POLITECNICO DI TORINO

Master's Degree in Computer Engineering

M.Sc. Thesis

Towards Autonomous Robotic Spray Painting with Unsupervised Reinforcement Learning

Supervisors
Prof. Tatiana Tommasi
Prof. Raffaello Camoriano
M.Sc. Gabriele Tiboni

Candidate
Marco Pratticò

Academic Year 2022-2023

Abstract

Long-standing problems in robotics such as cleaning and spray painting require the generation of task-specific trajectories satisfying physical constraints. Accelerating the generation process by autonomously deriving robotic paths would reduce the manual effort currently required in such settings. Furthermore, path generation needs to adapt to the specific geometries of the target objects. These stringent requirements are usually met by designing ad-hoc heuristics for each specific object category.

Reinforcement Learning (RL) has been successfully employed to tackle autonomous robotic tasks in the literature, from robotic manipulation to locomotion. However, RL algorithms can suffer from low generalization capabilities and low sample efficiency, i.e., a large number of agent-environment interactions are often needed for the algorithm to converge to successful, yet specific, policies. Unsupervised Reinforcement Learning (URL) has been proposed to speed up policy training by introducing a task-agnostic policy pre-training phase. In particular, pre-training does not involve task-specific reward signals. Instead, it exploits intrinsic motivation to encourage exploration of the underlying environment. This approach enables collecting transferable knowledge from the environment to be later used for fine-tuning the policy on a range of more specific downstream tasks. Recent works successfully employ URL to drive exploration, e.g., by maximizing the entropy of the state visitation distribution. Nevertheless, such algorithms have only been investigated in locomotion tasks or gridworld environments, despite their potential to be applied to coverage path planning (CPP) problems, which may benefit from state-entropy maximization.

This thesis investigates the utilization of (U)RL for path-planning problems. In particular, we focus on robotic spray painting, a fundamental industrial manufacturing task and strongly related to the class of CPP problems. In this context, we adopt the URL framework to speed up the training process of an object-specific policy by pre-training a single policy with a state-entropy maximization objective. Our experimental results characterize the impact of intrinsic motivation on the training process, comparing the final learning outcomes when training from scratch with those obtained by starting from a pre-trained URL policy.

Acknowledgements

First of all, I would like to thank my thesis supervisors, Raffaello, Gabriele, and Professor Tommasi, for their valuable support and experience during the development of this work, with the hope of collaborating on new research projects in the future. I am grateful to the Vandal group with whom I spent the last months in the lab and for the great aperitifs together. This was my first time spending an office-like experience and I really hope to find a similar atmosphere in the future. I think I have grown a lot professionally over the past period, and thanks to my lab colleagues, I have learned valuable insights into what I hope will be my future job, as an AI researcher. I wish you all good luck with your research projects.

Then, I would like to mention my family for always supporting my studies, my interests and my passions. This great step in my life I owe mostly to them. Another mention of honour is to the family I chose, my friends. I want to thank all my friends from Rome, with whom I spent the last years on incredible adventures and in simple pub nights, which, however, should not be taken for granted. I am profoundly grateful for your friendship. Although I am not a person who expresses his feelings explicitly, however, I owe you all a thank you. Special mentions go to my friend Niccolò T. and my former flatmate Alessandro C., with whom I shared a substantial part of my university journey.

Last but not least, I want to thank my "Turin Party Group" friends for the memorable nights in Turin, hoping for more celebrations ahead. I am also grateful to all the colleagues I've met throughout my university years who turned great friends, your friendships have enriched my academic journey in profound ways.

Contents

List of Tables

List of Figures

List of Algorithms

If a machine can think, it might think more intelligently than we do, and then where should we be?

[A. Turing, Computing Machinery and Intelligence, 1950]

Chapter 1

Introduction

Many real-world problems require making a sequence of choices in order to achieve an objective. Reinforcement Learning (RL) [1] can be employed to formalize and tackle these sequential decision-making problems, which are commonly modelled as a *Markov Decision Process (MDP)*. Within this framework, a decision-maker, often referred to as an *agent*, operates within an *environment*, making decisions and taking *actions* to interact with it. The agent's behaviour, or its strategy for selecting actions, is commonly called *policy*. We refer to the current condition of the environment as the *state* of the environment, which fully describes the relevant properties of the environment at any given time step. On the other hand, the *observation* is a potentially partial description of the state to which the agent has access and it may omit some information. This interaction process is typically divided into discrete time intervals. At training time, the agent executes an action according to its exploration policy and its observations, receives a *reward*, and then observes the environment after the action is taken. The standard MDP formulation assumes that the next environmental state depends only on the current state and the current action, known as the Markov Property.

The primary objective of an RL agent is to maximize the cumulative reward over a series of successive interactions. This entails learning how to optimize rewards not just in the immediate future but also over the long term.

RL has recently achieved significant success as this paradigm can be used to solve different types of tasks, such as robotic manipulation [2], playing Atari games [3], GO [4].

The Reward The goal of a learning process is the maximization of a cumulative reward signal. This signal is designed to carry information on how well is the agent performing a task, in order to guide the trial-and-error process. At first, the agent acts according to a random policy. The information gathered during these interactions is then used to adapt and refine the agent's behaviour. The reward, serving as the driving signal behind this learning process, is generally shaped by a human designer. This design task can be complex and costly, as various tasks may necessitate reward signals that are diverse in nature. Any suboptimal reward design choices can result in unintended behaviours.

To address this complexity, a common approach is to use sparse reward signals, meaning that the agent receives a reward only when it successfully completes the required task, with no incentives offered for intermediate steps leading to that action. The process of effectively and efficiently exploring the environment becomes crucial; if the agent cannot acquire any rewards, it becomes incapable of learning. Even in cases where the agent can collect rewards, if they are sporadic, the learning process can become prohibitively time-consuming, and the agent may fail to attain the desired behaviour. Consequently, the effectiveness of the learning process is profoundly influenced by the structure of the reward and the environment's characteristics.

Unsupervised Reinforcement Learning Despite recent achievements, RL faces several drawbacks, including issues related to sample efficiency [5] and generalization [6]. A key challenge in RL is the reliance on a well-shaped reward function [7] to describe the task performance. However, in practice, since the reward functions are often manually shaped, a careful design to describe the desired learning process toward desirable outcomes is necessary. This requirement presents a significant obstacle to autonomous learning since each new task demands a costly and task-specific reward formulation. Additionally, the transferability of knowledge from solving one RL problem to another is limited, reducing the potential for generalization.

The Unsupervised Reinforcement Learning (URL) approach [8] seeks to address this generalization problem. In the URL framework, the learning agent interacts with the environment in two distinct phases. First, during the unsupervised pre-training phase, no task-specific reward is employed, and the agent's goal is to pre-train a policy to capture knowledge for future use. This pre-training can take various forms, including learning transition dynamics, devising exploratory policies, generating relevant interactions with the environment, encoding abstract state representations, or constructing a dataset of informative interactions.

Then, during fine-tuning, the agent strives to efficiently learn a policy that maximizes a task-specific reward, starting from the pre-trained model and not from a random policy. Using a pre-trained policy as the initial one to learn a subsequent new task enhances the efficiency of the learning process, even in settings with sparse reward signals.

This thesis aims to establish an efficient pipeline, employing the Unsupervised RL approach, for enabling an agent to acquire the skill of painting objects.

Robot Spray Painting The primary objective of this thesis is to address the challenge of robotic spray painting from the perspective of coverage path planning. The task deals not only with the precise application of paint, covering entirely the surfaces but also with smooth and uniform paint thickness over the target surface. To address this problem, we take advantage of RL and unsupervised RL with the goal of improving autonomous path generation for robotic spray painting.

Coverage path planning (CPP) is a key robotic planning problem, including spray painting tasks. It is aimed at finding a robot path ensuring complete and uniform coverage of the target surface and avoiding obstacles [9]. Some recent works make progress in solving CPP problems with the use of RL [10], [11]. This thesis investigates the complexities of formulating the robotic spray painting task as a coverage path planning problem and of dealing with different object shapes, emphasizing the need for a two-step strategy — pre-training and fine-tuning — for an efficient robot learning process.

State-Entropy Maximization In the unsupervised pre-training process driven by state entropy maximization, the objective is to identify a policy that maximizes the entropy of the induced state distribution [12]. It is important to note that this approach significantly differs from a policy that assigns equal probability to all actions in a given state. Such a policy overlooks the sequential nature of problems, where a specific sequence of actions is often required to reach certain states with high probability, rendering state entropy maximization a non-trivial task. Despite its non-trivial nature, obtaining this policy does not necessitate any feedback embedded in the environment, rendering its learning process entirely unsupervised.

A policy pre-trained to maximize the entropy of the induced state distribution can offer valuable advantages during the subsequent fine-tuning phase. Indeed, the pre-training allows the agent to interact with the environment in diverse and informative ways collecting knowledge to use in the latter fine-tuning phase. Then, thanks to pre-training, this latter phase will require less data thus tackling one of the key issues of RL, which is sample efficiency [7]. From a theoretical point of view, many works demonstrated the significance of state coverage in data collection for offline [13]–[17] and online RL methods [18], [19].

In this thesis, we show that pre-training, guided by a state-entropy maximization objective, enhances performance and sample efficiency when fine-tuning to refine the policy behaviour according to the characteristics of a specific CPP problem. Practically speaking, our intuition is based on the similarity between the state-entropy maximization objective and the final coverage goal belonging to problems in the context of Coverage Path Planning.

Contribution The work of this thesis aims to take a step further in the automation of robotic spray painting processes within industrial settings. While a handful of works [20]–[22] address this task in the literature, it remains far from being considered a solved problem. Challenges such as object-conditioned learning and sample inefficiency significantly limit the effectiveness of learning methods for spray painting and other CPP problems.

In our approach, we employ the Reinforcement Learning framework to automate trajectory generation for covering target surfaces. Additionally, we incorporate the Unsupervised Reinforcement Learning framework to tackle the challenges posed by object-conditioned learning. Our method involves pre-training policies across various target shapes using Unsupervised Reinforcement Learning, followed by fine-tuning to generate trajectories tailored to specific shapes. The two-step training process, as demonstrated in our work, proves to be advantageous in terms of both performance and training time. Leveraging pre-trained policies allows the RL agent to learn to generate effective trajectories with fewer environment interactions.

In summary, this work contributes by formalizing the setup for employing Reinforcement Learning algorithms in continuous spaces for robotic spray painting tasks and proposing a pre-training method that reduces the fine-tuning time for object-conditioned training.

Overview This thesis is structured as follows:

- **Chapter 2**: covers essential mathematical foundations, technical terminology, and background concepts necessary for understanding the contents of the thesis. It introduces the basic background for sequential decision-making and Reinforcement Learning, delves into the mathematical definition of entropy, and provides an overview of spray painting and Coverage Path Planning problems.

- **Chapter 3** is dedicated to Unsupervised Reinforcement Learning, presenting the general problem formulation and an overview of the different literature approaches.

- **Chapter 4** presents the state-entropy maximization objective, illustrating the problem formulation and several algorithms guided by this objective.

- **Chapter 5** illustrates the approaches found in the literature for addressing robotic spray painting tasks.

- **Chapter 6** describes the details of our pipeline and formulates the underlying problem to be solved.

- **Chapter 7** provides an experimental evaluation of our method, showcasing trajectories generated by pre-trained and fine-tuned policies, evaluated on a painter simulator.
- **Chapter 8** offers a concluding summary, highlighting key takeaways, and suggesting directions for future research.

Chapter 2

Background

This chapter serves as a foundational introduction that lays the necessary background for the subsequent parts of this thesis. Section 2.1 delves into the principles of Reinforcement Learning (RL), explaining its goals and the terminology used in RL algorithms. We introduce the framework of Markov Decision Processes (MDPs), which plays a pivotal role in formalizing the problems addressed within the realm of RL. Next, Section 2.2 provides an exposition of additional mathematical concepts for understanding entropy, which is the core of the unsupervised RL algorithm used in this work. Finally, Section 2.3 provides an overview of the robotic spray painting task, and how it is modelled according to the CPP problem formulation. These foundational elements collectively serve as the cornerstone for the comprehensive exploration and analysis presented in this thesis.

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes

A Markov Decision Process (MDP) is the foundational model for decision-making problems [23]. Within this framework, a decision-maker, often referred to as an *agent*, is situated within an *environment*. The agent can interact with the environment by executing *actions*, thereby influencing and altering the environment's current *state*. The environment's current state, typically referred to as *state*, or a part of it, is observed by the agent, allowing it to make informed decisions.

This interaction occurs in a sequence of discrete time intervals, represented as $t = 0, 1, 2, 3, ..., T$. At each time step, denoted as t, the agent uses the information from the current state s_t or observations to select an action to execute. After performing the action, the agent receives numerical reward signal r_{t+1} from the environment and can observe how the environment changed, observing the new state s_{t+1}. In Figure 2.1, we can observe the scheme of an MDP, in which are described the interactions between the environment and the agent. The agent's

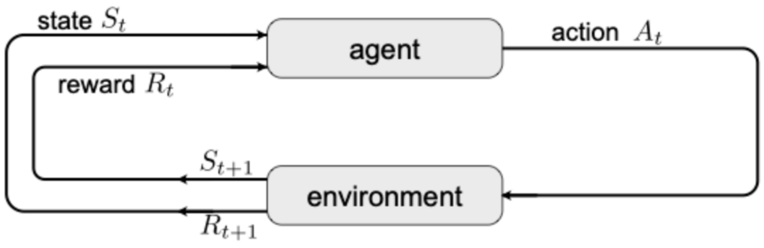


Figure 2.1: Agent-Environment interaction loop

interactions with the Markov Decision Process (MDP) result in the generation of a sequence including States, Actions, and Rewards, often referred to as a *trajectory*. A discrete-time MDP can be formally defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, d_0, T \rangle$, where:

- $\mathcal{S} \in R^{n_s}$ represents an n_s-dimensional continuous space, commonly referred to as the state space. This state space embodies what the agent can perceive in its environment, such as measurements from sensors in a robot.

- $\mathcal{A} \in R^{n_a}$ corresponds to an n_a-dimensional continuous space, typically denoted as the action space. The action space encapsulates the set of actions at the agent's disposal, signifying the range of possible torques that a motor can output, for instance.

- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ serves as a Markovian transition model, where $\Delta(\mathcal{S})$ denotes the simplex over the state space. $P(s'|s, a)$ indicates the conditional probability of transitioning to the next state s', given the current state s and an action a. This function defines the dynamics of the environment.

- $R : \mathcal{S} \times \mathcal{A} \to R$ represents the reward function, with $R(s, a)$ indicating the anticipated scalar reward received after taking action in a state s. Alternatively, the reward function can be defined as $R : \mathcal{S} \to R$, wherein $R(s)$ is the scalar reward granted upon reaching state s. This signal plays a pivotal role in guiding the learning process and encoding the desired task performance the agent is expected to achieve.

- $d_0 \in \Delta(\mathcal{S})$ corresponds to the initial state distribution, with $d_0(s)$ specifying the probability that the initial state s_0 is equivalent to s.

- T is the time horizon, which denotes the maximum length of a trajectory and $T \in \mathbb{N}$.

While we have defined continuous action and state spaces, it's worth noting that they can also be formulated as discrete sets. It's essential to highlight that the transition model $P(s'|s,a)$, given the Markov property, implies that transitions depend solely on the current state and action, without any consideration of previously visited states or actions taken.

2.1.2 Policies

In this section, we describe the role of an agent in the interaction process with the environment. This interaction is characterized by a policy, denoted as $\pi : \mathcal{S} \to \mathcal{A}$, which represents the agent's strategy for selecting actions given observations within an MDP. The policies can be divided into two categories:

- **Markovian (M) policies** Π_M: These policies collapse to a single, time-invariant decision rule $\pi = (\pi, \pi, \ldots)$, i.e., the same policy at each time-step, such that $\pi : \mathrm{S} \to \Delta(\mathcal{A})$. They map a state s to a probability distribution $\pi(\cdot|s)$.
- **Non-Markovian (NM) policies** Π_{NM}: In this category, each policy $\pi \in \Pi_{NM}$ simplifies to a single time-invariant decision rule $\pi = (\pi, \pi, \ldots)$, represented as $\pi : \mathcal{H} \to \Delta(\mathcal{A})$, where $\mathcal{H}$ denotes the history space. These policies consider the entire history $h \in \mathcal{H}$ for action selection.

Additionally, there are other two categories:

- **Deterministic** policies, which select an action with probability 1, represented as $s_{t+1} = f(s_t, a_t)$.
- **Stochastic** policies, which select an action with a probability $p \in$ [0,1], represented as $s_{t+1} \sim P(\cdot|s_t, a_t)$.

With the formalization of the MDP and the policy's definition in mind, we now formalize the agent's interaction with the MDP. This interaction starts at time $t = 0$, with the agent situated in an initial state s_0 determined by the distribution d_0. At each time step, the agent chooses an action a_t to execute, according to the policy $\pi(\cdot|s_t)$. The environment then changes into a new state s_{t+1} according to its transition model, and the agent receives a reward signal r_{t+1}. This interaction persists until the predefined horizon T is reached. A trajectory $\tau \in \mathcal{T}$, where $\mathcal{T}$ is the set of trajectories, is defined as a sequence $\langle s_t, a_t, r_t \rangle_{t=0,\ldots,T-1}$, encapsulating the states, actions, and rewards gathered during the agent's interaction process. The probability that a policy π induces a trajectory τ in the environment is defined as:

$$\rho(\tau|\pi) = d_0(s_0) \prod_{t}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.1}$$

2.1.3 State Distribution

An agent, which acts according to a policy π in an MDP, induces a distribution over the states of the MDP. The probability of reaching a state s at time step t during the interaction loop can be defined as:

$$d_t^\pi(s) = P(s_t = s|\pi). \tag{2.2}$$

This distribution can also be expressed in terms of the set of possible trajectories $\mathcal{T}$ as:

$$d_t^\pi(s) = \int_{\mathcal{T}} P(\tau|\pi, s_t = s)\, d\tau. \tag{2.3}$$

Here, $P(\tau|\pi, s_t = s)$ is the probability of being in a trajectory τ when acting according to policy π and being in the state s_t within that trajectory.

2.1.4 The Reward

The reward function R plays a crucial role in reinforcement learning and depends on the current state of the world, the taken action, and the next state of the environment, often represented as:

$$r_t = R(s_t, a_t, s_{t+1}) \tag{2.4}$$

The reward functions could be stochastic or deterministic, but we restrict our focus on deterministic ones. Frequently, the reward function is simplified considering just the current state, $r_t = R(s_t)$, or the state-action pair, $r_t = R(s_t, a_t)$.

The primary goal of the agent is to maximize the return over a trajectory, commonly called *cumulative reward*, which can take on various forms. One type of return is the infinite or finite-horizon undiscounted return, which represents the sum of rewards obtained in the sequence of time steps:

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{2.5}$$

Another type of return is infinite or finite-horizon discounted return, which accounts for the sum of all rewards acquired by the agent, but these rewards are discounted based on how far in the future they are obtained. This formulation of reward introduces a discount factor $\gamma \in (0,1)$:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \tag{2.6}$$

Introducing a discount factor, denoted as γ, on an intuitive level, reflects the principle that having access to rewards in the present holds a more significant value than acquiring them in the distant future. Mathematically, an unbounded summation of rewards over an infinite horizon may fail to converge to a finite value, rendering it challenging to incorporate into equations and analytical frameworks. However, the inclusion of a discount factor, when applied under reasonable conditions, ensures the convergence of the infinite sum. Thus, the introduction of γ in the reward formulation strikes a balance between capturing the value of immediate rewards and facilitating mathematical treatment.

