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 Dynamic Environments

Dissertação de Mestrado

Recife, Agosto 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 Dynamic Environments

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós-Graduação acadêmico em ENGENHARIA DE COMPUTAÇÃO da Universidade de Pernambuco como requisito parcial para obtenção do título de Mestre em Engenharia de Computação.

Prof. Dr. Bruno José Torres Fernandes
Orientador
Dr. Francisco Javier Cruz Naranjo
Coorientador

Recife, Agosto 2019

Dados Internacionais de Catalogação-na-Publicação (CIP)
Universidade de Pernambuco – Recife

Arias, Cristian Camilo Millán

A696i Interactive reinforcement learning for continuous spaces and dynamic environments. / Cristian Camilo Millán Arias . – Recife: UPE, Escola Politécnica, 2019.

47 f.: il.

Orientador: Dr. Bruno José Torres Fernandes
Co-orientador: Dr. Francisco Javier Cruz Naranjo

Dissertação (Mestrado – Engenharia da Computação) Universidade de Pernambuco, Escola Politécnica, Programa de Pós-Graduação em Engenharia da Computação.

1. Ambientes Dinâmicos. 2. Aprendizagem de Máquina. 3. Aprendizagem por Reforço Interativo. 4. Aprendizagem por Reforço Robusto. 5. Policy-Shaping. I. Engenharia da Computação - Dissertação. II. Fernandes, Bruno José Torres (orient.). III. Naranjo, Francisco Javier Cruz (co-orient.). IV. Universidade de Pernambuco, Escola Politécnica, Mestrado em Engenharia da Computação. V. Título

CDD: 006.3

UNIVERSIDADE
DE PERNAMBUCO

Dissertação do Mestrado apresentada por **Cristian Camilo Millan Arias,** à Pós-Graduação em Engenharia de Computação da Escola Politécnica de Pernambuco da Universidade de Pernambuco, sob o título **"Interactive Reinforcement Learning for Continuous Spaces and Dynamics Environments",** orientado pelo Prof. Bruno José Torres Fernandes – Doutor, onde foi aprovado pela Banca Examinadora formada pelos professores:

Roberta Andrade de Araujo Fagundes

Roberta Andrade de Araujo Fagundes - Doutora
(Primeira Examinadora)

[illegible]

Richard Dazeley - Doutor
(Segundo Examinador)

[illegible]

Bruno José Torres Fernandes - Doutor
(Orientador)

Visto e permitido a impressão.

Recife, 09 de agosto de 2019.

Roberta Andrade de A. Fagundes

Profª. Roberta Andrade de Araujo Fagundes
Vice-Coordenadora da Pós-Graduação em Engenharia de Computação
da Escola Politécnica de Pernambuco da Universidade de Pernambuco

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 in which an agent interacts with the environment to learn how to perform a task. Many times, learning is affected by the characteristics of the environment and the way the agent perceives it. These characteristics may change over time or be affected by outer 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 discretization process. Moreover, in continuous spaces, the agent takes too long to find optimal actions. *Interactive Reinforcement Learning* is an approach in which an external entity helps the agent to learn through feedback. There are also reliable approaches, such as *Robust Reinforcement Learning*, that allow the agent to learn a task regardless of disturbances in the environment. In this dissertation, we propose two approaches to tackle these difficulties. First, we present Interactive Reinforcement Learning in problems where states and actions are continuous. Next, we address policy-gradients methods for Robust Reinforcement Learning. Finally, we propose Robust Interactive Reinforcement Learning, an approach that includes advice in scenarios where the environment is dynamic. We propose algorithms that combine learning under Interactive Reinforcement Learning with the Robust Reinforcement Learning approach. To evaluate our proposal, we implement the dynamic version of the Robust Reinforcement Learning task, where the characteristics of the environment change in each episode. Our results show that the proposed approach allows an agent to complete the task satisfactorily in a dynamic, continuous action state domain. Moreover, experimental results suggest agents trained with our approach are less sensitive than interactive reinforcement learning agents in front of 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 aprendizado de máquina, onde, um agente interage com o ambiente para aprender como realizar uma tarefa. Muitas vezes, a aprendizagem é afetada pelas características do ambiente e a forma como o agente percebe-o. Aquelas características 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 parra desarrolhar uma tarefa. No entanto, a informação perde-se durante o proceso de discretização. Além disso, em espaços contínuos, o agente demora muito para encontrar as ações ótimas. O aprendizado por reforço interativo é uma abordagem na qual uma entidade externa ajuda aprender ao agente através do feedback. Também existem abordagens robustas, como a *Aprendizagem por Reforço Robusto*, que permitem ao agente aprender uma tarefa, independentemente das perturbações produzidas no ambiente. Nesta dissertação, propõe-se duas abordagens para enfrentar essas dificuldades. Primeiro, apresenta-se a Aprendizagem por Reforço Interativo em cénarios onde os estados e as ações estão em espaços contínuos. Logo, os métodos de gradientes de políticas para a Aprendizagem por Reforço Robusto são abordados. Por fim, propõe-se a Aprendizagem pr Reforço Interativo Robusto, uma abordagem que inclui informação externa em cenários donde o ambiente é dinâmico. Propõe-se algoritmos que combinam a Aprendizagem por Reforço Interativo com a abordagem de Aprendizagem por Reforço Robusto. Para avaliar a proposta, implementa-se a versão dinâmica da tarefa do *balanceamento de carrinho de pólocart-pole balancing*, onde as características do ambiente mudam em cada episódio. Os resultados mostram que a abordagem proposta permite que um agente conclua a tarefa satisfatoriamente em um domínio de estado de ação dinâmico e contínuo. Além disso, os resultados experimentais sugerem que os agentes treinados com a abordagem proposta são menos sensíveis do que os agentes de Aprendizagem por reforço Interativo diante de 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, en el cual, un agente interactúa con el ambiente para aprender como realizar una tarea. Muchas veces, el aprendizaje se ve afectado por las características del entorno y la forma en que el agente lo percibe. Estas características pueden cambiar con el tiempo o verse afectadas por perturbaciones externas no controladas por el agente. 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 la tarea. Sin embargo, la información se pierde durante el proceso de discretización. Además, en espacios continuos, el agente toma bastante tiempo encontrando las acciones optimas. El *Aprendizaje por Refuerzo Interactivo* es un enfoque en el que una entidad externa ayuda al agente a aprender a través de la retroalimentación. También existen enfoques robustos, como *Aprendizaje por Refuerzo Robusto*, que permiten al agente aprender una tarea independientemente de las perturbaciones que modifican el ambiente. En esta disertación, se proponen dos enfoques para abordar estas dificultades. Primero, se presenta el Aprendizaje por Refuerzo Interactivo en problemas donde los estados y las acciones son continuas. A continuación, se aborda los métodos de gradientes de políticas para el Aprendizaje por Refuerzo Robusto. Finalmente, se propone el Aprendizaje por Refuerzo Interactivo Robusto, un enfoque que incluye información externa en escenarios donde el ambiente es dinámico. Se proponen algoritmos que combinan el Aprendizaje por Refuerzo Interactivo con el enfoque de Aprendizaje por Refuerzo Robusto. Para evaluar la propuesta, se implementa la versión dinámica de la tarea *cart-pole balancing*, donde las características del entorno cambian en cada episodio. Los resultados muestran que el enfoque propuesto permite a un agente completar la tarea satisfactoriamente en un dominio de estado de acción dinámico y continuo. Además, los resultados experimentales sugieren que los agentes entrenados con el enfoque propuesto son menos sensibles que los agentes de Aprendizaje por Refuerzo Interactivo frente a los 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 dissertation 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, patience, and all the direction.

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 right 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

In this chapter, we introduce the problem that is addressed in this work. First, we present the characterization of the problem and motivation. We explain the context and issues related to Reinforcement Learning, as well as possible successes found in the literature. Then, we present the general objective and the specific objectives that are carried out to solve the problem.

1.1. Problem Characterization

Reinforcement learning (RL) is a learning approach that tries to solve the problem of an agent interacting with the 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 it tries to maximize throughout the learning [1]. Thus, the agent learns from its 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. Temporal-difference (TD) learning [1] is a class of methods to adjust the state value function. The conventional algorithms from TD learning are Q-learning [4], and SARSA algorithm [5], which update the action-value function [1], and the actor-critic, that represents the policy (actor) independent of the state-value function (critic) [6, 7].

One RL problem, which is still open, is the time spent by an RL agent during learning [8]. To leave an agent finds a proper policy, requires excessive time, mainly due to a complex or large space of states and actions. Moreover, the agent may explore different regions of space to find the state-action pair that produces a better reward [9]. To overcome this problem, an RL agent may be guided by a trainer to carry out the task more rapidly. Interactive Reinforcement Learning (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 leading to improve the performance of the task and speed up the learning. Allowing to reduce the search space and to learn autonomous the problem faster compared to an RL agent performing exploration [8]. Thomaz and Breazeal [10] 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 autonomous RL. In their work, they show that the users (humans) guide the agent towards actions and give anticipatory rewards but provides more positive than negative feedback. In other work, Thomaz and Breazeal [9] involve a boolean feedback signal by the trainer when the agent needs it. The *UNDO* behavior is implemented to changing the selected action in some step.

