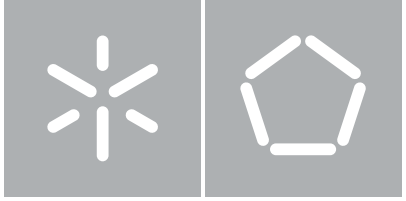
Universidade do Minho
Escola de Engenharia

Ana Filipa de Sampaio Calçada Duarte

Using Reinforcement Learning in the tuning of Central Pattern Generators

Outubro de 2012



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ana Filipa de Sampaio Calçada Duarte

Using Reinforcement Learning in the tuning of Central Pattern Generators

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor Cesar Analide Rodrigues
Professora Cristina Peixoto dos Santos

Outubro de 2012

"*What we have to learn to do, we learn by doing*" –

Aristotle, Nicomachean Ethics

Acknowledgments

This dissertation project was possible thanks to the several contributions that I have been received over time.

Firstly, and with due prominence, I thank my advisors: Professor Cesar Analide, for encouraging the choice of Artificial Intelligence area to develop my thesis work, for the advices, concerns and motivation; Professor Cristina Santos, for giving me the opportunity to address such an interesting work, for her frequent incentives and interest expressed throughout the entire project, for knowledge sharing and guidance.

I would like to thank Michel Tokic and Jan Peters for the immediate availability in answering my questions.

Also, I am grateful to my lab colleagues Vítor Matos, Pedro Silva and Miguel Campos for their help in several phases of the project, and the provision of resources used by them.

I want to thank my father for his wise advices, which greatly helped to make this a better work; to my mother, for her support, especially in the more stressful phases; and to my sister, for not being upset for the little time I have dedicated to her, due to the large amount of work that this dissertation has demanded.

I am thankful to Eduardo, for his support and his help provided throughout the entire project.

This work had the support of FEDER Funding supported by the Operational Program Competitive Factors - COMPETE and National Funding supported by the FCT - Portuguese Science Foundation through project PTDC/EEACRO/100655/2008 and FCOMP/01/0124/FEDER/022674.

Resumo

É objetivo deste trabalho aplicar técnicas de *Reinforcement Learning* em tarefas de aprendizagem e locomoção de robôs. *Reinforcement Learning* é uma técnica de aprendizagem útil no que diz respeito à locomoção de robôs, devido à ênfase que dá à interação direta entre o agente e o meio ambiente, e ao facto de não exigir supervisão ou modelos completos, ao contrário do que acontece nas abordagens clássicas. O objetivo desta técnica consiste na decisão das ações a tomar, de forma a maximizar uma recompensa cumulativa, tendo em conta o facto de que as decisões podem afetar não só as recompensas imediatas, como também as futuras.

Neste trabalho será apresentada a estrutura e funcionamento do *Reinforcement Learning* e a sua aplicação em *Central Pattern Generators*, com o objetivo de gerar locomoção adaptativa otimizada.

De forma a investigar e identificar os pontos fortes e capacidades do *Reinforcement Learning*, e para demonstrar de uma forma simples este tipo de algoritmos, foram implementados dois casos de estudo baseados no estado da arte. No que diz respeito ao objetivo principal desta tese, duas soluções diferentes foram abordadas: uma primeira baseada em métodos *Natural-Actor Critic*, e a segunda, em *Cross-Entropy Method*. Este último algoritmo provou ser capaz de lidar com a integração das duas abordagens propostas. As soluções de integração foram testadas e validadas com recurso ao simulador Webots e ao modelo do robô DARwIN-OP.

Palavras-chave: Inteligência Artificial; Aprendizagem; Reinforcement Learning; Central Pattern Generators; Locomoção Robótica; Otimização; Natural Actor-Critic; Cross-Entropy Method.

Abstract

In this work, it is intended to apply Reinforcement Learning techniques in tasks involving learning and robot locomotion. Reinforcement Learning is a very useful learning technique with regard to legged robot locomotion, due to its ability to provide direct interaction between the agent and the environment, and the fact of not requiring supervision or complete models, in contrast with other classic approaches. Its aim consists in making decisions about which actions to take so as to maximize a cumulative reward or reinforcement signal, taking into account the fact that the decisions may affect not only the immediate reward, but also the future ones.

In this work it will be studied and presented the Reinforcement Learning framework and its application in the tuning of Central Pattern Generators, with the aim of generating optimized robot locomotion.

In order to investigate the strengths and abilities of Reinforcement Learning, and to demonstrate in a simple way the learning process of such algorithms, two case studies were implemented based on the state-of-the-art. With regard to the main purpose of the thesis, two different solutions are addressed: a first one based on Natural-Actor Critic methods, and a second, based on the Cross-Entropy Method. This last algorithm was found to be very capable of handling with the integration of the two proposed approaches. The integration solutions were tested and validated resorting to Webots simulation and DARwIN-OP robot model.

Keywords: Artificial Intelligence; Machine Learning; Reinforcement Learning; Central Pattern Generators; Robot Locomotion; Natural Actor-Critic; Cross-Entropy Method; Optimization

Contents

List of Figures

List of Tables

List of Acronyms

ANN	Artificial Neural Networks
CAM-ES	Covariance Matrix Adaptation – Evolution Strategy
CEM	Cross Entropy Method
COP	Center of Pressure
CPG	Central Pattern Generator
DMP	Dynamical Movement Primitive
DOF	Degrees Of Freedom
DP	Dynamic Programming
GA	Genetic Algorithms
MDP	Markov Decision Process
NAC	Natural Actor-Critic
PI^2	Policy Improvement with Path Integrals
RL	Reinforcement Learning
RMS	Root Mean Square
TD	Temporal-Difference

List of Symbols

$\pi(s)$	Learning Policy
$\pi(s)^*$	Optimal Learning Policy
V(s)	State Value
V(s)*	Optimal State Value
Q(s,a)	State-action Value
γ	Discount Factor
ε	Exploration Rate
α	Learning Rate
$\boldsymbol{P_{ss'}^{a}}$	Probability of a transition from a state *s* to a state *s'*, caused by the execution of action *a*
$\boldsymbol{R_{ss'}^{a}}$	Reward acquired via a transition from a state *s* to a state *s'*, caused by the execution of action *a*
θ	Policy parameters
$N(\theta,\Sigma)$	Normal distribution with mean θ and covariance Σ
$\pi(u\|x) = p(u\|x,\theta)$	Parameterized policy with parameters θ, following a distribution $p(u\|x, \theta)$

Chapter 1

Introduction

The purpose of this thesis is to study learning techniques regarding to Reinforcement Learning (RL) for the tuning of Central Pattern Generators (CPGs) in the optimization of biped locomotion. This chapter presents the motivation which led to the development of this work and provides for the importance of such investigation in the context of Artificial Intelligence. It also depicts the major aims for achieving this particular task, as well as some of the issues that are desired to be answered with the development of this project.

1.1 - Motivation

Wheeled robot locomotion was for a long time subject of study. However, it has proved to not be adapted to several environments. So, recent contributions emerged to the development of walking robots, which revealed numerous advantages over wheeled robots. Specifically, they are able to navigate in uneven terrains, to overcome or avoid obstacles and holes/ditches, to climb steps, and to better balance on unstructured and inclined terrains (Vítor Matos 2009). Such locomotion mechanisms are often inspired by biological systems, which means that robots are brought closer to real living beings (animal or human).

Legged robots involve the coordination of a high number of degrees-of-freedom (DOF) and parameters, and therefore, the learning and control of their movements, in real time, are assumed as very complex problems. Other difficulties are related to the body balancing in order to support the robot and not letting it to tip over. The design of a suitable controller is therefore not trivial, and the generation of autonomous, flexible

and adaptive locomotion is a very challenging problem that is still subject of current studies. This makes legged locomotion one of the most important and hardest control problems.

Learning plays a major role in this subject, such as providing gait generation and robot stability, and the capability of autonomously decide when and how to move. In robotics, the ability to adequately respond to a certain stimulus taking into account a goal, poses one of the greatest challenges on the field of autonomous systems. Such capability can be achieved by means of a relationship between the perceived information, collected by the robot about the surrounding environment, and the motor response that it is supposed to exhibit.

In this sort of decision-making and control problems, it is very difficult to provide explicit supervision to a learning algorithm, since it is impossible to gather a training set, capable of covering all feasible cases. The provision of a complete model of the environment, in these cases, is not a practicable alternative, since there are always plenty of unknown, unpredictable and/or stochastic components in the ambience. For these grounds, it is necessary to take full advantage of the interaction between the agent and its environment, to cope with problems such as balance and steering, gait switching, terrain changing, and any other type of adaptation to perturbations in the surrounding environment.

Considering the above, it is necessary an Artificial Intelligence approach capable of: mapping any situation (stimulus) to an action (motor reaction); providing all the advantages of interaction; and overcoming the fact of not existing elements of supervision. Thus, Reinforcement Learning will be indicated to address these problems, since it is very different from other classic approaches, where it is required some supervision or environment models. In this case, the feedback provided constitutes merely in the information if the robot is doing well or poorly (rewards). Learning to act in ways that are rewarded or punished reveals intelligence, and it is broadly how humans and animals gain knowledge and experience to apply in the most numerous situations.

Apart from locomotion, Reinforcement Learning can be used for several applications, and provides a very important component in any decision making: the possibility of this being sequential, if an action entails future consequences (more planning). This constitutes one of the aspects that contribute to the great complexity of this technique, when compared to others. The work (S. J. Russell and Norvig 1995)

provides further details on these sequential decision problems in contrast to single decision problems.

Reinforcement Learning is therefore a computational approach for understanding and automating goal-oriented learning and decision making (Sutton & Barto, 1998). Learning is accomplished through interaction with the environment and involves deciding which actions to take so as to maximize a reward signal, and such actions can impact the long-term.

1.2 - Problem Statement

"Adaptative System Behaviour Group" is a work group at University of Minho, whose main research focus are the new challenges that arise from the development of robotic and computer technologies embedded in dynamical environments and situations. This work group developed a bio-inspired architecture that applies autonomous differential equations to model the manner how behaviors related to locomotion are programmed in the oscillatory feedback systems of Central Pattern Generators in the nervous systems.

One of the disadvantages of CPG driven methods is that it is not trivial to determine appropriate oscillators parameter settings in order to achieve a stable gait or a desired movement. In this work, it will be addressed the problem of real time adjustment of CPGs, in order to generate biped locomotion. The robot must be able to learn to improve its own performance in the proposed motor tasks, through what is perceived about the environment.

Despite the difficulty of tuning the CPGs in order to execute the adequate motor movements, a basis locomotion pattern is already available. But *will the chosen parameters be the more suitable for the task at hand*? In addition to this, there are some other issues that need to be answered at the end of this project:

Are the available methods for tuning CPGs parameters suitable for this particular CPG architecture? Are the selected approaches capable of optimizing the basic movement and thus achieving the designed goals? And is that optimization worthwhile when examining the robot's evolution? Is this conceived integrated approach a good line of attack for the task that it is intended to achieve?

Along with these CPG-related issues, it is also intended to evaluate RL itself. So, *is RL capable of successfully obtaining the desired results?*

1.3 - Objectives

The major aim of this work is then to tackle the problem of tuning the Central Pattern Generators (CPGs) parameters in order to achieve the desired locomotion, imlementing Reinforcement Learning as an automatic learning technique of those parameters.

Thus, and in order to pursue this purpose, the following intermediate goals have to be achieved:

- To gather the state-of-the-art considering the problem of optimization and learning techniques regarding to Reinforcement Learning, mainly addressing the subject of robotic locomotion;

- To understand the RL framework and functioning;

- To study and implement experiments based on the state-of-the-art, in order to better understand RL and to gain experience for addressing the following objectives;

- To study the CPGs implemented and to gather the state-of-the-art regarding the use of integration techniques of the two approaches;

- To idealize a RL solution for the humanoid locomotion problem, considering the use of CPGs;

- To implement the solution in Webots simulation (Michel 2004), evaluating the achieved results.

1.4 - Thesis Outline

The following chapter, Chapter 2, will show the work resulting from the research phase, which was mainly focused in the RL application to robotic locomotion. The CPGs will be described and briefly discussed at the end of the chapter, along with a state-of-the-art review.

Chapter 3 describes the Reinforcement Learning problem, including an overview of its framework. The functioning of Markov Decision Processes (MDPs) will be addressed as a method of solving RL problems, as well as some available techniques based on the mentioned framework. RL advantages, bottlenecks and challenges will be also exposed throughout the chapter.

