Master in Intelligent Interactive Systems

Universitat Pompeu Fabra

Imitation Learning and policy representation for Constrained Reinforcement Learning

Ana Caicoya Ros

Supervisor: Miguel Calvo-Fullana

Tuesday 2nd July, 2024

Contents

Acknowledgement

I would like to sincerely thank my supervisor, Miguel Calvo-Fullana, for his time, dedication, patience, and availability throughout this thesis. His guidance and support were crucial to this project's completion.

I am also grateful to my friends and classmates for their support and patience during this process. Your encouragement made a big difference.

Finally, I want to thank my family for their constant support and patience. Your belief in me kept me going.

Abstract

This thesis explores the integration of imitation learning and policy representation within the domain of constrained reinforcement learning (CRL) to enhance decision-making in environments with stringent limitations. Reinforcement Learning (RL) is a machine learning paradigm focused on training agents to make decisions by maximizing cumulative rewards. However, real-world scenarios often require additional constraints, such as safety and regulatory requirements, necessitating the use of CRL to ensure these constraints are respected while optimizing performance.

The research addresses the challenges of solving CRL problems using Generative Adversarial Imitation Learning (GAIL) within the framework of Constrained Markov Decision Processes (CMDPs). CMDPs provide a mathematical structure that incorporates constraints into the RL process. The methodology involves two key phases: the first phase uses the state-augmented CRL algorithm to obtain policies that satisfy the constraints in an augmented space, incorporating dual variables. The second phase refines these policies using GAIL to map them back to the original state space, leveraging imitation learning techniques to ensure robust performance.

Numerical results from simulations demonstrate the effectiveness of this approach in achieving constraint satisfaction while maintaining high performance. The findings indicate that the integration of CRL with imitation learning can lead to significant improvements in policy robustness and compliance with constraints. This research contributes to the broader field of machine learning by providing new insights and methods for developing constrained intelligent systems.

Keywords: Imitation Learning; Constrained Reinforcement Learning; Policy Representation; Generative Adversarial Imitation Learning; Constrained Markov Decision Processes

Chapter 1

Introduction

1.1 Introduction

When we want to solve a real-world problem, one of the methods we can use is **Reinforcement Learning** (**RL**), a subfield of machine learning concerned with decision-making in dynamic environments. Reinforcement Learning is essentially the process by which an agent learns which actions to take based on the state it finds itself in, endeavoring to maximize a numerical reward that depends on the environment in which it operates. In this process, the learner is not told which actions to take but instead must discover them through trial and error, analyzing the reward obtained in these situations.

However, in many real-world scenarios, there are additional **constraints** or requirements that must be satisfied during the decision-making process. These constraints may arise due to safety considerations, regulatory requirements, or domain-specific limitations. An example could be self-driving vehicles. In this scenario, we would like to have an autonomous car capable of driving safely and efficiently in various potential situations. To achieve this, a crucial aspect to consider is preventing collisions. One approach to addressing this is to impose the requirement that the car always maintain a certain safe distance from any object or living being in its vicinity.

In this type of problems, it is highly likely that conventional RL may not yield the

best results, as its sole objective is to maximize the obtained reward, disregarding the specified requirements. Therefore, in such cases, we focus on the use of **Constrained Reinforcement Learning** (CRL), which employs an approach that takes into account achieving these additional constraints. Thus, the primary goal of CRL is to find policies that not only maximize expected rewards but also adhere to specified constraints.

Similarly to how an RL problem can be modeled as a **Markov Decision Process** (MDP), CRL can be studied as a **Constrained Markov Decision Process** (CMDP). CRL problems involve finding policies that not only maximize expected rewards but also adhere to specified constraints, which is crucial in many real-world applications.

CMDPs serve as the mathematical framework for CRL, ensuring that solutions respect constraints while optimizing performance. In practice, transition probabilities are rarely known, making the direct application of CMDP solution methods impractical. Instead, CRL algorithms are employed to learn these probabilities and derive optimal policies. This distinction is analogous to how RL operates when the transition dynamics of MDPs are unknown.

Knowing this, the aim of this thesis is to solve a CRL problem through one of the methods that will be discussed later. However, this method will have the disadvantage that the solution will be found in an augmented space. Once the policy in the augmented space is found, we will conduct an inverse reinforcement learning process to obtain a new policy in the original space.

1.2 Objectives

The primary goal of this Master Thesis is to investigate constrained reinforcement learning (CRL) problems with a focus on policy representations. To achieve this, two specific objectives have been identified:

- **Study and implementation of CRL Algorithms**: This thesis aims to

explore CRL problems and the contemporary methodologies used to address them. Specifically, the state-augmented CRL algorithm developed in [1] will be studied and implemented. This method generates a state-augmented policy that ensures compliance with constraints while optimizing reward maximization.

- **Dimension reduction via imitation learning**: Given the importance of state augmentation for convergence in CRL problems [1], the second objective is to reduce the dimension of the resulting policy using imitation learning techniques. For this purpose, once the state-augmented policy has been learned, imitation learning will be employed to learn a new policy in the original unaugmented space.

1.3 State of the art

We have already mentioned that Constrained Reinforcement Learning problems can be modeled as Constrained Markov Decision Processes. Because of that, we can use this mathematical formulation to approach the resolution of the problem, which has already been investigated by several authors. In this section, we present and briefly explain some of them, providing a general context for the problem at hand.

Several methods have been developed to solve Constrained Markov Decision Processes (CMDPs). The first method, based on a Linear Program (LP), was pioneered by Cyrus Derman and Morton Klein in 1965, [2], further developed by C. Derman and Arthur F. Veinott in 1972, [2], Lodewijk Kallenberg in 1983, [3], and Arie Hordijk and L. Kallenberg in 1984, [4]. This LP-based method efficiently calculates the value of CMDPs for finite state and action spaces, considering both discounted or total cost, as well as average cost with unichain structure. However, it becomes computationally expensive for expected average cost with general multi-chain ergodic structure. Later, D. Krass in 1989, [5], proposed an alternative efficient way to obtain optimal policies from the LP for average cost.

A second method, introduced by Frederick J. Beutler and Keith W. Ross in 1985, [6],

and 1986, [7], utilizes a Lagrangian approach for single-constraint CMDPs. Linn I. Sennott in 1991, [8] and 1993, [9], extended this approach to countable state spaces. Lagrangian techniques for multiple constraints have been explored more recently but remain relatively unexploited.

A third method, also based on an LP, was presented by Altman and Shwartz in 1989, [10], and in 1993, [11], and further studied by Ross in 1989, [12]. This method involves mixing stationary deterministic policies through a time-sharing mechanism. Feinberg in 1993, [13], introduced a similar LP approach for finite MDPs, using a randomization approach equivalent to mixing policies. Although these LP-based approaches may require a large number of decision variables, they have been efficiently applied to problems with finite state spaces and even infinite state spaces in some cases.

These three solution methods are interconnected, providing insights into CMDPs and enabling the development of a unified theory. Additionally, they allow for generalization to multiple constraints and provide asymptotic results on convergence. Altman in 1996, [14], introduced an LP that computes the solution of CMDPs for all discount factors simultaneously, offering a solution in a finite number of steps despite using non-standard decision variables.

Nevertheless, CMDPs rely on transition probabilities to formulate the problem, since they are used to calculate the expected rewards. In many real-world scenarios this probabilities are unknown. Relying on them would result in an incomplete model. Therefore, we will adopt CRL techniques that do not directly depend on these transitions.

1.3.1 Constrained Reinforcement Learning algorithms

Several methodologies have been developed to address Constrained Reinforcement Learning (CRL) problems, each offering distinct advantages and applications. One prominent approach employs Lagrange multipliers, [15], a technique that transforms a constrained optimization problem into an unconstrained one by incorporating the

restrictions into the objective function as penalties. This method simplifies the optimization process, making it possible to apply standard reinforcement learning algorithms to the transformed problem.

Another significant strategy involves the use of trust regions, [16], which ensure the stability and convergence of the policy during training by limiting the extent of policy updates within a predefined region. This technique helps maintain the robustness of the learning process, particularly in environments where large policy updates could lead to suboptimal or unstable behavior. Trust region methods are widely recognized for their effectiveness in achieving reliable performance on CRL tasks.

In their work, Chen Tessler, Daniel J. Mankowitz and Shie Mannor, [17], introduced the Reward Constrained Policy Optimization (RCPO) method, which integrates the Lagrange multiplier framework with an actor-critic approach. This technique trains both the actor and the critic using a modified reward function that penalizes the agent for violating constraints, thereby guiding the learning process to adhere to the specified requirements while maximizing the reward. The RCPO method has shown promising results in various constrained environments, demonstrating its practical applicability and effectiveness.

Similarly, Joshua Achiam, David Held and Aviv Tamar and Pieter Abbeel, [18] proposed Constrained Policy Optimization (CPO), a method based on trust regions. CPO uses a policy search algorithm that generates trajectories from the current policy and estimates the expected cumulative reward. The algorithm then optimizes a new policy within the trust region defined by the Kullback-Leibler divergence, ensuring that policy updates remain stable and reliable. This approach effectively balances exploration and exploitation, making it suitable for complex CRL problems where stability is crucial.

