Mines ParisTech
Université Paris-Dauphine

Master's thesis

Combining safe learning and learning safety

Author:
Pierre-François Massiani

Supervised by:
Dr Steve Heim
Friedrich Solowjow
Prof. Dr Sebastian Trimpe

Master IASD
Intelligent Control Systems group
Max Planck Institute for Intelligent Systems

September 21st, 2020

Abstract

As the complexity of systems keeps increasing, learning controllers directly from data becomes an appealing alternative to model-based controller design. Ensuring that these controllers behave safely is critical in order to use them on real-world systems. Yet, giving such guarantees is generally hard, and requires a lot of system knowledge: this largely diminishes the interest of model-free approaches such as reinforcement learning (RL). Finding a general, model-free way to provably learn safe behaviours is a key challenge for RL. This thesis examines the relationship between two approaches to this question: penalizing a set of failure states, and constraining the agent to avoid these states.

The first major contribution of this thesis is proving that, as long as the penalty is large enough, penalizing failures gives the same safety guarantees as constraining the agent. Therefore, the two problems are theoretically equivalent and the penalty can be provably scaled with minimal system knowledge.

The second contribution is to demonstrate empirically how this theoretical equivalence can be used to learn safe policies while reducing the number of failures during training.

These two contributions are based on a new theoretical understanding of the safe learning problem enabled by viability theory. We show that learning a policy that avoids a set of failure states boils down to constraining the agent to stay in the largest possible safe set: the viability kernel. The first consequence is that the safe learning problem can be treated as a simple, unconstrained problem in a subset of the state-action space, which is an uncommon property. The second consequence is that the viability kernel provides a general, model-free answer to the question of what staying safe means.

This thesis also includes a third contribution on the impact of parameterization on learning safe policies through penalization. We provide a counterexample showing that the parameterized, optimal policy learned by penalizing failures does not enjoy any safety guarantees in general. We demonstrate that the duality gap between the penalized and the constrained problems is a very poor indicator of the risk. This result weakens the conclusions of recent work [18] where this duality gap is minimized to approximately solve the constrained problem.

Résumé

La complexité grandissante des systèmes dynamiques rend leur modélisation et leur contrôle de plus en plus difficiles. *Apprendre* à contrôler ces systèmes directement à partir de données est donc une alternative intéressante. Pour utiliser de tels contrôleurs sur des systèmes physiques, il est important de s'assurer qu'ils se comportent de façon sûre. Ces garanties sont souvent difficiles à obtenir, et nécessitent de modéliser partiellement le système: cela diminue fortement l'intérêt de méthodes sans modèle telles que l'apprentissage par renforcement. Trouver comment apprendre un comportement sûr sans modèle est donc un défi majeur pour appliquer ce dernier à des systèmes physiques. Dans cette thèse, nous étudions la relation entre deux approches communément opposées: pénaliser un ensemble d'états d'échec, ou contraindre l'agent à ne pas explorer cet ensemble.

La première contribution majeure de ce travail est de prouver que, tant que la pénalité est assez grande, pénaliser l'échec fournit les mêmes garanties de sécurité que de contraindre l'agent à ne pas échouer. Les deux approches sont donc équivalentes, et trouver une valeur pour la pénalité ne nécessite qu'une connaissance minime du système.

Notre seconde contribution est l'utilisation de cette équivalence théorique dans des algorithmes apprenant un comportement sûr tout en minimisant le nombre d'échecs durant la période d'entraînement.

Ces deux contributions utilisent la théorie de la viabilité pour fournir une nouvelle interprétation théorique du problème d'apprentissage d'un comportement sûr. Nous démontrons que ce problème consiste simplement à contraindre l'agent à rester dans le plus grand ensemble d'états sûrs: le noyau de viabilité. Ainsi, un agent sûr n'explore qu'un certain sous-ensemble de l'ensemble des états-actions, ce qui est peu commun. La seconde conséquence est que le noyau de viabilité permet de définir la sécurité d'un agent sans modéliser le système sous-jacent. Il s'agit donc d'un outil très prometteur pour définir et apprendre la sécurité sans modèle.

Cette thèse contient également une troisième contribution portant sur l'influence de la paramétrisation sur l'apprentissage de politiques sûres par pénalisation. Nous développons un contre-exemple montrant que les politiques paramétrées apprises par pénalisation ne bénéficient d'aucune garantie de sécurité en général. De plus, le saut de dualité entre les approches par pénalisation et par contrainte ne permet pas d'inférer le risque de la politique optimale apprise par pénalisation. Ce résultat affaiblit les conclusions de travaux récents [18], où le saut de dualité est minimisé afin de résoudre approximativement le problème sous contraintes.

Contents

Chapter 1

Introduction

1.1 Motivation

How can robots learn how to act safely model-free ?

In the past 30 years, *reinforcement learning* (RL) has found its way into the control engineer's toolbox. Its main promise is the ability to learn controllers without having to model the *dynamical system* at hand. Such models are indeed critical for conventional control approaches, but deriving them from first principles is often hard and requires a lot of domain knowledge. Moreover, the guarantees derived this way are only as good as the underlying models, and controllers that are optimal for the model may perform quite poorly on real-world robots. With the ever-increasing availability of experimental data and computational power, learning optimal controllers directly from that data has quickly become an appealing alternative. While RL has historically been focused on solving tasks in controlled environments like video games, a lot of research effort has recently been put in applying it to real-world systems, where dynamics are often hard to model. As a matter of fact, RL shares a lot of common traits with optimal control [20], with the major difference that the former is model-free and data-driven.

One key advantage traditional approaches still have is the additional *safety* guarantees they come with. Indeed, there is generally no way of knowing *a priori* what *policy* a RL agent learns, and in practice, such policies can behave spectacularly differently from what is expected. While this is usually seen as a feature in optimization-based controller design [4], it can quickly become problematic in the traditional RL framework where the engineer's only degree of freedom[1] is the *reward function*. Designing such a function then becomes a very complex task, since it needs to reflect all safety concerns (so learned controllers are guaranteed safe) without interfering with the function's primary role: specifying the task the agent should learn. There has been some recent interest in guaranteeing the safety of learned controllers when such a function is available [12], [21], [6], but designing it typically requires a lot of system knowledge . This exhibits the paradox of safe RL, where system knowledge is used to craft complex reward functions that are then used to teach a model-free algorithm.

The idea of putting together safety concerns and task specification in the same reward function can also be criticized. Indeed, it seems inappropriate to put on the same scale actions optimizing a process and constraints that should never be

[1] Once the problem is modeled.

violated no matter how much reward this produces. Think of controlling a nuclear power plant for example, where core meltdown should be avoided at all costs: simply penalizing such a state just does not seem right.

A key question that stems from these two considerations is then: is it possible to design agents that can *provably* learn safety constraints without this system knowledge?

1.2 Related work

There are many very different approaches to safety in RL. In this thesis, we define it as avoiding at all times a set of failures, possibly with some non-zero probability[2]. We briefly review here the most common approaches to learning safe controllers with RL.

1.2.1 Constrained Markov decision processes

The classical formulation of RL is based on Markov decision processes (MDPs). Therefore, Constrained Markov decision processes (CMDPs) are the most natural formalization [10] of safety in RL. The core idea is to introduce other metrics of how well the agent is doing, and to constrain them. Mathematically speaking, a CMDP is simply a constrained optimization problem over the set of policies and with the reward function as an objective. The two major questions they raise are: what is a good constraint? How do we solve a CMDP?

Ergodicity-based safety When exploring an environment, an agent may take irreversible actions that impact the set it is able to reach in the future. Such a policy in such an environment is called *nonergodic*. In [16], Moldovan and Abbeel define safety as only following ergodic policies, and formulate and solve a CMDP ensuring that chosen policy is ergodic with at least some probability. This effectively constraints the agent to stay in the *ergodic set* of its initial state. In [21], the authors assume that this set can be described by a Lipschitz safety function and derive a scalable model-based algorithm capable of exploring the whole ergodic set without ever leaving it. While such a definition of safety can be sufficient for a class of applications, it often only yields very conservative policies. Indeed, the ergodic set is in general only a subset of the set of safe states, and an agent may be perfectly fine after taking a non-ergodic action. Another limitation of this definition is that the ergodic set may itself contain undesired states: consider for example a quadcopter bumping against a wall. The quadcopter may be fine after the shock, but we still want to avoid such a situation.

Chance constraints A generalization of this ergodic constraint that can be used to define CMDPs is called *chance constraints*. The user defines a *safe set* (resp. *failure set*), and the agent is constrained to stay inside (resp. outside) that set up to some acceptable *risk*. The main perk of this definition is that it captures very well the definition that we gave of safety, provided that the safe or failure set is defined

[2]Only in this section, where we discuss related work: our own results only apply when safety means having a zero probability of failing.

correctly. The main challenge they pose is that the underlying CMDPs are generally hard to solve. It is also tricky to ensure that such problems are even feasible: it may not be possible to ensure failure from any initial condition, or the considered safe set may not be controllable invariant[3]. Designing safe sets or acceptable sets of initial states generally requires a lot of system knowledge, and is therefore challenging in the model-free setting. This approach has been explored in [18] or [11], and feasibility is generally assumed.

Cost constraints The final type of constraints that we mention here are *cost constraints*. These are the most general type of constraints, since the two previous cases can generally be reformulated in terms of costs. In this setting, the agent collects one or many *costs* at each timestep. It then incurs these costs in a total cost function, whose expected value should be bounded[4]. This problem has been extensively studied [10] [15], both in the model-based and model-free settings, but the biggest milestone it probably the Constrained Policy Optimization [1] algorithm (CPO) by Achiam et al., which is the first scalable algorithm for cost-constrained MDPs suitable for deep RL. To this day, it is however unclear how such algorithms can be extended to methods other than policy gradients [8].

There is no general answer to whether cost constraints give a satisfactory definition of safety. As mentioned, some costs may be interpreted in the setting of chance constraints, but not all costs enjoy such a property. For instance, constraining the minimal expected return [12] or its variance is highly dependent on the reward function, and is therefore not a suitable definition of safety for all problems [10].

Solving CMDPs The main piece of litterature in the field of CMDPs is [2]. In this book, Altman shows the equivalence between a particular type of reward shaping and CMDPs (we will come back to this in Chapter 2), and uses it to solve CMDPs. To this day, Altman's results are still part of the theoretical foundation of most algorithms solving CMDPs[5] [25]. However, most of these approaches are model-based, and often suffer from the curse of dimensionality [15]. Therefore, finding efficient methods to solve CMDPs is still an open problem. Recently, Paternain et al. [18] have explored an approach based on Lagrangian duality exploiting Altman's results. The results presented in this thesis highlight some of the theoretical and practical shortcomings of this work, and were derived independently.

1.2.2 Reward shaping

Although CMDPs are the most natural framework to formalize safety, they are not the most practical way of *enforcing* it. The roboticist's way of communicating its goal to the agent is through the reward function, and making this function safety-aware is called *reward shaping*.

The simplest and most common way of proceeding is to change the instantaneous reward that the agent collects by penalizing *failure states*, that is, states that the agent should not explore. Geibel and Wysotzki [11] propose an algorithm to solve that problem while empirically tuning the penalty. In particular, they emphasize the

[3]This means that no policy can ensure that the agent stays in it.

[4]Constraining other moments of the cost is possible, but the expected value is the most common.

[5]The CPO algorithm is actually one of the few that does not use it.

close theoretical relation between penalizing failures and constraining the probability of failing. This idea is of critical importance in this thesis, and we formalize their empirical approach in Chapter 3. A recent survey on safe RL [10] summarizes the three main critics faced by this approach. First, it is generally considered as a heuristics, and policies learned this way generally are not guaranteed to avoid failure states. Second, such policies - when safe - may be overly pessimistic. We will see later that these concerns are, in fact, not justified. More recently, Paternain et al. [18] have derived a systematic way of scaling that penalty to ensure safety based on Lagrange duality.

1.3 Contributions

The goal of this thesis is to bridge the gap between the chance-constrained formulation of safe RL and penalty methods. Our major theoretical contribution is showing that penalizing failures is a theoretically guaranteed way of learning an optimal, safe policy, and this for an unbounded interval of penalties. In doing so, we provide a theoretical analysis of our CMDP of interest: the reward maximization problem constrained to policies with a 0 probability of failing. We demonstrate that this CMDP can be solved with tools for classical MDPs such as value functions by using tools from viability theory [3]. We then take a first step in using these results by defining and benchmarking algorithms leveraging this new theoretical insight to solve the 0-risk CMDP. In particular, we compare algorithms that enforce the 0-risk constraint directly to penalized methods on two environments presenting different challenges, and compare the safety and the performance of the learned policies.

Outline The rest of the thesis is structured as followed. In Chapter 2, we introduce the formalism of MDPs and CMDPs and *viability theory*, which is critical to formulate our results. Chapter 3 presents our main theoretical contributions: characterization of safe policies with viability theory, the strong duality theorem that connects the cost-constrained formulation with the reward shaping one, and the limitations of that theorem. Finally, we describe and benchmark algorithms leveraging these results in Chapter 4.

Chapter 2

Preliminaries

In this chapter, we introduce the concepts that this work builds upon. We start by presenting in Section 2.1 the well-established RL formalism and its connection with MDPs. We also discuss the main differences between MDPs and CMDPs, as they will happen to be of critical importance to understand the limitations of the penalized methods of Chapter 3. We then introduce viability theory in Section 2.2, which is a very general framework typically used to describe dynamical systems that avoid failure without converging to an equilibrium. This powerful conceptual tool will enable us to formalize our definition of safety in Chapter 3, and will be central for the algorithms derived in Chapter 4.

2.1 Reinforcement learning

Herein, we give a brief introduction to RL. For a complete and thorough overview of the field, we refer the reader to the book [19] by Sutton and Barto. The definitions and concepts presented here can be found in [19, Chapter 3].

The general purpose of RL is learning what to do, that is, to map states to actions. The learner - also called the *agent* - is placed in an *environment*, and its goal is to maximize a *long-term reward*. The catch is that the agent is not told which actions are good: it has to discover this through *trial and error*.

2.1.1 Markov decision processes

The goal of RL is generally formalized through the mathematical model of MDPs, which are the simplest way to describe learning from interactions with an environment.

States, actions, and rewards

The agent interacts with the environment in a sequence of discrete time steps $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent is in a given state $S_t \in \mathcal{S}$ and can pick an action $A_t \in \mathcal{A}$ to transition to the next state, S_{t+1}. While this transition occurs, the agent also collects a scalar *reward* R_{t+1}[1]. Hence, a notion of *trajectory* naturally

[1]In this work, we follow the convention adopted in [19] and use R_{t+1} instead of R_t at time t to emphasize the fact that the reward is determined *after* the action A_t is chosen, jointly with S_{t+1}.

emerges from an agent evolving in an MDP:

$$T = (S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots).$$

The set of all possible trajectories is noted $\mathcal{T} = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^{T_f}$, where T_f is the stopping time of the process and is defined below. What's more, $\mathcal{Q} = \mathcal{S} \times \mathcal{A}$ is called the *state-action set*, or state-action space[2].

Environment and Markov property

In this work, we will restrict ourselves to *finite* MDPs, that is where $\mathcal{S}$ and $\mathcal{A}$ are finite sets. Hence, the random variables describing the successive states, actions and rewards follow well-defined discrete probability distributions. A fundamental assumption in MDPs is that the current state of the agent contains all of the available information about the future: the *history* of the agent - that is, its trajectory up to the current time t - dos not give any additional information. This is formalized in the following assumption:

Assumption 1. *States and rewards satisfy the* Markov property*, that is, for all states* $(s_0, s_1, \ldots, s_t)$ *and actions* $(a_0, a_1, \ldots, a_t)$*, and all* $s' \in \mathcal{S}$ *and* $r \in \mathbb{R}$*:*

$$\mathbb{P}(S_{t+1} = s' | S_0 = s_0, A_0 = a_0, \ldots, S_t = s_t, A_t = a_t) = \mathbb{P}(S_{t+1} = s' | S_t = s_t, A_t = a_t) \tag{2.1}$$

$$\mathbb{P}(R_{t+1} = r | S_0 = s_0, A_0 = a_0, \ldots, S_t = s_t, A_t = a_t) = \mathbb{P}(R_{t+1} = r | S_t = s_t, A_t = a_t) \tag{2.2}$$

This fundamental assumption of MDPs states that the *transition probabilities* to the next state only depend on the current state and the chosen action, and not on the rest of the history of the agent. The function giving the probabilities of the next state s' and the collected reward r given the current state s and action a is called the *dynamics* of the environment:

$$f(s', r|s, a) = \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a). \tag{2.3}$$

If the environment is *stationary* - which we will assume from now on - the dynamics does not depend on time.

The Markov assumption should be understood as a condition on the state rather than on the environment. The state must indeed include all information that make a difference on the future of the agent.

Return function

When interacting with its environment, the agent's goal is to maximize a "long term reward", also called the *return*[3]. Surely, this return should be related to the immediate reward: if the agent has collected the rewards $R_t, R_{t+1}, \ldots$ after the time step t, what is the associated return? To answer this question, we must define an aggregation function that will be the agent's definition of the return.

[2]In all of this work, the words "set" and "space" will be used interchangeably when refering to the sets $\mathcal{S}$, $\mathcal{A}$, or $\mathcal{Q}$.

[3]This should not be mixed up with the reward: the reward is an instantaneous signal that the agent collects, and the return is the way these instantaneous reward are aggregated.

Episodic tasks There are several reasonable definitions for the return. For example, the *total return* is simply the sum of all collected rewards:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_{T_f} = \sum_{u=t+1}^{T_f} R_u, \tag{2.4}$$

where T_f is the final time step[4]. This definition is particularly suited when there is such a notion of final time step. In that case, the interactions between the agent and the environment naturally break into subsequences called *episodes*, and the MDP is called episodic. Such a setting describes, for example, an agent whose goal is to find a flag in an environment or an agent playing a video game. The episode is over when the agent reaches a *terminal state*: the flag is found, the game is over. After an episode ends, the agent is generally reset in the environment and a new episode can start, independently of the outcome of the previous one.

Continuing tasks When the agent-environment interaction does not break into such episodes, the total return is in general not well-suited since it can easily take infinite values. In that case, the task is said to be *continuing*. A simple way-out of these infinite returns is to consider the *discounted return*[5], where the agent discounts the rewards in the future and maximizes the sum of these discounted rewards:

$$G_t = \sum_{u=0}^{\infty} \gamma^u R_{t+u+1}, \tag{2.5}$$

where $0 \leq \gamma < 1$ is the *discount rate*, or *discount factor*. The value of the discount factor defines how much the agent values long-term rewards compared to immediate ones. If $\gamma = 0$, the agent is myopic and will only care about its next immediate reward. Such an agent has no ability for long-term planning. Contrarily, as γ gets close to 1, future rewards are more and more taken into account. The discounted reward model is also popular in other fields such as economy, where it is commonly used to describe interest rates. The fact that it naturally gives less importance to long-term reward can either be seen as a feature - since long-term rewards are generally subject to more uncertainty - or as a shortcoming, as the evaluation it gives of situations often leads to short-term decision making. Whether the discounted reward is adapted to describing the task at hand is another modeling question. Alternatives to this definition exist, but discussing them is beyond the scope of this work.

