distance to the goal influences the reward.

Two different formulations of the distance d_t can be employed:

- Unnormalized Euclidean Distance: the raw distance in environment coordinates. It is straightforward to compute and preserves the absolute spatial meaning of agent positions. However, it requires careful adjustment of the scaling factor λ_4 based on the size of the environment. In large environments, d_t can become significantly larger, especially during the early stages of an episode, which can lead to dominant penalties in the all reward structure. To maintain balance with other reward terms, especially with the water content component, λ_4 must be decreased accordingly.

- Normalized Euclidean Distance: the distance is normalized by dividing by the maximum possible distance in the environment, typically the diagonal length of the plane. This normalization constraints the distance to a fixed range, usually $[0, 1]$, and provides more consistent reward scaling across environments of varying size. As a result, the corresponding values of λ_4 can remain relatively stable across different settings, thus simplifying the reward tuning, and are typically higher than those required when using unnormalized distances.

4. Step Penalty: a small constant penalty $r_{\text{step}} = -\lambda_5$ is applied at every timestep to discourage unnecessary movements and incentivize efficient planning.

5. Goal Achievement Reward: a sharp bonus $r_{\text{goal}} = +\lambda_6$ is provided when the agent reaches within a small distance ϵ from the goal:

$$r_{\text{goal}} = \begin{cases} +\lambda_6 & \text{if } \|\mathbf{p}_t - \mathbf{g}\| < \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{113}$$

6. Boundary Penalty: a negative reward $r_{\text{boundary}} = -\lambda_7$ is applied whenever the agent attempts to move beyond the spatial limits of the environment and its position is clipped to

remain within valid bounds.

7. Explore Reward: a positive reward $r_{\text{explore}} = +\lambda_8$ is given when the agent visits previously unexplored regions, encouraging exploration, while revisiting areas results in a penalty $r_{\text{explore}} = -\lambda_9$, discouraging redundant behaviours. Formally:

$$r_{\text{explore}} = \begin{cases} +\lambda_8 & \text{if the current region is newly visited} \\ -\lambda_9 & \text{if the current region has been visited before} \end{cases} \tag{114}$$

To determine whether a region has been visited, the agent maintains a list of previously visited positions. A region is considered already visited if the Euclidean distance between the agent's current position $\mathbf{p}_t$ and any previously stored position $\mathbf{v} \in \mathcal{V}$ satisfies:

$$\|\mathbf{p}_t - \mathbf{v}\| < \tau \tag{115}$$

where τ is a predefined threshold. If no such $\mathbf{v}$ exists, the current position is appended to the set $\mathcal{V}$ of visited regions.

8. Timeout penalty: if the agent fails to complete the task within a maximum number of steps $T_{\max}$, a termination penalty is applied:

$$r_{\text{timeout}} = \begin{cases} -\lambda_{10} & \text{if } t = T_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{116}$$

In our project, we set $T_{\max} = 150$ steps, which is appropriate for the environments we tested, with a maximum size of 15×15. However, this parameter should be adjusted proportionally to the size and complexity of the environment, as well as to any clipping applied in the action space, to ensure effective training and prevent premature episode termination.

Putting them all together, the total reward at timestep t is computed as:

$$r_t = r_{\text{water}} + r_{\text{progress}} + r_{\text{step}} + r_{\text{goal}} + r_{\text{explore}} + r_{\text{boundary}} + r_{\text{timeout}} \tag{117}$$

Table 2 summarizes the complete set of key hyperparameters used in the reward formulation.

This reward design incorporates richer local context, encourages forward progress and efficient planning, and penalizes inefficient or unsafe behavior. It represents a key step toward the final reward system, although some challenges remain.

In particular, the agent still tends to overexploit its initial trajectories, resulting in limited exploration. In addition, the agent often gets attracted to nearby highly saturated areas, from which it struggles to escape, leading to suboptimal navigation and occasional repetitive loops.

Figure 35 presents the performance metrics of the agent trained with this new reward function design defined in Equation 117. As shown in the plot, the agent consistently reaches the goal

Table 2: Reward components with corresponding hyperparameters and their values for the continuous framework

Reward Component	**Hyperparameters to Tune**	**Values Used**
r_t^{water}	λ_1, λ_2, λ_3, T_{dry}	$\lambda_1 = +1.0$, $\lambda_2 = +0.2$, $\lambda_3 = +1.5$, $T_{\text{dry}} = 0.15$
r_t^{progress}	λ_4	$\lambda_4 = +0.1$
r_t^{step}	λ_5	$\lambda_5 = +0.1$
r_t^{goal}	λ_6, ϵ	$\lambda_6 = +7.0$, $\epsilon = 0.8$
r_t^{boundary}	λ_7	$\lambda_7 = +1.0$
r_t^{explore}	λ_8, λ_9, τ	$\lambda_8 = +0.2$, $\lambda_9 = +0.5$, $\tau = 0.3$
r_t^{time}	λ_{10}, $T_{\max}$	$\lambda_{10} = +1.5$, $T_{\max} = 150$

across different scenarios. Compared to the baseline, this new reward design achieves a significant improvement by reducing the average cumulative water content. Following empirical results, the agent now reaches the goal by actually following specific dry paths rather than mainly moving along the diagonal trajectory, demonstrating more efficient navigation. The comparison of the cumulative water content between the two reward function designs is shown in Figure 36.

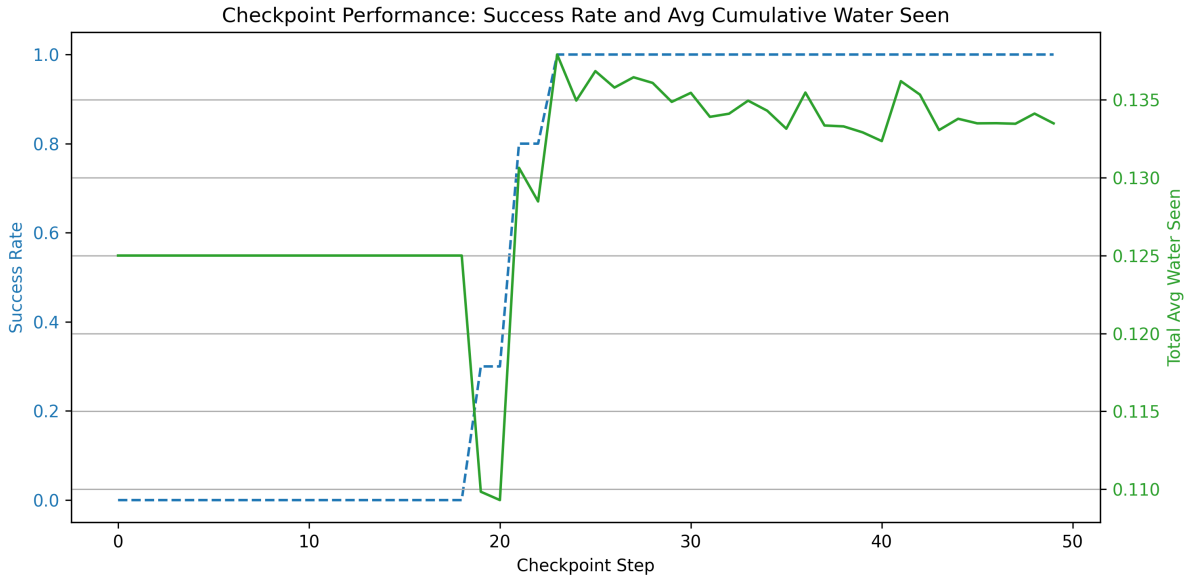


Figure 35: Performance evaluation of the agent with the new reward function design over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

The remaining limitations identified, such as limited exploration and the inclination to get attracted near saturated zones, are addressed in the final reward function, which is presented in the next section.

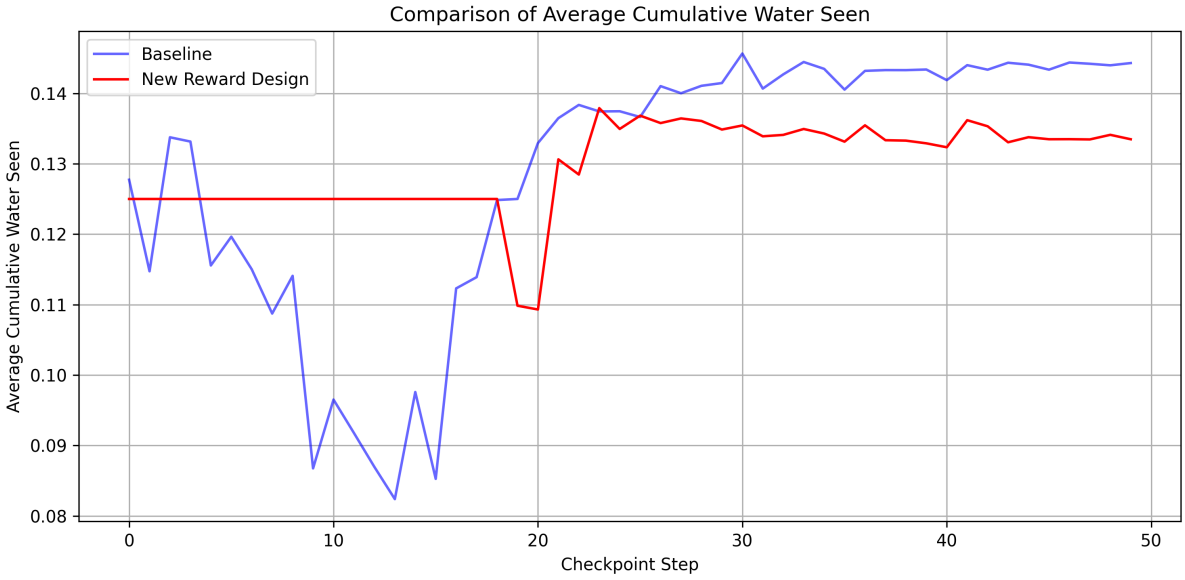


Figure 36: Average cumulative water content comparison between the baseline reward function in blue, and the new reward function design in red.