Then, Chapter 4 will present two experiments studied and implemented, based on the state-of-the-art, and which are ideally suited for simple demonstrations of RL

algorithms. The Crawler robot and the Inverted Pendulum problem are exposed and the results are analyzed and discussed.

Chapter 5 deals with the first set of experiments carried out in order to tackle the main objective. The used CPGs are readily exposed and Natural Actor-Critic is presented as the method used in the development of the solution.

In Chapter 6, it is discussed an alternative method to solve Reinforcement Learning problems, which is curiously very similar to some of the classic RL methods. The Cross-Entropy Method is then introduced and a solution is designed for the optimization of biped locomotion with CPGs.

Last of all, the conclusions are presented in Chapter 7, including an evaluation and introspection of all the work carried out. Final reviews and considerations are made regarding the proposed goals, and some challenges are anticipated for future work.

1.5 - Contributions

The work carried out in the first phase of the thesis, regarding the case studies based on the state-of-the-art, led to a publication in International Conference of Numerical Analysis and Applied Mathematics (ICNAAM) conference (Duarte, Silva, and Santos 2012). This conference was held in Greece, in September 2012, covering numerous topics on Mathematics, including Optimization.

A crawler robot model was developed from scratch, which may be helpful for several other future experiments.

Another contribution of this study is the application of a solution that was not encountered in the literature for similar aims of this thesis, e.g., robot locomotion using CPGs. This solution resulted in an efficient optimization, as will be shown in 6.3.

Chapter 2

State of the art

The previous chapter focused on the importance and motivation for using RL as the learning technique to apply to robot locomotion. This chapter, the state-of-the-art gathered during the development of the thesis is exposed, where further details of the history of Reinforcement Learning are introduced. Although RL is useful in many machine learning domains, the investigation carried out is mainly focused in RL application in the field of Robotics, particularly, in locomotion. CPGs are then introduced as biologically inspired methodologies for providing rhythmic movements that can be optimized by learning algorithms.

2.1 - General Reinforcement Learning

Learning and Reinforcement have been first studied in psychology, and those researches had a very strong impact in the area of Artificial Intelligence. Trial-and-error is one of these examples, started in psychology of animal learning, where actions followed by good outcomes have their tendency to be reelected by the animal, rather than the ones followed by bad outcomes, as stated in (Sutton and Barto 1998).

The term "Reinforcement Learning" first appeared in (Minsky 1961). But previously, it had already been proposed and implemented a learning method with temporal-difference ideas to manage delayed reward, in Samuel's checkers player (Samuel 1959).

In 1972, it was brought trial-and-error learning together with an important component of temporal-difference learning (Klopf 1972). This work and animal learning theories strongly influenced the followed research on temporal-difference

learning. It was then developed a method for using temporal-difference learning in trial-and error learning, known as the actor-critic architecture (A. G. Barto, R. S. Sutton, and C. W. Anderson 1983). In 1988, it was introduced the TD(λ) algorithm, and proved its convergence properties (Sutton 1988).

Temporal-difference learning was brought together with optimal control in 1989, with the introduction of Q-learning algorithm (C. J. C. H. Watkins 1989). With this approach, it was developed a backgammon player, TD-Gammon (Tesauro 1992). Tabular TD(0) was previously proposed for use as part of an adaptive controller for solving MDPs in (Witten 1977).

Although there has been a recently development in this area, as it will be verified throughout this section, RL makes use of earlier frameworks. MDPs were known at least as early as the 1950s (Bellman 1957). In 1994, Putterman studied the advances in Markov Decision Processes theories and applications, to provide an "up-to-date, unified and rigorous treatment of theoretical, computational and applied research" (Puterman 1994). There are also approaches to non-Markov environments (Whitehead and Lin 1992).

An useful survey was provided by Kaelbling, Littman, and Moore, with a general coverage of Reinforcement Learning problems (Kaelbling, Littman, and Moore 1996).

Recently, it has been explored RL algorithms to resolve the problem of continuous tasks (Ravindran 1996; Smart and Kaelbling 2000). Q-learning was adapted to adequately address problems with both continuous states and actions, which makes use of Artificial Neural Networks (ANN) (Gaskett, Wettergreen, and Zelinsky 1999). In 2003, a biologically-based approach for solving continuous state and action problems was proposed in (Strosslin and Gerstner 2003), consisting in a spatial representation to represent the state space. A new class of algorithms, named "Continuous Actor Critic Learning Automaton" (CACLA) was presented in (H. van Hasselt and Wiering 2007) to handle continuous actions and states, and showed better performance than other existing methods.

A very complete survey of continuous RL problems can be found in (H. V. Hasselt 2012), where the author covers methods like function approximation, policy gradient, evolutionary policy search or actor-critic algorithms.

Integrated techniques of RL and evolutionary methods are also good approaches which become increasingly used for solving large RL problems. The application of evolutionary algorithms to Reinforcement Learning is presented in (Moriarty, Schultz,

and Grefenstette 1999), where alternative policy representations are emphasized. An integrated technique of RL and Genetic Programming that allows a robot to execute real-time learning and to carry an object to a goal area is proposed in (Kamio, Hideyuki Mitsuhashi, and Iba 2003). Q-learning and Genetic Algorithms were combined to introduce a novel algorithm which has shown great effectiveness in the acquisition of locomotion patterns (Ito and Matsuno 2002).

Apart from evolutionary computation, RL has been combined with several different techniques. Tabu Search, a search method for mathematical optimization, was used to address the problem of exploring solutions in on-policy RL problems, without getting stuck in a local optimum (Abramson and Wechsler 2003). Particle filters, also known as Sequential Monte Carlo Methods, were recently integrated with RL for creating a novel algorithm for direct global policy search, capable of finding the globally optimal policy (Kormushev and Caldwell 2012).

Inverse RL, that is, the problem of extracting a reward function given the optimal behavior, also had its prominence. In (A. Y. Ng and S. Russell 2000) it is overviewed the algorithms used in this approach.

In (Wang and Laird 2007), it is investigated the importance of action history, namely, the hypothesis that historical information plays an important role in learning action selection via reinforcement learning. Other variants to the RL problem have emerged, such as the using of decision trees to learn a model, on a humanoid robot (Hester, Quinlan, and Peter Stone 2010).

(Niekum, Andrew G Barto, and Spector 2010) shows an alternative way of finding reward functions. Namely, it is used genetic programming to find novel reward functions that allow systems to learn more quickly or more effectively. Multiple reward functions were also a research target (Lizotte, Bowling, and Murphy 2010).

A new method for balancing exploration/exploitation was proposed in (Tokic 2010), where the exploration parameter of ε-greedy is adapted with regard to the temporal-difference error observed from value function backups. This method revealed to be more parameter robust than, for example, the original ε-greedy.

2.2 - Applications and Robotics

RL techniques are widely applied, in the most varied applications and areas, as seen in many different works, i.e., robotics, animal learning, scheduling, games, etc.

In 1995, RL was applied to the task of elevator dispatching, using a team of RL agents, each of which responsible for controlling one single elevator. This approach resorts to ANNs to store information (Crites and Andrew G. Barto 1996).

Reinforcement Learning was successfully applied in a job-shop scheduling problem, in the context of scheduling payload processing for NASA's space shuttle program (Zhang and Dietterich 1995). The problem of channel assignment in cellular telephone systems was also addressed as a RL problem in (Singh and Bertsekas 1997), in order to maximize service in a stochastic caller environment. The results are presented on a large cellular system with approximately 49^{49} states.

The task of exploring the web to find pages of a particular topic can be efficiently solved by a web spider which learns with resort to Reinforcement Learning techniques (Rennie and McCallum 1999).

Autonomous helicopter flight, a challenging control problem, was tackled in several works. In (Bagnell and Schneider 2001), it is addressed a successful application to this problem, by means of policy search methods.

Focusing straight in robotics applications, it has been done several experiments on mobile robots. In 1991, it was discussed a way of pushing large boxes by a mobile robot, with resort to Q-learning algorithm (Mahadevan and Connel 1991). In (Mataric 1994), four mobile robots learn how to most efficiently search, find and collect small disks to transport to a destination region, by grasping and lifting them. Learning to drive a bicycle was also a real-world problem addressed with Reinforcement Learning, applying SARSA(λ) algorithm (Randløv and Alstrøm 1998).

RL can be used to learn just simple tasks like following a corridor or avoiding obstacles (Smart and Kaelbling 2002) or to learn to appropriately place a swing leg, regarding to biped walking (Morimoto et al. 2004). In (Kohl and P. Stone 2004), it was presented a policy gradient approach for automatically learning a fast walk on the quadruped robot Aibo, given a parameterized walk, which achieved an efficient policy evaluation.

The problem of learning how to intercept a ball, regarding to soccer player robots, was addressed in terms of reward and punishment (Muller et al. 2007).

2.3 - Central Pattern Generators

In animals' locomotion mechanism, rhythmic motor patterns are controlled by neural oscillators, which are known as the Central Pattern Generators. These biological studies provided a starting point for developing locomotion robots controlled by CPGs controllers. For a more complete overview of this methodology, Ijspeert reviews research carried out on CPGs, both on animals and robots (Auke Jan Ijspeert 2008). This mentioned work also shows that CPG models, which are biologically inspired, are increasingly used in robotics field, for controlling a variety of different types of robots with different types of locomotion. Moreover, "an increasing number of projects try to provide something back, i.e. to specifically use robots as scientific tools to test biological models" (Auke Jan Ijspeert 2008).

CPGs have been used in robots inspired by insect, biped, quadruped or even swimming locomotion. For example, in (Klaassen et al. 2002), it is approached a control scheme for a 8-leg scorpion robot. In (Arena 2001), CPGs were used to generate proper swimming patterns for a swimming machine, where the robot reproduces the undulatory-like swimming of a sea-lamprey. With a similar aim, an amphibious snake robot was designed to achieve not only these swimming patterns, but also crawling or serpentine locomotion (Crespi and Auke Jan Ijspeert 2006). Also capable of these types of locomotion is the fish robot presented in (Lachat, Crespi, and Auke Jan Ijspeert 2006), where the robot performs a variety of locomotor behaviors, such as swimming forwards, swimming backwards, turning, rolling, moving upwards/downwards, and crawling.

Quadruped walking control has also been a target for the use of CPGs. Inducing a robot to walk on irregular terrains (Fukuoka, Kimura, and Cohen 2003) or generating two different types of gaits (walk and pace), testing stability and improving forward velocity (Rutishauser et al. 2008), were some of the aims approached to achieve this type of locomotion.

For controlling biped robots, CPGs are widely used. For example, to control the humanoid robot HOAP-1 with many DOFs, walking locomotion is approached in (Shan and Nagashima 2002), where the robot even learns to walk up and down stairs. In (Jun Nakanishi et al. 2004), a natural-human like locomotion is the proposed goal, which is achieved by learning from demonstration. The biped robot learns demonstrated trajectories through CPGs by locally weighted regression.

Meanwhile, the work group at the University of Minho, "Adaptative System Behaviour Group" developed a bio-inspired architecture that applies autonomous differential equations to model the manner how behaviors related to locomotion are programmed in the oscillatory feedback systems of Central Pattern Generators in the nervous systems (Matos, Santos, and Pinto 2009; Matos and Santos 2012; Santos 2004). In (Vítor Matos 2009), it was developed a CPG network that appropriately generates omnidirectional locomotion for quadruped robots.

It is important to take notice that one disadvantage of CPG driven methods is that it is not trivial to determine appropriate oscillators parameter settings in order to achieve a stable gait or a desired movement. So, many works reveal the necessity to address the problem of real time adjustment of CPGs, in order to generate optimized locomotion. Specifically, the robot must be able to learn to improve its own performance in several motor tasks, through what is perceived about the environment.

Therefore, works presenting RL methods for CPGs controllers have emerged, allowing the achievement of humanoid locomotion. The earliest references to this integration seems to refer back to 1997 (Benbrahim and Franklin 1997), where the CPG acts as a central controller and interacts with some peripheral controllers that intervene when their own control policies are contradicted, thus requiring knowledge integration. The central controller, as well as some of the other controllers, uses CMAC neural networks to represent the CPG, whose weights are updated based on the reinforcement signals received from every controller. The configuration taken in this work is based on the actor-critic, and it is also used algorithms like the Self-Scaling Reinforcement and the Stochastic Real-Valued.