2.1.5 The RL Problem

Irrespective of the selected return function — infinite or finite-horizon and discounted or undiscounted — and regardless of the chosen policy, the core objective within the domain of Reinforcement Learning (RL) [1] is to identify a policy that maximizes the expected return when performing a task. Let's consider a scenario in which both environmental transitions and the policy are stochastic. In this case, the probability of observing a T-step trajectory is expressed as in Equation 2.1. The expected return, denoted as $J(\pi)$, can be formally represented as:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right] \tag{2.7}$$

As a result, the central optimization problem in RL can be stated as:

$$\pi^* \in \arg\max_{\pi\in\Pi} J(\pi), \tag{2.8}$$

where π^* signifies the optimal policy actively sought within the realm of RL.

2.1.6 Value Functions and Bellman Equations

In Reinforcement Learning, understanding the concept of value functions is very important. These functions allow us to quantify the expected returns an agent can achieve in various states and state-action pairs, which are fundamental for the development of effective RL algorithms. Indeed, the goal of an agent interacting with an MDP is to maximize the reward sum collected in the *long term.* This means that the agent is far-sighted, i.e., it looks not only at the reward it can obtain in the short term but also at the one it can obtain in the future. This concept is expressed by the so-called *value functions.* The value function is defined as follows:

$$V_t^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{t'=t}^{T-1} R\left(s_{t'}, a_{t'}\right) \mid s_t = s\right] \tag{2.9}$$

The equation defines the value $V_t^{\pi}(s)$ when the agent is the state s at the time t and following the policy π. Similarly, we can define the value function for a discounted MDP as:

$$V_\gamma^\pi(s) := \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{T-1} \gamma^t R\left(s_t, a_t\right) \mid s_0 = s\right] \tag{2.10}$$

The value function $V_\gamma^\pi(s)$ at state s depends on the value in the next state s'. For this reason, the Equation 2.10 can be written in a recursive formulation:

$$V_\gamma^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[R(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}\left[V_\gamma^\pi\left(s'\right)\right]\right] \tag{2.11}$$

which is named *Bellman expectation equation* [24].

In the same way, we can define the action-value function $Q_\gamma^\pi(s,a)$ based on the state-action pair. The two defined functions are linked in the following way:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right]$$
$$V^*(s) = \max_a Q^*(s,a)$$

where * means the optimal value.

2.1.7 Classes of Algorithms

In the world of Reinforcement Learning, several dichotomies arise from different algorithms' aspects. In [1] all the main characteristics and distinctions between the algorithms are described. One of the most important dichotomies is related to the question of whether the agent has access to or has to learn a model of the environment. The model of the environment is a function to predict state transitions and rewards. The algorithms are divided into:

- **Model-based**: The agent has a model of the environment,
- **Model-free**: The agent learns the model of the environment. It doesn't have any a-priori knowledge of the environment transition dynamics.

Another distinction is related to the length of trajectories, and the algorithms are divided into:

- **Infinite-horizon**: The experience is composed of a unique infinite-length trajectory;
- **Episodic**: The experience is divided into multiple finite-length trajectories.

Additionally, the algorithms are differentiated in when the learning happens and how the policy uses the collected data to learn:

- **On-policy**: The agent collects data using the latest learned policy and it uses these data to improve;
- **Off-policy**: The agent learns through the experience collected at any point of the training, collecting experience through the last learned policy and older ones;
- **Offline**: The agent learns through batches of previously collected trajectories.

Another distinction in an RL algorithm is related to the question of what to learn. The algorithm can be defined as:

- **Value-based**: The agent learns a value function and generates actions optimizing it at each step;
- **Policy-based**: The agent learns a representation of the policy and tries to optimize it;
- **Actor-Critic**: The agent learns the policy as in the policy-based methods and learns the value function as in the value-based methods. It has been designed to combine the advantages of value-based and policy-based methods [25].

2.1.8 Model-based and Model-free

One of the critical decisions in reinforcement learning algorithms revolves around whether the agent has access to or has to learn a model of the environment, which refers to a function capable of predicting state transitions and rewards. Generally, the two approaches are referred to as model-based or model-free. Model-based algorithms offer a significant advantage, indeed they empower the agent to plan, reasoning about the current state and action by predicting the consequences on the environment. The success of this approach is exemplified by AlphaZero [26], which substantially improves sample efficiency compared to model-free methods. However, a notable drawback is that a ground-truth model of the environment is typically unavailable to the agent. In such cases, the agent, to employ an environmental model, must learn it solely through experiential data, presenting several challenges. The most substantial challenge is the potential for bias in the learned model, which can be exploited by the agent, leading to a situation where the agent performs well based on the learned model but behaves sub-optimally or poorly in the actual environment. Model learning is intrinsically challenging, and even intensive efforts—investing considerable time and computational resources—may not yield significant benefits. In the end, while model-free methods may sacrifice the potential advantages in sample efficiency associated with having a model, they are often more straightforward to implement and fine-tune. All algorithms we will see in the next chapters, such as SARSA, Q-Learning, and MEPol, are model-free methods and, generally, these are more prevalent, having undergone more extensive development and testing compared to model-based methods.

2.1.9 On-policy, Off-Policy, and Offline RL

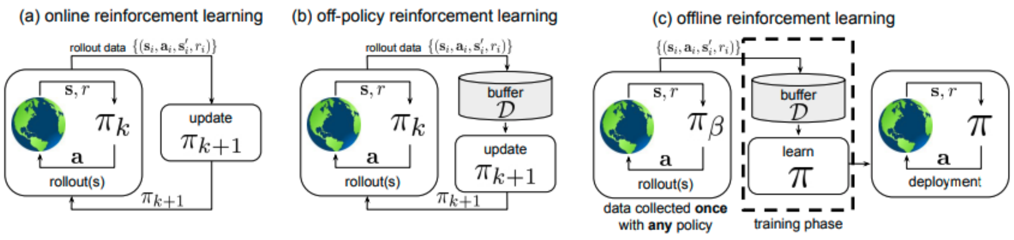

(a) online reinforcement learning
rollout data $\{(s_i, a_i, s'_i, r_i)\}$
s, r
π_k
a
rollout(s)
update
π_{k+1}
π_{k+1}
(b) off-policy reinforcement learning
rollout data $\{(s_i, a_i, s'_i, r_i)\}$
buffer
$\mathcal{D}$
update
π_{k+1}
rollout(s)
(c) offline reinforcement learning
$\{(s_i, a_i, s'_i, r_i)\}$
π_β
rollout(s)
data collected once with any policy
buffer
$\mathcal{D}$
learn
π
training phase
π
deployment

Figure 2.2: Visual representation depicting three distinct reinforcement learning scenarios, from [27]: classic on-policy reinforcement learning (a), classic off-policy reinforcement learning (b), and offline reinforcement learning (c). In on-policy reinforcement learning (a), the policy π_k continuously updates itself using the real-time data it generates. In classic off-policy reinforcement learning (b), the agent accumulates its experiences in a data buffer (also referred to as a replay buffer) D, with each new policy π_k gathering additional data, thus comprising samples from $\pi_0, \pi_1, \ldots, \pi_k$, all of which is employed to train an updated policy π_{k+1}. In contrast, offline reinforcement learning (c) utilizes a dataset D collected by a (potentially unknown) behaviour policy π_β. This dataset is collected once and remains unaltered during training, allowing for the utilization of large previously collected datasets. The training process remains decoupled from the Markov Decision Process (MDP), and the policy is deployed only after comprehensive training.

On-policy In *On-policy* RL, the agent updates it policy using recent data collected with the most recent policy [1]. This process involves continuous interaction between the agent and the environment to collect experience. Generally, in on-policy methods, the policy is *soft*, which means that $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$. A common example of on-policy learning is the *SARSA* algorithm, which stands for State-Action-Reward-State-Action. This algorithm updates the value of the current Q-function based on the reward and the value of the next action and observed state.

The SARSA algorithm learns a policy that strikes a balance between exploration and exploitation, finding applications in various domains such as robotics,

Algorithm 1 SARSA

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

gaming, and decision-making. Nevertheless, it's worth noting that SARSA's convergence can be slow, especially when dealing with large state spaces. In such cases, alternative reinforcement learning algorithms might offer more effective solutions.

On the bright side, on-policy learning exhibits greater consistency and stability. However, its drawback lies in its potential inefficiency due to the need to carefully manage exploration and exploitation.

In numerous scenarios, the on-policy approach may not be feasible or practical, primarily because data collection carries inherent risks (e.g., in autonomous driving) or substantial costs (e.g., in robotics) as discussed in Levine et al. [27].

Off-policy In *off-policy* RL, the agent collects experiences and stores the data in a buffer (also referred to as a replay buffer) D, with each new policy π_k gathering additional data, thus comprising samples from $\pi_0, \pi_1, \ldots, \pi_k$, all of which is employed to train a new updated policy π_{k+1}. That means the agent can use data from other sources or policies to improve its own policy. A common example of off-policy learning is Q-learning [28], which is a variant of SARSA that updates the value of the current action based on the reward and the maximum value of the next state, based on:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{2.12}$$

Here, the learned Q-function directly approximates the optimal action-value function, independent of the policy being followed [1]. This setting simplifies the

Algorithm 2 Q-Learning

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

analysis of the algorithm and enables early convergence. The policy has the power to choose which state-action pairs will be visited and updated in the Q-function. Off-policy learning offers the advantage of potentially greater speed and flexibility as it can leverage any available data and exploit optimal actions. However, it comes with the drawback of being more susceptible to errors and instability since it might either overestimate or underestimate the values of certain actions.

Offline RL In the category of *offline* RL [27], algorithms exclusively rely on pre-existing data and do not engage in additional online data collection. Consequently, the agent loses its capacity to interact with the environment and accumulates new transitions following the latest learned behaviour policy. Instead, the learning algorithm operates with a static dataset denoted as $D = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$, and it must derive the best possible policy solely from this fixed dataset. Then, the learning process lacks access to supplementary data since there is no interaction with the environment. In essence, offline reinforcement learning necessitates the learning algorithm to gain a comprehensive knowledge of the dynamic system inherent to the Markov Decision Process (MDP) solely from a fixed dataset. Subsequently, it builds a policy $\pi(a|s)$ that maximizes the cumulative reward when this policy is deployed for interactions with the MDP. As discussed by Levine et al. [27], a few scenarios in which offline RL might be involved are decision-making in health care, learning robotic manipulation skills, and Learning goal-directed dialogue policies.

2.1.10 Value-based Algorithms

In value-function methods, we implicitly learn the policy. Indeed, the goal is essentially to learn to estimate the value of specific states (or state-action pairs) based on past experiences. In general, in environments featuring multiple states and sequential decision-making, the value of a state is defined by the expected long-term return given that state as a starting point. Importantly, the policy becomes implicit, shifting the problem's complexity from optimizing π explicitly to estimating expected cumulative rewards. Then, these methods learn an estimator $Q_\theta(s, a)$ of the optimal action-value function, $Q^*(s, a)$. Since the goal is to learn the approximation Q, these algorithms are called Q-learning methods [28]. In Algorithm 2, we report the pseudocode of these methods. As mentioned in the previous chapter, this optimization is almost always performed off-policy, meaning that each update can use data collected at any point during training, regardless of how the agent chose to explore the environment when the data was obtained. The corresponding policy is obtained via the connection between Q^* and π^* and the actions taken by the Q-learning agent are given by

$$a(s) = \arg\max_a Q_\theta(s, a). \tag{2.13}$$

Practically speaking, tabular methods can explicitly represent these functions by storing each state entry $s \in \mathcal{S}$ and the corresponding optimal action $a \in \mathcal{A}$. However, for continuous state and action spaces, parameterized function approximators like neural networks are commonly employed. The primary challenge is to maximize these value functions while exploring the environment. Monte Carlo methods address this by sampling return estimates G_t at each time step while collecting complete episodes with a current policy π. These methods are particularly effective for episodic tasks and generally result in lower variance in return estimates. In contrast, Temporal Difference (TD) methods attempt to alleviate the need to wait for exact sampled returns by using bootstrapping. This involves the use of learned value function estimates to begin learning from a single transition. For example, TD methods can approximate $G_t \approx r_t + \gamma \hat{V}(s_{t+1})$ and update their belief about $\hat{V}(s)$ accordingly as soon as r_t is obtained. SARSA and Q-learning are examples of value-based TD methods for reinforcement learning problems with discrete action spaces. The latter has found success in the field of Deep Reinforcement Learning, particularly when estimating value functions in high-dimensional state spaces (e.g., images) using convolutional neural networks.

2.1.11 Policy Search Algorithms

In this section, we cover the mathematical foundations of policy optimization algorithms. This family of RL algorithms aims to find the optimal policy by directly searching in the policy space. The main Unsupervised RL algorithm used in this thesis is built upon gradient-based methods [29], [30], based on gradient ascent to conduct the policy optimization. Consider a stochastic, parameterized policy π_θ, and the goal of maximizing the expected return $J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}}[R(\tau)]$. To optimize the policy via gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \left.\nabla_\theta J(\pi_\theta)\right|_{\theta_k}. \tag{2.14}$$

Here, θ represents the vector of the parameters which define the policy. The policy gradient $\nabla_\theta J(\pi_\theta)$ is the key component of policy gradient algorithms. Indeed, the learning process is performed by computing the gradient of the policy performance with respect to the parameter vector. For this reason, the policy must be stochastic and differentiable in θ. For practical use, we need an expression for the policy gradient, that we can numerically compute with data from a finite number of agent-environment interaction steps. We can derive the gradient expression [25], as:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathbb{E}} R(\tau) \\
&= \nabla_\theta \int_\tau \rho(\tau|\theta) R(\tau) \\
&= \int_\tau \nabla_\theta \rho(\tau|\theta) R(\tau) \\
&= \int_\tau \rho(\tau|\theta) \nabla_\theta \log \rho(\tau|\theta) R(\tau) \\
&= \underset{\tau\sim\pi_\theta}{\mathbb{E}} \nabla_\theta \log \rho(\tau|\theta) R(\tau).
\end{aligned}$$

Then, we can define:

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau). \tag{2.15}$$

This expectation can be computed from the sampled data, i.e., the trajectory. The gradient estimation can be computed considering a batch of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$, collected during the agent-environment interaction process according to a policy π_0. We can compute the policy gradient as:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau), \tag{2.16}$$

where $|\mathcal{D}|$ is the number of trajectories in the batch $\mathcal{D}$. It's noteworthy that this formulation resembles the well-known cross-entropy loss employed in supervised learning. Indeed, with this gradient, we are updating the policy parameters to increase the likelihood of actions performed in trajectories with higher performance and reduce the likelihood of actions performed in the less promising ones. Equation 2.16 offers a straightforward approach for computing the gradient, typically by utilizing sample estimates of the expected value, a method akin to Monte Carlo techniques, such as REINFORCE [29].

In summary, the policy gradient algorithms' goal is to maximize the likelihood of each action chosen during an episode based on the rewards received. Negative returns result in a negative gradient, discouraging higher probabilities during the update step, while positive returns have the opposite effect. In practical applications, additional strategies are often employed to expedite the training process. These strategies include subtracting a constant (or state-dependent) baseline from the rewards and employing techniques like bootstrapping, as seen in Temporal-Difference methods.

An example of policy gradient methods widely adopted in reinforcement learning is the Trust Region Policy Optimization (TRPO) [31] and the Proximal Policy Optimization (PPO) [32] family of algorithms. These algorithms have gained popularity due to their ease of implementation and tuning while still achieving results close to the current state-of-the-art. Consequently, they have become the default choice for various applications in reinforcement learning, as demonstrated in Chapter 4.2, in which we show the MEPol [33] algorithm which is based on the principles of TRPO.

2.1.12 Actor-Critic Algorithms

Actor-critic methods [34], [35] represent a category of policy gradient algorithms that simultaneously learn a parameterized policy (actor) and estimate the value function (critic). The critic's role is to approximate future rewards in a Temporal-Difference fashion for policy gradient estimation, facilitating single-step updates. There are a number of different variants of actor-critic methods, including on-policy variants that directly estimate $V^\pi(s)$ [34], and off-policy variants that estimate $Q^\pi(s,a)$ via a parameterized state-action value function $Q^\pi_\phi(s,a)$ [36], [37], the pseudocode in the latter case is reported in Algorithm 3. Actor-critic methods can be seen as a policy-based method extension, in which the Q-function estimates the policy gradient (Equation 2.15) and it is learned through experience. In the actor-critic methods, the Soft Actor-Critic (SAC) [38] algorithm reached remarkable empirical success by optimizing an entropy-regularized objective in addition to the standard objective (Equation 2.8).

2.1.13 Open Challenges in RL

Despite notable advancements, RL faces several noteworthy challenges that impact its practicality. These challenges are related to sample efficiency [5] and generalization [6], which are essential for RL agents to perform effectively and adapt to new tasks. One of the central problems in RL revolves around the necessity for

Algorithm 3 Q Actor-Critic

```
Require: Initialize parameters $\theta$, $w$, learning rates $\alpha_\theta$, $\alpha_w$; sample $a \sim \pi_\theta(a \mid s)$.
 1: for $t = 1$ to $T$ do
 2:     Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s' \mid s, a)$
 3:     Sample the next action $a' \sim \pi_\theta(a' \mid s')$
 4:     Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a \mid s)$
 5:     Compute the correction (TD error) for action-value at time $t$:
 6:         $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
 7:     Use it to update the parameters of the Q function:
 8:     $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
 9:     Move to $a \leftarrow a'$ and $s \leftarrow s'$
10: end for
```

a specifically hand-crafted reward function, "*Curse of Goal Specification*" [7], [39], to define the desired task. Indeed, RL algorithms heavily rely on this reward signal to guide their learning process. In theory, having a perfectly shaped reward function would simplify the learning process, but in reality, these reward functions are often manually crafted and hard to optimize. This manual shaping of rewards poses a significant limitation to autonomous learning as it demands careful and task-specific formulation for each new task that an RL agent needs to tackle. Moreover, this task-specific reward formulation requirement results in a substantial cost and time overhead. Each time a new task is introduced, it necessitates the labour-intensive process of designing a tailored reward function, often referred to as reward shaping. This task-specificity not only escalates the complexity of RL implementation but also limits the transferability of knowledge from one RL problem to another. As a result, the potential for generalization across diverse tasks is reduced, hindering the efficiency and scalability of RL algorithms.

Other issues emerge when we use RL in the real world. Indeed, as discussed by Kormushev et al. [40], the application of RL on real robots faces several challenges such as safety, scalability, etc. The safety problems are certainly non-negligible, indeed, the robot policy should be safe not only for the robot itself, that it might break down during the training and repairing it would inevitably raise the cost, but also for the people around. Additionally, the training, or the early stage of the training, is performed in simulation. However, the use of simulated environments brings several problems, such as the so-called "*Curse of Under-Modeling and Model Uncertainty*" [39], [41]. In simulation, we need to use models to describe the real world, but, often, these models use approximations or they can't describe exactly the reality. The inaccuracy of models creates the problem of transferring the policy from simulation to the real world, "*Sim2Real Problem*" [42].

In this thesis, we show how the *Unsupervised Reinforcement Learning* approach

[8] seeks to solve some RL issues, such as the generalization and the sample efficiency problems.