In many RL implementations, the space of states and actions is usually considered a discrete domain [11, 1, 12] or a discrete representation (coarse coding or function approximation) [13, 14, 8]. Still, in real-world applications, this is 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 [15, 16]. Nevertheless, a very fine discretization can be considered to capture all the possible information. However, this leads to slow learning by considering spaces with many elements.

Doya [17] proposes an approach of reinforcement learning in continuous spaces of time, states, and actions. Besides, he suggests the continuous actor-critic method and a gradient approximation to performs the policy. In this work, the author uses Normalized Gaussian Networks [18] as a function approximation, and implements her approach in two tasks, swinging up a pendulum and the cart-pole balancing. The results show that continuous actor-critic has an advantage over the discrete one; it is more efficient and stable. Also, agents learn in fewer iterations. Van Hasselt and Wiering [16] propose the Continuous Actor-Critic Learning Automaton (CACLA) to carry actions and states in continuous spaces, an algorithm relatively simples o implement and low computational requirements. They use Neural Networks as function approximation and implement their algorithm in the tracking and the cart-pole balancing task. They demonstrate that their algorithm performs better than other algorithms, uniquely combining it with Gaussian exploration.

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 its policy and better perform the 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 essential to develop robust algorithms that help to overcome uncontrollable disturbances. One of the most relevant works is from Morimoto and Doya [19], where propose Robust Reinforcement Learning, an approach capable of resisting the disturbances present in the environment based on the control H^{∞} paradigm. They implement their methodology in the cart-pole balancing task, where they change the physical parameters of the cart-pole after training. Their experiments show a poor performance during the training; however, the agent is more robust in front of the change of the physical parameters.

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 assess the performance of the method. In this research, we propose an Interactive Reinforcement Learning approach 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. Besides, we propose a group of algorithms to address our propose and help in the implementation. To evaluate this approach, we implemented the classic control problem, cart-pole balancing task, with the characteristics necessary to assess the performance of the method. To determine the performance of the proposal, we compare the curves of the average collected reward and the average number of steps per episode through a graphical method (observing the trajectories) as commonly performed in the Reinforcement Learning area [1, 17, 20, 14, 8, 21].

1.2. Objectives

This work aims to propose a robust approach to implement Interactive Reinforcement Learning in problems where states and actions are in a continuous domain and a dynamic environment so that external factors to the environment are not influential during learning. The approach will be efficient if: it can accelerate the training of an agent that faces different types of settings, and the agent is robust in front of the change of the characteristics of the 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 research about Robust Reinforcement Learning approach based on dynamic environments.
3. To include the concepts of Interactive Reinforcement Learning in dynamic environments based on the approaches 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 approach is effective in a classic control benchmark.

1.3. Structure of the Document

This master dissertation is organized in five chapters. In Chapter 2, we describe the theoretical framework from reinforcement learning for the understanding of this work and the related approaches. In Chapter 3, we present a way to implement Interactive Reinforcement Learning for continuous spaces. We also show the proposal Robust Reinforcement Learning approach with policy gradients, and Robust Reinforcement Learning that including advice from an external trainer. The experimental setup, results, and discussion are presented in Chapter 4. Finally, in Chapter 5, we describe our 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 of modification in stimulus-response relations is developed as a consequence of interaction with the environment, via the senses [22]. An approach based on learning is appropriate when the environment is wholly unknown, or it is not available at the moment of designing a solution; this provides autonomy to an organism.

It is generally agreed that learning involves a relatively permanent change in behavior due to experience, reward, and punishment. It is a consequence of the law of reinforcement, an essential principle in all learning theories [23, 24]. Reinforcement learning (RL) [1] is a learning approach that allows autonomous agents to learn new skills, using the reaction from the environment to an action. The main idea is trying to select actions 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 [24]. The RL problem is reduced to learn how to select optimal actions to be performed in each situation to reach a given goal [25].

2.1.1. Elements of Reinforcement Learning

Besides the agent and its environment, exist fourth main elements can be identified 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 it 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 the 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 long-term desirability that indicates how good is the state concerning the goal. Therefore, starting from an anyone state, the value is the accumulative reward over the future an agent can 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 [24]. 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

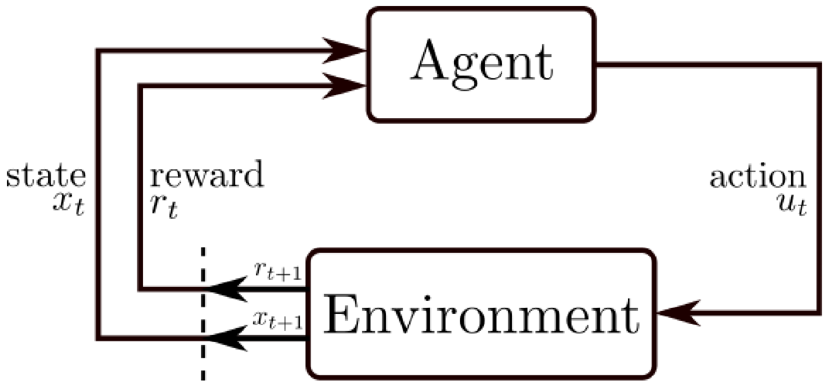
Reinforcement learning is a learning approach that involves an autonomous agent learning from interactions with its environment to achieve the goal [1]. The agent must be able to learn from its own experience selecting actions that affect the environment and, with these actions, reach new situations to the agent. In this approach, the learning agent receives from its environment numerical reward information from the environment that attempts to maximize [26]

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 $x_t \in X$, and of the action selected by the agent $u_t \in U(x_t)$, where X is a set of the all possibles states, and $U(x_t)$ 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. RL method specifies how the agent change the policy as consequence of its experience. Thereby, the goal of the RL 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 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 [27]. 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 [28]. 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. In a particular case, the return can be

[2] The sets X and U_x can be arbitrary finite or countably infinite, discrete or continuous sets [27].

defined as the accumulated sum of the reward sequence [1]:

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

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 [29].

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:

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

where γ is a parameter called the discount rate, $0 < \gamma \leq 1$, 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 [30].

The *state value function* of x, under all the actions, is the expected return of the discounted sum starting in the state x an following a policy π. Formally we can defined it by [1, 30]:

$$V^\pi(x) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| x_t = x \right], \tag{2.2}$$

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

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

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

In addition, the value functions satisfy a recursive property. The expression (2.2) can be defined recursively in terms of the so-called *Bellman equation* [1, 31]. It is denoted as the expected return

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

[4] A summation or integrals define the expected value according to the nature of the sets X and U.

in terms of the immediate reward and the value of the next state, defined formally by:

$$V^\pi(x) = \int_{\mathbb{U}} \pi(u|x) \int_X P(x'|x,u) \left[\rho(x,u,x') + \gamma V^\pi(x')\right] dx'du, \tag{2.4}$$

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 in these state. The RL goal is to find an optimal policy leading to maximizing the reward over the long run [32]. 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 (2.4).

The *temporal-difference learning* (TD) is a method for solving the Bellman equation. Algorithms based on TD learn estimates values based on another estimates by adjusting the gain against the ideal equilibrium that holds locally when the gain estimates are correct[5]. 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 [33, 30]. 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:

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

where α is a learning rate that is determined by how much values get updated [33]. 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, 34], SARSA [5], and R-Learning [1] are TD methods that learn the action-value function based on state-action pairs. In this work, we do not focus on these methods; however, the action-value function is essential to introduce topics in the next section. Following, we present the actor-critic algorithm, an iterative method, and the basis of our approach.

2.4.1. Actor-Critic

The actor-critic (AC) algorithm is a method based on TD learning that keeps a separate memory structure to represent the policy independent of the value function [6, 1]. The agent is