Also making use of Neural Networks, there are other approaches integrating CPGs with RL. In (M. Sato, Y. Nakamura, and S. Ishii 2002), a Recurrent Neural Network is representing the CPG, and a new method is proposed, based on the actor-critic configuration: the CPG-actor-critic. To complement this method, it is also used a Gaussian neural network with EM algorithm to approximate the Q-function. Variations of this work include the implementation of Policy gradient methods along with the CPG-actor-critic (Mori et al. 2004; Nakamura et al. 2007), whose task consists in adjusting the sensory feedback connections, or adjusting a feedback and connection controllers concerning to quadruped locomotion (Sato, Watanabe, and Igarashi 2010). Improved policy gradient methods, like the Off-policy Natural Policy Gradient, allowed the resolution of exploration/exploitation problem and the achievement of better results.

This method was also used together with CPG-actor-critic, to enable the adjustment of CPG connections (Nakamura et al. 2005).

Another way of representing the CPG controller consists of using a neural oscillator model proposed by Matsuoka (Matsuoka 1985). Although its emergence dates back to some time ago, it is still used in the present. A common approach to this representation lies on the use of Policy gradient methods, aforementioned (Matsubara et al. 2006, 2007).

There were other methods that studied the integration between RL and CPGs, but with the drawback of using discretization methods, avoiding the problem of directly tackling continuous spaces of states/actions (Jacob, Polani, and Nehaniv 2005; Ogino et al. 2004). (Jacob et al. 2005) diverges even more, since it concerns to quadruped robots, in which the legs are individually trained and then re-attached to the robot. Also diverging from humanoid locomotion is the controlling of a hexapod robot, using a Recurrent Neural Network to represent the CPG, and a TD-learning approach to decide about direction changes (Snel, Whiteson, and Kuniyoshi 2011).

Other interesting works have emerged, not only regarding to the actual movement and balancing, but also with other aspects in the locomotion problem. For example, it is important to reduce energy consumption when generating locomotion. This problem can be approached by introducing a torque-free period, where no torque is applied. Using GARB algorithm to determine when this period is applied, it was achieved a reduction of 40% of the energy (Tomoyuki, Azuma, and Tomoshiro Shibata 2009).

Apart from locomotion, the development of other useful robotic skills has also been approached. (Ciancio et al. 2011) proposes a computational bio-inspired model to investigate the development of functional rhythmic hand skills, i.e., the rotation of bottle cap-like objects. The model is based on an actor-critic model that searches the parameters of a set of CPGs. This specific work was found to be valuable for the study of the development of rhythmic manipulation skills in primates.

Despite the use of RL techniques for tackling locomotion problems, all these CPGs approaches are very different from the architecture of differential equations developed by the “Adaptative System Behaviour Group”, which will be presented in 5.2.

Dynamical Movement Primitives (DMPs) have a direct relation with CPGs, as they combine a set of differential equations to reflect a certain rhythmic movement (Ijspeert, J. Nakanishi, and S. Schaal 2002). They provide a general approach to motor control in

robotics and biology, being very useful due to their high flexibility in creating complex rhythmic and discrete behaviors, which can be quickly adapted to environment changes and perturbations (Stefan Schaal 2003).

Three different algorithms have been found to address this RL continuous problems regarding to DMPs: Natural Actor-Critic (NAC) (Peters, Vijayakumar, and Schaal 2003), Policy learning by Weighting Exploration with the Returns (PoWER) (Kober and Jan Peters 2008) and Policy Improvement with Path Integrals (PI^2) (E. Theodorou, J. Buchli, and S. Schaal 2010).

Natural Actor-Critic showed efficiency in motor tasks performed by a robot arm, for example, playing baseball, where the goal is hitting the ball so that it flies as far as possible (J. Peters and S. Schaal 2006).

Through a modified version of DMPs, robot motor skills are demonstrated with recourse to an Expectation-Maximization based Reinforcement Learning, known as PoWER (Kormushev, Calinon, and Caldwell 2010). In this work, the robot had to perform a reaching task, where the learned movement required an adaptation in order to avoid obstacles, and a dynamic pancake-flipping task. These two tasks used imitation learning as an initialization phase, in order to allow RL to explore for better solutions. Also making use of PoWER, several other motor skills can be learned on a real robot, i.e., ball-in-a-cup, consisting in moving a small cup to catch the ball that is attached to it, and ball-padding, having a ball bouncing above a tennis paddle (Kober and Jan Peters 2009).

A learning experiment on a robot dog illustrates the functionality of PI^2 in a real-world scenario, demonstrating its efficiency and robustness (Evangelos Theodorou, Jonas Buchli, and Stefan Schaal 2010). A hierarchical RL approach using algorithm PI^2 to sequences of DMPS was proposed in (Stulp and Stefan Schaal 2011), where a 11-DOF arm and hand learns a pick-and-place task, grasping an object from a shelf and placing it on another shelve in the same cupboard.

In the context of humanoid robots, (Stulp et al. 2010) shows two tasks which were addressed with PI^2. A first task consists in a robot passing through a way-point with its right hand, and returning to the initial standing position without falling, while in the second task the robot learns to open a door.

It is important to be aware of other alternatives for the problem of robot locomotion. Genetic Algorithms is widely applied in robotics field. As an example, in

1999, sony quadruped robot was object of study and evolutionary algorithms have contributed for the improvement of different gaits, like pace or trot (Hornby et al. 1999). Also for a sony quadruped robot (AIBO), a modified version of Genetic Algorithms is presented in (Chernova and Veloso 2004) to autonomously optimize fast forward gaits. The author's approach revealed to be very efficient, improving 20% over the best previous hand-tuned behaviors.

A different method for robotic locomotion is based on human's gait pattern analysis, which can easily be applied to generate the natural and stable gait pattern of any biped robot (Ha, Han, and Hahn 2007). Genetic Algorithms (GA) is used to approximate the biped locomotion to the desired human-like trajectory. (Picado 2008) discusses the development of some trajectory planning methods for biped locomotion, where CPGs are also an approach addressed. Other ways of generating trajectories are showed and robot walking is optimized by the application of GA. This machine learning technique, combined with a method based on partial Fourier series for joint trajectory planning, provides automatic generation of a walking gait (Picado et al. 2009).

Alternative learning techniques have also been addressed, i.e., the design of a controller for gait balancing based on a back-propagation ANN (Shieh et al. 2007), or the application of Support Vector Machines in humanoid robotics (Branislav, Mirko, and Milutin 2012). Imitation learning, which consists in learning and recognizing specific demonstrated movements to later perform them, can provide a basis for further optimization of robotic motions (S. Schaal 1999).

2.4 - Summary

This chapter showed the several improvements arising since the concept of Reinforcement Learning first appeared. A state-of-the-art review, which was proposed as a goal in 1.3, was accomplished in this phase, in which works of major importance to RL evolution were exposed. It is realized that new methods and approaches are always being investigated, in order to tackle RL problems in a wide range of applications.

It must be noticed that this chapter corresponds to the work developed through all stages of this thesis project. And although many works had arisen for the purpose of generating locomotion, this topic is still a subject of study in the present, and it was not

found a specific solution for the problem at hand, namely, the integration of RL and CPGs similar to the ones developed by the “Adaptative System Behaviour Group".

Chapter 3

Reinforcement Learning

Reinforcement Learning functioning and related concepts are exposed in the present chapter. A framework widely used for RL problems is the Markov Decision Processes, which will be presented here. The objective of this thesis stage is to comprehend the whole interaction process inherent to these learning problems, and how the learning algorithms act before certain stimuli. As continuous cases are very common in the real world, RL methods for solving such problems are also presented.

3.1 - The Reinforcement Learning Framework

Reinforcement Learning is a Machine Learning approach. In turn, Machine Learning is a field of Artificial Intelligence "concerned with the question of how to construct programs that automatically improve with experience" (Mitchell 1997). From data-mining programs to detect fraud or learn preferences in a recommender system, to autonomous robots that learn how to react facing a certain situation, successful machine learning applications have been developed in recent years.

An interesting definition of learning on the part of machines is introduced in (Mitchell 1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.".

There are many approaches to allow this learning from experience. Genetic Algorithms (GA), Artificial Neural Networks (ANN), Support Vector Machines (SVM), or Reinforcement Learning, are widely known techniques to address machine

learning problems. These and other machine learning algorithms are further detailed in (Mitchell 1997; Sammut and Webb 2011).

Machine learning algorithms are commonly organized in two major categories of learning. Supervised Learning refers to a learning process capable of mapping an input to a corresponding output, with recourse to examples with both input and output values, or to an environment model. Unsupervised learning refers to a learning process that occurs in the absence of known examples or environment models, where there is no hint about the correct outputs.

An additional learning category has been added to these two described. In some works in the literature (S. J. Russell and Norvig 1995; Sammut and Webb 2011), Reinforcement Learning is considered as a third type of learning, making the bridge between supervised and unsupervised learning. Unlike in supervised learning problems, now there are no examples of what should be the agent behavior. However, unlike unsupervised learning problems, the agent receives some evaluation, but without being told about the correct actions.

"Reinforcement learning is an approach to Artificial Intelligence that emphasizes learning by the individual from its interaction with its environment. This contrasts with classical approaches to artificial intelligence and machine learning, which have downplayed learning from interaction, focusing instead on learning from a knowledgeable teacher, or on reasoning from a complete model of the environment" (Sutton, 1999). This learning by interaction is constantly present in our lives and experiences, since childhood: learning how to walk or talk, or simply to learn what every single item around us is for.

An agent that learns by means of Reinforcement Learning must be able to sense the state of environment and take actions according to the information gathered, must have goals, and must be able to learn from its own experience and its failures/successes.

RL allows the agent to learn what to do depending on the circumstances in which it is inserted, mapping situations to actions, and always taking its goals into account. And these mapping has to be discovered by means of trial-and-error, favoring the actions that yield results that most approach the achievement of the agents' goals. Nevertheless, trial-and-error can only be useful if the agent can be aware if the actions performed are being correctly or incorrectly chosen. So, the concept of *reward* is the most important in Reinforcement Learning problems, allowing an evaluation of each executed action.

In many cases, actions may affect the rewards of the next actions, and not only the immediate one, which makes it a very challenging topic in RL area. So, it is highlighted two important characteristics regarding to this kind of learning: *trial-and-error search*, and *delayed reward* (Sutton & Barto, 1998).

One of the most challenging issues in Reinforcement Learning problems is the tradeoff between exploration and exploitation. It is important that the agent chooses the actions that yield the maximum reward, so as to reach its goal of maximizing the total accumulated reward (exploitation). However, it is also important to try to perform actions not selected before, so as to discover those best actions (exploration). Neither exploration nor exploitation can be used exclusively, so there must be a compromise between these two approaches.

The role of exploration is further detailed in (Thrun 1992), where several techniques are described and evaluated for exploration in finite and discrete RL domains.

3.2 - Markov Decision Processes

As mentioned above, the information gathered by the agent includes the state of the environment. At each time step, it is perceived the current state of the environment, from a set of all possible states. The agent takes an action, from a set of all possible actions, that causes a transition to some successor state, and then the achievement of a reward (Figure 3.1). This procedure recalls to the Markov Decision Processes (MDPs), a framework widely used in RL research. "MDPs provide a simple, precise, general, and relatively neutral way of talking about a learning or planning agent interacting with its environment to achieve a goal" (Sutton, 1997).

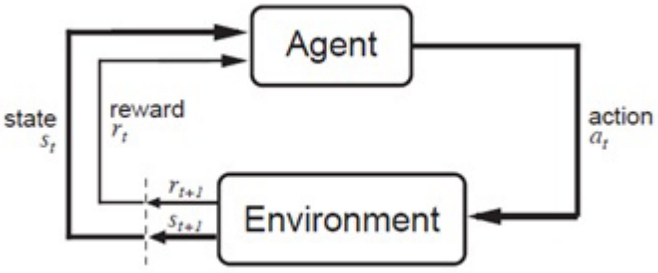


Figure 3.1 – The Reinforcement Learning Framework (Sutton 1999)

In order to a RL process be called a MDP, it must take place in an environment that fulfills the Markov property. To accomplish this, it is required that the probability to transit to a certain state and the reward received, solely depend on the current state and the action selected by the agent. Moreover, if the sets of actions and states are finite, the RL process is called a finite MDP.

