UNIVERSIDADE
DE PERNAMBUCO

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Programa de Pós-Graduação Acadêmica em Engenharia de Computação

Cristian Camilo Millán Arias

Interactive Reinforcement Learning for Continuous Spaces and Dynamics Environments

Master Thesis

Recife, July 2019

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Programa de Pós-Graduação Acadêmica em Engenharia de Computação

Cristian Camilo Millán Arias

Interactive Reinforcement Learning for Continuous Spaces and Dynamics Environments

Master Thesis

Thesis presented to the Programa de Pós-Graduação Acadêmica em ENGENHARIA DE COMPUTAÇÃO at the Universidade de Pernambuco as a partial requirement to obtain the title of Master in Computer Engineering.

Prof. Dr. Bruno José Torres Fernandes
Advisor
Dr. Francisco Javier Cruz Naranjo
Co-advisor

Recife, July 2019

To my parents and my brother

Success is not final, failure is not fatal: it is the courage to continue that counts.
– Winston Churchill

Abstract

Reinforcement learning refers to a machine learning paradigm where an agent interacts with a surrounding environment to learn how to perform a task. Many times, learning is affected by the characteristics of the environment and the way how the agent perceives it. The characteristics of the environment may change over time or even be affected by external disturbances not controlled by the agent. Furthermore, discrete representations of the environment allow the learning to be fast, and the algorithms are more straightforward to learn the task. However, the information is lost during the discretization process. Moreover, in continuous spaces, the agent takes too long to find optimal actions. Some proposals solve these problems, e.g., *Interactive Reinforcement Learning* is an approach in which an external entity helps the agent to learn through feedback. There are also robust approaches, such as *Robust Reinforcement Learning*, that allow the agent to learn a task regardless of the disturbances produced in the environment. In this thesis, we propose a methodology that implements Interactive Reinforcement Learning in scenarios where states and actions are continuous and in dynamic environments. To evaluate our proposal, we implemented a simple scenario, the *Cart-pole* balancing task, where the characteristics of the environment change in each episode. The results show that the proposed approach increases the accumulated reward concerning the autonomous learning method, besides the agents are more robust against changes in the characteristics of the environment.

Keywords: Dynamic Environment. Interactive Reinforcement Learning. Machine Learning. Policy Shaping. Robust Reinforcement Learning.

Resumo

A *Aprendizagem por Reforço* refere-se a um paradigma de aprendizagem de maquina onde um agente interage com um ambiente circundante para aprender como realizar uma tarefa. Muitas vezes, a aprendizagem é afetado pelas características e a forma de como o agente percebe o ambiente. As características do ambiente podem mudar sobre o tempo ou ser afetadas por perturbações externas que o agente não pode controlar. Por outro lado, as representações discretas do ambiente permitem que a aprendizagem seja rápida, e os algoritmos sejam simples para desarrolhar uma tarefa. No entanto, a informação perde-se durante o processo de discretização. Além disso, em espaços contínuos, o agente demora muito para encontras as ações ótimas. Algumas propostas resolvem esses problemas, por exemplo, a *Aprendizagem por Reforço Interativo* é uma abordagem no qual uma entidade externa ajuda aprender ao agente através de um feedback. Também tem abordagens robustas, como *Aprendizagem por Reforço Robusto*, que permite ao agente aprender uma tarefa considerando perturbações produzidas no ambiente. Nesta dissertação, propõe-se uma metodologia para implementar Aprendizagem por Reforço Interativo em cenários onde os estados e as ações estão em espaços contínuos e o ambiente é dinâmico. Para avaliar a proposta, implementou-se um cenário simples, o problema do *Cart-pole*, onde as características do ambiente mudam em cada episodio. Os resultados mostraram que a abordagem proposta aumenta a recompensa acumulada em relação ao método de aprendizagem autônomo, além disso, o agente é robusto contra mudanças nas características do ambiente.

Palavras-chave: Ambientes Dinâmicos. Aprendizagem de Maquina. Aprendizagem por Reforço Interativo. Aprendizagem por Reforço Robusto. *Policy Shaping*.

Resumen

El *Aprendizaje por Refuerzo* se refiere a un paradigma de aprendizaje de maquina donde un agente interactúa con un ambiente que lo rodea para aprender como realizar una tarea. Muchas veces, el aprendizaje es afectado por las características del ambiente y la forma de como el agente lo percibe. Las características del ambiente pueden cambiar en el tiempo o ser afectadas por perturbaciones externas que el agente no puede controlar. Por otro lado, las representaciones discretas del ambiente permiten que el aprendizaje sea rápido, y los algoritmos sean simples de implementar en el desarrollo de una tarea. Sin embargo, la información se pierde durante el proceso de discretización. Además, en espacios continuos, el agente toma un tiempo en encontrar las acciones optimas. Algunas propuestas resuelven estos problemas, por ejemplo, el *Aprendizaje por Refuerzo Interactiv*o es un enfoque en el cual una entidad externa ayuda a aprender al agente a través de una retroalimentación. También, existen enfoques robustos, como el *Aprendizaje por Refuerzo Robusto*, que permite al agente aprender una tarea considerando perturbaciones que modifican el ambiente. En este trabajo, se propone una metodología para implementar Aprendizaje por Refuerzo Interactivo en escenarios donde los estados y las acciones están en espacios continuos y el ambiente es dinámico. Para evaluar la propuesta, se implementó un escenario simple, el problema del *Cart-pole*, donde las características del ambiente cambian en cada episodio. Los resultados muestran que la metodología propuesta aumenta la recompensa acumulada en relación con el método de aprendizaje autónomo, además, el agente es robusto contra cambios en las características del ambiente.

Palabras-clave: Ambientes Dinámicos. Aprendizaje de Maquina. Aprendizaje por Refuerzo Interactivo. Aprendizaje por Refuerzo Robusto. *Policy Shaping*.

Contents

List of Figures

List of Tables

List of abbreviations and acronyms

AC	Actor-Critic (from Actor-Critic Algorithm)
ADC	Actor-Disturber-Critic (from Actor-Disturber-Critic Algorithm)
IRL	Interactive Reinforcement Learning
IRRL	Interactive Robust Reinforcement Learning
MDP	Markov Decision Process
MLP	Multilayer Perceptron
RL	Reinforcement Learning
RRL	Robust Reinforcement Learning
TD	Temporal Difference
π	The policy
V^{π}	The state value function under the policy π
V^{*}	The state value function under the optimal policy π^{*}
Q^{π}	The state-action value function under the policy π
Q^{*}	The state-action value function under the optimal policy π^{*}
δ_t	The TD error

Acknowledgements

This master's thesis could not be concluded without the support of several people.

I cannot fail to thank my advisor, Professor Doctor Bruno José Torres Fernandes, for the trust, patience, support and for the opportunity to work with him, and for the guidance. I also want to thank my co-advisor, Doctor Francisco Javier Cruz Naranjo, for his advice, for his support, for patience and for all the guidance.

On the other hand, I want to thank my cousins Diego José Rativa Millán and Miguel Alejandro Zorro Millán for receiving me in Recife and accompanying me in the process of adaptation in the country. To my uncles Saul Efren Cruz Torres and Marco Tulio Millán Ramos for their collaboration and support during the master's degree. I also thank all the people who accompanied me during these two years of expertise, for their friendship and support.

Finally, and not least, I thank my parents, Yesid Millán Ramos and Irma Naidú Arias Cruz, and my brother, Ivan Felipe Millán Arias for the support and good encouragement they offered me.

I also would like to gratefully acknowledge to financing in part by the Coordenacão de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the Brazilian agencies FACEPE and CNPq. Also, the partial support by Universidad Central de Chile under the research project CIP2017030.

Chapter 1

Introduction

This work proposes a methodology to implement Interactive Reinforcement Learning for continuous spaces and an approach to dynamic environments. A methodology is developed to include advice so that the agent policy is affected. Furthermore, the policy gradient methodology was combined with the so-called Robust Reinforcement Learning to select actions and disturbances. Finally, an algorithm is proposed that includes advice in Robust Reinforcement Learning to perform actions in dynamic environments.

1.1. Problem Characterization

Reinforcement learning (RL) is a learning approach that tries to solve the problem of what to do, in which agents interact with an environment to learn the desired task autonomously. The agent must be able to sense a state from the environment and take actions that affect it to reach a new state. For each action taken, the agent receives from the environment a reward signal that will try to maximize it throughout the learning [1]. Thus, the agent will learn from his own experience, taking actions, and discovering which one produces the greatest reward. The agent does not learn the policy directly, but it can approximate it storing values for each state and each action. Many RL algorithms approximate the value function, which maps state or state-action pairs in an expected reward amount. Among the algorithms are the Q-learning [4], temporal difference TD [1], SARSA algorithm, that update every state-action pair [1], and the Actor-Critic, that represents the policy (actor) independent of the value function (critic) [5, 6].

One RL problem, which is still open, is the time spent by an RL agent during learning. To leave that an agent finds a proper policy requires excessive time, mainly due to a complex or large space of states and actions. Besides, the agent may explore different regions of space to find the state-action pair that produces a better reward [7]. To overcome this problem, sometimes an RL agent may be guided by a trainer in order to help the agent to perform the task more rapidly. Interactive Reinforcement Learninf (IRL) is an approach that allows an external trainer to perform a feedback process with an RL agent. In this process, the trainer advises the RL agent

to improve the performance of the task and speed up the learning. This allows to reduce the search space and to learn the task faster compared to an RL agent that performs exploration autonomously [8]. Thomaz and Breazeal [7] present the first IRL approach where enables a human trainer to provide positive and negative rewards, and anticipatory guidance to performs future actions. First, they experiment whether the human reward is compatible with the reward signal in the classic RL, in their work they show that the users (humans) guide to agent towards action and give anticipatory rewards but provides more positive than negative feedback. In their next work [9], they involve a boolean feedback signal by the trainer when the agent needs it, and, the *UNDO* behavior that the trainer guides to the agent from changing the action selected by the opposite action.

In many RL implementations, the space of states and actions is usually considered discrete, but in real-world applications, the set of states and actions can be more complex and challenging to represent. A priori discretization prevents to identify which regions of space are more important than others. Moreover, information is lost, and it is difficult to learn from past experiences [10, 11]. Nevertheless, it can be considered a very fine discretization to capture all the possible information. However, this leads to slow learning by considering spaces with many elements. Doya [12] implements a framework of reinforcement learning in continuous spaces of time, states and actions, also proposes the continuous Actor-Critic method and a gradient approximation to performs the policy. In this work, the author uses Normalized Gaussian Networks [13] as function approximation. Van Hasselt and Wiering [11] propose the Continuous Actor-Critic Learning Automaton (CACLA) to carry actions and states in continuous spaces, using Neural Networks as function approximation.

It is clear that during learning, the agent performs actions that modify the environment. Depending on how it explores, the agent can obtain samples of the states to improve his policy and perform a better task in the future. One of the main problems is when the environment is not controlled, i.e., it is not guaranteed that the environment is kept in constant condition, avoiding some external noise input. For example, the wear of a wheel or its friction may vary over time. In a maze, the walls could change position eliminating paths that the agent learned in the past. Therefore, it is important to develop robust algorithms that help to overcome uncontrollable disturbances. One of the most characteristic works is of Morimoto and Doya [14], they propose Robust Reinforcement Learning, an approach capable of resisting the disturbances present in the environment based on the control paradigm H^{∞}. They implement their methodology in the Cart-pole balancing task where they change the physical parameters of the Cart-pole after training.

In this research, we propose an Interactive Reinforcement Learning methodology to act in scenarios where states and actions belong to continuous space, and the environments have dynamic features independent from the performance of the agent. To evaluate this methodology, we implemented the classic control problem, Cart-pole balancing task, with the characteristics

necessary to evaluate the performance of the method. To determine the performance of the proposal, we compare the curves of the reward collected and the number of steps per episode.

1.2. Objectives

This work aims to propose a new robust methodology to implement Interactive Reinforcement Learning where states and actions are in a continuous domain; this proposal is also implemented in dynamic environments so that external factors to the environment are not influential during learning. The methodology will be efficient if it is able to accelerate the learning of an agent that faces this type of environment.

We highlight the specific objectives:

1. To propose an algorithm to implement Interactive Reinforcement Learning in scenarios where states and actions are continuous.
2. To inquire about robust Reinforcement Learning methodologies based on dynamic environments.
3. To include the concepts of Interactive Reinforcement Learning in dynamic environments based on the methodology of state of the art.
4. To propose an Interactive robust reinforcement Learning algorithm to learn about dynamic environments.
5. To verify that the proposed algorithms are effective in a classic control benchmark.

1.3. Structure of the Document

This master thesis is organized in five chapters. Chapter 2 presents the basic knowledge about Reinforcement Learning for the understanding of this work. Chapter 3 presents a proposal to implement Interactive Reinforcement Learning for continuous spaces, a Robust Reinforcement Learning approach with policy gradients and including advice from an external trainer is also presented. In chapter 4, the experiments and the results, the configurations of the proposed approaches are specified. Finally, chapter 5 presents the main conclusions of this work and suggestions for future work.

Chapter 2

Theoretical Framework and Related Approaches

2.1. Reinforcement Learning

Learning, as a pattern of human and animal behavior, is the process by which a modification in stimulus-response relations is developed as a consequence of interaction with the environment via the sense [15]. Therefore, an approach based on learning is appropriate when the environment is full unknown, or it is not available at the moment of designing a solution; this provides autonomy to one organism.

It is generally agreed that learning involves a relatively permanent change in behavior due to experience, rewards, and punishments. It is a consequence of the law of reinforcement, the most essential principle in all learning theory [16, 17]. Reinforcement Learning (RL) [1] is an learning approach that allows autonomous agents to learn, using the reaction from the environment to an action. The main idea is trying to select actions in order to observe what situations occur in the environment. If an action produces a favorable reaction, the organism trend to apply this behavior again; otherwise, the tendency is to avoid such behavior in the future [17]. The RL problem is reduced to learn how to select optimal actions to be performed in each situation to reach a given goal [18].

2.1.1. Elements of Reinforcement Learning

Besides the agent and its environment, exist fourth main elements that can be identify in a Reinforcement Learning task: a policy, a reward function, a value function and a model of the environment.

A *policy* defines the way the agent behaves at a specific time. Roughly speaking, the policy is a process that associates the perceived states and the actions taken in those states. In learning theory and psychology, this corresponds to the stimulus-response rules. A policy is the

core of the reinforcement learning agent in the sense that is enough to determine the behavior [1].

A reward function defines the aim of a reinforcement learning problem. It associates each perceived state (or state-action pair) of the environment to a numerical value that determines the favorable and unfavorable of that state. The only objective from reinforcement learning agent is to maximize the reward received in the long run. In biological organisms, the reward can be related to pleasure or pain.