6.16 Final Comprehensive Reward Function

Building upon the improved reward function just presented, we now propose a final, comprehensive reward function design that addresses its remaining limitations. This enhanced formulation introduces several key innovations to better balance the exploration-exploitation trade-off and robustness across different environmental conditions.

In particular, a new distance progress component has been adopted, based on the concept first introduced in the final reward function for the discrete setting (see Section 5.9.3). Instead of being proportional to the absolute Euclidean distance to the goal, the reward is now computed based on whether the agent improves its distance relative to the previous step. This approach aligns more naturally with the partially observable nature of the environments, providing a more flexible and generalizable framework.

While this modification successfully mitigates issues related to over-exploitation and improves generalization, it also introduces increased variability in the rewards. This arises from the continuous nature of the environments and the complexity of the real-world conditions. Consequently, although the revised reward function effectively captures better real-world conditions, the resulting reward variability should be considered as an inherent trade-off effect.

1. Enhanced Water Content Reward To provide a more robust representation of local moisture conditions, the water content reward now combines both minimum and average water values within the agent's local observation mask $\mathcal{M}_t$ of size $N \times N$:

$$r_{\text{water\_final}} = \begin{cases} +\lambda_1 & \text{if } w_t \leq T_{\text{dry}} \text{ and } w_t \approx \min(\mathcal{M}_t) \\ +\lambda_2 & \text{if } w_t \leq T_{\text{dry}} \\ +\lambda_3 & \text{if } \text{avg}(\mathcal{M}_t) \leq T_{\text{avg}} \\ -\lambda_4 & \text{if } \text{avg}(\mathcal{M}_t) > T_{\text{avg}} \\ -\lambda_5 & \text{otherwise} \end{cases} \tag{118}$$

where w_t is the water content at the agent's position at time t, $\text{avg}(\mathcal{M}_t)$ is the average water content in the mask, and and T_{dry}, T_{avg} are dryness thresholds tuned according to environmental conditions.

This multi-criteria reward encourages the agent not only to prefer the driest local spot but also to avoid extended wet regions, thus promoting more effective and strategic navigation behaviours.

2. Generalized Distance Progress Term Rather that directly penalizing the agent proportionality to its distance d_t to the goal, we adopt a different approach that rewards the agent based on its progress relative to the previous timestep. Specifically, let d_{t-1} and d_t denote the unnormalized Euclidean distances to the goal at timesteps $t-1$ and t, respectively. We compute the difference $\Delta d = d_{t-1} - d_t$ to quantify how much closer the agent has moved towards the goal.

The reward is then assigned according to the following rule:

$$r_{\text{progress\_diff}} = \begin{cases} +\lambda_6 & \text{if } \Delta d > \delta \\ -\lambda_7 & \text{otherwise} \end{cases} \tag{119}$$

Here, $\delta > 0$ is a small threshold that defines a minimum meaningful improvement in distance, preventing the agent from receiving rewards for negligible or oscillatory movements. The constant factors $\lambda_6 > 0$ and $\lambda_7 > 0$ control the magnitude of the rewards.

This difference-based progress reward formulation offers several advantages over a direct proportional penalty. By focusing on relative improvement, it promotes greater exploration, allowing other components of the reward function to better guide the agent along optimal trajectories, while still encouraging consistent progress toward the goal.

Moreover, the use of relative progress allows the agent to operate even under partial observability or uncertainty about the exact goal locations. Since the reward depends on the change in distance rather that its absolute value, the agent does not require perfect knowledge of the goal's position to learn useful behaviours. This flexibility often results in richer exploration patterns and more robust policy learning. This could be particularly useful in complex or noisy environments where direct distance information may be unreliable or unavailable.

As a potential future improvement, the constant values λ_6 and λ_7 could be dynamically scaled in proportion to the difference Δd. This would preserve the balance between reward sensitivity and stability, allowing the system to adapt more naturally to different precision requirements and exploration scenarios.

3. Exponential Goal Attractor Another key component of our final reward design is the use of an exponential goal attractor function, which provides a smooth and nonlinear incentive for the agent to approach the goal. Specifically, the attractor is defines as a sigmoid-shaped function of the distance to the goal, given by

$$\text{goal\_attraction} = \lambda_8 \cdot \frac{1}{1 + \exp\left(\lambda_9 \cdot (d_t - \lambda_{10})\right)} \tag{120}$$

where d_t is the Euclidean distance between the agent and the goal at timestep t. This functions sharply increases as the agent moves within roughly $\lambda_{10} > 0$ units of the goal, transitioning from values near zero at larger distances to values approaching λ_8 near the target. The parameter $\lambda_9 > 0$ controls the steepness of the sigmoid curve, determining how rapidly this transition occurs.

This attractor provides a strong and smoothly increasing gradient that encourages the agent to close the final gap to the goal. This property is particularly beneficial in scenarios where the goal is located in unfavorable conditions, such as near wet areas or other challenging conditions.

Figure 37 shows possible goal attraction functions derived from different combinations of the hyperparameters involved. By varying the values of λ_8, λ_9 and λ_{10}, we can observe how each parameter influences the shape of the attraction curve, and consequently, the agent's behaviour. This figure provides a clear visual representation of the impact of the hyperparameters on the overall goal attraction function.

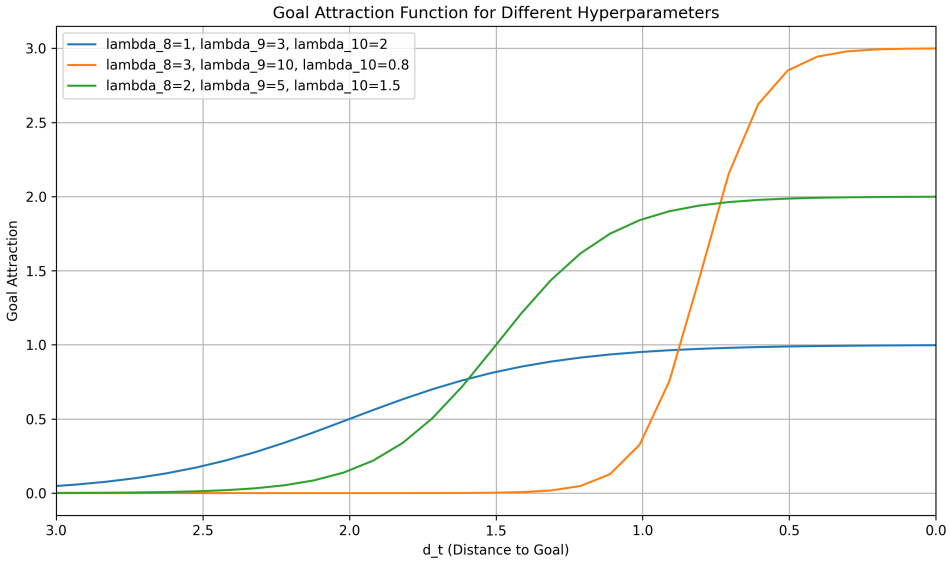


Figure 37: Comparison of goal attractor functions for different combinations of hyperparameters. The second combination, shown in orange, is the one we selected for the final reward design, as it proves to be the most suitable for our tasks.

4. Other Components All previously introduced terms remain part of the final reward function, including:

- Goal achievement bonus $r_{\text{goal}} = +\lambda_{11}$ for reaching the goal within radius ϵ,
- Step penalty $r_{\text{step}} = -\lambda_{12}$ to encourage efficiency,
- Boundary penalty $r_{\text{boundary}} = -\lambda_{13}$ for clipped out-of-bounds actions,
- Timeout penalty $r_{\text{timeout}} = -\lambda_{14}$ upon exceeding maximum episode length $T_{\max}$,
- Exploration incentive component for improving coverage and environmental awareness. Formally defined as:

$$r_{\text{explore}} = \begin{cases} +\lambda_{15} & \text{if the current region is visited for the first time} \\ -\lambda_{16} & \text{if the region has been previously visited} \end{cases} \tag{121}$$

5. Total Reward Computation The total reward at timestep t is then computed as:

$$r_t = r_{\text{water\_final}} + r_{\text{progress\_diff}} + r_{\text{goal\_exp}} + r_{\text{explore}} + r_{\text{step}} + r_{\text{goal}} + r_{\text{boundary}} + r_{\text{timeout}} \tag{122}$$

This final reward design successfully addresses the limitations of earlier versions by combining an improved distance progress metric, a richer local moisture evaluation and a smooth and informative goal attraction signal. These elements jointly promote efficient navigation, robust exploration, and improved generalization across different terrain conditions. Empirical results confirm that this comprehensive reward significantly reduces issues such as early trajectory exploitation, agent looping, and stagnation near difficult terrain features, leading to more reliable and adaptable policies.

The complete set of hyperparameter values used in this final reward function is summarized in Table 3.

Table 3: Reward components of the final reward function, their corresponding hyperparameters and values for the continuous framework

Reward Component	**Hyperparameters to Tune**	**Values Used**
$r_t^{\text{water\_final}}$	λ_1, λ_2, λ_3, λ_4, λ_5 T_{dry}, T_{avg}	$\lambda_1 = +1.2$, $\lambda_2 = +0.2$, $\lambda_3 = +0.5$, $\lambda_4 = +1.0$, $\lambda_5 = +1.0$, $T_{\text{dry}} = 0.1$, $T_{\text{avg}} = 0.22$
$r_t^{\text{progress\_diff}}$	λ_6, λ_7, δ	$\lambda_6 = +2.0$, $\lambda_7 = +4.0$, $\delta = +0.2$
$r_t^{\text{goal\_exp}}$	λ_8, λ_9, λ_{10}	$\lambda_8 = +3.0$, $\lambda_9 = +10.0$, $\lambda_{10} = +1.0$
r_t^{goal}	λ_{11}, ϵ	$\lambda_{11} = +11.0$, $\epsilon = 0.8$
r_t^{step}	λ_{12}	$\lambda_{12} = +0.3$
$r_t^{\text{boundaries}}$	λ_{13}	$\lambda_{13} = +1.0$
r_t^{time}	λ_{14}, $T_{\max}$	$\lambda_{14} = +1.5$, $T_{\max} = 150$
r_t^{explore}	λ_{15}, λ_{16}, τ	$\lambda_{15} = +1.0$, $\lambda_{16} = +0.5$, $\tau = 0.3$

Figure 38 presents the performance metrics of the agent trained with this comprehensive reward function design. The plot displays the best-performing checkpoints saved during the training,

recorded each time a new best mean reward was achieved. This approach highlights the true progression of the agent's performance over the learning progress, since it is important to remember that variability in the reward signal remains due to the inherent complexity of the environment and the new distance reward logic.