So, in addition to the set of spaces, the set of actions, and the reward function ($R^a_{ss'}$) a MDP also is defined by a discount factor (γ) measuring the influence of future rewards; and the state transition probabilities ($P^a_{ss'}$) which means the probability of transit to a certain successor state s', being in a state s, and performing action a. To synthetize, a Markov Decision Process is a 5-tuple (S, A, $P^a_{ss'}$, γ, $R^a_{ss'}$).

In order to be able to find a cycle of moves for the desired locomotion, it has to be taken into account the notion of *policy*, π(s). A policy is a mapping from states to actions, and dictates which action to take when it is perceived a certain state s. An optimal policy is intended to be found by the RL algorithms, for instance, Value Iteration or Policy Iteration, among others. Since an optimal policy reflects the actions to be taken for maximizing the accumulated reward, it is required a means to represent these estimates. So, these algorithms assign to each state a value, V(s), which represents the expected total reward of starting from that state and respecting a given policy. These state values (Eq. 3.1) and the respective greedy policy (Eq. 3.2) are calculated according to Bellman's equations:

$$V(s) = max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')] \quad (3.1)$$

$$\pi(s) = argmax_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')] \quad (3.2)$$

After the successive calculation of state values with recourse to the greediest actions or to the chosen policy, the algorithms end up converging.

If separate averages are kept for each action taken in a state, it is required a different notation. The expected total reward of each state, previously represented by V(s), is now represented by Q(s,a) which symbolizes the q-values. Instead of representing the value of a state, the q-values Q(s,a) denote the value of taking action a in state s. A very important algorithm making use of such notation is Q-learning, which allows comparing the expected return of the available actions, by using Eq. 3.3.

An additional learning rate α is included in the calculation of q-values, informing about the level of sensitivity of the reaction to rewards. Namely, it denotes to what

extent the new information or learned value ($r + \gamma max_{a_{t+1}} Q(s', a_{t+1})$) will overwrite the old ($Q(s, a_t)$), causing q-values to experience minor or abrupt changes. If the learning rate is assigned to its maximum value, 1, the old information will be completely overwritten.

The respective policy can be calculating according to Eq. 3.4.

$$Q(s, a_t) = Q(s, a_t) + \alpha\left[r + \gamma max_{a_{t+1}} Q(s', a_{t+1}) - Q(s, a_t)\right] \quad (3.3)$$

$$\pi(s) = argmax_a(Q(s, a)) \quad (3.4)$$

Several different approaches are used to deal with Markov Decision Processes and try to find out the optimal policy. RL algorithms can be classified into model-based or model-free methods. The former assume prior knowledge about the environment, requiring beforehand a transition probability function $P_{ss'}^a$ and the reward table, containing the information of reinforcement signals for every state and action. If these data is not known, the system needs to acquire them before applying such methods, which can be hard to learn or represent. Contrariwise, model-free techniques have to learn in an unknown world since they do not assume prior knowledge, thus requiring more experience to achieve better value estimates. There is even a hybrid approach, which only requires an approximate model and which returns a near-optimal policy, capable of outperforming a model-based approach (Abbeel, Quigley, and Andrew Y. Ng 2006).

Dynamic Programming (DP) addresses MDPs in a deterministic fashion, requiring a full model of the environment. DP is computationally heavy in terms of memory and complexity, being applicable only to smaller problems.

Besides the RL methods of Dynamic Programming, that encloses Value Iteration and Policy Iteration algorithms, there are other approaches to deal with MDPs, such as Monte Carlo Methods and Temporal Difference (TD) Learning, this last being a combination of the two previous mentioned. (Szepesvári 2010) provides a general coverage of algorithms suitable for solving RL problems.

Unlike DP, Monte Carlo and Temporal Difference methods do not require a full model of the environment to achieve their aim. They rely on sample experiences (samples of states, actions and rewards) to compute approximations of the value functions, V(s), or action-value functions, Q(s,a), and to find optimal policies, π^*.

Besides sharing this idea with Monte Carlo methods, TD also shares principles with DP approaches, updating estimates based in part on other learned estimates. Some of the TD algorithms include the TD(0), SARSA and Q-Learning.

Eligibility traces provide a bridge from TD to MC methods. They provide for a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action, and these will be eligible for undergoing learning changes (Sutton and Barto 1998).

RL algorithms can also be categorized as on-policy or off-policy. Learning in an on-policy way means that the algorithm is learning the state or state-action values regarding the policy the agent is following. An alternative to this, consists in learning such values with respect to the optimal policy, i.e., to the greediest actions in each time step (off-policy) (Poole and Mackworth 2010).

3.3 - Continuous Reinforcement Learning

So far, the RL discussion has been focused on MDPs with a finite number of states and actions. However, in real world, spaces and actions are almost always continuous and not discrete.

A simple way of addressing continuous problems in RL is to discretize the state and action spaces, to be able to apply the algorithms earlier described, like Value Iteration, Q-learning, etc. This approach can be efficiently applied to many small problems (see Chapter 4), but there are some cases that do not support discretization. By discretizing, it is assumed that a set of infinite values take the same state values, which may represent a distorted reality. To many smooth functions, a reasonably good representation is only achieved with a very fine discretization, leading to the problem known as the curse of dimensionality. Note that the problem grows exponentially quickly, regarding the number of state variables and the intervals in which each one is divided.

There are two distinct model-free approaches to solve continuous RL problems: methods that search in the space of value functions and methods that search in the space of policies.

In order to generalize a state or state-action value, value functions are typically represented using function approximators, which provide estimates of the values. In this context, these approximators would be able to estimate the expected return even for

states that were never been experienced, taking examples from a desired function (e.g., a value function) and attempting to generalize from them to construct an approximation of the entire function (Sutton and Barto 1998). Value functions approximation was successfully applied to several applications (Beitelspacher et al. 2006; Buck, Beetz, and Schmitt 2002; Irodova and Sloan 2005). Despite the successful implementations, the calculated estimates may not be capable of accurately representing the truth state or action values. Moreover, most implementations based on this approach lead to deterministic policies, even when optimal policy is stochastic, which occurs quite often (Richard S Sutton et al. 2000). For more effective representations for value functions approximators, alternative approaches can be implemented, i.e., evolutionary functions approximation, which combines neuroevolutionary optimization techniques with classic RL methods (Whiteson and Peter Stone 2006), or even fuzzy approximators (Lucian Busoniu et al. 2005).

Policy Search methods are being used as an alternative to the Value Function Approximation techniques. These are the model-free methods that directly search in the space of policies. Policies have a simpler form than value functions, since the latter can become very complex for high-dimensional problems. Furthermore, Policy Search techniques are prepared to deal with continuous action spaces. An action is now represented by a parameterized policy $\pi(u|x) = p(u|x,\theta)$, which means policy follows a distribution $p(u|x, \theta)$ and is dependent of a set of parameters θ, which will be updated throughout the learning process. A very common representation for these parameterized policies is, for example, an ANN, whose weights correspond to the policy parameters.

A widely known policy search approach is based on gradients. “Gradient-based policy search is based on the assumptions that the policy is differentiable and that the locally optimal parameters found by the gradient method are near the global optimum. When these assumptions are not satisfied, global, gradient-free optimization algorithms must be used to search for the policy parameters.” (L. Busoniu et al. 2011). An extensive survey of policy gradient methods, along with its application and evaluation in some tasks of robotics domain, is presented in (Jan Peters and Stefan Schaal 2006).

With respect to gradient-free approaches, several optimization techniques can be applied, including mathematical optimization techniques, evolutionary computation, the cross-entropy method, etc. Policy search has been combined with evolutionary computation to create efficient and powerful algorithms like CoSyNE (Gomez, Jurgen

Schmidhuber, and Miikkulainen 2006), or Evolutionary Random Policy Search (H. S. Chang et al. 2006). Cross-entropy optimization was also used with policy search (Lucian Busoniu et al. 2009).

Combined approaches are also an alternative for tackling these continuous problems. Natural evolution strategies (NES) algorithm combines ideas from natural policy-gradients and evolutionary strategies, providing efficient update steps and preventing premature convergence (Wierstra et al. 2008). Furthermore, Kalyanakrishnan and Stone investigate the strengths of both policy search and value-function based RL to integrate them into a new algorithm (Kalyanakrishnan and Peter Stone 2009).

3.4 - Summary

By reading this chapter, one should become familiar with Reinforcement Learning framework and functioning. This stage provided a basis for implementing and designing RL methodologies and solutions to tackle the state-of-the-art case studies proposed as an objective in 1.3.

While in the beginning there was a greater focus on discrete algorithms to provide familiarization to the commonly used RL techniques, soon it became evident the need for a continuous methods research, in order to manage the CPGs parameters, as occurred with the gathered works presented in the previous chapter.

Chapter 4

Case Studies

This chapter presents the application of RL on two case studies: the Crawler and the Inverted Pendulum. In the first, a simple robot with a two Degrees of Freedom arm learns a basic locomotion pattern, and in the latter, the widely known Inverted Pendulum is studied. A few common algorithms, such as Q-Learning and Value Iteration, are applied to the mentioned case studies. The purpose of this particular study is to investigate the strengths and abilities of RL in the prospective generation of legged locomotion, gaining knowledge and experience for future application to more complex tasks.

4.1 - The Crawler

In order to characterize a locomotion problem and demonstrating in a simple way the learning process of RL algorithms, in this chapter it is developed the Crawler robot (Tokic et al. 2010; Tokic, Ertel, and Fessler 2009). Through this case study it is sought the ability of a simple robot to autonomously learn to move by interacting with the environment. The Crawler's controller is formulated with the aim of achieving an emerging locomotion pattern that arises from the necessity to move. Experiments were performed in the physically plausible Webots simulator.

The Crawler morphology is very simple: a two Degrees of Freedom arm at the front and passive wheels at the back (Figure 4.1 a)). The robot is expected to displace by moving the robot arm (joints g_y and g_x, Figure 4.1 a)), and the wheels only provide for hind support, not having any role in locomotion. RL algorithms are applied in order to find out the optimal policy, i.e., cyclical pattern of arm movements that generate

maximum forward displacement of the Crawler. In this context, a positive reward should be acquired whenever this desired displacement is accomplished.

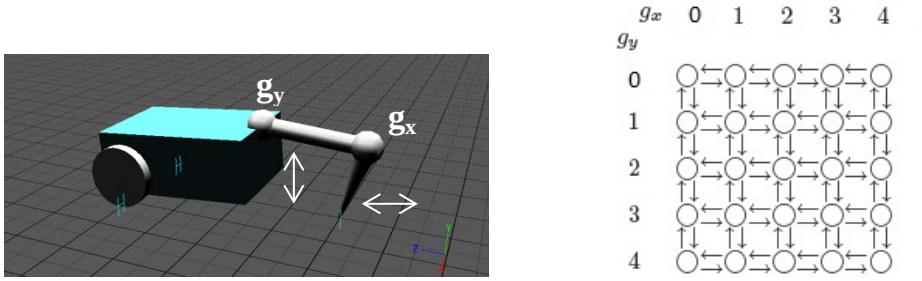


Figure 4.1 - a) The simulated Crawler robot rendered in Webots™ and its configuration. **b)** 5x5 Grid World for State representation. Possible actions are denoted by arrows (Tokic et al. 2009).

4.1.1 Solution Outline

As the learning approach is based on a MDP, the first step of the solution outline involves the definition of the set of the 5-tuple of representative elements of the MDP framework:

- The space of **states** is defined as the set of all possible arm's positions. The continuous angular space of both joints was then discretized to form 25 possible arm's positions: 5 positions for each joint, g_x and g_y, whose combinations are represented in a two dimensional grid world (Figure 4.1 b)). It is essential that the definition of this space of states takes into account the positioning of the states' referential beneath the robot's one, to allow the border of the arm to push the ground, leading to the raise of the robot's body and thus supporting the forward thrust.

- The **actions** denote movements to neighbor states of the grid: {up, right, down, left} (Figure 4.1 b)). Each action corresponds to the movement of exclusively one joint, causing the g_x to move face a horizontal state transition, while the g_y will only move against a vertical state transition. It should be also noted that a state may not allow the action selection from the entire set of possible actions, but only a few of them (i.e., the edges of the grid world representation). Figure 4.1 b) can also be seen as a transition graph, where the nodes represent each state and the action-arrows denote each possible state-action pair.

- In this specific case, the **transition probabilities** are deterministic, which means that an action performed in a certain state, leads to the successor state with probability equal to one. This is due to the noise-free movement of the joint's robot.