We define a common notation in order to treat both settings in the same way. The total return of Equation (2.4) can also be written as:

$$G_t = \sum_{u=0}^{T_f} \gamma^u R_{t+u+1}, \tag{2.6}$$

with $\gamma = 1$. Similarly, the discounted return of (2.5) can be written as in Equation (2.6) with $\gamma < 1$ and $T_f = \infty$. Hence, we will take Equation (2.6) as the general definition of the return, and allow $T_f = \infty$ whenever $\gamma < 1$.

[4]In general, T_f is also a random variable since the stopping time is not known in advance. In this work, we will not consider such a technicality and will assume T_f to be constant, although our results can extend to the case where T_f is random.

[5]There are other ways of defining finite returns for continuing tasks (such as the average return), but we will not consider them here.

Initial state

The agent needs to be *initialized* at the beginning of the experiment (or at the beginning of each episode for episodic tasks). The law of the variable S_0 describing the initial state of the agent is a free parameter[6] in MDPs. We call μ the probability distribution of the initial state:

$$\forall s \in \mathcal{S},\ \mathbb{P}(S_0 = s) = \mu(s).$$

The support of μ is the set of states in which the agent can be initialized with a non-zero probability, and is noted:

$$\text{supp}(\mu) = \{s \in \mathcal{S} \mid \mu(s) > 0\}.$$

So far, we do not put any assumption on μ. Such assumptions will become in Chapter 3 in order to ensure the existence of safe policies.

Policies and optimization objective

At each time step t, the agent needs to choose an action. The decision rule that the agent follows is called a *policy*: when in state s, the agent will take the action a with probability $\pi(a|s) \in [0, 1]$. For a given state, a policy determines a probability distribution over the set of actions, and so the following identity holds:

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1.$$

Formally, a policy is just an ordinary function $\pi : \mathcal{Q} \to [0, 1]$, and the space of policies is denoted Π.

The attentive reader may notice that our definition of policies is quite restrictive. Indeed, nothing suggests in what we have already established that the decision rule of the agent should not depend on time, or should only use the information of the *current* state to decide what action to take (compared to using all the available history). Such policies are called *stationary Markov policies*, and [2, Theorem 2.1] states that the optimization problem defined below can be restricted to such policies without loss of generality. In the following, we will only consider stationary Markov policies and refer to them simply as "policies".

The choice of a policy, combined with the dynamics and the distribution μ of the initial state, induces a probability distribution over the set of trajectories. Because of the stochasticity of both the dynamics and the policy, the return itself is a random variable defined on that set of trajectories. Hence, the agent's goal is to maximize the expected value of the return:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \mathbb{E}_\pi \left(G_t \right). \tag{2.7}$$

This is the fundamental optimization objective of MDPs that RL algorithms aim at solving. The main difference with classical optimal control is that, in RL, the objective function cannot be easily evaluated.

[6] In practice, this parameter is not free but is determined by the true system that we model. For example, the initial state is generally fully determined in a video game.

Value functions and Bellman equations

A useful tool for solving Problem (2.7) is *value functions*[7]. The key idea here is to define what the *value of a state* (or a state-action) is under a given policy:

Definition 1. *Let $\pi \in \Pi$. The state and state-action value functions of π are:*

$$V_\pi : s \in \mathcal{S} \mapsto \mathbb{E}_\pi \left[G_t \mid S_t = s \right]. \tag{2.8}$$

$$Q_\pi : (s,a) \in \mathcal{Q} \mapsto \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]. \tag{2.9}$$

According to the stationarity assumption, these functions do not depend on the time. The state (resp. sate-action) value function represents the expected return that the agent will get by following the policy π when in state s (resp. when in state s and after taking action a). The state value function satisfies a recursive identity called the *Bellman equation*[8] [19, Section 3.5]:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} f(s', r|s, a) \left[r + \gamma V_\pi(s') \right], \quad \forall s \in \mathcal{S}. \tag{2.10}$$

The Bellman equation expresses a relation between the value of a state and the value of its successor states. We can think of what it means as a one-step-ahead lookup on the return the agent can collect. In a state s, the agent can take an action a and reach a state s' with reward r. The additional return that can be collected from s' is $V_\pi(s')$: it aggregates all the possible rewards the agent will get when starting again from s'. The Bellman equation discounts this future return that the agent will only enjoy from the next step, and averages all these possible outcomes with their probability of happening: this defines the value of the current state s.

Optimal value functions The power of using value functions becomes clear when defining the *optimal value function*:

Definition 2. *The optimal value functions of the MDP are:*

$$V^* : s \in \mathcal{S} \mapsto \max_{\pi \in \Pi} V_\pi(s) \tag{2.11}$$

$$Q^* : (s,a) \in \mathcal{Q} \mapsto \max_{\pi \in \Pi} Q_\pi(s,a) \tag{2.12}$$

Now, the functions V^* and Q^* are the value functions associated to a policy, π^*. Namely, π^* is the policy that always picks the action with the best value[9] for Q^*:

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \operatorname{argmax}_{a'} Q^*(s,a'), \\ 0, & \text{otherwise.} \end{cases} \tag{2.13}$$

The fact that this policy has V^* and Q^* as value functions is proven in [19, Chapter 4]. And it is now straightforward to see that π^* is an optimal policy for the optimization problem of MDPs (2.7):

$$\mathbb{E}_{\pi^*}(G_t) \leq \max_\pi \mathbb{E}_\pi(G_t) = \sum_s \mu(s) V_\pi(s) \leq \sum_s \mu(s) V_{\pi^*}(s) = \mathbb{E}_{\pi^*}(G_t).$$

[7] Value functions are in fact so convenient that they are often used to define the RL optimization problem (2.7) [19]. We chose not to take this approach because value functions do not easily extend to CMDPs.

[8] The state-action value function also satisfies a Bellman equation.

[9] We assume here that there is only one, but if this is not the case, any such action works.

So, the complicated optimization problem (2.7) can be solved by approximating the state-action value function Q^*, since an optimal policy can simply be derived from it. Most of the famous algorithms for RL like Q-Learning [22], TD methods [19], or SARSA [19], are methods that try to estimate this Q^* function.

Optimal policies Markov decision processes are usually said to have a unique optimal policy [19], which then has to be the greedy policy π^* of the previous paragraph. This property is only true under some assumptions on the initial state distribution μ. For example, this property holds if the agent may be initialized in any state. However, consider the example where a region of the state space cannot be reached because the dynamics forbid it: the behaviour of the policy in that region does not matter, and the MDP has therefore several different optimal policies. This subtlety is particularly important when dealing with safe MDPs, where the goal is that the agent does not reach specific regions of the state space.

2.1.2 Constrained Markov decision processes

While Markov decision processes are commonly used for RL tasks, they are not always the best-suited model. We have seen in the introduction that CMDPs are particularly adapted to describe RL tasks with safety concerns. We provide here a brief introduction to CMDPs. This section is mainly based on [2, Chapter 3]. We will not be as complete in our presentation of CMDPs as we have been for MDPs, and we refer the reader to [2] for a more detailed presentation.

A constrained optimization problem

The mathematical formalism of CMDPs is almost identical to the one of MDPs. In this work, we only consider MDPs constrained by a single scalar cost. A CMDP is therefore defined by its state space $\mathcal{S}$, action space $\mathcal{A}$, dynamics f, reward R, initial state distribution μ, and cost function $c : \mathcal{Q} \to \mathbb{R}$. Similarly to the reward, the cost is a scalar signal that the agent samples at every step. The agent aggregates this cost with the cost discount factor $\bar{\gamma}$. Note that the cost function is different from the reward function: the latter defines the maximization objective, whereas the former defines the constraint.

We say that an agent is solving the CMDP when it is solving the following optimization problem:

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \mathbb{E}_\pi \left[G_t \right], \\ \text{s. t.} \quad & \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \bar{\gamma}^t c(S_t, A_t) \right] \leq d, \end{aligned} \tag{2.14}$$

where $d \in \mathbb{R}$ is the maximal aggregated cost. Note that Problem (2.14) is very similar to Problem (2.7), with the difference that the only policies allowed are the ones that collect less cost than d for the initial state distribution μ.

Example 1 (Chance constraints are a special case of cost-constrained CMDPs)**.** *For example, assume we are given a safe set $\mathcal{S}_{safe} \subset \mathcal{S}$, and a maximal acceptable*

risk δ. Also assume that states outside of $\mathcal{S}_{safe}$ are terminal. We want to constrain the policies to stay in $\mathcal{S}_{safe}$ with probability at least $1-\delta$:

$$\mathbb{P}_\pi\left[\bigcap_{t=0}^{T_f}\{S_t \in \mathcal{S}_{safe}\}\right] \geq 1-\delta.$$

By taking $\bar{\gamma}=1$ and $c(s,a)=\mathbb{1}_{\mathcal{S}\setminus\mathcal{S}_{safe}}(s)$, we have:

$$\mathbb{E}_\pi\left[\sum_{t=0}^{T_f}\bar{\gamma}^t c(S_t,A_t)\right] = 1-\mathbb{P}_\pi\left[\bigcap_{t=0}^{T_f}\{S_t \in \mathcal{S}_{safe}\}\right].$$

Hence, choosing $d=\delta$ gives the desired constraint.

Optimal policies

One of the main differences between standard MDPs and CMDPs is their set of optimal policies. We have seen in Section 2.1.1 that there always exists a deterministic policy that solves an MDP. This came from the strong relationship between optimal policies and the optimal value function. This property is not preserved in CMDP. We give a simple example to understand why below, and refer the reader to [2] for a more detailed explanation.

The main theoretical tool used to study CMDPs is the *occupation measure* of a policy π:

$$\eta_\pi \,:\, (s,a)\in\mathcal{Q} \mapsto \lim_{t\to T_f}\mathbb{E}\left[\frac{1}{t}\sum_{u=0}^{t}\mathbb{1}_{\{s,a\}}(S_t,A_t)\right]. \tag{2.15}$$

The occupation measure intuitively represents the fraction of time spent in state-action (s,a). A policy π can be easily retrieved from its occupation measure η by noting that:

$$\pi(a|s) = \frac{\eta(s,a)}{\sum_{a'\in\mathcal{A}}\eta(s,a')}. \tag{2.16}$$

Hence, a trick similar to what we did for value functions can be used here as well: solve for the optimal occupation measure directly, instead of finding the policy from the complicated problem (2.14). And it turns out that this is much easier a problem, since it boils down to solving a constrained linear program [2, Theorem 3.3], for which lots of methods exist.

Intuitive explanation of stochastic optimal policies Why is there such a difference between MDPs and CMDPs? Why don't CMDPs have deterministic optimal policies in general? Let us provide some intuition on this question based on the example of Figure 2.1. An unconstrained agent - that is, one which does not care about the cost - simply goes to $s=2$ and stays there, collecting a reward of 10 every time. Now, consider a constrained agent with $\bar{\gamma}=0.9$ and $\delta=1$. It has an upper bound on the cost it is allowed to collect. If the agent goes in $s=2$ and stays there, as the unconstrained agent does, its expected discounted cost is $\frac{1}{1-0.9}=10>\delta$: this solution is not feasible. The other deterministic option is to go in $s=1$ and stay there: in this case, the expected cost is simply 0. This is feasible,

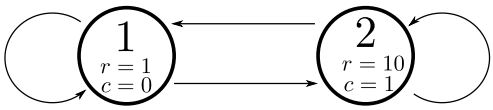


Figure 2.1: Here, r is the reward collected when entering the state, and c is the cost. The unconstrained optimal policy goes and stays in $s = 2$. The constrained one switches between $s = 1$ and $s = 2$ depending on the accepted cost.

but suboptimal. Indeed, the agent could collect more reward by visiting $s = 2$ from time to time, and the frequency of its visit there is bounded by the total expected cost the agent is allowed to collect. Therefore, the optimal constrained agent strikes a balance between the reward-optimal policy of the unconstrained agent and other actions that are less costly and permit an acceptable average cost. On this simple example, we can already infer that this stochastic behaviour should stop when the agent is not allowed to collect *any* cost, that is, when $\delta = 0$. We will see in Chapter 3 that this is indeed the case.

Solving CMDPs Since CMDPs can be expressed as a linear program, they can be solved using tools from linear programming. The main advantage of this is that linear programs can be very efficiently solved. For example, they always satisfy *strong duality*, which allows them to be treated as an unconstrained optimization problem simply by penalizing points that do not satisfy the constraints. Algorithms based on this linear program suffer from two main issues. The first one is scalability: the number of variables in the linear program scales linearly with the number of state-action pairs, and so exponentially with the number of dimensions of the problem. The second problem is that this approach is fundamentally model-based, since the linear program itself depends on the model. Hence, it greatly limits the applications of this result. Recently, Paternain et al. [18] have re-discovered the strong duality result for chance constraints in safe sets, and derive a primal-dual algorithm solving CMDPs in a model-free setting. We believe this approach is very promising: our main theoretical result (Theorem 2) is an extension of their results, and was derived independently. Based on another approach, the CPO algorithm [1] extends policy gradient methods - which are a local optimization scheme for approximately solving the MDP problem (2.7) - to propose the first scalable CMDP solver. This is a milestone in making CMDPs computationally tractable, but this algorithm is hard to generalize to methods other than policy gradients because it provides little understanding of the structure of the underlying problem (contrary to value-based or duality-based methods).

2.2 Viability theory for RL

The definition of safety that we take in Chapter 3 is based on a failure set that the agent should avoid at all times. Forbidding the agent to take steps that immediately result in failure is not sufficient, since it may be in a state where it is doomed to fail without having failed yet. An intuitive example would be a bipedal robot in the process of falling an without any means of controlling its fall, but before it touches the ground: it is already too late. This is a general question not only in RL, but in the field of dynamical systems. An answer to it was formalized by Aubin et al.

in [3] and is called *viability theory*. Recently [14], viability theory has started to be applied to RL problems. We provide here a brief introduction to viability theory in the context of MDPs.

2.2.1 Viability kernel, viable set

Problem setting

In this section, and in the rest of this work, we restrict ourselves to *deterministic* MDPs. That is, the next state is a deterministic function of the previous state and action. With an abuse of notation (from the definition of the dynamics (2.3)), we note:

$$s' = f(s, a), \tag{2.17}$$

where $s' \in \mathcal{S}$ is the successor state of $s \in \mathcal{S}$ when taking action $a \in \mathcal{A}$. We are given a *failure set* $\mathcal{S}_F \subset \mathcal{S}$ assumed to be *absorbing*: all states in the failure set are terminal and the agent needs to be reset after visiting them[10]. This failure set can be defined arbitrarily, and typically contains bad outcomes the agent should avoid.

A toy model: the hovership

We use the toy model developed in [14] to illustrate the concepts. The toy model is described by the transition map of Figure 2.2. Intuitively, it can represent a hovering spaceship affected by gravity, which is stronger near the ground. The ship can apply two level of thrusters, or allow itself to fall. Gravity gets stronger than the thrusters close to the ground. The failure set is $\mathcal{S}_F = \{5\}$, and corresponds to crashing the ship.

Viability kernel and viable set

We can now define the main mathematical object of viability theory [3, Chapter 1]:

Definition 3 (Viability kernel)**.** *The* viability kernel $\mathcal{S}_V \subset \mathcal{S} \setminus \mathcal{S}_F$ *is the maximal set of all states* $s \in \mathcal{S}$*, from which there exists an action that keeps the system inside* $\mathcal{S}_V$*.*

By definition, states outside $\mathcal{S}_V$ have already failed or will fail within finite time. On the hovership example, the viability kernel is $\mathcal{S}_V = \{1, 2, 3\}$. From each of these states, it is possible to avoid failure at all times. Note that the viability kernel is in general not ergodic: it is impossible to go back to states 1 or 2 when in state 3. The viability kernel is a superset of the ergodic set, which is used in [16] and [21]. In fact, the viability kernel is rigorously defined as the largest *controllable invariant*[11] set of states with an empty intersection with the failure set [3].

Following the work of [14], we define the viable set:

Definition 4 (Viable set)**.** *The* viable set $\mathcal{Q}_V \subset \mathcal{Q}$ *is the maximal set of all state-actions* (s, a)*, such that* $s' = f(s, a) \in \mathcal{S}_V$*.*

[10] This work can be extended to the case where this assumption is not true and agents can recover from failure, but this will ease the notations and the reasoning.

[11] A set is controllable invariant if there exists a policy that makes the agent stay in it at all times.

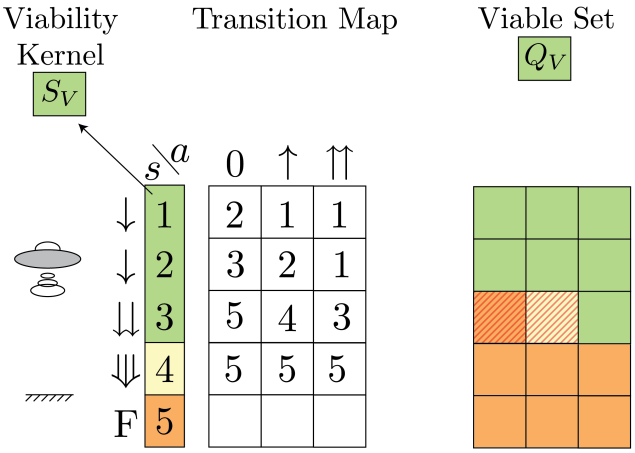


Figure 2.2: The state space and viability kernel $\mathcal{S}_V$ of the toy model (left), its transition map (middle), and viable and critical sets $\mathcal{Q}_V$ and $\mathcal{Q}_C$ (right). On the transition graph, the figures represent the next state when taking the column's action when in the row's state. Both $\mathcal{Q}_V$ and $\mathcal{S}_V$ are highlighted in green. State-action pairs that result directly in failure are highlighted in orange, and the ones that result in unviable states are in yellow. The critical set $\mathcal{Q}_C$ is striped in light red. Source: Adapted from [14]

The viable set naturally extends the viability kernel in state-action space. There exists an intricate relationship between $\mathcal{S}_V$ and $\mathcal{Q}_V$. For example, the viability kernel is empty if, and only of, the viable set is empty. Or similarly: for every state $s \in \mathcal{S}_V$, there exists an action $a \in \mathcal{A}$ such that $(s, a) \in \mathcal{Q}_V$. These simple properties are immediate consequences of the definition, but will play a crucial role. On the hovership example, the viable set is the set of state-actions that end up in states $1, 2$ or 3.

Finally, we define a new notion called the critical set:

Definition 5. *The* critical set $\mathcal{Q}_C \subset \mathcal{Q} \backslash \mathcal{Q}_V$ *is the set of state-actions* $(s, a) \in \mathcal{Q} \backslash \mathcal{Q}_V$ *such that* $s \in \mathcal{S}_V$.

The critical set is the set of state-actions leaving the viability kernel. If the agent visits a state-action in the critical set, it is doomed to fail. Conversely, an agent in the viability kernel has to pass through the critical set in order to go to failure.

We also for convenience the unviable set $\mathcal{S}_U = \mathcal{S} \setminus \mathcal{S}_V$. It is the set of states from which the agent fails within finite time.