[5] The bootstrapping methods are strategies to estimate through 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: $\gamma$, $\alpha$
 1: Initialize $V(x_t)$ arbitrarily, $\pi$ to the policy to be evaluated
 2: for each episode do
 3:     initialize $x_t$
 4:     repeat
 5:         $u_t \leftarrow$ given by $\pi(u|x_t)$
 6:         Taken action $u_t$, observe reward $r_{t+1}$ and next state $x_{t+1}$
 7:         $V(x_t) \leftarrow V(x_t) + \alpha\left[r_{t+1} + \gamma V(x_{t+1}) - V(x_t)\right]$
 8:         $x_t \leftarrow x_{t+1}$
 9:     until $x_t$ is terminal
10: end for
```

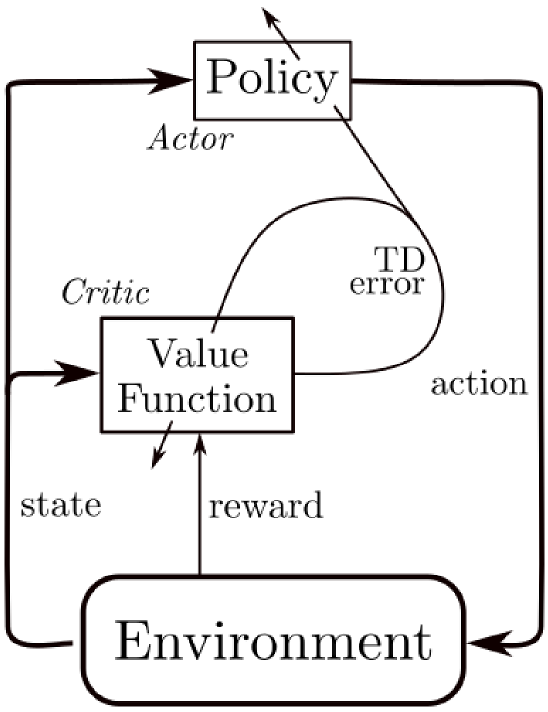
separate into two entities: the actor and the critic. The policy takes the role of the actor, selecting actions in each iteration. The critic, commonly a state-value function, evaluates or criticizes the actions performed by the actor [35]. Fig. 2 shows the schematic structures of the AC algorithm.

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

$$\delta_t = r_{t+1} + \gamma V(x_{t+1}) - V(x_t), \tag{2.5}$$

which is the output of the critic and the diference between the right hand and left hand sides of the Bellman equation (2.4). 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. However, if the TD error is negative, the tendency should be weakened that action selection in a determined state. [30]. The action is the output of the actor. The TD error improves

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



the preference to select each action. In each iteration is adjusted depending on the performance of the agent control over the environment. There exist variations of the AC algorithm 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

$$\upsilon(u_t, x_t) \leftarrow \upsilon(u_t, x_t) + \alpha_\upsilon \delta_t. \tag{2.6}$$

The following sections will be introduced other methods to select actions based on AC algorithm. An advantage of using this algorithm is the separate structure of the policy, which requires minimal computation to choose actions. In large action spaces, to consider the $X \times U$ or all Q is no necessary. Moreover, an additional advantages is to allow an agent to 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 [36] are a methodology to approximate functions in RL. These methods are the most popular class of continuous action RL algorithms [37]. With this approach, a stochastic policy is approached through an approximation function, independent of the value function, with its 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, parametrize by a column vector of weights $\upsilon \in \mathbb{R}^{N_a}$, for $N_a \in \mathbb{Z}$. $\pi(u|x)$ 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:

$$\Gamma(\pi) = \int_X d^\pi(x) \int_U \pi_\upsilon(u|x) Q^\pi(x,u) du dx, \tag{2.7}$$

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 [38] and $Q^\pi(x,u)$ the action-value function defined in (2.3). 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* [36], which defines the gradient as:

$$\nabla_\upsilon \Gamma(\pi) = \int_X d^\pi(x) \int_U \nabla_\upsilon \pi_\upsilon(u|x) Q^\pi(x,u) du dx \tag{2.8}$$

The main difficulty with this method is to find 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 [36, 39]. Let $h_\theta : X \times U \longrightarrow \mathbb{R}$ be 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

$$\epsilon_t^\pi(x,u) = \frac{1}{2}\left[Q^\pi(x,u) - h_\theta(x,u)\right]^2 . \tag{2.9}$$

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 (2.8) is expressed as:

$$\nabla_v \Gamma(\pi) = \int_X d^\pi(x) \int_U \nabla_v \pi_v(u|x) h_\theta(x,u) du dx \tag{2.10}$$

This approximation is acceptable if $\nabla_\theta h_\theta(x,u) = \nabla_v \log\left(\pi_v(u|x)\right)$ is satisfied. The expression is called compatible features [36, 7], establishes 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 given 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 $Q^\pi(x,u)$ [40]. 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 [39]. The value function $V^\pi(x)$ is a baseline in the advantage function, such that the variance of the policy-gradient is minimized [41].

2.5.1. Actor-Critic with Policy-gradient

In other view,

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

then, according to Sutton and Barto [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 (2.5). Thus h_θ would approximate a value function $V^\pi(x)$, and the gradient in (2.10) taken form of:

$$\begin{aligned} \nabla_v \Gamma(\pi) &= \int_X d^\pi(x) \int_U \nabla_v \pi_v(u|x) \hat{\delta}_t du dx \\ &= E_\pi\left[\hat{\delta}_t \nabla_v \log\left(\pi_v(u|x)\right)\right], \end{aligned} \tag{2.11}$$

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 θ. The parameter θ is adjusted by the gradient of the quadratic error (2.9) 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 (2.11) 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 (2.6) 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 approach in the following chapters.

Algorithm 2 General episodic actor-critic algorithm with policy-gradient.

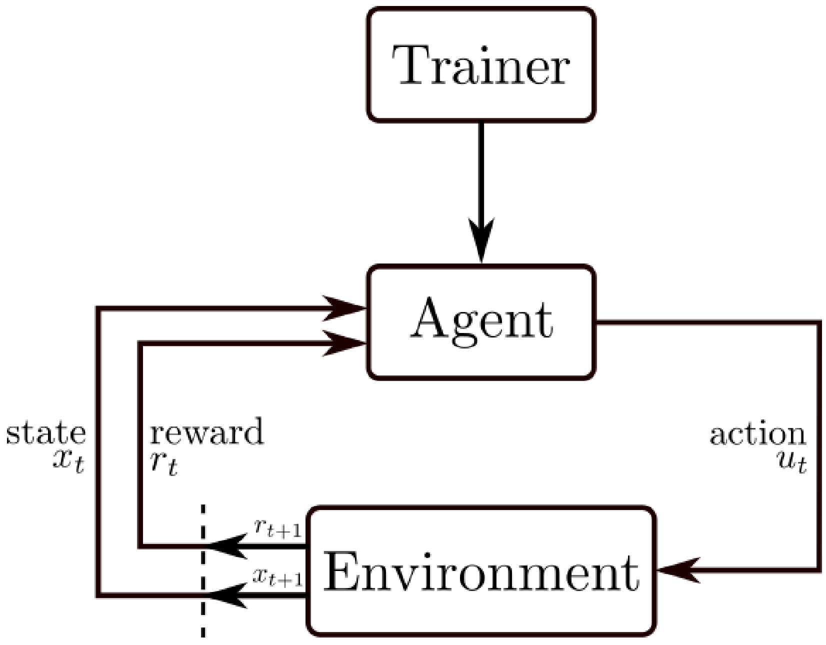
```
Inputs: $\gamma$, $\alpha_c$, $\alpha_a$
 1: Initialize $\theta_t$ and $\upsilon_t$ arbitrarily
 2: for each episode do
 3:     initialize $x_t$
 4:     repeat
 5:         $u_t \leftarrow$ given by $\pi_\upsilon(u|x_t)$
 6:         Taken action $u_t$, observe reward $r_{t+1}$ and next state $x_{t+1}$
 7:         $\delta_t \leftarrow r_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
 8:         $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log\left(\pi_{\upsilon_t}(u_t|x_t)\right)$
 9:         $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
10:         $x_t \leftarrow x_{t+1}$
11:     until $x_t$ is terminal
12: 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 perceive states 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 [42].

On some occasions, to leave that an agent learns a task by itself is impractical and involves problems to find the proper policy [43]. Interactive Reinforcement Learning (IRL) is an approach that considers a knowledgeable trainer, which gives advice or guidance to the RL agent, having an effect of restricting the actions selection to those related to the target object

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



[44]. Fig. 3 shows a general view of the IRL approach, an external trainer is added during the learning process to modify the agent decision.

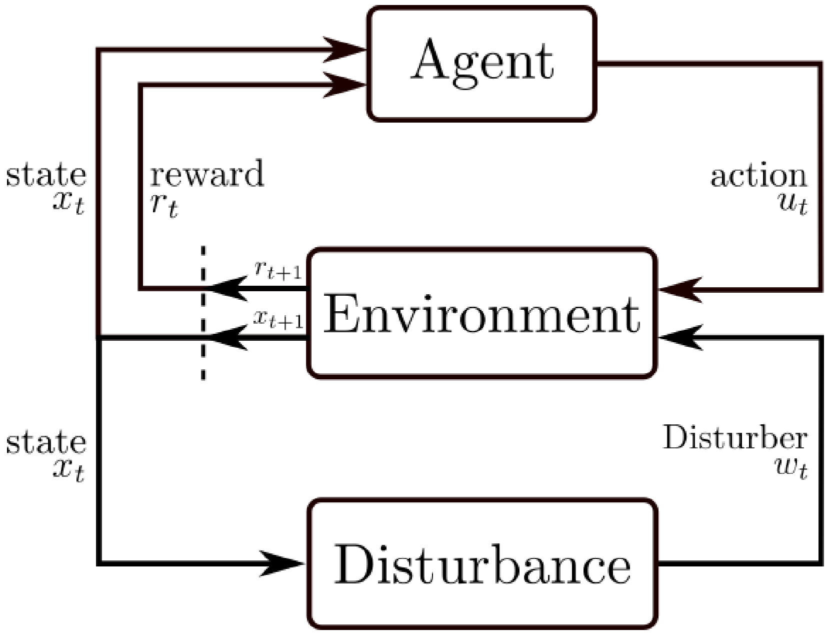
In an IRL scenario, it is desired that the interaction between the external trainer and the agent be as minimal as possible; otherwise, it could become supervised learning. Another important aspect is the quality of the advice, since the external trainer may make mistakes the agent may not improve with its training [14, 45]. The guidance 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 [46]. The first is *reward-shaping*, where the external trainer modifies the reward by sending its reward to the agent to inform how good the selection of the action was in the previous step [12, 44]. The second approach is *policy-shaping*, where the external trainer modifies the action just selected by the agent. This approach tells the agent that its current performance is wrong and should improve for the future [10, 43].