A value function is a long-term desirability indicates how good is the state with respect to the goal. Therefore, starting from a anyone state, the value is the accumulative reward over the future an agent can expect receive [1].

A model is something that imitates the behavior of the environment. It is how the reactions to the stimulus given by an external agent are represented [17]. for instance, given a state and an action, the model must predict the next new state and the new associated reward.

2.2. Reinforcement Learning Framework

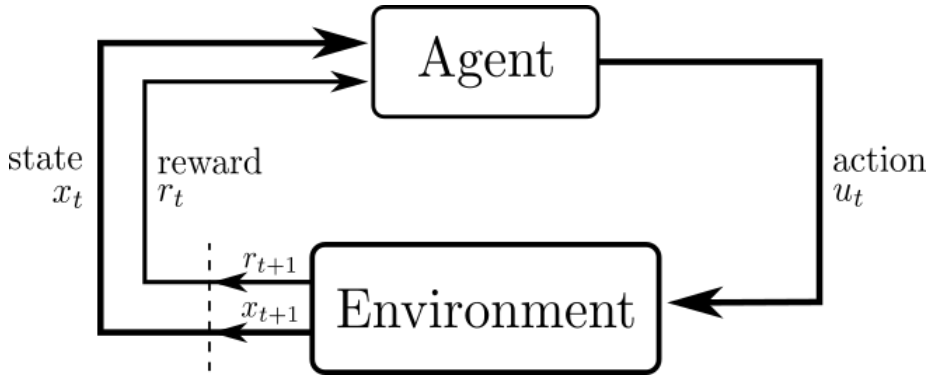
Reinforcement learning is a learning approach that involves an autonomous agent learning from interactions with its environment to achieve the goal [1]. The agent must be able to learn from its own experience selecting actions that affect the environment and with these actions reach new situations to the agent. In this approach, the learning agent receives from its environment a numerical reward signal from the environment that attempts to maximize [19]

The agent and the environment interact at each of a sequence of discrete time step $t = 0, 1, 2, ...$ At each time step, the learning agent receives a representation of the state of the environment $x_t \in X$, where X is a set of the all possibles states, and of the action selected by the agent $u_t \in U(x_t)$, where $U(x_t)$ is a set of the actions available in the state x_t[1]. Eventually, as a result of performing an action, the agent receives a scalar reward $r_{t+1} \in \mathbb{R}$ and reaches a new state x_{t+1} [1]. Fig. 1 shows the diagram of the interaction between the agent and the environment in the context of RL.

At each time step, the agent implements relations between the states to probabilities to select each possible action. These relations are called the agent policy and denoted by π_t, where $\pi_t(u_t|x_t)$ is the probability to select u_t given the current state x_t at time t. Reinforcement Learning method specifies how the agent change the policy as consequence of its experience. Thereby, the goal of the reinforcement learning agent is to approximate a function $\pi : X \times U \longrightarrow (0, 1)$ such that maximizes the total amount of reward it receives over the long run.

[1] The nature of the sets is not specified in order to generalize the methodology to sets with more complex characteristics.

Figure 1 – Interaction between the agent and the environment in the Reinforcement Learning context. Figure adapted from Sutton and Barto [1].



2.3. Markov Decision Process

Reinforcement learning problem can be easily described using a Markov Decision Process (MDP). An MDP is specified by the tuple $< X, U_x, f, \rho >$ where [20]:

- X is a set of the states and U_x is a set of actions available in the state x[2].
- f is the transition function, $f : X \times U \longrightarrow X$, that gives the next state given the action selected and the current state.
- ρ is the reward function, $\rho : X \times U \times X \rightarrow \mathbb{R}$, this function evaluates the immediate performance of the action selected.

In an MDP, as a result of selecting an action in the current state, the next state is determined by a transition probability, and a reward amount is received [21]. Thus, the dynamic system is wholly determined when the current state is known, besides, the state maintains the Markov property.

The action u_t taken in the state x_t is selected following the policy π. The policy can be either deterministic or stochastic: in deterministic case, the policy is determined by a function $\pi : X \longrightarrow U$ that maps the action u_t in function of state x_t. In stochastic case, the policy is a probability density distribution function $\pi : X \times U \longrightarrow (0, 1)$, therefore, the action u_t is drawn aleatory from π given the current state x_t.

2.3.1. Task and Return

Solving an MDP implies finding the policy that maximizes an optimality criterion, in particular, maximizing an expected return [22]. The return, R_t, can be defined as a function of a reward sequence, $r_{t+1}, r_{t+2}, r_{t+3}, ...$, received at time t. A particular case, the *return* is [1]

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T, \tag{2.1}$$

[2] The sets X and U_x can be arbitrary finite or countably infinite, discrete or continuous sets [21].

where T is a final time step. Although the criterion of the expression (2.1) is valid, trying to maximize R_t could converge to infinity in scenarios where the agent-environment interaction does not end naturally ($T \longrightarrow \infty$). The corresponding tasks are referred to as continuing task, when the interaction process goes on continually without limit [1]. Eventually, in episodic task, the agent-environment interaction is performed in finite sequences of steps called episodes. Each episode ends in a terminal state and consequently restarts to a standard state or to a random state defined by a distribution of initial states [23].

A criterion based in discount is the most commonly used; thus, the agent tries to select actions, so that maximize the cumulative sum of discounted rewards received over the future:

$$R_t^{\gamma} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \tag{2.2}$$

where γ is a parameter, $0 < \gamma \leq 1$, called the discount rate and T is the horizon, including the possibility $T = \infty$[3].

2.3.2. Value function and Bellman Equations

In several MDP algorithms, optimal policies are computed by learning value function. A value function represents an estimate of how good is the agent to perform a particular action in that state, expressed in term of expected return [24].

The value of state x of *state value function*, over all the actions, is the expected return of the discounted sum starting in the state x an following a policy π thereafter. Formally we can defined it by [1, 24]:

$$V^{\pi}(x) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| x_t = x\right], \tag{2.3}$$

where $E_{\pi}[\cdot]$ denotes the expected value[4] given that the agent follows the policy π.

Similarly, it is defined as a *state-action value function* as the value of taking action u in the state x under the policy π. Roughly speaking, it can be defined as the expected return starting from state x, taking action u and following the policy π [1]

$$Q^{\pi}(x, u) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| x_t = x, u_t = u\right] \tag{2.4}$$

In addition, the value functions satisfy a recursive property. The expression (2.3) can recursively be defined in terms of a so-called *Bellman equation* [1, 25]. It is denotes the expected return in terms of the immediate reward and the value of the next state, defined formally by:

$$V^{\pi}(x) = \int_{\mathbb{U}} \pi(u|x) \int_{X} P(x'|x, u) \left[\rho(x, u, x') + \gamma V^{\pi}(x')\right] dx' du, \tag{2.5}$$

[3] In episodic task, it is only accepted $\gamma = 1$.

[4] It is clear that the expected value is defined by a summation or integrals according to the nature of the sets X and U.

where x is the current state, u the selected action and x' is the new state reached by perform the action u in the state x. In the expression, $\pi(u|x)$ is the probability of select the action u given the current state x, $P(x'|x,u)$ is the probability of reach the state x' given the current state x and the selected action u. For discrete or finite sets, the integrals are adequately replaced by summations with index u and x' respectively.

2.4. Temporal Difference Learning

A learning agent observes an input state from the environment, it selects an action from a policy π, and then it receives scalar reward feedback, indicating how was the performance of its input. The reinforcement learning goal is to find an optimal policy leading to maximizing the reward over the long run [26]. Optimal policies are denoted by π^* and share the same optimal value functions which are denoted by $V^*(x)$ and $Q^*(x,u)$ [2]. These optimal value functions can be solved through the equation in (2.5).

The *temporal difference learning* (TD) is a method for solving the Bellman equation. Algorithms based on TD learn estimates values based on another estimates[5] by adjusting the gain against the ideal equilibrium that holds locally when the gain estimates are correct. In each step, it generates a learning example that approximates some value concerning the immediate reward and the value of the next state or state-action pair [27, 24]. Therefore, when occurs a transition from the state x_t to state x_{t+1}, the tabular update from the value function $V^*(x_y)$ is:

$$V^*(x_t) \leftarrow V^*(x_t) + \alpha \left[r_{t+1} + \gamma V^*(x_{t+1}) - V^*(x_t) \right]. \tag{2.6}$$

Here α is a learning rate that is determined by how much values get updated [27]. This TD method is known as $TD(0)$ [1]. Algorithm 1 shows a completely episodic learning method $TD(0)$ with an iterative tabular update of $V(x_t)$. Other methods based on temporal difference learning estimates $Q^*(x_t, u_t)$ rather than $V(x_t)$ for some policy π. The Q-Learning [4, 28], SARSA [29], and R-Learning [1] are TD methods that learn the value function based on state action pairs and computing the state-action value function. In this work, we do not focus on these methods; however, the state-action value function would be essential to introduce topics in future sections. Following, we present the Actor-Critic algorithm, an iterative method and basis of our methodology.

2.4.1. Actor-Critic

The Actor-Critic algorithms (AC) are method based on TD learning that keeps a separate memory structure to represent the policy independent of the value function [5, 1]. The agent is separate into two entities: the actor and the critic. The policy takes the role of the actor, selecting

[5] The bootstrapping methods are strategies to estimate althrough others estimates; these are often used in RL and Dynamic Programming problems [1].

Algorithm 1 General algorithm of episodic TD learning from tabular $TD(0)$.

Inputs: γ, α

Initialize $V(x_t)$ arbitrarily, π to the policy to be evaluated
for each episode **do**
  initialize x_t
  repeat
    $u_t \leftarrow$ given by $\pi(u|x_t)$
    Taken action u_t, observe reward r_{t+1} and next state x_{t+1}
    $V(x_t) \leftarrow V(x_t) + \alpha\left[r_{t+1} + \gamma V(x_{t+1}) - V(x_t)\right]$
    $x_t \leftarrow x_{t+1}$
  until x_t is terminal
end for

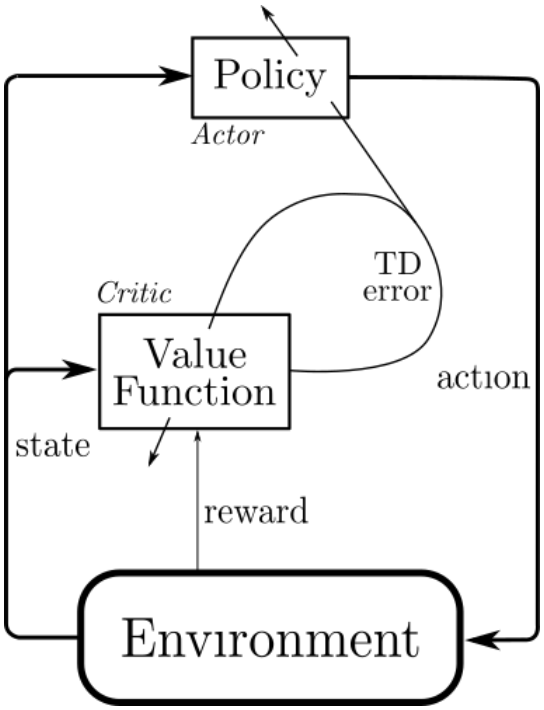
actions in each iteration. The critic, commonly a state-value function, evaluates or criticizes the actions performed by the actor [30]. Fig. 2 shows the schematic structures of the AC algorithm.

In each iteration, the critic values each action through TD error:

$$\delta_t = r_{t+1} + \gamma V(x_{t+1}) - V(x_t). \tag{2.7}$$

This signal is output of the critic and the diference between the right hand and left hand sides of the Bellman equation (2.5). The propose of TD error is determined if the task has gone better or worse than expected. If the TD error is positive, the selected action tend to be strengthened for the future, although if the TD error is negative, the tendency should be weakened that action selection in a determined state. [24]. The action is the output of the actor. The preference to select each action is improved by the TD error, in each iteration is adjusted depending on the

Figure 2 – Schematic overview of an Actor-Critic algorithm. Taken from Sutton and Barto [1].



performance of the agent control over the environment. Exists variations of the AC method to obtain actions in different ways, for example [1]: suppose a tabular case when the actor generates actions based in a modifiable parameter $\upsilon(u_t, x_t)$, this represents the preference of taking action u in the current state x. The preference can increase or decrease using

$$\upsilon(u_t, x_t) \leftarrow \upsilon(u_t, x_t) + \alpha_\upsilon \delta_t. \tag{2.8}$$

In the following sections will be introduced other methods to select actions based on AC algorithms. An advantage of using AC algorithms is the separate structure of the policy; they require minimal computation to select actions. If there are many actions, or in large action space, it is no necessary to consider the $X \times U$ or all Q values of the actions to select them. A second advantage is that they can learn stochastic policies, i.e., learn optimal probabilities to choose from a set of actions.

2.5. Policy Gradient Methods and Function Approximation

Policy gradient methods [31] are a methodology to approximate functions in RL. These methods are the most popular class of continuous action RL algorithms [32]. With this approach, a stochastic policy is approached through an approximation function, independent of the value function, with its own parameters. A measure is used to improve the performance of the policy, and adjusts the parameters.

In this context, consider a standard RL setting presented in section 2.2 and section 2.3. The Policy Gradient framework uses a stochastic policy π that is parametrized by a column vector of weights $\upsilon \in \mathbb{R}^{N_a}$, for $N_a \in \mathbb{Z}$, such that $\pi(u|x)$ denotes the probability density for taking an action u in the state x. The objective function $\Gamma(\pi)$ maps policies to scalar measure of performance, defined by:

$$\Gamma(\pi) = \int_X d^\pi(x) \int_U \pi_\upsilon(u|x) Q^\pi(x,u) du dx, \tag{2.9}$$

where $d^\pi(x) := \int_X \sum_{t=0}^{\infty} \gamma^{t-1} P(x|x_0, u)$ is the stationary distribution of the discounted states occupancy under π [33] and $Q^\pi(x,u)$ the state-action value function defined in (2.4). The basic idea of policy gradient methods is adjust the parameter υ of the policy in direction of the gradient $\nabla_\upsilon \Gamma(\pi)$:

$$\upsilon_{t+1} - \upsilon_t \approx \alpha_\upsilon \nabla_\upsilon \Gamma(\pi).$$

The fundamental result of these methods is the *policy gradient theorem* [31], this define the gradient as:

$$\nabla_\upsilon \Gamma(\pi) = \int_X d^\pi(x) \int_U \nabla_\upsilon \pi_\upsilon(u|x) Q^\pi(x,u) du dx \tag{2.10}$$

The main problem with the algorithm is finding an approximation of the gradient. It is also to consider the explicit relationship between the policy gradient and the value function. Therefore

it is crucial to consider a candidate to represent the value function [31, 34]. Consider $h_\theta : X \times U \longrightarrow \mathbb{R}$ as an approximation of the value function $Q^\pi(x,u)$ with the parameter $\theta \in \mathbb{R}^{N_c}$, for $N_c \in \mathbb{Z}$, so that it does not affect the unbiasedness of the policy gradient estimate. To find a close approximation of $Q^\pi(x,u)$ by $h_\theta(x,u)$, we try to find θ to minimize the quadratic error of the approximation as follows

$$\epsilon_t^\pi(x,u) = \frac{1}{2}\left[Q^\pi(x,u) - h_\theta(x,u)\right]^2. \tag{2.11}$$

The gradient of the quadratic error can be used to find an optimal value of θ. Considering the approximation of Q^π by h_θ, equation (2.10) would be expressed as:

$$\nabla_\upsilon \Gamma(\pi) = \int_X d^\pi(x) \int_U \nabla_\upsilon \pi_\upsilon(u|x) h_\theta(x,u) du dx \tag{2.12}$$

This approximation is acceptable if $\nabla_\theta h_\theta(x,u) = \nabla_\upsilon \log\left(\pi_\upsilon(u|x)\right)$ is satisfied. The expression is called compatible features [31, 6], and establishes that the features of the value function (in the form of the gradient) are compatible with the features of the policy.