2.2.2 A learnable safety measure

We introduce here the notion of $\mathcal{Q}$-safety measure first proposed in [14]. This concept will be of particular interest to derive our algorithms in Chapter 4.

The safety measure is defined on state space as follows:

Definition 6 (Safety measure)**.** *The* safety measure Λ *is the* n*-dimensional volume of the viable set* $\mathcal{Q}_V$*. When applied to a point* $s \in \mathcal{S}$*,* $\Lambda(s) \in \mathbb{R}_{\geq 0}$ *is the measure of the corresponding slice of* $\mathcal{Q}_V$.

This definition is a formalization of the first properties of the viable set that we have highlighted in the previous paragraph: large values of $\Lambda(s)$ mean that there are a lot of actions that keep the agent in the viability kernel. Similarly, saying that $\Lambda(s) = 0$ exactly means that s is not viable[12]. We immediately extend this notion to state-action space by mapping it through the dynamics:

Definition 7 ($\mathcal{Q}$-safety measure)**.** *The* $\mathcal{Q}$-safety measure $\Lambda_{\mathcal{Q}}$ *is defined as:*

$$\Lambda_{\mathcal{Q}}(s, a) = \Lambda[f(s, a)], \quad \forall (s, a) \in \mathcal{Q}.$$

The $\mathcal{Q}$-safety measure can be seen as a one-step look-ahead of the safety measure. It answers the question: "what is the measure of the state I end up in by taking this action ?". The $\mathcal{Q}$-safety measure already takes the dynamics into account. An agent that knows this measure is able to always avoid failure (although this agent may be very bad at maximizing the return), by only following actions with a non-zero measure.

Learning the $\mathcal{Q}$-safety measure

The $\mathcal{Q}$-safety measure is generally unknown, even in the model-based setting. Indeed, computing it is equivalent to computing the viability kernel which is a task known to be computationally expensive [3]. Heim et al. [14] propose an algorithm to learn the $\mathcal{Q}$-safety measure from data by modeling it with a Gaussian process (GP). This algorithm will be central for the ones we present in Chapter 4, so we briefly describe it here.

Gaussian processes as function approximators Gaussian processes are a powerful and adaptable machine learning tool commonly used to model unknown functions from data. More specifically, the prediction of a GP is composed of a *mean* - which is often taken as the prediction - and a *covariance* - which expresses how confident the GP is about the value it gives. For a more thorough explanation of GPs, we refer the reader to [23].

Probabilistic level sets Given an approximation $\hat{\Lambda}_{\mathcal{Q}}$ of the safety measure, the authors of [14] derive two estimates of the viable set: the *optimistic set* $\mathcal{Q}_{\text{opt}}$, and the *cautious set* $\mathcal{Q}_{\text{caut}}$. They are parameterized by the three scalars $\gamma_{\text{opt}}, \gamma_{\text{caut}}$, and λ_{caut}, that can be interpreted as follows: for any state-action (s, a) in $\mathcal{Q}_{\text{opt}}$ (resp. $\mathcal{Q}_{\text{caut}}$), the safety measure is greater than 0 (resp. λ_{caut}) at (s, a) with probability γ_{opt} (resp. γ_{caut}). Hence, these two sets can be interpreted as different estimates of the viable set $\mathcal{Q}_V$ corresponding to difference confidence intervals. The parameters are chosen such that $\mathcal{Q}_{\text{caut}} \subset \mathcal{Q}_{\text{opt}}$.

Iterative estimation of the measure The safety measure is estimated iteratively as follows [14]:

- Input: initial state s_0, initial safety measure $\hat{\Lambda}_{\mathcal{Q}}$;
- Repeat until convergence, with i the number of steps:

[12] In the finite setting. Indeed, there may exist unmeasurable sets of unviable actions in the infinite setting, but we will stay in the finite setting here.

1. Evaluate the probabilistic level sets $\mathcal{Q}_{\text{opt}}$ and $\mathcal{Q}_{\text{caut}}$from $\hat{\Lambda}_{\mathcal{Q}}$;
2. Choose an action a to take:
   - If an action is available in $\mathcal{Q}_{\text{caut}}$ from the current state, then pick such an action that maximizes the information gain[13];
   - Otherwise, pick the action that maximizes the probability of that action being in $\mathcal{Q}_{\text{caut}}$;
3. Take that action: end up in state s_{i+1};
4. If the agent failed: add $((s_i, a_i), 0)$ to the dataset of $\hat{\Lambda}_{\mathcal{Q}}$, and reset the agent;
5. Otherwise, compute the state measure estimate $\hat{\Lambda}(s_{i+1})$ from $\mathcal{Q}_{\text{opt}}$, and add $((s_i, a_i), \hat{\Lambda}(s_{i+1}))$ to the dataset of $\hat{\Lambda}_{\mathcal{Q}}$.

This algorithm has been able to correctly learn the viable sets of the examples of [14].

[13]The covariance of the GP

Chapter 3

A viability-based characterization of safety

This chapter defines the problem of learning policies achieving a 0 risk and demonstrates the main contribution of this thesis: safe, optimal policies can be learned by penalizing failures, and this with an unbounded interval of possible values for the penalty. To prove this result, we first provide a characterization of safe policies as the ones that give a 0 probability of leaving the *viability kernel*. This property is the fundamental reason that enables us to treat the CMDP we consider as a classical MDP on a subset of the state-action space. We then state our main result, and discuss in Section 3.4 how parameterizing policies breaks the safety guarantees that the solution to the penalized problem has.

3.1 Problem setting

We consider a deterministic Markov decision process $(\mathcal{S}, \mathcal{A}, f, R)$ with unknown dynamics and a finite state-action space $\mathcal{Q} = \mathcal{S} \times \mathcal{A}$. The trajectories and the return are defined as:

$$S_{t+1} = f(S_t, A_t) \tag{3.1}$$

$$S_0 \sim \mu, \tag{3.2}$$

$$G_t = \sum_{u=0}^{T_f} \gamma^u R_{t+u}, \tag{3.3}$$

where $\gamma < 1$ if $T_f = \infty$.

We are given a failure set $\mathcal{S}_F \subset \mathcal{S}$ composed of states that the agent should not explore. The failure set is assumed to be absorbing[1]: the agent needs to be reset after visiting $\mathcal{S}_F$.

The combination of the failure set and the dynamics naturally give rise to the viability kernel $\mathcal{S}_V$,the viable set $\mathcal{Q}_V$ and the critical set $\mathcal{Q}_C$ of the system. We assume that $\mathcal{S}_V$ and $\mathcal{Q}_V$ are not empty: if they are, any policy reaches the failure set in finite time, and so there are no safe policies. In order for safe policies to exist, we also assume that $\text{supp}(\mu) \subset \mathcal{S}_V$, where μ is the initial state distribution: we explain why in Section 3.2.2.

[1]This assumption is not critical to the results presented here, but it eases their formulation.

We can now define safe policies as in [14]:

Definition 8 (Safe policy)**.** *A policy* $\pi \in \Pi$ *is* safe *if:*

$$\mathbb{P}_\pi \left[\bigcup_{t=0}^{T_f} \{S_t \in \mathcal{S}_F\} \right] = 0. \tag{3.4}$$

This definition of safety belongs to the broad class of *chance constraints*, which is a standard way of defining safety in RL [11] [18]. In our case, this chance constraint is sharp, in the sense that it does not allow any probability of visiting the failure set. This is quite restrictive, and the existence of policies satisfying it is not trivially clear.

The safe learning problem can now be stated as finding a safe policy that maximizes the expected return:

$$\begin{aligned} \underset{\pi \in \Pi}{\text{maximize}} \quad & \mathbb{E}_\pi \left(G_t \right), \\ \text{s.t.} \quad & \mathbb{P}_\pi \left[\bigcup_{t=0}^{T_f} \{S_t \in \mathcal{S}_F\} \right] = 0. \end{aligned} \tag{3.5}$$

This optimization problem can be reformulated as a CMDP with a cost constraint. This is a simple consequence of the following lemma, whose proof is elementary yet insightful:

Lemma 1. *For any policy* $\pi \in \Pi$*, it holds that:*

$$\pi \text{ is safe} \iff \rho_\pi(\mu) \doteq \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t \mathbb{1}_{\mathcal{S}_F}(S_{t+1}) \right] = 0. \tag{3.6}$$

The quantity $\rho_\pi(\mu)$ is called the *discounted risk* of the policy [11]. It mainly depends on the dynamics and the initial state distribution μ. In the following, we will only write it ρ_π and drop the explicit dependency in μ.

Proof. It is clear that a policy π is safe if, and only if, for all $t, \mathbb{P}_\pi(S_t \in \mathcal{S}_F) = 0$. Moreover, it is also clear that for all t, we have:

$$\mathbb{P}_\pi(S_t \in \mathcal{S}_F) = \mathbb{P}_\pi(\mathbb{1}_{\mathcal{S}_F}(S_t) = 1) = \mathbb{E}_\pi \left[\mathbb{1}_{\mathcal{S}_F}(S_t) \right],$$

where the last equality holds because $\mathbb{1}_{\mathcal{S}_F}(S_t)$ is a Bernoulli variable.

Assume π is safe. By the aforementioned properties, we have that:

$$\forall t, \mathbb{E}_\pi \left[\mathbb{1}_{\mathcal{S}_F}(S_t) \right] = 0.$$

Summing all of these equalities for $t \geq 1$ and multiplying them by γ^t shows that $\rho_\pi = 0$.

Conversely, assume that $\rho_\pi = 0$. Since all the variables in the definition of ρ_π are nonnegative, it holds that $\forall t \geq 1, \mathbb{E}_\pi \left[\mathbb{1}_{\mathcal{S}_F}(S_t) \right] = 0$. This entails that for all $t \geq 1, \mathbb{P}_\pi(S_t \in \mathcal{S}_F) = 0$. This property also holds for $t = 0$, since $\text{supp}(\mu) \subset \mathcal{S}_V$. Therefore, π is safe. □

So, problem (3.5) can be reformulated as:

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \mathbb{E}_\pi \left(G_t \right), \tag{3.7}$$

$$\text{s.t.} \quad \rho_\pi = 0. \tag{3.8}$$

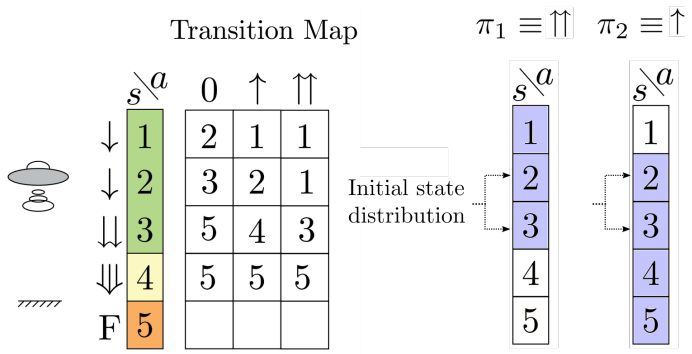


Figure 3.1: The reach of two policies on the hovership example. The agent is initialized in $s = 2$ or $s = 3$ with equal probability. The policy π_1 always applies one thruster, and the policy π_2 always applies two thrusters. The reach is shown in light blue. Note that π_2 is safe, since $\text{Reach}_{\pi_2} \subset \mathcal{S}_V$, whereas π_1 is not.

Guarantees during training In this work, we do not focus on solving problem (3.7)–(3.8) while guaranteeing that constraint (3.8) is satisfied during training, which is hopeless in the model-free setting. Instead, we provide a practical characterization of safety and use it to show that penalized methods asymptotically find safe optimal policies. In Chapter 4, the algorithms we propose learn the constraint (3.8) in a sample-efficient manner.

3.2 Safe policies stabilize the viability kernel

Because our definition of safety puts a sharp constraint on policies - no probability mass is allowed on failing, safe policies are the ones that stabilize the viability kernel, as explained in this section. This result of viability theory has an interesting consequence in RL: it provides a simple characterization of safe policies.

3.2.1 Reach of a policy

We define here the reach of a policy:

Definition 9 (Reach)**.** *The* reach *of a policy* $\pi \in \Pi$ *is the set* $\text{Reach}_\pi(\mu) \subset \mathcal{S}$ *defined as:*

$$\text{Reach}_\pi(\mu) = \{s \in \mathcal{S} \mid \mathbb{P}_\pi(\exists t, S_t = s) > 0\}\,. \tag{3.9}$$

The reach of a policy is the set of states that are reached with a nonzero probability by following this policy. An illustration of this object is given in Figure 3.1. Like the risk ρ_π, the reach largely depends on the initial state distribution[2]. The reach as we define it is different from the standard notion of *forward reachable set* [5], which is the set of states that can be reached with a well-chosen control input. As a matter of fact, the reach of a policy is a subset of the forward reachable set of the support of the initial state distribution. Note that, from the first definition of safety, a policy π is safe if and only if $\text{Reach}_\pi \cap \mathcal{S}_F = \emptyset$.

[2]In the following, we will not explicitly write this dependency and will use the notation Reach_π.

3.2.2 Safe policies characterization

This enables us to state the following theorem:

Theorem 1 (Characterization of safe policies)**.** *For any policy $\pi \in \Pi$, it holds that:*

$$\pi \text{ is safe} \iff \text{Reach}_\pi \subset \mathcal{S}_V. \tag{3.10}$$

Proof. This theorem is a direct consequence of the definition of the viability kernel as the largest controllable invariant set $K \subset \mathcal{S}$ such that $K \cap \mathcal{S}_F = \emptyset$ [3, Definition 4.1.1].

Assume π is safe. Then, it holds that $\text{Reach}_\pi \cap \mathcal{S}_F = \emptyset$. Moreover, Reach_π is invariant under the policy π. Indeed, let $s \in \text{Reach}_\pi$ and $a \in \mathcal{A}$ such that $\pi(a|s) > 0$: the action a is a possible action in state s for the policy π. We show that $s' = f(s,a) \in \text{Reach}_\pi$. Since $s \in \text{Reach}_\pi$, there exists t such that $\mathbb{P}_\pi(S_t = s) > 0$. By the law of total probability, we also have $\mathbb{P}_\pi(S_{t+1} = s') \geq \mathbb{P}_\pi(S_t = s \text{ and } A_t = a) = \pi(a|s) \cdot \mathbb{P}_\pi(S_t = s) > 0$. Therefore, $s' \in \text{Reach}_\pi$. Consequently, Reach_π is a controllable invariant set with an empty intersection with the failure set: by the aforementioned definition of $\mathcal{S}_V$, it holds that $\text{Reach}_\pi \subset \mathcal{S}_V$.

Conversely, assume that $\text{Reach}_\pi \subset \mathcal{S}_V$. Then, π is trivially safe: any $s \in \mathcal{S}_F$ satisfies $s \notin \text{Reach}_\pi$, and therefore $\mathbb{P}_\pi(\exists t, S_t = s) = 0$, which concludes the proof. □

We can restate this theorem in a more practical way by using the critical set:

Corollary 1. *For any policy $\pi \in \Pi$, it holds that:*

$$\pi \text{ is safe} \iff \forall\, (s,a) \in \mathcal{Q}_C \cap (\text{Reach}_\pi \times \mathcal{A}),\ \pi(a|s) = 0. \tag{3.11}$$

Proof. Let $\pi \in \Pi$. Assume π is safe, and consider $(s,a) \in \mathcal{Q}_C \cap (\text{Reach}_\pi \times \mathcal{A})$. We show that $\pi(a|s) = 0$. First, since $s \in \text{Reach}_\pi$, there exists a time t for which $\mathbb{P}_\pi(S_t = s) > 0$. Moreover, since $(s,a) \notin \mathcal{Q}_V$, then Theorem 1 ensures that $s' = f(s,a) \notin \text{Reach}_\pi$, since $s' \notin \mathcal{S}_V$. Therefore, $\mathbb{P}_\pi(S_{t+1} = s') = 0$. By the law of total probability, we also have $\mathbb{P}_\pi(S_{t+1} = s') \geq \pi(a|s) \cdot \mathbb{P}(S_t = s)$. Combining this with $\mathbb{P}_\pi(S_t = s) > 0$ and $\mathbb{P}_\pi(S_{t+1} = s) = 0$ yields $\pi(a|s) = 0$.

Conversely, assume that π is not safe. Then, $\text{Reach}_\pi \not\subset \mathcal{S}_V$: let $s \in \text{Reach}_\pi \setminus \mathcal{S}_V$. We find an action $a \in \mathcal{A}$ such that $\pi(a|s) > 0$ and $(s,a) \in \mathcal{Q}_C$. Consider a trajectory $(s_0, a_0, s_1, a_1, \ldots)$ such that:

- there exists a time t such that $s_t = s$;
- for all $u < t, \pi(a_u|s_u) > 0$.

Such a trajectory exists since $s \in \text{Reach}_\pi$. Consider the set $\mathcal{U} = \{u \in \mathbb{N} \mid s_u \notin \mathcal{S}_V\}$. This set is not empty, so it has a smallest element $u = \min \mathcal{U}$. Since $s_0 \in \text{supp}(\mu)$ and $\text{supp}(\mu) \subset \mathcal{S}_V$, we have $u \geq 1$. We also have $u \leq t$. Now, we show that $(s_{u-1}, a_{u-1}) \in \mathcal{Q}_C$. Indeed, $s_u \notin \mathcal{S}_V$, by definition of u. Yet, $s_{u-1} \in \mathcal{S}_V$, still by definition of u. It follows from the definition of $\mathcal{Q}_C$ that $(s_{u-1}, a_{u-1}) \in \mathcal{Q}_C$. Since we also clearly have $(s_{u-1}, a_{u-1}) \in \text{Reach}_\pi \times \mathcal{A}$, we have found $(s_{u-1}, a_{u-1}) \in \mathcal{Q}_C \cap (\text{Reach}_\pi \times \mathcal{A})$ such that $\pi(a_{u-1}|s_{u-1}) > 0$. This proves that π is not safe implies the negation of equation (3.11), and concludes the proof. □

Discussion

Safety does not constrain the policy everywhere Corollary 1 shows that safety does not constrain the policy everywhere. Indeed, it makes no statement on how actions should be picked in states that cannot be reached. Of course, the condition of equation (3.11) can still be enforced on the whole critical set $\mathcal{Q}_C$, but this is only a sufficient condition for safety.

A classical MDP in a subset of $\mathcal{Q}$ Policies that satisfy the safety constraint are exactly the ones that effectively prevent the agent from exploring a whole region of the state-action space (namely, $\mathcal{Q} \setminus \mathcal{Q}_V$). Moreover, any time the agent picks an action in $\mathcal{Q}_V$, it ends up in $\mathcal{S}_V$again: the viability kernel is *positively invariant* under safe policies. Hence, it is possible to see the safety-constrained CMDP (3.5) as a classical MDP with a modified state space $\mathcal{S}_V$ and a state-dependent action space $\mathcal{A}(s) = \{a \in \mathcal{A} \mid (s, a) \in \mathcal{Q}_V\}$. This consideration hints at the fact that most tools specific to MDPs (such as value functions) can actually be used for this CMDP. The main point of Section 3.3 is to formalize this intuition.