Primal-dual methods, [19], offer another approach to solving CRL problems by simultaneously addressing the primal and dual aspects of the optimization problem. These methods iteratively adjust the policy parameters and the Lagrange multipli-

ers, seeking to maximize the cumulative reward while minimizing constraint violations. The optimization process alternates between updating the policy to improve performance and adjusting the multipliers to enforce the constraints, ensuring that the learned policy not only performs well but also adheres to the specified constraints.

Collectively, these methodologies highlight the diverse strategies available for tackling constrained reinforcement learning problems. Each method has its own unique strengths, and the choice of approach depends on the specific requirements and constraints of the task at hand.

1.3.2 Imitation Learning

We have now seen diverse approaches for solving CRL problems. In this paper, we will follow the one presented in [1], which trains a state-augmented policy. Nevertheless, having a policy in an augmented space is not a desirable scenario; it is preferable to have one in the original state space. This is why we introduce a new field: **Imitation Learning (IL)**. If we achieve a satisfactory result when learning a policy in the augmented space, IL will allow us to mimic this behavior and thus train a new policy in the non-augmented space.

Imitation learning, where an agent learns by mimicking an expert, includes several notable methods. Initially, behavioral cloning treats the task as supervised learning, mapping states to actions based on expert demonstrations. This straightforward approach works well with abundant and comprehensive data but can falter when encountering novel states [20].

Advancements in this field have led to more sophisticated techniques. Generative Adversarial Imitation Learning (GAIL), developed by Ho and Ermon, [21], combines the adversarial training of GANs with imitation learning, allowing for more detailed behavior replication and better handling of new situations. This method trains a policy and a discriminator simultaneously, with the discriminator distinguishing between expert and generated trajectories.

To address the limitations of behavioral cloning, Dataset Aggregation (DAgger), proposed by Ross et al., [22], iteratively refines the policy. Initially trained on expert demonstrations, the policy is then corrected by the expert, reducing compound errors and improving robustness.

Further refinement in the field is seen with Adversarial Inverse Reinforcement Learning (AIRL). Introduced by Fu, Luo, and Levine, [23], AIRL not only mimics the expert but also infers the underlying reward function the expert is optimizing, leading to more interpretable and transferable policies.

These methods collectively push the boundaries of imitation learning, enhancing the ability of agents to learn complex behaviors effectively. From the foundational behavioral cloning to the sophisticated AIRL, each approach addresses specific challenges, contributing to the development of intelligent and autonomous systems.

Chapter 2

Methods

2.1 Preliminaries

As mentioned in **Section 1.1**, **Reinforcement Learning** is a subfield of Machine Learning that focuses on mapping different situations to actions in order to maximize a reward. The decisions are not pre-established, but the learner must explore the environment in order to discover which of them can lead to the most positive outcome.

In this description we can find two main elements of the problem: the **agent** and the **environment**:

- **Agent**: the learner or decision-maker that interacts with the environment. The agent must be able to know the state of the environment and take actions that can affect it.
- **Environment**: world context in which the agent operates. It provides feedback on the agent's actions in the form of rewards and new states.

Beyond these two main components, we can identify the following subelements:

- **Policy** (π): defines the strategy followed by the agent. It basically represents a mapping of environment's states into agent's actions.

- **Reward signal ($r \in \mathbb{R}$)**: Numerical feedback the environment sends to the RL agent at each time step. The agent's objective is to maximize the total sum of rewards it receives over the long run. This signal is the main reason for altering the policy, since it is used by the agent during its learning process.

- **Value function**: in contrast with the rewards, which are immediate, the value function is a long-term indicative in the sense that it estimates the total gains over the long run from a specific state.

- **Model of the environment**: optional component that allows to infer the dynamics of the environment; that is, given a state-action pair, the model might predict the following state and reward. Reinforcement learning problems that do not follow *model-based* methods, are called *model-free* methods and the agent learns through trial-and-error.

2.1.1 Solving Reinforcement Learning problems

Let's define a standard RL problem. Let $t \in \mathbb{N} \cup 0$ denote the time variable, $\mathcal{S} \subset \mathcal{R}^n$ and $\mathcal{A} \subset \mathbb{R}^d$ be compact sets denoting the state and action spaces of an MDP with transition probability $p(s_{t+1}|s_t, a_t)$. The agent selects actions sequentially, guided by a policy $\pi \in \mathcal{P}(\mathcal{S})$. In this context, $\mathcal{P}(\mathcal{S})$ specifies a probability distribution over the action set $\mathcal{A}$ which is determined by the current state of the system.

Another vital element in a Reinforcement Learning problem is the reward function. Whenever the agent picks an action and moves to another state, it picks a reward that reflects how good the action was. The reward function then takes a pair state-action and returns a real number:

$$\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$
$$(s, a) \to r(s, a)$$

Since Reinforcement Learning is based on state-action transitions, typically these problems are formalized using Markov Decision Processes. MDPs involve not only

immediate reward, but also the total sum, forcing a trade-off between them. In an MDP, the interaction between the agent and the environment consists of the agent choosing an action based on a state, and the environment returning a new state and a numerical reward. This behavior is described in **Figure 1**. During this process, we will assume the system satisfies the **Markov property**, which states that the next state only depends on the current one and the action taken, not on the previous events.

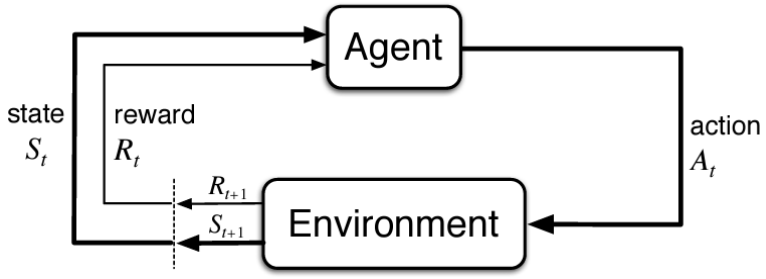


Figure 1: Agent-environment interaction in a Markov Decision Process. [24].

In order to solve an RL problem, as previously mentioned, we can use different methods. The first ones are model-based methods, which include value iteration or policy iteration, and in order to apply them, we must know the model of the environment. On the other hand, we have the free-model methods, where we can find Q-learning or State-Action-Reward-State-Action (SARSA)and here the agent learns a policy directly interacting with the environment.

Another possibility is to use **Actor-critic** methods. In these methods, we have two different networks so that we can combine the benefits of policy-based RL (update the policy by directly manipulating it) and value-based methods (update the policy by finding the optimal value function). The actor corresponds to the policy structure since it is used to select actions at each time step, while the estimated value function is known as the critic since its main function is to decide how good the action taken by the actor is. Among these methods, we can find Proximal Policy Optimization (PPO).

2.1.2 Into Constrained Reinforcement Learning

In real-world problems, sometimes maximizing the accumulated reward is not enough, and we must also meet some constraints regarding the behavior of the agent. To address these problems, we use Constrained Reinforcement Learning, which modifies the RL framework to include these constraints the agent must satisfy while, at the same time, maximizing the reward.

There are various methodologies to include these constraints, among which we can find the following:

- **Reward shaping**: modifying the reward signal to introduce penalties for not satisfying the constraints. [25].
- **Hard constraints**: directly introducing the constraints into the problem definition. [18], [17].

2.1.3 Constrained Markov Decision Processes

Just as for RL problems, MDPs serve as a mathematical framework; in the case of a CRL problem, we will use Constrained Markov Decision Processes. CMDPs introduce constraints into the formulation so that the trained policy can meet the requirements. Mathematically speaking, what a CMDP does is include restrictions for the rewards as follows:

$$\begin{aligned} \max_{\pi \in \mathcal{P}(\mathcal{S})} \quad & \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_0(s_t, a_t) \right] \\ \text{s.t.} \quad & \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_i(s_t, a_t) \right] \geq c_i, \ i = 1, ..., m \end{aligned} \tag{2.1}$$

2.2 Constrained Reinforcement Learning

If we remember the formulation of a RL problem in **Section 2.1.1**, we had the state and action spaces and a reward function. Since an agent could have multiple goals,

this function can be multidimensional, based on the environment set up, so we could have a collection of rewards $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for $i = 0, ..., m$ collected when the agent takes action a_t at state s_t. The main interest is to maximize the cumulative reward collected by the agent over time, which is represented by the value functions

$$V_i(\pi) \triangleq \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_i(s_t, a_t) \right] \tag{2.2}$$

Based on this, we introduce a Constrained Reinforcement Learning problem, where the first of these rewards (r_0) is interpreted as an objective to maximize, while the remaining ones are those that must satisfy a series of requirements.

$$\begin{aligned} \max_{\pi\in\mathcal{P}(\mathcal{S})} \quad & \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_0(s_t, a_t) \right] \\ \text{s.t.} \quad & \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_i(s_t, a_t) \right] \geq c_i, \ i = 1, ..., m \end{aligned} \tag{2.3}$$

Where the constants $c_i \in \mathbb{R}$ are the reward limitations that must hold. In this thesis, we will address the resolution of these problems following the methodology described in [1].