Under this approach, the expected value of h_θ with respect to policy π is zero, indicating that the value function has zero mean in each of the states. The most convenient is to approximate an advantage function $A^\pi(x,u) = Q^\pi(x,u) - V^\pi(x)$ instead of $Q^\pi(x,u)$ [35]. This implies that the approximation function only represents the relative value of an action u in some state x and not the absolute value of Q^π [34]. The value function $V^\pi(x)$ is a baseline in the advantage function, such that the variance of the policy gradient is minimized [36].

2.5.1. Actor-Critic with Policy Gradient

In other view, as $Q^\pi(x,u) = E_\pi\left[r_{t+1} + V^\pi(x_{t+1})|x_t = x, u_t = u,\right]$, then [1]

$$A^\pi(x,u) = E_\pi\left[r_{t+1} + V^\pi(x_{t+1}) - V^\pi(x_t)|x_t = x, u_t = u,\right],$$

than is the expected value of the TD error (2.7). Thus h_θ would approximate a value function $V^\pi(x)$, and the gradient in (2.12) taken form of:

$$\begin{aligned}\nabla_\upsilon \Gamma(\pi) &= \int_X d^\pi(x) \int_U \nabla_\upsilon \pi_\upsilon(u|x) \hat{\delta}_t du dx \\ &= E_\pi\left[\hat{\delta}_t \nabla_\upsilon \log\left(\pi_\upsilon(u|x)\right)\right]\end{aligned} \tag{2.13}$$

where $\hat{\delta}_t = r_{t+1} + \gamma h_\theta(x_{t+1}) - h_\theta(t)$, and $h_\theta(x) : X \longrightarrow \mathbb{R}$ the function approximation of value function $V^\pi(x)$.

In the context of AC, let $h_\theta(x) = V_\theta(x)$ be the approximate state value function from the critic. The update from the value function is through of θ. The parameter θ is adjusted by the gradient of the quadratic error (2.11) as follows:

$$\theta_{t+1} = \theta_t + \alpha_\theta \hat{\delta}_t \nabla_\theta V_{\theta_t}(x_t),$$

where $\alpha_\theta > 0$ is a step size parameter of the critic. It is clear that the gradient of the quadratic error is the gradient of the approximation scaled by the TD error. In other hand, the policy, that represents the actor, is updated based in the gradient in (2.13) as follow:

$$\upsilon_{t+1} = \upsilon_t + \alpha_a \hat{\delta}_t \nabla_\upsilon \log\left(\pi_\upsilon(u|x)\right),$$

where $\alpha_\upsilon > 0$ is a step size parameter of the actor. Note that the parameter υ and the preferences $\upsilon(u, x)$ in (2.8) are similarly updated except for the gradient of the approximation. The algorithm 2 shows a general episodic AC algorithm with policy gradient and function approximation, this algorithm will be the basis for implementing the proposed methodology in the following chapters.

Algorithm 2 General episodic Actor-Critic algorithm with Policy Gradient.

Inputs: γ, α_c, α_a

Initialize θ_t and υ_t arbitrarily
for each episode **do**
  initialize x_t
  repeat
    $u_t \leftarrow$ given by $\pi_\upsilon(u|x_t)$
    Taken action u_t, observe reward r_{t+1} and next state x_{t+1}
    $\delta_t \leftarrow r_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
    $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t)\right)$
    $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
    $x_t \leftarrow x_{t+1}$
  until x_t is terminal
end for

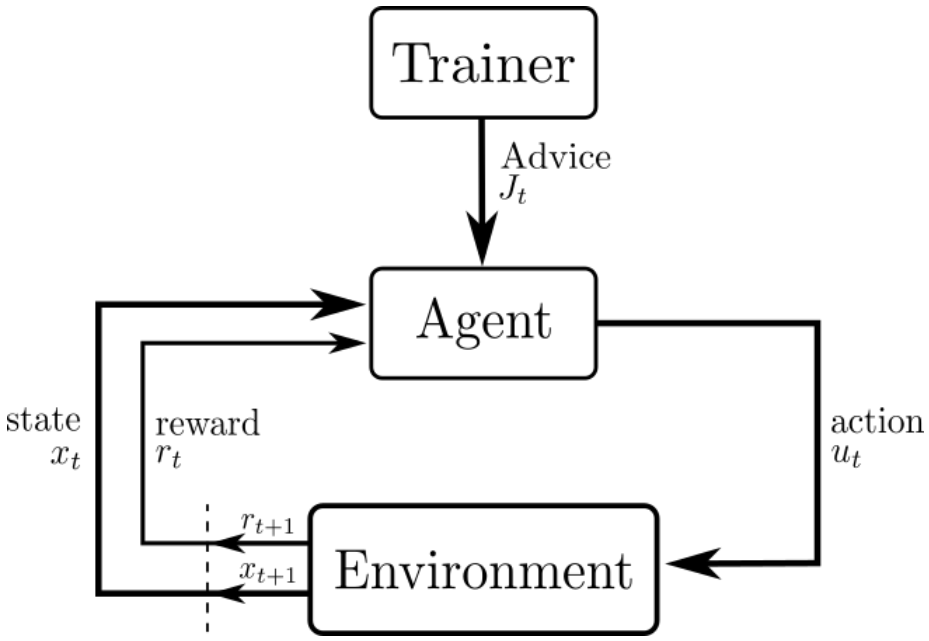
2.6. Interactive Reinforcement Learning

As aforementioned, RL is a learning technique based on trial-and-error in association with an environment. In each step, a stimulus-response process is performed to find optimal actions and achieve an objective. An autonomous agent can feel states of an environment and is also able to select actions that influence the environment in a certain way. The goal of the agent is to try to maximize the received reward for the entire task [37].

On some occasions, to leave that an agent learn a task by itself is impractical and involve problems to find the proper policy [38]. Interactive Reinforcement Learning (IRL) is an approach that considers a knowledgeable trainer it gives advice or guidance to the RL agent, having an effect of restricting the actions selection to those related to the target object [39]. Fig. 3 shows a general view of the IRL approach, an external trainer is added during the learning process to modify the agent decision.

In an IRL scenario, it is desired that the interaction rate between the external trainer and the agent be as minimal as possible, otherwise, it could become supervised learning. Another

Figure 3 – Interactive Reinforcement Learning scheme including an external trainer. Taken from Cruz [2].



important aspect is the quality of the advice, since the external trainer may make mistakes the agent may not improve with its learning [40]. The advice can be obtained from expert or non-expert trainers, artificial agents with perfect knowledge of the task or previously trained [8].

There are two approaches to receiving advice from an external trainer [41]. The first is *reward shaping*, where the external trainer modifies the reward by sending its own reward to the agent to inform how good the selection of the action was in the previous step [42, 39]. The second approach is *policy shaping*, where the external trainer modifies the action just selected by the agent, this tells the agent that its current performance was wrong and should improve for the future [7, 38].

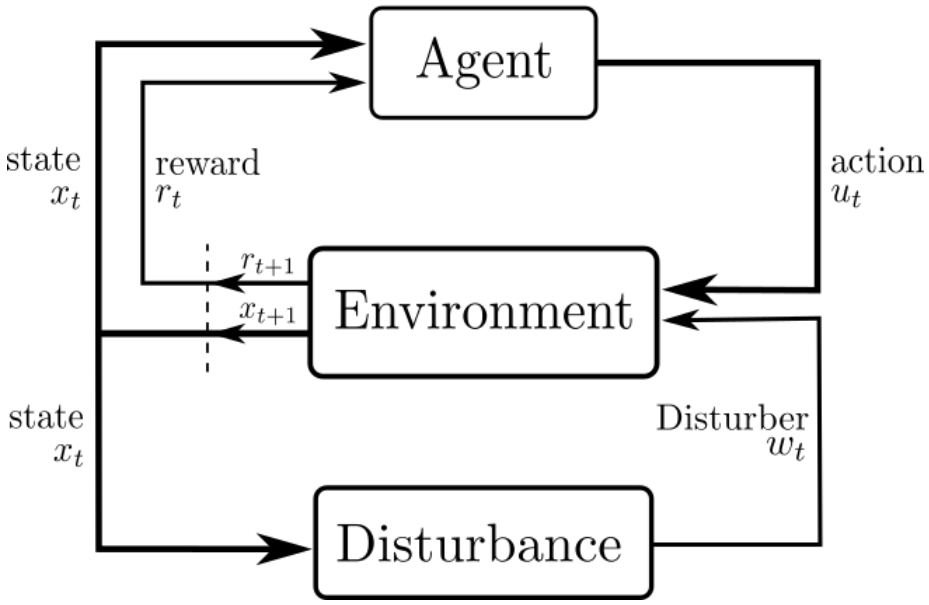
2.7. Dynamic Approach: Robust Reinforcement Learning

During the learning, the agent performs an action that stimulates the environment in some way. Assuming that the environment is governed by a parametric system (deterministic or stochastic), the action acts as an input that modifies the output values, but not the model of the system. There could be errors in the system, independent of the actions, which randomize it by disturbing the output of the system. Errors can be strongly correlated, so assumptions about independence may not be valid in these systems [43]. There are also parameters in the system that change concerning time; these parameters can be independent of actions and states. These properties are characteristic of a dynamic environment, in the sense that some features of the system change independently control agent.

An RL agent has to interact with its environment to gather enough knowledge about the

desired task. However, the environment can give different amounts of reward for the same action (or for a subset of actions). Different approaches consider robust agents with less sensitivity to these changes, and with the ability to resist an entry disturbance.

Figure 4 – Reinforcement Learning scheme including a dynamic environment. The environment is affected by the disturbance input that modifies the state output.



Morimoto and Doya [14] present the Robust Reinforcement Learning (RRL), an approach that introduces a disturber who provides noise to the environment. To resist a disturbance input, it considered an additional reward $\omega(w_t)$ that modifies the reward of the environment, where $w_t \in \mathbb{W}(x_t)$ is the disturbance input, and $\mathbb{W}(x_t)$ is a set of the disturbing input in the state x_t. Therefore, the increased reward is defined by

$$q(u_t, x_t, w_t) \leftarrow \rho(u_t, x_t) + \omega(w_t). \tag{2.14}$$

Eventually, as a result of performing an action, the disturbance is generated by a function $\kappa : X \longrightarrow W$, and the agent receives an augmented reward q_{t+1}. The function $\omega(\cdot)$ is taken as a quadratic cost so that it can withstand the maximum possible disturbance [44]. Fig. 4 shows a view of the RRL approach [14], where a external disturber that provides noise to the environment is added to the learning process.

2.8. Related Approaches

In the area of RL, there are diverse researches oriented to various problems. In many works, representations of the space of states and actions that enhance the loss of information are used. Many others show that learning is slow and the task cannot be solved properly. With these

approaches, only discrete advice is considered, also that the action belongs to a discrete space so it could hardly be generalized to a space of greater complexity.

For continuous spaces, we can identify the work of Doya [12] where implements a framework of RL in continuous spaces of time, states and actions, also proposes the continuous Actor-Critic method and a gradient approximation to performs the policy. Van Hasselt and Wiering [11] propose the Continuous Actor-Critic Learning Automaton (CACLA) to carry actions and states in continuous spaces. Policy gradient methods and function approximator [10, 31] are more used in continuous RL. Researches in robotics and parametrize motor skill are developed and applied in the works [45, 46, 33]. An advantage of using several Actor-Critic algorithms [34] is consider different policy nature. Silver et al.[32] present a deterministic policy gradient algorithms, this approach reaches better results by integrating only states to improve the policy.

In the context of interaction with external trainers, [47] presents training by demonstration, in this work the agent learns from the human trainer through demonstrations, they show a speed convergence using traditional RL methods and using bias to address the agent exploration to cover the search space adequately. Subramanian et al. [48] presents exploration by demonstration. In his work, he guides the exploration of agents through human demonstrations. They find an advantage over other interaction-based methods of accelerating learning. Their implementations are carried out in environments with discretized spaces; however, they use function approximation to estimate the value function. Thomaz and Breazeal [7] present the first IRL approach, an approach where an external trainer guides the agent to complete a task.

Stay and Chernova [39] present a study the IRL in a real-world robotic system. This work shows that the trainer makes a feedback process with the agent: it provides a reward amount depending on whether it just did was good or bad, and, it can drive it to select the subsequent action [42]. When the action and state spaces grown, using guidance trainer significantly reduce the learning rate and its impact increases in more complex spaces.

Griffith et al. [40] proposes the advice method which uses two likelihoods, $\mathcal{C}$ to refer to the consistency of feedback which comes from an external trainer, and $\mathcal{L}$ to refer to the probability of receiving feedback. Besides, this methodology considers two sources of variation, the agent policy, and feedback policy. The authors propose that the feedback policy be the optimal probability of performing a pair (x, u); this is computed using the "right" and "wrong" labels associated with it.

Cruz et al. [8, 2] integrate IRL and contextual affordances; the proposal presents an improvement in the rates of success, a fewer number the actions are performed during the learning and faster convergence is reached when including a negative reward after to perform an action. They carry out their implementation in a robotic domestic scenario, where an agent tries to learn to perform a domestic task.