Safety and initial state distribution The existence of safe policies is largely conditioned by the initial state distribution μ. Its support $\text{supp}(\mu)$ must indeed be included in the viability kernel for safe policies to exist, since we always have $\text{supp}(\mu) \subset \text{Reach}_\pi$. This formalizes the intuition that safety cannot be ensured if the agent is initialized in unviable states. If this condition is not satisfied, then the definition of safety needs to be changed, for example by allowing a non-zero probability of failure.

The viability kernel is a natural safe set Some traditional definitions of safety as a chance constraint either rely on a safe set [18] or on a set of states from which the agent should avoid failure [11]. These sets are typically chosen based on system knowledge, and correspond to states from where it is known that a safe behaviour exists. As it can be seen in the examples of [18] and [11], the underlying goal is still to avoid a failure set, whether it is explicitly named or not. Theorem 1 establishes that, once the failure set is defined, the viability kernel is a natural choice for such a safe set. The role of the engineer in defining what safety is is therefore minimized: the failure set is generally an intrinsic property of the task or the system, and the definition of safety naturally emerges from the dynamics [13]. And even if the viability kernel is generally unknown, it can be learned [14].

3.3 Theoretical guarantees on reward shaping

The characterization of safe policies given by Corollary 1 enables us to prove that policies learned through failure penalization are safe and optimal for the constrained problem. It is already known [2], [18] that there always exists a value for the penalty so this policy learned policy is safe and optimal. However, finding this value is highly non-trivial, and often relies on specialized *primal-dual* algorithms [9] [18]. We show here that the set of penalties ensuring optimality and safety for the original problem contains an upper-unbounded interval. This result is stronger than what can be found in [2] or [18], and was derived independently.

3.3.1 The penalized problem

Formulating RL algorithms to solve problem (3.7)–(3.8) is generally hard (see Section 2.1.2). Thus, safety constraints are often enforced in practice by solving the *penalized problem* instead:

$$\underset{\pi\in\Pi}{\text{maximize}} \quad \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t \left(R_{t+1} - p \cdot \mathbb{1}_{\mathcal{S}_F}(S_{t+1}) \right) \right], \tag{3.12}$$

where $p \in \mathbb{R}$ is the *penalty*. The penalized problem is a variant of the original MDP problem (2.7) where the reward function has been changed: a penalty is subtracted to the reward when visiting failure states. The penalized problem is unconstrained, and for $p = 0$, it boils down to the original MDP problem (2.7).

Since the penalized problem is unconstrained, we can define its optimal value function (equation (2.11)):

$$\forall p \in \mathbb{R},\ V_p \,:\, s \in \mathcal{S} \mapsto \max_\pi \ \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t \left(R_{t+1} - p \cdot \mathbb{1}_{\mathcal{S}_F}(S_{t+1}) \right) \right]. \tag{3.13}$$

The penalized problem can be reformulated as follows:

$$\underset{\pi\in\Pi}{\text{maximize}} \quad \mathbb{E}_\pi \left[G_t \right] - p \cdot \rho_\pi. \tag{3.14}$$

The effect of penalizing agent failures is thus to put a cost on unsafe policies, and the value of p represents "how much" it costs to break that constraint. By doing so, we hope that the agent will not only learn a safe policy, but also that this policy is optimal for the original constrained problem (3.7)–(3.8). Finding a value of p such that this is true is called *penalty scaling*. It is underlined in [10] that there is in general no guarantee that the penalty can be scaled. Indeed, the policy learned by solving (3.14) may be unsafe, or on the contrary overly conservative: too high a penalty may result in suboptimal behaviour. The next section shows that such concerns are, in fact, not justified.

3.3.2 The penalized problem solves the constrained one

We first introduce the constrained value function before stating our main theoretical result.

Constrained value function

We can also define a optimal value functions for the CMDP (3.7)–(3.8) as follows:

Definition 10. *The* constrained value functions *of the problem* (3.7)–(3.8) *are:*

$$V^c \,:\, s \in \mathcal{S}_V \quad \mapsto \max_\pi \quad \mathbb{E}_\pi \left[G_t \mid S_t = s \right], \tag{3.15}$$

$$\text{s.t.} \quad \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t \mathbb{1}_{\mathcal{S}_F}(S_{t+1}) \,\middle|\, S_t = s \right] = 0, \tag{3.16}$$

$$Q^c \,:(s,a) \in \mathcal{Q}_V \mapsto \quad V^c \left[f(s,a) \right]. \tag{3.17}$$

Domains of definition The constrained state value function is defined on $\mathcal{S}_V$ only. Indeed, the constraint in the definition of the function is not feasible if $s \notin \mathcal{S}_V$: this is a simple consequence of Theorem 1 in the case where $\text{supp}(\mu) = \{s\}$. The constrained state-action value function is also well-defined, since any state-action in $\mathcal{Q}_V$ maps in $\mathcal{S}_V$.

Interpretation Let us anticipate a bit on Theorem 2 and provide intuition on constrained value functions. They have exactly the same interpretation as optimal value functions for unconstrained MDPs: the value of a state (resp. state-action) represents the remaining return that can be collected from that state (resp. state-action) by following an optimal policy.

Remark 1. *As we emphasized it in Section 2.1.2, CMDPs cannot be solved in general by value methods. Hence, constrained value functions are not a standard tool to study CMDPs. The definition of safety as a* 0*-risk chance constraint makes the CMDP* (2.14) *behave* as if *it were a standard MDP evolving in a subset of the state-action space (namely,* $\mathcal{Q}_V$*), as stated by Corollary 1. This very peculiar property makes constrained value functions relevant here.*

Remark 2 (Conditioning of the constraint)**.** *It can be argued that the constraint 3.16 should not be conditioned on* $\{S_t = s\}$*, since the resulting constraint is less demanding than* (3.8)*. This is true and general. However, there is no universal definition of constrained value functions to the best of our knowledge, so we choose the definition that allows an intuitive interpretation.*

Penalized policies are safe and optimal

We are now ready to state the two main results of this section.

Theorem 2. *The following two conditions hold:*

$$\exists p^* \in \mathbb{R},\ \forall p > p^*,\ \forall s \in \mathcal{S}_V, \qquad V_p(s) = V^c(s), \tag{3.18}$$

$$\forall s \in \mathcal{S} \setminus \mathcal{S}_V,\ V_p(s) \xrightarrow[p\to\infty]{} -\infty. \tag{3.19}$$

The proof of Theorem 2 can be found in Appendix A. The following corollary makes explicit one of the main interests of this theorem by stating that solving the penalized problem with any $p > p^*$ gives optimal, safe policies.

Corollary 2. *There exists a threshold* $p^* \in \mathbb{R}$ *such that, for all* $p > p^*$ *any solution of the penalized problem* (3.12) *is safe and optimal for the constrained problem* (3.5)*.*

Proof. Let p^* be a threshold as given by Theorem 2. Since $\mathcal{Q}$ is finite, we can assume without loss of generality that:

$$\forall p > p^*,\ \max_{s\in\mathcal{S}\setminus\mathcal{S}_V} V_p(s) < \min_{s\in\mathcal{S}_V} V^c(s). \tag{3.20}$$

For all $p \in \mathbb{R}$, let π_p be an optimal solution of the penalized problem (3.12). We first show that, for all $p > p^*$, the policy π_p is safe. From the established theory of MDPs [19], we know that π_p satisfies the optimal Bellman equation on the states it can reach. Hence, for all $s \in \text{Reach}_{\pi_p}$:

$$\pi_p(a|s) > 0 \iff a \in \operatorname*{argmax}_{a'} V_p(f(s,a')), \tag{3.21}$$

By combining equations (3.18) and (3.21)–(3.20), we get that:

$$\forall (s,a) \in \mathcal{Q}_C \cap (\text{Reach}_{\pi_p} \times \mathcal{A}),\ \pi_p(a|s) = 0,$$

which means that π_p is safe according to Corollary (1).

Now, the following holds:

$$\begin{aligned}
\mathbb{E}_{\pi_p}[G_t] &= \mathbb{E}_{\pi_p}[G_t] - p \cdot \rho_{\pi_p}, && \text{since } \rho_{\pi_p} = 0, \\
&= \max_{\pi} \mathbb{E}_{\pi_p}[G_t] - p \cdot \rho_{\pi_p}, && \text{since } \pi_p \text{ is optimal}, \\
&\geq \max_{\pi} \mathbb{E}_{\pi_p}[G_t], && \text{by the weak duality lemma 2}, \\
&\quad \text{s.t. } \rho_\pi = 0, && \\
&\geq \mathbb{E}_{\pi_p}[G_t], && \text{since } \pi_p \text{ is safe}.
\end{aligned} \tag{3.22}$$

Consequently, all of the inequalities are equalities, and so π_p achieves the optimal value of the constrained problem. This concludes the proof. □

Scaling the penalty is easy This result is stronger than what is found in the literature. In [2] or [18], the only guarantee is that there exists a penalty for which the penalized policy is safe and optimal. The fact that *any* penalty greater than p^* proves that it is unnecessary to use primal-dual algorithms that also optimize on the value of the penalty: any high enough value ensures both safety and optimality.

Scaling the penalty is still hard in practice The assumptions of Theorem 2 are simple, but critical for its guarantees to hold. These assumptions are the fact that safety means having a 0 probability of failing, and that the optimization problem is solved on the set of *all* policies. When allowing for a non-zero risk of failure, there still exists a scaling of the penalty for which the penalized problem (3.12) has the same optimal value than the constrained problem (3.5)[3] [18]. However, it is unclear whether *any* policy optimal for the penalized problem is safe: the only guarantee is that it achieves the optimal constrained return. Similarly, the guarantees given by Theorem 2 do not hold when parameterizing the set of policies. Then, there exists in general no finite value of the penalty that ensures that the penalized optimal policy is safe: this is discussed more thoroughly in Section 3.4.

Bounds on the minimal value of p^* The theorem does not provide any information on what is the minimal value of p^*. It turns out that upper bounds can be given: for example, it can be shown[4] that in the case of a deterministic reward R, the quantity:

$$\frac{1}{\gamma^{|\mathcal{S}\setminus\mathcal{S}_V|}} \max_{(s,a)\in\mathcal{Q}\setminus\mathcal{Q}_V} R(s,a) + (1-\gamma) \min_{(s,a)\in\mathcal{Q}_V} R(s,a),$$

is a value of p^* that satisfies Theorem 2. This threshold grows as γ gets close to 0. This makes a lot of sense: as γ goes to 0, the agent is more and more oblivious to long-term rewards. Hence, the penalty needs to be scaled accordingly.

[3]For this result to hold rigorously, the objective of the penalized problem needs to be modified by adding $\delta \cdot p$, where δ is the risk, so it can be interpreted as the dual of the constrained problem.

[4]This is just given as an example and is not proven here.

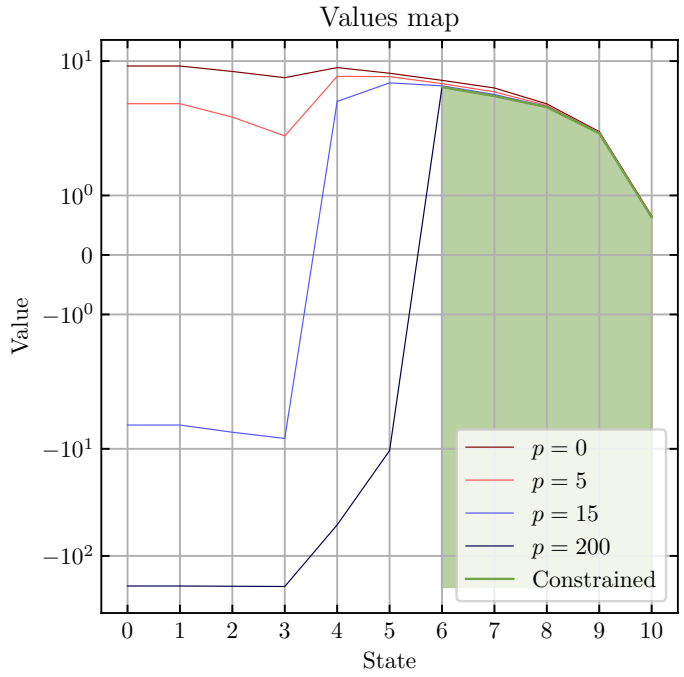


Figure 3.2: The optimal value function for different values of the penalty on the hovership example. The viability kernel is $\mathcal{S}_V = \{6, 7, 8, 9, 10\}$ and is highlighted in green. The constrained value function is plotted in green, and is equal to the penalized functions for the biggest penalties. The policy for $p = 15$ is not safe, since $V_{15}(5) > V_{15}(6)$ and $s = 5$ is not viable. The different values we tested for the penalty indicates that the lowest threshold p^* is somewhere between 15 and 200.

Influence of the penalty on the learning Whether or not an agent trained on the penalized reward finds the optimal policy (which is safe, according to Theorem 2) largely depends on what learning algorithm is used. In particular, high values of the penalty may break some algorithms by making them numerically unstable, or very sample-inefficient. Since theoretical guarantees hold as long as $p > p^*$, the algorithm's sensitivity to the penalty should be the main consideration to choose p. According to the numerical proof of concept that we present below, Q-Learning does not seem to be very sensitive to the value of p: the number of iterations required to find the optimal policy on does not noticeably vary with p.

Application to the hovership example

We describe here a numerical experiment showing how Theorem 2 and Corollary 2 can be used in practice. The environment is a variation of the hovership presented in Section 2.2.1 with larger (finite) state and action spaces. The reward is an affine function of the altitude: it is 10 when the altitude is minimal, and 10 when it is maximal. This incentivizes the agent to go as low as possible. The discount rate was set to $\gamma = 0.3$. We chose such a low value for γ to emphasize that the value of p^* can grow very quickly depending on how the problem is parameterized.

Solving the penalized problem The algorithm used to solve the penalized problem is Q-Learning [22], with an ε-greedy policy with an exploration parameter $\varepsilon = 0.1$ and a step size of 0.6.

Solving the constrained problem We also solved the constrained problem with an iterative method to illustrate how the constrained problem can be solved directly by only looking for policies that satisfy the constraint in Corollary 1. The algorithm used is a modified version of Q-Learning with an ε-greedy exploration policy:

- with probability $1 - \varepsilon$: the agent picks the action that maximizes its current estimate of Q^c;
- the agent picks any other *viable* action with uniform probability.

Note that this is exactly the Q-Learning algorithm where the set of actions is constrained to viable actions. We call this algorithm "constrained Q-Learning". Of course, it requires the viable set to be known.

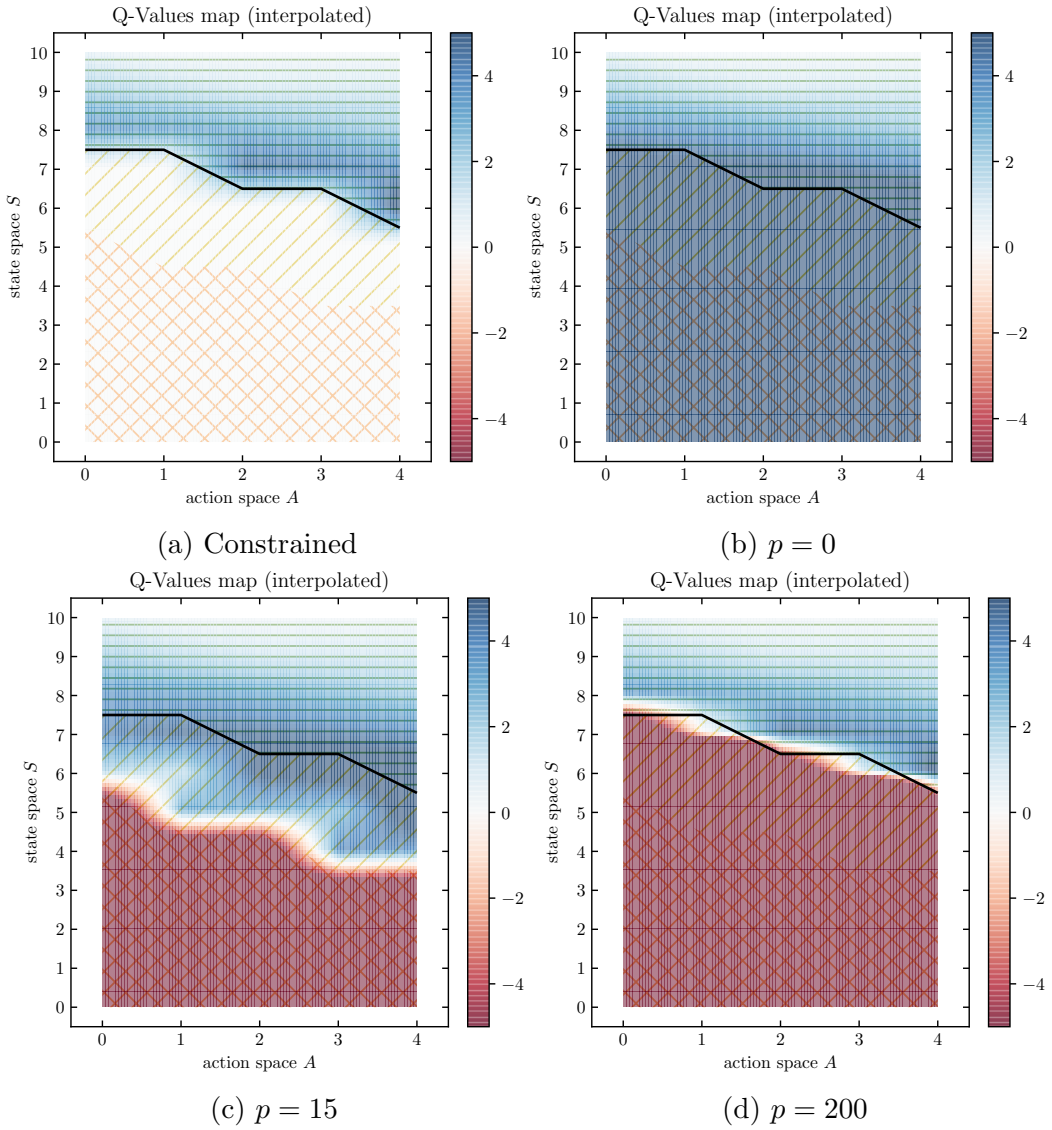
Results As we increase the penalty, the values of all states decrease (see Figure 3.2). When the penalty exceeds the lowest threshold p^*, the penalized values in the viability kernel stop decreasing and stabilize at *exactly* the constrained values. Outside of the kernel, the values go arbitrarily low. The resulting policies can be examined on Figure 3.3: for a given value map, the corresponding policy is the one that takes the action with maximal value in every state. For high values of the penalty, this policy stays in $\mathcal{Q}_V$.

3.4 The price of parameterization

In many of the real-world applications of RL, the dimensionality of the state-action space is intractable. Because of that, finding an optimal policy becomes an impossible task, since it would require not only lots of computational power and memory, but also unreasonable amounts of data. This last point is critical when applying RL to dynamical systems, where collecting data is risky and expensive. For these reasons, we restrict the policies to a set of *parameterized policies*, and only look for the parameters that maximize the expected return. This reduces the problem's dimensionality while reducing the set of possible solutions. An important question is then: do the guarantees on penalty scaling still hold when parameterizing the policies? Is it still reasonable to solve the easier penalized problem to learn safe policies? The short answer is no, and Section 3.4.1 gives a counterexample showing it. In Section 3.4.2, we emphasize on another example that the resulting *duality gap* between the penalized and the constrained problem is a very poor indicator of whether the policy learned by penalized methods is close to being safe.