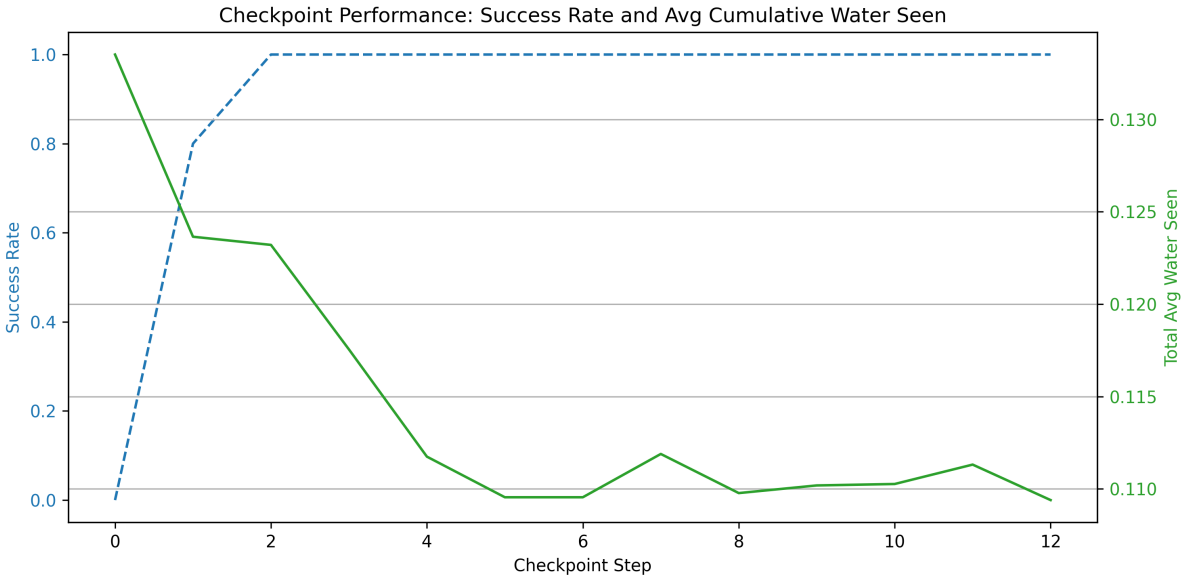


Figure 38: Performance evaluation of the agent with the new reward function design over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

As shown in the plot, the agent is able to reduce the average cumulative water content level to approximately 0.110, a substantial improvement compared to the previous two reward functions (see Figure 36), where the asymptotic values were around 0.144 for the baseline and 0.133 for the second reward design. Further tests reveal that this performance gap increases as the size and complexity of the environments increase, underscoring the critical importance of the final comprehensive reward function here presented. Its ability to effectively guide the agent through more challenging and variable scenarios makes it essential for achieving robust and scalable learning outcomes within the scope of the project. Furthermore, in environments where reaching the goal strictly via dry paths is impossible, the agent learns to position itself at the closest and safest location near the goal. This behaviour mainly comes from the more general distance progress reward component presented before, which does not excessively pressure the agent to directly reach the goal but instead encourages it to follow and select the best possible path to get as close as possible. This highlights the important duality of the objective of this work. In scenarios where both dry path following and goal reaching cannot be fully achieved, prioritizing water-aware navigation over strict goal attainment is crucial for meaningful performance and tasks.

6.17 Results

This section presents a synthesis of the key outcomes obtained within the continuous learning framework, focusing on the agent's performance across various training configurations. As in the discrete case, all evaluated policies consistently achieved a 100% success rate in the goal-reaching objective, confirming the agent's reliable path-following capability across different continuous environments. Consequently, our analysis centers on the agent's ability to minimize water exposure, which becomes the more discriminative criterion for evaluating policy effectiveness in this setting.

Figure 39 presents the average cumulative water level encountered during the learning process for both the baseline and intermediate reward formulations.

The baseline reward function, depicted in blue, lacks sufficient structure to encourage water-sensitive behaviour. As a result, the agent converges to an average cumulative water level of approximately 0.144, indicating limited environmental awareness and suboptimal navigation performance.

The intermediate reward formulation, represented in red, introduces key improvements, including refined local water-awareness shaping and enhanced distance-based incentives. These improvements enable the agent to effectively learn how to avoid high-water content regions, reducing the average cumulative water exposure to around 0.133.

Although the numerical improvement over the baseline may appear modest, the performance gap becomes more significant as the environment increases in size and complexity, where more informed and adaptive behaviour is essential for efficient exploration and successful task execution.

To further improve policy stability across different environments, a final reward structure was introduced, which incorporates improved reward components for both water awareness and goal-directed navigation. Specifically, the water-awareness term leverages particular features extracted from the agent's local observation mask, using both the minimum and average water content as references to guide decision-making. In parallel, the distance-based components were refined to better reflect partial observability concepts, encouraging consistent and efficient progress toward the goal across different environmental conditions.

This refined formulation leads to a substantial reduction in average water exposure, reaching an asymptotic value of approximately 0.110. However, this improvement comes with a slight increase in variability, which is inherently linked to the nature of the newly introduced reward components.

Figure 40 illustrates both the success rate and the average cumulative water content observed at the best-performing model checkpoints during training. The plot offers a direct view of the model's learning dynamics at its peak performance points.

It is important to note that this plot is conceptually different from Figure 39, as both visualizations report the same performance metrics but under different environmental formulations and conditions.

For a more detailed discussion of each individual reward formulation, including the specific

components introduced and their respective impacts on agent behaviour, refer to the earlier sections where each reward design is described and analyzed in depth.

The selection of hyperparameters followed a dual evaluation strategy, combining both quantitative metrics and qualitative insights. Quantitatively, the success rate and the average cumulative water content were used to evaluate agents trained in different configurations. In parallel, visual analyses of the agent trajectories provided a deeper understanding of their behavioural patterns, helping to verify alignment with the intended objectives. This combined evaluation approach enabled informed adjustments to key hyperparameters, resulting in policies that not only achieved strong numerical performance, but also demonstrated clear, objectives-aligned behavioural patterns.

Overall, these findings underscore the importance of iterative reward refinement and validate the use of progressive shaping strategies in continuous environments.

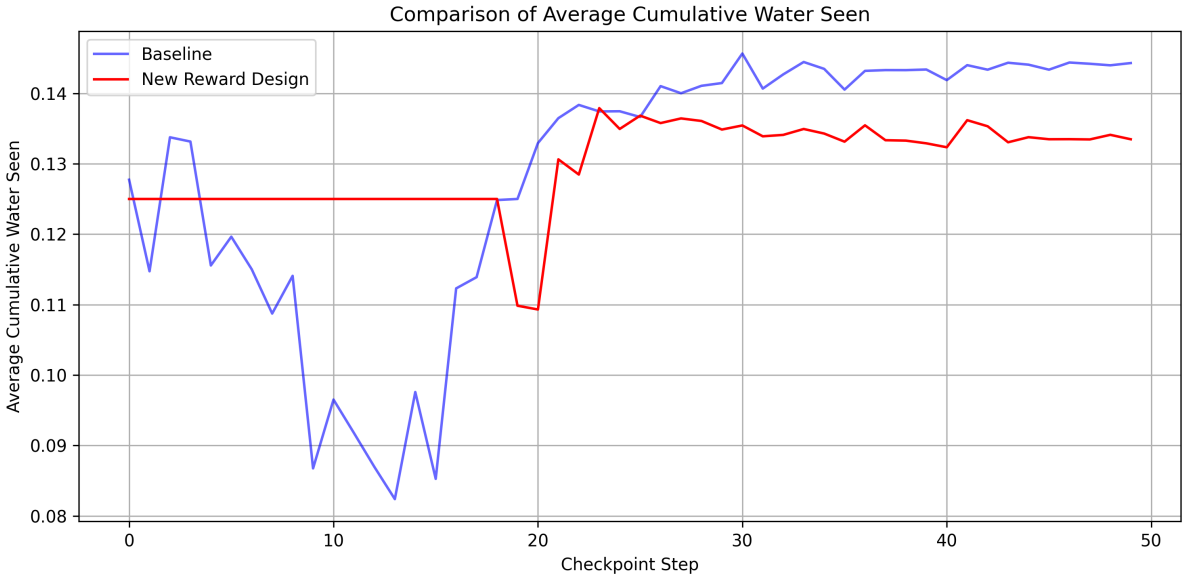


Figure 39: Average cumulative water content comparison between the baseline reward function in blue, and the intermediate reward function design in red.

To conclude, Figures 41a–41b provide qualitative insight into the agent's behaviour under the final reward formulation. These examples illustrate successful navigation across different field configurations, where the agent consistently starts from the top-right corner and moves toward the designated goal area, indicated by a yellow square located in the bottom-left region. The goal is considered reached as soon as the agent enters this square, regardless of the specific side or direction of entry.

These trajectories are the result of the training process, showcasing the agent's ability to generalize its policy across different scenarios. Each red dot in the visualization corresponds to a single step take by the agent, reflecting a specific decision made during the navigation. These visual outcomes emphasize the effectiveness of the final reward design. The agent consistently avoids high water content areas while maintaining a reliable path to the goal.