In the area of dynamic environments, Sutton [49] presents a problem of changing environments where it implements Q-Learning to learn a navigation task in a scenario where obstacles change position. Morimoto and Doya [14] present Robust Reinforcement Learning, a methodology to train more robust agents against dynamic environments. They propose the Actor-Disturber-Critic, an algorithm that introduces a disturbance in the classic Actor-Critic. Although learning is slow, they implement continuous domains for actions. Obayashi et al. [44, 50] implement the Robust Reinforcement Learning methodology using the concept of sliding mode control on the inverted pendulum.

Dongsun and Park [51] use an approach where situations (transition from state to state) are related to settings (environment characteristics) during learning. The goal is to find the best relationship between situations and settings. Applying Q-Learning they apply the algorithm in Robocode where they simulate a battle between robots. Puriel-Gil et al. [43] use PD control (Proportional-Derivate control) to adapt actions to environments that change their characteristics after learning. They train from a Q-Learning algorithm, extract the matrix Q, and make a modification when selecting actions. The modification is made by adding new parameters to the action. The implementation is done in cart-pole balancing, modifying the weight of the pendulum.

Finally, there are some works on RL with external feedback in continuous spaces. Vien y Ertel [52] propos ACTAMER, an extension of TAMER RL for continuous states and actions that includes an external reinforcement signal to improve learning. Their proposal was implemented in the Cart-pole balancing task and the Mountain-cart task. Stahlhut et al. [53] powered by Interactive Continuous Actor-Critic Automaton (ICACLA), an algorithm based on CACLA [11] that includes an external capacitor. Based on the policy shaping methodology and the capacitor guides the agent to change the selected action through UNDO [7]. Millán et al. [54] proposes a methodology based on policy shaping to include variable advice on an Actor-Critic algorithm for states and continuous actions.

Chapter 3

Interactive Reinforcement Learning Approach for Continuous Action Space and Dynamic Environments

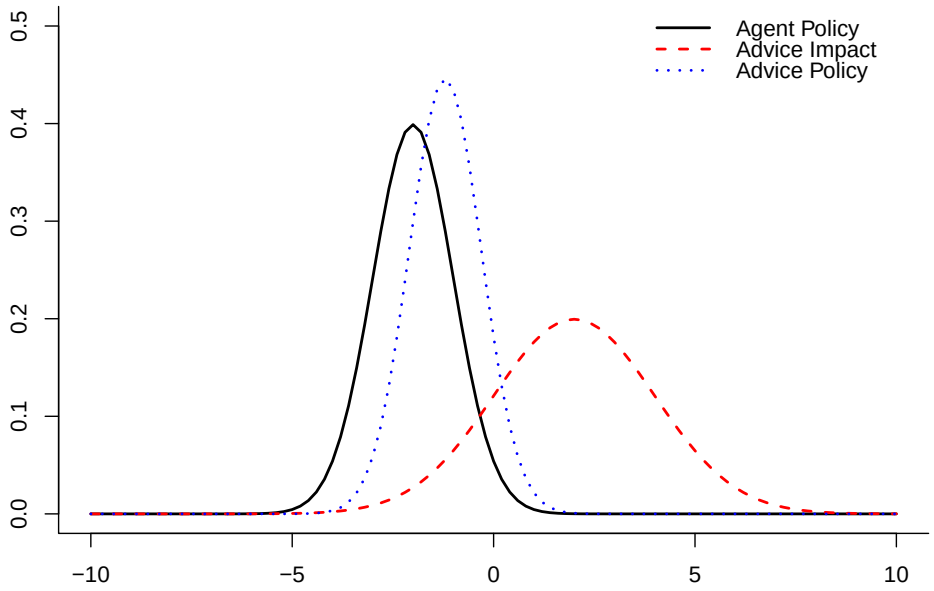
In this chapter, the proposed methodology for the development of the problem is discussed. First, the issue of Interactive Reinforcement Learning for continuous spaces will be addressed. Then, a methodology to implement Robust Reinforcement Learning with policy gradients. Finally, Interactive Robust Reinforcement Learning will be addressed.

3.1. Policy Shaping Approach

Consider the standard RL setting presented in section 2.2 and 2.3. During the interaction with the environment, the agent selects an action following the policy in the current state. In the context of IRL, an external trainer gives advice to the agent that modifies the action selected, let $J_t \in \mathbb{J}(x_t)$ be the *advice* provides by an external trainer at time t, with $\mathbb{J}(x_t)$ the set of all possibles instructions that allow to reach a goal. In some iteration steps, the trainer may not provide feedback. Thus, the likelihood of receiving feedback [40] has probability $0 < L < 1$.

Suppose that any action u_t is selected from a policy π given that an external trainer provides advice J_t in the state x_t. Let $\pi(u|x_t, J_t)$ be the probability density function of the actions after taking into account the advice J_t and the state x_t. In this sense, the policy is modified just after the trainer gives advice; nevertheless, the agent has only access to its policy $\pi(u|x_t)$, and the actions are chosen from this. If it assumes that J_t and x_t are random variables with some distribution of probability, from properties of conditional probability, the policy $\pi(u|x_t, J_t)$ can be expressed by:

Figure 5 – Impact of the receiving advice $\frac{P_J(J|u,x)}{P_J(J|x)}$ on the agent policy $\pi(u|x)$.



$$\begin{aligned}\pi(u|x,J) &= \frac{P(u,x,J)}{P(x,J)} = \frac{P_J(J|u,x)P(u,x)}{P(x,J)} \\ &= \frac{P_J(J|u,x)P(u|x)P_x(x)}{P_J(J|x)P_x(x)} \\ &= \frac{P_J(J|u,x)}{P_J(J|x)}\pi(u|x). \end{aligned} \tag{3.1}$$

This equation shows a clear relation between the policy $\pi(u|x,J)$ and the agent policy. The probability distribution $P_J(J|u,x)$ would express the evidence to give advice when the agent chooses the action u in the state x. Besides, the probability distribution $P_J(J|x)$ would express the evidence to give advice when the agent reaches the state x. The agent policy $\pi(u|x)$ would be the probability of selecting an action u before obtaining advice J, in the context of Bayesian inference it represents the prior distribution of the action u in the state x before the advice J is observed [55].

The factor $\frac{P_J(J|u,x)}{P_J(J|x)}$ in (3.1) can be interpreted as the impact of receiving advice on the selection of the action u (or on the probability of select the action). If $\frac{P_J(J|u,x)}{P_J(J|x)} \leq 1$, $(\pi(u|x,J) \leq \pi(u|x))$ the advice is irrelevant (with equality) or decrease the probability of select one action. If $\frac{P_J(J|u,x)}{P_J(J|x)} > 1$, $(\pi(u|x,J) \geq \pi(u|x))$ the advice improve the policy and increase the probability of select one action. Particularly, $P_J(J|u,x)$ favors a subset the actions during the process of selecting actions. Fig. 5 show an example of the influence of $\frac{P_J(J|u,x)}{P_J(J|x)}$ on the policy $\pi(u|x)$. In this graph, the agent policy gives a options to choose actions in one region with greater probability, however, the factor $\frac{P_J(J|u,x)}{P_J(J|x)}$ grant a privilege to the actions in other region of space. Thus the actions with higher probability are placed in another region that favors the

actions chosen for the policy as much as actions selected by the feedback.

The expression (3.1) involves two sources of variation as in [40], however, it generalizes their approach to implement advice from different spaces, it also provides a measure of information to assess the influence of the advice on the selection of actions.

3.1.1. Actor-Critic including advice information

Consider a stochastic policy $\pi_\upsilon(u|x_t, J_t)$ that is parametrized by a column vector of weights $\upsilon \in \mathbb{R}^{N_a}$. In the expression (3.1), it is clear that $P_J(J|u,x)$ and $P_J(J|x)$ are not known, and it is not possible to obtain a large sample of J_t during each step. We can consider any probability density function $\pi^*_J(u|x)$ to approximate $P_J(J|u,x)$. Thus, the advice policy $\pi_\upsilon(u|x,J)$ can be represented by:

$$\pi_\upsilon(u|x,J) \propto \pi^*_J(u|x) \times \pi_\upsilon(u|x), \tag{3.2}$$

where $\pi^*_J(u|x)$ has the characteristic to grant a privilege to actions in some region of space determined by the advice J. The parameter vector is allowed to be only part of the agent policy; after all, the objective is to find the optimal policy that maximizes the future reward. However, the probability function may have parameters that improve the selection of actions.

Algorithm 3 Episodic Actor-Critic algorithm with advice and Policy Gradient.

Inputs: γ, α_c, α_a, L

Initialize θ_t and υ_t arbitrarily
for each episode **do**
  initialize x_t
  repeat
    $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t, J_t)$ with $J_t = \emptyset$ (Non advice)
    if $rand(0,1) < L$ **then**
      Get advice J_t
      Change actions $u_t \leftarrow \pi_{\upsilon_t}(u|x_t, J_t)$
    end if
    Taken action u_t, observe reward r_{t+1} and next state x_{t+1}
    $\delta_t \leftarrow r_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
    $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t, J_t)\right)$
    $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
    $x_t \leftarrow x_{t+1}$
  until x_t is terminal
end for

In the context of AC with the policy gradients, the agent policy is modified to include the advice. Now the parameter is updated by a gradient $\nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t, J_t)\right)$ where J_t is the advice which comes from an external trainer at time t. If the advice is missing, it is correct to use the agent policy because it is not relevant information to improves the probability ($\pi(u|x,J) = \pi(u|x)$).

The full implementation of the actor-critic with interaction is shown in Algorithm 3. In this algorithm we use the *advice* method parameter for interaction [40], i. e., the probability of receiving advice L.

3.2. Policy Gradient Methods for Robust Reinforcement Learning

In work presented by Morimoto and Doya [14], they implement an AC algorithm with a disturbance input, the so-called Actor-Disturber-Critic (ADC). In their methodology, an approximation of functions is used to represent the value function, the agent policy, and the function that generates the disturbance, the disturber. As mentioned in section 2.7, an increased reward is used to withstand the maximum noise generated by the model.

We include policy gradients in the ADC of this approach; the main idea is to consider an objective function as in (2.9) for agent policy and the disturber. We consider that the disturber is a probability density function $\kappa_\omega(x_t)$ parameterized by the weight vector $\omega \in \mathbb{R}^{N_d}$. The parameter ω is adjusted in the direction of the gradient $\nabla_\omega \Gamma(\kappa)$ to generating the highest possible disturbance:

$$\omega_{t+1} - \omega_t \approx \alpha_\omega \nabla_\omega \Gamma(\kappa)$$

where α_ω is a learning rate of the disturber. Algorithm 4 presents an episodic ADC with policy gradients for the actor and the disturber.

Algorithm 4 Episodic Actor-Disturber-Critic algorithm with Policy Gradients.

Inputs: γ, α_c, α_a, α_a

Initialize θ_t and υ_t arbitrarily
for each episode **do**
  initialize x_t
  repeat
    $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t)$
    $w_t \leftarrow$ given by $\kappa_{\omega_t}(w_t|x_t)$
    Taken action u_t, observe reward r_{t+1} and next state x_{t+1}
    $\delta_t^w \leftarrow q_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
    $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t)\right)$
    $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
    $\omega_{t+1} \leftarrow \omega_t - \alpha_\omega \delta_t^w \nabla_{\omega_t} \log\left(\kappa_\omega(w_t|x_t)\right)$
    $x_t \leftarrow x_{t+1}$
  until x_t is terminal
end for

3.3. Interactive Robust Reinforcement Learning Approach

In order to include advice during learning when the agent interacts with a dynamic environment, we combine the IRL and RRL approaches to propose Interactive Robust Reinforcement Learning, an algorithm that involves advice for the agent to learn a task from an environment that has dynamic features. Algorithm 5 shows the episodic ADC with advice in the context of IRRL.

Algorithm 5 Episodic Actor-Disturber-Critic algorithm with advice.

Inputs: γ, α_c, α_a, α_a, L

Initialize θ_t and υ_t arbitrarily
for each episode **do**
  initialize x_t
  repeat
    $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t, J_t)$ with $J_t = \emptyset$ (Non advice)
    if $rand(0,1) < L$ **then**
      Get advice J_t
      Change actions $u_t \leftarrow \pi_{\upsilon_t}(u|x_t, J_t)$
    end if
    $w_t \leftarrow$ given by $\kappa_{\omega_t}(w_t|x_t)$
    Taken action u_t, observe reward r_{t+1} and next state x_{t+1}
    $\delta_t^w \leftarrow q_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
    $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t, J_t)\right)$
    $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
    $\omega_{t+1} \leftarrow \omega_t - \alpha_\omega \delta_t^w \nabla_{\omega_t} \log\left(\kappa_\omega(w_t|x_t)\right)$
    $x_t \leftarrow x_{t+1}$
  until x_t is terminal
end for

Chapter 4

Results and Discussion

This section presents the experimental setup used to carry out the experiments, the results obtained, and the discussion of the research.

4.1. Experimental Setup

4.1.1. Cart-pole Balancing task

To evaluate the performance of our methodology, we apply it to the classic *Cart-pole balancing task* [5]. The objective of this problem is to balance a pendulum by applying a force on the cart to which it is attached. Cart-pole balancing is a continuous problem, but in our implementations, we use it as an episode problem. Thus, the agent task ends when the pole falls, i.e., the pendulum cannot be balanced. Fig. 6 shows a schematic representation of the Cart-pole balancing problem. The cart is free to move within the limits of a one-way road. The pole is free to move on a pivot on the vertical axis of the cart and the track. The controller applies a force F to the right or left of the cart; this allows the cart to move and keep the pole balanced [3]. The force is bounded by the interval $(-F_{max}, F_{max})$, where F_{max} is a system parameter. The Cart-pole model has four output variables:

- $\chi \in [-2.4, 2.4]$ position of the cart in the track
- $\dot{\chi}$ cart velocity
- $\phi \in [-\pi/15, \pi/15]$ angle of the pole with the vertical axis
- $\dot{\phi}$ angular velocity (rate of change of the angle)

The model has other parameters such as the pole length and mass, cart mass, coefficients of friction between the cart and the track and at the hinge between the pole and the cart, the impulsive control force magnitude, the force due to gravity, and the simulation time step size. Table 1 shows the magnitude of each parameter [5].

Figure 6 – System of the Cart-pole balancing task, a cart attached to a track where it is free to move to balance a pivot attached to it. The applied force F produces a linear movement on the cart and an angular movement on the pole. Figure adapted from Nagendra et al. [3].