- Every time the robot performs an action, and consequently, transits to a new state, a **reward** is set according to the measured distance traveled.

- In addition, it is employed a **discount factor**, γ, with value 0.9, as suggested in (Tokic et al. 2009), as a starting point for experimenting several others discount values. This is used to specify the portion of influence of neighbor–state values on V(s). The reward gathered at timestep *t* will be discounted by a factor of γ^t.

Besides the experiments outlined for the different values of the discount factor, aforementioned, there is another parameter requiring attention and study: the exploration rate, ε. Assays were performed in order to meet an adequate solution for the crawler locomotion, taking into account the learning parameters γ and ε, and the learning process of the reward model (i.e., how to collect and learn all the rewards). Two potential solutions have been delineated to address this last issue:

1. Learning based on a complete reward model

It requires a comprehensive mockup of the rewards that would be achieved in each state transition. In this specific case, the entry to a certain state from different previous states leads to different rewards. That is, the reward depends not only on the current state, but also on the action that caused that transition. This fact is due to the strong dependence of certain states/actions on their successors, since sometimes it is only possible to achieve the objective after a proper sequence of joint movements.

This approach comprises two different types of simulations. The first one, which is also the simplest, makes use of the available reward model provided by the original work (Tokic et al. 2009), which is shown in Table 4.1. The reward table R_{ss}^{a}, presents the rewards that are achieved in each state transition (e.g., a transition from state (2,2) to state (3,2) leads to a reward of 6).

Table 4.1 – Original reward table $R^{a}_{ss'}$ for all state transitions (Tokic et al. 2009)

	0	1	2	3	4
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 -1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 -1
2	0 0 0 1	0 0 0 10	0 0 0 6	0 0 0 -1	0 0 0 0
3	-3 0 -41 46	-22 18 -57 50	-5 32 -45 35	-1 29 0 40	1 1 0 16
4	-56 0 -31 0	-68 6 -28 0	-48 28 -80 0	-64 46 -37 0	-19 43 0 0

The second set of simulations involves the whole process of exploration of all possible actions performed in all possible states, while the rewards are being observed and updated. Note that in this stage, the robot is not concerned to learn any policy, but rather to gather all rewards. Once the model is complete, i.e., after a long run of exploration, it is possible to execute the learning algorithm and evaluate the results.

2. Real-time Learning

This approach is quite important as it is more suitable for real problems, which usually demand immediate adaptation to the requirements of the surrounding environment. Thus, the execution of the algorithm will occur at the time of the acquisition and update of the rewards. These two stages, which in the previous method were implemented separately, are now merged in a single one. In the beginning of the execution, there is an empty reward model, which is being built throughout the learning process. This real-time gathering process of the achieved rewards is represented in Figure 4.2.

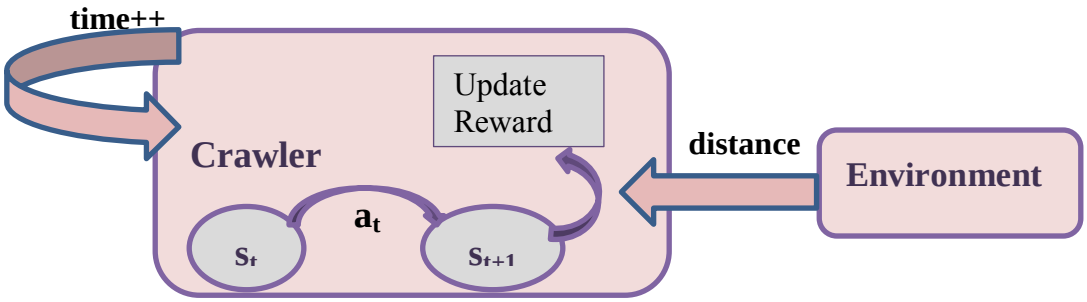


Figure 4.2 – Real-time updating of reward model

4.1.2 Value Iteration

By completing the delineation of the experiments and perceiving the objectives to be accomplished, it is possible to make the next move and implement the learning algorithms. It is intended to study and implement Value Iteration, a DP approach suitable for MDPs resolution. DP is computationally heavy in terms of memory and complexity, being applicable only to smaller problems, such as this case study.

Further experiments will address the assessment of the selected algorithm, as well as comparison with other RL algorithms which have also been implemented for evaluation purposes (presented in 4.1.6).

Subsequently, the functioning of Value Iteration algorithm will be presented.

Recalling Chapter 3, a very important concept for understanding Value Iteration is the value V(s), which represents the expected total reward of starting from the state *s* and respecting a given policy (Eq. 3.1). The algorithm task consists in assigning this numerical value, V(s), to each state *s*, repeatedly performing updates to the existing ones. As the state values depend on the rewards obtained, which in turn depend on the actions executed, some actions will lead to greater values. The algorithm must be capable of finding out which will be the actions that shall be taken at each particular circumstance, in view of the calculated values.

In order to succeed in its purpose, it is necessary to supply the required information, i.e., the initial state values, the initialization of the learning parameters, the initial state of the robot, the complete reward model and the transition probabilities, when required.

This last requirement of both the reward model and transition probabilities can be seen as a disadvantage, since it is assumed the knowledge of a model of the environment. Considering the above, one is faced with two inconveniences: first, it may not be trivial to get a reliable model, and second, the learned locomotion may only work in the modeled environment. The delineated approach of the learning process based on a complete reward model relies on this assumption, and thus, shares the mentioned disadvantages.

However, this drawback can be overcome if the reward model is constantly being updated depending on the effect of the action on the environment, which is the principle used in the second approach of learning (real-time learning). As a model is required as

an input to the algorithm, all state transitions lead to a reward of zero before the robot starts to explore/exploit. So, a matrix of zeros will be the initialized reward table.

Once all initializations and inputs required by Value Iteration are determined, the algorithm has now all the information needed to the calculation of the state values, which are expected to converge soon, given the small scale of the problem under study. Due to the Value Iteration property of calculating and updating all state values in each iteration of the algorithm, there is a need for storing those values. In this case, and since the space of states is of small dimension, a matrix will be used for storing the updated values, as also occurs with the storage of the rewards.

As soon as the state values convergence is verified, Eq. 3.2 can be used to find the optimal policy π^*, which is independent of the initial robot's arm position.

The representation of the complete Value Iteration algorithm is depicted in Algorithm 4.1.

Algorithm 4.1 Value Iteration

Inputs: Initial robot state; Learning parameters γ and ε; Matrices V for value states, and R for rewards

1. For each state *s*, initialize V(s) = 0
2. For each state transition, initialize $R^a_{ss'}$
3. *Repeat*
4. Choose and execute action (see 4.1.4)
5. Update reward
6. Observe next state
7. For every state *s*,
8. $V(s) = max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$ [1]
9. *Until convergence*
10. Return the optimal policy

For a more detailed explanation on how this algorithm proceeds along its iterations, at the end of this document (Appendix - Value Iteration Step-by-Step) it is provided a step-by-step for Value Iteration execution on the approach "Learning based on a complete reward model", resorting to the reward model of Table 4.1.

4.1.3 Q-Learning

This algorithm is a TD-learning approach, and in contrast with Value Iteration, does not require the definition of the transition probabilities, as desired. Furthermore,

[1] For this specific case, as the problem is deterministic, the equation can be simplified by removing the term $P^a_{ss'}$

neither a complete reward model is a requisite, so the learning process, including the gathering and updating of the rewards, is conducted in real-time. These two advantages fairly compensate the assumptions required in Value Iteration. However, a disadvantage of Q-learning can be considered regarding Value Iteration, when dealing with small scale problems. In contrast to DP approaches, that update all state values in each iteration, Q-learning updates each Q(s,a) only when that state-action pair (s,a) is visited, providing less information to the learning process.

In addition to the parameters already used in Value Iteration algorithm, Q-learning makes use of a further one: the learning rate, α, which indicates to what extent the new information will overwrite the old. A value of 1 leads to a complete overwrite and only the most recent information is considered.

For Q-learning, the concept that stands for the expected reward of each state is represented by the q-values, Q(s,a), denoting the value of taking action *a* in state *s* (Eq. 3.3).

Similarly to Value Iteration, the algorithm task consists in assigning this numerical value, Q(s,a), to each pair (state *s*, action *a*), performing updates to the existing ones. As these q-values depend on the rewards obtained, which in turn depend on the actions executed, some actions will lead to greater values. The algorithm must be capable of finding out which will be the actions that shall be taken at each particular circumstance, in view of the calculated values.

In order to succeed in its purpose, it is firstly necessary to provide some information to the algorithm, i.e., the initial q-values, the initialization of the learning parameters and the initial state of the robot.

Afterwards, the algorithm proceeds to the calculation of the state-action values of the state-action pairs it is experiencing, resorting to both exploration and exploitation. As soon as convergence is verified, Eq. 3.4 can be used to find the optimal policy π^*, which is independent of the initial robot's state.

The representation of the complete Q-learning algorithm is depicted in Algorithm 4.2.

Algorithm 4.2 Q-Learning

Inputs: Initial robot state; Learning parameters γ, α and ε; Matrix Q for q-values; Reward function or table

1. For each pair state-action (s,a), initialize Q(s,a) = 0
2. *Repeat*
3. Choose and execute action
4. Calculate reward
5. Observe next state
6. Update Q(s,a),
7. $Q(s,a_t) = Q(s,a_t) + \alpha[r + \gamma max_{a_{t+1}} Q(s',a_{t+1}) - Q(s,a_t)]$
8. *Until convergence*
9. Return the optimal policy

4.1.4 Exploration vs. Exploitation

An important step of any algorithm lies on the selection and execution of the action. It was already mentioned that an RL algorithm should be capable of finding out which actions lead to greater calculated values. But, as also stated, it is also important to choose different actions from the greediest ones.

Both approaches for the learning process of the reward model are implemented resorting to the use of exploration techniques. This represents one of the greatest challenges in the field of RL and it is important inasmuch as it will be possible to explore new solutions, which would not be possible to achieve with exploitation only. It is also important to opt for those that really lead to higher rewards, which would not be possible with exploration only. It is then intended that these two components are balanced, such that the results are the most satisfactory as possible.

The former solution, where a reward model should be available, requires a first phase with total exploration to gather the complete model. It should be noted that the first method of this approach assumes an already existing model, provided in (Tokic et al. 2009), which makes this first phase of exploration disposable. The second phase consists in pure exploitation, since the existing reward model will not be modified and all the necessary information for the learning process is known, which simply leads to the concern of discovering the movements that approximate the robot to its main objective.

In the second approach the process is not that simple, as it is not split into two distinct phases. There are two hypotheses under consideration which are expected to solve the exploration/exploitation trade-off: *Is it enough to assign a value to the*

exploration rate? Or is it better to assign different rates at different points of the execution? And what would be the best way to manage these variations? Such issues shall be answered upon the analysis of the results.

A very simple exploration technique is called the ε-Greedy exploration. This technique consists in selecting an action randomly with a probability ε, independently of V(s), instead of following the actual policy π(s). The greediest actions are chosen with probability 1- ε. This procedure is the one used to manage the selection of the actions.

4.1.5 Introduction of Stochasticity

The purpose of this section is to create a modified situation of this case study, where stochasticity is introduced in state transitions. This means that an execution of an action in a certain state may not lead to a specific state, as occurs with the case previously described, prevailing several possibilities to the actual next state. For each of those possibilities it is assigned a given probability.

There are several examples in which this situation can be verified. For example, if the actions space comprises the possibilities {north, south, east, west}, and the accomplishment of each possibility is via a compass, it is likely that it results in a certain deviation from the real direction of the cardinal point, due to the noise derived from the used device.

By using this case study it is possible to create a situation capable of demonstrating non deterministic state transitions, by forcing the introduction of noise in the robot joints that leads to the occurrence of one of two different possibilities: the transition to the expected state, or the permanence in the same state.

4.1.6 Other algorithms and their comparison

Following (Tokic et al. 2009), Value Iteration is the original algorithm implemented, and it has already been presented in 4.1.2. Policy Iteration follows the same methodology and assumptions adopted for the Value Iteration. It is also a DP approach and differs from Value Iteration to the extent that the component iterated this time is the policy, rather than the state values. Namely, in this case it is used the policy as the central component of the learning process, initialized randomly, to subsequently calculate the Values, instead of what occurred with Value Iteration. These two

algorithms are very similar and it is intended to compare each other, checking for main differences in terms of results.