2.2 Information Entropy

2.2.1 Information Content

The main idea of information theory [43] is that the meaningful content of a message is related to the degree to which the content of the message is surprising. The more often an event occurs, the less informative it is and vice versa. Mathematically, considering a random variable or vector X, with possible outcomes $x \in \{x_1, \dots, x_n\}$, the information content associated with X assuming a value x_i can be defined as:

$$I(x_i) = \log \frac{1}{P(x_i)} = -\log P(x_i). \tag{2.17}$$

Here, $P(x_i)$ is the probability of X to assume the value x_i. As mentioned above, the information content gives a value indicating how unlikely an event is.

2.2.2 Discrete Entropy

The information level in a random vector or variable X is measured using the discrete entropy defined by Shannon et al. [43]:

$$H(X) = \mathop{\mathbb{E}}_{x_i \in X} I(x_i) = -\sum_{i=1}^{N} P(x_i) \log P(x_i) \tag{2.18}$$

The entropy measures the expected information which brings a random variable X. Indeed, a higher entropy value corresponds to a less predictable variable, indicating that it carries more information. If we read the concept of disorder as unpredictability, we can easily understand why entropy is often interpreted as a measure of disorder. The maximum entropy for a discrete variable occurs when the distribution of outcomes is uniform, indicating that no outcome is more likely than any other and all outcomes are equally meaningful. It is instead minimal when $P(x_i)$ corresponds to a *Dirac* distribution, which indicates that one single outcome has the maximum probability, while the others have a zero probability. The entropy value is always positive, since the probabilities $0 \leq P(x_i) \leq 1$ and the $\log P(x_i) \leq 0$. As a result, each term in the summation is either negative or zero, ensuring that the overall discrete entropy is always positive.

2.2.3 Differential Entropy

The concept of discrete entropy can be extended to random vectors with continuous support. Let $f(X)$ be the probability density of the random vector X with values in $\mathbb{R}^p$, its differential entropy [43] is defined as:

$$H(X) = -\mathbb{E}\left[\log(f(x))\right] = -\int_{\mathcal{X}} f(x) log(f(x))dx \tag{2.19}$$

where $\mathcal{X}$ is the support region of the random variable X and $f(x)$ is the probability density function of X.

2.2.4 Non-Parametric Differential Entropy Estimator

When the distribution f is not available, the differential entropy can be estimated through the realization of $X = \{x_i\}_{i=1}^N$ [44]. In the Unsupervised RL algorithms used in this thesis, to deal with high-dimensional data, we used the non-parametric k-Nearest Neighbors entropy estimators defined by Sing et al. [45]:

$$\widehat{H}_k(f) = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{k}{NV_i^k} + \log k - \Psi(k) \tag{2.20}$$

where $\log k - \Psi(k)$ serves as a bias correction term for the estimate, while $\log \frac{k}{NV_i^k}$ measures the density of samples in proximity to the i-th sample. The latter is then averaged across all samples to provide an estimation of their distribution. In Equation 2.20, Ψ is the digamma function and V_k represents the volume of the hyper-sphere of radius $R_i = |x_i - x_{\text{knn}}|$, which denotes the Euclidean distance between x_i and its k-nearest neighbor x_{knn} and:

$$V_i^k = \frac{\left|x_i - x_i^{knn}\right|^p \cdot \pi^{p/2}}{\Gamma\left(\frac{p}{2}+1\right)}, \tag{2.21}$$

where Γ is the gamma function, and p is the dimension of X. In the end, the estimator in 2.20 is known to be asymptotically unbiased and consistent [45].

Considering an off-policy scenario, we can obtain samples from a distribution f that may differ from the target distribution f'. In this case, we can provide an estimate of $H(f')$ by means of an Importance-Weighted (IW) k-nearest neighbour estimator [46] to compute the entropy under f', based on samples collected using f:

$$\widehat{H}_k\left(f' \mid f\right) = -\sum_{i=1}^{N} \frac{W_i}{k} \ln \frac{W_i}{V_i^k} + \ln k - \Psi(k), \tag{2.22}$$

where $W_i = \sum_{j \in N_i^k} w_j$ is the sum of the normalized importance weights w_j computed over the set N_i^k of k-nearest neighbour samples of x_i. The normalized importance weights w_j can be computed as:

$$w_j = \frac{f'(x_j)/f(x_j)}{\sum_{n=1}^{N} f'(x_n)/f(x_n)}. \tag{2.23}$$

Starting from the normalized importance weights, it is also possible to compute an estimate of the Kullback-Leibler (KL) divergence:

$$\hat{D}_{KL}(f' \| f) = \frac{1}{N} \sum_{i=1}^{N} \log \frac{k/N}{\sum_{N_i^k} w_i}. \tag{2.24}$$

This measure is closely related to entropy and is sometimes referred to as relative entropy. It serves as a divergence measure between two distributions. We can note that, when $f' = f$ and $w_j = \frac{1}{N}$, the estimator in Equation 2.22 is equivalent to Equation 2.20 and $\hat{D}_{KL}(f' \| f) = 0$.

2.3 Robotic Spray Painting and Coverage Path Planning

2.3.1 Robotic Spray Painting

Spray painting has played a pivotal role in industrial applications for over a century, providing an efficient means to achieve uniform paint coverage on a wide range of surfaces and components. For years, this process relied on manual operations, which still persists even in highly automated industries. Nevertheless, the landscape of industrial spray painting has evolved significantly over the last 50 years, with automation becoming increasingly prevalent. This transition towards automation is driven by its potential to enhance quality while simultaneously reducing costs and health risks [47].

From early developments in automation, the importance of automated spray systems emerged. Over time, these systems have evolved into fully automated and robotized spray-painting booths that are commonplace in many industries today. Robotic spray painting, in particular, has become a standard technique in highly automated production processes, notably within the automotive industry. Often, the process involves the use of manual teaching to record and then replicate robot trajectories. However, the advent of offline simulation has revolutionized this domain by simplifying the work for spray-painting engineers.

Accurate simulation of the spray-painting process enables engineers to predict the resulting paint thickness for a given robot trajectory, offering valuable feedback for trajectory optimization. This approach not only reduces the need for physical testing but also allows for the modification and improvement of robotic trajectories before implementation and testing in real-world scenarios. With a focus on achieving precise simulations, attention subsequently shifts towards solving the painting problem. This problem consists of finding a painting trajectory that, when executed, produces a paint thickness that closely matches a specified target thickness. This optimization problem seeks to determine an optimal painting trajectory given a desired paint thickness and a model of the surface to be painted.

Despite the importance of spray painting in the industry, trajectory generation has not reached a sufficient level of automation. Researchers' interest is shifting to this topic, but there is not yet a well-established literature on it. The goal of this thesis is to combine the RL framework with the robotic spray painting problem, focusing not only on the final model performance but also on the computational efficiency of the entire training process.

2.3.2 Coverage Path Planning Problems

Coverage Path Planning (CPP) is the task of determining a path that passes over all points of an area of interest while avoiding obstacles [48]. This task plays a crucial role in many robotic applications, such as vacuum cleaning robots [49],[11] and painter robots [50]. However, the main objective of CPP is to determine a path or a sequence of waypoints, which guarantee the complete coverage of an area and, simultaneously, the trajectory must satisfy several criteria. As discussed in [48], the requirements that a generated trajectory must satisfy are:

- the robot must move through all the points of the target area,
- robot must fill the region without overlapping paths,
- robot must avoid obstacles,
- simple trajectory must be used (e.g., straight lines or circles),
- an *optimal* path is desired.

However, it is not always possible to satisfy all these criteria in complex environments. Therefore, sometimes a priority assignment is required, for example, giving priority to optimal coverage with respect to the trajectory length.

The coverage algorithms can be classified as heuristic or complete depending on whether or not it is proven that they guarantee complete coverage of the space [9]. As in [51], another distinction in these algorithms is related to the data they use. Indeed, an offline algorithm relies on stationary information, supposing that there is a full prior knowledge about the environment. This latter scenario is unrealistic in many real-world scenarios. On the other hand, we can classify a coverage algorithm as online if we do not assume any prior knowledge related to the environment. In this case, the robot relies on real-time data coming from sensor measurements to sweep the target space. This latter class could be also called sensor-based coverage algorithms. Since robots usually work in unknown scenarios or partially unknown environments, CPP can be combined with other tasks, such as environmental mapping. Additionally, these problems are proven to be NP-hard, since the computational time to solve them drastically increases as the dimensionality of the problem grows. In certain settings, a random policy with an infinite horizon is a valid approach to solving coverage problems. Indeed, if an agent moves randomly over a surface for an infinite interval of time is guaranteed that the agent sooner or later will pass over all points on the surface. Nevertheless, this solution is impractical in real-world applications and researchers created algorithms, based on cellular decomposition and adjacency graphs, such as trapezoidal [52], [53] or boustrophedon decomposition [54]. Instead, in this thesis, we show a method to tackle coverage path planning using the RL framework.

Chapter 3

Unsupervised Reinforcement Learning

In this chapter, we will explore the details of the *Unsupervised Reinforcement Learning* (URL) framework, introducing the fundamental principles of these algorithms. Following a concise introduction to the topic, Section 3.2 presents an examination of the diverse techniques employed by URL algorithms, along with an analysis of their inherent constraints.

3.1 Introduction

Previously, in Section 2.1, we introduced the RL framework [1] for enabling robots to acquire task-solving capabilities in challenging environments. Despite its successes, RL has exhibited various challenges and limitations, including issues like reward shaping and sample efficiency during the learning process. These challenges pose significant obstacles to the development and generalization of RL.

In response to these limitations, the *Unsupervised RL* (URL) framework [8] has emerged as a promising solution. The URL framework operates through a two-step learning process. The initial phase involves an unsupervised pre-training step in which the agent acquires knowledge, such as information related to transition dynamics and state representations within its interacting environments. The primary objective of this phase is to provide a pre-trained model for the subsequent supervised fine-tuning phase. In the supervised fine-tuning phase, the agent learns how to accomplish specific tasks using predefined reward functions. Utilizing a pre-trained policy, rather than a randomly initialized one, offers substantial advantages in the training process [8], [55], [18]. After unsupervised pre-training, the model leverages the knowledge acquired during the unsupervised phase, enhancing its efficacy in mastering the desired task.

This approach also empowers the agent to discover latent dataset characteristics that conventional supervised methods might neglect, evaluating them as insignificant or rare outliers. In traditional learning processes, overlooked characteristics can impede efficient learning for such data samples. Then, this unsupervised approach allows the agent to discover improbable states that during the task-specific learning process might be neglected. Furthermore, during the supervised training phase, a randomly initialized policy can often struggle to proficiently tackle a specific task, particularly when dealing with sparse reward signals. By employing unsupervised pre-training, wherein the agent is tasked to acquire valuable skills and explore a wide range of states, the URL framework facilitates the development of a policy that can effectively tackle the downstream task learning even when confronted with sparse rewards.

3.2 Intrinsic Rewards

The URL framework is characterized by the absence of an extrinsic hand-crafted reward function. However, there is still a reward signal, called *Intrinsic Reward*, to make the agent explore the environment. Intrinsic reward, unlike task-specific extrinsic reward functions, refers to exploration and it encourages the agent to learn to collect diverse experiences or to develop useful skills. In this section, we will provide a complete overview of the commonly used intrinsic reward functions and we will underline their advantages and disadvantages. We can classify the approaches of intrinsic reward into three categories: curiosity-driven exploration, skill discovery, and maximal data coverage [56].

3.2.1 Curiosity-driven Exploration

Curiosity-driven exploration aims to increase knowledge about the environment. The exploration follows the principle that if the agent fails the prediction on the next state, it has to explore this new state to reduce its uncertainty. Indeed, the agent has to explore the states in which there is more uncertainty and, then, the prediction error is likely higher. In practice, the goal of the agent is to maximize the error of the next-state prediction made by the learned dynamics model. A concrete example of this principle is the ICM algorithm [57], in which the intrinsic reward is proportional to the prediction error as:

$$r_{\text{intrinsic}} \propto \left\| f(\phi(s_t), a_t) - \phi(s_{t+1}) \right\|_2^2 , \tag{3.1}$$

where f and ϕ represent respectively the learned forward dynamics model and the feature encoder.

Limitations One of the most important issues is related to the difficulties in distinguishing the *epistemic* and *aleatoric* uncertainty [56]. The first refers to uncertainty in the lack of knowledge, while the latter is related to variability in the outcome due to random effects. An example of this phenomenon appears in the noisy TV problem [58], where the agent gets trapped by its curiosity in a highly stochastic environment.

3.2.2 Skill Discovery

Another URL is based on learning a set of useful skills, which can be useful for solving downstream tasks, in a model-free setting. This approach is commonly called skill discovery. The core intuition is based on the fact that the learned skill controls which set of states the agent will visit. For example, in the Ant Mujoco environment [59] the visited states will be different depending on whether the ant moves forward or backward. Practically speaking, the skill discovery objective is to maximize the mutual information between the skill latent vector z and the state s:

$$I(s; z) = H(z) - H(z|s) = H(s) - H(s|z). \tag{3.2}$$

The use of mutual information between skills and states encodes the idea that the skill determines which states the agent will visit. An algorithm based on the skill discovery idea is DIAYN [60]. Here, Eysenbach et al. [60], provided a method based on mutual information for learning skills without an extrinsic reward function and, once the agent learns different skills in an MDP, it can learn easier downstream tasks.

Limitations One of the main issues of the skill discovery-based algorithms is that an agent does not necessarily visit a huge amount of states, since it can maximize the mutual information objectives in a small state variation [61], [62]. This limitation in state coverage could in turn limit the learning of downstream tasks in complex environments [61].

Additionally, due to the restricted skill space, the performance of skill discovery methods is lower than other pertaining methods [8].

3.2.3 Data Coverage Maximization

Other pertaining strategies are based on the objective of maximizing the data diversity. This idea is directly related to exploration and coverage, since the agent, to satisfy the goal, has to visit different states. In the literature, this objective is formalized in two ways: using counters of the visited states or using entropy.

Count-based exploration

This first way of formalizing the data coverage objective is called count-based exploration. This method directly counts the visited states to induce the agent to go towards new and unexplored states [63]. In this class of algorithms, the state-action counters $N(s_t, a_t)$ serves as exploration bonus reward:

$$r_{\text{intrinsic}} \propto N(s_t, a_t)^{-\frac{1}{2}} \tag{3.3}$$

However, the previous equation is intractable in high-dimensional spaces. To solve this limitation, Bellaemare et al. [63] introduced the *pseudo-counts* using density models and the counter is defined as follows:

$$\hat{N}(s) = \frac{\rho_t(s)(1 - \rho'_t(s))}{\rho'_t(s)) - \rho_t(s))}, \tag{3.4}$$

in which ρ is the density model over state space $\mathcal{S}$, $\rho_t(s)$ is the density computed after the training on a set of states and $\rho'_t(s)$ is the density computed after the training on an additional set of states.

Limitations In previous works [64], it has been proven that count-based methods suffer from *detachment*, because the agent loses track of interesting areas to explore, and *derailment*, because the agent, following the exploration objective, is prevented from going back to previously visited states. Additionally, this approach drives the agent to be stuck in local minima [65].

Entropy Maximization

This latter class of algorithms is based on the idea of maximizing the entropy to encourage novel state visitation and achieve data diversity. This objective can be formalized as:

$$\pi^* \in \underset{\pi \in \Pi}{arg\,max}\, H(d_\pi) \tag{3.5}$$

In this equation, d_π is the state distribution induced by a policy $\pi_\theta(a|s)$ and $H(\cdot)$ could be the Shannon entropy [43], the k-NN entropy estimator [45], etc.

The state entropy maximization objective is notoriously hard to estimate and optimize. In recent years, many algorithms have used different ways to perform the optimization. Several methods have been proposed to solve this objective, such as the algorithms APT [66], ProtoRL [67] and Mepol [33]. All three of them use the particle-based entropy estimator (2.20) trying to maximize the distance between k-nearest neighbours.

Limitations In addition to the complexity of the state entropy optimization, it has been shown theoretically that the Markovian class of policies are insufficient to solve the entropy objective, while non-Markovian policies guarantee good exploration [68].

Chapter 4

The State Entropy Objective

This chapter covers the state-entropy maximization objective. First, we offer a mathematical formulation of the objective in Section 4.1. Later, we explain the main algorithms used to reach this objective. The first algorithm we present in Section 4.2 is Maximum Entropy POLicy optimizaton (MEPol), which belongs to the policy gradient methods [69] category and is guided by the state-entropy maximization objective. We then present two extensions of MEPol, α-MEPol [70] and MEMENTO [71], with better performance for working in multi-environment settings.

4.1 Introduction

Maximum state entropy methods aim to define an objective that facilitates the learning of an exploratory policy, which can subsequently be fine-tuned to effectively address various tasks. The ideal objective here is to encourage a uniform exploration of the state space, essentially maximizing the entropy of the state distribution generated by the policy. Hazan et al. introduce this objective in their work [12] as follows:

$$\pi^* \in \arg\max_{\pi \in \Pi} H(d_\pi). \tag{4.1}$$

Here, Π represents the policy space, d_π denotes the state distribution induced by policy π, and $H(d_\pi)$ represents the entropy of the state distribution. Several methods [12], [33], [66], [67], [72]–[75] have been proposed to optimize this particular objective. Hazan et al. [12] propose a method that employs the Frank-Wolfe algorithm to approximate gradients for learning an optimal mixture of policies. In another approach outlined in [76], the agent is provided with a target distribution d^* that represents the desired state distribution. The exploration objective is reformulated as:

$$\min_{\pi \in \Pi} D_{KL}(d_\pi || d^*) = \max_{\pi \in \Pi} \mathbb{E}_{d_\pi(s)} \log d^* + H(d_\pi) \tag{4.2}$$

Here, D_{KL} represents the Kullback-Leibler divergence between the distribution induced by policy π and the target distribution d^*. This objective is used to optimize a mixture of policies, employing a density model to estimate the entropy of the state distribution. In [33], a novel approach employs a K-nearest neighbour estimator for the entropy of the state distribution induced by a policy. Building upon this concept, Seo et al. [75] extend the idea to environments with visual inputs, utilizing estimates in the reduced dimensionality obtained through randomly initialized convolutional encoders.

4.2 MEPol

Mutti et al. [33], introduced an algorithm known as *Maximum Entropy POLicy optimizaton (MEPol)*, to train a single policy in a continuous high-dimensional domain. The algorithm is designed to compute at every epoch the maximization for the entropy estimation using gradient descent [77], according to policy gradient methods. MEPol employs a non-parametric entropy estimation method based on the importance-weighted entropy estimator described in Section 2.2.4 to calculate the entropy of the state distribution d_π across the state space.