We can transform the problem in **Equation 2.3** into an unconstrained MDP by considering the penalized version, which allows us to solve the problem via common RL algorithms. We define the Lagrangian by introducing penalty coefficients λ_i as follows:

$$\mathcal{L}(\pi, \lambda) \triangleq V_0(\pi) + \sum_{t=0}^{m} \lambda_i \left(V_i(\pi) - c_i \right) \tag{2.4}$$

So now our objective is to find policies that maximize this Lagrangian

$$\Pi(\lambda) \triangleq \underset{\pi}{\text{argmax}} \quad V_0(\pi) + \sum_{t=0}^{m} \lambda_i \left(V_i(\pi) - c_i \right) \tag{2.5}$$

If we define a new reward functions as follows:

$$r_\lambda(s_t, a_t) = r_0(s_t, a_t) + \sum_{t=0}^{m} \lambda_i \left(r_i(s_t, a_t) - c_i\right) \tag{2.6}$$

we can rewrite the problem in 2.5 and define a value function $V(\pi, \lambda)$ associated with this reward and whose maximization gives us the set $\Pi\lambda$)

$$\Pi(\lambda) = \underset{\pi}{\operatorname{argmax}} \lim_{t\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi} \left[\sum_{t=0}^{T} r_\lambda(s_t, a_t)\right] \triangleq \underset{\pi}{\operatorname{argmax}}\, V(\pi, \lambda) \tag{2.7}$$

In the following, we will denote each of the maximizing policies for the Lagrangian by $\pi(\lambda)$ and their evaluation at a state s by $\pi(s, \lambda)$.

To ensure the proximity between 2.3 and 2.7 we introduce an iteration index k, a step size η_λ, and an epoch duration T_0 and update the coefficients by iteration

$$\lambda_{i,k+1} = \left[\lambda_{i,k} - \frac{\eta_\lambda}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \left(r_i(s_t, a_t) - c_i\right)\right] \tag{2.8}$$

where $a \sim \pi(s_t, \lambda_\mathbf{k})$.

After each iteration, the penalty coefficient $\lambda_{i,k}$ will be reduced if the accumulated reward during the k-th epoch exceeded the constraint c_i or increased if it was lower than the constraint. This procedure can be seen as a stochastic gradient descent update on the dual function.

Equation 2.8 generates trajectories nearly optimal by the **continuous execution** of the primal iteration in 2.7 and the dual iteration in 2.8. This means that it is not possible to stop the iteration of the coefficients at some k and state that the found policy $\pi(\lambda_\mathbf{k})$ is optimal.

Then, if we apply a primal-dual method to solve the CRL problem, what we observe is a switching behavior in the policies, which are not instantaneously optimal, but they satisfy the constraints on average. Let's visualize this with a simple example depicted in **Figure 2**.

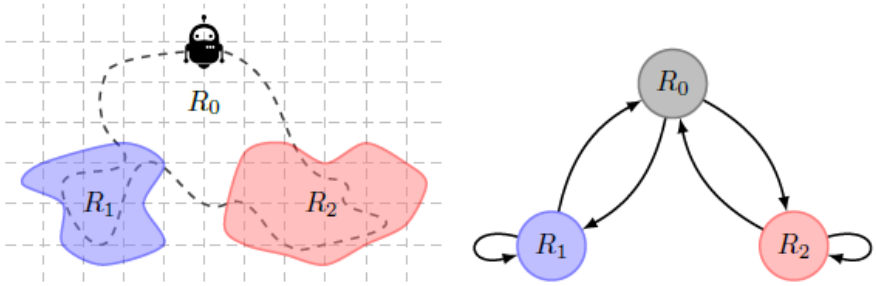
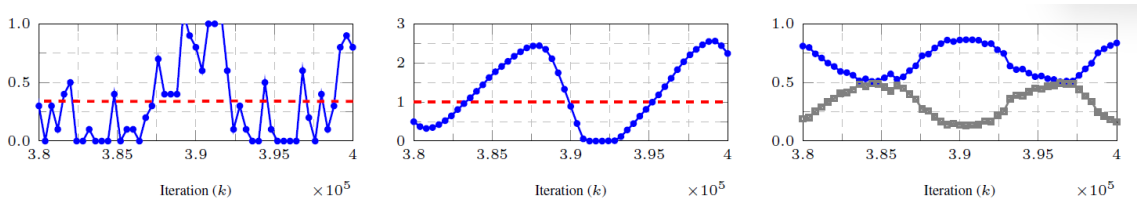


Figure 2: Graphical representation of the problem where an agent must expend c_1 of its time at region R_1 and c_2 at R_2. [1].

Suppose we have an MDP with three different states R_0, R_1 and R_2 and where the action taken determines the next state $s_{t+1} = a_t$. The possible actions are the following:

- if the actual state is R_0, the possible actions are $\{R_1,\ R_2\}$;
- if the actual state is $R_i,\ i \neq 0$, the possible actions are $\{R_0,\ R_i\}$

We want to spend at least $1/3$ of the time in each of the states R_1 and R_2, so we set $c_1 = c_2 = c = 1/3$. If we run the primal-dual method over this example, what we get is an oscillating behavior represented in **Figure 3**.



(a) Constraint satisfaction $V_1(\pi)$. (b) Dual variable $\lambda_{1,k}$. (c) Policy $\pi(s_t = R_1, \lambda_k)$.

Figure 3: Primal-dual method for the constrained MDP previously mentioned. [1]

To address this issue, we will approach the resolution of CRL problems through policies that reside in an augmented state space, including regularization coefficients. From this, under certain conditions, we will be able to obtain a policy that solves the original problem.

2.2.1 State Augmented Constrained Reinforcement Learning

To address the presented oscillatory issue, we will proceed as it is done in [1], following the **Augmented Constrained Reinforcement (A-CRL)** algorithm. By incorporating the dual variables into an augmented state space, we are allowing the learning of the Lagrange multipliers, improving the performance. The main objective of this algorithm is to obtain a policy that maximizes the Lagrangian (**Equation 2.4**) of the original problem.

A-CRL Algorithm

Training Phase

1. **Initialize Parameters**: Initialize the policy parameters θ and the dual variables λ.

2. **Sample Augmented State Space**: Sample states s and dual variables λ from the augmented space $S \times \Lambda$.

3. **Construct Augmented Rewards**: Calculate the augmented rewards $r_\lambda(s, a)$ as follows:
$$r_\lambda(s, a) = r_0(s, a) + \sum_{i=1}^{m} \lambda_i (r_i(s, a) - c_i)$$

4. **Policy Update**: Use a reinforcement learning algorithm (e.g., policy gradient, [26], or actor-critic methods, [27]) to update the policy parameters θ. The objective is to maximize the long-term expected reward:
$$\theta^* = \arg\max_\theta \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi_\theta} \left[\sum_{t=0}^{T} r_\lambda(s_t, a_t) \right]$$

Execution Phase

1. **Initialize Execution Parameters**: Given an initial state s_0 and initial dual variables $\lambda_0 = 0$.

2. **Epoch-Based Execution**: Divide the time into epochs of duration T_0.

3. **Rollout Actions**: For each epoch k, rollout T_0 steps with actions $a_t \sim \pi_{\theta^*}(s_t, \lambda_k)$.

4. **Update Dual Variables**: Update the dual variables using the following dynamics:

$$\lambda_{i,k+1} = \left[\lambda_{i,k} - \eta_\lambda \frac{1}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} (r_i(s_t, a_t) - c_i) \right]^+$$

This phase ensures that the policy $\pi_{\theta^*}(s, \lambda)$ adapts to the dual variables λ, effectively controlling the policy switching to satisfy the constraints.

At this juncture, our aim will be to seek a new policy that replicates this behavior but is not located in the augmented space. It is important to mention that the policy in the non-augmented space may not exist, as its existence depends on the fulfillment of certain mathematical conditions related to CMDPs. However, in this work, we will not delve into this and instead focus on the cases where this policy can indeed be obtained.

2.3 Imitation Learning

Imitation learning is a machine learning technique in which an agent learns to perform a task by mimicking the behavior of an expert. The primary objective is to replicate the expert's performance in a given environment, using only its observations, without access to the reward function that the expert may be optimizing.

Based on this, what we need to perform imitation learning, is a set of trajectories $\tau_E = \{s_0, a_0, s_1, a_1, ...\}$ generated from the expert policy, which is denoted as π_E. This approach contrasts with traditional reinforcement learning (RL), where an agent learns by interacting with the environment and receiving feedback in the form of rewards or penalties.

A key component to solving these kinds of problems will be the **occupation measure** (ρ_π) of a policy π. This measure can be seen as the distribution of state-action

pairs the policy encounters when navigating the environment, and is defined as follows:

$$\begin{aligned} &\rho_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \\ &(s,a) \to \rho_\pi(s,a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi) \end{aligned} \tag{2.9}$$

Where γ is a discount factor.

In [21], we can see how we can compare policies based on their occupancy measures. If the occupancy measure of the learned policy, ρ_π, closely matches the one from the expert, ρ_{π_E}, then the behavior of the trained learner will be similar to that of the expert. This is a key result used in many imitation learning algorithms.

There are two main approaches to imitation learning:

- **Behavioral Cloning**: treats the problem as a reinforcement learning one. It has as input a set of state-action pairs $\{(s_i, a_i)\}$ from the expert and its goal is to learn a policy that maps states into actions trying to minimize the discrepancies with the choice that would have made the expert:

$$\min_\pi \sum_{(s,a)\in\tau_E} \|a - \pi(s)\|^2$$

- **Inverse Reinforcement Learning (IRL)**: the aim of this approach is to infer the expert's reward function and then obtain a policy solving the reinforcement learning problem. Based on this, this methodology can be formulated as:

$$\min_{r\in\mathcal{R}} \|\rho_{\pi_r} - \rho_{\pi_E}\|^2$$

where π_r denotes the policy obtained by the RL process.