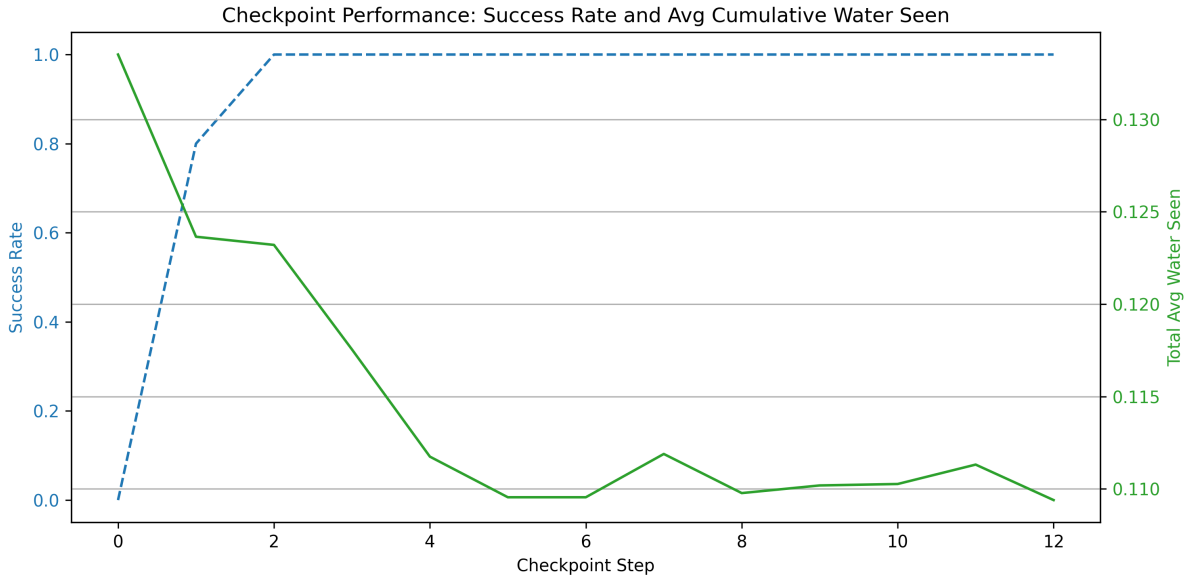
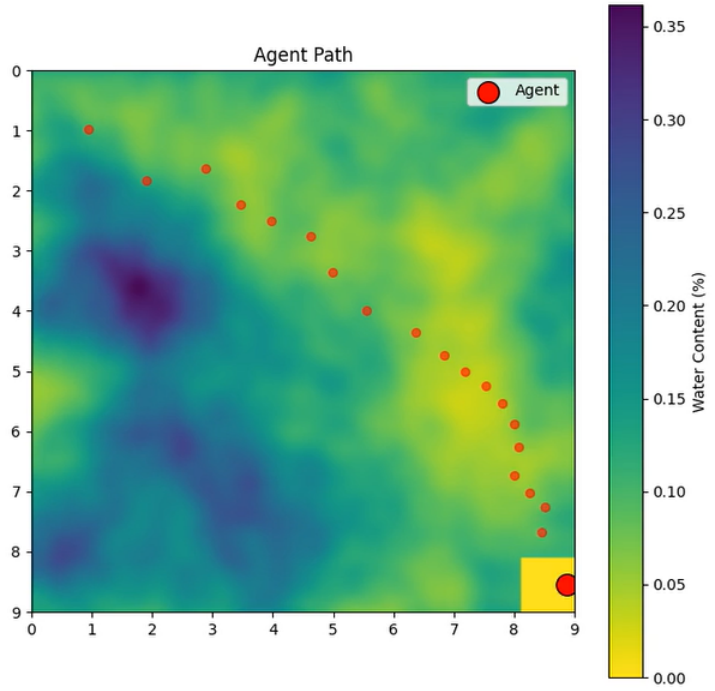
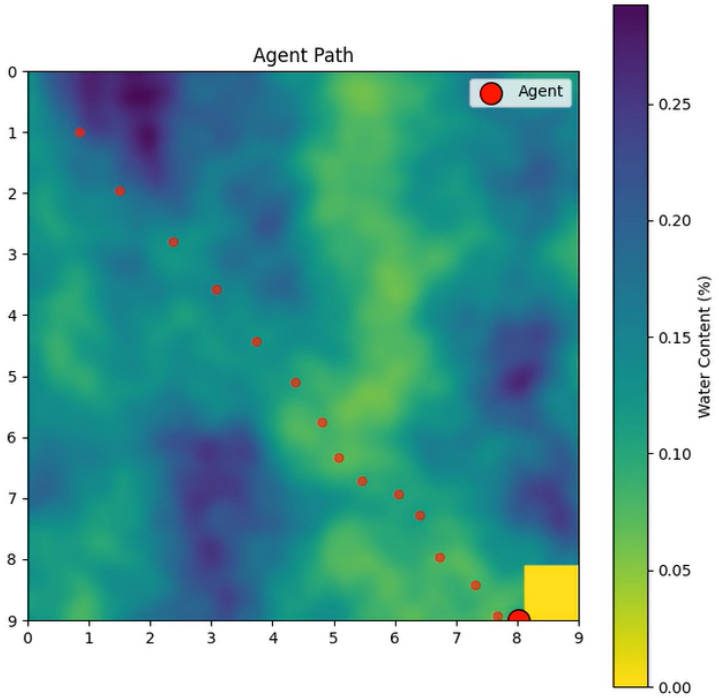


Figure 40: Performance evaluation of the agent with the final reward function design over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.



(a)



(b)

Figure 41: Examples of final trajectories of the agent

7 Advanced Continuous Reinforcement Learning Framework

Up to this point, we have explored both the discrete and continuous reinforcement learning frameworks, focusing on two primary objectives: minimizing water content exposure and successfully reaching defined goals. We have studied the design of tailored reward functions, the implications of the environment modeling, and the agent's learning behavior in these settings.

In this chapter, we take a step forward in the continuous learning framework by integrating the subsurface information used so far, specifically the spatial distribution of water content, with surface-level data obtained from conventional autonomous navigation sensors such as LiDARs and cameras. This sensor fusion approach aims to create a more comprehensive understanding of the environment, enabling agents to make more informed decisions that account for both visible and hidden terrain characteristics.

Hence, we introduce and test novel reinforcement learning algorithms that incorporate fused sensory inputs. These models will evaluate how subsurface moisture information, when combined with surface observations, influences navigation policies. In particular, we will introduce field obstacles and analyze the agent's response in this new perceptual context. This represents a key step toward robust and environmentally-aware autonomous systems capable of navigating real-world agricultural and natural environments.

Finally, to better reflect real-world deployment conditions, we transition our control strategy from position-based to velocity-based modeling. In the previous chapters, the agent was trained to select positional displacements, direct movements from its current location to a desired one. However, most autonomous mobile robots in real-world applications operate under low-level controllers that accept continuous velocity commands rather than absolute positions. Therefore, in this chapter, we adopt a velocity-based control framework in which the agent learns to produce appropriate linear and angular velocities to achieve its navigation objectives.

This shift introduces additional complexity, as the agent must now account for its orientation and the continuous nature of motion over time. However, it enables a more realistic representation of robotic control and facilitates the deployment of trained policies on actual hardware platforms. Consequently, we evaluate reinforcement learning algorithms under this new paradigm, developing policies that are both effective in simulation and transferable to real-world, velocity-controlled robotic systems.

7.1 Obstacle Modeling

To simulate physical obstacles within the environments, we generate a discrete obstacle mask $\mathcal{O}$ over the spatial domain Ω by randomly sampling obstacle centers according to a prescribed obstacle density ρ. This density parameter determines the approximate ratio of obstacle-covered area relative to the total area of the field. Each sampled obstacle center (x_i, y_i) is mapped to a grid coordinate (c_x, c_y), and a square patch of cells centered at this coordinate is marked as occupied. The size of each obstacle is controlled by a tunable parameter r_{obstacle}, which defines the half-width of the square region.

Formally, the obstacle mask is defined as:

$$\mathcal{O} = \bigcup_{i=1}^{N_{\text{obstacle}}} \left\{ (x, y) \in \Omega \;\middle|\; |x - c_x^{(i)}| \leq r_{\text{obstacle}},\ |y - c_y^{(i)}| \leq r_{\text{obstacle}} \right\}, \tag{123}$$

where $(c_x^{(i)}, c_y^{(i)})$ are the integer-valued grid coordinates of sampled obstacle centers and $N_{\text{obstacle}} \approx \rho \cdot |\Omega|$.

This square-shaped obstacle mask emulates data from surface-level sensors such as LiDARs or depth cameras. At each time step t, the agent receives a local observation composed of two parts, a patch of subsurface water content values $\mathcal{M}^{(t)} \in \mathbb{R}^{\texttt{ms}\times\texttt{ms}}$, and a binary obstacle mask $\mathcal{B}^{(t)} \in \{0,1\}^{\texttt{ms}\times\texttt{ms}}$, where $\text{ms} = \texttt{mask\_size}$.

Formally, the combined local observation is expressed as:

$$\mathcal{O}^{(t)} = \left(\mathcal{M}^{(t)}, \mathcal{B}^{(t)}\right) \tag{124}$$

This fusion of subsurface and surface sensing allows the agent to make navigation decisions based on terrain feasibility and obstacle avoidance.

In conclusion, we adopt a discrete grid-based representation of obstacles rather than a continuous one primarily due to considerations of computational efficiency and practical applicability. The main objective of this framework is to study how the agent perceives and reacts to new scenarios in real time. A discrete formulation enables rapid occupancy checks, simplifies integration with the previously developed frameworks, and supports online interaction and learning without requiring costly geometric computations.

Moreover, modeling obstacles over a grid is a reasonable abstraction from a robotic perspective. In practice, safety buffers are often applied around obstacles to account for localization uncertainty and motion control imprecision. Hence, representing obstacles with padded regions inherently counts for this safety margin, eliminating the need to determine their exact shape or boundary at fine spatial resolution. This approach allows the agent to treat obstacle avoidance conservatively while maintaining high computational efficiency.

7.2 New State and Action Spaces

Building upon the continuous learning framework introduced in Chapter 6, both the state and action spaces have been extended to incorporate obstacle information and to better reflect realistic control paradigms.