MEPol combines this entropy estimator with an offline importance sampling optimization technique [78]. This approach allows for the efficient recycling of samples collected using earlier policies. After the selection of the horizon T and the parameter $k \in \mathbb{N}$, a batch of trajectories with T elements is sampled from the environment, obtaining N samples and a set of states $\mathcal{D} = \{s_i\}_{i=1}^N$, where $s_i \sim d_T^{\boldsymbol{\theta}}$. Additionally, it employs gradient ascent within a trust-region boundary [31] to update the agent's policy, limiting the policy's divergence from the one used during sample collection. As in the TRPO method [31], the divergence is quantified as the Kullback-Leibler (KL) divergence between the distributions induced by the two policies, as estimated in equation 2.24. Then, given a trust-region threshold δ, the algorithm aims to solve:

$$\begin{aligned} \underset{\boldsymbol{\theta}' \in \Theta}{\text{maximize}} \quad & \widehat{H}_k \left(d_T^{\boldsymbol{\theta}'} \right) \\ \text{subject to} \quad & \widehat{D}_{KL} \left(d_T^{\boldsymbol{\theta}} || d_T^{\boldsymbol{\theta}'} \right) \leq \delta. \end{aligned} \tag{4.3}$$

In summary, the algorithm operates by gathering batches of trajectories and using these batches to estimate the entropy of the state distribution. Subsequently,

Algorithm 4 MEPol

```
Input: Horizon $T$, sample-size $N$, trust-region threshold $\delta$, learning rate $\alpha$,
nearest neighbors $k$
Initialize $\theta$
while not converged do
    Draw a batch of $\left\lceil \frac{N}{T} \right\rceil$ trajectories of length $T$
    Build a dataset of particles $D_\tau = \{(\tau_i^t, s_i)\}$
    $\theta' \leftarrow \theta$, $g \leftarrow 0$
    while $D_{KL}\left(\hat{d}_T(\theta) \| \hat{d}_T(\theta')\right) \leq \delta$ do
        $\theta' = \theta' + \alpha \nabla_{\theta'} \hat{H}_k \left(\hat{d}_T(\theta') \mid \hat{d}_T(\theta)\right)$
    end while
    $\theta \leftarrow \theta'$
end while
return task-agnostic policy $\pi_\theta$
```

it updates its policy using off-policy updates based on this estimate, i.e., the agent does more policy updates using the same entropy estimation. Indeed, the collected samples are reused until the policy surpasses a predefined threshold δ for the estimated KL divergence. This iterative process continues until convergence is achieved.

4.3 Learning in Multiple Environments

Our goal in this section is to to present advanced URL methods considering the problem of *state entropy maximization in multiple environments*. Indeed, the algorithms presented in the previous sections are environment-specific. In the multiple environment setting, during the pre-training phase, the agent faces a series of different reward-free environments belonging to the same domain, with the same states and action spaces, but with a different transition dynamic. Several works [70], [79] tackle the multiple environment settings to improve URL policy transferability across environments. Mutti et al. [70], during the learning process, situate the agent into an environment sampled from the environment class, where it interacts before facing a new environment. As they underlined, the ultimate goal of the agent is to pre-train a maximum entropy policy helpful to solve *any* subsequent fine-tuning task that can be specified over *any* environment of the class. In this setting, the pre-training becomes a *multi-objective* problem, since we aim to maximize the entropy over different environments. Before presenting the α-MEPol [70] algorithm, we present other previous works which faced the above-mentioned setting.

In a prior study, Rajendran et al. [80] explore a learning procedure that consists of two distinct phases: an initial agnostic pre-training phase, referred to as *practice*, followed by a subsequent supervised fine-tuning phase known as *match*. In their framework, these two phases are interleaved, and the supervision signal obtained during the fine-tuning phase facilitates the learning of reward signals for the practice phase through a meta-gradient approach. On the other hand, Parisi et al. [79] explore the domain of unsupervised reinforcement learning in multiple environments in which they introduce a fundamentally distinct approach with respect to the one used in Section 4.3.1. They adopt a pre-training objective inspired by count-based methods [63] instead of the entropy-based objective. Furthermore, in the context of multiple environments, they devise a specific bonus system for establishing a uniform preference across the entire class rather than prioritizing the worst-case environment as in Section 4.3.1.

4.3.1 αMEPol

A straightforward method for extending a single-environment algorithm to discover an exploratory policy across multiple environments involves averaging the entropies acquired from training in each of these environments. Given a set of environments, denoted as $M = \{M_1, \dots, M_I\}$, characterized by diverse transition models P_i but sharing common action and state spaces, and a probability distribution p_M over this set, the problem can be viewed as multi-objective. It allows for expressing preferences among the environments within M by assigning different weights when computing the average entropy.

However, a drawback of simply averaging the entropies across trajectories from different environments is that it overlooks the tail end of entropies stemming from unfavourable or infrequent environments. To address this limitation, an extension called *α-sensitive Maximum Entropy POLicy optimization* (αMEPol) [70] has been introduced, building upon the original MEPol approach. αMEPol incorporates a novel objective, in which they consider the mean of a critical percentile of the objective function, i.e., its Conditional Value-at-Risk (CVaR) [81] at level α, to prioritize the entropy achieved in particularly rare or adverse environments, applied to the entropies of collected trajectories:

$$\mathrm{CVaR}_\alpha\left(H_\tau\right) = \underset{\substack{\mathcal{M}\sim p_{\mathcal{M}} \\ \tau\sim p_{\pi,\mathcal{M}}}}{\mathbb{E}} \left[H_\tau \mid H_\tau \leq \mathrm{VaR}\, R_\alpha\left(H_\tau\right)\right] \tag{4.4}$$

In this equation, H_τ stands for the entropy of the state distribution, estimated in Equation 2.22, with the states visited along trajectory τ. This objective enhances the optimization of entropy, particularly in unfavourable or rare environments, making it more resilient to challenging transition functions. The algorithm acts as

Algorithm 5 αMEPol

```
Input: percentile $\alpha$, learning rate $\beta$
Output: policy $\pi_{\boldsymbol{\theta}}$
Initialize $\boldsymbol{\theta}$
for epoch = 0,1, ..., until convergence do
    for $i = 1,2,\ldots,N$ do
        Sample an environment $\mathcal{M}_i \sim p_{\mathcal{M}}$
        Sample a trajectory $\tau_i \sim p_{\pi_{\boldsymbol{\theta}},\mathcal{M}_i}$
        Estimate $H_{\tau_i}$ with (3)
    end for
    Estimate $\mathrm{VaR}_\alpha(H_\tau)$ with (4)
    Estimate $\nabla_{\boldsymbol{\theta}}\mathcal{E}^{\alpha}_{\mathcal{M}}(\pi_{\boldsymbol{\theta}})$ with (5)
    Update parameters $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{E}^{\alpha}_{\mathcal{M}}(\pi_{\boldsymbol{\theta}})$
end for
```

a policy gradient method [29], [30]. It directly optimizes the parameters θ for finding the optimal policy. The algorithm updates repeatedly the policy parameters until a stationary point is reached. The update is performed as:

$$\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta} + \beta\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{E}^{\alpha}_{\mathcal{M}}(\pi_{\boldsymbol{\theta}}). \tag{4.5}$$

In this equation, β is the learning rate and the $\widehat{\nabla}_{\boldsymbol{\theta}}\mathcal{E}^{\alpha}_{\mathcal{M}}(\pi_{\boldsymbol{\theta}})$ is the gradient of Equation 4.4 with respect to the parameters $\boldsymbol{\theta}$ of the policy.

4.4 Non-Markovianity in URL

In their work, Mutti et al. [68] demonstrate the limitations of a Markovian policy in the context of unsupervised pre-training. They start from the intuition that using the history of interactions becomes valuable when the agent's objective is to explore the environment uniformly. Having knowledge of past visits allows for more informed decision-making. To illustrate this point, consider an illustrative example in which the agent is placed in the middle of a two-room domain (as depicted in Figure 4.1) and has a limited budget of interactions, just enough to visit every state within a single episode. In this scenario, it becomes evident that an optimal Markovian strategy, optimized for the Maximum State Entropy (MSE) objective, would involve randomizing between moving left and right from the initial position. Subsequently, it would follow the most efficient route within a room, ultimately returning to the initial position. As a result, each episode would either involve visiting the left room twice, the right room twice, or both rooms once, with all these outcomes carrying equal probabilities. Thus, while the agent's

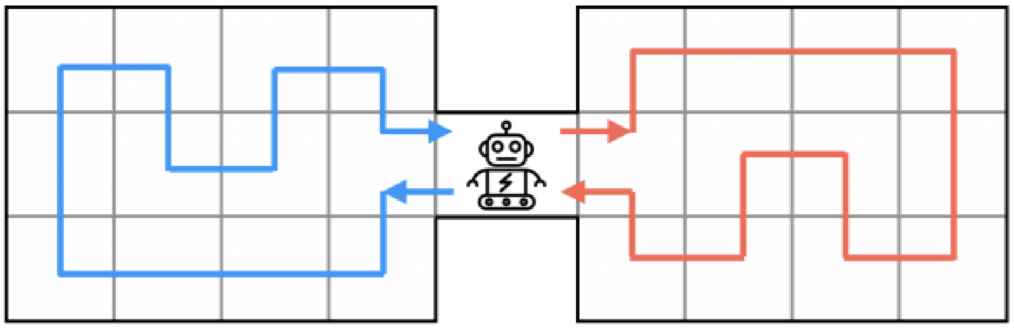


Figure 4.1: Illustrative two-rooms domain from [68]. The agent starts in the middle, colored traces represent optimal strategies to explore the left and the right room

exploration may appear suboptimal when considering a single episode, it achieves uniform exploration when averaged over an infinite number of trials. In summary, Mutti et al. [68] show that non-Markovianity allows us to find the optimal policy when we are in a finite horizon setting. On the other hand, they show that not-Markovianity does not provide any advantage in an infinite horizon setting, but this is impractical in real-world applications.

Notably, the Markovian approach differs significantly from how a human would approach the problem. A human would intentionally decide to explore one room before the other when positioned in the middle. This strategy ensures uniform exploration of the environment in every trial but introduces a non-Markovian element to the decision-making process. This is the reason why they highlighted that a Markovian stochastic policy may excel in exploring an environment over an infinite number of trials but can underperform in any single trial. Such a policy can effectively explore an environment or a class of environments given a sufficiently large number of trials. In certain environments, a Markovian policy frequently encounters the same state with multiple possible actions, enabling exploration of various parts of the environment. However, relying solely on the information of the current state, the Markovian policy resorts to randomization, resulting in an exploration of all available choices on average. When considering exploration in a single trial, the Markovian policy proves suboptimal. Randomization prevents it from avoiding revisiting states it has already explored. In contrast, employing a history-based perspective instead of solely relying on the current state equips the agent with the ability to retain information about the environment. This, in turn, allows the agent to disambiguate its actions and make informed choices. By remembering previously visited states, the agent can opt for different actions, facilitating more effective single-trial exploration and a better understanding of the

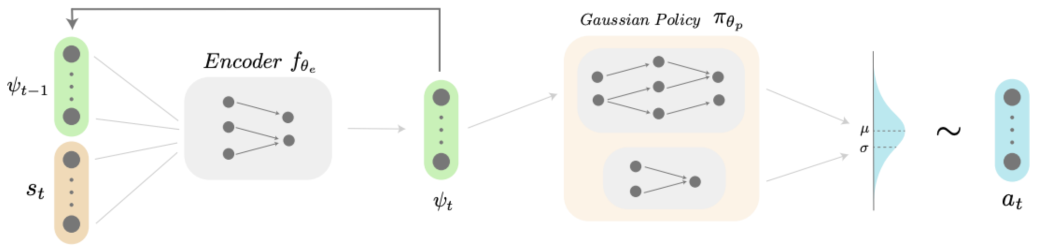


Figure 4.2: Visualization of the policy structure used by MEMENTO from [71]. The recursive aspect is emphasized through the link connecting the output of the history encoder to its subsequent input in the following time step.

environment.

4.4.1 MEMENTO

In this section, we present the policy gradient algorithm *MEmory-based Maximum ENTropy Optimization* (MEMENTO) [71]. The algorithm aims to learn a non-Markovian policy to reach the optimal performance over a multi-environment setting. By definition, in this case, the trajectory history is available and they use a function to provide a compact representation to analyze the history. In particular, we are concerned with a parametric architecture denoted as $\pi_\theta, \theta \in \Theta \subseteq \mathbb{R}^q$, which is the composition of two modules, each implemented by distinct neural networks: a *recursive history encoder* characterized by parameters θ_e, and a *Gaussian* policy characterized by parameters θ_p, with θ being the union of θ_e and θ_p. Figure 4.2 provides an illustration of the entire architecture. The recursive history encoder, shown on the left side of Figure 4.2, serves the purpose of obtaining a concise representation ψ_t for the current historical information h_t. Specifically, at each time step t, the previous representation ψ_{t-1} and the current state s_t are transformed into a new historical representation ψ_t using the following recursive equation:

$$\begin{cases} \psi_0 = 0 \\ \psi_t = f_{\theta_e}\left(s_t, \psi_{t-1}\right), \end{cases} \tag{4.6}$$

where f_{θ_e} represents a multi-layer perceptron parameterized by θ_e. The resulting output from the recursive history encoder is subsequently input into a Gaussian policy, depicted on the right side of Figure 4.2, which defines the strategy for selecting actions. The MEMENTO algorithm could be seen as an extension of MEPol [33], described in Section 4.2, in which the employed architecture follows what is described in Figure 4.2. Indeed, in MEMENTO the MSE objective is pursued using the entropy estimator (Equation 2.22), as in MEPol.

Algorithm 6 MEMENTO

```
**Input**: Horizon $T$, number of batches $N$, batch-size $B$, trust-region threshold $\delta$, learning rate $\alpha$, nearest neighbors $k$, max off-policy iterations $\sum_{\max}$
Initialize $\theta$
while not converged do
    Draw $N$ batches of $B$ trajectories of length $T$
    $\theta' \leftarrow \theta$, $\sum_{\max} \leftarrow 0$
    while $D_{KL}\left(\hat{d}_T(\theta) \| \hat{d}_T(\theta')\right) \leq \delta \& \sum_{epoch} \leq \sum_{max}$ do
        $\theta' = \theta' + \alpha \nabla_{\theta'} \hat{H}_k\left(\hat{d}_T(\theta') \mid \hat{d}_T(\theta)\right)$
        $\sum_{epoch} \leftarrow \sum_{epoch} + 1$
    end while
    $\theta \leftarrow \theta'$
end while
return Recursive history encoder $f_{\theta_e}$, Gaussian policy $\pi_{\theta_p}$
```

4.4.2 MEMENTO in Multiple Environments

In their work, Maldini and Mutti [71], show the performance of the MEMENTO algorithm compared to αMEPol and MEPol, in a multi-environment setting. In particular, they show that non-Markovian policy outperforms MEPol and αMEPol in a small or large set of environments and that non-Markovian policy can efficiently deal with the *identification-exploitation dilemma*, i.e., deciding when to gather information about the environment, and when to exploit this information with the best environment-specific strategy.

They test the above-mentioned algorithms in a small class of environments, named *GridWorld with Slope* [70], which is composed of four rooms connected by four corridors. The non-Markovian policy learned by MEMENTO outperforms the other algorithms in both cases: when the sampling probability of the environment remains the same in the training and testing time and in the unfavorable configuration, in which the most sampled environment during the training phase becomes less probable during the testing phase. Additionally, they also exploited a large class of environments, and, as previously, MEMENTO outperforms the Markovian counterparts.

In gridworlds [70] with different transition dynamics, the act of identifying the environment with the aim of leveraging the most suitable environment-specific strategy can often entail a substantial cost in terms of initial performance during pre-training. This gives rise to a challenging identification problem that we refer to as the "*identification-exploitation dilemma*", i.e., understanding when it is convenient to seek the environment identification rather than directly optimizing the pre-training objective under uncertainty.

A Markovian policy represents one extreme end of the spectrum, consistently prioritizing the exploitation of interactions to augment entropy without precise knowledge about the particular environment. On the contrary, non-Markovian policy can deal with environmental uncertainty, making decisions about whether to exploit existing knowledge or gather additional information while taking into account the expected cost associated with the identification of the environment. Indeed, history can help to identify correctly the current environment. In the end, the proposed non-Markovian algorithm MEMENTO can help in a multi-environment setting.

Chapter 5

Spray Painting Tasks

In this chapter, we show the various approaches that have been explored in the context of solving the spray painting problem in the literature. We illustrate the spray painting task and the diverse methodologies employed for its resolution. In Section 5.1, we discuss the formalization of this challenge as an optimization problem, where the objective is to discover an optimal trajectory that minimizes a predefined cost function. Subsequently, in Section 5.2, we present a recent study that introduces a neural network-based approach for trajectory generation. Finally, we examine recent advancements employing reinforcement learning techniques.

5.1 Spray Painting as an Optimization Problem

In recent years, the field of robotic spray painting has garnered significant attention both in academic research and industrial applications. Robotic spray painting presents an automated solution for the painting process in the industry, offering the potential to enhance quality and safety while reducing costs.

The core problem is formulated as follows: generating a set of painting trajectories that achieves a target paint thickness on a given surface. Notably, this challenge has been rigorously formalized in works such as [22] and [82] as an optimization problem.

Gleeson et al. [22] introduced an optimization framework tailored to capture the real-world behaviour of spray-painting robots. Furthermore, they devised a method for generating feasible trajectories to guide the painting process. Their approach includes the development of a projection model to faithfully emulate the physical paint deposition process on surfaces. Leveraging this model, they could create trajectories that satisfy various constraints, including paint thickness requirements. The methodology involves refining pre-existing trajectories while minimizing deviations from the target paint thickness.

Subsequently, Gleeson et al. [82] enhanced their prior work by presenting an algorithm capable of addressing paint thickness optimization along with additional constraints, such as the inter-sweep distances, maximum segment lengths, and more. Moreover, their algorithm can generate trajectories with minimal or no intersections. Similar to their previous work, they cast this problem as a continuous non-linear optimization task. The algorithm's primary objective is to minimize a comprehensive cost function that encapsulates various trajectory attributes and requirements.

The main limitations of this type of approach relate to object-conditioned optimization, meaning that an entire optimization process is required for every specific target surface, and this has high computational costs.

5.2 Spray Painting and Deep Learning

The main goal of [20] is the robotic spray painting task involving 3D objects. Tiboni et al. [20] aims to introduce a novel 3D deep learning approach that deals with this task and operates on unstructured input - point clouds - and mixed-structure output spaces - unordered sets of painting strokes.

Nowadays, despite the growing interest in robotic spray painting and its relevance in product manufacturing, this task remains a largely understudied topic. One of the main reasons is the lack of a sufficient amount of objects and annotated painting trajectories. Therefore, [20] offers the first supervised dataset, with complex 3D objects for learning robotic spray painting path generators, such as cuboids, windows, shelves, etc. Then, they formalize the task as a 3D deep-learning problem, building a pipeline which takes as input a set of point clouds - to describe the 3D object - and provides a set of poses as output. Their approach is able to learn a representation of the 3D object, capturing its properties and details. Through this representation, they can predict path segments to be concatenated, building a long-horizon path to paint the 3D object. Unlike other heuristic methods, this method is not object-specific but, instead, it can be applied to any 3D object. Additionally, this data-driven approach needs to learn only from a set of human demonstrations but it can be applied to any object, regardless of its complexity provided that its characteristics are covered by the training set.

In the end, they provide a method for making a qualitative comparison between strokes, through the paint coverage, that is the percentage of surface covered, and the Pose-wise Chamfer Distance, which is a metric to compare the predicted and the ground truth paths. With this method, they are able to achieve a good percentage of paint coverage with respect to ground truth.