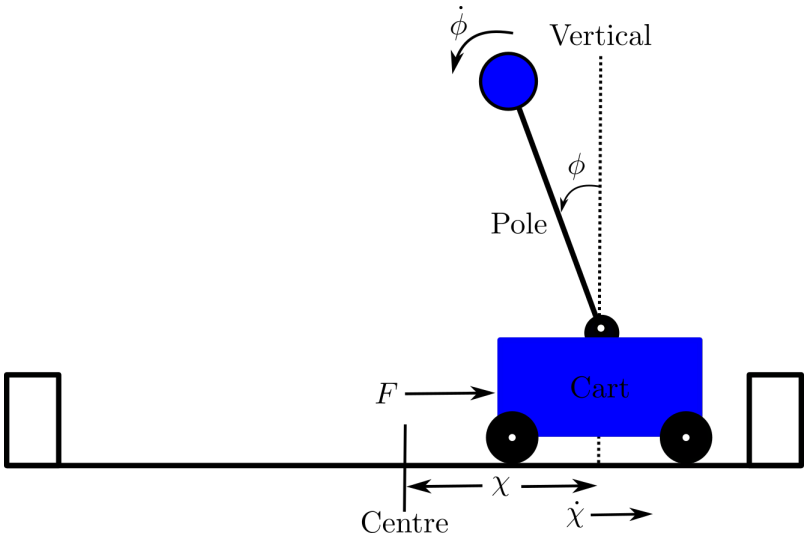


Table 1 – System parameters of the Cart-pole balancing task.

Parameter	value
g gravity	$-9.8\ m/s^2$
m_c mass of cart	1.0 Kg
m mass of pole	0.1 Kg
l half-pole length	0.5 m
μ_p friction of pole on cart	0.000002
μ_c friction of cart on track	0.0005
F_{max} force applied to cart center	10 N

In our implementation we use the *OpenAI Gym* toolkit [56], where the Cart-pole system is implemented based on the nonlinearities and reactive forces of the physical characteristics of the system. A modification was made in the Python source code to implement continuous actions and our own reward function. Details on this modification are presented in Annex A. On the other hand, we assume that the system model is unknown and that there is no agent that has previous knowledge of the task.

4.1.2. Reinforcement Learning Setting and Neural Architecture

In the context of RL, we define the states and actions under the Cart-pole balancing environment. We also define the policy, value function, and disturbance of the dynamic approach. We use a representation of a one dimensional action $u \longleftarrow F$ and four dimensional states as

$$x \longleftarrow \left\{ \frac{\chi}{2.4}, \frac{\dot{\chi}}{2}, 15\frac{\phi}{\pi}, \frac{\dot{\phi}}{1.5} \right\}.$$

This allows us to delimit the states in an interval $(-1, 1)$ in order that the magnitude of the variable is not an influence on learning. We define the reward function in our implementation as

$$\rho(x, u) = \rho_0 - 0.2 * |F_u - u|, \tag{4.1}$$

where $F_u = \min\{\max\{-F_{max}, u\}, F_{max}\}$, and

$$\rho_0 = \begin{cases} \cos(\frac{15}{2}\phi) & \text{if} \quad |\chi| < 2.4 \wedge |\phi| < \frac{\pi}{15} \\ -10 & \text{if} \quad |\chi| \geq 2.4 \\ -30 & \text{if} \quad |\phi| \geq \frac{\pi}{15} \end{cases}.$$

As aforementioned, the actions are obtained from a stochastic policy based on a probability distribution function. We define politics as a Gaussian distribution function [57] of mean $\mu_\upsilon(x)$ and standard deviation σ_x. The value $\mu_\upsilon(x)$ is interpreted as *the action with the highest probability of selection*. The value σ_x gives a *explore range* to search for actions. The disturber is defined in the same way as policy; however, in our experiments, we use a disturbance with two degrees of freedom. Thus, the disturbance is generated by a two-dimensional Gaussian distribution function with mean vector $\mu_\omega(x)$ and covariance matrix Σ_w. In both cases, policy and disturber, the variances are fixed during learning but with the ability to give the necessary exploration [57].

To use policy gradients and function approximations, we implemented a multilayer perceptron (MLP) which is a feedforward network with a hidden layer for each of the elements. The approximation is carried as follows:

- For the value function $V(x)$, an MLP with an input layer of 4 units, a hidden layer of 50 units, and an output layer with one unit.
- For the mean $\mu_\upsilon(x)$ of the policy, an MLP with an input layer of 4 units, a hidden layer of 20 units, and an output layer with one unit.
- For the mean $\mu_\omega(x)$ of the disturbance, an MLP with an input layer of 4 units, a hidden layer of 20 units, and an output layer with two units.

In the three architectures, we apply as activation function the hyperbolic tangent (*tanh*) in the hidden layer, and in the output layer a linear activation. Learning rates are empirically set at $\alpha_\theta = 0.001$, $\alpha_\upsilon = \alpha_\omega = 0.0001$, and the discount factor γ is set at 0.9. Each set-up was carried out 15 times using the average steps and average collected reward for the analysis. During the learning, we decided that the episode ends in three situations: If the pole falls ($|\phi| \geq 2.4$), if the cart collides with the ends of the track ($|\chi| \geq \frac{\pi}{15}$), or if the pole is swinged up for 400 iterations. The values of the weights in the neural networks are randomly initialized from a normal probability distribution with mean 0 and standard deviation 1.

4.2. Experimental Results

4.2.1. Training an agent using classic RL

First, we perform the training of agents with the AC algorithm (see algorithm 2) using the reward function presented in equation (4.1). We test diverse values of σ_x to investigate the influence on the learning process in terms of performed actions and collected reward. Due to the domains are continuous sets, the agent needs even more time to explore space and find enough patterns to learn the task. Then, more than 800 training episodes are required to keep the pole balanced.

Figure 7 – Average steps over 15 runs using classic RL in 1500 episodes with different standard deviation σ_x. The shaded area shows the confidence bands at 95%. In each episode, the agent can perform a maximum of 400 steps (actions). The agent has a better performance with $\sigma_x = 2$ due to it has more actions available for selecting.

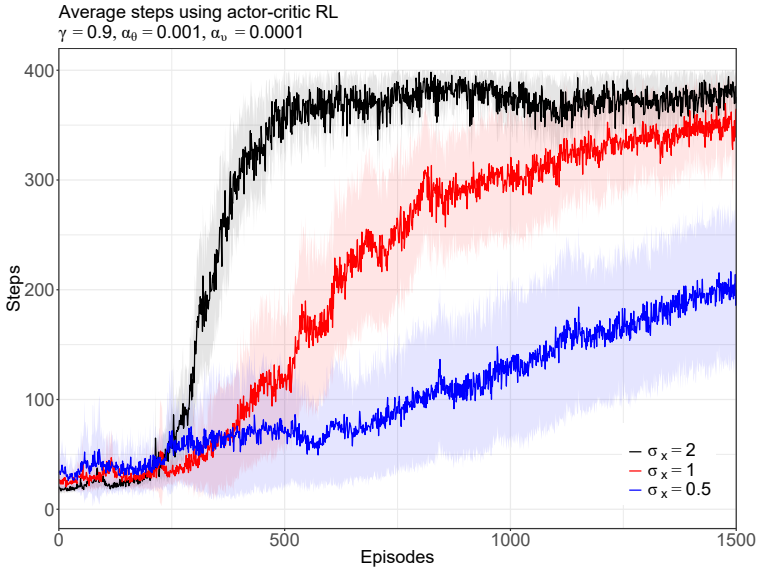
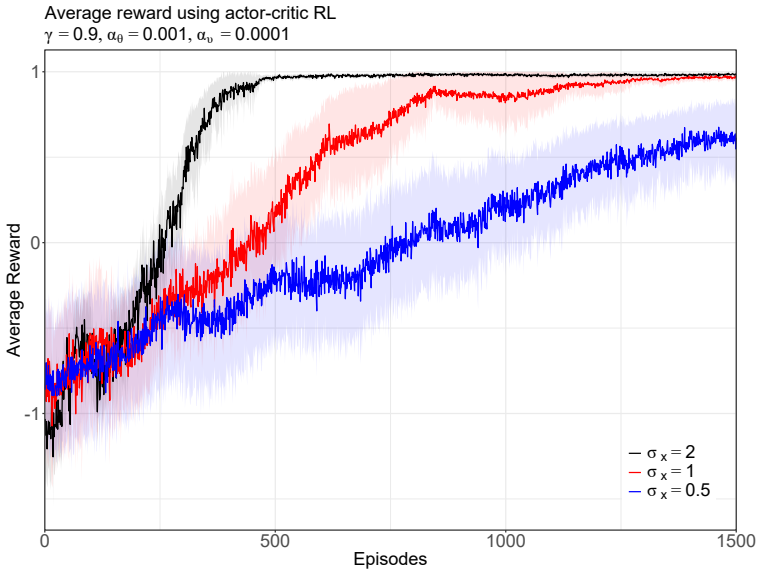


Fig. 7 shows the average steps performed with $\sigma_x \in \{0.5, 1, 2\}$. We can observe that with a value of standard deviation $\sigma_x = 2$ the agent has a better performance in the first episodes, this is due to it has more actions available for selecting around to a value with high probability $\mu_\upsilon(x)$. After 500 episodes the agent is able to keep the pole balanced by more than 350 steps. Even so, an agent with standard deviation $\sigma_x = 1$ can improve his performance before reaching 1500 episodes. The shaded area shows the confidence bands at 95%. It is noted that the band is wider for low values of σ_x, indicating that the performance of the agent is different in each run.

Fig. 8 shows the average rewards collected by the agent over 1500 episodes for different values of σ_x. It can be seen that the curves start with values around -1 which means that at the beginning the agent is not able to keep the pole balanced or the cart collides with the limit of the track, that behavior is maintained for 500 episodes. After 500 episodes the agent can keep the pole balanced by more steps and increase the collected reward up to 1.

Figure 8 – Average collected reward over 15 runs using classic RL in 1500 episodes with different standard deviation σ_x. The shaded area shows the confidence bands at 95%. In each episode, the agent can collect at most 1 average reward.



4.2.2. Training an agent using IRL

To approach a real scenario, an external trainer observes the pole attached to the cart and, based on its criteria, will give advice. In this sense, the trainer will say "*push to the right"* if the pendulum is falling to the right, and say *"push to the left"* if the pole falls to the left. Thus, acceptable advice is a dichotomous variable that indicates the direction where the cart has to be pushed. In our implementation, we create an *oracle*, a function $J_O(x, u)$ defined by:

$$J_O(x,u) = \begin{cases} -1 & \text{if} \quad \textit{push to the left} \\ 1 & \text{if} \quad \textit{push to the right} \end{cases} . \tag{4.2}$$

We calculated the necessary force F_R to remain the pole balanced during the next step, then we make the difference between F_R and the action selected by the agent u. The return is the sign of this difference.

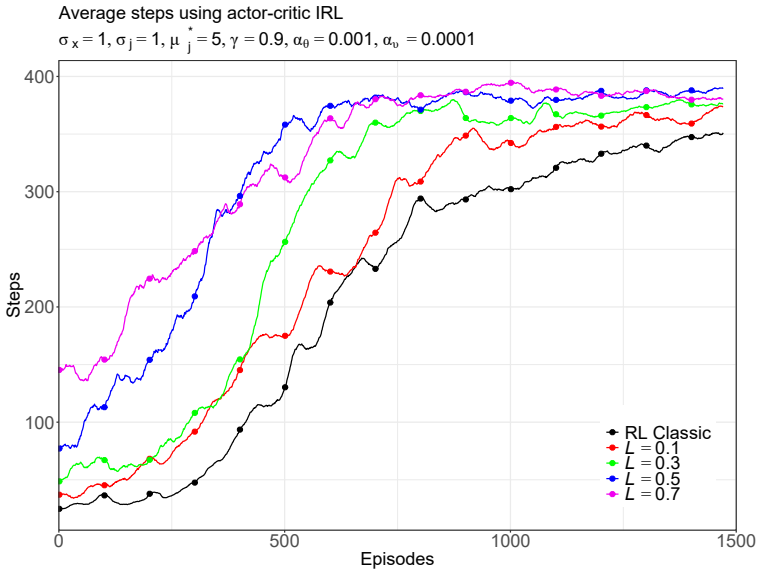
Once the advice is obtained, a subset of actions is privileged according to the equation (3.2). We define the function $\pi_J^*(u|x)$ as a Gaussian distribution with mean $\mu_J(x) = \mu_\upsilon(x) + J\mu_j^*$ and standard deviation σ_J where $J \longleftarrow J_O(x, u)$. This function favors actions that are μ_j^* units more (or less) than the most likely action given by the policy $\mu_\upsilon(x)$. In this way, the *advice policy* takes the form of a Gaussian distribution defined as:

$$\pi_\vartheta(u|x, J) = \frac{1}{\sqrt{2\pi\sigma_{XJ}^2}} \exp\left\{-\frac{(u-\mu_{XJ})^2}{2\sigma_{XJ}^2}\right\},$$

where $\mu_{XJ} = \frac{\sigma_J^2\mu_\vartheta(x)+\sigma_x^2\mu_J(x)}{\sigma_J^2+\sigma_x^2}$ and $\sigma_{XJ}^2 = \frac{\sigma_J^2\sigma_x^2}{\sigma_J^2+\sigma_x^2}$.

We test different values of L, σ_J and μ_J^* to investigate its influence during the learning process in terms of preformed steps and collected reward. We used a fixed standard deviation of

Figure 9 – Average steps over 15 runs using IRL with different probability of likelihood L, standard deviation $\sigma_J = 1$ and mean $\mu_J^* = 5$. The black line shows the number of steps with the classic RL that is equivalent to $L = 0$. With interaction probabilities as small as $L = 0.1$, the agent takes advantage to increase the number of steps to keep the pole balanced.



policy $\sigma_x = 1$ in these experiments. Besides, the average steps and the average collected reward were smoothed by a *moving average* with a empirical window size 30.

Figure 10 – Average collected reward over 15 runs using RL (black line) and IRL approach with different probability of likelihood L, standard deviation $\sigma_J = 1$ and mean $\mu_J^* = 5$. After 700 episodes all approaches reach a reward over 0.75.