7.3 State Space

The state space is augmented to include surface-level sensor data in addition to the subsurface water content information. Specifically, at each time step t, the agent receives:

- A local patch of subsurface water content values, represented as a matrix

$$\mathcal{M}^{(t)} \in \mathbb{R}^{\texttt{mask\_size}\times\texttt{mask\_size}} \tag{125}$$

- A corresponding binary obstacle mask from surface sensors

$$\mathcal{B}^{(t)} \in \{0,1\}^{\texttt{mask\_size}\times\texttt{mask\_size}} \tag{126}$$

These two matrices are flattened and concatenated to form the core of the agent's perceptual input:

$$\mathbf{s}_{\text{mask}}^{(t)} = \left[\text{vec}\left(\mathcal{M}^{(t)}\right),\ \text{vec}\left(\mathcal{B}^{(t)}\right)\right] \in \mathbb{R}^{2\times\texttt{mask\_size}^2} \tag{127}$$

where vec$(\cdot)$ denotes the vectorization operator converting the matrix into a flattened vector.

Additionally, since, as we will discuss shortly, the action space consists of linear and angular velocities, the state vector is augmented with the agent's orientation information through the sine and cosine of the heading angle θ^t, along with the agent's current linear and angular velocities. Formally, the extended state representation at time t is given by:

$$\mathbf{s}^{(t)} = \left[\mathbf{s}_{\text{mask}}^{(t)},\ \cos(\theta^{(t)}),\ \sin(\theta^{(t)}),\ v^{(t)},\ \omega^{(t)}\right] \in \mathbb{R}^{2\times \texttt{mask\_size}^2+4} \tag{128}$$

where

- $\mathbf{s}_{\text{mask}}^{(t)}$ is the concatenated flattened vector of subsurface water content and surface obstacle masks
- $\cos(\theta^{(t)})$ and $\sin(\theta^{(t)})$ represent the agent's heading orientation
- $v^{(t)}$ and $\omega^{(t)}$ are the linear and angular velocities at the current time step

This representation effectively captures the agent's local perceptual inputs alongside its dynamic state, enabling learning algorithms to make informed decisions based on both environmental context and motion state.

7.4 Action Space

In contrast to previous chapters where the agent directly controlled positional displacements, the current framework employs a velocity-based action space.

Specifically, at each time step t, the agent learns an action vector consisting of incremental changes in linear and angular velocities, denoted as

$$\mathbf{a}^{(t)} = \begin{bmatrix} \Delta v^{(t)} \\ \Delta \omega^{(t)} \end{bmatrix} \in \mathbb{R}^2 \tag{129}$$

where $\Delta v^{(t)}$ and $\Delta \omega^{(t)}$ represent the increments to teh linear and angular velocities, respectively.

The actual commanded velocities are updated by integrating these increments over time:

$$v^{(t+1)} = v^{(t)} + \Delta v^{(t)}, \quad \omega^{(t+1)} = \omega^{(t)} + \Delta \omega^{(t)} \tag{130}$$

subject to predefined bounds $v_{\min} \leq v^{(t+1)} \leq v_{\max}$ and $\omega_{\min} \leq \omega^{(t+1)} \leq \omega_{\max}$ to ensure safe and physically feasible commands.

The agent's orientation $\theta^{(t)}$ is updated by integrating the angular velocity and then normalized to stay within the interval $[-\pi, \pi]$:

$$\theta^{(t+1)} = \text{wrapToPi}(\theta^{(t)} + \omega^{(t+1)}), \tag{131}$$

where $\text{wrapToPi}(\cdot)$ denotes the normalization function.

Subsequently, the agent's position $\mathbf{p}^{(t)} = (x^{(t)}, y^{(t)})$ is updates based on the new velocities and updated orientation $\theta^{(t+1)}$ by computing the displacement:

$$\Delta x = v^{(t+1)} \cos(\theta^{(t+1)}), \quad \Delta y = v^{(t+1)} \sin(\theta^{(t+1)}) \tag{132}$$

and setting

$$\mathbf{p}^{(t+1)} = \mathbf{p}^{(t)} + \Delta\mathbf{p}^{(t+1)}, \quad \text{where} \quad \Delta\mathbf{p}^{(t+1)} = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = v^{(t+1)} \begin{bmatrix} \cos(\theta^{(t+1)}) \\ \sin(\theta^{(t+1)}) \end{bmatrix}. \tag{133}$$

By learning velocity increments rather than absolute velocity commands, the agent can potentially achieve smoother and more stable control, as it naturally respects continuity in its motion commands.

Overall, this velocity displacement action space introduces realistic motion dynamics while maintaining computational tractability and compatibility with the continuous reinforcement learning framework.

7.5 Training and Evaluation

The training and evaluation procedures largely follow the framework established in the previous chapter on continuous reinforcement learning, hence for the full explanation details see Chapter 6. Here, we continue to use the Soft Actor-Critic algorithm as our primary learning method due to its robustness and effectiveness in continuous control tasks.

However, given the increased complexity of the environment, which now incorporates velocity-based control, obstacle awareness, and orientation information, training durations are extended and hyperparameters are adjusted to ensure sufficient policy convergence and stability. Despite these additions, the overall training structure, including environment setup and evaluation protocols, remains consistent with the earlier framework.

7.6 Reward Function

The core of the reward function remains basically unchanged from the continuous learning framework described in Section 6.16. However, to account for the obstacles and orientation awareness, two additional reward components are incorporated.

7.7 Obstacle Penalty

To encourage safe navigation and obstacle avoidance, the agent receives a significant negative penalty when it approaches too close to any obstacle. Formally, let $d_{\text{obs}}^{(t)}$ denote the minimum Euclidean distance between the agent's position $\mathbf{p}^{(t)}$ and the nearest obstacle cell. The obstacle penalty term is defined as:

$$r_{\text{obstacle}}^{(t)} = \begin{cases} -\lambda_{\text{obs}} & \text{if } d_{\text{obs}}^{(t)} \leq d_{\text{threshold}} \\ 0 & \text{otherwise} \end{cases} \tag{134}$$

where $\lambda_{\text{obs}} > 0$ is a tunable penalty coefficient and $d_{\text{threshold}}$ is a safety distance threshold. This term strongly discourages the agent form navigating into unsafe proximity with obstacles.

This obstacle penalty term embodies the sensor fusion of surface-level obstacle information, typically acquired from LiDARs and cameras. By integrating obstacle proximity into the reward signal, the agent learns to interpret and act upon fused sensory data that combines both subsurface water content and surface obstacle detections.

7.8 Orientation Reward

To encourage the agent to align its heading direction with the goal, thus facilitating smoother and more efficient movement that respects the agent's underlying dynamics, an orientation-based reward term is introduced. This reward quantifies the alignment between the agent's current heading vector and the vector pointing from the agent's position to the goal. Let

$$\mathbf{h}^{(t)} = \begin{bmatrix} \cos(\theta^{(t)}) \\ \sin(\theta^{(t)}) \end{bmatrix} \quad \text{and} \quad \mathbf{g}_{\text{dir}}^{(t)} = \frac{\mathbf{g} - \mathbf{p}^{(t)}}{\|\mathbf{g} - \mathbf{p}^{(t)}\|} \tag{135}$$

be the unit vectors representing the agent's heading and the direction to the goal, respectively. The orientation reward is computed using cosine similarity:

$$r_{\text{orientation}}^{(t)} = \lambda_{\text{orient}} \cdot \text{clip}\left(\mathbf{h}^{(t)} \cdot \mathbf{g}_{\text{dir}}^{(t)}, -1, 1\right) \tag{136}$$

where $\mathbf{h}^{(t)} \cdot \mathbf{g}_{\text{dir}}^{(t)}$ denotes the inner product between the two vectors and measures how closely the agent's heading aligns with the direction toward the goal. The clipping operation ensures the value remains bounded in $[-1, 1]$, effectively implementing the cosine similarity function and preventing numerical instabilities. The scaling factor $\lambda_{\text{orient}} > 0$ controls the relative weight of this orientation reward within the overall reward function.

This reward term promotes smooth and efficient movement toward the goal, while respecting the agent's dynamic and kinematic constraints.

7.9 Combined Reward

The total reward at each time step is then given by:

$$r^{(t)} = r_{\text{base}}^{(t)} + r_{\text{obstacle}}^{(t)} + r_{\text{orientation}}^{(t)} \tag{137}$$

where $r_{\text{base}}^{(t)}$ represents the final comprehensive reward function from the continuous learning framework. Hence:

$$r_{\text{base}} = r_{\text{water\_final}} + r_{\text{progress\_diff}} + r_{\text{goal\_exp}} + r_{\text{explore}} + r_{\text{step}} + r_{\text{goal}} + r_{\text{boundary}} + r_{\text{timeout}} \tag{138}$$

By augmenting the original reward function from the continuous learning framework with only two additional components, it becomes possible to effectively generalize and address more complex tasks involving obstacles avoidance and sensor fusion. This minimal yet important modification demonstrates the power and flexibility of reinforcement learning methods. This adaptability highlights reinforcement learning's suitability for real-world autonomous navigation challenges characterized by environmental complexity and multi-modal perception.

To effectively balance the influence of the new obstacle and orientation reward components with the existing terms, we tuned each individual hyperparameter to ensure that the penalties and incentives introduced by these additions integrate smoothly with the prior reward structure. This tuning promotes stable learning and efficient policy convergence.

All hyperparameters used in these new experiments are summarized in Table 4.