5.3 Spray Painting and Reinforcement Learning

In the existing literature, there is a scarcity of research addressing the intersection of spray painting and the application of RL to path generation for robotic spray painting. The pioneering work in this domain was introduced by Kiemel et al. [21], who presented a framework for optimizing industrial spray painting in a 2D setting. Their approach to approximating the spray painting process encompasses several key steps. Firstly, they model the spray gun as an array of rays and compute intersections between these rays and the pixels on surfaces, thereby determining paint deposition based on a beta distribution. They employ the PPO algorithm [32]. The state representation is composed of the spray gun's position (i.e., the intersection of the spray arrays) and the ratio of unpainted pixels to total pixels. Actions are discrete, signifying movements in the up, down, left, or right directions. The reward is exclusively based on the number of newly painted pixels. Their results show that the generated coverage path for painting a car door exhibits performance comparable to a manually implemented zigzag baseline. Nonetheless, this method has several limitations, such as relying on simplified models, assuming quasi-planar surfaces and other approximations, particularly in considering a discrete movement of the spray gun. Additionally, it does not consider uniform coverage, a vital aspect in industrial painting, as the generated paths often exhibit excessive twists and overlapping. In this thesis, we propose a novel methodology to address the limitations of the approach mentioned above.

5.3.1 Reinforcement Learning for CPP

Although RL applications in the context of spray painting are limited, there exists related work focusing on Complete Coverage Path Planning (CCPP) problems, including scenarios like robot cleaning [10], [11].

In [10], Lakshmanan et al. presented a method for solving CCPP using Deep RL, employing the hTetro reconfigurable robot. This robot consists of four blocks connected by three hinges, allowing for freedom of movement and seven different morphological configurations. The robot can change its configuration through transformations. These are not cost-free since the robot needs to consume energy to change its configuration. Additionally, the robot can rotate by maintaining the same configuration and there is a cost proportional to the energy consumption. Therefore, the goal is not only to optimize the path but also to minimize energy consumption, considering that each action, be it a move, transformation, or rotation, consumes energy. Their approach operates in discrete action and state spaces, where states are transformed into images before being fed to a neural network composed of a Convolutional Neural Network (CNN) [83] and a Long Short-Term Memory (LSTM) [84] network. The reward function encourages the agent to cover new cells while penalizing energy consumption and illegal moves toward obstacles. The results indicate lower cost - based on path length and energy consumption - and time efficiency compared to baseline methods such as zigzag or spiral pattern generators, as well as greedy search and genetic algorithms [85]. However, the proposed method can lead to suboptimal paths (in terms of length) in certain cases. For example, if the optimal path (in terms of length) requires a large number of transformations, the algorithms will generate a less expensive path (in terms of energy consumption), since most transformations are as expensive or more expensive than a simple translation, which could be necessary for the optimal path. In summary, the method proposed in [10] generates a trajectory minimizing energy consumption, which refers not only to moves but also to rotations and transformations and the generated trajectory may not be optimal if we consider only the path length. However, the proposed method works well also in the real world in which the generated path for complete coverage presents a few recovered areas.

In another study by Moon et al., [11], a discrete setting is also studied. Their state representation encompasses the observation of eight surrounding tiles around the robot. The reward function encourages covering new cells and discourages unnecessary robot rotations. To prevent the robot from getting stuck in an area with all cells cleaned, they implement a heuristic to find the nearest uncleaned tile. While their approach outperforms random and zigzag policies, it has been primarily tested in simulations and not validated in real-world settings.

Overall, the algorithms discussed in this chapter are tailored to specific environments. With this thesis, we aim to generalize the application of RL in solving CPP problems.

Chapter 6

Methodology

In this chapter, we provide a comprehensive presentation of the learning pipeline devised to address the challenges posed by robotic spray painting tasks. Our approach involves a two-step methodology, comprising initial pre-training and subsequent fine-tuning. Firstly, in Section 6.1, we elucidate the motivation and foundational concepts that underlie our methodology. This section aims to provide a clear understanding of the key principles and design choices of our proposed approach. Subsequently, in Section 6.2, we provide the mathematical details of our method, illustrating the formulation of both the pre-training and fine-tuning rewards. Later, we present an overview of the process for selecting the key parameters that characterize the algorithm, and we provide insights into the implementation details. In conclusion, we illustrate how we implement the spray painting simulator, which plays a key role in our work since it has been used for evaluating all the generated trajectories.

6.1 Overview

Tasks within the domain of CPP problems, such as robotic cleaning and spray painting, present a multitude of challenges. The generated trajectory must not only ensure optimal coverage of the underlying environment but also minimize unnecessary energy consumption by avoiding obstacles and overlapping areas, for example. Moreover, the path generation needs to adapt to the specific geometries of the target objects.

To address the task of autonomous robotic path generation, we leverage RL, a framework proven to successfully solve diverse problems, including robotic manipulation, video games, and locomotion. However, RL algorithms often suffer from limitations in generalization and sample efficiency, requiring a substantial number of agent-environment interactions to converge to successful yet specific policies.

Specifically, we consider the spray painting task in the setting of CPP problems, which plays a pivotal role in industrial manufacturing. The main problem, when addressing solutions for this task using RL, is that policy training must be performed on a specific target object. This setting is essential to generate satisfactory solutions tailored to that particular object.

To overcome the bottleneck, related to object-specific training, we introduce a first prior pre-training phase. This additional step serves to accelerate the object-specific training process, performing a pre-training in which the agent is able to collect valuable knowledge about multiple target shapes. This step is performed before the fine-tuning step, in which the policy specializes on a single shape.

In the pre-training phase, we employ algorithms belonging to the URL framework. Specifically, we adopted entropy-based algorithms, in which the goal is to maximize the entropy within the agent states. Since the entropy defined in Equation 2.22 is proportional to the distances between particles, i.e., states visited by the agent, the entropy-maximization objective is close to our final coverage objective. Moreover, the incorporation of intrinsic rewards enables us to conduct pre-training without the need for the computationally intensive painting simulator tool, thereby avoiding a significant slowdown in this prior phase.

6.2 The Method

6.2.1 Problem Formulation

In this section, we will outline the problem formulation for our RL approach, building upon the concepts introduced in Section 2.1.1.

In the context of our spray painting scenario, a state s encapsulates the agent's position at time step t and low-level environmental features representing the target shape $\mathcal{M}$. This state information is used as input to the policy π_θ to choose actions a and transition to new states s_{t+1}. The sequence of states collected within the same episode constitutes the trajectory $\tau = (s_0, s_1, \dots, s_{T-1}, s_T)$. The problem we address in this work can be stated as follows: given a target shape $\mathcal{M}$, the policy π_θ must generate the best trajectory τ, in order to maximize the expected discounted return $J(\pi)$.

6.2.2 Method Overview

To speed up the object-specific training process and enhance performance, particularly with respect to the final reward, we consider a two-step pipeline comprising prior pre-training and final fine-tuning. The fine-tuning phase has the goal of maximizing the reward related to paint coverage and paint thickness variations on the target shape. These evaluations are carried out using the spray painter simulator discussed in Section 6.3.2.

For the pre-training phase, we leverage the concept of *intrinsic reward* from the URL framework, which allows us to avoid interactions with the computationally expensive painter simulator. In the following sections, we show the details of both the pre-training and fine-tuning steps.

The pre-training process This prior pre-training is based on the MEPol algorithm described in Section 4.2, belonging to the URL algorithms with a state-entropy maximization objective. As mentioned above, the employment of this class of algorithms is due to the nature of the entropy-maximization objective which is related to our coverage-maximization goal.

In our case, the pre-training process for the policy, π_θ, starts with sampling a set of B target shapes $\mathcal{M}_i$, with which the agent interacts. During this interaction, the agent generates a set of B trajectories, each with a length of T and in a different environment $\mathcal{M}_i$. The introduction of the parameter B is also motivated by the variance and the bias problems of the entropy estimator defined in Equation 2.22. To obtain a reliable estimate, a substantial number of samples is required. We provide further details in Section 6.3.

The entropy of each trajectory, denoted as $H_{\tau i}$, is computed using Equation 2.22. We underline that the entropy is computed considering only the agent position information of the state s, and the entropy is computed considering the states within the same trajectory. Subsequently, the policy gradient is computed as described in Section 2.1.11. The policy update is performed off-policy, following a similar approach to [33], [70]. This process continues until the updated policy, π', falls within a trust-region bound [31]. The trust-region bound is computed as the Kullback-Leibler (KL) divergence between π_θ (the current policy) and the sampling policy. Alternatively, the process may conclude if the number of off-policy iterations exceeds a predefined threshold. The overall structure of the algorithm resembles that of Algorithm 4, but it involves processing multiple trajectories over different target shapes $\mathcal{M}_i$ instead of a single trajectory. The other difference lies in the state structure, which also includes the target shape information, in addition to the agent position at step t. In particular, in the experiments, we mainly deal with rectangular shapes rotated and with different dimensions. In our experiments, we use a state representation denoted as s, which is defined as follows:

$$s = \begin{bmatrix} (x, y) \\ (x_{\min}, y_{\min}) \\ (x_{\max}, y_{\max}) \\ \sin(\alpha) \\ \cos(\alpha) \end{bmatrix} \tag{6.1}$$

Here, each element of the state vector is associated with specific parameters:

- (x, y) represents the Cartesian coordinates, defining the agent's spatial position within the environment,
- $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ define the minimum and maximum extents of a rectangular bounding box, offering insights into the agent's operational space,
- $\sin(\alpha)$ and $\cos(\alpha)$ capture the sine and cosine of the rotation angle α, respectively, characterizing the orientation of the rectangular bounding box.

This state representation describes the essential spatial and geometric parameters, enabling the agent to effectively perceive and navigate within the target shape.

The other difference with respect to MEPol [33], consists of sampling B target environments at each epoch, and, consequently, B trajectories on each target shape $\mathcal{M}_i$. In the next chapter, we provide a detailed account of the experimental results achieved through training using this approach.

RL Fine-Tuning In the fine-tuning phase of our pipeline, we aim to further enhance the performance of our agent. In this stage, we select a single target shape $\mathcal{M}$, which remains the same throughout the entire finetuning process. Within this fine-tuning step, we use the same state representation defined in 6.1, to encapsulate essential information for the agent's interaction with the target shape.

The primary objective during this phase is to train the agent to maximize a hand-crafted extrinsic reward using the TRPO algorithm [31]. We employ a temporal difference learning method, and at each time step t, the agent receives a reward defined as follows:

$$R(t) = \Delta_{PC}(t) - \mathit{done} \times \sigma^*_{PT} \tag{6.2}$$

Then, in our work, we employ a multi-objective reward function [86], in which we provide feedback related to the increase in the number of newly painted pixels $\Delta_{PC}(t)$ between the current time step t and the previous time step $t-1$. Additionally, we apply a penalty $\sigma^*_{PT}(t)$ proportional to the variations in paint thickness among the painted pixels. This latter penalization is given only in the terminal time step of the episode because, once we have the full trajectory, we are able to evaluate the paint variations over the target shape. This multi-objective reward function allows the agent to strike a balance between coverage and thickness variation. The paint thickness variations are quantified using the coefficient of variation, a metric that measures the relative variability used in our setting to measure paint thickness variations within the painted region, defined as:

$$\sigma^*_{PT} = \frac{\sigma_P}{|\mu_P|} \tag{6.3}$$

In this equation, σ_P stays for the standard deviation computed considering only the painted pixels P, while μ_P is the average paint thickness among the painted pixels P. Our goal is to reduce the paint variations on the target surface, and then reduce the variation of coefficient value.

This fine-tuning step is pivotal in refining the agent's performance and optimizing its painting strategy, ultimately contributing to achieving the desired painting results tailored to a specific target.

6.3 Key Design Choices

Within our pipeline, we encounter a multitude of parameters and implementation options that demand careful selection in alignment with our ultimate goals. In the subsequent sections, we will elucidate our process for determining the key parameters and implementation choices.

6.3.1 Pre-training

In this phase, aimed at preparing the policy before encountering the last phase, we have to carefully choose different hyperparameters in accordance with our final painting objectives. In the next paragraphs, we offer an overview of the hyperparameters to be chosen and how we should set their values.

Batch size The original MEPol [33] objective is to maximize the entropy within a multiple set of trajectories on the same target shape. This goal slightly differs from ours. Indeed, we aim to maximize the coverage, and, subsequently, the entropy within the same stroke. This suggests shifting the original MEPol focus from maximizing the entropy of multiple trajectories on the same shape to considering a single stroke. Then, in our scenario, a batch should consider a single trajectory for computing the entropy value.

K-th Neighbour Another important hyperparameter to be chosen is related to which neighbour we have to consider for computing distances to then compute entropy as in Equation 2.22. Given that we consider a single stroke per shape, we have to consider that our goal is to stretch our stroke as much as possible until it reaches the target shape boundaries or previously visited regions. This intuition makes us consider not choosing a neighbour too far away, since we may lose the possibility to stretch trajectory.

Number of Batches and KL-threshold The parameter indicating the number of batches (B) plays an important role in our setting. The entropy estimator of Equation 2.22 suffers from variance and bias. The estimator is proven to be asymptotically unbiased [45], but in our scenario we work with a finite number of samples. Moreover, if the distance between the sampling distribution f and the target f' grows large, a high variance might negatively affect the estimation, as stated in [68]. For this reason, to have a reliable entropy estimation, we average the entropy values computed considering different batches.

Additionally, the KL-divergence threshold in algorithms such as TRPO [32] and PPO [31] is used to quantify the difference between the new policy that the algorithm wants to update and the old policy that was used to collect the current batch of data. The threshold is a way to ensure that the policy update doesn't introduce drastic changes that could lead to instability or poor performance. If the KL divergence between the old and new policies exceeds the specified threshold, the update is scaled down or rejected to keep the changes within a certain acceptable range. This helps maintain a degree of stability during the learning process. Due to instability problems of the entropy estimator, we use a low value of the KL-Divergence threshold to prevent wrong policy updates and ensure that the learning process is more controlled and stable.

Markovianty vs Non-Markovianity Another important choice to be addressed in the pretraining setting is regarding whether or not to use a history-based policy. Practically speaking, the use of history can help the agent to identify which is the underlying target shape. The use of non-Markovian policies showed remarkable results in the identification of the environment sampled from a finite set [71]. However, in our setting, we already provide the information related to the target shape and we consider the history information redundant. Moreover, as explained in the next chapter, we generate target shapes every time we need to sample a trajectory. This means that we sample shapes from an infinite set of shapes and non-Markovian policies serve to identify the underlying target shape sampled from a finite set of shapes.

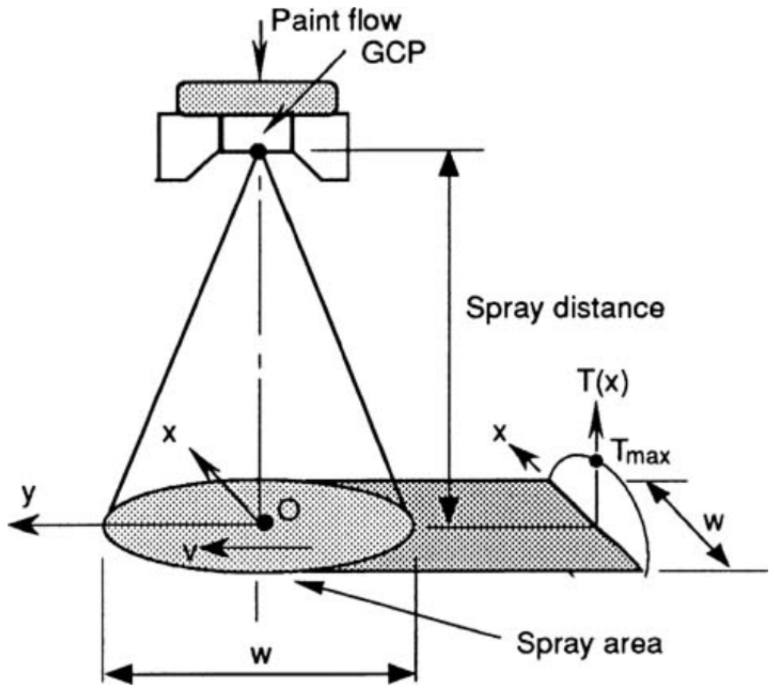


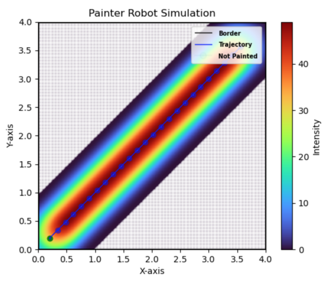
Figure 6.1: Spray gun and coordinate system (from [87]).

6.3.2 Painting Simulator

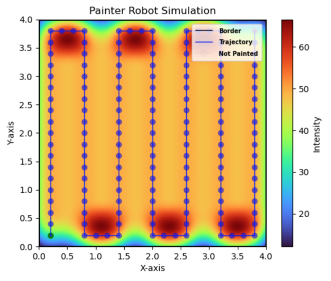
The painter simulator is a computational tool designed to simulate the behaviour of a robotic painter. It operates by modelling the paint deposition process on a pixel map as the robot follows a given trajectory. To model the behaviour of a spray painter, the trajectory is divided into points at which the paint spray occurs, which we define as *spray points.* The simulator calculates the thickness of paint deposited on each pixel of the target environment, considering the distance between the pixel target points and pixel *spray points.* The paint thickness is computed as in [87], following a parabolic distribution:

$$T(x) = \frac{16}{3\pi w^4}\frac{Q_O}{v}\left(w^2 - 4x^2\right)^{3/2} = \frac{16}{3\pi w}\frac{Q_O}{v}\left(1 - \frac{4x^2}{w^2}\right)^{3/2}, \quad -w/2 \le x \le w/2. \tag{6.4}$$

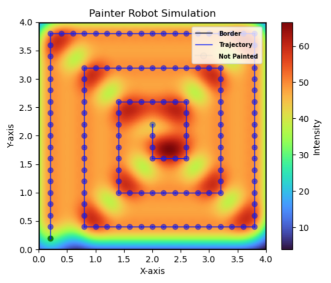
This equation models the paint thickness distribution on a flat surface as in Figure 6.1. In the equation, x is the distance between the pixel and the spray point, w represents the diameter of the circular spray area, Q_0 is the paint flux, the amount



(a) Simple line trajectory



(b) Raster pattern



(c) Squared spiral

Figure 6.2: Painter simulator examples.

of paint sprayed by the painter, and v is the moving velocity of the spray gun. The pixel map is then printed as an intensity map. In Figure 6.2, we show some examples of the painter simulator outcomes.

The key advantage of this simulator lies in its ability to visualize and analyze the paint deposition process, allowing for the evaluation of painting strategies. However, the simulator has some limitations. It simplifies the real-world painting process by assuming a parabolic distribution model, which may not accurately represent all painting scenarios. Additionally, the simulator does not currently consider external factors or the dynamics of the painting environment, such as the electrostatic effect which draws the paints towards the edge of the shapes when the gun paints near the target shape borders [82]. Additionally, the painter simulator accuracy strongly depends on the number of pixels in the map, which is the resolution, but, higher accuracy means higher computational costs. Nevertheless, it is a valuable tool for studying and improving the efficiency of painting robots.

Chapter 7

Experimental Analysis

In this section, we illustrate the experiments conducted with our method. First, we present the results following the pre-training step and showcase the trajectories generated by the pre-trained policy in relation to our final painting objective.

We empirically demonstrate that our entropy-based algorithm, inspired by MEPol [33], provides effective pre-training for CPP problems. We further compare the fine-tuning results starting from scratch and from a pre-trained policy, analyzing the strengths and weaknesses of the latter. To conduct this comparison, we design the following experiments to test the effectiveness of our pipeline in terms of coverage and smooth paint thickness:

- **Pre-training on a Single Shape and Fine-tuning on a Single Shape**: in this case, pre-training is performed considering a single target shape, and fine-tuning targets the same shape.