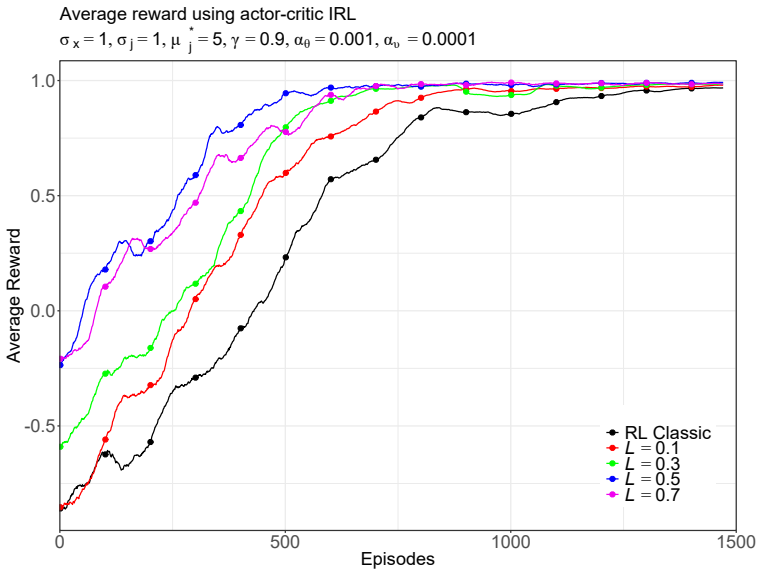
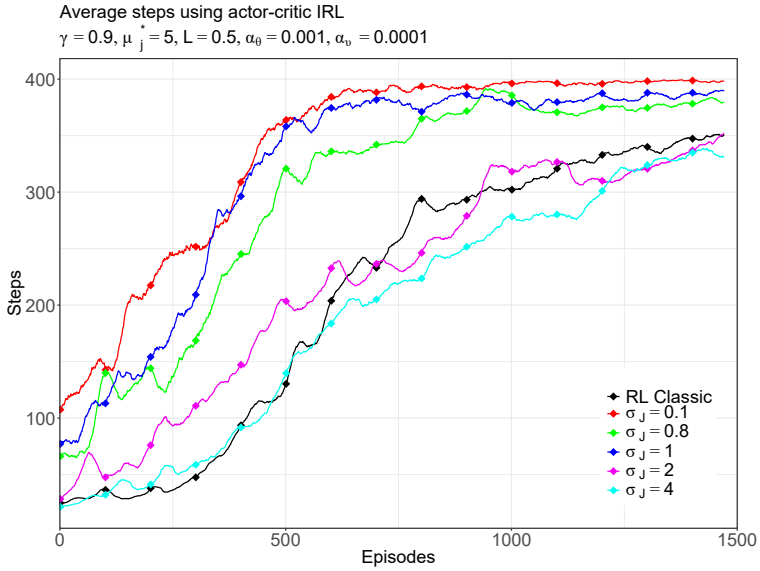


Fig. 9 shows the average steps performed with probability of likelihood $L \in \{0.1, 0.3, 0.5, 0.7\}$, standard deviation $\sigma_j = 1$, and mean $\mu_j^* = 5$. The black line shows the average steps with RL which is equivalent to $L = 0$. We can observe that with the even smaller probability the agent can

improve its performance, mainly in the first episodes. Additionally, Fig. 10 shows the average collected reward by the agent over the episodes for different values of L.

Afterward, we explore the learning process with different values of σ_J. For high values of σ_J, concerning σ_x, the advice does not have a high impact on the agent, i.e., it is irrelevant for learning. In another case, for low values of σ_J, the advice has a relevant effect on the agent decision.

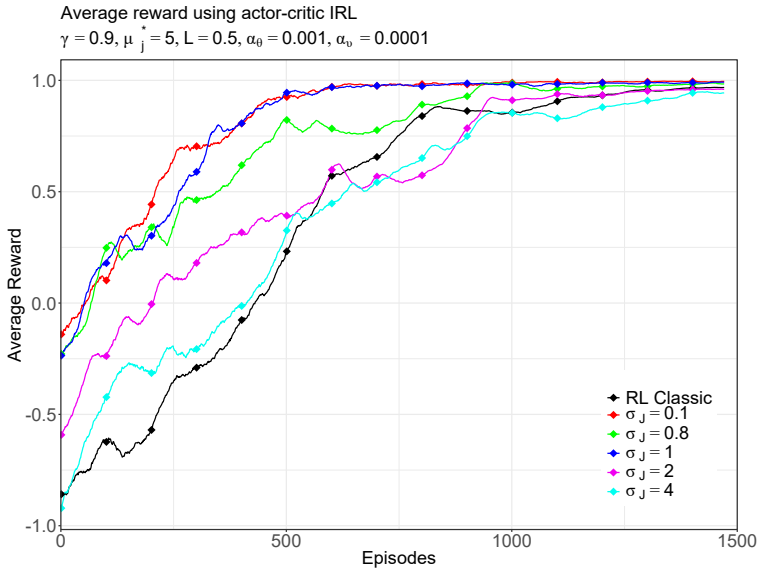
Figure 11 – Average steps over 15 runs using IRL with different values of standard deviation σ_J, mean $\mu_J^* = 5$ and probability of likelihood $L = 0.5$. The black line shows the average steps with RL which is equivalent to $\sigma_J \longrightarrow \infty$. The lower the values of σ_J, the agent takes advantage by augmenting the steps which it remains the pole balanced.



We set the probability of likelihood to $L = 0.5$ and the mean $\mu_J^* = 5$, and investigate the learning performance for different standard deviations $\sigma_J \in \{0.1, 0.8, 1, 2, 4\}$. Fig. 11 shows the average steps performed by the agent in this experiment. The black line represents an autonomous RL agent that is equivalent to $L = 0$ and $\sigma_J \longrightarrow \infty$. The smaller the value of σ_J, the agent achieves a better performance in its task keeping the pole balanced by more than 350 steps. We observe that with a value of $\sigma_J = 1$ a better performance is obtained than with a value of $\sigma_J = 0.8$, however, the performance is even greater with smaller values, such as $\sigma_J = 0.1$. Higher values suggest that the IRL agent performs similarly to an RL agent in terms of steps. Concerning the average collected reward (see Fig. 12), the agent achieves rewards of 1 with standard deviations less than $\sigma_J = 1$.

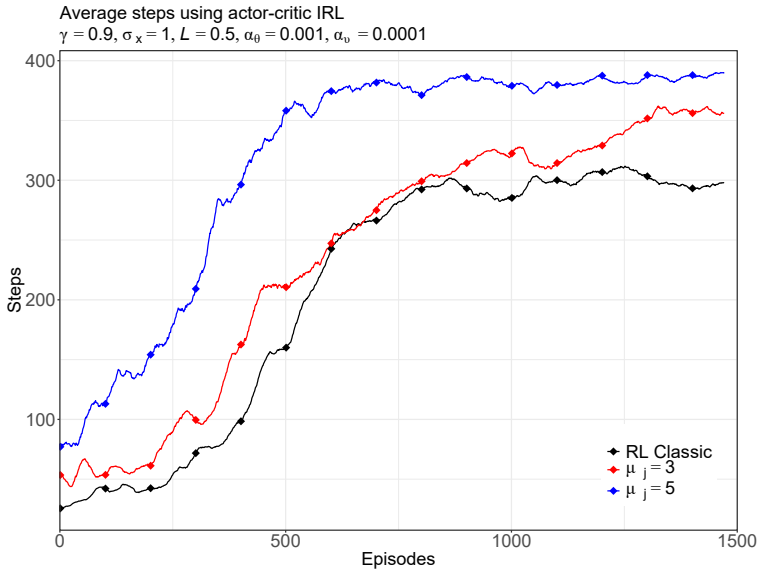
Finally, we investigate the learning behavior with different values of μ_J^*. This unit should not be considered as a model parameter since it is only part of a strategy to unify the space of two variables, the advice (discrete) and the actions (continuous). However, it explains how advice induces a tendency towards the privileged region, whether it is right or wrong. For example, suppose that $\sigma_J = \sigma_x$ and $\mu_J(x) = \mu_v(x) + J\mu_j^*$, from the advice policy, we can say that

Figure 12 – Average collected reward over 15 runs using classic RL (black line) and IRL approach with different standard deviation σ_J, mean $\mu_J^* = 5$ and probability of likelihood $L = 0.5$.



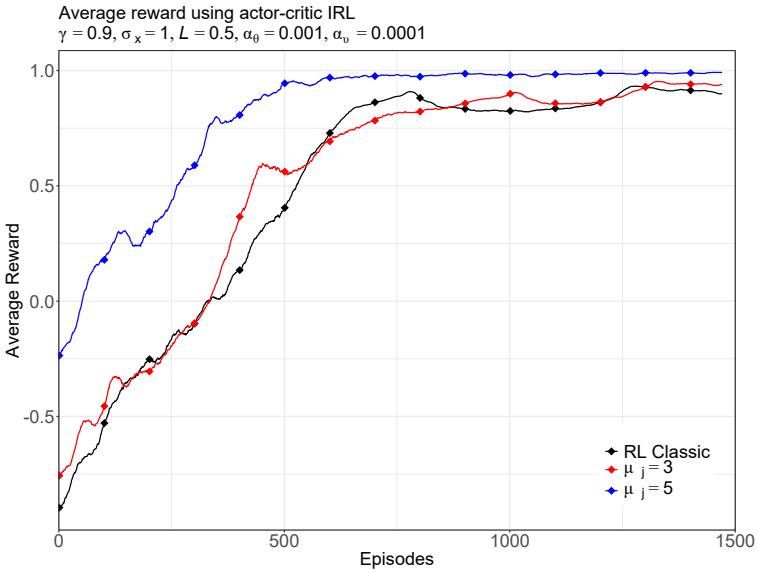
$\mu_{JX} = \mu_\upsilon(x) + J\mu_j^*/2$. Then, if the value of μ_j^* is close to zero, the advice will not be influential in making the decision, in fact, it could be said that the trainer acts in the same way as the agent. Instead, if the value is quite far from zero, the advice will be influential in making the decision and drives the agent to select actions in another region of the space.

Figure 13 – Average steps over 15 runs using IRL with different values of μ_J^*, probability of likelihood $L = 0.5$ and $\sigma_J = 1$. The black line shows the average steps with RL which is equivalent to $\mu_J^* = 0$. The agent has a better performance with $\mu_J^* = 5$, after 500 times it keeps the pole balanced for more than 350 steps.



We investigate the learning performance for different values of mean $\mu_J^* \in \{3, 5\}$, we set the probability of likelihood $L = 0.5$ and the standard deviation $\sigma_J = 1$. Fig. 13 shows the average steps taken by the agent to complete the task; the black line indicates the steps of

Figure 14 – Average collected reward over 15 using classic RL (black line) and IRL approach with different values of μ_J^*, probability of likelihood $L = 0.5$ and $\sigma_J = 1$. After 500 episodes, the agent collects more than 0.75 of reward with, however, with $\mu_J^* = 5$, it is reached to 1 of awards.



a classic RL implementation. We observed that the performance of the agent is higher with a value of $\mu_J^* = 5$. Also, the learning time is reduced, achieving more than 350 steps after the 500 episodes. With values of μ_J^* close to zero, the agent performance is similar to that of a classic RL agent, however, in the last episodes, we perceive that more than 350 steps are reached. In terms of reward, Fig. 14 shows that the agent receives a reward greater than 0.75; however, with $\mu_J^* = 5$ the maximum reward value is reached.

4.2.3. Training an agent using RRL

In this part of the experiments, we performed the training of agents with the ADC algorithm (see algorithm 4) using the reward function in the equation (4.1). In order to resist the disturbance, we consider the additional reward $\omega(w_t)$ in (2.14) of the form

$$\omega(w_t) \longleftarrow \eta^2 w_t^\dagger w_t$$

where $\dagger$ is the transpose of a vector and η is a parameter of robustness [14]. Empirically we set the robustness value $\eta = 0.45$, exploration standard deviation $\sigma_x = 1$, and the covariance matrix $\Sigma = I_2$, where I_2 is the identity matrix of order 2.

We investigate how the learning process is using RRL in a fixed environment. Fig. 15 represents the average steps taken by the agent to complete the task. The black line shows an autonomous RL agent that is equivalent to $\eta = 0$. We observe that the performance of an agent using RRL is lower than that of an agent trained with RL. Morimoto and Doya [14] discuss on this behavior in their work. RL agent has better performance (is faster) than an RRL agent; this is because the disturbance is explicitly considered during learning. The curves begin performing

Figure 15 – Average steps over 15 run using RRL in a fixed environment during the learning with $\eta = 0.45$. The black line represents the average steps using RL, the performance of the RRL do not overcome to RL agent.

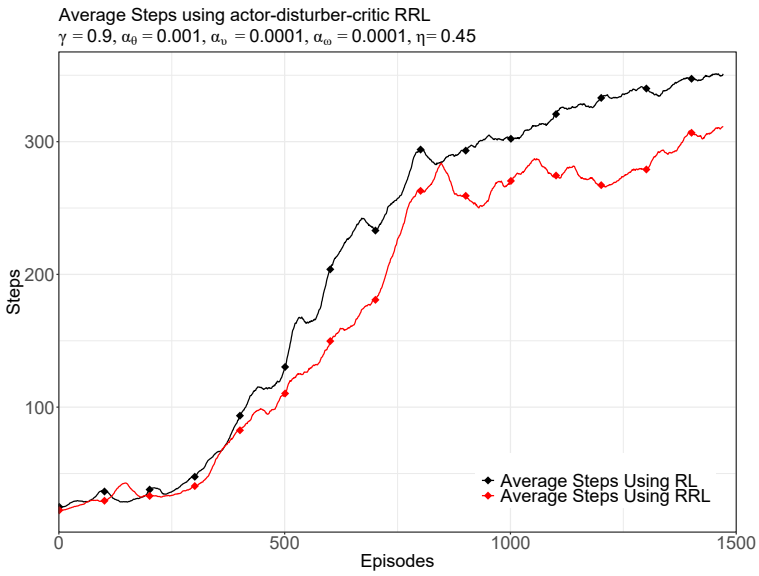
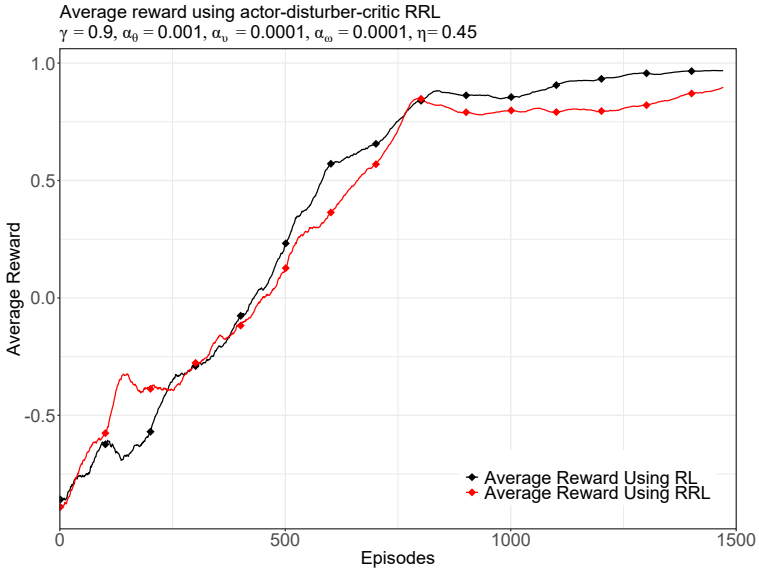


Figure 16 – Average collected reward over 15 runs using RL (black line) and RRL approach in a fixed environment with $\eta = 0.45$. Here, the augmented reward is not taken into account. The collected reward is higher than 0.75 after 700 episodes for the two methodologies.



the task in less than 50 steps; after episode 500, the RL agent begins to perform better. Fig. 16 shows the average reward collected for this experiment, the reward illustrated is that obtained directly from the environment.