Table 4: Reward components and their corresponding hyperparameters and values for the enhanced continuous framework

Reward Component	Hyperparameters to Tune	Values Used
r_t^{obstacle}	λ_{obs}, $d_{\text{threshold}}$	$\lambda_{\text{obs}} = +40.0$, $d_{\text{threshold}} = 1.0$
$r_t^{\text{orientation}}$	λ_{orient}	$\lambda_{\text{orient}} = +2.0$
$r_t^{\text{water\_final}}$	λ_1, λ_2, λ_3, λ_4, λ_5 T_{dry}, T_{avg}	$\lambda_1 = +1.2$, $\lambda_2 = +0.2$, $\lambda_3 = +0.5$, $\lambda_4 = +1.0$, $\lambda_5 = +1.0$, $T_{\text{dry}} = 0.1$, $T_{\text{avg}} = 0.22$
$r_t^{\text{progress\_diff}}$	λ_6, λ_7, δ	$\lambda_6 = +2.5$, $\lambda_7 = +4.5$, $\delta = +0.2$
$r_t^{\text{goal\_exp}}$	λ_8, λ_9, λ_{10}	$\lambda_8 = +2.0$, $\lambda_9 = +10.0$, $\lambda_{10} = +1.0$
r_t^{goal}	λ_{11}, ϵ	$\lambda_{11} = +11.0$, $\epsilon = 0.8$
r_t^{step}	λ_{12}	$\lambda_{12} = +0.2$
$r_t^{\text{boundaries}}$	λ_{13}	$\lambda_{13} = +1.0$
r_t^{time}	λ_{14}, $T_{\max}$	$\lambda_{14} = +1.5$, $T_{\max} = 150$
r_t^{explore}	λ_{15}, λ_{16}, τ	$\lambda_{15} = +1.0$, $\lambda_{16} = +0.5$, $\tau = 0.3$

7.10 Results

Figure 42 presents the performance metrics of the agent trained using the final, comprehensive reward function within this advanced continuous setting. The plot tracks the best-performing checkpoints saved during the training, recorded each time a new best mean reward was achieved. It is important to emphasize that this reward function builds upon the core principles of the previous reward formulation presented in the continuous framework, (see Equation 93), and therefore shares its inherent strengths and limitations. In particular, the observed variability in the reward function persists, although it is partially reduced due to the effects of applying a velocity-based control instead of the previous position-based one. This change provides the agent with greater freedom of action, helping to counterbalance fluctuations in the reward signal.

As shown in the plot, the agent successfully learns to reach the goals even within the new framework settings. The introduction of velocity-based control provides greater flexibility, allowing the agent to avoid obstacles more efficiently and leverage momentum to navigate particularly wet areas more quickly. This results in an asymptotic average cumulative water level of approximately 0.10, which is a remarkable result. Additional experiments, such as varying the mask size or changing the dimensions of the field, demonstrate that the agent consistently adapts and continues to achieve its objectives across different configurations.

These results underscore the importance of the new setting, where the agent learns to avoid obstacles while controlling its velocities. Unlike position-based control, the same position can now be reached with different orientations and speeds. This allows the agent to leverage its previous speeds and orientations to avoid or overcome particularly saturated areas and challenging spots.

Moreover, further tests demonstrated that these trained agents can be effectively scaled and deployed in larger and more complex scenarios without requiring significant retuning of the hyperparameters. This robustness indicates that the learned policies generalize well beyond

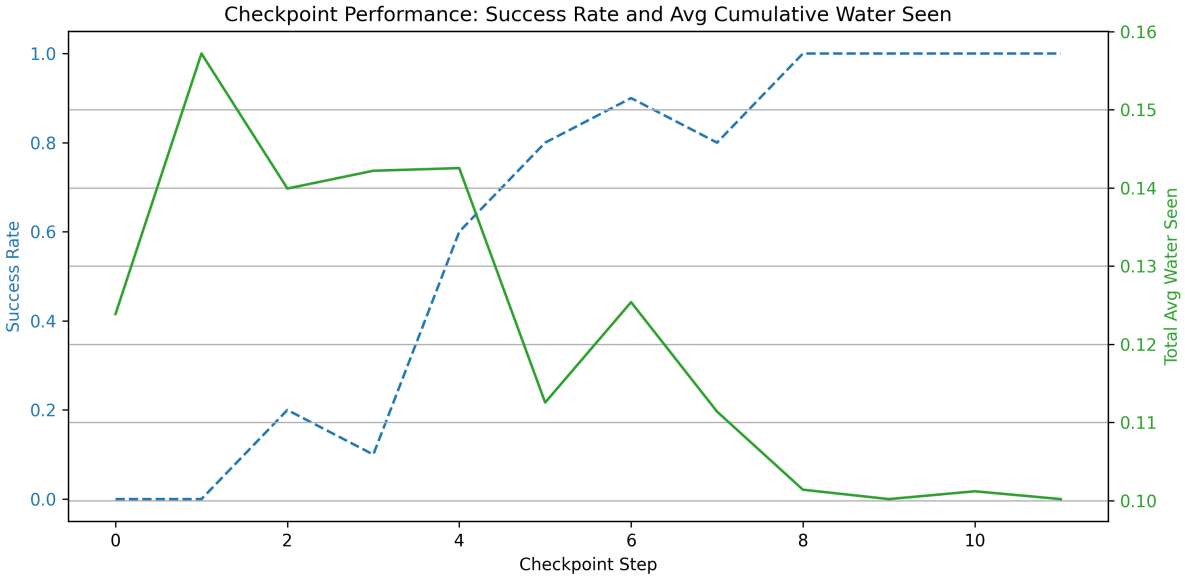


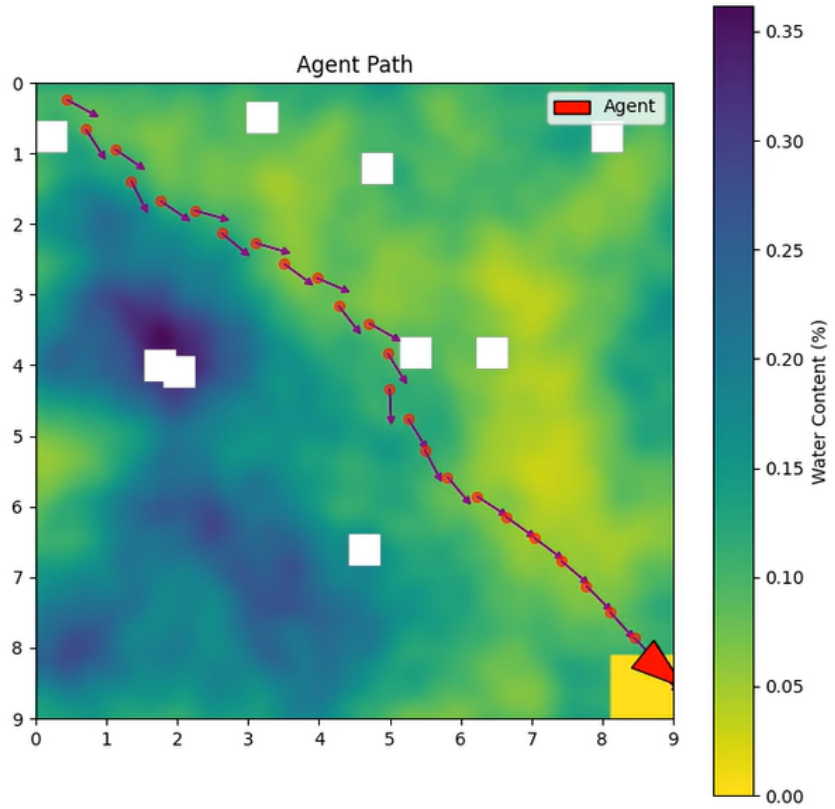
Figure 42: Performance evaluation of the agent with the new settings over training progress. The success rate is shown as a dashed blue line, while the solid green line represents the average cumulative water content encountered during episodes.

the initial training environments. Additionally, agents trained in general scenarios can be specialized for different environmental conditions by simply adjusting few parameters, mainly those related to the water reward components. This flexibility highlights the versatility of the agents and the potential power of this reinforcement learning approach.

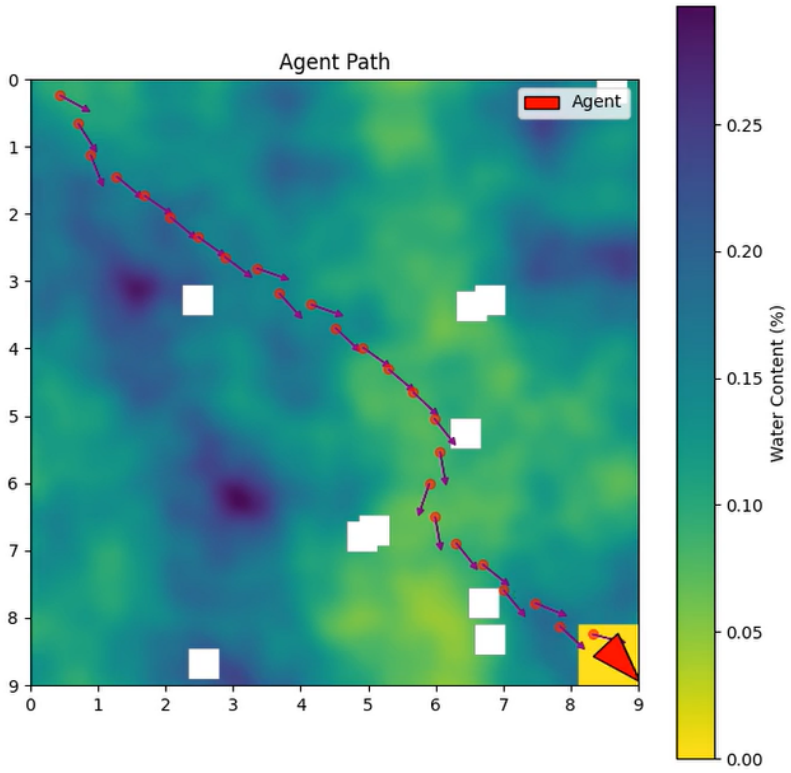
To conclude, Figures 43a–43b provide qualitative insight into the agent's behaviour under the final reward formulation and velocity-based control strategy. These examples illustrate successful navigation across different field configurations, where the agent consistently starts from the top-right corner and moves toward the designated goal area, indicated by a yellow square located in the bottom-left region.

These trajectories showcase the agent's ability to follow optimal paths, successfully avoiding obstacles and generalizing its learned policy across different scenarios. Each red dot in the visualization corresponds to a single step take by the agent, while purple arrows indicate the agent's orientation at each step, capturing the decision-making dynamics in terms of direction and heading.

These visual outcomes emphasize the effectiveness of the final reward design. The agent consistently avoids high water content areas and navigates around possible obstacles, while maintaining a reliable trajectory toward the goal.



(a)



(b)

Figure 43: Examples of final trajectories of the agent

8 Planning

This chapter presents the overall planning and timeline of the project, which beagn informally in early February 2025, with an initial literature overview, and officially began on February 20th, 2025, with the first in-person meeting held at the CDEI office.