- **Pre-training on Multiple Shapes and Fine-tuning on a Single Shape**: here, pre-training involves various target shapes, including rectangles of different dimensions, rotations, and positions in space. However, the fine-tuning phase focuses on a single target shape, similar to the ones faced during pre-training.

- **Pre-training on Multiple Shapes and Fine-tuning on a Novel Shape**: In this experiment, the pre-training phase is similar to the previous one, but the fine-tuning is conducted on a different shape referred to as *window*. This shape resembles a rectangle but has two vertically disposed holes (see Figure 7.1).

Additionally, we provide a comparison of our final results to the setting inspired by [21], which we refer to as the *baseline*.

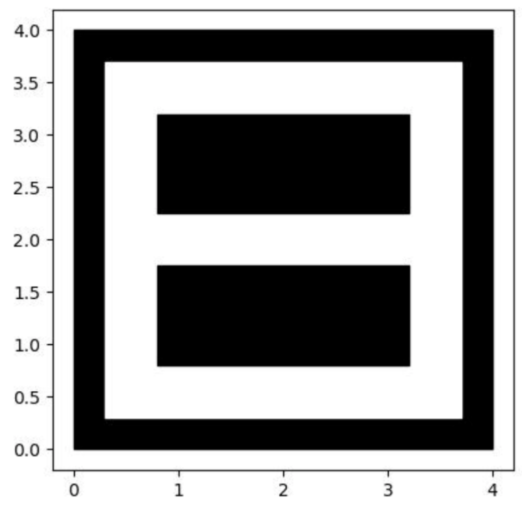


Figure 7.1: Example of a window target surface

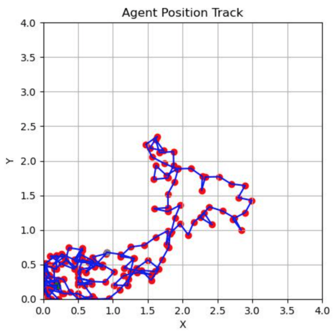
7.1 Pre-training

The primary objective of the pre-training phase is to make the agent learn valuable knowledge about the target shapes. We leverage the *Intrinsic Reward* formulation defined in Equation 2.22 to achieve this objective. In this section, we present the intermediate results, showcasing the trajectories a policy can generate solely after the pre-training.

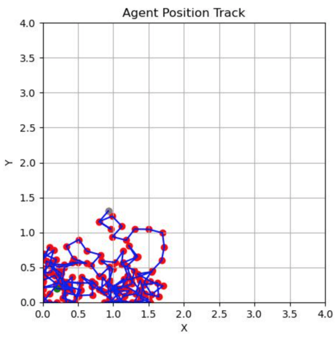
7.1.1 Pre-training on a Single Shape

In this scenario, the agent undergoes training employing a single target shape. In practical terms, the agent's objective is to maximize entropy by increasing distances between k-neighbours. Since the agent explores a single environment, the bounding box and the rotation information are redundant. The detailed hyperparameters for this experiment are outlined in A.1.1.

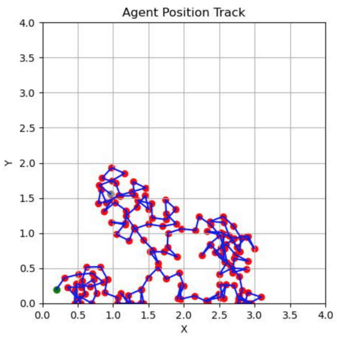
As illustrated in Figure 7.2, the agent, at the end of pre-training, can generate trajectories resembling spirals that already achieve an acceptable coverage of the target surface. Figure 7.2 presents the results for three different seeds. These empirical findings underscore the effectiveness of entropy as a target metric for improving coverage. However, it's noteworthy that sometimes the plotted trajectory doesn't converge toward the centre of the target shape, which is crucial for attaining satisfactory coverage results. To emphasize the impact of pre-training, we compare these results with those of a non-trained policy, which tends to navigate towards boundaries or stay near the starting position, without sufficiently covering



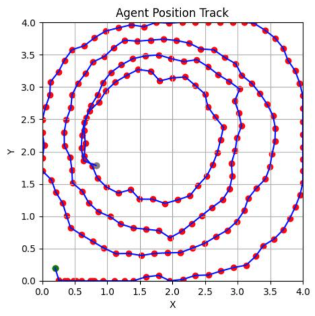
(a)



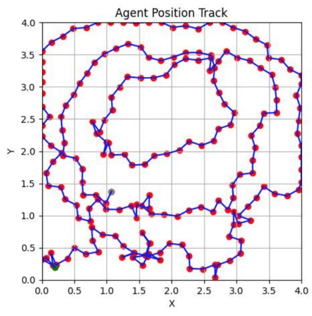
(b)



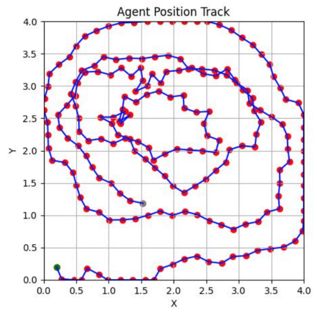
(c)



(d)



(e)



(f)

Figure 7.2: In this figure, we plot the trajectories generated by policies during the fine-tuning phase. In Figure 7.2a, 7.2b, 7.2c we plot trajectories of the non-pre-trained policies, while Figure 7.2d, 7.2e, 7.2f show the generated trajectories at the end of pre-training. The columns represent 3 different seeds.

the surface. The training curve in Figure 7.3 demonstrates quite a stability and relatively fast convergence.

7.1.2 Pre-training on Multiple Shapes

To make the agent explore different shapes, we randomly pick B shapes for each episode. Practically, we generate bounding boxes and rotations whenever we sample a trajectory, giving the agent a wide range of shapes to handle. This choice is intentional since we want the agent to develop a general behaviour without over-specializing on a fixed set of shapes. Unlike before, now the bounding box and rotation details are crucial for understanding each shape. When creating rectangles, we add a rule to ensure they have a minimum area.

Figure 7.4, shows that, as in the previous experiment, the agent learns a policy that, at the end of pre-training, can cover different shapes effectively. In Table

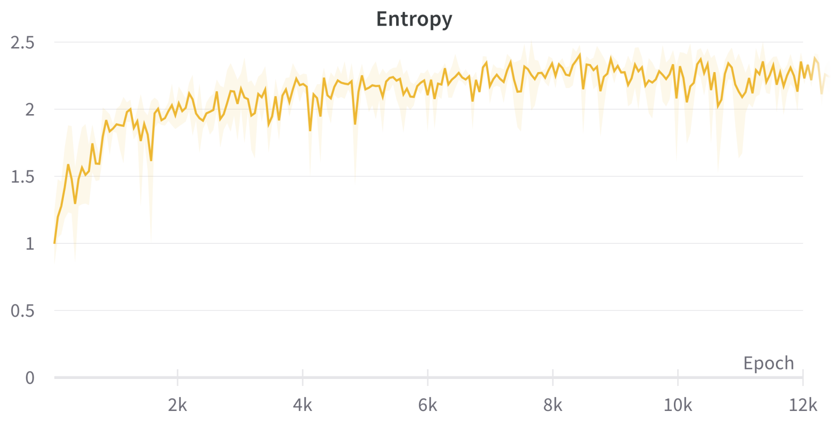


Figure 7.3: Entropy curve during the pre-training on a single shape.

	Horizon (T)	Starting Entropy	Final Entropy
Single Shape	150	1.1 ± 0.3	2.53 ± 0.11
Multiple Shapes	100	0.25 ± 0.01	2.8 ± 0.8

Table 7.1: In this table, we show how the entropy increases using the entropy-based algorithm described in Section 6.2.2. We thus compare the outcomes of both kinds of pre-training, when considering a single target shape and we consider multiple variations.

7.1, we can see a comparison of the starting entropy, i.e., the entropy obtained when the agent is not yet trained, and the maximum average entropy obtained during the pre-training process. Indeed, for the fine-tuning step, we are going to use the best-performing policy we found in this prior step. As you may notice from Table 7.1, we used a shorter horizon T in this latter setting, since the area of the target shapes encountered here is generally lower than the example shown in Section 7.1.1.

7.2 Fine-tuning

In this section, we provide an analysis of the last step of our pipeline, which is fine-tuning. With this step, our goal is to refine the policy behaviour making it able to generate a trajectory for covering the target shape. First, we provide a baseline experiment, which is developed in the simpler case, where the agent policy starts from scratch and receives feedback based on the newly painted pixels. The baseline setting is inspired by [21]. However, the overall setting is different.

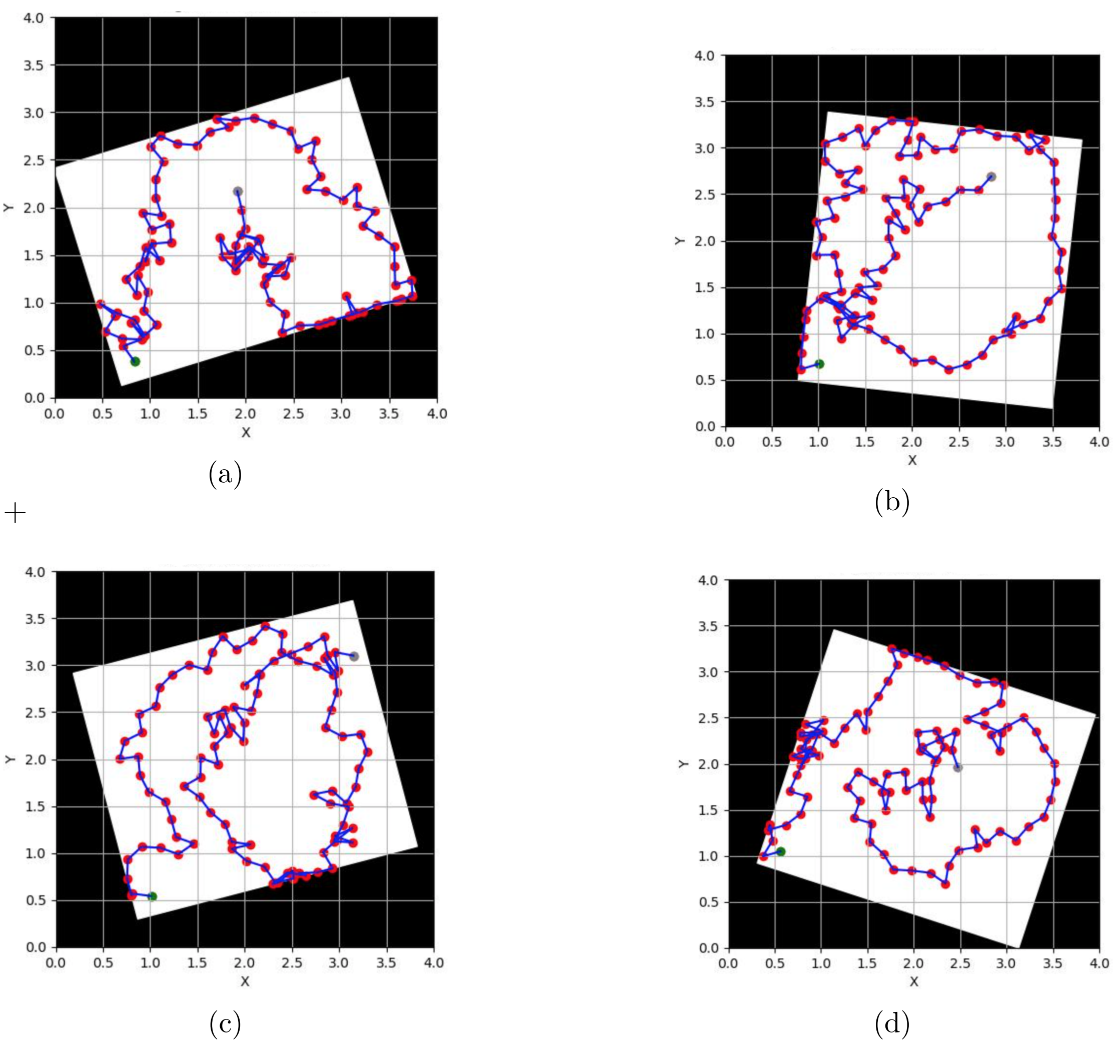


Figure 7.4: Trajectories generated by policies after pre-training on multiple target shapes.

Then, we perform the same experiment using pre-trained policies as described in the previous sections. Since our main goal is to speed up the training process, we compare results after a few episodes when starting the training using random policies and taking advantage of pre-training. Generally, we compare the generated trajectories comparing their paint coverage (PC), which is the percentage of the painted pixels over the total number of pixels within the target shape, and their paint thickness variation using the coefficient of variation (σ^*_{PT}), as defined in Equation 6.3. What we desire from these experiments is to have the highest paint coverage and the lowest paint thickness possible.

7.2.1 Baseline

This first experiment focuses on policy training over a single target shape, in which, the goal is to maximize only the coverage objective. This means that, for our baseline, we do not consider thickness constraints and we use the following reward function:

$$R(t) = \Delta_{PC}(t), \tag{7.1}$$

in which positive feedback is given only according to newly painted pixels $\Delta_{PC}(t)$.

In Figure 7.5, we show the best results for three different seeds in the first 150 episodes. We can observe that the best trajectories are obtained when performing the fine-tuning phase starting from a pre-trained policy as in Section 7.1.1. Indeed, it is clear how the final training process benefits from pre-training. This is also shown in the following plot (see Figure 7.6), where we can observe that the curve of a pre-trained policy starts from a higher level with respect to the fine-tuning with a randomly initialized policy. Generally, an episode during the fine-tuning process lasts for $\sim 9s$, then, a training performed for 150 episodes lasts for $\sim 22\,min$ [1].

7.2.2 Paint Thickness Objective

With respect to the baseline, we introduce the paint thickness minimization objective, this means we want to reduce as much as possible the variations of the paint deposited on the underlying surface. To do that, we introduce the variation of coefficient (σ^*_{PT}) and we use the reward function defined in Equation 6.2. As shown in Table 7.2, the fine-tuning conducted following our pipeline outperforms the case with random initialization. Indeed, we obtain satisfactory results in terms of coverage and thickness, when starting from a pre-trained policy. However, the results in Table 7.2 show that, in a pre-trained setting, the paint thickness objective helps in reducing the thickness variations at the expense of the paint coverage.

Short Fine-tuning

To clarify how our pipeline improves the standard training process, in this section we analyze a very short fine-tuning. We show the best results in time slots, showing the best results obtained after 20, 40 and 60 episodes. We recall that 20 episodes are performed in $\sim$ 3 minutes. From Figure 7.7 and Table 7.3, we note that we need a few episodes to obtain a coverage higher than 80%, if we leverage pre-trained policies.

[1]We refer to a CPU AMD Ryzen 9 3950X: featuring 16 cores, 32 threads, a 72 MB cache, and a maximum boost clock of 4.7 GHz.

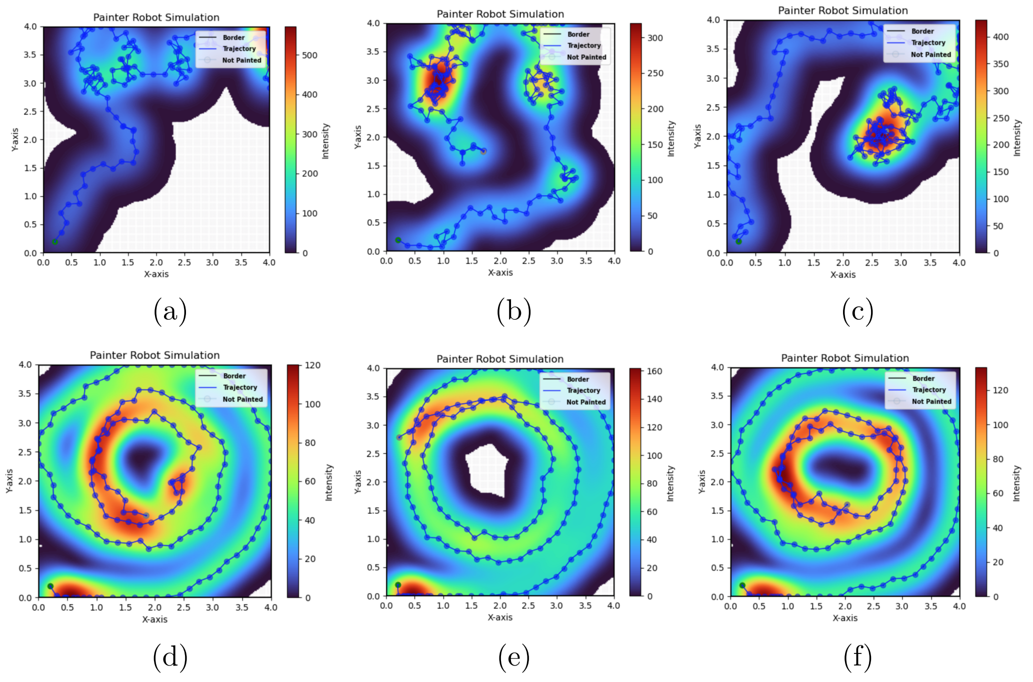


Figure 7.5: In this figure we show fine-tuning outcomes focusing solely on the Paint Coverage objective. Figures 7.5a, 7.5b, and 7.5c depict results from three distinct seeds during the fine-tuning of randomly initialized policies. Conversely, Figures 7.5d, 7.5e, and 7.5f showcase results from fine-tuning pre-trained policies.

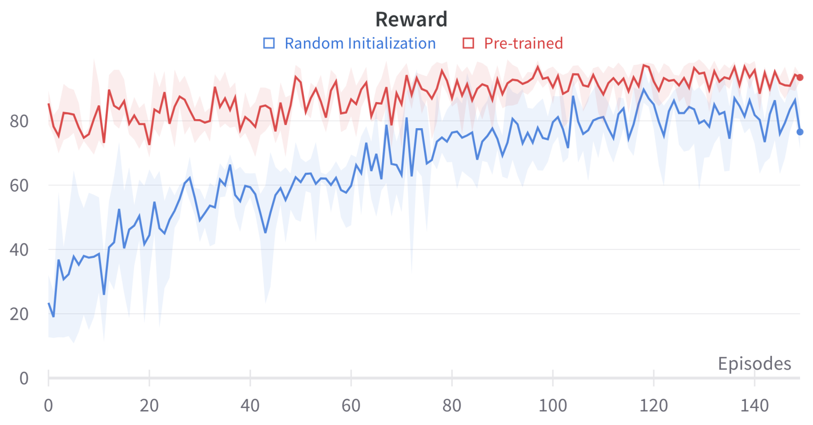


Figure 7.6: Plot of the reward curve in the baseline setting.

	PC objective	PT objective	$PC \uparrow$	$PT(\sigma^*_{PT}) \downarrow$
Random	✓	✗	$(84.4 \pm 12.7)\%$	70.2 ± 39.8
Pre-trained	✓	✗	$(99.5 \pm 0.3)\%$	13.2 ± 2.9
Random	✓	✓	$(43.9 \pm 18.9)\%$	161.7 ± 80.1
Pre-trained	✓	✓	$(99.2 \pm 0.7)\%$	9.45 ± 1.8

Table 7.2: Baseline Setting comparison, considering only the Paint Coverage objective or both, Paint Coverage and Paint Thickness. Here, ↑ indicates that we prefer higher results, while ↓ means that lower is better.