Apart from DP, other approaches can be applied to this case study, as for example, Temporal Difference Learning. As a complement to this study, and for evaluation purposes, it has been selected a set of 5 more algorithms: Q-learning, SARSA, SARSA(λ), TD and TD(λ).

TD methods use experience to solve a prediction problem and estimate state values, and the differences of predictions over successive time steps do justice to the name "temporal difference". While in Value Iteration, state values are computed by iterating Eq. 3.1, a stochastic version is now used to estimate those values, without having knowledge of the transition model of the probabilities (model-free). The new update rule for the one-step TD, also known as TD(0), is presented in Eq. 4.1. A new parameter α appears, indicating the learning rate, that is, to what extent the new information will overwrite the old.

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)] \quad (4.1)$$

Every time a state is visited, this update rule is used to estimate the value of that state, in contrast to Value Iteration, where all state values are updated in each step of the simulation.

Q-learning, a TD-learning technique, differs from Value Iteration and TD(0) to the extent that the expected total reward of each state is now represented by Q(s,a). But as occurred with TD(0), each Q(s,a) is only updated when that that state-action pair (s,a) is experienced. The update rule for this algorithm can be observed recalling Eq. 3.3. Further detail concerning this Q-learning algorithm was discussed in section 4.1.3.

Another TD-learning algorithm which is very similar to the aforementioned is SARSA. The name SARSA comes from the update rule "State-Action-Reward-State-Action", and was first introduced in 1994 (Rummery and Niranjan 1994), where it was initially known as modified Q-learning. Instead of learning values with respect to the optimal policy, as in Q-learning, SARSA learns values regarding the policy the agent is following. So, while Q-learning is an off-policy algorithm, SARSA is on-policy, and this single difference is translated by using this time the next state-action pair according to the policy rather that the greedy state-action pair. So, Eq. 3.3 of Q-learning was adapted to represent SARSA's update rule, which is depicted in Eq. 4.2. Note that

SARSA is very similar to one-step TD, except that it makes use of action-state values, instead of state values.

$$Q(s, a_t) = Q(s, a_t) + \alpha[r + \gamma Q(s', a_{t+1}) - Q(s, a_t)] \quad (4.2)$$

In 3.2, the concept of eligibility traces was introduced as a reinforcement learning mechanism that provides for a temporary record of the occurrence of an event. Some TD methods can resort to eligibility traces for the purpose of improving the learning process, resulting in techniques like TD(λ) or SARSA(λ), where λ refers to the use of an eligibility trace.

Recalling one-step TD method, or TD(0) an estimate of the value of a certain state, V(s), combines old estimates of both V(s) itself and V(s') (Eq. 4.1). The next time step in the learning process consists in updating V(s'), which in turn resorts to old estimates of V(s''). TD(λ) takes advantage of the availability of this recent estimate of V(s') to improve the V(s) earlier updated. So, in each time step, all experienced states are updated, accounting for the new estimates of the current visited state.

Considering that e(s) denotes the trace of *s*, and $\delta_t = \alpha[r + \gamma V(s') - V(s)]$, the improved estimate of V(s) is given by Eq. 4.3. Similarly, the improved Q(s,a) estimates for SARSA(λ) algorithm is given by Eq. 4.4. (Sutton and Barto 1998) provides further details on these λ-methods.

$$V(s) = V(s) + \alpha\delta_t e(s) \quad (4.3)$$

$$Q(s, a) = Q(s, a) + \alpha\delta_t e(s, a) \quad (4.4)$$

The functioning of these 5 temporal-difference techniques is very similar, except in the update rule, or in the case of the last two described, the use of eligibility traces.

4.1.7 Results

The results achieved in the several experiments carried out are presented throughout this section. For a better analysis, the assays were divided into two main groups, corresponding to the two approaches described for learning the rewards model.

1. Learning based on a complete reward model

The obtained results coincided with those exposed in Tokic's work, as expected. The movement learning occurs quickly as it fails few moves until it initiates the cycle of the optimal policy and repeats the same movements through all the execution. This

optimal policy along with the resulting crawler movement is shown in Figure 4.3. The resulting displacement achieved by the Crawler was around 6.30 units of distance (u.d.).

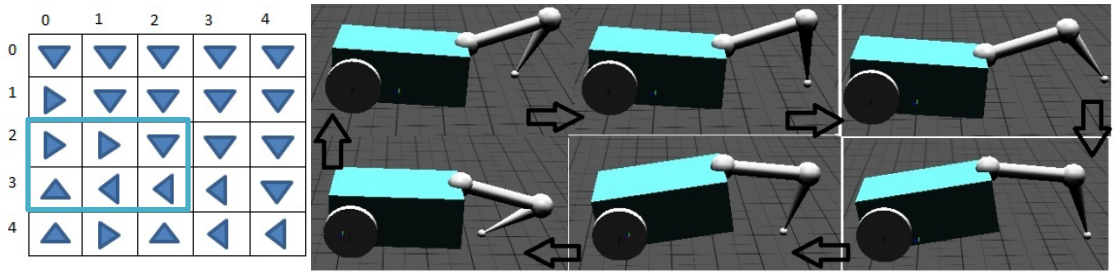


Figure 4.3 - a) Optimal Policy of Value Iteration with the provided reward model. The optimal cycle is encompassed by the rectangle. **b)** Resulting crawler movement.

Whichever the initial state defined, the robot will soon enter and get trapped in the optimal cycle, as it can be verified by Figure 4.3 a).

The discount factor, used as being 0.9, revealed a great impact in the results, because it dictates how much influence the future rewards have in the learning process. If this value is low and little influence is exerted by the future rewards, the algorithm will converge faster, but on the other hand, it may not reach the optimal cycle. To validate this statement, experiments were carried out, and for values below 0.8 it was confirmed that the algorithm is not capable of achieving the desired movement of walking forward, and the policy is maintained practically from the beginning to the end of the execution, converging too soon.

In order to compare two different learning algorithms, it was also implemented Policy Iteration, since it follows the same methodology and assumptions adopted for the Value Iteration. This algorithm is also a DP approach and is very similar to the implemented one, but the policy is the component to be iterated, rather than the state values.

Figure 4.4 graphically displays the state values evolution for the 6 states that compose the cycle of the optimal policy, along 30 iterations and for both Value and Policy Iteration algorithms. By analyzing the graphics, it is clear that Value Iteration converges faster, with regard to the state values of the six states comprised in the cycle of the optimal policy. This faster convergence by the Value Iteration might be explained by the fact that it focuses in the iteration of the values, unlike what happens with Policy Iteration.

Nevertheless, this difference is not significant, as the results achieved were quite similar and denote the same optimal policy.

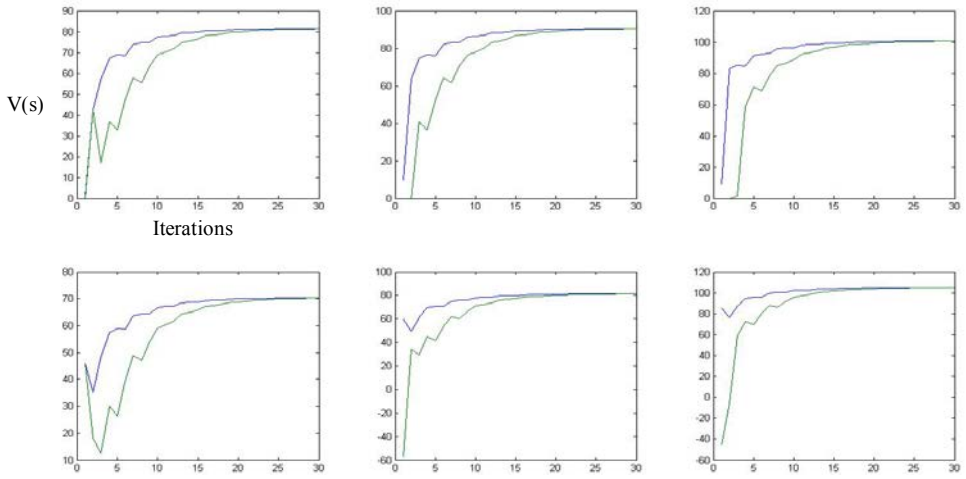


Figure 4.4 - Comparison between Value (in blue) and Policy (in green) Iteration; State values are presented in y-axis, while x-axis denote the iteration number. Each panel depicts a state variable of the cycle of the optimal policy

The second set of experiments consists in gathering the reward model in a first phase, using complete exploration to achieve this task. In this approach, there is one important factor to be taken into account: for how long should the gathering and updating be performed? It was necessary to carefully analyze the several options, and formulate a variety of tests. After a long period of updating the reward table, the model achieved was given as an input to Value Iteration, in order to allow the robot to exploit and choose the best actions, in the second phase of these experiments.

In terms of results, those were quite different depending on the runtime of the first phase. There were even a few tests that result in a movement totally opposed to what was expected (backwards). A reasonably acceptable solution was found after 1000 iterations of exploration. Still, the results were fairly worse than the previous, as the displacement performed by the robot in each cycle was greatly reduced.

2. Real-time Learning

In these set of simulations, the reward model had to be collected while running the learning process, comprising the two stages in the previous method in only one.

To tackle this approach, it is essential to perceive that the core of the issue is the exploration/exploitation trade-off. Therefore, several experiments were formulated in order to conclude about the best way to manage the exploration rate, concerning ε-Greedy technique.

As the exploration rate should not be very high, as it is intended to accomplish the best state value as possible, the value 0.1 was tested (i.e., 10% of cases resort to random actions, exploring, while 90% resort to the selection of greedy actions, exploiting).

However, this attempt has shown to be inadequate. A low exploration rate when there is no reward model is a bad idea to the extent that the robot chooses the greediest actions more often. And as the rewards are initialized as 0, what happens with excessive exploitation is that the algorithm reaches the optimal cycle soon, since it was not granted enough time to explore what was necessary. After a few movements, some rewards are updated, contributing to a value V(s) above 0. This small elite of actions already performed has a tendency to be selected more often than the remaining, whose rewards are still set to 0. So, the noise is insufficient, and it is important to include a higher noise rate.

Though, if the value of the rate is increased, the results are slightly different, which does not mean that are best. A high exploration in the beginning can be a good option, to provide the algorithm with a complete and accurate reward model. However, this rate is not appropriate after the reward model is gathered, since it prevents the robot from choosing the actions it knows that will trigger higher accumulated values. It is indispensable to find some way of balancing which allows to better manage this issue. So far it can be concluded that it is not enough to assign a value to the exploration rate that holds along the entire execution, in this specific case study.

Given the results, it would be ideal to assign a great rate holding at the beginning of the execution, which would decrease over time. This will ensure that all possibilities are tried out by the robot, leading to a complete model of rewards, so that it is able to select actions more wisely until convergence.

After an exhaustive set of ad hoc experiments, a quite acceptable result was achieved with $\varepsilon = 0.9$ in the first 100 iterations, followed by a drastic reduction until $\varepsilon = 0$, in iteration number 300. Specifically, in the beginning of the execution, 90% of cases resort to exploration, by executing random actions. This percentage decreases over the algorithm iterations, to allow the robot to perform sometimes the greediest actions. When 0% is reached for exploration, the robot only exploits, thus reaching the optimal policy.

By allocating more time of high exploration, the robot ended up losing track of which had previously been the best actions, updating the earlier rewards by some achieved by worse actions. Thus, also the number of iterations set for the initial exploration must be validated, for being a great influence on the final results. From the moment when the exploration is lower (after 100 iterations in this case), the number of iterations was no longer found as an influential factor.

It must be noticed that even though the displacement is a good reward function for the current case study, if a big penalization is assigned to bad moves, i.e., moving backwards, the robot may opt to stand still, opting for the actions that lead to a reward of zero (no displacement), which is not desired.

With these influential factors determined, the result consisted in a forward displacement, as expected. The states corresponding to the policy cycle are not the same as the implementation based on Table 4.1, because the mechanisms used and even the characteristics and dimensions of the robot, differ from the original (due to the lack of total knowledge of the problem). Moreover, the rewards model turned out to be totally different from that achieved in Tokic's work. And the latter fact might be connected with the different ground used in the assays, since it can lead to different learning movements, or even with the chosen positions for the states, which was not supplied in the original work. Consequently, the distance traveled by the robot also differed, being approximately 7.57 u.d., which is a greater displacement than the one provided by the original policy.