2.4 Generative Adversarial Imitation Learning

2.4.1 Introduction

Generative Adversarial Imitation Learning (GAIL) is an imitation learning algorithm first proposed in [21], which combines Generative Adversarial Networks [28] with imitation learning. The basic idea behind this algorithm is to train both a **generator**, which will be the learned policy, and a **discriminator**, whose function is to act as a classifier, distinguishing between actions taken by the generator and the expert ones.

GAIL operates within the framework of Markov Decision Processes, where policies π map states into actions and the transition probabilities are given by the dynamical model $P(s'|s, a)$. As we will later see, one of the central components of GAIL is the use of occupancy measures to evaluate and improve the policy.

2.4.2 Framework

GAIL introduces a direct approach to imitation learning by drawing an analogy between IRL and GANs. The key idea is to bypass the cost function recovery step in IRL and directly learn a policy through adversarial training.

- **Generative Adversarial Networks (GANs)** splits the training phase into two different networks. It trains a generator $\boldsymbol{G}$ that creates data samples and a discriminator $\boldsymbol{D}$ that distinguishes actions taken by the generator and the expert. In this approach, the generator's goal is to fool the discriminator.

- **GAIL** uses a similar adversarial setup where the generator is replaced by the policy π and the discriminator $\boldsymbol{D}$ differentiates between state-action pairs from the expert π_E and those from π. The objective is to find a policy that minimizes the Jensen-Shannon divergence between the occupancy measures of π and π_E

2.4.3 Mathematical formulation

As previously mentioned, GAIL relies on the use of the occupancy measure ρ_π, introduced in **Equation 2.9**. This variable represents the expected discounted distribution of state-action pairs encountered when following policy π.

This algorithm is based on an IRL approach, more specifically a maximum causal entropy IRL, [29], which tries to fit a **cost function**[1], c, from a family of functions $\mathcal{C}$ with the optimization problem:

$$\underset{c\in\mathcal{C}}{\text{maximize}} \left(\min_{\pi\in\Pi} -H(\pi) + \mathbb{E}_\pi \left[c(s,a)\right] \right) - \mathbb{E}_{\pi_E} \left[c(s,a)\right] \tag{2.10}$$

where $H(\pi) \triangleq \mathbb{E}_\pi \left[-\log \pi(a|s)\right]$ is the γ-discounted causal entropy.

Based on this, the procedure will be to perform RL over the cost function obtained via IRL on the largest possible set of cost functions: $\mathcal{C} = \mathbb{R}^{\mathcal{S}\times\mathcal{A}} = \{c : \mathcal{S}\times\mathcal{A} \to \mathbb{R}\}$. This selection for $\mathcal{C}$ has the drawback of being too large, which can lead to overfitting. This is why a closed, proper and convex cost function regularizer $\varphi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \bar{\mathbb{R}}$. The resulting IRL problem remains as follows:

$$IRL_\varphi(\pi_E) = \underset{c\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}}{\text{argmax}} -\varphi(c) + \left(\min_{\pi\in\Pi} -H(\pi) + \mathbb{E}_\pi \left[c(s,a)\right] \right) - \mathbb{E}_{\pi_E} \left[c(s,a)\right] \tag{2.11}$$

and the wanted policy would be $RL(\tilde{c})$ where $\tilde{c} \in IRL_\varphi(\pi_E)$ and

$$RL(c) = \underset{\pi\in\Pi}{\text{argmin}} -H(\pi) + \mathbb{E}_\pi \left[c(s,a)\right] \tag{2.12}$$

Finally, the objective problem can be written as:

$$RL \circ IRL_\varphi(\pi_E) = \underset{\pi\in\Pi}{\text{argmin}} -H(\pi) + \varphi^*(\rho_\pi - \rho_{\pi_E}) \tag{2.13}$$

where φ^* represents the convex conjugate of the regularizer.

[1] A cost function $c \in \mathcal{C}$ can be interpreted as the opposite of the reward function ($c(s,a) = -r(s,a)$). It describes the penalization for taking certain actions.

2.4.4 GAIL Algorithm

The optimization problem which is been tried to solve is the following:

$$\min_{\pi} \max_{D} \mathbb{E}_{\pi_E} [\log D(s,a)] + \mathbb{E}_{\pi} [\log (1 - D(s,a))] \tag{2.14}$$

where D is the discriminator and π is the generator trying to minimize the discriminator accuracy by mimicking the expert's behaviour.

The algorithm to perform this optimization can be described as follows:

1. **Initialize** the policy π_θ and the discriminator D_ϕ with random parameters θ and ϕ respectively.

2. **Sample trajectories** from both the expert, π_E, and the actual policy, π_θ.

3. **Update the discriminator** D_ϕ by training it to distinguish between expert trajectories and the ones from the actual policy. The objective here is to minimize the loss:

   $$\mathcal{L}_D(\phi) = -\mathbb{E}_{\pi_E} [\log D_\phi(s,a)] - \mathbb{E}_{\pi_\theta} [\log (1 - D_\phi(s,a))]$$

   which is done updating the parameters ϕ via gradient descent:

   $$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \mathcal{L}_D(\phi)$$

   where α_ϕ is a chosen learning rate.

4. **Update the policy** π_θ by maximizing the reward provided by the discriminator. This is equivalent to minimizing :

   $$\mathcal{L}_\pi(\theta) = -\mathbb{E}_{\pi_\theta} [\log D_\phi(s,a)]$$

   which again is accomplished by updating the parameter θ via gradient descent in an analogous form as the previous step.

5. **Repeat**.

2.5 Proposed methodology

Once we have all the necessary information and concepts, we can proceed to put everything together. The methodology will be as follows:

1. First of all we will obtain a policy that solves the CRL in the augmented space by following the A-CRL algorithm, which can be written as follows:

Algorithm 1 A-CRL Training Phase

Output: Trained policy π_{θ^*}

1: Sample (s, λ) from augmented space $S \times \Lambda$

2: Construct the augmented rewards [cf. 2.6]

$$r_\lambda(s,a) = r_0(s,a) + \sum_{i=1}^{m} \lambda_i \left(r_i(s,a) - c_i\right)$$

3: Use the RL algorithm to obtain policy

$$\theta^* = \arg\max_{\theta \in \mathbb{R}^d} \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{s,a\sim\pi_\theta} \left[\sum_{t=0}^{T} r_\lambda(s_t, a_t)\right]$$

Algorithm 2 A-CRL Execution Phase

Input: Policy π_{θ^*}, step η_λ, requirements c_i, epoch T_0

Output: Trajectories (s_t, a_t) satisfying Theorem 1

1: **Initialize:** Given initial state s_0, dual variable $\lambda_0 = 0$

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

3: Rollout T_0 steps with actions $a_t \sim \pi_{\theta^*}(s_t, \lambda_k)$

4: Update dual dynamics [cf. (6)]

$$\lambda_{i,k+1} = \left[\lambda_{i,k} - \frac{\eta_\lambda}{T_0} \sum_{t=kT_0}^{(k+1)T_0-1} \left(r_i(s_t,a_t) - c_i\right)\right]_+$$

5: **end for**

2. Once we have a trained policy, we can extract expert trajectories in order to apply apply the GAIL algorithm over them and obtain a new policy in the unaugmented state space. This algorithm can be written as follows:

Algorithm 3 Generative adversarial imitation learning

1: **Input**: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0

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

3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$

4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_w \log(D_w(s,a))\right] + \hat{\mathbb{E}}_{\tau_E}\left[\nabla_w \log(1 - D_w(s,a))\right]$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta),$$

$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}\left[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$$

6: **end for**

With these steps, we can obtain a policy that resides in the original space and complies with the constraints established in the problem.

Chapter 3

Numerical Results

In this chapter, we will consider a practical example to illustrate our approach. Suppose we have an agent capable of navigating a space divided into three distinct regions. The agent's objective is to spend a specified portion of time in two of these regions, with the remaining time spent freely. This scenario can be formulated as a Constrained Reinforcement Learning (CRL) problem, where the constraints are the time requirements for the two regions.

To solve this problem, we will follow the methodology outlined in **Chapter 2**. First, we will define the environment, which includes specifying the state space and the constraints. From this environment, we will derive a policy in an augmented space that includes two additional dimensions to account for each constraint. With this augmented policy, we will sample trajectories and apply imitation learning to obtain a new policy in the original state space. This process allows us to effectively manage the time constraints while guiding the agent's behavior.

3.1 Problem set up

We propose a scenario in which an agent can move continuously in a two-dimensional space. This space is a square with a side length of 10, so the state space can be written as $\mathcal{S} = [0, 10] \times [0, 10]$. Each state s_t will then be given by the x and

y coordinates of the agent at time t. Within this square, there are two circular regions with centers at $(2.5, 7.5)$ and $(7.5, 2.5)$, each with a radius of 2. This setup is illustrated in **Figure 4**.