3.4.1 Penalized parameterized policies are not safe

In this section, we show that penalized optimal policies are in general not safe for any *finite* value of the penalty.



(a) Constrained (b) $p = 0$

(c) $p = 15$ (d) $p = 200$

Figure 3.3: The optimal state-action value function for the constrained problem and different values of the penalty. The values are interpolated for plotting: only integer states and actions are meaningful. The solid black line is the border of the viable set: state-actions above are viable, the ones below are not. On Figure 3.3a, the value function is not defined outside of the viable set and was arbitrarily set to 0 there. On Figure 3.3d, the values inside the viable set are equal to the ones on Figure3.3a and are greater than the ones outside of the viable set: this is a a consequence of Theorem 2.

Problem setting

The MDP we consider is described in Figure 3.4. The policies are parameterized by a scalar parameter $\theta \in [0, \theta_{\max}]$, with $\theta_{\max} = \frac{\sqrt{5}-1}{2}$:

$$\pi_\theta(a|s) = \begin{cases} \theta, & \text{if } a = a_1, \\ 1 - \theta - \theta^2, & \text{if } a = a_2, \\ \theta^2, & \text{if } a = a_3. \end{cases}$$

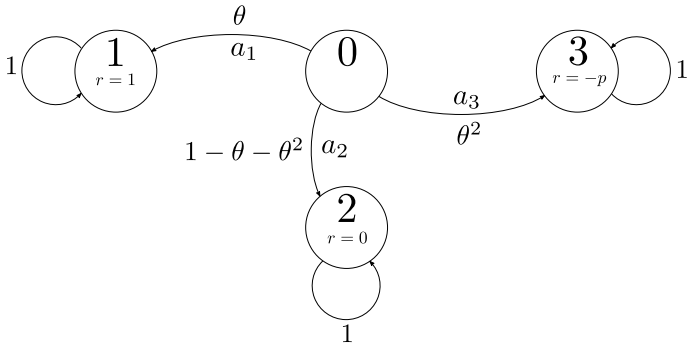


Figure 3.4: The agent is initialized in $s = 0$. Once it reaches $s = 1, 2$ or 3, it cannot go back. The state $s = 3$ is considered a failure. The policy is parameterized by the scalar $\theta \in [0, \theta_{\max}]$, where $\theta_{\max} = \frac{\sqrt{5}-1}{2}$.

If $s \neq 0$, all the actions have the same effect and keep the agent in its current state. The constrained problem can be rewritten as follows:

$$\underset{0 \leq \theta \leq \theta_{\max}}{\text{maximize}} \ (1-\gamma) \cdot \theta, \text{s.t. } \theta^2 = 0,$$

and the penalized one is:

$$\underset{0 \leq \theta \leq \theta_{\max}}{\text{maximize}} \ (1-\gamma) \cdot \theta \cdot \left[1 - p \cdot \theta\right],$$

The question we answer now is: does there exist a scaling of the penalty p for which the solution of the penalized problem is safe and has the same value as the constrained problem?

Solution

The constrained problem obviously has only one feasible parameter: $\theta = 0$. The return achieved by the corresponding policy is simply $G^c = 0$. For a given penalty p, the optimal parameter for the penalized problem is $\theta_p = \frac{1}{2p}$. This policy achieves a return of $G_p = \frac{1-\gamma}{4p}$. The duality gap is then:

$$\Delta = G_p - G^c = \frac{1-\gamma}{4p}.$$

We can clearly see that there exists no finite value of the penalty that makes $\Delta = 0$: the penalty cannot be scaled. What's more, for any value of the penalty, the optimal policy for the penalized problem has a probability of failing equal to $\frac{1}{4p^2} > 0$: the policy is not safe.

Discussion

The penalized problem gives unsafe policies This counterexample demonstrates that the theoretical guarantees of Theorem 2 are lost when parameterizing the policy. In general, the penalized and the constrained problems have different expected returns, and the optimal policy of the penalized problem is not safe. However, weak duality (Lemma 2) still holds: this is shown in [18]. Note however that, in the example, the guarantees still hold asymptotically when $p \to \infty$. We suspect that this is always true, but have not proved it.

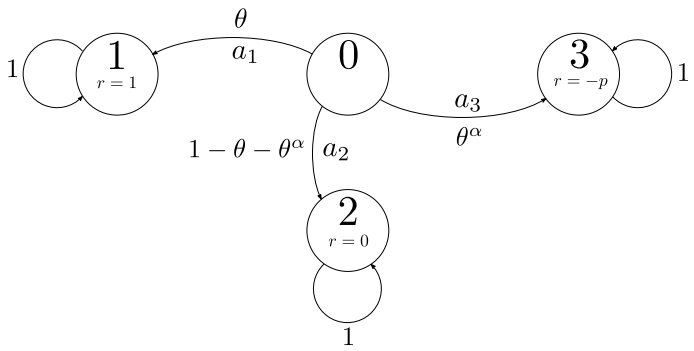


Figure 3.5: The agent is initialized in $s = 0$. Once it reaches $s = 1, 2$ or 3, it cannot go back. The state $s = 3$ is considered a failure. The policy is parameterized by the scalar $\theta \in [0, \theta_{\max}]$, where $\theta_{\max}$ depends on the hyperparameter α.

Comparison with the unparamaterized problem We have compared here the return of the *parameterized* problems. In [18], the authors compare the return of the parameterized penalized problem with the one the unparameterized constrained problem. They bound this suboptimality with the approximation error of the policy class they consider. This result is important, but it does not answer whether penalizing failures give safe, optimal policies. Our example shows that, without any further assumptions on the policy class, this is not true and that penalizing policies do not have any safety guarantees, contrary to the unparameterized case.

3.4.2 The duality gap is a poor indicator of the risk

One of the main results of [18] is showing that the difference between the optimal expected returns of the penalized and the constrained problems is bounded by the approximation error of the policy class. The underlying motivation is guaranteeing when parameterizing policies can be done without sacrificing too much performance. We argue here on a counterexample that guaranteeing a small gap in the return does *not* give any guarantees on the underlying safety of the policy. Our example shows that, for a given duality gap, the hyperparameters of the parameterization and the penalty can be chosen such that the resulting policy can have an arbitrarily high risk of failure. This is of particular importance since tuning the penalty and the rest of the hyperparameters is often done at the same time, and can result in risky policies without this being visible on the return signal.

Problem setting

The MDP we consider here is described in Figure 3.5. The dynamics are the same as in the previous section, but the class of policies is now parameterized by the hyperparameter $\alpha > 1$:

$$\pi_\theta(a|s) = \begin{cases} \theta, & \text{if } a = a_1, \\ 1 - \theta - \theta^\alpha, & \text{if } a = a_2, \\ \theta^\alpha, & \text{if } a = a_3. \end{cases}$$

Since α is a hyperparameter, the RL agent does not optimize on it: its only degree of freedom is the parameter $\theta \in [0, \theta_{\max}]$, with $\theta_{\max}$ depending on α. Once again,

the constrained and the penalized problem can be rewritten as[5]:

$$\begin{aligned} &\underset{0\leq\theta\leq\theta_{\max}}{\text{maximize}}\ \theta, \\ &\qquad \text{s.t. } \theta^\alpha = 0, \\ &\underset{0\leq\theta\leq\theta_{\max}}{\text{maximize}}\ \theta - p\cdot\theta^\alpha. \end{aligned}$$

The expected return of the constrained problem is G^c, and the one of the penalized problem is G_p^α (the superscript is a notation, and does not represent "G_p to the power α"). The resulting duality gap is $\Delta_p^\alpha = G_p^\alpha - G^c$.

Definition of $\theta_{\max}$ The parameter $\theta_{\max}$ is the only positive solution to the equation:

$$\theta_{\max}^\alpha = 1 - \theta_{\max}. \tag{3.23}$$

This equation comes from the fact that the probability of every transition must be nonnegative.

Optimal values and parameters

The only value of the parameter satisfying the safety constraint is $\theta = 0$, and so the expected return of the constrained problem is once again $G^c = 0$.

For a given penalty p and hyperparameter α, the penalized expected return is maximal in :

$$\theta_{p,\alpha} = \frac{1}{(\alpha p)^{\frac{1}{\alpha-1}}}. \tag{3.24}$$

But recall that the parameter θ should be lower than $\theta_{\max}$. Hence, the expected return is:

$$G_p^\alpha = \begin{cases} \theta_{p,\alpha} - p\theta_{p,\alpha}^\alpha, & \text{if } \theta_{p,\alpha} \leq \theta_{\max}, \\ \theta_{\max} - p\theta_{\max}^\alpha, & \text{otherwise.} \end{cases}$$

By combining this equation with equations (3.23)–(3.24), we get the following expression for the duality gap:

$$\Delta_p^\alpha = \begin{cases} \dfrac{1-\frac{1}{\alpha}}{(\alpha p)^{\frac{1}{\alpha-1}}}, & \text{if } \theta_{p,\alpha} \leq \theta_{\max}, \\ p\cdot(\theta_{\max}-1) + \theta_{\max}, & \text{otherwise.} \end{cases} \tag{3.25}$$

Different hyperparameters for a given duality gap

Now, assume that we aim at parameterizing the problem so the duality gap Δ_p^α is set to an acceptable value Δ_{target}, corresponding to our tolerance on the suboptimality of the penalized problem compared to the constrained one. What are the acceptable values of (p, α) that achieve this objective? This question can be answered by

[5]We omit the $1-\gamma$ factor for clarity since it does not have any impact on the solutions. This is equivalent to choosing $\gamma = 0$.

expressing the nonlinear relation between p and α given by Equation 3.25, where Δ_p^α is now considered fixed and equal to Δ_{target}. We get:

$$p = \begin{cases} \frac{1}{\alpha}\left(\frac{1-\frac{1}{\alpha}}{\Delta_{\text{target}}}\right)^{\alpha-1}, & \text{if } \theta_{p,\alpha} \leq \theta_{\max}, \\ \frac{\theta_{\max} - \Delta_{\text{target}}}{1-\theta_{\max}}, & \text{otherwise.} \end{cases} \tag{3.26}$$

This relation expresses the value the penalty should have in order to achieve the duality gap Δ_{target} with the hyperparameter α. Recall that $\theta_{\max}$ also depends on α. There may be better better penalties that achieve a lower duality gap: this value is the one for which the tolerance is met. Typically, it can be found by primal-dual algorithms whose stopping criterion depend on the gap.

The risk cannot be inferred from the duality gap

For a given choice of α and with the penalty p given by (3.26), the risk ρ_α of the optimal penalized policy is:

$$\rho_\alpha = \begin{cases} \left(\frac{\alpha\Delta_{\text{target}}}{\alpha-1}\right)^{\alpha}, & \text{if } \theta_{p,\alpha} \leq \theta_{\max}, \\ 1-\theta_{\max}, & \text{otherwise.} \end{cases} \tag{3.27}$$

Getting an explicit formula in terms of α is quite hard because of the implicit dependency in the conditioning. The risk can still be evaluated using numerical methods, and is plotted on Figure 3.6.

Discussion Figure 3.6 shows that the risk can vary a lot depending on the value of α. No matter the value of $\Delta_{\text{target}} > 0$, a poor choice of α can result in agents that have a 50% probability of failing. And, for all values of α, these agents achieve *exactly the same duality gap*: this duality gap is thus a very poor indicator of the risk of the learned policy. When solving safe RL problems by penalizing failures with parameterized policies, *controlling that the resulting duality gap is small is not sufficient to ensure approximate safety of the learned policy*: this risk needs to be evaluated and controlled independently.

3.5 Conclusion on the penalization-based approach

Solving the penalized problem instead of the constrained problem is much easier to implement in practice, but it suffers from a lot of other problems. While it is guaranteed to find safe and optimal policies in the unparameterized setting, all these guarantees vanish whenever the policies are parameterized - which is always the case in real-world applications. These policies can still perform well from the expected return perspective, as demonstrated in [18], but giving any guarantees on the safety of the learned policy is a different problem altogether. For applications to real robots, the problem of sample-efficiency is also critical and is absent from this analysis. Indeed, reducing the number of failures is important to protect both the robot and its surroundings. While the agent may be guaranteed to learn a safe

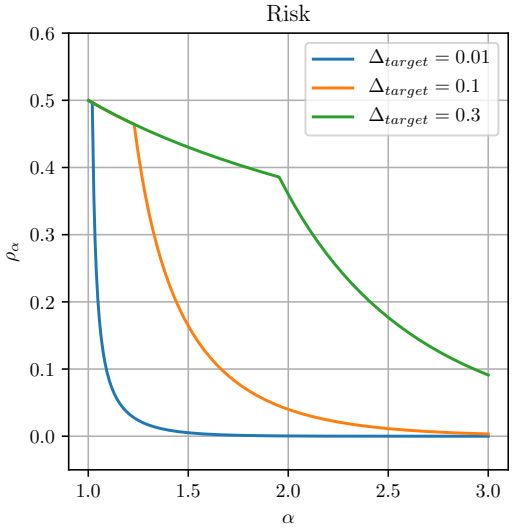


Figure 3.6: The risk (3.27) of the optimal policy when p is chosen to ensure that $\Delta_p^\alpha = \Delta_{\text{target}}$. For low values of α, we have $\theta_{p,\alpha} > \theta_{\max}$: the expression defining the risk in Equation (3.27) changes, causing the slope break visible on every curve. Then, the risk does not depend on Δ_{target}.

policy when failures are penalized, the number of failures required to ensure this condition is not known.

Safe policies enjoy a very simple characterization based on the viable set: they are simply policies that only consider actions inside that set. Even though the viable set is generally not known in practice, this characterization opens the door for a new way of solving the CMDP(3.5): only consider policies that stay inside the viability kernel - or inside an estimate of it. The constrained Q-Learning algorithm that we described on the example of the hovership in Section 3.3.2 is a very simple implementation of this consideration.

Chapter 4

Safely learning values: a benchmark

In this chapter, we derive and benchmark algorithms to learn safe, optimal policies leveraging the theoretical insight developed in the previous chapter. The goal of this benchmark is to test how successful the algorithms are at maximizing the reward while also quickly learning how to avoid failure. We first present the benchmarked algorithms and the set-up in Section 4.1 before giving the experimental results in Section 4.2.

4.1 Experimental set-up

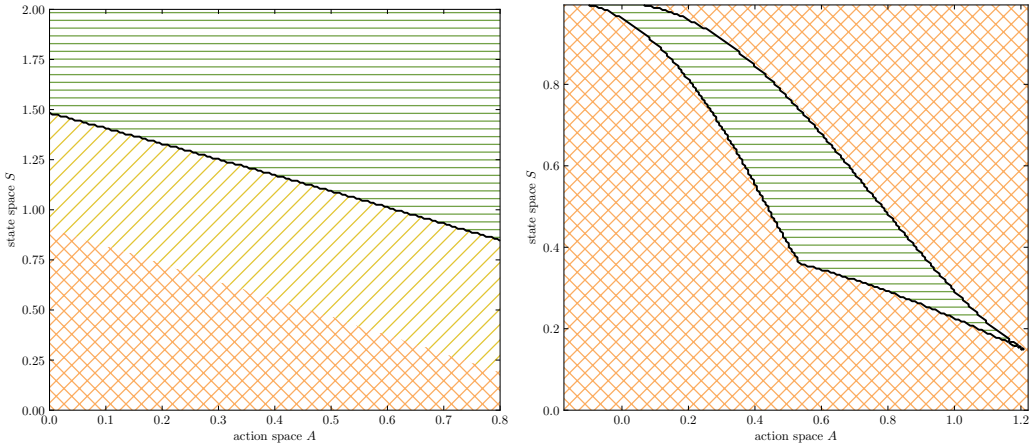
We first describe the experimental set-up of this benchmark. Two environments were tested: an extension of the hovership environment presented in Section 2.2.1 with a continuous state-action space [14], and the spring-loaded inverted pendulum (SLIP) model [13]. We test the four algorithms presented below on each of these environments. Their performances are evaluated on several metrics: the failure rate of the learned policy, the total reward it collects before the environment terminates, the discrepancy between the true viable set and its estimate (when applicable), and the number of failures during training.

4.1.1 Environments

The proposed methods are tested on different systems, whose dynamics, termination conditions, and rewards are described below. In each case, solving the dynamics involved solving one or many differential equations. The values of the parameters of the dynamics were taken from the code of Heim et al. [14], where the same environments are used for numerical examples. Each of these environments illustrates a different challenge: dealing with large unviable states, and complex dynamics where failing can happen at every step.

Hovership

The first system is the hovership environment, which can be parameterized to have a large unviable set.



(a) Hovership (b) SLIP

Figure 4.1: The state-action spaces of the two environments. On each figure, the viable set $\mathcal{Q}_V$ is in green, unviable state-actions are in yellow, and failing state-actions are in orange. A safe agent only picks actions in the green area.

State-action space and dynamics The example based on the hovership spaceship from Section 2.2.1 is modified with a continuous state-action space. The state s is now the altitude of the spaceship, and is a continuous variable in time. It satisfies the following differential equation:

$$\dot{s} = -g - 0.75 \cdot \tanh\left(s_{\max} - s\right) \cdot \nabla g + a, \tag{4.1}$$

where $g = 0.1$, $\nabla g = 1$, and $s_{\max} = 1$ are parameters, and a is the action. The state is constrained to stay in $[0, s_{\max}]$. In order to make the dynamics time-discrete, the agent is only allowed to change its action on regularly spaced time steps: the *control frequency* $f = 1$ Hz of the agent is another parameter. Hence, the action a in the dynamics (4.1) is piecewise constant.

Environment termination The failure set is $\mathcal{S}_F = \{0\}$: this corresponds to the ship crashing on the ground. There are two different ways that the environment terminates: either the agent fails, or it manages to take 10 consecutive steps without failing. This value is taken as a heuristic: we estimate that any agent that should fail will fail within this time horizon[1]. The dynamics (4.1) and the parameters were chosen so that a large portion of the state action space is unviable, as it can be seen on Figure 4.1a.

Reward The reward is an affine function of the state. It decreases from 10 for $s = 0$ (the lowest altitude) to 0 for $s = s_{\max}$ (the highest altitude). The optimal policy is then to go as low as possible, with the maximum possible thrust: this corresponds to reaching the lower-right region of the viable set.

[1]The results of the simulations proved that this is not true: some agents managed to end up in the unviable set after 10 steps without having failed yet. This only had minor consequences on the results, but will be corrected for future experiments.

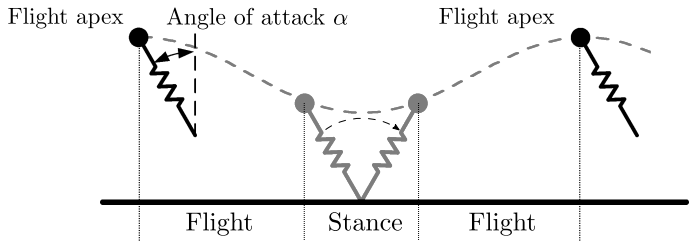


Figure 4.2: A qualitative trajectory of the SLIP model. The control input α is chosen at the first flight apex. The foot lands, pivots around its contact point, and the system springs again. At the next flight apex, a new angle of attack α can be chosen and a new cycle begins. Source: [13].