This time the optimal cycle comprises only two states (Figure 4.5), which implies the movement of just one joint. The fact that the robot only needs to perform two movements to return to the cycle start, against the 6 of the original approach, can be an explanation for this higher displacement achieved. One step of the first approach takes then three times longer, since one execution step is equivalent to one joint movement. Even though the robot steps in this real-time learning are considerably smaller, the Crawler is able to perform the triple the steps in the same learning time, which turned out to compensate.

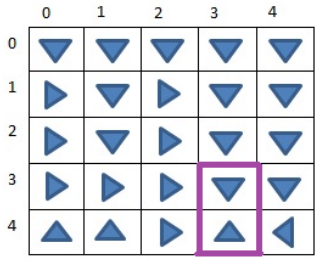


Figure 4.5 - Optimal Policy of Value Iteration with real-time learning

The final reward table presenting the rewards achieved in all state transitions, and which originates this optimal policy, is presented in Table 4.2.

Table 4.2 – Final reward table $R_{ss'}^{a}$ for all state transitions, achieved with real-time learning

	0	**1**	**2**	**3**	**4**
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 -1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 -1
2	-4 0 -1 -15	0 -31 -32 0	-30 -6 -22 -27	-6 -18 -18 -6	-7 0 0 0
3	-38 0 -36 -38	-62 -74 28 -53	-28 -28 -28 -48	-15 -46 -46 0	-39 -39 0 -29
4	32 0 30 0	-38 -57 -38 0	-40 -40 -85 0	51 -10 -61 0	-38 45 0 0

The following graphic (Figure 4.6) presents the results of all experiments performed with different algorithms with real-time learning approach, for evaluation purposes and as a complement to the study carried out. As the metrics used in each algorithm can vary in comparison with each other (i.e., state value or action value), the results are evaluated in terms of the Root Mean Square (RMS) Error, which evaluates the algorithms in terms of the convergence velocity. This error is calculated based on the difference between the optimal values and the values achieved in each step of the simulation, averaged over the states.

Notwithstanding the different convergence times, all algorithms were able to achieve the desirable forward displacement, although some in a not as effective way. Q-learning (Tokic et al. 2010) revealed very good results in terms of convergence, along with Value Iteration, that despite the higher value of error in the beginning, soon decreases to the same level as the former. Value Iteration evinced to be very suitable for this small dimension problem, because all value states are updated in each step of the simulation, in contrast with the other approaches. This fact is assumed as an advantage in this specific case, but it can turn out to be a drawback, due to the heavy computation. The primary algorithm also revealed a very good displacement, as already stated (of 7.57 u.d.). Q-Learning, despite the good convergence results, led to poorer displacement results, along with the two SARSA algorithms (less than 2 u.d.). Further, the resulting movement was visually quite unnatural. On the other hand, TD and TD(λ) achieved reasonable displacements, of around 6.93 and 5.26 u.d., respectively. TD(λ)

provided interesting results, reflecting a very natural and smoother movement, when compared to all the others, causing the actuation of the two joints, and comprising 4 states, leading to larger steps.

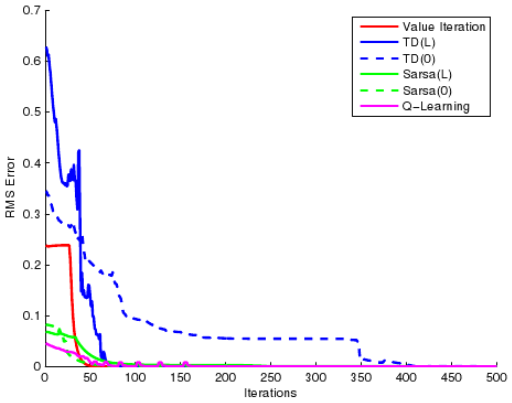


Figure 4.6 - Comparison of a set of algorithms according to Root Mean Square Error

The experiments addressed to the created nondeterministic situation, showed that despite the increasing of complexity, it is also possible to quickly achieve good solutions. The desired movement is achieved and the optimal cycle comprises four states, involving the use of both joints.

It is demonstrated that Reinforcement Learning is capable and prepared to solve non deterministic problems, such as the existing of noise in any device.

Videos of the results exposed are available in Youtube (Duarte 2012).

4.2 - The Inverted Pendulum

The Inverted Pendulum, also referred to as the pole balancing problem, is a classic control problem introduced by Barto, Sutton, and Anderson (A. G. Barto et al. 1983), and later discussed in Scherffig's work (Scherffig 2002). As the interests in the context of the thesis include biped locomotion, the inverted pendulum is particular appealing, due to its resemblance to the balancing problem in biped locomotion. As Scherffig states, “problems very similar to the inverted pendulum problem have to be solved by robots and biological organisms that walk upright”.

The inverted pendulum problem consists in a cart that can move freely in a track with a pendulum attached on the top (Figure 4.7). The goal is to maintain the pole in an

upright position by compensating the gravity effect through forces applied on the cart, and maximizing the total accumulated reward through Reinforcement Learning techniques. In this context, a greater reward should be acquired whenever the pendulum is closer to the desired position. The cart is then moved along the track to the right or to the left, according to the pendulum position.

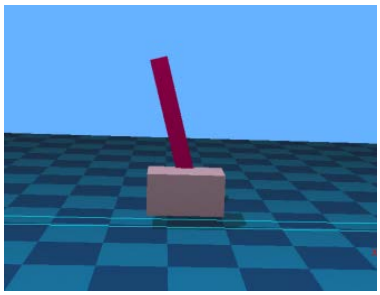


Figure 4.7 – Inverted Pendulum rendered in Webots™

4.2.1 Solution Outline

Just as the Crawler problem, this is based on a MDP, requiring the definition of the set of the 5 representative attributes of the MDP framework:

- The space of **states** is experimented taking into account several sets defined, in order to compare and study the influence of certain components comprising the states, namely, angular positions and angular velocities. As mentioned, the position of the pendulum is an essential feature for decision making, because it is what will influence the movement of the cart. As this feature is measured in terms of the angular position regarding the vertical position, it is concluded that this is a continuous problem that must be overcome. The space of angular positions is then discretized between $-\pi/2$ and $\pi/2$ rad, where the initial upright position of the pendulum corresponds to 0 rad. If these boundaries are exceeded, the pendulum is restarted to its original position and the learning process is continued. In order to know how to best discretize this interval, several non-overlapping and uniform intervals were formulated.

It has also been considered important to verify the influence of the angular velocity in the definition of a state. Thus, this factor was also subject to discretization in several uniform intervals, together with the angular position, since the angular velocity itself is not enough information to decide something accordingly. The interval defined was within [-40,40] rad/s, according to some experiments carried out.

- The **actions** denote movements performed by the cart to equilibrate the pendulum, due to external forces. These forces are the actions chosen for this concrete

problem, and are applied positively or negatively, depending if the intention is to move the cart to the right or to the left, respectively. An alternative version also includes an action in which the force applied is zero, which leads the algorithm to not influence the current movement of the cart (do nothing).

- In this specific case, the **transition probabilities** are very difficult to assign, since it involves a series of calculations dependent on the velocity and position of the pendulum, as well as the force applied to the car. For this reason, Value Iteration algorithm applied previously is not suitable for the inverted pendulum problem, as it requires the definition of this element. It is then selected an algorithm in which these probabilities do not need to be defined to accomplish a reliable solution.
- Every time the robot performs an action, and consequently, transits to a new state, a reward is acquired. The maximum **reward** is reached when the pendulum meets the upright position, and is reduced depending on the distance to that desired angle. Therefore, it is given by $-(\textit{angular position})^2$.
- It is employed a **discount factor**, γ, with value 0.9, as it was the best alternative found in the previous case study. This value acts a starting point for assessing several other discount values.

As the exploration rate was found to be a very influent parameter, experiments were outlined in order to achieve a better solution.

It must be retained that in a real environment, the position of the cart would be important since it is not possible to perform an infinite route. However, in this problem, this point will not be taken into account, thus assuming an infinite track whereby the cart can move freely. Note that in order to simulate an environment as close as possible to reality, physic issues must be taken into account (e.g., the gravity). And that is one of the major advantages of Webots simulator, because these issues can be automatically considered in the simulation, which facilitates a process that otherwise would have to be addressed, lying outside the scope of this dissertation.

4.2.2 Experiments

By completing the delineation of the experiments and perceiving the objectives to be accomplished, it is now available all the information required to formulate the learning process. It is intended to study and implement Q-learning, a TD-learning

technique also suitable for MDPs resolution (Watkins and Dayan 1992). Q-learning was chosen according to the followed approach (Scherffig 2002). The complete algorithm was shown in Algorithm 4.2, in section 4.1.3.

The experiments formulated for the Inverted Pendulum problem will be focused mainly on the 3 learning parameters (γ, ε and α) and the space of states and actions picked.

States and actions

As remains unclear how to best split the space of states and actions, these issues shall be answered upon the analysis of the results.

Learning parameters

It is intended to obtain the best value for the discount factor and evaluate the influence of future rewards. *In this specific case, will it be expected to obtain the same best value as for the Crawler? Will the learning of an optimal policy take place even with lower rates, unlike the Crawler problem?*

To evaluate and be aware of whither the new information should overlap the new one, it is necessary to test the range of possible values, αЄ[0,1]. *If α=0, no learning occurs (Eq. 3.3), but is it appropriate if the rate is too high?*

Regarding the trade-off of exploration/exploitation, the technique to be assayed in this case study is the already mentioned ε-Greedy exploration. The process to be followed is equivalent to the second approach implemented in the Crawler case (real-time learning). As different situations are being dealt, the assumptions made for the former case study cannot be formulated for the current one. So, the exploration/exploitation balancing issue remains for consideration, in order to analyze the best way of managing the exploration rate.

4.2.3 Results

The results are analyzed and evaluated throughout this section, for each formulated and carried out experiment.

The execution starts with the pendulum in the upright position, i.e., angular position = 0 rad. After, it is expected that the system will achieve the desirable behavior, after a number of unsuccessful attempts, keeping the pendulum in a vertical position without tumble. Specifically, it is expected that the cart will perform contrary actions repeatedly, moving to the left and to the right in an attempt to keep the

pendulum balanced. By means of common sense, it is not difficult to deduce that this cart movement should be towards the direction of the fall of the pendulum, so as to compensate the gravity forces.

States and actions

The selection of 100 states for representing the angular positions was shown to be a poor option, as the pendulum cannot achieve an infinite equilibrium. In this specific case, the number of actions (2 or 3) did not influence the results, since the states definition was not properly formulated. Note that these trials result in a period of stability for the pendulum, which is not too extended. It was also difficult to perform actions in all visited states, as the pendulum velocity is quite high to allow enough time for executing and evaluating them all. This suggests a decrease in the number of states in the hope of providing the necessary time for the learning algorithm. The division of the space of states in 20 intervals results in better outcomes. It is achieved a state of equilibrium, but nonetheless with several oscillations.

By changing the reward model, for evaluation purposes, and in order to provide more clues about the actions that should be performed, it was noticed an improvement in the results achieved. The referred reward consists in distinguish whether if the pendulum is moving away or approaching the upright position. This reward was under consideration because of the following hypothesis: sometimes, a position closer to the goal may suggest worst results than one farther away, if in the present situation the pendulum is moving away in the former case, and approaching the goal in the latter.

In fact, this last experiment led to good results, even if there are 100 intervals for defining the states space, which previously did not work. All these trials suggest the inclusion of the angular velocity as part of the state definition, as it denotes the direction of the pendulum movement. Regular intervals were defined in the range of [-40, 40] rad/s.

Several combinations were tested in order to conciliate these two state components. The results have indicated that a space of possible states of 4x20 (4 position states and 20 angular states) or 20x20 yield particularly good results, when compared with several other combinations tested, including the previously described ones. Regarding the number of actions, the selection of only two possibilities {+force, -force} did not result in the pendulum equilibrium. The addition of the third possibility that does not influence the cart movement (null force) revealed to be necessary, because not always

the movement must be counteracted or further strengthened. For example, there are situations when it is too soon to contradict the movement, causing a fall, but at the same time, reinforcing it, would cause the car and pendulum to move faster, being more difficult to cause a change of the pendulum direction.

The results achieved matched the expected ones, as a fall to the right direction implies applying a positive force to counteract the pendulum movement, and a fall to the left direction leads to a negative force applied to the cart (Figure 4.8). The cart's alternate movements to the left and to the right are sometimes imperceptible.