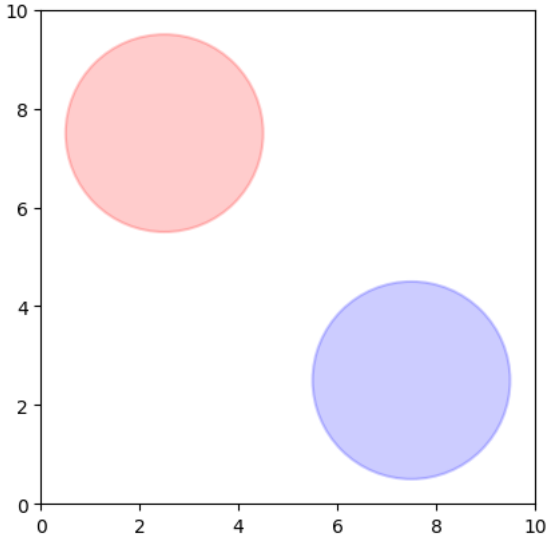


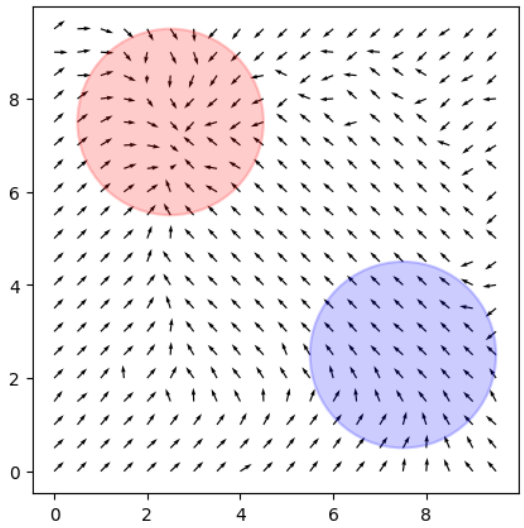
Figure 4: Environment set up.

The constraints are established on the proportion of time the agent must spend in each of these regions, which is a minimum of 40% in each. The agent will navigate the environment by sampling actions from a Gaussian policy, where the mean and standard deviation are trained by neural networks. This action, a_t, will always be normalized to be a unit vector, and the next state of the agent will be given by the expression $s_{t+1} = s_t + Ta_t$, where $T = 0.5$.

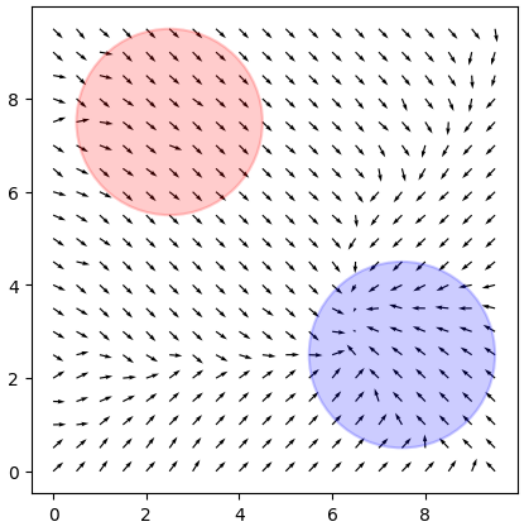
3.2 Augmented Constrained Reinforcement Learning

As previously mentioned, our first step will be to obtain a policy in the augmented space by incorporating the dual variables of the Primal-Dual formulation into the policy, as described in **Section 2.1.2**. For this purpose, we define the reward function as $r_0(s_t, a_t) = 0$ and $r_i(s_t, a_t) = 1$ when the agent is inside the $i - th$ region. Since the environment set up is the one illustrated in **Figure 4**, we have $i \in \{1, 2\} = \{red, blue\}$.

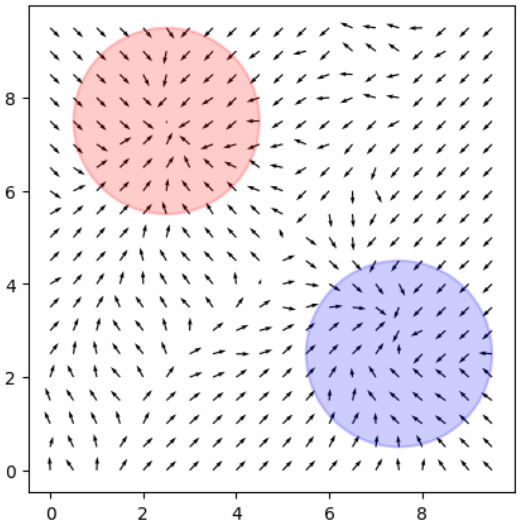
For the training phase, we have used a Soft Actor-Critic (SAC) algorithm, [30], already implemented in the *StableBaselines* library, [31]. In this algorithm, the policy is represented by a Neural Network, which has been chosen to be a Multi-Layer Perceptron, and it outputs the mean and standard deviation of a Gaussian distribution from which the actions will be sampled. For this training, we have specified a maximum number of steps $T = 40$, a step size of $\eta_\theta = 0.1$ and $, 1000, 000$ iterations.



((a)) Policy for $\lambda = [7, 0]$.



((b)) Policy for $\lambda = [0, 7]$.



((c)) Policy for $\lambda = [7, 7]$.

Figure 5: Representation of the policy trained with the A-CRL algorithm for different values of the Lagrange multipliers.

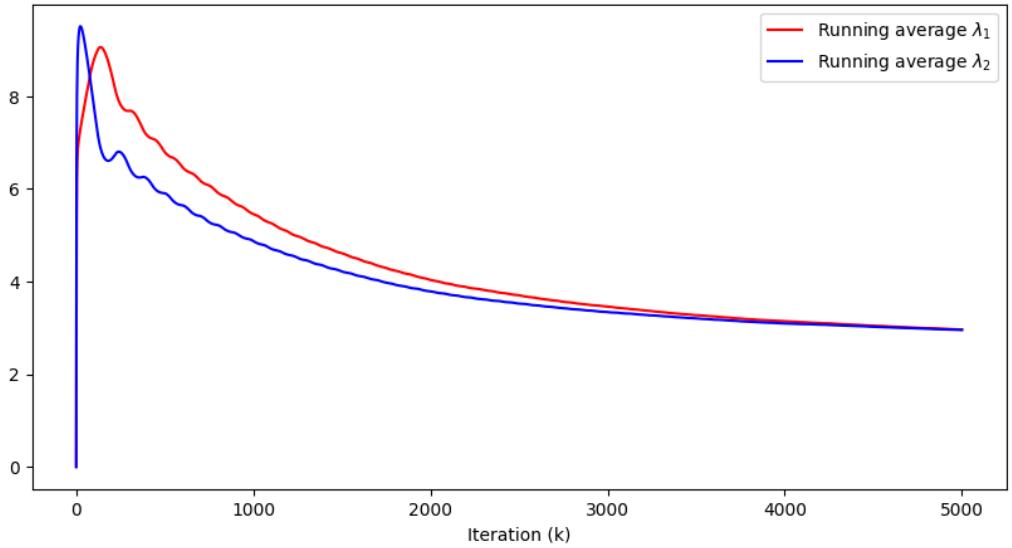
The result is a policy that includes the Lagrange multipliers for each constraint as part of the observation space, and updates them according to **Equation 2.8**. When the multiplier associated with a region takes a higher value than the other one, the policy leads the agent towards that region. This can be explained by the way the multipliers are updated. If one of the two values is higher, it could be due to the agent spending less time in that area than required, resulting in an increase in its value. Additionally, by keeping the lambda values constant, the policy does not change, remaining fixed while attempting to compensate for this lack of time in the region by guiding the agent towards it.

This behavior can be observed in **Figure 5**, where we can appreciate that when $\lambda_1 = 7$ and $\lambda_2 = 0$ (**Subfigure 3.5(b)**) the policy tends to move towards the red region, but when we change the value for the multipliers to $\lambda_1 = 0$ and $\lambda_2 = 7$ (**Subfigure 3.5(b)**) it drives the agent in the direction of the blue region. If we assign the same value to both multipliers as it is done in **Subfigure 3.5(c)**, we observe that the environment is divided in half by a diagonal, with each half containing one region. Depending on which half the agent is located in, it will be guided to the corresponding region.

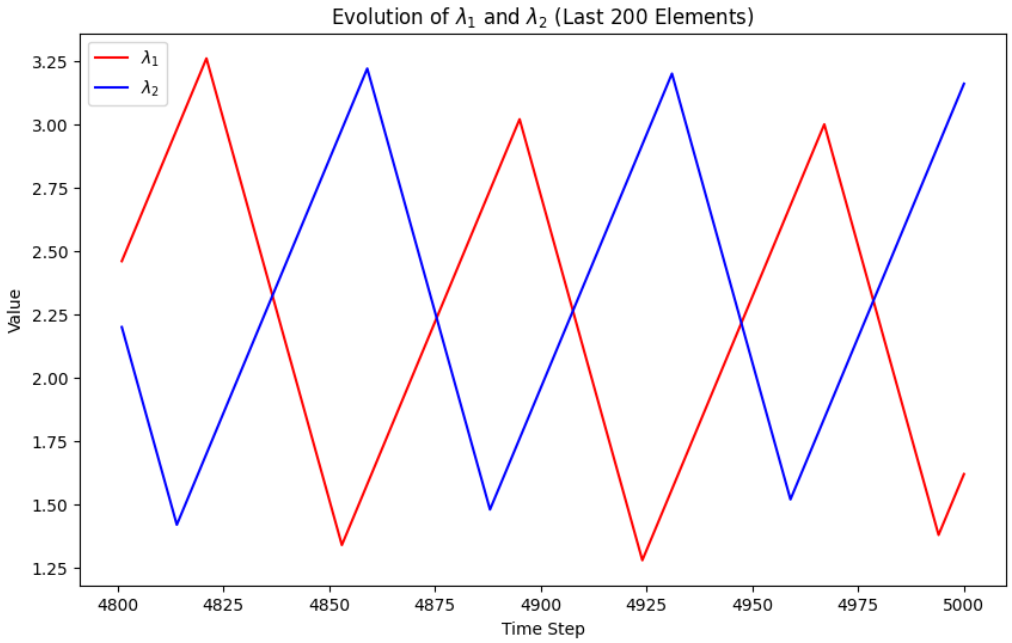
As we can see in **Algorithm 1**, the policy is trained for random choices of λ and initial state, so the resultant policy maximizes the Lagrangian of the problem, described in **Equation 2.4** for every possible choice of the Lagrange multipliers. When executing this policy, we will use a step size $\eta_\theta = 0.1$ and an epoch length $T_0 = 1$, so that the lambdas will be updated according to **Equation 2.8** after each action taken by the agent.