2.7. Dynamic Approach: Robust Reinforcement Learning

During the learning, the agent performs an action that stimulates the environment in some way. Assuming that, this last one 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 [47]. There are also parameters in the system

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



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 agent can receive 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.

Morimoto and Doya [19] present the Robust Reinforcement Learning (RRL), an approach that introduces a disturber who provides disturbance to the environment. To resist a disturbance input, it considers 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 [21]:

$$q(u_t, x_t, w_t) \leftarrow \rho(u_t, x_t) + \omega(w_t), \tag{2.12}$$

where $\rho(u_t, x_t)$ is the reward function defined in the section 2.3.

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 [48]. Fig. 4 shows a view of the RRL approach [19], where an external disturber providing 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 [17] where implements a framework of RL in continuous spaces of time, states and actions, also proposes the continuous actor-critic algorithm and a gradient approximation to performs the policy. Van Hasselt and Wiering [16] propose the Continuous Actor-Critic Learning Automaton (CACLA) to carry actions and states in continuous spaces. Policy-gradient methods and function approximator [15, 36] are more used in continuous RL. In our work, we use the Actor-Critic algorithm and policy-gradients to learn an optimal policy. Unlike Doya [17], we do not use information from the environment model, only the states and the reward obtained. During training, the policy is updated in each iteration, contrary to what is presented in [16].

Researches in robotics and parametrize motor skill are developed and applied in the works [49, 50, 38]. An advantage of using several variations of actor-critic algorithms [39] is consider different policy nature. Silver et al.[37] 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, Schaal [51] 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. [52] 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 [10] present the first IRL approach, an approach where an external trainer guides the agent to complete a task.

Suay and Chernova [44] 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 [12]. 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. [14] 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. The algorithm used in this approach is Bayesian Q-learning [11]. Besides, this approach 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.

Unlike the works presented by Thomaz and Breazeal [10, 9], Suay and Chernova [44], and Griffith et al. [14], our work focuses on states and continuous actions. We use the actor-critic algorithm, in contrast to the Q-learning algorithm that is used in the works mentioned above. Our approach takes elements from Griffith et al. [14], agent policy, and feedback policy as sources of variation. However, our approach assumes that the feedback policy, or the probability of giving advice, is unknown, and its form depends on the characteristics of the environment and the external trainer. On the other hand, our implementation does not consider Bayesian inference assumptions during the estimation of parameters as considered by Bayesian Q-learning [11].

In the area of dynamic environments, Sutton [53] 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 [19] present Robust Reinforcement Learning, a methodology to train more robust agents against dynamic environments. They propose the Actor-Disturber-Critic (ADC), an algorithm that introduces a disturbance in the classic A actor-critic. Although learning is slow, they implement continuous domains for actions. In our proposal, we use policy-gradients, based on a cost function, to synchronize an optimal policy and the distribution that generates the disturbances. On the other hand, we carry learning independent from the environment model. Obayashi et al. [48, 54] implement the Robust Reinforcement Learning methodology using the concept of sliding mode control on the inverted pendulum.

Dongsun and Park [55] 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. [47] 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.

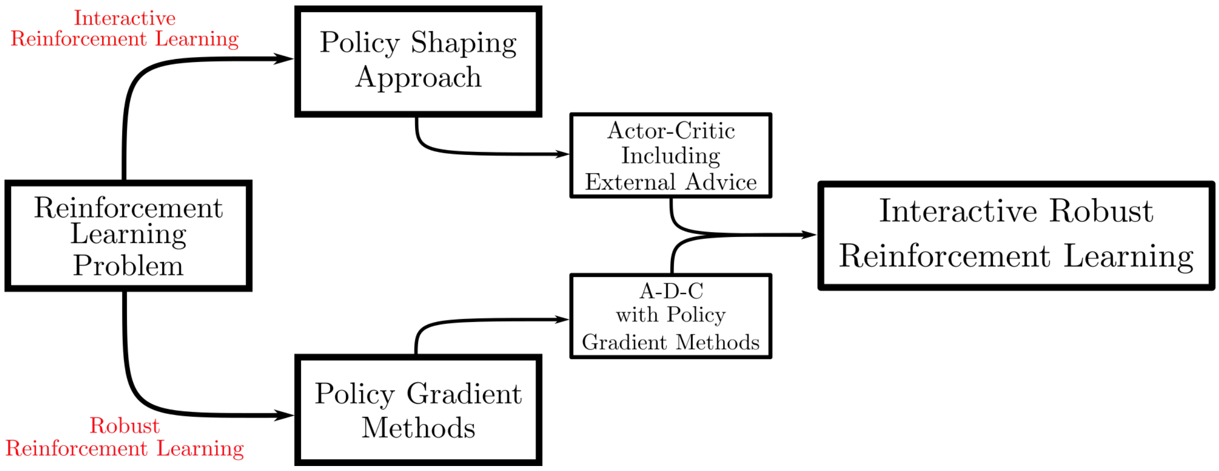
Finally, there are some works on RL with external feedback in continuous spaces. Vien y Ertel [56] propos ACTAMER, an extension of TAMER RL for continuous states and actions that includes an external reinforcement signal to improve learning. Their proposal was implemented in the cart-pole balancing task and the mountain-cart task. The TAMER framework considers learning an optimal policy that compares to a way of acting from the external trainer. Our approach considers the external trainer as a guide that will give feedback so that the agent learns his policy. Stahlhut et al. [57] powered by Interactive Continuous Actor-Critic Automaton (ICACLA), an algorithm based on CACLA [16] that includes an external capacitor. Based on the policy shaping methodology and the capacitor guides the agent to change the selected action through UNDO [10]. Millán et al. [21] 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. The flow chart of the Fig. 5 shows how our methodology is carried out. Based on a Reinforcement learning methodology: first, the issue of Interactive Reinforcement Learning for continuous spaces will be addressed. A policy-shaping approach is suggested to including external advice in the actor-critic algorithm. Then, we including policy gradient methods in an actor-disturber-critic algorithm to implement Robust Reinforcement Learning. Finally, Interactive Robust Reinforcement Learning will be addressed.

Figure 5 – A flowchart that explains how our methodology is carried out. In the upper part, external advice is including in the actor-critic algorithm. In the lower part, the policy gradient method is including in the actor-disturber-critic algorithm.


Interactive
Reinforcement Learning
Policy Shaping Approach
Actor-Critic Including External Advice
Reinforcement Learning Problem
Interactive Robust Reinforcement Learning
A-D-C with Policy Gradient Methods
Policy Gradient Methods
Robust
Reinforcement Learning

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 reaching a goal. In some iteration steps, the trainer may not provide feedback. Thus, the likelihood of receiving feedback [14] has probability $0 < L < 1$.

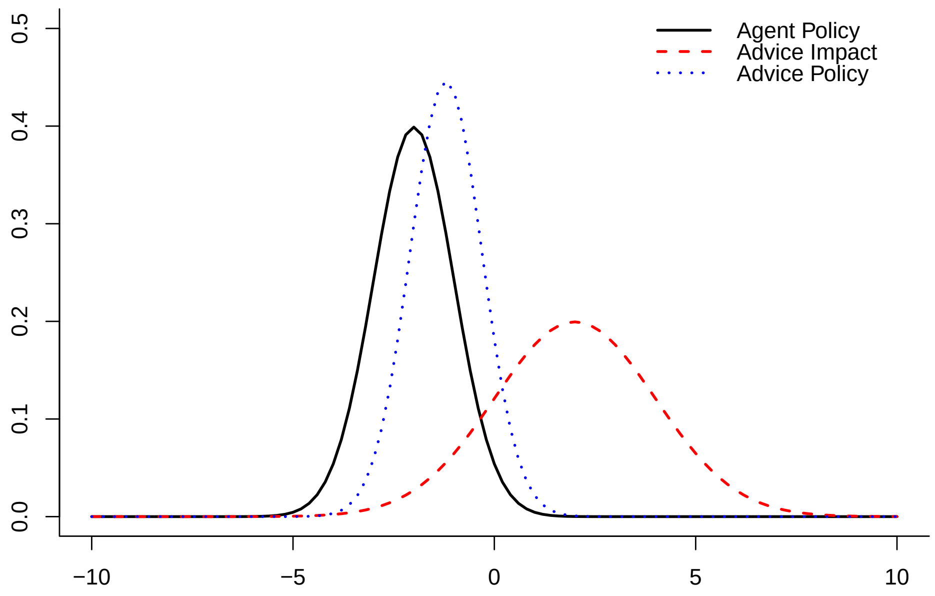
Suppose that any action u_t is selected from a policy π given that an external trainer provides advice J_t in the state x_t. Let $\pi(u|x_t, J_t)$ be the probability density function of the actions after taking into account the advice J_t and the state x_t. In this sense, the policy is modified just after the trainer gives advice; nevertheless, the agent has only access to its policy $\pi(u|x_t)$, and the actions are chosen from there. 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 [21]:

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

This equation shows a clear relation between the policy $\pi(u|x, J)$ and the agent policy. The probability distribution $P_J(J|u, x)$ expresses evidence to give advice when the agent chooses the action u in the state x. Besides, the probability distribution $P_J(J|x)$ represents evidence to give advice when the agent reaches the state x. The agent policy $\pi(u|x)$ is the probability of selecting an action u before obtaining advice J. In the context of Bayesian inference, this represents the prior distribution of the action u in the state x before the advice J is observed [58].