Our goal with this methodology is to train more robust agents that resist an external disturbance to the environment. Thus, we perform a test where a previously trained agent confronts changes in the environment. Here, we compare the success rate of a robust agent against an RL agent. The success rate is calculated with the number of steps in 100 episodes. If

the pendulum is held for 400 steps, the episode ends. We use 15 runs to compare the average of the rate.

Figure 17 – Average success rate over 15 learned agents using RRL in a fixed environment during the learning. The black line represent the success rate using RL. The success rate using a robust agent with the friction coefficient $\mu_c = 1$ was about 45% while using a classic RL agent it was about 42%.

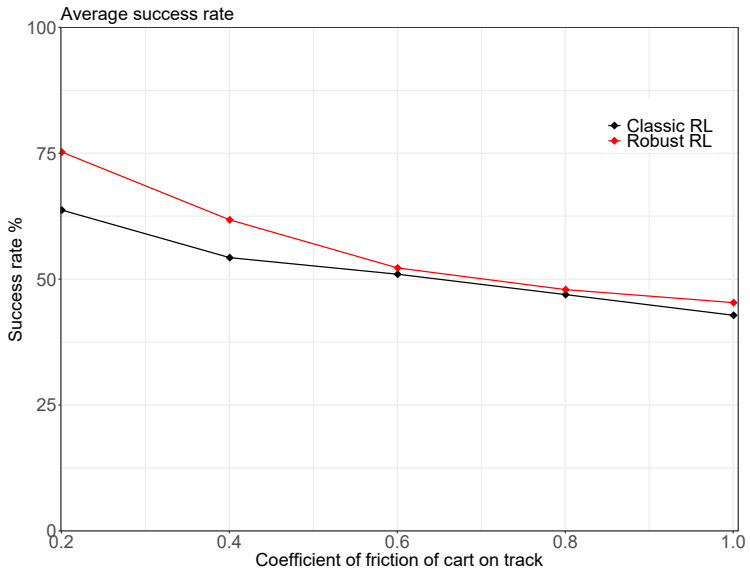


Fig. 17 shows the success rate of the Cart-pole balancing task. We take the friction of the cart on track in $\mu_c \in [0.0005, 1]$. Each trial was initiated from an angle $\phi(0)$ selected from a uniform distribution with limits $(-0.05, 0.05)$. The success rate using a robust agent with the friction coefficient $\mu_c = 1$ was about 45% while using a classic RL agent it was about 42%. We note that, for friction coefficient μ_c close to the training value ($\mu_c = 0.0005$), the success rate of a robust agent is higher than the rate of a classic RL agent, however, when the friction approaches $\mu_c = 1$, the success rate tends to be the same in both experiments.

Following, we explore the learning process in a dynamic environment where we modify the friction of the cart on the track at each step. The friction μ_c is generated from a uniform distribution with limits $(0.0005, 0.002)$, the number of episodes was set at 3000. Fig. 18 shows the average steps taken by the agent to complete the task. Like the fixed environment, the RRL agent tends to take fewer steps during learning compared to the RL agent. Fig. 19 presents the average reward collected by the agents during the learning. After 1500 episodes, the reward collected is close to 1; this shows the agent difficulty in finding optimal actions during learning. The selected actions can follow patterns that are not present in the environment, and when they are applied, they do not have the same effect as they would have in previous steps or with different friction values.

Figure 18 – Average steps over 15 run using RRL in a dynamic environment during the learning with $\eta = 0.45$. The black line represents the average steps using RL, the performance of the RRL do not overcome to RL agent.

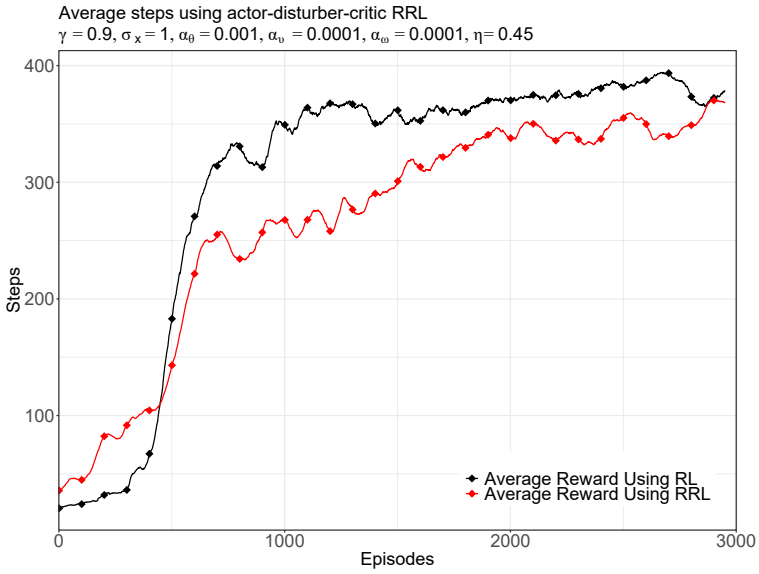
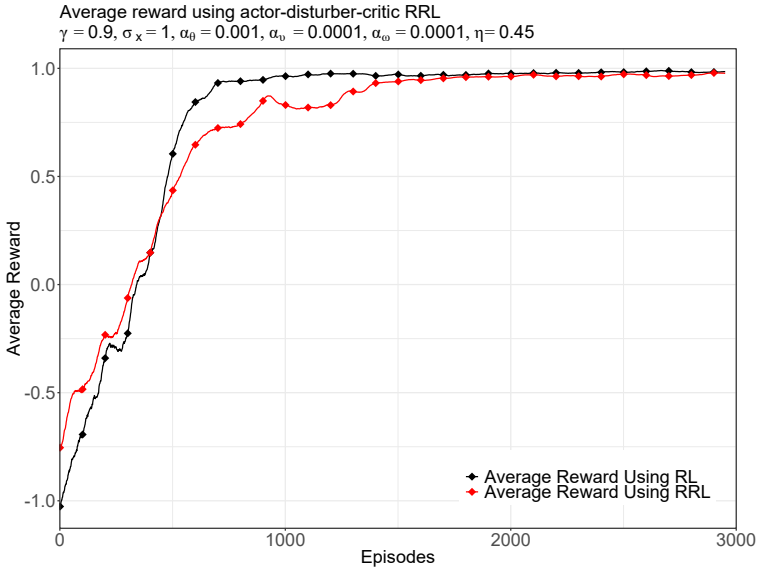


Figure 19 – Average collected reward over 15 runs using RL (black line) and RRL approach in a dynamic environment with $\eta = 0.45$. Here, the augmented reward is not taken into account. The collected reward is 1 after 1500 episodes for the two methodologies.



4.2.4. Training an agent using IRRL

Finally, we train agents using the ADC algorithm with advice (see algorithm 5). The reward function in the equation (4.1) will be used. We carry out these experiments with 3000 episodes and fixed exploration standard deviation $\sigma_x = 1$. In this part we only investigate the learning behavior for different values of the probability of likelihood $L \in \{0.1, 0.3, 0.5, 0.7\}$. We set the value of the standard deviation $\sigma_J = 1$, mean $\mu_J^* = 5$, covariance matrix $\Sigma_2 = I_2$ and robustness parameter $\eta = 0.45$.

Figure 20 – Average steps over 15 runs using IRRL with different probability of likelihood L with dynamic environment. The black line shows the average steps using RRL. With interaction probabilities as small as $L = 0.1$, the agent takes advantage to increase the number of steps to keep the pole balanced.

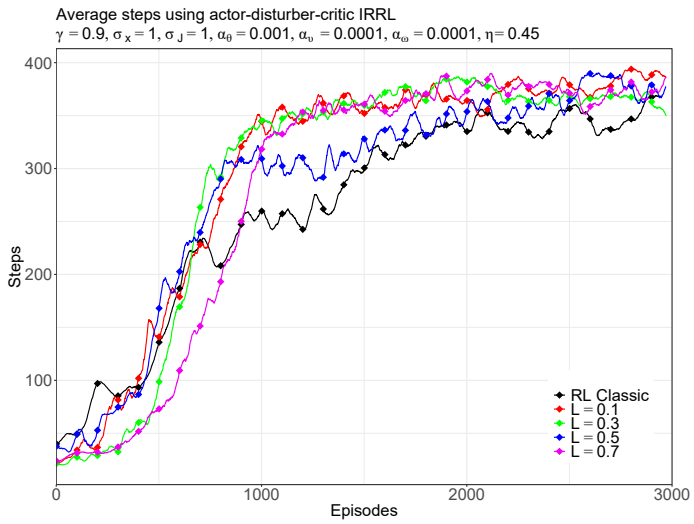


Figure 21 – Average collected reward over 15 runs using RRL (black line) and IRRL approach with different probability of likelihood L. After 1500 episodes, the collected reward is close to 1 for all curves.

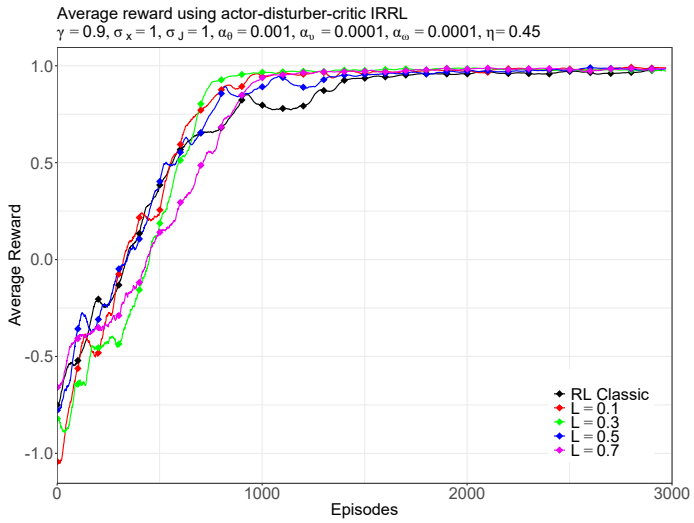


Fig. 20 represents the average steps taken by the agent to keep the pole balanced. We observe a better performance of the agents which receive advice compared to the RRL agent. In the first episodes, agents which receive a lot of advice may take longer episodes to improve their performance, however, after 1000 episodes the average number of steps is higher than 300 and continues to increase. With a probability of interaction $L = 0.7$, learning begins with a low performance, however, after 1000 episodes its performance improves at the same time than other probability of likelihood values L. Fig. 21 shows the average reward collected during learning. It should be noted that after 1500 episodes the agents get rewards close to 1.

Chapter 5

Conclusion

This thesis presents a methodology to implement IRL in scenarios where states and actions are continuous in dynamic environments. The proposal is applied in the Cart-pole balancing task and we evaluate the ability of agent to learn the task by comparing the number of steps and the reward collected by episodes. We individually evaluate each of the algorithms to have a comparative basis and be able to combine them at the end. In addition, different parameter settings are implemented to investigate the effect on learning.

In terms of IRL, we obtained a better performance of the agent by giving advice, compared to the classic RL. With our methodology, the agent is able to learn the task in fewer episodes, and the number of steps per episode reaches approximately the maximum (400 steps). In terms of RRL, the agent requires a greater number of episodes to learn the task, however, the agent is more robust in dynamic environments. Finally, with IRRL in terms of average steps, advice influences learning as well as IRL. In terms of reward, we note that a cumulative reward of 1 is achieved for all probability of likelihood L; however, values such as $L = 0.7$ have greater difficulty in the first episodes of learning.

5.1. Future Works

As future work, we propose:

- To consider previous environmental information to accelerate learning further. To use a methodology based on traces of eligibility or experience replay.
- To implement deep networks in our methodology, in particular, to use as function approximation a *Convolutional Neural Network* architecture based on the concepts of *Deep Reinforcement Learning*.
- To apply the proposed methodology in a robotic environment, where the action space is multidimensional.

5.2. Published Contributions

Title: Human feedback in continuous actor-critic reinforcement learning.

Autors: Cristian Millán, Bruno Fernandes, Francisco Cruz

Abstract: Reinforcement learning is utilized in contexts where an agent tries to learn from the environment. Using continuous actions, the performance may be improved in comparison to using discrete actions, however, this leads to excessive time to find a proper policy. In this work, we focus on including human feedback in reinforcement learning for a continuous action space. We unify the policy and the feedback to favor actions of low probability density. Furthermore, we compare the performance of the feedback for the continuous actor-critic algorithm and test our experiments in the cart-pole balancing task. The obtained results show that the proposed approach increases the accumulated reward in comparison to the autonomous learning method.

Bibliography

[1] SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: [s.n.], 1998. ISBN 0-262-19398-1.

[2] CRUZ, F. *Teaching Robots With Interactive Reinforcement Learning*. 169 p. Tese (Doutorado) — University of Hamburg, 2017.

[3] NAGENDRA, S.; PODILA, N.; UGARAKHOD, R.; GEORGE, K. Comparison of Reinforcement Learning algorithms applied to the Cart Pole problem. oct 2018. Disponível em: <http://arxiv.org/abs/1810.01940http://dx.doi.org/10.1109/ICACCI.2017.8125811>.

[4] WATKINS, C. J. C. H. *Learning from delayed rewards*. 234 p. Tese (Doutorado), 1989.

[5] BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, n. 5, p. 834–846, sep 1983. ISSN 0018-9472. Disponível em: <http://ieeexplore.ieee.org/document/6313077/>.

[6] KONDA, V. R.; TSITSIKLIS, J. N. Actor-Critic Algorithms. In: *Advances in neural information processing systems*. [s.n.], 2000. p. 1008–1014. Disponível em: <https://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf>.

[7] THOMAZ, A. L.; BREAZEAL, C. Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. In: *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*. Boston, Massachusetts, USA: AAAI, 2006. p. 1000–1005. ISBN 1577352815. Disponível em: <https://www.aaai.org/Papers/AAAI/2006/AAAI06-157.pdf>.

[8] CRUZ, F.; MAGG, S.; WEBER, C.; WERMTER, S. Training Agents With Interactive Reinforcement Learning and Contextual Affordances. *IEEE Transactions on Cognitive and Developmental Systems*, v. 8, n. 4, p. 271–284, 2016. ISSN 2379-8920. Disponível em: <http://ieeexplore.ieee.org/document/7458195/>.