In **Figure 3.6(a)** we represent the running averages of both Lagrange multipliers over a single run of the agent with 5000 steps. As we can appreciate, at the beginning, both variables take high values. This can be seen as a calibration step. When starting each run, the agent picks a random position, and since the action vector is restricted to be unitary, it can take a while to arrive to one of the regions and accomplish the time requirements. Not meeting the requirements initially will necessitate adjusting the variables, increasing their value. However, as the agent

progresses along its path, being already trained, it will start meeting the constraints, and therefore the value of the multipliers will decrease and tend to have the same value. This is due to the fact that the temporal requirement for the regions is the same for the two of them.



((a)) Running averages.



((b)) Instant values.

Figure 6: Dual variables when executing a policy π_θ trained with the A-CRL algorithm. The values are shown for a single run of 5000 steps and in the case of the instant values only the last 200 steps are shown.

On the other hand, in **Figure 3.6(b)**, we can see the instantaneous value of the dual variables for the last 200 steps. Here, the oscillatory behavior of the primal-dual methods mentioned in **Section 2.1.2** is evident. This behavior posed a problem when trying to achieve a stationary policy without incorporating the Lagrange multipliers as part of the state, as it created the need to execute the policy permanently.

To conclude the analysis of these results, we can consider whether, as a general rule, the trained policy meets the established temporal requirements, which, let's recall, are to spend at least 40% of the time in each of the regions. For this purpose, in **Figure 7**, we can observe the proportion of time spent in each of the circles for each step. In this image, a solid line represents the average of these proportions over 100 runs of the agent, each with 5,000 steps. On the other hand, the shaded region corresponds to the range between the execution with the lowest constraint satisfaction and the average across all executions.

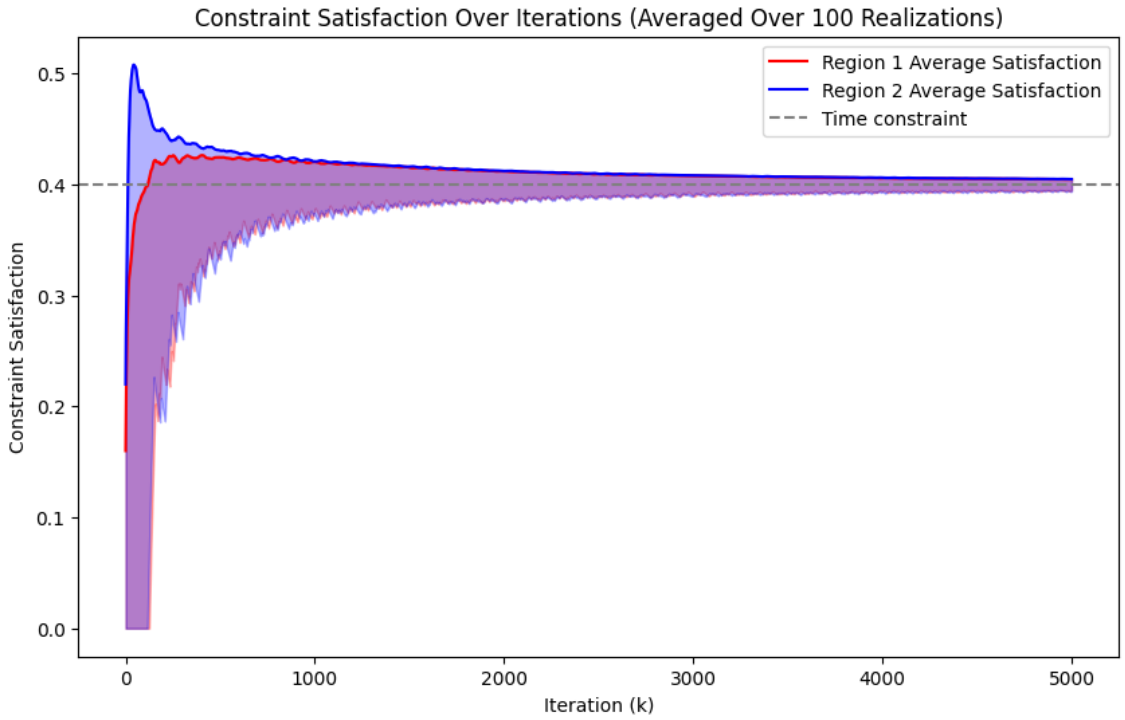


Figure 7: Constraint satisfaction when executing the policy π_θ 100 times over 5,000 steps. The solid line corresponds to the average constraint satisfaction for each region over the 100 runs.

Here we can appreciate how, in general, the policy trained with the A-CRL algorithm, satisfies the established constraints. However, it's a bit tight because, with the requirement to spend 40% of the time in each region, the agent only has 20% left to move around more freely. Given that it has to travel between regions, that does not leave much room for flexibility.

3.3 Imitation learning

Now that we have our policy in the augmented space, which in this case is four-dimensional, it is time to obtain a new one in the reduced bi-dimensional space. For this purpose, we generate $10,000$ trajectories with $1,000$ steps each. The occupation measure for these trajectories can be seen in **Figure 8**, and this is the behavior we wish to clone.

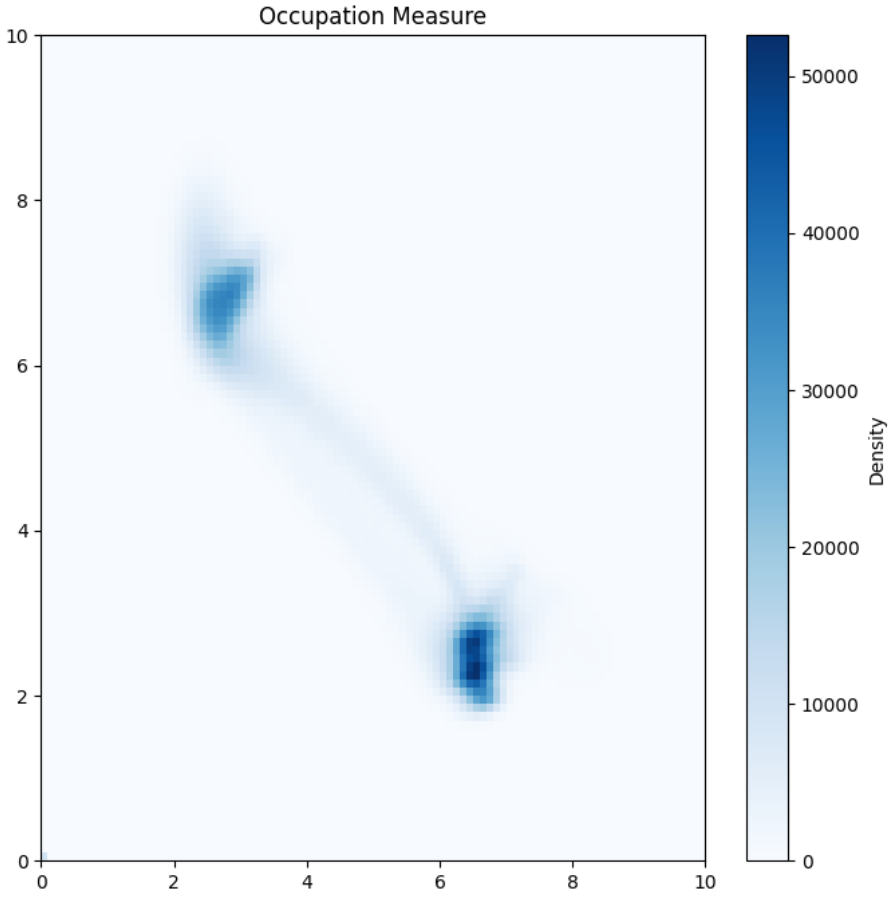


Figure 8: Occupation measure after executing the policy π_θ trained with the A-CRL algorithm for 10000 runs of 1000 steps each.

For this second step, we have chosen to use the GAIL algorithm to perform imitation learning. In this case, we do not have to properly define a reward function based on the environment. Instead, we will need to provide a reward network that will be trained as part of the process and will play the role of a discriminator, trying to distinguish between actions taken by the generator and those of the expert. Apart from this network, we must also specify the environment in which the agent will operate and initialize the latter.

In this case, instead of using SAC to train the learner, we have chosen to use the Proximal Policy Optimization (PPO) algorithm, [32], which is also implemented in *StableBaselines*. This algorithm is similar to TRPO in that it also aims to optimize the policy without taking excessively large steps; however, it is simpler to implement. It uses stochastic gradient descent and achieves a similar performance to TRPO.