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 improves the policy and increases the probability of select one action. Particularly, $P_J(J|u, x)$ favors a subset the actions during the process of selecting actions [21]. Fig. 6 shows 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 options to choose actions in one region with greater probability, however, the factor $\frac{P_J(J|u,x)}{P_J(J|x)}$ grants 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 feedback.

Figure 6 – Impact of the receiving advice $\frac{P_J(J|u,x)}{P_J(J|x)}$ on the agent policy $\pi(u|x)$.



The expression (3.1) involves two sources of variation as in [14], 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.

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 improve the probability ($\pi(u|x, J) = \pi(u|x)$).

The full implementation of the actor-critic with interaction is shown in Algorithm 3. The

Algorithm 3 Episodic Actor-Critic algorithm with advice and policy-gradient.

```
Inputs: $\gamma$, $\alpha_c$, $\alpha_a$, $L$
 1: Initialize $\theta_t$ and $\upsilon_t$ arbitrarily
 2: for each episode do
 3:     initialize $x_t$
 4:     repeat
 5:         $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t, J_t)$ with $J_t = \emptyset$ (Non advice)
 6:         if $rand(0, 1) < L$ then
 7:             Get advice $J_t$
 8:             Change actions $u_t \leftarrow \pi_{\upsilon_t}(u|x_t, J_t)$
 9:         end if
10:         Taken action $u_t$, observe reward $r_{t+1}$ and next state $x_{t+1}$
11:         $\delta_t \leftarrow r_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
12:         $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log(\pi_{\upsilon_t}(u_t|x_t, J_t))$
13:         $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
14:         $x_t \leftarrow x_{t+1}$
15:     until $x_t$ is terminal
16: end for
```

red statements show the differences to actor-critic from the Algorithm 2. In this algorithm we use the *advice* method parameter for interaction [14], i. e., the probability of receiving advice L (line 6).

3.2. Policy-Gradient Methods for Robust Reinforcement Learning

In the work presented by Morimoto and Doya [19], they implement an AC algorithm with a disturbance input, the so-called Actor-Disturber-Critic (ADC). In their methodology, a function approximation is used to represent the value function, the agent policy, and the function that generates the disturbance, the so-called disturber. In section 2.7 is mentioned that an increased reward is used to withstand the maximum noise generated by the model.

The policy-gradient method presented by Sutton [36], and described in section 2.5, proposes using the cost function (2.7) to maximize the performance of the policy. We include policy-gradients in the ADC of this approach; the main idea is to consider an objective function for agent policy and the disturber. In the disturber, the cost function evaluates the performance of the distribution in generating disturbances that have a more significant impact on the states, and in the selection of the next action. 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 generate 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: $\gamma$, $\alpha_c$, $\alpha_a$, $\alpha_a$
 1: Initialize $\theta_t$ and $\upsilon_t$ arbitrarily
 2: for each episode do
 3:     initialize $x_t$
 4:     repeat
 5:         $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t)$
 6:         $w_t \leftarrow$ given by $\kappa_{\omega_t}(w_t|x_t)$
 7:         Taken action $u_t$, observe reward $r_{t+1}$ and next state $x_{t+1}$
 8:         $\delta_t^w \leftarrow q_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
 9:         $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log(\pi_{\upsilon_t}(u_t|x_t))$
10:         $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
11:         $\omega_{t+1} \leftarrow \omega_t - \alpha_\omega \delta_t^w \nabla_{\omega_t} \log(\kappa_\omega(w_t|x_t))$
12:         $x_t \leftarrow x_{t+1}$
13:     until $x_t$ is terminal
14: end for
```

Algorithm 5 Episodic actor-disturber-critic algorithm with advice.