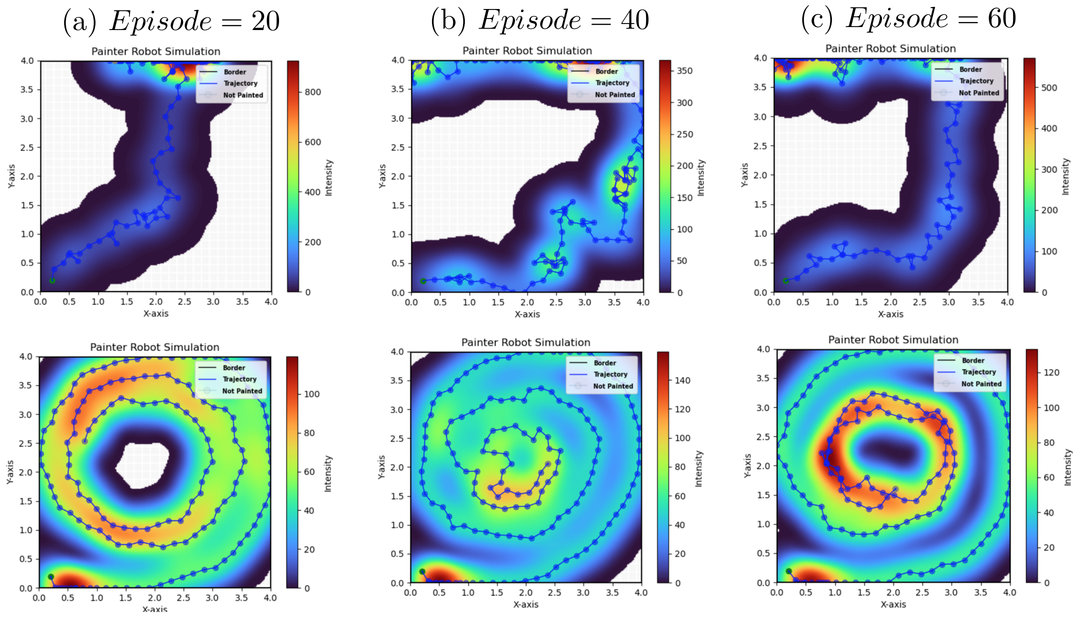


Figure 7.7: Illustrated in these figures are the optimal outcomes achieved after 20, 40, and 60 episodes, corresponding to intervals of approximately 3 minutes each. The top row displays results from fine-tuning using randomly initialized policies, while the bottom row showcases outcomes following prior pre-training.

7.2.3 Fine-tuning after Pre-training over Multiple shapes

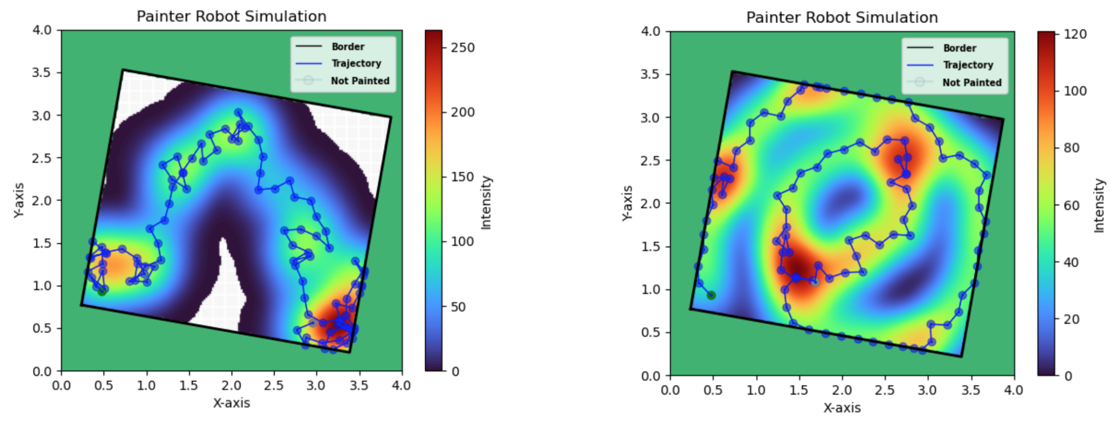
The setting of this experiment is more complex than the ones seen in the previous sections. Indeed, the pre-training phase is conducted considering different target shapes as described in Section 7.1.2. Then, in the fine-tuning step, we consider a single target shape to refine the policy behaviour. Here, we aim to show that pre-trained policies, which face multiple target shapes, are able to refine their behaviour on a single target shape using a few environment interactions. Indeed, as

	$20\,(\sim 3\,min)$	$40\,(\sim 6\,min)$	$60\,(\sim 9\,min)$
Random	$(29.8 \pm 8.7)\%$	$(30.2 \pm 9.2)\%$	$(36.2 \pm 13.1)\%$
Pre-trained	$(97.2 \pm 1.9)\%$	$(97.8 \pm 1.6)\%$	$(98.46 \pm 1.3)\%$

Table 7.3: This table shows the average percentage of coverage we obtain after brief fine-tuning of 20, 40 and 60 episodes.

in the previous sections, we compare the results of this latter phase when employing randomly initialized or pre-trained policies.

Table 7.4 shows that the results of pre-trained policies outperform the best results of fine-tuning random initialized policies after 150 episodes. As expected, the coverage performance when involving pre-trained policies is higher and with a lower variance. This means that employing pre-training guarantees a higher coverage ratio after a few episodes. Moreover, considering the paint thickness objective, we generally obtain smoother trajectories, but it may happen that we sacrifice a few percentage points of coverage ratio.



(a) Fine-tuning results considering random initialization

(b) Fine-tuning results of a pre-trained policy

Figure 7.8: Best results of fine-tuning, when starting from random initialized or from pre-trained policies.

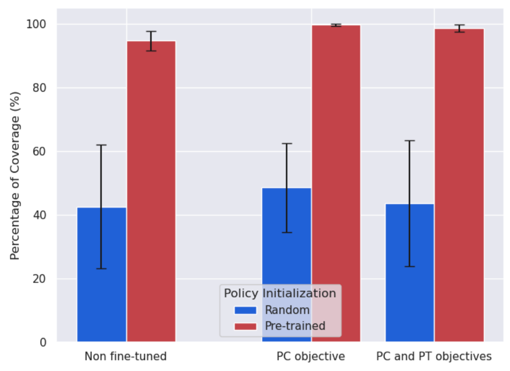
Short Fine-tuning

As in the previous sections, we show the effectiveness of our pipeline by considering the results after a few epochs. Differently from the previous experiment, we show the best results considering only 20 episodes, which requires 1.5 minutes of fine-tuning. As shown in Figure 7.9, pre-training initialization favours fine-tuning

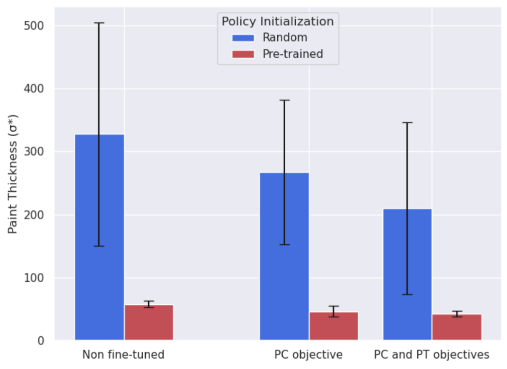
	PC objective	*PT* objective	$PC \uparrow$	$PT(\sigma^*_{PT}) \downarrow$
Random	✓	✗	$(88.7 \pm 9.6)\%$	94.4 ± 34.9
Pre-trained	✓	✗	$(99.7 \pm 0.3)\%$	46.4 ± 9.0
Random	✓	✓	$(52.4 \pm 23.0)\%$	209.8 ± 136.9
Pre-trained	✓	✓	$(98.7 \pm 1.3)\%$	42.4 ± 4.4

Table 7.4: We compare the results obtained when fine-tuning the behaviour on a single target shape, considering only the Paint Coverage objective or both, Paint Coverage and Paint Thickness. The pre-trained policy employed in this setting faced multiple target shapes during pre-training. $\uparrow$ higher the better, and $\downarrow$ lower the better.

process even in the very first episodes. Using pre-trained policies allows us to obtain fine-tuned policies to generate trajectories covering more than the 98% of the target shape with few interactions. Slightly lower coverage values occur when considering the paint thickness objective, this is due to the fine-tuned policy dealing with multi-objectives, so it may learn to sacrifice the paint coverage ratio favouring a smoother paint coverage (see Figure 7.9a). Indeed, in Figure 7.9b, we can observe a slightly lower value and lower variance of paint thickness when considering the paint thickness minimization objective. Visual results are presented in Appendix A.2.3.



(a) Paint Coverage



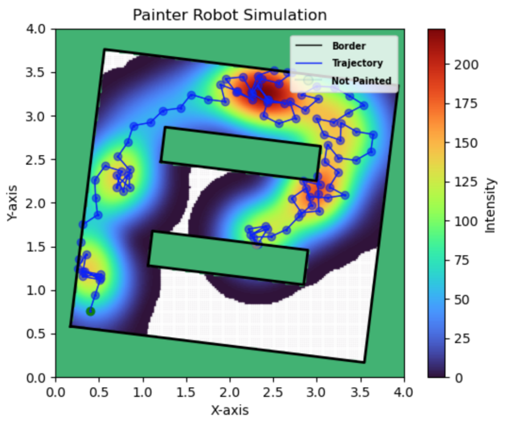
(b) Paint Thickness

Figure 7.9: These bar plots illustrate the coverage achieved by trajectories generated by non-fine-tuned policies (on the left of the figure) compared to those fine-tuned for only 20 episodes (on the right of the figure), requiring a training time of 1.5 minutes. For the paint coverage objective, the higher the better, for pain thickness, the lower the better.

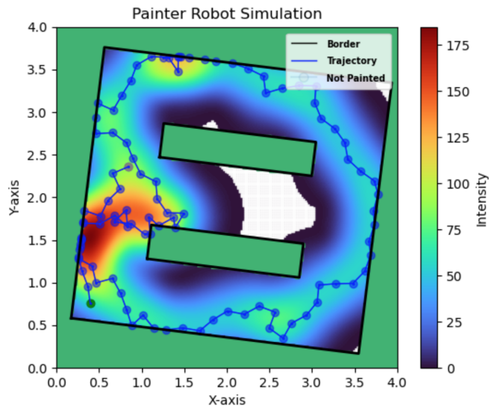
7.2.4 Fine-tuning on a Novel Shape

This experiment introduces a new shape referred to as a *window*. An example of the window shape is illustrated in Figure 7.1.

In this setting, the policy behaviour is fine-tuned on this unfamiliar shape, which was not encountered during the initial pre-training phase. The objective is to demonstrate the adaptability of our pipeline in fine-tuning considering target shapes different from those encountered during pre-training. Throughout this phase, the agent learns to navigate and adapt to the new properties of the target shape, particularly in this case, learning to avoid the holes. Figure 7.10 visually depicts how the refined policy behaviour, especially when leveraging pre-trained policies, achieves superior performance, often surpassing an 85% coverage on the window surface.



(a) Fine-tuning results starting from a random initialized policy



(b) Fine-tuning results starting from a pre-trained policy

Figure 7.10: Results after fine-tuning on a single target shape.

Short Fine-tuning

In Figure 7.12, we compare the results after few epochs of fine-tuning. The initial epochs contribute significantly to the increase in coverage percentage compared to non-fine-tuned policies. Generally, achieving coverage above 85% requires approximately 20 episodes (equivalent to ~ 1.5 minutes) of fine-tuning.

Figure 7.11 further illustrates the evolution of coverage percentage during training. Notably, when employing pre-trained policies, we observe an initial peak in coverage, followed by a temporary decrease. This behaviour may suggest that the agent escapes from a local maximum, subsequently re-learning and discovering a new maximum. For this reason, the best results shown in Figure 7.10 are obtained during the first episodes of fine-tuning.

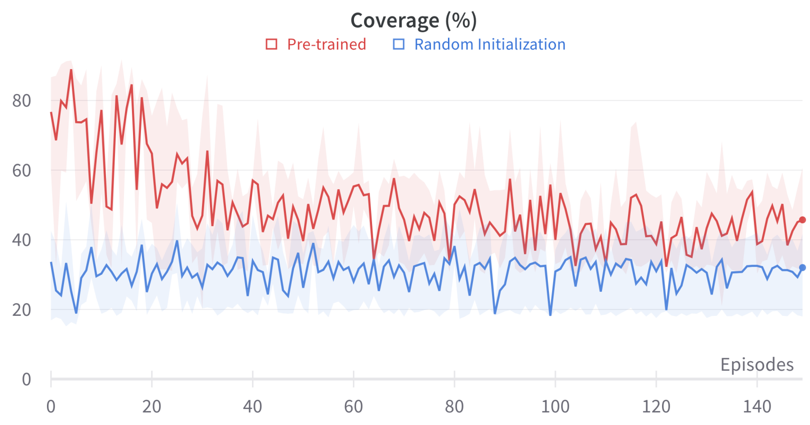
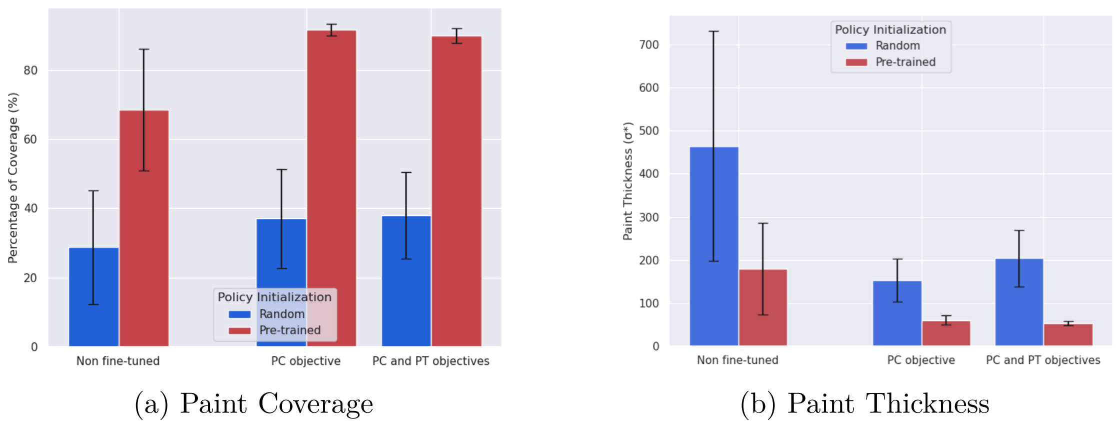


Figure 7.11: The curve represents the coverage ratio during the fine-tuning considering a window as a target policy.



(a) Paint Coverage (b) Paint Thickness

Figure 7.12: These bar plots illustrate the coverage achieved by trajectories generated by non-fine-tuned policies (on the left of the figure) compared to those fine-tuned for only 20 episodes (on the right of the figure), requiring a runtime of 1.5 minutes, considering a window target shape. The bars on the right are further divided into two groups, representing experiments conducted with a focus on paint coverage optimization only and experiments considering both paint coverage and paint thickness objectives. For the paint coverage objective, the higher the better, for pain thickness, the lower the better.

	PC objective	*PT* objective	*PC* ↑	$PT(\sigma^*_{PT})$ ↓
Random	✓	✗	$(59.7 \pm 12.3)\%$	153.1 ± 49.9
Pre-trained	✓	✗	$(91.8 \pm 1.9)\%$	60.2 ± 10.7
Random	✓	✓	$(42.3 \pm 11.0)\%$	204.1 ± 65.5
Pre-trained	✓	✓	$(90.4 \pm 2.3)\%$	53.0 ± 4.5

Table 7.5: We compare the results obtained when fine-tuning the behaviour on a window-like target shape, considering only the Paint Coverage objective or both, Paint Coverage and Paint Thickness. The pre-trained policy employed in this setting faced multiple target shapes during pre-training, but it didn't encounter window-like shapes. ↑ higher the better, and ↓ lower the better.

7.3 Discussion

The application of our pipeline for addressing robotic spray paint leads us to significant new results. We thus report our discussion on the results we obtained.

7.3.1 Pre-training

Employing entropy-based algorithms demonstrated to benefit the final results in our spray painting scenario. Indeed, the agent can generate acceptable trajectories for covering target shapes already after the pre-training. Figures 7.2 and 7.4 show the trajectories generated by pre-trained policies and we can observe that these can sufficiently cover the target shape. Moreover, considering batches of trajectories sampled from different target shapes showed to be a good choice for improving the generalization of the agent. Indeed, at the end of pre-training, it can generate satisfactory results on the different target shapes it faces.

Limitations The pre-training algorithm we provided and inspired by MEPol [33] demonstrated to give satisfactory results. However, the generated trajectories tend to converge to a spiral-like shape. Generally, the best trajectories for painting a surface are similar to raster patterns as shown in Figure 6.2b. Then, in the first few episodes of fine-tuning, the agent learns to refine this sub-optimal behaviour and it is not able to find a new optimum, which may generate raster-like trajectories.

7.3.2 Fine-tuning

Thanks to fine-tuning, the policy can refine its behaviour to generate trajectories tailored to a specific target surface. The results we reported in the previous section achieved high performance after a few episodes of fine-tuning the pre-trained policies. Indeed, the trajectories generated can cover more than the 85% of the target surface. Thanks to pre-training, we can deal with the limitation of object-conditioned training, since we are now able to obtain good results after a few minutes of fine-tuning. These advantages in terms of time are shown in Figures 7.9, 7.12 and in Table 7.3, in which we illustrate that we need only a few minutes of fine-tuning to reach high performance.

Moreover, the introduction of the paint thickness optimization objective, as in Equation 6.3, helps the agent to generate smoother trajectories, even if it needs to trade off with few coverage percentage points.

In the end, we demonstrated how pre-training can benefit fine-tuning of new target shapes. As expected, the performance of this latter case is slightly lower than the other two experiments, but still with a coverage higher than 85% and much better than the random initialized policies.

Limitations As we mentioned above, the fine-tuning phase serves to refine a sub-optimal behaviour and it may reach a new optimal if we consider a long fine-tuning. However, the scope of this work is to obtain good results in a few episodes of fine-tuning. Additionally, there are a few limitations in the setting since the agent can't go out of the target shape boundaries. Indeed, some optimal patterns for covering target surfaces assume the trajectories turning outside the target shape, to reduce paint thickness variations.

The main difference between our setting and human behaviour is that humans remember the previous positions in the trajectories. We think that adding the past history to the state information defined in Equation 6.1 may be a potentially promising direction to help the agent avoid previously seen regions, reducing thickness variations.

Chapter 8

Conclusions

In this work, we present a two-step pipeline for addressing robotic spray painting leveraging URL. The primary objective of this thesis is to tackle the challenge of object-conditioned training, a significant limitation in existing learning methods for spray painting [20], [21]. Object-specific training is crucial for obtaining policies capable of generating trajectories tailored to specific shapes. To overcome this limitation, we introduce prior pre-training to reduce the overall training time.

For pre-training, we employ an entropy-based algorithm inspired by MEPol [33]. The entropy maximization objective aligns with our subsequent coverage goal. We demonstrate that focusing on entropy during pre-training significantly benefits the final fine-tuning process. During the prior pre-training phase, we consider rectangles of various shapes and poses to generalize the agent's behaviour. Consequently, the same pre-trained policies can be fine-tuned by choosing various target shapes.