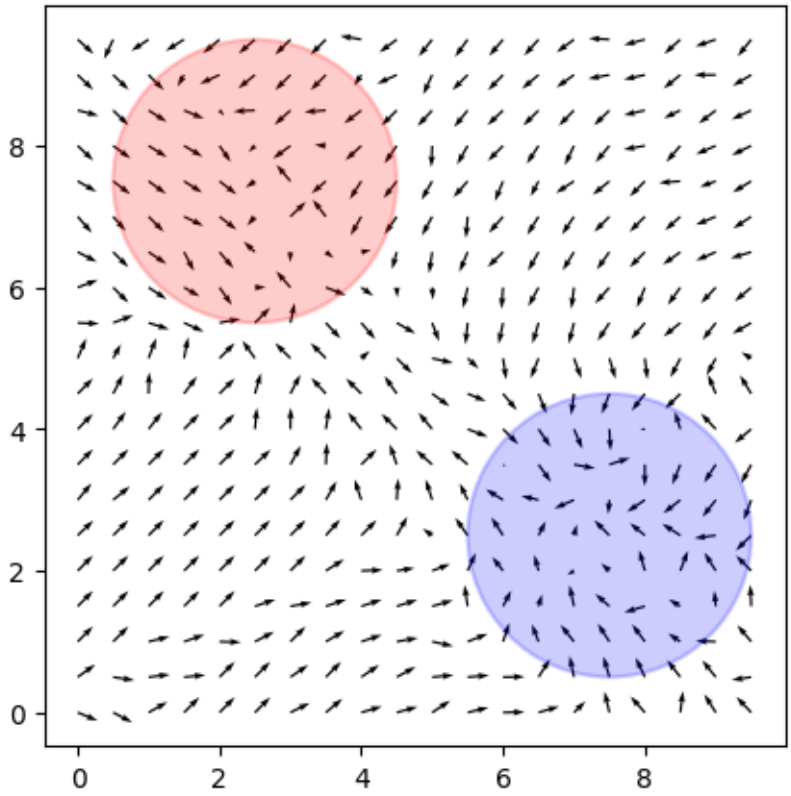
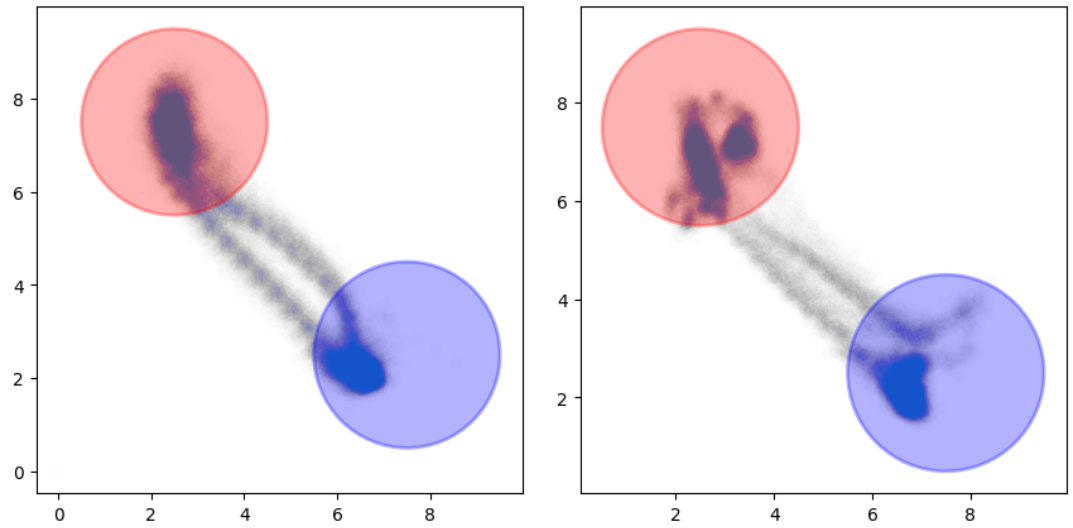


Figure 9: Policy trained by the GAIL algorithm, imitating trajectories from the expert, which in this case is the previously trained agent with the A-CRL algorithm. This policy has been trained for 15 million steps.

It is important to note that in this case, there are no dual variables involved, so the agent's next step will be determined solely by its current state, and the representation of the policy is then static. After initializing the reward network and the learner, we trained the new policy for 15 million steps. The resulting policy can be seen in **Figure 9**. Based on this, we can predict how the agent will navigate the environment.

When the learner is outside any of the regions, the policy guides it to one of these areas. However, when it is inside, we see that the most common movement is to stay within that circle. We can also observe a sort of channel between both regions, which will ensure that the agent does not get trapped inside one of them but can move between both and thus fulfill both requirements.



((a)) Agent trained with the A-CRL algorithm.

((b)) Agent trained with the GAIL algorithm.

Figure 10: Comparison of trajectories following each of the trained policies. The represented trajectories are each 25,000 steps long, and the starting point is random, not the same for each of them.

If we generate a trajectory based on this policy, we can verify that the agent has succeeded in the task of learning to be 40% of the time in each region. In **Figure 10**, we can observe the comparison between a 25,000-step trajectory generated with each of the trained agents. On the left, in **Figure 3.10(a)**, we have the agent trained with the A-CRL algorithm (**Algorithms 1** and **2**), which follows a policy

that resides in the augmented space. Therefore, after each step, it not only updates its position but also the Lagrange multipliers.

On the other hand, in **Figure 3.10(b)**, a trajectory under the same conditions as the previous one is represented, but in this case, the agent has been trained by imitating the trajectories of the previous agent through the GAIL algorithm (**Algorithm 3**). Remember that in this case, the policy followed by the agent is static in the sense that it only depends on its position. If we compare both trajectories, we see that they are very similar, so we have achieved the desired result.

Lastly, if we seek greater evidence of satisfaction in the training of this new policy, we can verify whether it truly meets the established requirements. To this end, in **Figure 11**, we can observe the average portion of time the agent has spent in each of the regions over 100 trajectories of 5,000 steps each. Here, the solid line represents the average at each step, while the shaded area encompasses this average value down to the lowest value across all trajectories.

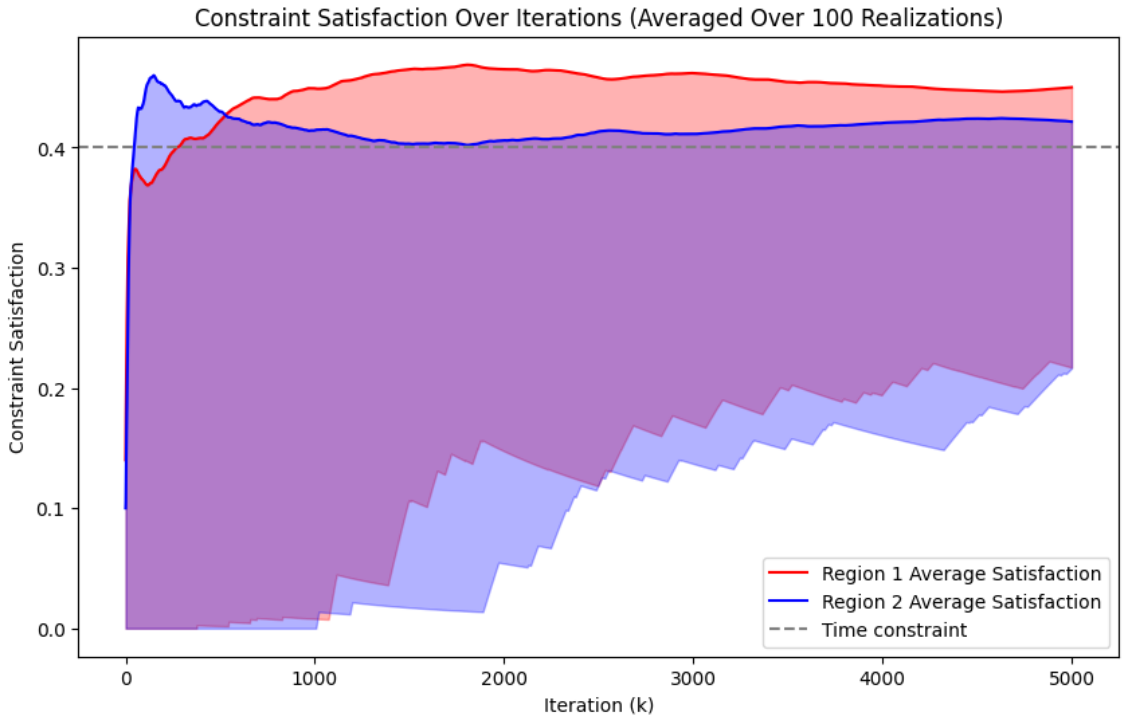


Figure 11: Constraint satisfaction when executing the policy π trained with GAIL, 100 times over 5,000 steps. The solid line corresponds to the average constraint satisfaction for each region over the 100 runs.

If we compare **Figure 11 (GAIL)** with **Figure 7 (A-CRL)**, we can observe that the margin of error for the policy trained through imitation is greater than for the policy in the augmented state space. This may be due to the difficulty of learning the exact behavior solely through imitation of an expert's trajectories. Nonetheless, the average exceeds the established requirements, so the obtained result is as desired.