```
Inputs: $\gamma$, $\alpha_c$, $\alpha_a$, $\alpha_a$, $L$
 1: Initialize $\theta_t$ and $\upsilon_t$ arbitrarily
 2: for each episode do
 3:     initialize $x_t$
 4:     repeat
 5:         $u_t \leftarrow$ given by $\pi_{\upsilon_t}(u|x_t, J_t)$ with $J_t = \emptyset$ (Non advice)
 6:         if $rand(0,1) < L$ then
 7:             Get advice $J_t$
 8:             Change actions $u_t \leftarrow \pi_{\upsilon_t}(u|x_t, J_t)$
 9:         end if
10:         $w_t \leftarrow$ given by $\kappa_{\omega_t}(w_t|x_t)$
11:         Taken action $u_t$, observe reward $r_{t+1}$ and next state $x_{t+1}$
12:         $\delta_t^w \leftarrow q_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$
13:         $\upsilon_{t+1} \leftarrow \upsilon_t + \alpha_\upsilon \delta_t \nabla_\upsilon \log(\pi_{\upsilon_t}(u_t|x_t, J_t))$
14:         $\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \delta_t \nabla_{\theta_t} V_\theta(x_t)$
15:         $\omega_{t+1} \leftarrow \omega_t - \alpha_\omega \delta_t^w \nabla_{\omega_t} \log(\kappa_\omega(w_t|x_t))$
16:         $x_t \leftarrow x_{t+1}$
17:     until $x_t$ is terminal
18: end for
```

3.3. Interactive Robust Reinforcement Learning Approach

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 (IRRL). This approach involves advice for the agent to learn a task from an environment that has dynamic features. The advantage of this is that the agent learns the task quickly, compared to an autonomous agent, and is more robust against changes in the environment independent of actions. Algorithm 5 shows the episodic ADC with advice in the context of IRRL. The underlined statements show the differences to ADC from Algorithm 4.

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* [6]. The objective is to balance a pendulum by applying a force on the cart to which is attached. Cart-pole balancing is a continuous task, but in our implementations, we use it as an episodic problem. Thus, the agent task ends when the pole falls, i.e., the agent does not reach to balance the pendulum, or after 400 iterations in the episode. Fig. 7 shows a schematic representation of the cart-pole balancing problem. The cart is free to move within the limits of a one-axis road. The pole is free to spin 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 [6].

Figure 7 – 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].

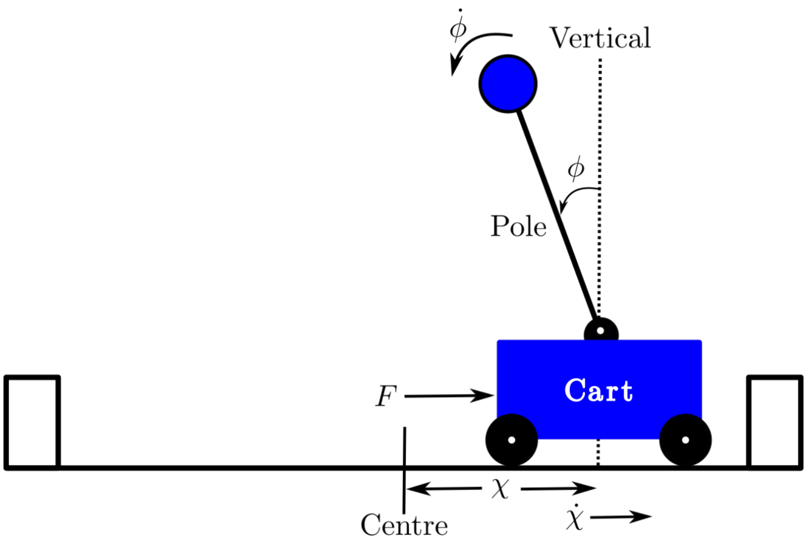


Table 1 – System parameters of the Cart-pole balancing task.

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 [59], 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 normalization allows us to delimit the states in an interval $(-1, 1)$ so 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 [60] of mean $\mu_v(x)$ and standard deviation σ_x. The value $\mu_v(x)$ 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 the 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 the mean vector $\mu_\omega(x)$ and covariance matrix Σ_w. Variances from policy and disturber are fixed during learning, however, with the ability to give the necessary exploration [60].

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 [61]. The approximation is carried out 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 $\mu_v(x)$ 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 $\alpha_\theta = 0.001$, $\alpha_v = \alpha_\omega = 0.0001$, and the discount factor γ is set at 0.9. Each set-up has been carried out 20 times using the average steps and average collected reward for the analysis. During the learning, we decided that the episode end 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 network are randomly initialized from a normal probability distribution with mean 0 and standard deviation 1 [16, 61].

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.

Fig. 8 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.

Figure 8 – 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.

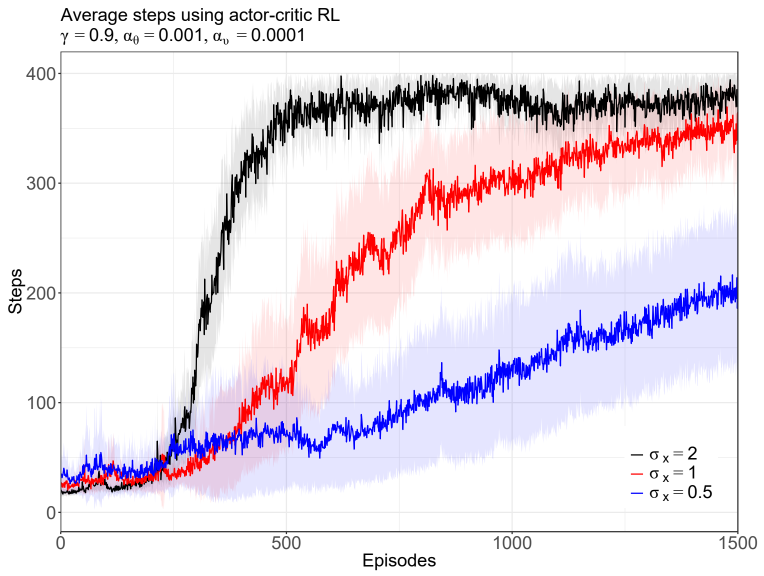
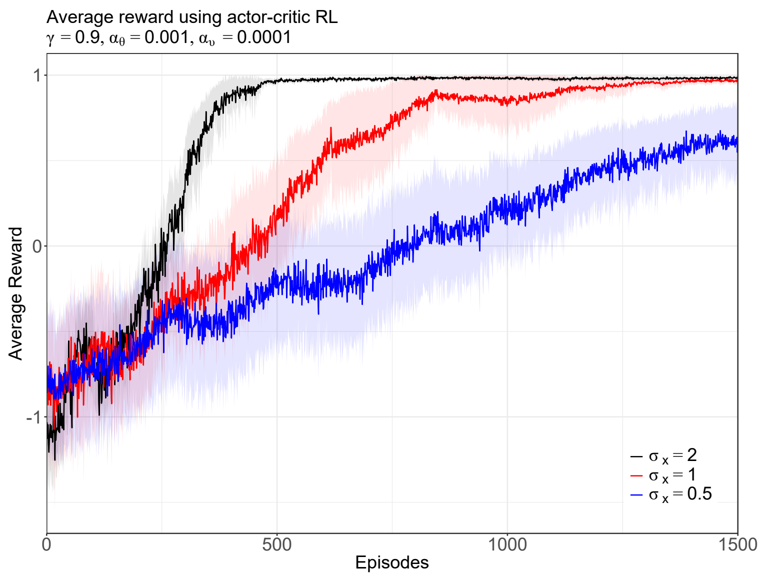


Fig. 9 shows the average rewards collected by the agent over 1500 episodes for different values of σ_x. It can be seen that the curves start with values around -1 which means that at the beginning the agent is not able to keep the pole balanced or the cart collides with the limit of the track, that behavior 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 9 – 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:

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

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:

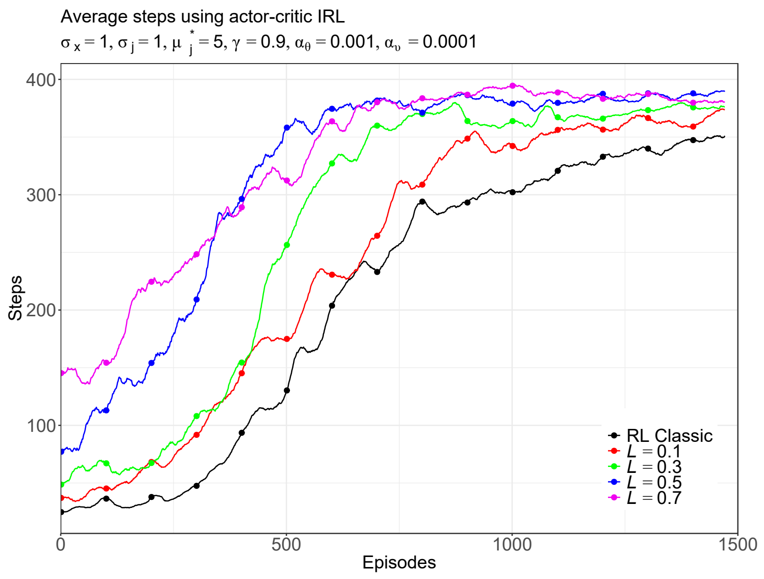
$$\pi_\vartheta(u|x,J) = \frac{1}{\sqrt{2\pi\sigma_{XJ}^2}} \exp\left\{-\frac{(u-\mu_{XJ})^2}{2\sigma_{XJ}^2}\right\},$$

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 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.

Fig. 10 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. 11 shows the average collected reward by the agent over the episodes for different values of L.

Figure 10 – 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.



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.

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. 12 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. 13), the agent achieves rewards of 1 with standard deviations less than $\sigma_J = 1$.

Figure 11 – 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.

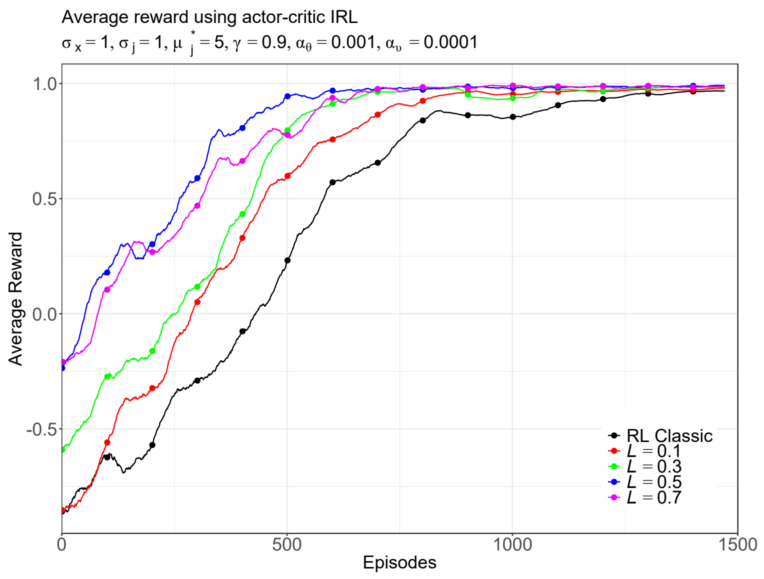
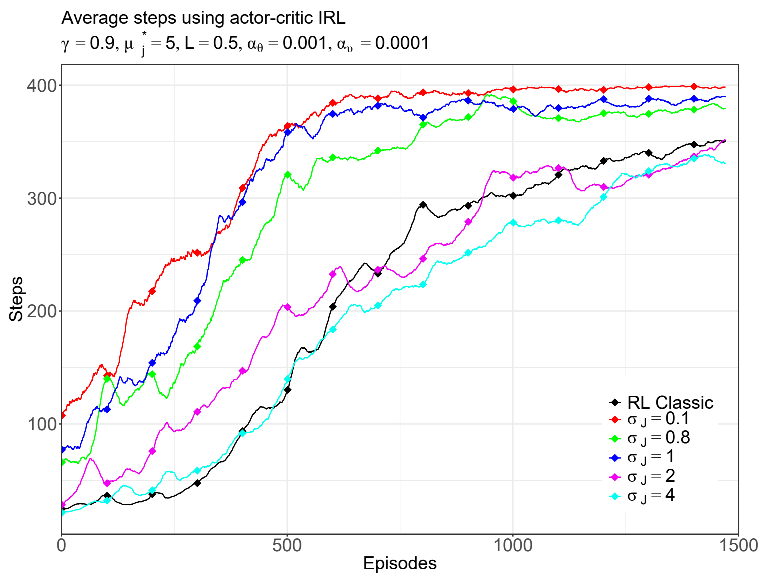
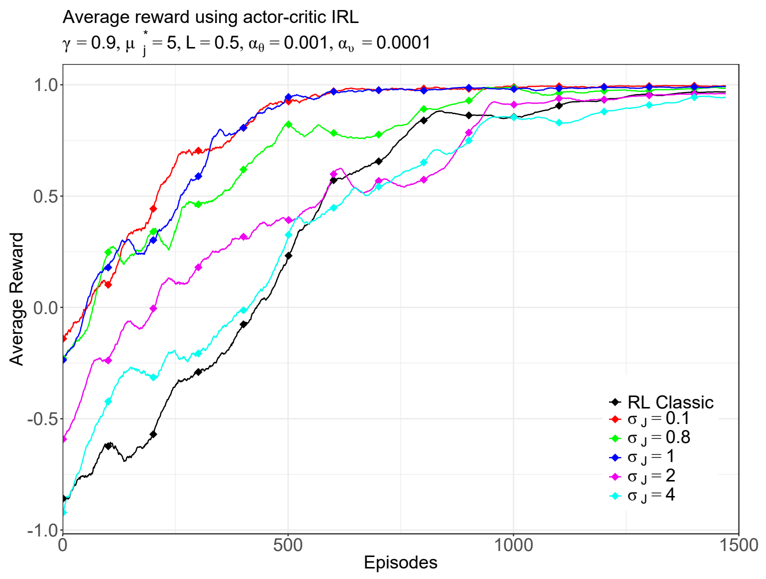