Spring-loaded inverted pendulum

The spring-loaded inverted pendulum (SLIP) is a model commonly used to describe running [13]. It has hybrid and highly non-linear dynamics, and the complex shape of its viable set make it a challenging task for control.

State-action space and dynamics The SLIP model is composed of a spring loaded with a mass. The dynamics can be decomposed in three phases: the flight phase, the stance phase, and another flight phase. The summary of the phases can be found on Figure 4.2. We use the approach from [13] to describe this system using a 1-dimensional state $s \in [0, 1]$, representing the normalized altitude of the mass at the flight apex. The control consists in changing the angle of attack of the spring leg and is applied at each flight apex, so one time per cycle. The dynamics are then treated as a discrete-time, nonlinear, and deterministic map with a two dimensional state-action space.

Environment termination Failure is defined as the mass hitting the ground, or changing direction during the stance phase: it is evaluated on the full state space of the continuous dynamics. The environments can then terminate in two different ways: either the agent fails, or it manages to take 10 consecutive steps without failing. This is here again a heuristics, but is supported by the fact that any state can be reached from any other state in at least two steps [24]. The state-action space and the viable set of the SLIP model can be seen on Figure 4.1b. Note that the shape of the SLIP's viable set make it a much more challenging task for control than the hovership, as failing is possible from every state.

Reward The reward is here again an affine function of the state, and is maximal when the state is lowest. The dynamics in the lower region of the viable set map back to the upper part of the state space. Depending on the discount factor, the optimal policy is then a trade off between going low (immediate reward) and staying at a low altitude on average (long term reward).

Remark 3. *Terminating environments after a certain number of steps is taken is a common practice in RL. However, it can be argued that it breaks the Markov assumption. This is a very legitimate concern, and is a topic of study in the RL*

	Parameters	Sampling	$\hat{\Lambda}_{\mathcal{Q}}$ update
Safety-Q switch	$\gamma_{\text{opt}}, \gamma_{\text{caut}}, \lambda_{\text{caut}}, t_{\text{switch}}$	From [14], then constrained Q-Learning	$t \leq t_{\text{switch}}$: Every step $t > t_{\text{switch}}$: On failure
Safety Q-Learning	$\gamma_{\text{opt}}, \gamma_{\text{caut}}, \lambda_{\text{caut}}$	Constrained Q-Learning	Every step
Soft Safety Q-Learning	$\gamma_{\text{opt}}, \gamma_{\text{caut}}, \lambda_{\text{caut}}, \gamma_{\text{soft}}$	Constrained Q-Learning	$a \notin \mathcal{Q}_{\text{soft}}$ On failure
Penalized Q-Learning	p	Q-Learning	None

Table 4.1: The four algorithms and their main parameters and behaviour. All algorithms also have parameters to configure the underlying Q-Learning: probability of not acting greedily ε, step size, discount rate. The "Constrained Q-Learning" update means that the chosen action in s is $\text{argmax}_{a,(s,a)\in\mathcal{Q}_{\text{caut}}} Q(s,a)$ with probability $1-\varepsilon$, or another random action a such that $(s,a) \in \mathcal{Q}_{\text{caut}}$ otherwise. The "Q-Learning" one means the same thing without the state-actions being in $\mathcal{Q}_{\text{caut}}$.

community [17]. However, the state-action spaces involved here are sufficiently small so the agent has the time to either reach one of its limit cycles or to fail within the alloted time. Moreover, agents are initialized randomly in the viability kernel: this randomization ensures that no part of the state space is insufficiently explored due to time constraints.

4.1.2 Algorithms

We implement four different learning strategies to find safe policies. Three of them build up on the work of Heim et al. [14] and use the safety measure presented in Section 2.2.2 to estimate the viable set $\mathcal{Q}_V$ to favor safe exploration. The main difference between these algorithms is how they select new samples and when they update their estimate of the viable set. These algorithms are compared with Q-Learning with and without penalization. A summary of all the algorithms and their parameterization is available on Table 4.1. The different functions that need to be learned are approximated with Gaussian Processes (GPs) [23].

Standard Q-Learning

The first algortihm that we implement is Q-Learning, and we test it on the penalized and on the unpenalized environments. The unpenalized Q-Learning is not expected to be particularly good, and is just evaluated here as a baseline. However, it is reasonable to expect that Q-Learning on the penalized environment actually learns a safe policy, from Theorem 2: Gaussian processes should not suffer from the problems emphasized in Section 3.4 since they can, in principle, approximate any function provided that enough data is available. Of course, the required amount of data largely depends on the regularity of the approximated function, and on the hyperparameters of the GP (the kernel, for example). Here, the regularity of the value function largely depends on the value of the penalty: higher penalties mean sharper local variations. Hence, depending on the value of the penalty, the penalized

Q-Learning algorithm may be able to learn the optimal safe policy without suffering too much from numerical issues.

Safety-Q switch

We make Q-Learning aware of the safety constraints by building up on the algorithm of Heim et al. [14] presented in Section 2.2.2. The idea of "Safety-Q switch" is to first run this viable set estimation algorithm, and then run a constrained Q-Learning algorithm that only allows the agent to pick actions inside the cautious estimate of the viable set $\mathcal{Q}_{\text{caut}}$. Deciding when to switch to Q-Learning is key: it can either be done by evaluating the convergence of the algorithm from [14], or after a certain number of iterations. We chose the second method for simplicity: we call this number of iterations the *switch time*, and note it t_{switch}. In these examples, this parameter is easy to tune since we have perfect system knowledge and know how many steps the algorithm learning $\mathcal{Q}_V$ requires. However, choosing t_{switch} is not straightforward in general. The main advantage of the next algorithms compared to Safety-Q Switch is that they do not have such a parameter.

Theoretical guarantees The Safety-Q switch algorithm enjoys very strong theoretical guarantees, as long as its estimate of $\mathcal{Q}_V$ is correct. Assume that the algorithm from [14] has found $\mathcal{Q}_{\text{caut}}$ such that $\mathcal{Q}_V = \mathcal{Q}_{\text{caut}}$. Then, any policy output by the Safety-Q switch algorithm for $t \geq t_{\text{switch}}$ is safe: this is a simple consequence of Corollary 1. Since the viability kernel is positively invariant under safe policies, the agent never goes out of $\mathcal{Q}_V$ ever again. Hence, the agent is simply doing Q-Learning on a stable subset of the state-action space, and so it is guaranteed to asymptotically find the optimal policy as soon as the convergence assumptions of Q-Learning are met [19]. If $\mathcal{Q}_V \neq \mathcal{Q}_{\text{caut}}$, this algorithm is not guaranteed to find an optimal policy anymore, but is still guaranteed to act safely as long as the $\mathcal{Q}_{\text{caut}}$ estimation algorithm has converged to a stationary value.

Safety Q-Learning

The previous algorithm explores large portions of the state-action space that may not be useful for the reward-maximization task. The idea of Safety Q-Learning is to *guide* the exploration with this reward-maximization task while *constraining* it to state-actions that are thought to be viable. Compared to Safety-Q switch, the Safety Q-Learning algorithm does not have a phase during which it explores for the viable set. Instead, it directly follows the Q-Learning update, and keeps updating the safety measure along the way. By doing so, we hope that the safety measure will primarily expand in regions that are deemed interesting by Q-Learning. Of course, the samples are now going to be less informative for the safety measure[2]. Hence, we expect this algorithm to be better at quickly accumulating reward, while having a slower and less accurate estimation of the viable set, especially in regions that are not interesting for reward maximization.

Learning what the agent needs to know The Safety Q-Learning algorithm enjoys the same strong theoretical guarantees as Safety-Q switch if it manages to

[2]The algorithm in [14] uses an active sampling criterion to maximize the information gain

correctly learn the viable set $\mathcal{Q}_V$. However, the goal of this algorithm is not to learn the whole viable set, but only to do it in specific, useful regions. In particular, $\mathcal{Q}_{\text{caut}}$ may converge to a superset of $\mathcal{Q}_V$, but only in regions where the value is not interesting. Wherever the optimal behaviour lies, $\mathcal{Q}_{\text{caut}}$ should locally match $\mathcal{Q}_V$ eventually.

Soft Safety Q-Learning

By following the Q-Learning update instead of actively sampling $\hat{\Lambda}_{\mathcal{Q}}$, the Safety Q-Learning algorithm generates a lot of samples that are uninformative for the safety measure. Indeed, the agent may find itself repeatedly taking actions that have been known to be safe for a long time. In practice, this can give rise to problems by creating an unbalanced dataset[3] or, in the case of GPs, by flooding the GP's dataset with points adding little information[4]. A way out of this problem is to introduce *sample selection*.

When are samples informative? Consider that we have evaluated the safety measure at $(s, a) \in \mathcal{Q}_{\text{caut}}$. Recall that we can compute from the output of the GP the probability that the measure at (s, a) is positive. If this probability is high, the GP is confident that the considered state-action is viable. If it is lower, it does not mean that the GP considers this point to be unviable (since it is in $\mathcal{Q}_{\text{caut}}$), but rather that this state-action *should* be viable, but the confidence is low. Hence, refining the estimate of $\hat{\Lambda}_{\mathcal{Q}}$ here would be useful. Consequently, we define a subset $\mathcal{Q}_{\text{soft}}$ of $\mathcal{Q}_{\text{caut}}$ which contains state-actions where the safety measure is above λ_{caut} with probability at least γ_{caut}, but *at most* γ_{soft}, where $\gamma_{\text{soft}} \in [\gamma_{\text{caut}}, 1]$ is a new parameter. In practice, this corresponds to defining a third probabilistic level set, in addition to $\mathcal{Q}_{\text{caut}}$ and $\mathcal{Q}_{\text{opt}}$. Then, the agent updates the safety measure with a new sample only if it was collected outside of $\mathcal{Q}_{\text{soft}}$.

The consequence of this is that the agent only updates the safety measure if the sampled state-action is near the current boundary of $\mathcal{Q}_{\text{caut}}$. Intuitively, this can be understood as a constraint activation condition: the estimate of $\mathcal{Q}_V$ is only expanded when the agent believes that the value of states that are currently outside of this estimate are interesting.

4.1.3 Metrics

Choice of the metrics The original goal is to derive algorithms that learn how to maximize the return while quickly learning how not to fail. Two important metrics

[3]This can become a problem for example when the safety measure is approximated with a neural network instead of a GP

[4]The time complexity of making a prediction with a GP increases polynomially with the number of samples [23].

are then the total expected return[5] G and the probability of failing F:

$$G = \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} R_{t+1} \right], \tag{4.2}$$

$$F = \mathbb{P}_\pi \left[\exists t, S_t \in \mathcal{S}_F \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \mathbb{1}_{\mathcal{S}_F}(S_{t+1}) \right], \tag{4.3}$$

where π is the evaluated policy[6]. For the agents that build an estimate of $\mathcal{Q}_V$, we also define the following two metrics:

$$K = \frac{|\mathcal{Q}_V \setminus \mathcal{Q}_{\text{caut}}|}{|\mathcal{Q}_V|}, \tag{4.4}$$

$$N = \frac{|\mathcal{Q}_{\text{caut}} \setminus \mathcal{Q}_V|}{|\mathcal{Q}_V|}, \tag{4.5}$$

where $|\cdot|$ is the cardinality of a set. We call K the *conservativeness* of the agent: it is the proportion of the viable set that the agent considers unsafe. Similarly, the *negligence* N is the normalized number of unviable state-actions that the agent considers to be safe.

Evaluating the metrics We evaluate these four metrics regularly during training. To evaluate the conservativeness and the negligence, we simply brute-force the computation of the set $\mathcal{Q}_{\text{caut}}$ and compare it with the true viable set, that is computed beforehand using the methods from [13]. The failure rate and the expected reward are evaluated by Monte Carlo methods. The current policy of the agent is evaluated on a number of *measurement episodes* that are not used for training[7]. The results are then averaged. The main bottleneck to precise and frequent evaluations is the available computational power: we chose to evaluate the policies every 10 training episodes, and use 20 measurement episodes each time. Finally, each agent was trained 10 times on each environment, and the metrics were once again averaged across trainings. As a consequence, each measurement point is an average over 200 episodes.

4.1.4 Sample forgetting

Gaussian processes get increasingly expensive to evaluate as the size of their dataset grows: the time-complexity of this operation is $\mathcal{O}(n^3)$, where n is the number of samples [23]. For this reason, limiting the number of samples quickly becomes a necessity. Doing so, giving priority to new samples makes a lot of sense in our setting: the value with which the dataset is updated depends on the past predictions on the GP. Hence, early values are *very* sensitive to the initialization. Forgetting

[5]We could have chosen the total discounted return instead, which is the return that the agents learn to maximize. This does not really make a difference, since the agents do not have any notion of time.

[6]For the second expression of F, recall that failure is a terminal state: if the agent fails, only one of the terms in the sum is non-zero.

[7]Hence, the policy does not change when it is evaluated.

Name	Symbol	Value
Exploration parameter	ε	0.1
Discount rate	γ	0.2
Step size	-	0.6
Optimistic confidence	γ_{opt}	$0.6 \to 0.8$
Cautious confidence	γ_{caut}	$0.7 \to 0.8$
Cautious threshold	λ_{caut}	0
Soft constraint confidence	γ_{soft}	$0.75 \to 0.85$
Switch time	t_{switch}	200
Number of episodes	-	500

Table 4.2: Values of the parameters. The first three parameters are the standard parameters of Q-Learning with an ε-greedy exploration policy. The confidence parameters are linearly increased during training. For the Safety-Q switch algorithm, the increase is until time t_{switch}.

them is thus a way of limiting the number of points while decreasing the long-term sensibility to the initial conditions.

The solution we implemented is based on a nearest neighbours approach. As new samples are added, old samples that are closer than a user-defined *forgetting radius* are forgotten. This general rule has two exceptions for the GP modeling the safety measure. First, failures cannot be forgotten, since they are the only data points where we update with the ground truth instead of an estimate. Second, failures cannot *cause* other samples to be forgotten. This enables for example to learn sharp shapes for the viable set, as it is the case for the SLIP model (see Figure 4.1b). For both environments, we chose a forgetting radius of 0.05, since it enabled a satisfactory learning while keeping the duration of each batch of 10 trainings between 3 and 7 hours[8] on a desktop PC with processor AMD Phenom 2 X4 with 4GB of RAM.

4.2 Results

All of the previous algorithms were run on both environments with the parameter values presented in Table 4.2. The parameters related to the safety measure were tuned from the results of Heim et al. [14]. The low value for the discount rate γ was chosen to illustrate the effect of unviable state-action pairs on the hovership example, and was kept consistent across all experiments. As in [14], the Gaussian processes used kernels of the Matérn family [23, Chapter 18], which have three parameters: a scalar amplitude, and one length scale for each input dimension. The length scales describe how far two samples should be so the GP considers their outputs to be independent. The GPs modeling the safety measure and the value function were parameterized identically, and their parameters were tuned by optimizing on the ground truth of the safety measure. The initialization of the estimates of the safety measure and the value function were kept consistent across all agents in a given environment. The values of the metrics were computed by the procedure described in Section 4.1.3

[8] Depending on the environment and on the agent.

	Hovership	SLIP
Safety-Q Switch	5.84	52.94
Safety Q-Learning	19.52	58.22
Soft Safety Q-Learning	22.84	68.84
Q-Learning	92.64	93.86
Penalized Q-Learning	76.62	72.18

Table 4.3: The percentage of episodes that ended in failure during training, averaged over 10 trainings. These values are very different from the ones in Figures 4.3b and 4.5b for the Q-Learning variants because, here, the ε-greedy exploration policy is active.

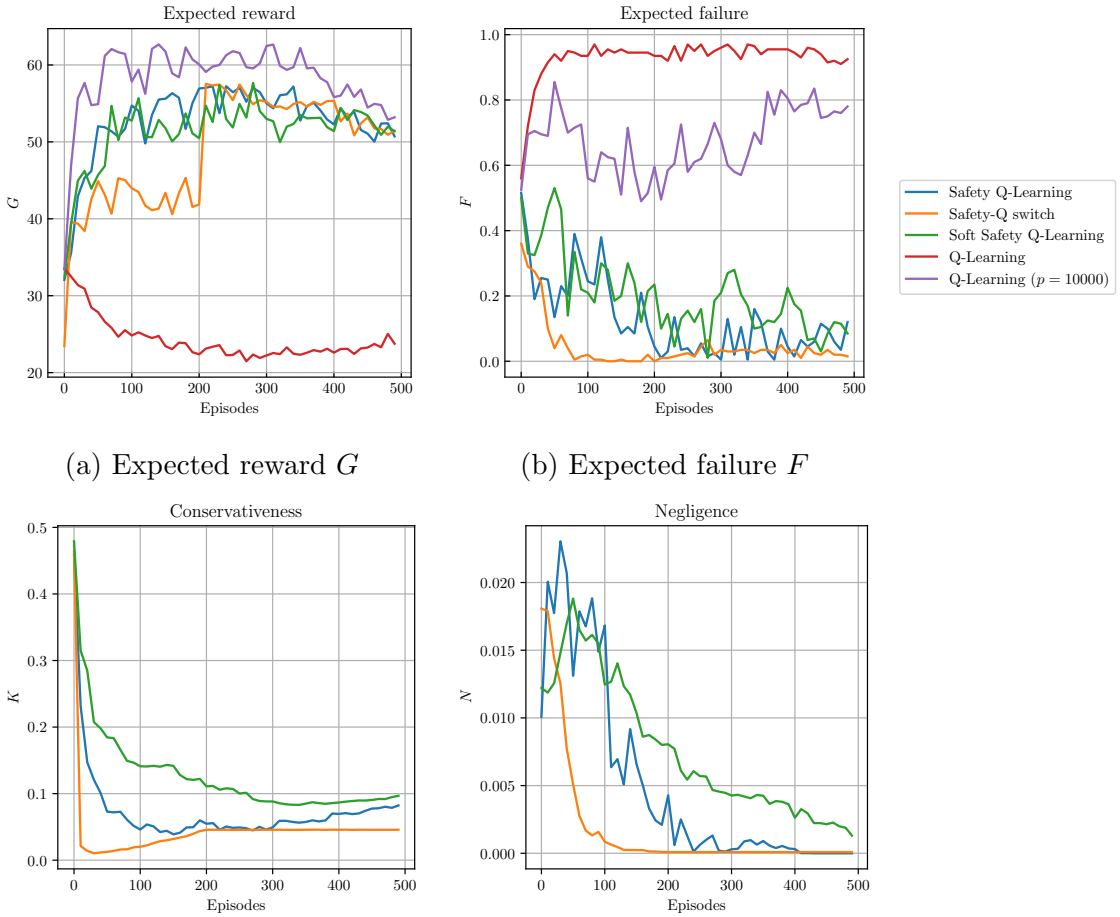
4.2.1 Hovership

The safety measure is initialized conservatively on this environment: the initial estimate of $\mathcal{Q}_{\text{caut}}$ is mainly contained in $\mathcal{Q}_V$, as it can be seen by the low initial values of the negligence N (Figure 4.3d). The learning curves of all algorithms are presented on Figure 4.3, and the learned value maps and safe sets on Figure 4.4. The general trends that emerge from these results is that all four safety-aware algorithms manage to achieve satisfactory return, but only the ones estimating the viable set learn safe behaviours: this comes from the existence of the unviable set, as we see it now.