[9] THOMAZ, A. L.; HOFFMAN, G.; BREAZEAL, C. Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots. In: *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*. Hatfield, UK: IEEE, 2006. p. 352–357. ISBN 1-4244-0564-5. Disponível em: <https://ieeexplore.ieee.org/document/4107833/>.

[10] BAIRD, L. C.; KLOPF, A. H. Reinforcement learning with high-dimensional, continuous actions. *US Air Force Technical Report WL-TR-93-1147*, v. 7644, n. 513, 1993.

[11] HASSELT, H. van; WIERING, M. A. Reinforcement Learning in Continuous Action Spaces. In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. Honolulu, HI, USA: IEEE, 2007. p. 272–279. ISBN 1-4244-0706-0.

[12] DOYA, K. Reinforcement Learning in Continuous Time and Space. *Neural Computation*, v. 12, n. 1, p. 219–245, jan 2000. ISSN 0899-7667. Disponível em: <http://www.mitpressjournals.org/doi/10.1162/089976600300015961>.

[13] BUGMANN, G. Normalized Gaussian Radial Basis Function networks. *Neurocomputing*, v. 20, n. 1-3, p. 97–110, aug 1998. ISSN 09252312.

[14] MORIMOTO, J.; DOYA, K. Robust Reinforcement Learning. *Neural Computation*, v. 17, n. 2, p. 335–359, feb 2005. ISSN 0899-7667. Disponível em: <http://www.mitpressjournals.org/doi/10.1162/0899766053011528>.

[15] LACHMAN, S. J. Learning is a process: Toward an improved definition of learning. *Journal of Psychology: Interdisciplinary and Applied*, v. 131, n. 5, p. 477–480, 1997. ISSN 19401019.

[16] GROSS, R. *Psychology: The Science of Mind and Behaviour*. 5. ed. [S.l.]: Hodder Education Publishers, 2005. ISBN 9780340900987.

[17] MENDEL, J. M.; MCLAREN, R. W. Reinforcement-learning control and pattern recognition systems. *Mathematics in Science and Engineering*, Academic Press, Inc., v. 66, n. C, p. 287–318, 1970. ISSN 00765392. Disponível em: <http://dx.doi.org/10.1016/S0076-5392(08)60497-X>.

[18] RIESER, V.; LEMON, O. *Reinforcement Learning for Adaptive Dialogue Systems: A Data-driven Methodology for Dialogue Management and Natural Language Generation*. Springer-Verlag Berlin Heidelberg, 2011. ISBN 3642249418,9783642249419. Disponível em: <https://www.springer.com/gp/book/9783642249419>.

[19] LIN, L. J. Programming Robots Using Reinforcement Learning and Teaching. In: *AAAI-91 THE NINTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. [s.n.], 1991. p. 781–786. ISBN 0-262-51059-6. Disponível em: <http://www.aaai.org/Library/AAAI/1991/aaai91-122.php>.

[20] KNOX, W. B.; SETAPEN, A.; STONE, P. Reinforcement Learning with Human Feedback in Mountain Car. *Artificial Intelligence*, p. 36–41, 2011. Disponível em: <https://www.aaai.org/ocs/index.php/SSS/SSS11/paper/view/2487/2888>.

[21] PUTERMAN, M. L. *Markov Decision Processes*. Hoboken, NJ, USA: Wiley-Interscience, 1994. ISBN 9780470316887.

[22] SIGAUD, O.; BUFFET, O. *Markov Decision Processes in Artificial Intelligence*. [S.l.]: Wiley-ISTE, 2010. ISBN 9781848211674.

[23] KULKARNI, P. *Reinforcement and Systemic Machine Learning for Decision Making*. [S.l.]: Wiley-IEEE Press, 2012. ISBN 9780470919996.

[24] LIM, M.-H.; ONG, Y.-S.; ZHANG, J.; SANDERSON, A. C.; SEIFFERTT, J.; WUNSCH, D. C. *Reinforcement Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 12. 978–3 p. (Adaptation, Learning, and Optimization, 12). ISBN 978-3-642-27644-6. Disponível em: <http://link.springer.com/10.1007/978-3-642-27645-3>.

[25] BELLMAN, R. E. *Dynamic programming*. [S.l.]: Dover Publications, 2003. ISBN 9780486428093.

[26] TESAURO, G. Temporal difference learning and TD-Gammon. *Communications of the ACM*, v. 38, n. 3, p. 58–68, mar 1995. ISSN 00010782. Disponível em: <http://portal.acm.org/citation.cfm?doid=203330.203343>.

[27] RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Prentice Hall, 2003. ISBN 0137903952,9780137903955,0130803022.

[28] WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *Machine Learning*, v. 8, n. 3-4, p. 279–292, 1992. ISSN 0885-6125. Disponível em: <http://link.springer.com/10.1007/BF00992698>.

[29] RUMMERY, A. G.; NIRANJAN, M. *On-Line Q-Learning Using Connectionist Systems*. [S.l.], 1994.

[30] GRONDMAN, I.; VAANDRAGER, M.; BUSONIU, L.; BABUŠKA, R.; SCHUITEMA, E. Efficient Model Learning Methods for Actor–Critic Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 42, n. 3, p. 591–602, jun 2012. ISSN 1083-4419. Disponível em: <http://ieeexplore.ieee.org/document/6096441/>.

[31] SUTTON, R. S.; MCALLESTER, D.; SINGH, S.; MANSOUR, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. Denver, CO: MIT Press Cambridge, 1999. p. 1057–1063.

[32] SILVER, D.; LEVER, G.; HEESS, N.; DEGRIS, T.; WIERSTRA, D.; RIEDMILLER, M. Deterministic Policy Gradient Algorithms. In: *International Conference on Machine Learning*. Beijing: ICML, 2014. Disponível em: <http://proceedings.mlr.press/v32/silver14.pdf>.

[33] DEGRIS, T.; PILARSKI, P. M.; SUTTON, R. S. Model-Free reinforcement learning with continuous action in practice. In: *2012 American Control Conference (ACC)*. Montreal, QC, Canada: IEEE, 2012. p. 2177–2182. ISBN 978-1-4577-1096-4. Disponível em: <http://ieeexplore.ieee.org/document/6315022/>.

[34] GRONDMAN, I. *Online Model Learning Algorithms for ActorCritic Control*. [S.l.: s.n.], 2015. ISBN 9789461864321.

[35] BAIRD, L. Reinforcement learning in continuous time: advantage updating. In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. IEEE, 1994. v. 4, n. 513, p. 2448–2453. ISBN 0-7803-1901-X. Disponível em: <http://ieeexplore.ieee.org/document/374604/>.

[36] BHATNAGAR, S.; SUTTON, R. S.; GHAVAMZADEH, M.; LEE, M. Natural actor-critic algorithms. *Automatica*, Elsevier Ltd, v. 45, n. 11, p. 2471–2482, 2009. ISSN 00051098. Disponível em: <http://dx.doi.org/10.1016/j.automatica.2009.07.008>.

[37] DIGIOVANNA, J.; MAHMOUDI, B.; FORTES, J.; PRINCIPE, J.; SANCHEZ, J. Coadaptive Brain–Machine Interface via Reinforcement Learning. *IEEE Transactions on Biomedical Engineering*, v. 56, n. 1, p. 54–64, jan 2009. ISSN 0018-9294. Disponível em: <http://ieeexplore.ieee.org/document/4540104/>.

[38] KNOX, W. B.; STONE, P.; STONE, P. Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. *The Fifth International Conference on Knowledge Capture*, 2009.

[39] SUAY, H. B.; CHERNOVA, S. Effect of human guidance and state space size on Interactive Reinforcement Learning. In: *RO-MAN 2011 - The 20th IEEE International Symposium on Robot and Human Interactive Communication*. Atlanta: IEEE, 2011. p. 1–6. ISBN 978-1-4577-1571-6. Disponível em: <http://ieeexplore.ieee.org/document/6005223/>.

[40] GRIFFITH, S.; SUBRAMANIAN, K.; SCHOLZ, J.; ISBELL, C. L.; THOMAZ, A. L. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, p. 2625–2633, 2013. ISSN 10495258. Disponível em: <https://papers.nips.cc/paper/5187-policy-shaping-integrating-human-feedback-with-reinforcement-learning>.

[41] PILARSKI, P.; SUTTON, R. Between Instruction and Reward: Human-Prompted Switching. In: *AAAI Fall Symposium: Robots Learning Interactively from Human Teachers*. [s.n.], 2012. p. 46–52. Disponível em: <http://www.aaai.org/ocs/index.php/FSS/FSS12/paper/viewPDFInterstitial/5496/5885>.

[42] THOMAZ, A. L.; BREAZEAL, C. Asymmetric Interpretations of Positive and Negative Human Feedback for a Social Learning Agent. In: *RO-MAN 2007 - The 16th IEEE International Symposium on Robot and Human Interactive Communication*. Jeju, South Korea: IEEE, 2007. p. 720–725. ISBN 978-1-4244-1634-9. Disponível em: <http://dx.doi.org/10.1109/ROMAN.2007.4415180VN-readcube.comhttp://ieeexplore.ieee.org/document/4415180/>.

[43] PURIEL-GIL, G.; YU, W.; SOSSA, H. Reinforcement Learning Compensation based PD Control for Inverted Pendulum. In: *2018 15th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*. Mexico City, Mexico: IEEE, 2018. p. 1–6. ISBN 978-1-5386-7033-0. Disponível em: <https://ieeexplore.ieee.org/document/8533946/>.

[44] OBAYASHI, M.; NAKAHARA, N.; KUREMOTO, T.; KOBAYASHI, K. A robust reinforcement learning using the concept of sliding mode control. *Artificial Life and Robotics*, v. 13, n. 2, p. 526–530, 2009. ISSN 14335298.

[45] PETERS, J.; VIJAYAKUMAR, S.; SCHAAL, S. Reinforcement Learning for Humanoid Robotics. *Proceedings of the IEEE International Conference on Humanoid Robots*, v. 18, n. 7, p. 1–20, 2003.

[46] PETERS, J.; SCHAAL, S. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, v. 21, n. 4, p. 682–697, 2008. ISSN 08936080.

[47] SCHAAL, S. Learning from demonstration. *Advances in neural information processing systems*, p. 1040–1046, 1997. Disponível em: <http://www8.cs.umu.se/research/ifor/dl/SEQUENCELEARINIG/learning-from-demonstration.pdf>.

[48] SUBRAMANIAN, K.; ISBELL, C. L.; THOMAZ, A. L. Exploration from Demonstration for Interactive Reinforcement Learning Kaushik. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. [s.n.], 2016. p. 447–456. ISBN 9781450342391. Disponível em: <https://dl.acm.org/citation.cfm?id=2936990>.

[49] SUTTON, R. S. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In: *Machine Learning Proceedings 1990*. Austin, Texas: Elsevier, 1990. p. 216–224. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/B9781558601413500304>.

[50] OBAYASHI, M.; NAKAHARA, N.; YAMADA, K.; KUREMOTO, T.; KOBAYASHI, K.; FENG, L. A Robust Reinforcement Learning System Using Concept of Sliding Mode Control for Unknown Nonlinear Dynamical System. In: *Robust Control, Theory and Applications*. [S.l.]: InTech, 2011.

[51] KIM, D.; PARK, S. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. Vancouver, BC, Canada: IEEE, 2009. p. 76–85. ISBN 978-1-4244-3724-5. Disponível em: <http://ieeexplore.ieee.org/document/5069076/>.

[52] VIEN, N. A.; ERTEL, W. Reinforcement learning combined with human feedback in continuous state and action spaces. In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. San Diego, CA, USA: IEEE, 2012. p. 1–6. ISBN 978-1-4673-4965-9. Disponível em: <http://ieeexplore.ieee.org/document/6400849/>.

[53] STAHLHUT, C.; NAVARRO-GUERRERO, N.; WEBER, C.; WERMTER, S. Interaction is More Beneficial in Complex Reinforcement Learning Problems than in Simple Ones. In: *Proceedings of the Interdisziplinärer Workshop Kognitive Systeme*. [S.l.: s.n.], 2015. p. 142–150.

[54] MILLÁN, C.; FERNANDES, B.; CRUZ, F. Human feedback in continuous actor-critic reinforcement learning. In: *Proceedings European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Bruges (Belgium): [s.n.], 2019. p. 661–666. ISBN 9782875870650.

[55] BERNARDO, J. M.; SMITH, A. F. M. *Bayesian Theory*. [S.l.]: John Wiley & Sons, 1994. ISBN 047149464X,9780471494645.

[56] BROCKMAN, G.; CHEUNG, V.; PETTERSSON, L.; SCHNEIDER, J.; SCHULMAN, J.; TANG, J.; ZAREMBA, W. OpenAI Gym. jun 2016. Disponível em: <http://arxiv.org/abs/1606.01540>.

[57] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, v. 8, n. 3-4, p. 229–256, may 1992. ISSN 0885-6125. Disponível em: <http://link.springer.com/10.1007/BF00992696>.

ANNEX A

Modification of the Cart-pole Balancing Environment from OpenAI Gym.

The Cart-pole environment, implemented in *OpenAI Gym* toolkit [56], is a problem of classic control of continuous states and discrete actions. In this tool, the Cart-pole balancing is implemented as an episode task, and then terminal states are defined. The task ends if the pole falls ($|\phi| \geq 2.4$) or if the cart collides with the ends of the track ($|\chi| \geq \frac{\pi}{15}$) The environment has two versions, "CartPole-v0" that performs a maximum 150 iterations per execution, and "CartPole-v1" that performs a maximum of 500 iterations per execution. We use the "CartPole-v1" version in our implementations. The action of the problem represents the force F applied to the system that is taken from a discrete space (0,1). The force applied to the cart is obtained under the condition:

$$F = \begin{cases} F_{max} & \text{if} \quad \text{action is 1} \\ -F_{max} & \text{if} \quad \text{action is 0} \end{cases}.$$

The reward function implemented grants a unit of reward in each iteration. The reward is zero if the task ends, e.g., the current state is terminal. The `step(action)` function of the source code receives an action as input, it is verified if it belongs to the discrete action space and returns the next state, the reward, a done (True if the state is terminal) and information.

To carry out our implementation, we modified the source code to allow continuous actions. We eliminate the condition that verifies if the actions are discrete,

```
assert self.action_space.contains(action), "%r (%s) invalid"%(action, type(action))
```

and limit the actions to the interval $(-F_{max}, F_{max})$. The reward function was replaced by the expression in (4.1). The number of iterations of the "CartPole-v1" version was modified from 500 to 400.

Finally, we add the friction of the car on the track in the equations of motion, according to Barto et al. [5]. This modification allows changing the value of the friction coefficient to alter the movement of the car and the pendulum.