During the initial weeks, our focus was on reviewing scientific literature and related work to build a strong foundational understanding of ground-penetrating radar (GPR) data analysis and reinforcement learning (RL) agricultural applications. Once this phase was completed, we proceeded with the collection of GPR data, followed by preliminary analysis to explore the characteristics and structure of the radargrams.

The next stage involved visualizing the radargrams to gain intuitive insights. Concurrently, we conducted an in-depth study of the Adriano Liso network, which was used to extract meaningful latent representations from the GPR data. These latent features formed the basis for downstream RL tasks.

With the latent space representations obtained, we shifted our focus to studying the existing systems and frameworks previously developed at CDEI. From this, we began the development of our own reinforcement learning pipelines.

The first implementation was a Discrete Reinforcement Learning Framework, which was then tested and refined. After evaluating its performance we progressed to the Continuous RL Framework, and finally to the Advanced RL Framework, which incorporated more complex dynamics and reward shaping strategies.

In parallel, the writing of this thesis began in early May, along side the experimental and development phases to ensure proper documentation and reflections on the project's evolution.

Figure 44 illustrates the complete pipeline described above.

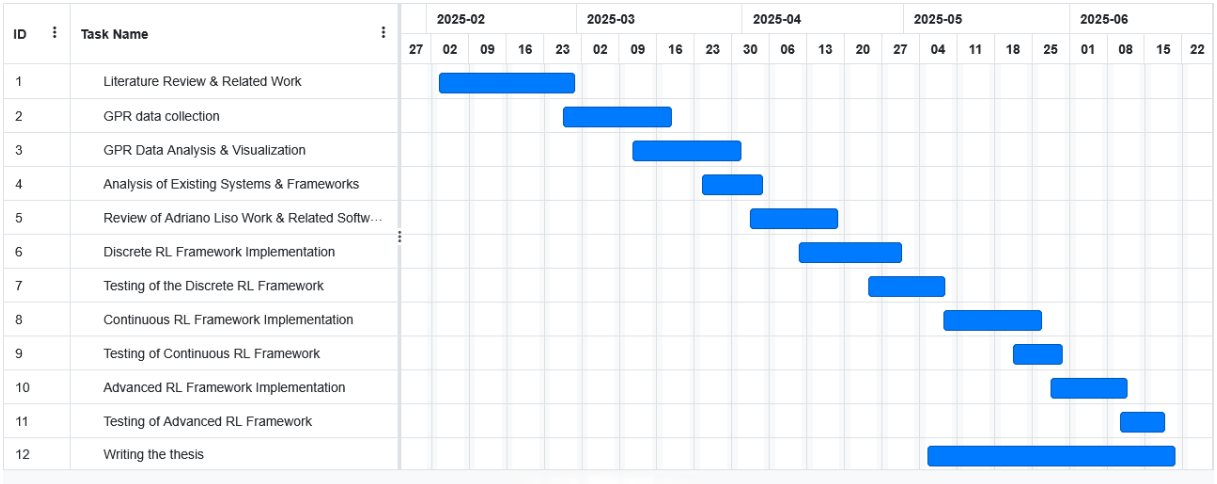


Figure 44: Gantt Diagram presenting the overall pipeline of this project.

9 Economic Assessment

In this chapter, we provide an evaluation of the economic aspects related to the project. Since the development primarily relied on robotic systems and tools already available at the CDEI office and laboratory, no additional hardware investments were required during the course of the work.

As a result, the overall project cost is primarily assessed based on human resource expenses, such as staff time and effort, and operational costs. These constitute the major contributors to the economic footprint of the project.

However, it is important to highlight that the GPR visualization software used, GPR-SLICE, is a commercial software and not freely available. Hence, its licensing cost must be considered in future replications of this work, particularly if one opts to use this software over alternative open-source visualization tools, which may be less powerful or feature-limited.

Furthermore, the choice of GPR sensor significantly impacts the total cost of the project. Depending on the required resolution, depth, and environmental conditions, the sensor selection can influence both the budget and the quality of the results. Therefore, a careful cost-benefit analysis should be conducted to determine the most suitable GPR device for similar future implementations.

Therefore, we now present the human resource costs associated with the major phases of the project. These estimates are based on the effort required for each task, including planning, development, analysis, and evaluation stages. To quantify these costs, we adopted a general assumption of a student researcher's average compensation of 15 euro per hour.

Table 5 summarizes the estimated cost in euros for each task.

Table 5: Estimated Human Resource Costs

Task	**Estimated Cost**
Literature Review	1,000
GPR Data Collection	600
GPR Analysis	400
Dataset Creation	400
Adriano Liso Network Study	200
Existing Framework Review	500
Develop Discrete RL Framework	1,200
Testing (Discrete RL)	500
Develop Continuous RL Framework	1,400
Testing (Continuous RL)	500
Develop Advanced RL Framework	1,600
Testing (Advanced RL)	600
Thesis Writing	1,500
Total Estimated Human Costs	**10,400**

It is important to note that the estimated costs for the GPR data collection and analysis tasks should also account for additional fixed expenses, including travel costs incurred to access ex-

ternal laboratory facilities. These costs may vary depending on type of transportation used.

10 Environmental Assessment

This chapter evaluates the environmental aspects of the project, focusing on both direct and indirect impacts associated with the development and implementation of GPR-based deep reinforcement learning frameworks. The assessment covers the entire project lifecycle, from data acquisition to computational processing and documentation.

The environmental footprint is relatively low, as it mainly relies on computational work and existing ground penetrating radar equipment available at the CDEI office and laboratories. Nonetheless, a quantitative approximation of emissions has been made based on standard energy usage and travel data.

According to 2024 data from the Comisión Nacional de los Mercados y la Competencia (CNMC), electricity consumption in Spain results in approximately 260 grams of CO_2 equivalent (CO_2eq) emissions per kilowatt-hour (kwh). Based on this factor, we estimate the carbon footprint of both computational operations and travel involved in the project.

The project relied on:

- A laptop workstation with a power draw of approximately 65 W, used for GPR data preprocessing, algorithms developing, and documentation.
- Occasional use of a GPU workstation (estimated at 250 W) during the training of reinforcement learning agents.

Assuming an average usage of:

- 5 hours/day on the laptop for 120 days (roughly 4 months):
  $65\,\mathrm{W} \times 5\,\mathrm{h} \times 120 = 39.0\,\mathrm{kWh}$
  $39.0\,\mathrm{kWh} \times 260\,\mathrm{g} = 10.14\,\mathrm{kg}\,CO_2\mathrm{eq}$
- 3 hours/day on the GPU system for 60 days:
  $250\,\mathrm{W} \times 3\,\mathrm{h} \times 60 = 45.0\,\mathrm{kWh}$
  $45.0\,\mathrm{kWh} \times 260\,\mathrm{g} = 11.7\,\mathrm{kg}\,CO_2\mathrm{eq}$

Total estimated CO_2eq form energy usage: $\sim$21.84 kg

Regarding GPR sensor usage, the system was powered using the existing laboratory power supply. Assuming an average power consumption of 20 W for 2 hours per day over a period of 10 days, the total energy consumption is:

- $20\,\mathrm{W} \times 2\,\mathrm{h} \times 10 = 0.4\,\mathrm{kWh}$
  $0.4\,\mathrm{kWh} \times 260\,\mathrm{g} = 104\,\mathrm{g}\,CO_2\mathrm{eq}$

This results in a total of approximately 104 grams of CO_2eq, which represents a negligible share of the project's overall emissions when compared to the energy demands of computational tasks.

In conclusion, it is also important to consider that the GPR data collection phase required travel to external laboratories and fields. These locations are accessible by multiple means of transportation, including bus, car, or train. Hence, due to the variability in transportation modes, it is difficult to provide a precise estimate of the associated energy consumption of CO_2 emissions.

Future replications of the project could reduce this uncertainty by prioritizing low-emission transport options when feasible.

11 Social and Gender Equality Assessment

This project does not have a direct impact on social issues or gender equality. The work was carried out in collaboration with the Centre De Disseny d'Equips Industrials (CDEI), an organization that promotes inclusive participation by employing and supporting both men and women in its technical and research teams.

While the project's primary objective is not explicitly social, it aligns with the principles of the United Nations Sustainable Development Goals (SDGs). Specifically, by contributing to advancements in environmental sensing, artificial intelligence, and sustainable land monitoring, it supports:

- SDG 5: Gender Equality, by encouraging equal participation in research and engineering activities;
- SDG 9: Industry, Innovation and Infrastructure, through the development of cutting-edge reinforcement learning techniques for geophysical data analysis;
- SDG 13: Climate Action, by enabling more efficient and informed decision-making in the context of environmental and agricultural monitoring.

Moreover, the project promotes academic collaboration, technical skills development, and open access to knowledge for individuals from diverse backgrounds, thus promoting a more inclusive and equitable research environment.

12 Conclusions and Future Work

In this thesis, we examined the application of Deep Reinforcement Learning techniques within the domain of agricultural robotics, aiming to address the unique challenges of autonomous navigation and path planning in unstructured and heterogeneous environments. Specifically, our primary objective was to design, develop, and evaluate autonomous agents capable of maneuvering through heterogeneous terrains while leveraging both surface-level and subsurface information.

While surface data, obtained through conventional sensors such as LiDARs, cameras, or depth sensors, provide essential geometric and visual context for tasks such as obstacle avoidance, terrain mapping, and path planning, they offer only a limited understanding of the environment, capturing what is accessible above the ground. Hence, subsurface data introduce a complementary layer of information, revealing critical characteristics of the underlying soil and terrain structure that are not visible with traditional surface sensors. This enhanced semantic understanding of the environment is particularly important in agricultural contexts, where the physical condition of the subsurface plays a significant role. This information can be used for enabling precision irrigation, crop health diagnostics, soil composition analysis, and particular detection tasks that are difficult to achieve relying only on surface data. By integrating both modalities, the agent gains a more comprehensive perception of its environment, enabling more informed and adaptive decision-making.

To support this, we developed novel reinforcement learning pipelines tailored to environments where both types of data are relevant. A central contribution of this work lies in the development and integration of Deep Reinforcement Learning frameworks designed for both discrete and continuous state-action spaces. These frameworks include the complete training pipeline, from environment construction and reward shaping to model training, evaluation, and logging.