The unpenalized Q-Learning algorithm does not learn how to stay safe. The low discount rate γ prevents any long-term planning, which leads the agent to maximize the immediate reward and go down as fast as possible, thus failing in a few steps (Figure 4.4d). As a result, the unpenalized Q-Learning algorithm is very bad at maximizing the total reward.

This changes drastically when adding a high penalty. The high value of the penalty“ back-propagates” from the failure set and lowers unviable values, while being scaled by γ at every step: the agent learns that it should avoid failing, at least in the short term. As a result, the agent learns how to act in a safer way, but still not safely (Figure 4.3b). Instead, it learns how to *fail slowly* (Figure 4.4e): once in the unviable set, it stays there for as long as possible and therefore collect high rewards (since the unviable set has better rewards than the viable one). This results in an unsafe policy that achieves better total reward in the given time horizon than the optimal safe one. As a matter of fact, the agent sometimes (about 25% of the time) managed to stay in the unviable set long enough that the environment terminates before it fails. This almost surely explain how the penalized agent manages to collect higher rewards than the supposedly optimal safety-aware algorithms (Figure 4.3a). This shows that the agent is able to survive for a long time in the unviable set: therefore, even a high penalty value like $p = 10000$ is unable to make the learned policy safe.

Safety measure-based algorithms do not suffer from such an issue, and all quickly achieve almost-safe behaviour (Figure 4.3b). Safety-Q Switch is the fastest algorithm to learn how to act safely, and quickly achieves almost-zero conservativeness and negligence(Figures 4.3c–4.3d). The steady non-zero value of the conservativeness is understood as a consequence of the forgetting radius discussed in Section 4.1.4: the agent is unable to find the correct border between the viable and the unviable sets.



(a) Expected reward G

(b) Expected failure F

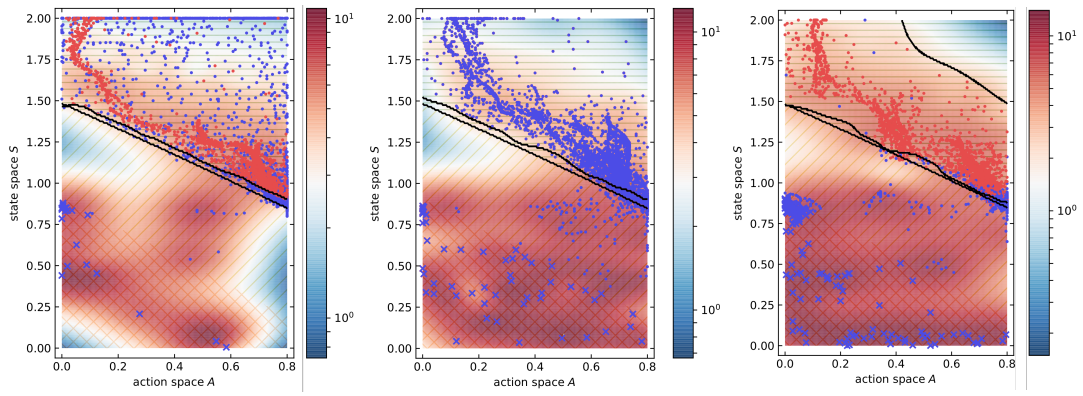
(c) Conservativeness K

(d) Negligence N

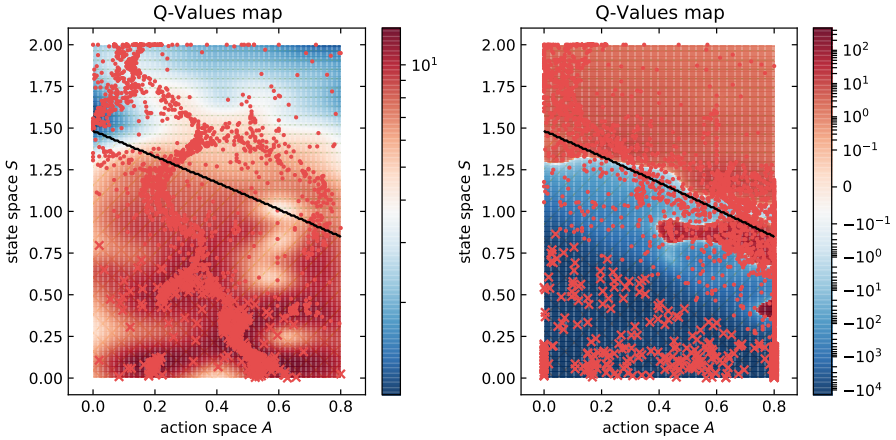
Figure 4.3: Training curves of the agents on the hovership environment. The standard deviation is not plotted for clarity and because it does not add any major information. For penalized environments, Figure 4.3a corresponds to the reward without the penalty.

Safety-Q Switch methodically explores the whole viable set during its exploration phase (Figure 4.4a), and therefore acts suboptimally from the reward-maximization perspective during early steps. This changes when it switches behaviour at $t = t_{\text{switch}}$: then, the algorithm starts exploiting all the knowledge it has collected about the reward function during the first phase, and instantaneously catches up with reward-driven algorithms.

The Safety Q-Learning and Soft Safety Q-Learning algorithms are therefore better at accumulating reward in early steps, but their advantage disappears after t_{switch}. These algorithms also have a surprisingly high failure rate: we would expect them to be much closer to the one of Safety-Q Switch. A possible explanation is their aggressive exploration policy. Indeed, when not acting greedily, they uniformly sample another action inside $\mathcal{Q}_{\text{caut}}$. Therefore, they may end up in regions of the state-action space in which they do know yet what actions are viable or not, especially in the beginning. Finally, Safety Q-Learning and its Soft variant seem to neglect the upper-right region of the state-action space in the exploration. They indeed quickly realize that this region is not interesting for the reward, contrary to



(a) Safety-Q switch (b) Safety Q-Learning (c) Soft Safety Q-Learning



(d) Q-Learning (e) Penalized Q-Learning ($p = 10000$)

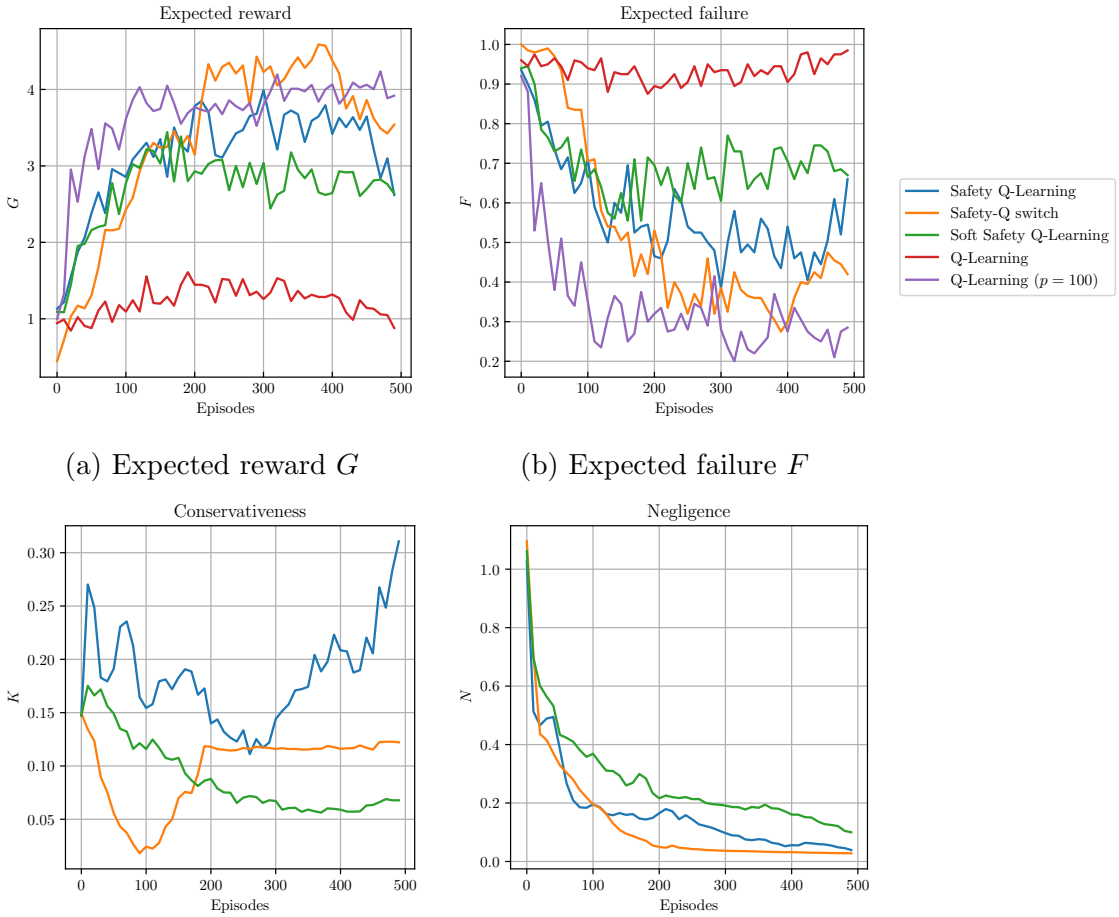
Figure 4.4: The learned value maps and $\mathcal{Q}_{\text{caut}}$ set. The coloured dots are the state-actions visited during training. Blue samples are used by both GPs, whereas the red ones are only used by the GP modeling the value function. On Figures 4.4a–4.4c, the sets $\mathcal{Q}_V$ and $\mathcal{Q}_{\text{caut}}$ are the solid black lines.

Safety-Q Switch.

Table 4.3 shows that the safety measure-based algorithms need to sample failure far less often than Q-Learning variants. The fundamental reason is the ε-greedy exploration policy of Q-Learning: even if it has already discovered the true value map and the optimal policy, it regularly deviates from this optimal policy and takes a random action for exploration. Therefore, it very often samples a failure. This argument indicates that safety measure based methods are more sample-efficient when sampling failures.

4.2.2 Spring-loaded inverted pendulum

The safety measure is initialized optimistically on this environment, and a lot of unsafe state-actions are considered safe initially: this is shown by the high initial negligence and low initial conservativeness (Figures 4.5c–4.5d). The learning curves of all algorithms are presented on Figure 4.5, and the learned value maps and safe sets on Figure 4.6. On this task, the penalized method performs similarly to the ones based on the safety measure on the expected return, and achieves a much lower failure rate.



(a) Expected reward G

(b) Expected failure F

(c) Conservativeness K

(d) Negligence N

Figure 4.5: Training curves of the agents on the SLIP environment. The standard deviation is not plotted for clarity and because it does not add any major information. For penalized environments, Figure 4.5a corresponds to the reward without the penalty.

The low failure rate achieved by the penalized Q-Learning method was expected. Indeed, contrary to the hovership environment, the penalty here is not hidden by the small discount factor. Therefore, the agent rapidly learns that failing actions are not interesting.

More surprising is the poor performance of safety-based methods in terms of failure rate (Figure 4.5b). The low negligence they all achieve indicates that they quickly learn a conservative estimate of the viable set (Figure 4.5d), and Figures 4.6a– 4.6c show that this conservativeness does not prevent them from knowing how to behave in almost-every situation: except at the very top and bottom of the viable set, all algorithms know how to perform in a viable way. We do not have a definitive explanation for these poor performances. Our best hypothesis is that the hyperparameters of the Gaussian Process modeling the safety measure are incorrectly tuned for this problem. Indeed, GPs with Matérn kernels assume smooth variations of the data over the lengthscale of the kernel. The viable set of the SLIP environment is very narrow for low and high values of the state (Figure 4.1b): therefore, the Q-safety function varies sharply in these regions. Moreover, these regions

play a critical role here since the reward incentivizes the agent to go as low as possible. We also believe that this is the reason why the conservativeness of some agents start rising again on Figure 4.5c. We plan on exploring this hypothesis in the coming weeks. It can still be noted that, even with this poor parameterization, Safety-Q Switch manages to mitigate its failure rate to reasonable levels.

Despite their general poor performance, Safety Q-Learning seems to achieve better return and a lower failure than its Soft counterpart. This indicates that the criterion we use to decide whether a sample is informative for the safety measure is a bit off, since it excludes samples that enable better performance. This can be seen on Figure 4.6c: the safety measure indeed only uses a very small number of viable samples. The added samples are still very meaningful for the viable set: they typically correspond to regions where refining the initial estimate was indeed necessary. However, a relaxation of this criterion should be considered.

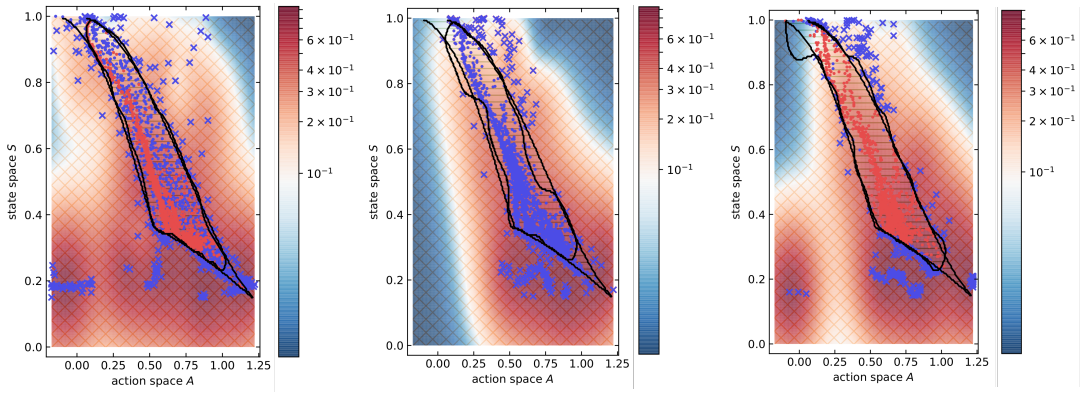
Finally, it should be noted that here again safety measure-aware methods largely outperform Q-Learning based methods in terms of the number of failures *during* training (Table 4.3). This is also caused by the exploration policy of Q-Learning methods, since they are likely to take a random action at every step. This is a clear advantage of methods estimating the safety constraints, since they avoid such a problem *by design*.

4.2.3 Discussion

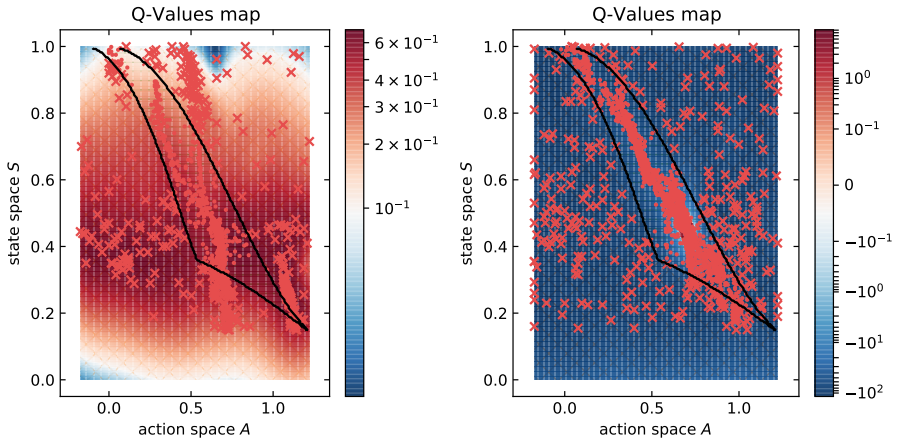
In the last sections, we have presented and benchmarked four algorithms to learn safe policies, three of them being based on building an estimate of the viable set $\mathcal{Q}_V$ to put a constraint on the actions available to the exploration strategy.

The method based on first building an estimate of the safe set, and then optimizing for the expected return is the more careful: it achieves the lowest failure rate when correctly parameterized, and a satisfying one when its parameters don't enable learning $\mathcal{Q}_V$ correctly. While building the estimate of the viable set, it also accumulates knowledge on the reward function, and this knowledge can be immediately used when switching behaviours. Therefore, even though this method is suboptimal during early steps, it quickly catches up on reward-focused methods after t_{switch}. This emphasizes the importance of this switch parameter. If it is too small, the estimate of the viable set is not good enough and the agent will not act safely. If it is too high, the agent will take a long time before taking reward-maximizing actions. Moreover, the systematic exploration of the whole viable set can become a problem in high dimensional systems (where learning the whole set is computationally intractable) or if data collection is costly.

The second and third algorithms get rid of this exploration phase, and immediately try to learn how to maximize the reward by guiding the exploration with constrained Q-Learning updates. This enables them to primarily refine their estimate of the viable set in regions that are interesting from a reward perspective. However, they seem to fail systematically more often than the previous method. This may be caused by the aggressive constrained ε-greedy exploration policy they implement from the very beginning, when the estimate of the viable set is very unprecise. More local exploration policies - such as adding a local noise around the optimal action - should reduce this tendency. Overall, the main advantage of these methods over the previous one is that they do not have an initial inefficient



(a) Safety-Q switch (b) Safety Q-Learning (c) Soft Safety Q-Learning



(d) Q-Learning (e) Penalized Q-Learning ($p = 100$)

Figure 4.6: The learned $\mathcal{Q}_{\text{caut}}$ set for the three algorithms based on the safety measure on the SLIP environment. Blue samples are used by both GPs, whereas the red ones are only used by the GP modeling the value function. On Figures 4.6a–4.6c, the sets $\mathcal{Q}_V$ and $\mathcal{Q}_{\text{caut}}$ are the solid black lines.

exploration phase, and therefore are easier to tune and should be better suited for high-dimensional systems.

In the presence of unviable states, these three algorithms outperform penalty-based methods in terms of failure rate. Indeed, the penalty scales exponentially with the number of steps that can be taken in the unviable set before failing. It therefore needs to be very high in order to learn safe policies, even in simple examples like the hovership. On the other hand, penalizing failures has proven to be very effective when the unviable set is empty, which confirms experimentally the statements of Corollary 2.

One key advantage that methods estimating safety have over penalized methods is their sample efficiency with the number of failures. Indeed, they manage to quickly learn the constraints and naturally stop violating them *during learning*. Penalized methods do not enjoy such a property, since their exploration policy (like ε-greedy) is generally unaware of the safety constraints, and therefore repeatedly sample failures. This is a major practical drawback to implement them on safety-critical systems, where the number of failures should be reduced during learning.

Chapter 5

Summary and outlook

5.1 Summary

This thesis tackles the problem of quickly learning reward-optimal safe policies. Learning safe policies in a sample efficient way is of critical importance when implementing learning methods on dynamical systems and real robots. Theoretically-guaranteed methods exist to solve learning problems with safety constraints, but are generally model-based. The question this thesis focuses on is therefore whether it is possible to *learn* these constraints while learning control at the same time, in a purely model-free setting.