Chapter 4

Discussion

4.1 Discussion

The results obtained from the application of the Augmented Constrained Reinforcement Learning (A-CRL) algorithm demonstrate the effectiveness of using an augmented state space and dual variables to address constrained reinforcement learning problems. By introducing the Lagrange multipliers into the policy, the A-CRL algorithm is able to adjust the dual variables dynamically and enforce the constraints during the execution. As shown in **Chapter 3**, the agent trained with **Algorithm 1**, has successfully learned to navigate the environment and satisfy the time requirements.

However, the process was not without its limitations. One significant challenge encountered was the computational complexity and time required for training the policies, particularly when applying the Generative Adversarial Imitation Learning (GAIL) algorithm. The first needed step was generating the trajectories, what already required a long time.

After that, training the agent was also considerably time-consuming, taking a considerable amount of computational resources to achieve convergence. It is important to remember here that it took 15 million steps in the training phase to accomplish the wanted behavior, compared with the one million steps needed for the state-

augmented policy. This could be due to the iterative nature of adversarial training, where both the policy (generator) and the discriminator are trained simultaneously.

Additionally, in **Figure 6**, we were able to observe the oscillatory behavior mentioned in **Section 2.1.2** when executing a primal-dual method. Nevertheless, thanks to introducing the dual variables into the state space, this continuous adjustment helped to satisfy the constraints.

The use of state augmentation, while effective, also introduced an added layer of complexity in terms of policy representation and interpretation. Going from an augmented policy to a reduced policy through imitation learning was a crucial step in simplifying the policy for practical applications. However, ensuring that the reduced policy retains the performance characteristics of the augmented policy remains a non-trivial task.

Despite these limitations, the A-CRL and GAIL approaches collectively provide a robust framework for addressing constrained reinforcement learning problems. The ability to train policies that adhere to complex constraints while optimizing performance opens up new possibilities for real-world applications where safety and regulatory compliance are critical.

4.2 Conclusions

This thesis has explored innovative approaches to solving constrained reinforcement learning (CRL) problems, focusing on efficient policy representations. Through the study and implementation of the State-Augmented Constrained Reinforcement Learning (A-CRL) algorithm and the Generative Adversarial Imitation Learning (GAIL) method, significant advancements were made in developing effective and practical solutions for CRL challenges.

The A-CRL algorithm effectively integrated dual variables into the state space, allowing the agent to continuously adjust its policy to meet constraints while maximizing rewards. This approach proved successful in scenarios requiring strict adherence to specified temporal constraints within different regions of the state space.

The agent was able to balance the trade-off between optimizing performance and satisfying these constraints, demonstrating the robustness of the A-CRL method.

Following the initial success of the A-CRL algorithm, the GAIL method was employed to reduce the dimensionality of the policy space. By training a new policy that mimicked the behavior of the state-augmented policy, GAIL facilitated the transition from an augmented to a reduced state space. This process simplified the policy representation, making it more practical for real-world applications while maintaining the desired performance and adherence to constraints.

Despite the promising results, several challenges were encountered during the research. One notable issue was the significant computational resources and time required for training the policies, especially when applying the GAIL algorithm. Generating the necessary trajectories and training the agent to achieve convergence demanded considerable computational power, highlighting the resource-intensive nature of these advanced algorithms.

In conclusion, the combination of A-CRL and GAIL offers a robust framework for addressing constrained reinforcement learning problems. The ability to train policies that comply with complex constraints while optimizing performance is crucial for applications where safety and regulatory compliance are paramount. The methodologies developed in this thesis provide a foundation for future research and development in CRL, paving the way for intelligent systems that can operate effectively in dynamic and constrained environments. These advancements hold promise for a wide range of real-world applications, from autonomous vehicles to robotic systems, where intelligent decision-making under constraints is essential.

List of Figures

List of Tables

Bibliography

[1] Calvo-Fullana, M., Paternain, S., Chamon, L. F. O. & Ribeiro, A. State augmented constrained reinforcement learning: Overcoming the limitations of learning with rewards. *CoRR* **abs/2102.11941** (2021).

[2] Derman, C. & Klein, M. Some remarks on finite horizon markovian decision models. *Operations Research* **13**, 272–278 (1965). URL `http://www.jstor.org/stable/168079`.

[3] Kallenberg, L. *Linear programming and finite Markovian control problems*, vol. 148 (1983).

[4] Hordijk, A. & Kallenberg, L. C. M. Constrained undiscounted stochastic dynamic programming. *Math. Oper. Res.* **9**, 276–289 (1984). URL `https://doi.org/10.1287/moor.9.2.276`.

[5] Krass, D. *Contributions to the Theory and Applications of Markov Decision Processes*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD (1989). Ph.D. thesis, Department of Mathematical Sciences.

[6] Beutler, F. J. & Ross, K. W. Optimal policies for controlled markov chains with a constraint. *Journal of Mathematical Analysis and Applications* **112**, 236–252 (1985). URL `https://www.sciencedirect.com/science/article/pii/0022247X85902884`.

[7] Beutler, F. J. & Ross, K. W. Time-average optimal constrained semi-markov decision processes. *Advances in Applied Probability* **18**, 341–359 (1986).

[8] Sennott, L. I. Constrained discounted markov decision chains. *Probability in the Engineering and Informational Sciences* **5**, 463–475 (1991).

[9] Sennott, L. I. Constrained average cost markov decision chains. *Probability in the Engineering and Informational Sciences* **7**, 69–83 (1993).

[10] Altman, E. & Shwartz, A. Optimal priority assignment: a time sharing approach. *IEEE Transactions on Automatic Control* **34**, 1098–1102 (1989).

[11] Altman, E. & Shwartz, A. Time-sharing policies for controlled markov chains. *Operations Research* **41**, 1116–1124 (1993). URL `http://www.jstor.org/stable/171605`.

[12] Ross, K. W. Randomized and past-dependent policies for markov decision processes with multiple constraints. *Operations Research* **37**, 474–477 (1989). URL `http://www.jstor.org/stable/171066`.

[13] Feinberg, E. A. Constrained semi-markov decision processes with average rewards. *Zeitschrift für Operations Research* **39**, 257–288 (1994). URL `https://doi.org/10.1007/BF01435458`.

[14] Altman, E., Hordijk, A. & Kallenberg, L. On the value function in constrained control of markov chains. *Math. Meth. Oper. Res.* **44**, 387–399 (1996).

[15] Bhatnagar, S. & Lakshmanan, K. An online actor-critic algorithm with function approximation for constrained markov decision processes. *Journal of Optimization Theory and Applications* **153** (2012).

[16] Schulman, J., Levine, S., Moritz, P., Jordan, M. I. & Abbeel, P. Trust region policy optimization (2017). `1502.05477`.

[17] Tessler, C., Mankowitz, D. J. & Mannor, S. Reward constrained policy optimization (2018). `1805.11074`.

[18] Achiam, J., Held, D., Tamar, A. & Abbeel, P. Constrained policy optimization (2017). `1705.10528`.

[19] Chen, Y., Dong, J. & Wang, Z. A primal-dual approach to constrained markov decision processes (2021). 2101.10895.

[20] Bain, M. & Sammut, C. A framework for behavioural cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*, 103–129 (Oxford University, GBR, 1999).

[21] Ho, J. & Ermon, S. Generative adversarial imitation learning (2016). 1606.03476.

[22] Ross, S., Gordon, G. J. & Bagnell, J. A. A reduction of imitation learning and structured prediction to no-regret online learning (2011). 1011.0686.

[23] Fu, J., Luo, K. & Levine, S. Learning robust rewards with adversarial inverse reinforcement learning (2018). 1710.11248.

[24] Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (A Bradford Book, Cambridge, MA, USA, 2018).

[25] Ng, A. Y., Harada, D. & Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, 278–287 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999).

[26] Neu, G. & Szepesvari, C. Apprenticeship learning using inverse reinforcement learning and gradient methods (2012). URL https://arxiv.org/abs/1206.5264. 1206.5264.

[27] Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M. & Lee, M. Natural actor–critic algorithms. *Automatica* **45**, 2471–2482 (2009). URL https://www.sciencedirect.com/science/article/pii/S0005109809003549.

[28] Goodfellow, I. J. *et al.* Generative adversarial networks (2014). 1406.2661.

[29] Ziebart, B. D., Maas, A., Bagnell, J. A. & Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, 1433–1438 (AAAI Press, 2008).

[30] Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018). URL https://arxiv.org/abs/1801.01290. 1801.01290.

[31] Raffin, A. *et al.* Stable-baselines3: reliable reinforcement learning implementations. *J. Mach. Learn. Res.* **22** (2021).

[32] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms (2017). URL https://arxiv.org/abs/1707.06347. 1707.06347.

[33] Altman, E. *Constrained Markov Decision Processes* (Routledge, 1999), 1st edn.

[34] Moldovan, T. M. & Abbeel, P. Safe exploration in markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, 1451–1458 (Omnipress, Madison, WI, USA, 2012).

[35] Chow, Y., Tamar, A., Mannor, S. & Pavone, M. Risk-sensitive and robust decision-making: a cvar optimization approach (2015). 1506.02188.

[36] Saunders, W., Sastry, G., Stuhlmueller, A. & Evans, O. Trial without error: Towards safe reinforcement learning via human intervention (2017). 1707.05173.

[37] Liu, W., Li, D., Aasi, E., Tron, R. & Belta, C. Interpretable generative adversarial imitation learning (2024). 2402.10310.