Figure 12 – Average steps over 15 runs using IRL with different values of standard deviation σ_J, mean $\mu_J^* = 5$ and probability of likelihood $L = 0.5$. The black line shows the average steps with RL which is equivalent to $\sigma_J \longrightarrow \infty$. The lower the values of σ_J, the agent takes advantage by augmenting the steps which it remains the pole balanced.



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 $\mu_J(x) = \mu_\upsilon(x) + J\mu_j^*$, from the advice policy, we can say that $\mu_{JX} = \mu_\upsilon(x) + J\mu_j^*/2$. Then, if the value of μ_j^* is close to zero, the advice will not be influential

Figure 13 – 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 $L = 0.5$.



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.

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. 14 shows the average steps taken by the agent to complete the task; the black line indicates the steps of 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. 15 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 (2.12) of the form

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

where † is the transpose of a vector and η is a parameter of robustness [19]. 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.

Figure 14 – 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.

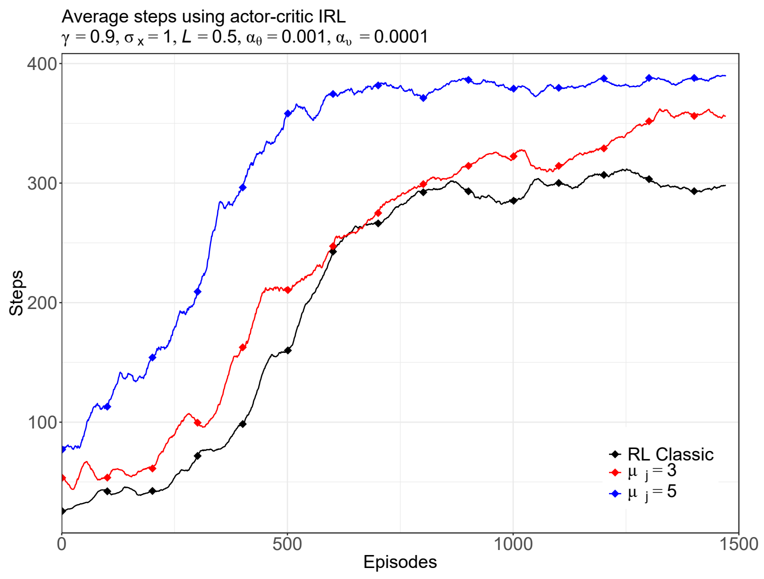
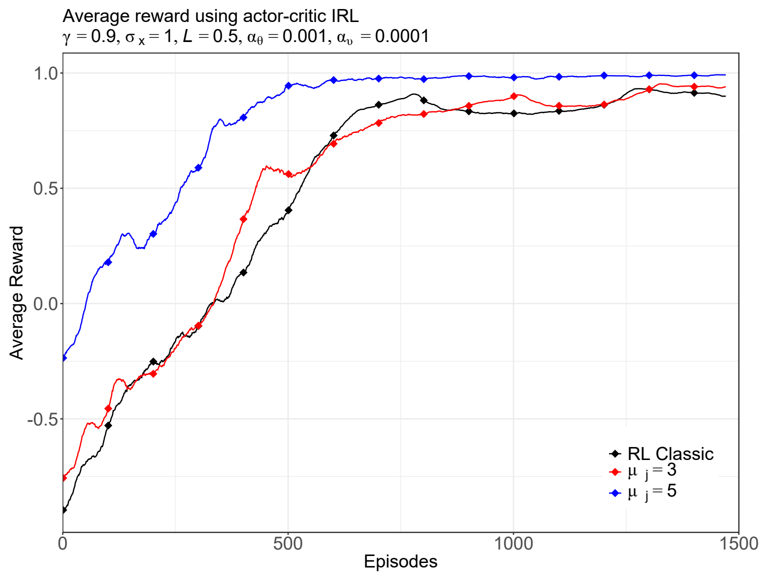


Figure 15 – Average collected reward over 15 using classic RL (black line) and IRL approach with different values of μ_J^*, probability of likelihood $L = 0.5$ and $\sigma_J = 1$. After 500 episodes, the agent collects more than 0.75 of reward with, however, with $\mu_J^* = 5$, it is reached to 1 of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



We investigate how the learning process is using RRL in a fixed environment. Fig. 16 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 [19] 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 16 – 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.

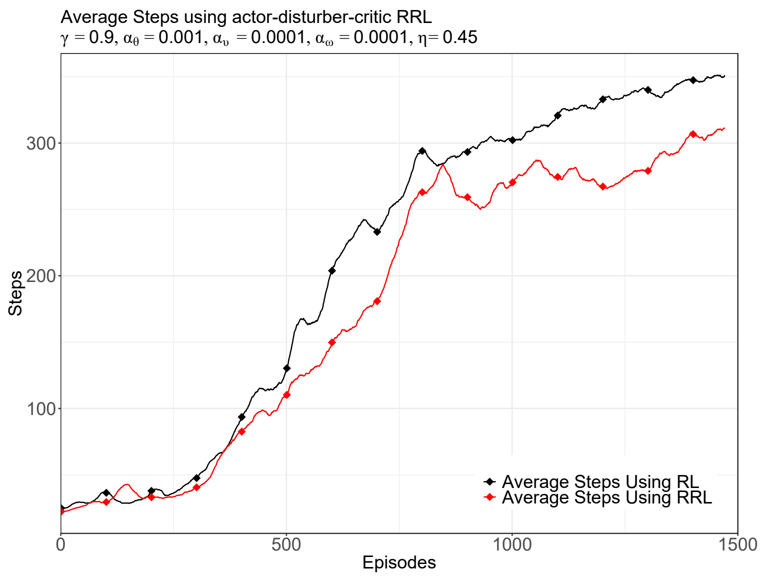
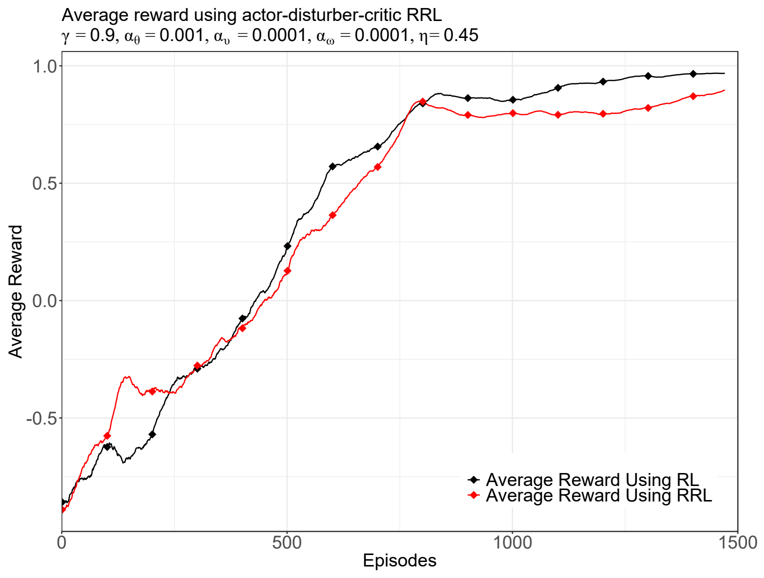


Figure 17 – Average collected reward over 15 runs using RL (black line) and RRL approach in a fixed environment with $\eta = 0.45$. Here, the augmented reward is not taken into account. The collected reward is higher than 0.75 after 700 episodes for the two methodologies.

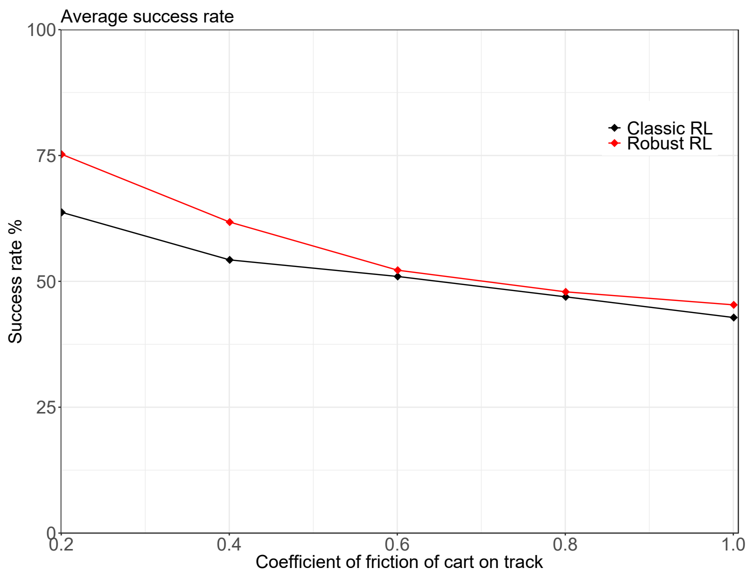


the task in less than 50 steps; after episode 500, the RL agent begins to perform better. Fig. 17 shows the average reward collected for this experiment, the reward illustrated is that obtained directly from the environment.

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.

Fig. 18 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.

Figure 18 – 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%.



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. 19 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. 20 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 19 – 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.

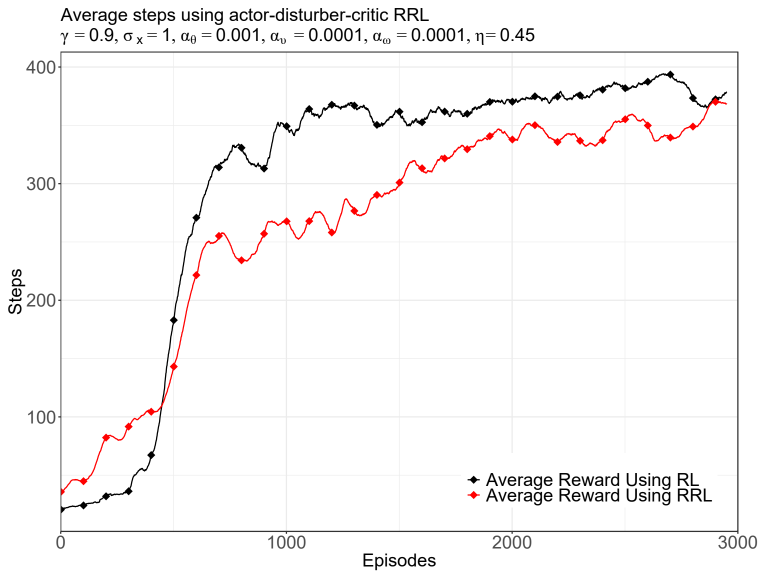
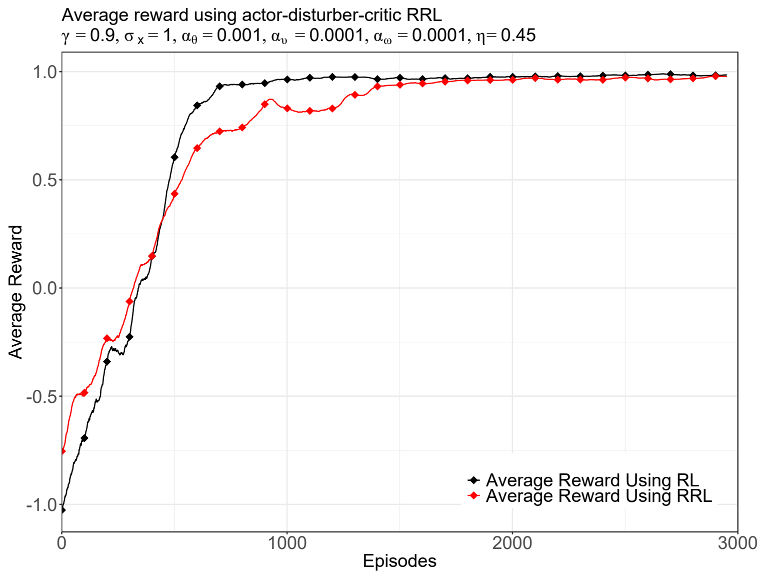


Figure 20 – Average collected reward over 15 runs using RL (black line) and RRL approach in a dynamic environment with $\eta = 0.45$. Here, the augmented reward is not taken into account. The collected reward is 1 after 1500 episodes for the two methodologies.



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 $\eta = 0.45$.

Fig. 21 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. 22 shows the average reward collected during learning. It should be noted that after 1500 episodes the agents get rewards close to 1.

Figure 21 – 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.

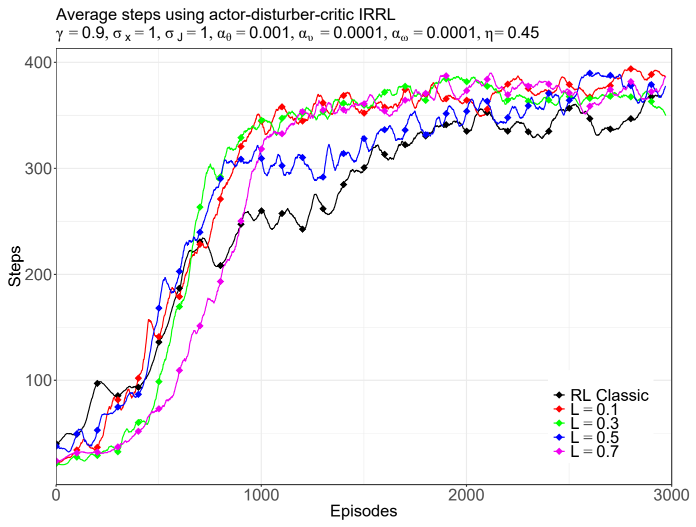
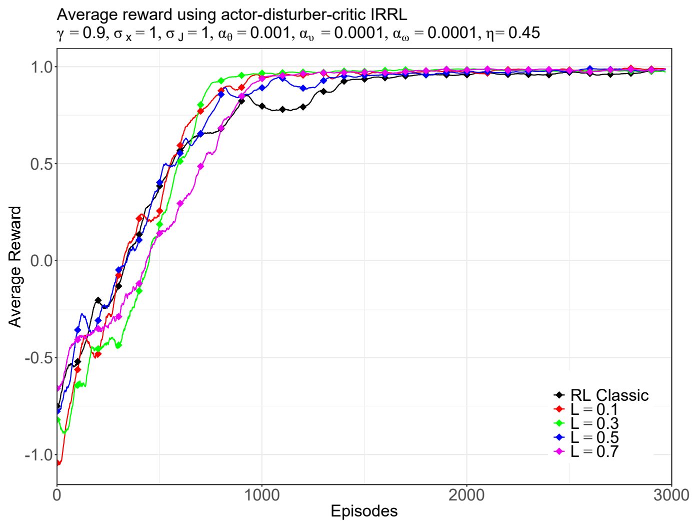


Figure 22 – Average collected reward over 15 runs using RRL (black line) and IRRL approach with different probability of likelihood L. After 1500 episodes, the collected reward is close to 1 for all curves.



Chapter 5

Conclusions

This thesis presents an approach 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 the agent to learn by comparing the number of steps and the reward collected over episodes. We individually evaluate each of the algorithms to have a comparative basis and be able to combine them at the end. Besides, different parameter settings are implemented to investigate the effect on learning. In the case of RL, learning depends on the value of the standard deviation of the distribution; this parameter assumes the role of exploration. Thus, the higher this value, the more actions are available around the value predicted by the neural network. However, exploitation is reduced, and learning is affected during the last episodes, approximately after 1000 episodes. It is necessary to choose a standard deviation value that merits the balance between exploration and exploitation.

In terms of IRL, we obtained better agent performance by giving advice compared to the classic RL. With our methodology, the agent can learn the task in fewer episodes, and the number of steps per episode reaches approximately the maximum (400 steps). We can note that with advice values $L = 0.5$ and $L = 0.7$, the performance agent is similar. Happening because the advice the agent receives is not very informative.

In terms of RRL, the agent requires a higher number of episodes to learn the task. However, the agent is more robust in dynamic environments and manages to perform the task facing environmental disturbances. Learning is indifferent to the dynamics of the environment. The reward obtained with RRL does not exceed that of the classical RL; however, this may be related to the same disturbance present in the reward (which depends on a disturbed state).