The first contribution is a theoretical characterization of safe policies using tools from the well-established viability theory. This enables us to demonstrate a peculiar property of learning with a 0-risk constraint compared to classical CMDPs: it behaves exactly as an unconstrained problem on a subset of the state-action space; the viable set. From this property, we derive a new result on penalty scaling showing that penalizing failures is a theoretically guaranteed way of learning safe optimal policies, and this for an unbounded interval of penalties. This provides an answer to recurring concerns about penalty-based methods. However, we also demonstrate on a counterexample that these guarantees do not hold when policies are parameterized - which is almost always the case in practice. We argue that, in the case of parameterized policies, penalizing failures to have a return similar to the one of the constrained problem does not provide *any* guarantees on the safety of the learned policy. Hence, penalty-based approaches do not enable to approximately solve the safe RL problem when theoretical guarantees do not hold.

Therefore, we build up on previous work estimating the viable set of a system and propose three variants of an algorithm to learn safe policies. These algorithms construct an estimate of the viable set, and restrict their exploration policy in this estimate to ensure approximately safe exploration. We benchmark these algorithms with penalized Q-Learning on two environments: a spaceship subject to gravity, and the spring-loaded inverted pendulum model. These systems exhibit different types of challenges: the first one has a large unviable set, and therefore requires long-term planning to avoid failure, whereas the second one has complex dynamics and can easily fail. This benchmark experimentally demonstrates that the viability-based approach of safety can learn optimal safe policies in a sample-efficient way, and offers and interesting comparison between penalty-based and constraints-based methods.

This thesis bridges the gap between the chance-constrained formulation of safe reinforcement learning and penalized methods. It also provides insight on what the effect of constraining Markov decision processes is, and takes a first step in using this insight to solve safety-constrained MDPs. As the interest for learning control for dynamical systems grows, we believe that this works contributes to unifying the different approaches to safety in model-free optimal control.

5.2 Outlook

5.2.1 Allowing a nonzero risk of failing

By defining safety as allowing a 0 risk of ending up in the failure set, we are able to treat the resulting CMDP as a simple MDP with a modified state-action space: the viable set $\mathcal{Q}_V$. In practice, specialized algorithms are still required in order to *learn* the constraint, but this theoretical decoupling between the unviable set and the viable set allows us to use a lot of tools from classical MDP theory.

We believe that using the viable set to characterize safe policies is a very promising perspective, and can be extended to the more general setting of allowing a nonzero risk ε of ending up in the failure set. With this definition, safe policies should be the ones that leave the viable set with probability at most ε. From this simple viability theory result, safe policies could be easily characterized as the ones that only put so much probability mass they on actions in the critical set $\mathcal{Q}_C$, similarly to what Corollary 1 does.

5.2.2 Parameterization and strong duality

We have given in Section 3.4 counterexamples where parameterizing the set of policies breaks the safety guarantees that the penalized problem enjoys. Deriving necessary conditions on the parameterization for strong duality to still hold could be very beneficial. Such conditions could be for example based on classic results from duality theory (Karush-Kuhn-Tucker conditions [7], ...) .

This differs from the work of Paternain et al. [18], on which we have already commented in Section 3.4: the authors study the difference between the parameterized penalized problem and the unparameterized constrained one. What we suggest here is to study when penalizing policies can be done without sacrificing performance compared to the parameterized constrained problem (which is in any case the best solution that can be found).

5.2.3 Scaling the algorithm

The three novel algorithms presented in Chapter 4 scale very poorly with the system's dimensionality. The main reason for that is the use of Gaussian processes as function approximators. Indeed, estimating the safety measure in higher dimensions requires a lot of samples, and GPs scale very poorly with the size of their dataset. Therefore, replacing them with other function approximators (like neural networks) is a very promising approach. This could be done for example by having a first network predict the set of viable actions in a given state. This output could then be fed into a policy network, whose role is to pick the best action. Such an architecture

would extend the idea that we have explored here of constraining the exploration policy into an estimated safe set.

Appendix A

Proof of Theorem 2

Notations In all of the following, we define the unviable set $\mathcal{S}_U = \mathcal{S} \setminus \mathcal{S}_V$. We also remind the reader that $T = (S_0, A_0, S_1, A_1, \ldots)$ is the random variable denoting the trajectory, and we define:

$$\rho(T) = \sum_{t=0}^{T_f} \gamma^t \mathbb{1}_{\mathcal{S}_F}(S_{t+1}).$$

With this notation, we have: $\rho_\pi = \mathbb{E}_\pi(\rho(T))$. Finally, for all $s \in \mathcal{S}$, we note $\mathcal{T}(s)$ the set of trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots)$ such that $s_0 = s$.

A.1 Weak duality

We start by proving the following lemma, that emphasizes the fact that the penalized optimal value function (3.13) is a Lagrangian relaxation of the constrained value function (3.15):

Lemma 2 (Weak duality)**.** *Let $s \in \mathcal{S}$. For any policy π and penalty p, consider the Lagrangian*[1]*:*

$$\mathcal{L}(\pi, p) = \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t R_t \middle| S_0 = s \right] - p \cdot \mathbb{E}_\pi \left[\rho(T) | S_0 = s \right]. \tag{A.1}$$

The function $\mathcal{L}$ is a Lagrangian associated to the problem defining the constrained state value function (3.15)*, and V_p is an evaluation of the corresponding Lagrange dual function. Formally:*

$$V^c(s) = \max_\pi \inf_p \mathcal{L}(\pi, p), \tag{A.2}$$

$$\forall p,\ V_p(s) = \max_\pi \mathcal{L}(\pi, p). \tag{A.3}$$

Then, weak duality holds:

$$\forall p,\ V_p(s) \geq V^c(s). \tag{A.4}$$

[1]The Lagrangian also depends on the state s, but this dependency is dropped in the notations for clarity.

Proof. Equation (A.2) comes from the fact that:

$$\inf_p \mathcal{L}(\pi, p) = \begin{cases} \mathbb{E}_\pi \left[\sum_{t=0}^{T_f} \gamma^t R_t \middle| S_0 = s \right], & \text{if } \mathbb{E}_\pi \left[\rho(T) | S_0 = s\right] = 0, \\ -\infty, & \text{otherwise,} \end{cases}$$

so the maximum of this quantity is obtained when the constraint is satisfied. Equation (A.3) is immediate by linearity of the conditional expected value. Now, weak duality is obtained by noting that, for all p:

$$V_p(s) \geq \inf_q V_q(s) = \inf_q \max_\pi \mathcal{L}(\pi, q) \geq \max_\pi \inf_q \mathcal{L}(\pi, q) = V^c(s),$$

where the second inequality comes from the general *max-min* inequality. □

A.2 Existence of safe policies

Another useful lemma is the following, which largely relies on the assumption that $\mathcal{S}_V \neq \emptyset$:

Lemma 3 (Existence of safe policies)**.** *There exists a deterministic safe policy.*

Proof. This is an immediate consequence of Corollary 1. Indeed, let us construct a safe policy. For any state $s \in \mathcal{S}_V$, pick an action $a_s \in \mathcal{A}$ such that $(s, a_s) \in \mathcal{Q}_V$: such an action exists, because s is viable. For any state $s \in \mathcal{S}_U$, pick any action $a_s \in \mathcal{A}$. Now, consider the policy:

$$\pi(a|s) = \mathbb{1}_{\{a_s\}}(a).$$

Let $(a, s) \in \mathcal{Q}_C \cap (\text{Reach}_\pi \times \mathcal{A})$. Since $s \in \mathcal{S}_V$ and $(s, a) \notin \mathcal{Q}_V$, then, $a \neq a_s$, and so $\pi(a|s) = 0$. So, by Corollary 1, π is safe. □

A.3 Value bound outside of $\mathcal{S}_V$

The following lemma gives a bound on the penalized value function outside of $\mathcal{S}_V$:

Lemma 4 (Bound on V_p)**.** *There exists $K \in \mathbb{R}$ such that the following holds:*

$$\forall p \in \mathbb{R},\ \forall s \in \mathcal{S}_U,\ V_p(s) \leq K - \gamma^{|\mathcal{S}_U|} p \tag{A.5}$$

Proof. Let $p \in \mathbb{R}$ and $s \in \mathcal{S}_U$. Classic results on the max function give:

$$V_p(s) \leq \max_\pi \left(\sum_{t=0}^{T_f} \gamma^t \mathbb{E}_\pi \left[R_t | S_0 = s\right] \right) - p \min_\pi \mathbb{E}_\pi \left[\rho(T) | S_0 = s\right].$$

Now, $K \doteq \max_{s' \in \mathcal{S}_U} \max_\pi \left(\sum_{t=0}^{T_f} \gamma^t \mathbb{E}_\pi \left[R_t | S_0 = s'\right] \right)$ is a constant. We show that:

$$\min_\pi \mathbb{E}_\pi \left[\rho(T) | S_0 = s\right] \geq \gamma^{|\mathcal{S}_U|}.$$

To do so, we adapt the proof of Lemma 1 from [14] and show that:

$$\rho(\tau) = \sum_{t=0}^{T_f} \gamma^t \mathbb{1}_{\mathcal{S}_F}(s_t) \geq \gamma^{|\mathcal{S}_U|},$$

for all trajectory $\tau = (s_0, a_0, \ldots) \in \mathcal{T}(s)$. Indeed, such a trajectory fails within finite time, since $s \in \mathcal{S}_U$. Now, before it fails, it can only explore states in $\mathcal{S}_U$, otherwise it would explore a state in $\mathcal{S}_V$ but this is not possible since $s \in \mathcal{S}_U$. Moreover, it cannot visit any state in $\mathcal{S}_U$ more than once: otherwise, there would be a cycle in the trajectory, and so it would be possible to avoid failure when starting from s. Consequently, the trajectory can take at most $|\mathcal{S}_U|$ steps before failing. Hence, there exists $t \leq |\mathcal{S}_U|,\ \mathbb{1}_{\mathcal{S}_F}(s_t) = 1$, which shows $\rho(\tau) \geq \gamma^{|\mathcal{S}_U|}$. Consequently:

$$\min_\pi \mathbb{E}_\pi \left[\rho(T) | S_0 = s\right] = \min_\pi \mathbb{E}_\pi \left[\rho(T) | S_0 = s\right] \geq \gamma^{|\mathcal{S}_U|}.$$

□

A.4 Main proof

We are now ready to prove Theorem 2.

Proof. First of all, it directly follows from Lemma 4 that Equation (3.19) holds. So we only have to prove Equation (3.18). Note that, from weak duality (Lemma 2), we already have $\forall p \in \mathbb{R},\ \forall s \in \mathcal{S}_V, V_p(s) \geq V^c(s)$. We will show that:

$$\exists p^* \in \mathbb{R},\ \forall p \geq p^*,\ \forall s \in \mathcal{S}_V,\ V_p(s) \leq V^c(s). \tag{A.6}$$

To do so, we will show that, for high enough penalties, there exists a policy that achieves the optimal penalized value (3.13) and that is also safe - thus suboptimal for the constrained value (3.15). The result will follow immediately.

For all $p \in \mathbb{R}$, the function $V_p : \mathcal{S} \to \mathbb{R}$ is the value function of a RL problem, so it solves the optimal Bellman equation [19, Chapter 3]:

$$\forall s \in \mathcal{S},\ V_p(s) = \max_{a \in \mathcal{A}} R^{\text{exp}}_{t+1}(s,a) - p \cdot \mathbb{1}_{\mathcal{S}_F}(f(s,a)) + \gamma \cdot V_p(f(s,a)), \tag{A.7}$$

where $R^{\text{exp}}_{t+1}(s,a) = \mathbb{E}(R_{t+1} | S_t = s, A_t = a)$. For any penalty p, let π_p be a policy that is greedy with respect to the optimal value function V_p. This policy chooses, for each state $s \in \mathcal{S}$, an action achieving the argmax of the right-hand side of Equation (A.7), and is optimal [19, Chapter 3]. We show that:

$$\forall\, (s,a) \in \mathcal{Q}_C,\ \exists p^* \in \mathbb{R},\ \forall p \geq p^*,\ (s \in \text{Reach}_{\pi_p} \implies \pi_p(a|s) = 0). \tag{A.8}$$

Proof of Equation (A.8). We prove this result by contradiction. Assume Equation (A.8) is false, that is, there exists $(s,a) \in \mathcal{Q}_C$ such that there exists a sequence of penalties $(p_n)_{n \in \mathbb{N}}$ such that:

- $\lim_{n \to \infty} p_n = \infty$,
- for all $n \in \mathbb{N},\ s \in \text{Reach}_{\pi_{p_n}}$

- for all $n \in \mathbb{N},\ \pi_{p_n}(a|s) > 0$.

Intuitively, s is a state in the reach of all policies π_{p_n} and from which all of these optimal policies give a nonzero probability of picking action a when the penalty grows, thus leaving the viability kernel. For all $n \in \mathbb{N},\ \pi_{p_n}$ acts greedily on V_{p_n}, so we have:

$$a \in \operatorname*{argmax}_{b \in \mathcal{A}} V_{p_n}(f(s,b)). \tag{A.9}$$

Since V_{p_n} satisfies the Bellman equation (A.7), Equation (A.9) is equivalent to:

$$\begin{aligned} \forall b \in \mathcal{A}, V_{p_n}(f(s,a)) \geq & \frac{1}{\gamma}(R^{\exp}_{t+1}(s,b) - R^{\exp}_{t+1}(s,a)) \\ & - \frac{p_n}{\gamma}(\mathbb{1}_{\mathcal{S}_F}(f(s,b)) - \mathbb{1}_{\mathcal{S}_F}(f(s,a))) \\ & + V_{p_n}(f(s,b)). \end{aligned} \tag{A.10}$$

Now, according to Lemma 4, and since $f(s,a) \in \mathcal{S}_U$, we have:

$$\lim_{n \to \infty} V_{p_n}(f(s,a)) = -\infty.$$

Take $b \in \mathcal{A}$ such that $(s,b) \in \mathcal{Q}_V$. Such a b exists, because $s \in \mathcal{S}_V$. Since $\mathbb{1}_{\mathcal{S}_F}(f(s,b)) = 0$, the function:

$$n \mapsto -\frac{p_n}{\gamma}(\mathbb{1}_{\mathcal{S}_F}(f(s,b)) - \mathbb{1}_{\mathcal{S}_F}(f(s,a))),$$

is lower bounded. Consequently, $\lim_{n\to\infty} V_{p_n}(f(s,b)) = -\infty$. According to weak duality (Lemma 2), V_{p_n} is an upper bound of V^c, so:

$$V^c(f(s,b)) = -\infty. \tag{A.11}$$

Equation (A.11) is a contradiction. Indeed, we know from Lemma 3 that there exists a safe policy, so for all $s \in \mathcal{S}_V,\ V^c(s) > -\infty$. Indeed, any feasible policy gives a finite value for the optimization objective of (3.15). Since $f(s,b) \in \mathcal{S}_V$, Equation (A.11) is indeed a contradiction, and thus Equation (A.8) holds. ■

Equation (A.8) gives us a *local* value for p^*, depending on the point of $\mathcal{Q}_C$ we want the policies to stop considering. We make this p^* global by taking the maximum. For any $q \in \mathcal{Q}_C$, let $p^*(q)$ be a p^* as given by Equation (A.8). Since $\mathcal{Q}_C$ is finite and with an abuse of notation, we can define $p^* = \max_{q \in \mathcal{Q}_C} p^*(q)$. For any penalty $p \geq p^*$, it holds:

$$\forall (s,a) \in \mathcal{Q}_C \ \cap\ (\mathrm{Reach}_{\pi_p} \times \mathcal{A}),\ \pi_p(a|s) = 0. \tag{A.12}$$

According to Corollary 1, this means that for all $p \geq p^*,\ \pi_p$ is safe. We are now ready to conclude the proof. For all $p \geq p^*$, the following three equations hold:

$$\forall\ s \in \mathcal{S}_V,\ V_p(s) \quad = \mathbb{E}_{\pi_p}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s\right] - p \cdot \rho^{\pi_p}(s), \tag{A.13}$$

$$\forall\ s \in \mathcal{S}_V,\ V^c(s) \quad \geq \mathbb{E}_{\pi_p}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s\right], \tag{A.14}$$

$$\forall\ s \in \mathcal{S}_V,\ \rho_{\pi_p}(s) \quad = 0. \tag{A.15}$$

Combining these show that Equation (A.6) holds, and so does Equation (3.18): this concludes the proof of Theorem 2. □

Bibliography

[1] Joshua Achiam et al. "Constrained policy optimization". In: *arXiv preprint arXiv:1705.10528* (2017).

[2] Eitan Altman. *Constrained Markov decision processes*. Vol. 7. CRC Press, 1999.

[3] Jean-Pierre Aubin, Alexandre M Bayen, and Patrick Saint-Pierre. *Viability theory: new directions*. Springer Science & Business Media, 2011.

[4] Bowen Baker et al. "Emergent tool use from multi-agent autocurricula". In: *arXiv preprint arXiv:1909.07528* (2019).

[5] Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2242–2253.

[6] Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. "Safe controller optimization for quadrotors with Gaussian processes". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 491–496.

[7] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] Yinlam Chow et al. "A lyapunov-based approach to safe reinforcement learning". In: *Advances in neural information processing systems*. 2018, pp. 8092–8101.

[9] Yinlam Chow et al. "Risk-constrained reinforcement learning with percentile risk criteria". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6070–6120.

[10] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[11] Peter Geibel and Fritz Wysotzki. "Risk-sensitive reinforcement learning applied to control under constraints". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 81–108.

[12] Alexander Hans et al. "Safe exploration for reinforcement learning." In: *ESANN*. 2008, pp. 143–148.

[13] Steve Heim and Alexander Spröwitz. "Beyond basins of attraction: Quantifying robustness of natural dynamics". In: *IEEE Transactions on Robotics* 35.4 (2019), pp. 939–952.

[14] Steve Heim et al. “A Learnable Safety Measure”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 627–639.

[15] Dongho Kim, Kee-Eung Kim, and Pascal Poupart. “Cost-sensitive exploration in Bayesian reinforcement learning”. In: *Advances in neural information processing systems*. 2012, pp. 3068–3076.

[16] Teodor Mihai Moldovan and Pieter Abbeel. “Safe exploration in markov decision processes”. In: *arXiv preprint arXiv:1205.4810* (2012).

[17] Fabio Pardo et al. “Time limits in reinforcement learning”. In: *International Conference on Machine Learning*. 2018, pp. 4045–4054.

[18] Santiago Paternain et al. “Safe policies for reinforcement learning via primal-dual methods”. In: *arXiv preprint arXiv:1911.09101* (2019).

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Richard S Sutton, Andrew G Barto, and Ronald J Williams. “Reinforcement learning is direct adaptive optimal control”. In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 19–22.

[21] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. “Safe exploration in finite markov decision processes with gaussian processes”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4312–4320.

[22] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[23] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[24] Petr Zaytsev, S Javad Hasaneini, and Andy Ruina. “Two steps is enough: No need to plan far ahead for walking balance”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 6295–6300.

[25] Liyuan Zheng and Lillian J Ratliff. “Constrained upper confidence reinforcement learning”. In: *arXiv preprint arXiv:2001.09377* (2020).