Our method is tested on rectangles and different shapes that the agent did not encounter during pre-training. During fine-tuning, the agent refines its behaviour for this new shape.

Results indicate that with a low number of fine-tuning episodes, trajectories covering over 85% of the target surface can be achieved using pre-trained policies. Furthermore, our pipeline outperforms the baseline experiment inspired by [21]. The incorporation of a multi-objective reward function aids in optimizing coverage while reducing paint thickness variations, generating smoother trajectories over the underlying target shape.

In summary, this work provides an efficient pipeline, consisting of pre-training and fine-tuning, for addressing robotic spray painting. Pre-trained policies can be used for short fine-tuning periods to efficiently adapt to a target shape.

Future Research While this work represents progress in automating robotic spray painting, challenges remain, particularly in more complex environments.

Simple planar shapes like rectangles and windows were chosen due to the relative simplicity of the associated state space definition (Equation 6.1). Generalization can be improved by incorporating masks to describe the target shape and employing a feature extractor [88] to extract target shape properties.

A promising and unexplored research direction involves extending this work to consider 3D shapes, utilizing point clouds [89] to describe target objects, as demonstrated in [20].

Another potential avenue is the creation of a dataset containing expert-human-made trajectories that optimally cover a target surface. This dataset can be leveraged to refine policy behaviour, incorporating desired thickness information [82]. Considering a desired thickness as in [20] may aid the agent in learning to generate smoother trajectories.

Appendix A

Empirical Analysis: further details

A.1 Pre-training

A.1.1 Pre-training on a Single Shape

	Single Shape
Number of epochs	12k
Horizon (T)	150
N Batches (B)	8
Kl threshold (δ)	10^{-2}
Learning rate (α)	10^{-5}
Max off-policy iters	30
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

Table A.1: Hyperparameters used for pretraining considering a single target shape.

A.1.2 Pre-training on Multiple Shapes

	Multiple Shapes
Number of epochs	12k
Horizon (T)	150
N Batches (B)	8
Kl threshold (δ)	10^{-2}
Learning rate (α)	10^{-5}
Max off-policy iters	30
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

Table A.2: Hyperparameters used for pretraining considering multiple target shapes.

A.2 Fine-tuning

A.2.1 Baseline

	Fine-tuning
Number of Episodes	150
Horizon (T)	100
Batch Size	100
Kl threshold (δ)	10^{-3}
Learning rate (α)	10^{-2}
Discount Factor (γ)	0.995
PC objective	✓
PT objective	✗
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

Table A.3: Hyperparameters used for fine-tuning in the baseline settings.

A.2.2 Baseline and the Paint Thickness Objective

	Fine-tuning
Number of Episodes	150
Horizon (T)	100
Batch Size	100
Kl threshold (δ)	10^{-3}
Learning rate (α)	10^{-2}
Discount Factor (γ)	0.995
PC objective	✓
PT objective	✓
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

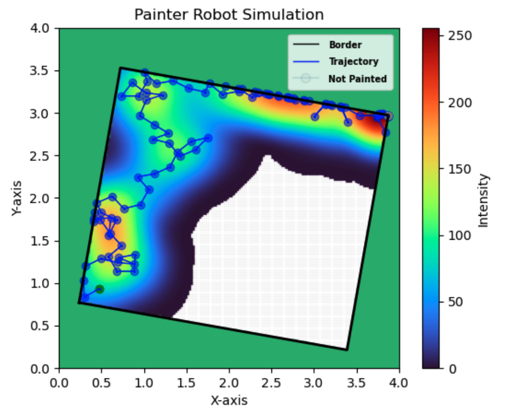
Table A.4: Hyperparameters used for fine-tuning in the baseline settings, considering also the Paint Thickness objective.

A.2.3 Fine-tuning after Pre-training over Multiple Shapes

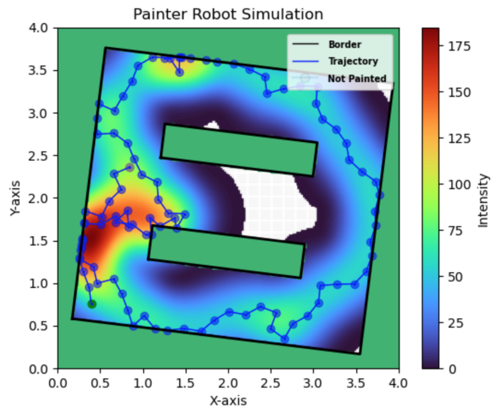
	Fine-tuning
Number of Episodes	150
Horizon (T)	100
Batch Size	100
Kl threshold (δ)	10^{-3}
Learning rate (α)	10^{-2}
Discount Factor (γ)	0.995
PC objective	✓
PT objective	✓
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

Table A.5: Hyperparameters used for fine-tuning on a single target shape.

Brief fine-tuning visual results



(a) Fine-tuning results starting from a random initialized policy



(b) Fine-tuning results starting from a pre-trained policy

Figure A.1: Results after fine-tuning for 20 episodes on a single target shape.

A.2.4 Fine-tuning on a Novel Shape

	Fine-tuning
Number of Episodes	150
Horizon (T)	100
Batch Size	100
Kl threshold (δ)	10^{-3}
Learning rate (α)	10^{-2}
Discount Factor (γ)	0.995
PC objective	✓
PT objective	✓
Number of neighbors (k)	3
Policy hidden layer sizes	(128,128)
Policy hidden layer act. function	ReLU

Table A.6: Hyperparameters used for fine-tuning on a novel target shape.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: `http://incompleteideas.net/book/the-book-2nd.html`.

[2] OpenAI, M. Andrychowicz, B. Baker, *et al.*, "Learning dexterous in-hand manipulation", *CoRR*, vol. abs/1808.00177, 2018. arXiv: `1808.00177`. [Online]. Available: `http://arxiv.org/abs/1808.00177`.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: `http://dx.doi.org/10.1038/nature14236`.

[4] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough", *Artif. Intell.*, vol. 299, p. 103 535, 2021. [Online]. Available: `https://api.semanticscholar.org/CorpusID:236236944`.

[5] M. S. Kakade, "On the sample complexity of reinforcement learning", Ph.D. dissertation, 2003. [Online]. Available: `https://www.ias.informatik.tu-darmstadt.de/uploads/Research/NIPS2006/SK.pdf`.

[6] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning", *CoRR*, vol. abs/2111.09794, 2021. arXiv: `2111.09794`. [Online]. Available: `https://arxiv.org/abs/2111.09794`.

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. DOI: `10.1177/0278364913495721`.

[8] M. Laskin, D. Yarats, H. Liu, *et al.*, "URLB: unsupervised reinforcement learning benchmark", *CoRR*, vol. abs/2110.15191, 2021. arXiv: `2110.15191`. [Online]. Available: `https://arxiv.org/abs/2110.15191`.

[9] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics.", *Robotics Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013. [Online]. Available: `http://dblp.uni-trier.de/db/journals/ras/ras61.html#GalceranC13`.

[10] A. K. Lakshmanan, R. E. Mohan, B. Ramalingam, *et al.*, "Complete coverage path planning using reinforcement learning for tetromino based cleaning and maintenance robot", *Automation in Construction*, vol. 112, p. 103 078, 2020. [Online]. Available: `https://api.semanticscholar.org/CorpusID:214475070`.

[11] W. Moon, B. Park, S. H. Nengroo, T. Kim, and D. Har, *Path planning of cleaning robot with reinforcement learning*, 2022. arXiv: `2208.08211 [cs.RO]`.

[12] E. Hazan, S. Kakade, K. Singh, and A. Van Soest, "Provably efficient maximum entropy exploration", in *International Conference on Machine Learning*, PMLR, 2019, pp. 2681–2691.

[13] A. Antos, C. Szepesvári, and R. Munos, "Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path", *Machine Learning*, vol. 71, pp. 89–129, 2008.

[14] J. Chen and N. Jiang, "Information-theoretic considerations in batch reinforcement learning", in *International Conference on Machine Learning*, PMLR, 2019, pp. 1042–1051.

[15] Y. Jin, Z. Yang, and Z. Wang, "Is pessimism provably efficient for offline rl?", in *International Conference on Machine Learning*, PMLR, 2021, pp. 5084–5096.

[16] D. J. Foster, A. Krishnamurthy, D. Simchi-Levi, and Y. Xu, "Offline reinforcement learning: Fundamental barriers for value function approximation", *arXiv preprint arXiv:2111.10919*, 2021.

[17] W. Zhan, B. Huang, A. Huang, N. Jiang, and J. Lee, "Offline reinforcement learning with realizability and single-policy concentrability", in *Conference on Learning Theory*, PMLR, 2022, pp. 2730–2775.

[18] T. Xie, N. Jiang, H. Wang, C. Xiong, and Y. Bai, *Policy finetuning: Bridging sample-efficient offline and online reinforcement learning*, 2022. arXiv: `2106.04895 [cs.LG]`.

[19] T. Xie, D. J. Foster, Y. Bai, N. Jiang, and S. M. Kakade, "The role of coverage in online reinforcement learning", *arXiv preprint arXiv:2210.04157*, 2022.

[20] G. Tiboni, R. Camoriano, and T. Tommasi, *Paintnet: Unstructured multi-path learning from 3d point clouds for robotic spray painting*, 2023. arXiv: `2211.06930 [cs.RO]`.

[21] J. Kiemel, P. Yang, P. Meißner, and T. Kröger, "Paintrl: Coverage path planning for industrial spray painting with reinforcement learning", Jun. 2019.

[22] D. Gleeson, S. Jakobsson, R. Salman, *et al.*, "Robot spray painting trajectory optimization", in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2020, pp. 1135–1140.

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

[24] R. Bellman, *Dynamic Programming.* Dover Publications, 1957.

[25] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients.", *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008. [Online]. Available: `http://dblp.uni-trier.de/db/journals/nn/nn21.html#PetersS08`.

[26] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. arXiv: `1712.01815 [cs.AI]`.

[27] S. Levine, A. Kumar, G. Tucker, and J. Fu, *Offline reinforcement learning: Tutorial, review, and perspectives on open problems*, 2020. arXiv: `2005.01643 [cs.LG]`.

[28] C. J. Watkins and P. Dayan, "Q-learning", *Machine learning*, vol. 8, pp. 279–292, 1992.

[29] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine learning*, vol. 8, pp. 229–256, 1992.

[30] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics", *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[31] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2017. arXiv: `1502.05477 [cs.LG]`.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: `1707.06347 [cs.LG]`.

[33] M. Mutti, L. Pratissoli, and M. Restelli, *Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate*, 2021. arXiv: `2007.04640 [cs.LG]`.

[34] V. Konda and J. Tsitsiklis, "Actor-critic algorithms", *Advances in neural information processing systems*, vol. 12, 1999.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning", *arXiv preprint arXiv:1509.02971*, 2015.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[37] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients", *Advances in neural information processing systems*, vol. 28, 2015.

[38] T. Haarnoja, A. Zhou, K. Hartikainen, *et al.*, *Soft actor-critic algorithms and applications*, 2019. arXiv: `1812.05905 [cs.LG]`.

[39] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[40] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges", *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[41] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates", in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8662–8668.

[42] S. Höfer, K. Bekris, A. Handa, *et al.*, "Perspectives on sim2real transfer for robotics: A summary of the r: Ss 2020 workshop", *arXiv preprint arXiv:2012.03806*, 2020.

[43] C. E. Shannon, "A mathematical theory of communication", *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948. [Online]. Available: `http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf` (visited on 04/22/2003).

[44] J. Beirlant, E. Dudewicz, L. Gyor, and E. Meulen, "Nonparametric entropy estimation: An overview", *International Journal of Mathematical and Statistical Sciences*, vol. 6, Jan. 1997.

[45] H. Singh, N. Misra, V. Hnizdo, A. Fedorowicz, and E. Demchuk, "Nearest neighbor estimates of entropy", *American journal of mathematical and management sciences*, vol. 23, no. 3-4, pp. 301–321, 2003.

[46] J. Ajgl and M. Šimandl, "Differential entropy estimation by particles", *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 11 991–11 996, 2011.

[47] H.-J. D. Streitberger and K.-F. Dssel, "Automotive paints and coatings", 2008. [Online]. Available: `https://api.semanticscholar.org/CorpusID:138819835`.

[48] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics", *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[49] F. Yasutomi, M. Yamada, and K. Tsukamoto, “Cleaning robot control”, in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, IEEE, 1988, pp. 1839–1841.

[50] P. N. Atkar, A. Greenfield, D. C. Conner, H. Choset, and A. A. Rizzi, “Uniform coverage of automotive surface patches”, *The International Journal of Robotics Research*, vol. 24, no. 11, pp. 883–898, 2005.

[51] H. Choset, “Coverage for robotics–a survey of recent results”, *Annals of mathematics and artificial intelligence*, vol. 31, pp. 113–126, 2001.

[52] J.-C. Latombe, A. Lazanas, and S. Shekhar, “Robot motion planning with uncertainty in control and sensing”, *Artificial Intelligence*, vol. 52, no. 1, pp. 1–47, 1991.

[53] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations.* MIT press, 2005.

[54] H. Choset and P. Pignon, “Coverage path planning: The boustrophedon cellular decomposition”, in *Field and service robotics*, Springer, 1998, pp. 203–209.

[55] C. Jin, A. Krishnamurthy, M. Simchowitz, and T. Yu, *Reward-free exploration for reinforcement learning*, 2020. arXiv: `2002.02794 [cs.LG]`.

[56] Z. Xie, Z. Lin, J. Li, S. Li, and D. Ye, *Pretraining in deep reinforcement learning: A survey*, 2022. arXiv: `2211.03959 [cs.LG]`.

[57] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, *Curiosity-driven exploration by self-supervised prediction*, 2017. arXiv: `1705.05363 [cs.LG]`.

[58] A. Mavor-Parker, K. Young, C. Barry, and L. Griffin, “How to stay curious while avoiding noisy TVs using aleatoric uncertainty estimation”, in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 15 220–15 240. [Online]. Available: `https://proceedings.mlr.press/v162/mavor-parker22a.html`.

[59] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control”, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033. DOI: `10.1109/IROS.2012.6386109`.

[60] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, “Diversity is all you need: Learning skills without a reward function”, *CoRR*, vol. abs/1802.06070, 2018. arXiv: `1802.06070`. [Online]. Available: `http://arxiv.org/abs/1802.06070`.

[61] V. Campos, A. Trott, C. Xiong, R. Socher, X. Giro-i-Nieto, and J. Torres, *Explore, discover and learn: Unsupervised discovery of state-covering skills*, 2020. arXiv: `2002.03647 [cs.LG]`.

[62] S. Park, J. Choi, J. Kim, H. Lee, and G. Kim, *Lipschitz-constrained unsupervised skill discovery*, 2022. arXiv: `2202.00914 [cs.LG]`.

[63] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation", *CoRR*, vol. abs/1606.01868, 2016. arXiv: `1606.01868`. [Online]. Available: `http://arxiv.org/abs/1606.01868`.

[64] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore", *Nature*, vol. 590, no. 7847, pp. 580–586, 2021. DOI: `10.1038/s41586-020-03157-9`. [Online]. Available: `https://doi.org/10.1038%2Fs41586-020-03157-9`.

[65] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, *Exploration by random network distillation*, 2018. arXiv: `1810.12894 [cs.LG]`.

[66] H. Liu and P. Abbeel, *Behavior from the void: Unsupervised active pretraining*, 2021. arXiv: `2103.04551 [cs.LG]`.

[67] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, *Reinforcement learning with prototypical representations*, 2021. arXiv: `2102.11271 [cs.LG]`.

[68] M. Mutti, R. D. Santi, and M. Restelli, *The importance of non-markovianity in maximum state entropy exploration*, 2022. arXiv: `2202.03060 [cs.LG]`.

[69] J. Peters, "Policy gradient methods", *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010.

[70] M. Mutti, M. Mancassola, and M. Restelli, *Unsupervised reinforcement learning in multiple environments*, 2021. arXiv: `2112.08746 [cs.LG]`.

[71] P. Maldini, M. Mutti, R. De Santi, and M. Restelli, "Non-markovian policies for unsupervised reinforcement learning in multiple environments", in *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022.

[72] Z. D. Guo, M. G. Azar, A. Saade, *et al.*, "Geometric entropic exploration", *arXiv preprint arXiv:2101.02055*, 2021.

[73] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov, "Efficient exploration via state marginal matching", *arXiv preprint arXiv:1906.05274*, 2019.

[74] M. Mutti and M. Restelli, "An intrinsically-motivated approach for learning highly exploring and fast mixing policies", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5232–5239.

[75] Y. Seo, L. Chen, J. Shin, H. Lee, P. Abbeel, and K. Lee, “State entropy maximization with random encoders for efficient exploration”, in *International Conference on Machine Learning*, PMLR, 2021, pp. 9443–9454.

[76] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov, *Efficient exploration via state marginal matching*, 2020. arXiv: `1906.05274 [cs.LG]`.

[77] J. Zhang, *Gradient descent based optimization algorithms for deep learning models training*, 2019. arXiv: `1903.03614 [cs.LG]`.

[78] A. M. Metelli, M. Papini, F. Faccio, and M. Restelli, *Policy optimization via importance sampling*, 2018. arXiv: `1809.06098 [cs.LG]`.

[79] S. Parisi, V. Dean, D. Pathak, and A. Gupta, *Interesting object, curious agent: Learning task-agnostic exploration*, 2021. arXiv: `2111.13119 [cs.LG]`.

[80] J. Rajendran, R. Lewis, V. Veeriah, H. Lee, and S. Singh, *How should an agent practice?*, 2019. arXiv: `1912.07045 [cs.AI]`.

[81] R. T. Rockafellar, S. Uryasev, *et al.*, “Optimization of conditional value-at-risk”, *Journal of risk*, vol. 2, pp. 21–42, 2000.

[82] D. Gleeson, S. Jakobsson, R. Salman, *et al.*, “Generating optimized trajectories for robotic spray painting”, *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1380–1391, 2022.

[83] K. O’Shea and R. Nash, “An introduction to convolutional neural networks”, *arXiv preprint arXiv:1511.08458*, 2015.

[84] J. Cheng, L. Dong, and M. Lapata, *Long short-term memory-networks for machine reading*, 2016. arXiv: `1601.06733 [cs.CL]`.

[85] A. V. Le, P.-C. Ku, T. Than Tun, N. Huu Khanh Nhan, Y. Shi, and R. E. Mohan, “Realization energy optimization of complete path planning in differential drive based self-reconfigurable floor cleaning robot”, *Energies*, vol. 12, no. 6, p. 1136, 2019.

[86] C. F. Hayes, R. Rădulescu, E. Bargiacchi, *et al.*, “A practical guide to multi-objective reinforcement learning and planning”, *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, 2022.

[87] M. S. Arikan and T. Balkan, “Process simulation and paint thickness measurement for robotic spray painting”, *CIRP Annals*, vol. 50, no. 1, pp. 291–294, 2001.

[88] W. K. Mutlag, S. K. Ali, Z. M. Aydam, and B. H. Taher, “Feature extraction methods: A review”, in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1591, 2020, p. 012 028.

[89] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, *Deep learning for 3d point clouds: A survey*, 2020. arXiv: `1912.12033 [cs.CV]`.