Finally, with IRRL in terms of average steps, our approach performs better than the autonomous RRL. However, the performance of the IRRL agent with probability $L = 0.5$ is close to that of the autonomous RRL agents. In terms of reward, we note that a cumulative reward of 1 is achieved for any probability L; however, values such as $L = 0.7$ have greater difficulty in the first learning episodes. This behavior is influenced by uninformative guidance, although the advice is correct concerning the space of actions that provide less information. Receiving much advice of this nature may not help in learning, even more, when the state is disturbed externally.

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 approach 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.; WUPPEN, P.; MAGG, S.; FAZRIE, A.; WERMTER, S. Agent-advising approaches in an interactive reinforcement learning scenario. In: *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, 2017. p. 209–214. ISBN 978-1-5386-3715-9. Disponível em: <http://ieeexplore.ieee.org/document/8329809/>.

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

[6] 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/>.

[7] 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>.

[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] 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>.

[11] DEARDEN, R.; RUSSELL, S. Bayesian Q-learning. In: *15th AAAI*. [S.l.: s.n.], 1998. p. 761–768.

[12] 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/>.

[13] SISBOT, E.; MARIN, L.; ALAMI, R.; SIMEON, T. A mobile robot that performs human acceptable motions. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006. p. 1811–1816. ISBN 1-4244-0258-1. Disponível em: <http://ieeexplore.ieee.org/document/4058640/>.

[14] 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>.

[15] 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.

[16] Van Hasselt, H.; 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.

[17] 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>.

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

[19] 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>.

[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] 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.

[22] 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.

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

[24] 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>.

[25] 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>.

[26] 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>.

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

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

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

[30] 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>.

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

[32] 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>.

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

[34] 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>.

[35] 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/>.

[36] 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.

[37] 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>.

[38] 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/>.

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

[40] 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/>.

[41] 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>.

[42] 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/>.

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

[44] 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/>.

[45] CRUZ, F.; MAGG, S.; NAGAI, Y.; WERMTER, S. Improving interactive reinforcement learning: What makes a good teacher? *Connection Science*, v. 30, n. 3, p. 306–325, jul 2018. ISSN 0954-0091. Disponível em: <https://www.tandfonline.com/doi/full/10.1080/09540091.2018.1443318>.

[46] PILARSKI, P.; SUTTON, R. Between Instruction and Reward: Human-Prompted Switching. In: *AAAI Fall Symposium: Robots Learning Interactively from Human Teachers*. [S.l.: s.n.], 2012. p. 46–52.

[47] 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/>.

[48] 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.

[49] 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.

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

[51] 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>.

[52] 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>.

[53] 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>.

[54] 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.

[55] 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/>.

[56] 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/>.

[57] 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.

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

[59] 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>.

[60] 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>.

[61] WAWRZYŃSKI, P. Real-time reinforcement learning by sequential Actor–Critics and experience replay. *Neural Networks*, v. 22, n. 10, p. 1484–1497, dec 2009. ISSN 08936080. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0893608009001026>.

ANNEX A

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

The cart-pole environment, implemented in *OpenAI Gym* toolkit [59], 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. [6]. This modification allows changing the value of the friction coefficient to alter the movement of the car and the pendulum.