A key focus of this project was the modeling of subsurface characteristics, with special attention to capture spatial soil moisture variability. This process began with the acquisition of large volumes of raw subsurface data using Ground Penetrating Radar, in the form of A-scan and B-scan profiles. The data was then processed using a custom convolutional autoencoder, which learned to compress the high-dimensional input scans into compact latent vectors that preserve essential spatial and physical features. From these encoded representations, we inferred the local soil moisture levels that populate the latent representations of the environments used for training the agents.

With the latent space established, we first trained agents in a discrete reinforcement learning framework. We mainly studied the Proximal Policy Optimization algorithm, initially using a baseline reward function based on minimal requirements. Through iterative refinement and the progressive introduction of different reward components, we reached a comprehensive reward function that successfully guided agents toward robust and efficient navigation behaviour. Extensive hyperparameter tuning was conducted to ensure stable training dynamics and maximize policy performance. The trained agents demonstrated the ability to successfully navigate a variety of field configurations, consistently identifying and following trajectories that minimized exposure to high soil moisture content.

Following this, we transitioned to the continuous control setting, where the discrete grid representation was replaced by a continuous latent field. In this context, agents were trained using

mainly the Soft Actor-Critic algorithm. The transition to continuous action spaces introduced new challenges, including precise movement control, sensitivity to local reward gradients, and the need for better exploration strategies. To address these challenges, the reward function was carefully redesigned, tuned, and extended with additional components. Evaluation metrics and trajectory visualizations confirmed that the agents successfully learned to navigate different field configurations, consistently selecting paths that minimized soil moisture content while adapting to variations in environmental conditions.

In the final and most advanced stage, we integrated both surface and subsurface modalities within the continuous framework. This involved training agents to simultaneously follow subsurface features while avoiding surface-level obstacles. This sensor fusion was achieved by combining the latent moisture maps with obstacle masks derived from simulated surface sensors, resulting in agent that can balance between hydrologically safe navigation and collision-free path planning.

Moreover, we transitioned from position-based control to a more realistic velocity-based control scheme, better aligned with the constraints and actuation models of real-world robotic platforms. This added complexity required further refinement of the reward structure and action representation. However, it also gave the agent more freedom of movement, enabling smoother trajectories and more adaptive navigation strategies.

Agents trained under this framework demonstrated robust performance in a variety of complex environments and represent a significant step toward real-world deployment in agricultural robotics.

Across all scenarios, we observed that the design of the agent's observation space, specifically the inclusion of a representative local mask, played a critical role in learning stability and final policy performance. Furthermore, reward design proved to be the key factor in shaping the agent's learning dynamics and overall performance. By considering different level of water-awareness in the local mask , we were able to better stabilize the training process, enabling the learning algorithms to converge toward more optimal behaviours.

Quantitative results show that agents trained with the final reward configurations in both discrete and continuous settings reached the goal in all evaluation episodes, while significantly reducing their exposure to high-moisture regions. In addition, trajectories tracking and visitation frequency heatmaps reveal that trained agents developed consistent and optimal navigation strategies even in previously unseen terrain configurations, highlighting the generalization capability of the learned policies.

These findings emphasize the effectiveness of multi-modal reinforcement learning pipelines in domains requiring complex environmental understanding, such as precision agriculture, and lay the groundwork for future real-world deployments of subsurface-aware robotic agents.

As future work, we propose extending this frameworks to multi-agent scenarios, where several agents operate simultaneously within the same environments. This will enable the study of distributed exploration strategies and the emergence of cooperative navigation behaviours, both very important for scaling up autonomous operations in large agricultural fields.

Another major direction involves the transfer of the trained policies from simulation to physical robotic platforms. This includes validating the agent's behaviour in real-world environments

equipped with actual surface and subsurface sensors, hence testing the robustness of the learned policies under real-world conditions, including sensor noise, environmental variability, and hardware constraints. This process will also provide opportunities for fine-tuning and adaptive calibration, helping to bridge the possible gap between simulated training and practical deployment.

Another possible direction involves the exploration of alternative subsurface sensing technologies to capture underground characteristics. Comparative studies could be conducted to evaluate the performance of different geophysical sensors against the Ground Penetrating Radar data used in this study.

Finally, while this thesis primarily focused on soil moisture as the key subsurface variable, other physical characteristics could be considered, such as soil density, composition, or temperature gradients. This would enable agents to perform a broader range of tasks, including pipeline or root system following, underground anomaly detection, or other particular trackings.

13 Acknowledgements

As I reach the conclusion of this project, I would like to take a moment to express my heartfelt gratitude.

First and foremost, I am sincerely thankful to my supervisor, Prof. Alba Pérez Gracia, my co-supervisor, Prof. David Caballero Flores, and my colleagues at CDEI for their support and guidance throughout this journey. I am also grateful to Prof. Gian Antonio Susto for his valuable insights and assistance during this work.

My deepest appreciation goes to my family for their constant support and understanding over the years. Their strength, encouragement, and belief in me have been the foundation that allowed me to pursue and persevere through this academic path.

I would also like to thank all the wonderful friends I met during my Erasmus experience. Their presence turned this chapter of my life into an unforgettable and enriching adventure.

Lastly, I am profoundly grateful to my lifelong friends, milestones in my life, whose constant support and friendship have always been a source of inspiration and comfort.

Barcelona, June 2025 Riccardo Rettore

Bibliografia

[1] H. M. Jol, ed., *Ground Penetrating Radar Theory and Applications*. Amsterdam: Elsevier, 2009.

[2] P. Kaniewski and T. Kraszewski, "Estimation of handheld ground-penetrating radar antenna position with pendulum-model-based extended kalman filter," *Remote Sensing*, vol. 15, p. 741, 01 2023.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] M. N. Alpdemir and M. Sezgin, "A reinforcement learning (rl)-based hybrid method for ground penetrating radar (gpr)-driven buried object detection," *Neural Computing and Applications*, vol. 36, no. 14, pp. 8199–8219, 2024.

[5] F. M. Riese and S. Keller, "Fusion of hyperspectral and ground penetrating radar to estimate soil moisture," *arXiv preprint arXiv:1804.05273*, 2018.

[6] X. Li, X. Cheng, Y. Zhao, B. Xiang, and T. Zhang, "Deep learning-based ground-penetrating radar inversion for tree roots in heterogeneous soil," *Sensors*, vol. 25, no. 3, p. 947, 2025.

[7] K. Weerakoon, A. J. Sathyamoorthy, J. Liang, T. Guan, U. Patel, and D. Manocha, "Graspe: Graph based multimodal fusion for robot navigation in unstructured outdoor environments," *arXiv preprint arXiv:2209.05722*, 2022.

[8] M. V. Gasparino, V. A. H. Higuti, A. E. B. Velasquez, A. N. Sivakumar, M. Becker, and G. Chowdhary, "Cropnav: a framework for autonomous navigation in real farms," *arXiv preprint arXiv:2411.10974*, 2024.

[9] A. E. B. Velasquez, V. A. H. Higuti, M. V. Gasparino, A. N. Sivakumar, M. Becker, and G. Chowdhary, "Multi-sensor fusion based robust row following for compact agricultural robots," *arXiv preprint arXiv:2106.15029*, 2021.

[10] B. Pongrac, D. Gleich, M. Malajner, and A. Sarjaš, "Cross-hole gpr for soil moisture estimation using deep learning," *Remote Sensing*, vol. 15, no. 9, p. 2397, 2023.

[11] K. Weerakoon, A. J. Sathyamoorthy, U. Patel, and D. Manocha, "Terp: Reliable planning in uneven outdoor environments using deep reinforcement learning," *arXiv preprint arXiv:2109.05120*, 2021.

[12] J. P. Fentanes, T. Duckett, and G. Cielniak, "Kriging-based robotic exploration for soil moisture mapping using a cosmic-ray sensor," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1104–1119, 2020.

[13] D. J. Daniels, *Ground Penetrating Radar*. IET, 2004.

[14] A. Neal, "Ground-penetrating radar and its use in sedimentology: principles, problems and progress," *Earth-Science Reviews*, vol. 66, no. 3–4, pp. 261–330, 2004.

[15] S. E. Technologies, "Gpr-slice software joins forces with screening ea-

gle technologies." https://www.screeningeagle.com/en/about-us/news/gpr-slice-now-part-of-screening-eagle, 2021. Accessed: 2025-05-13.

[16] B. Geophysical, "Gpr-slice v.7." https://bigmangeo.com/products/gpr-slice-v7-mt-ground-penetrating-imaging-software, 2025. Accessed: 2025-05-13.

[17] H. Geophysics, "Gpr-slice software." https://www.huntergeophysics.com/product/gpr-slice/, 2025. Accessed: 2025-05-13.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2 ed., 2020.

[19] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 12, 1999.

[20] N. Rudin, J. Merel, N. Heess, H. Bhatia, A. Vahdat, and A. A. Rusu, "Learning humanoid motor skills with deep reinforcement learning," *arXiv preprint arXiv:2208.02859*, 2022.

[21] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[22] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] H. v. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, pp. 2613–2621, Curran Associates, Inc., 2010.

[25] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 1606–1614, Curran Associates, Inc., 2016.

[26] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 799–806, 2010.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1889–1897, PMLR, 2015.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[29] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic

policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (*ICML*), PMLR, 2014.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[31] G. Geo, "Mala ground penetrating radar systems." https://www.guidelinegeo.com/products/ground-penetrating-radar/, 2024.

[32] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International conference on artificial neural networks*, pp. 52–59, Springer, 2011.

[33] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," in *Science*, vol. 313, pp. 504–507, American Association for the Advancement of Science, 2006.

[34] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1798–1828, 2013.

[35] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for neural networks in image processing," *IEEE Transactions on Image Processing*, vol. 26, pp. 4493–4507, 2017.

[36] A. Liso, "Title tbd." Work in preparation, 2025.

[37] K. Perlin, "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques* (*SIGGRAPH '85*), vol. 19, (New York, NY, USA), pp. 287–296, ACM, 1985.