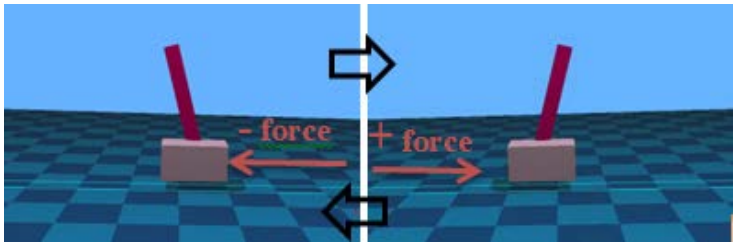


Figure 4.8 – Resultant movement of the car in order to balance the pendulum

After a few attempts, the cart is able to balance the pendulum for a certain period of time, until the oscillations become increasingly wider and the pendulum falls. Other falls are followed until an infinite stability is finally reached. The video of the results exposed can be found in (Duarte 2012).

Other alternatives to further refine the best number of actions were tested, given different forces magnitudes. In the view of the obtained results, the magnitude is not much relevant to the learning process; however, it seems to influence the pendulum oscillations during the balancing period. To verify this assumption, the initial configuration was tested by varying the magnitude of the applied forces. With a force value lower than the original, the cart was also able to balance the pendulum, although with less oscillations. The results of this experiment can be examined in Figure 4.9, where one can analyze the angular position in radians throughout 5000 iterations. From a certain point, if the force is 4N, there are more variations in the angular positions, meaning additional oscillations and instability. With the force of 0.5N, there is a higher stability of the pendulum, without major variations. When the pendulum reaches the 1.5 rad angular position, it means that the cart was not able to balancing it, causing a pendulum fall.

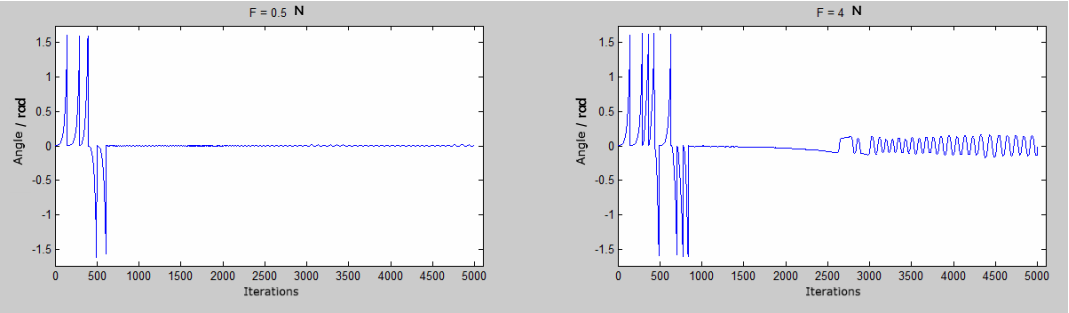


Figure 4.9 - The force magnitude as an influence factor in the pendulum oscillations

Learning parameters

The discount factor, which denotes the amount of influence of future rewards, was first tested as 0.9, which led to the aforementioned results. Nevertheless, for inferior values the achievement of similar results also takes place. For discount rates equal or inferior than 0.3, the cart was not capable of balancing the pendulum. As the future rewards are not taken into account, the learning of the optimal policy does not occur.

What seems to happen with the Inverted Pendulum problem is that it does not require much information on future rewards as the crawler does. In the Crawler problem, too much dependence on subsequent actions was noticed. If no dependence was taken into consideration, all actions and the consequent transition to a following state would lead to reward=0, because no displacement would take place. So, the rewards achieved in Crawler are dependent of a sequence of actions (minimum of two actions). In the current case, all actions lead to a reward different than 0, except for the exactly upright position, and so there is a not so strong dependence.

Considering the learning rate, if this has the value 0, there is no learning because the new information has no importance in the calculation of Q-values. For this reason, Q-values remain the same during all execution, being selected and performed constantly the same action. If $\alpha>0$, the learning with good performance occurs, independent of the value assigned. As a first glance, no difference within the range of learning rate values is noticed, and thus the learning period was extended. Scherffig argues that this parameter has a great influence on the oscillations even though sometimes this can only be proved throughout a very long execution (Scherffig 2002). A set of values was then

tested along 20,000 iterations and the results for 4 chosen rates are presented in Figure 4.10, for the variation of the angular position, in radians, over time.

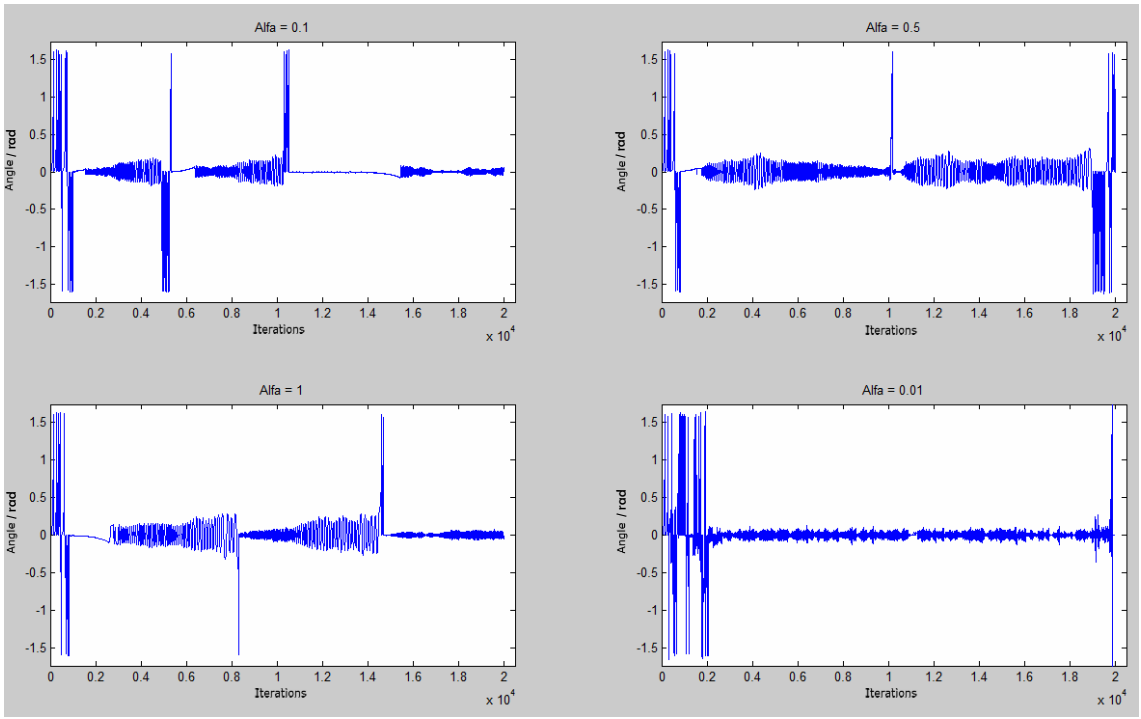


Figure 4.10 - Influence of learning rate in the angular position over time

Low values for the parameter α, reflect minor changes in the values of Q, while high values (α=1) reflect more abrupt changes (Eq. 3.3). This point revealed influence on the oscillations and stability of the pendulum in the long term. As verified via the graphic, the value for which α is minimum, α=0.01, offers the greater stability of the pendulum, so that it is maintained in a vertical position with constant oscillations over all iterations. On the other hand, learning takes longer, with more falls at the beginning of the execution that end up being compensated by the stability later found.

For α=0.1, whose value is also low, the results revealed to be less stable, even though a period of very weak oscillations (around iteration 12,000) was achieved. Despite having reached good results, some falls were noticed, as well as greater oscillations of the pendulum.

Higher values (α=0.5 and α=1) result in more instability and oscillations. They present too much variation of the angular position, along with some falls. However, the differences within this range of values were not as significant as those found for the lowest ones.

The exploration rate did not show considerable influence in this specific problem, as opposed to the Crawler case study. The results did not seem to improve or worsen,

except that it might take longer to reach the desired movement, depending on the period defined for the exploration phase. One explanation for this occurrence is related to the fact that the values are initialized with the maximum value that they could achieve. That is, as the rewards depend on the function $-(\textit{angular position})^2$, their values will always be negative, unless the position is exactly 0. As the rewards are negative, the q-values will also be. Therefore, when experiencing an action and updating the corresponding value, this will be negative and consequently the worst of all, since the others continue to be 0. This means that in the next situation that the system visits this state, it will choose another action, and so forth. This ensures that all actions are tested in each state, so there is no risk of not testing the greediest actions.

4.3 - Summary

The case studies discussed in this chapter provided experience and knowledge that are expected to be useful in the formulation of future experiments. In this chapter it was demonstrated the functioning of various important RL algorithms, and the theory described in the previous chapter was put into practice. The algorithms are analyzed and evaluated for the experiments carried out, for these two simple case studies. The results showed that the selected algorithms were able to achieve good results and successfully accomplish the goals delineated (for videos of the results, see (Duarte 2012)).

After this phase of the thesis' work, one should be familiar with the MDP framework and understand the major challenges of the used RL techniques. This phase also provided a familiarization with Webots simulator, who proved to be very useful in the development of the solutions.

The work developed so far resulted in a paper published in the International Conference of Numerical Analysis and Applied Mathematics (ICNAAM) (Duarte et al. 2012).

Chapter 5

A Natural Actor-Critic Solution

This chapter first introduces the elements provided for the solution development concerning the DARwIN-OP robot model and the CPGs architecture. The main purpose of the thesis is approached, with the help of the knowledge and experience acquired throughout the earlier stages, and it is presented a RL approach designed to tackle the problem of tuning CPGs parameters, with the aim of optimizing the basic movement. The applied RL methodology is Natural Actor-Critic (NAC) which is described along with the achieved results.

5.1 - DARwIN-OP

DARwIN is a family of autonomous robots capable of human walking and motions, developed by Dr. Dennis Hong and his team at the Robotics & Mechanisms Laboratory. The DARwIN humanoid robots first appeared in 2004 and have evolved thereafter through various generations.

DARwIN-OP[2], which stands for Dynamic Anthropomorphic Robot with Intelligence–Open Platform, is a recent miniature-humanoid robot that provides a research platform for studying robot locomotion, autonomous behaviors and several other topics on Artificial Intelligence.

This robot comprises a total of 20-DOF (6 DOF each leg + 3 DOF each arm + 2 DOF neck), measures approximately 455mm and weighs about 2.8Kg.

---

[2] See http://www.robotis.com/xe/darwin_ko and http://www.romela.org/main/DARwIn_OP:_Open_Platform_Humanoid_Robot_for_Research_and_Education

The DARwIn-OP simulation in Webots features a complete dynamics model of the robot. It is capable of achieving movements through its 20-DOF and perform tasks with its accelerometer, camera, gyro and LEDs, similarly to the real robot. This model is presented in Figure 5.1 b), along with the real robot platform (Figure 5.1 a)).

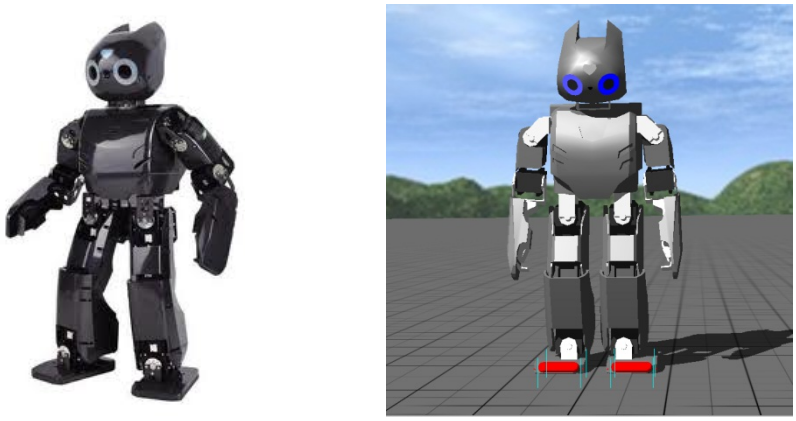


Figure 5.1 - a) The real DARwIN-OP robot **b)** The simulated Crawler robot rendered in Webots™

Biped locomotion is achieved by moving the robot legs, through its 12-DOF legs joints. Each joint is controlled by a specific motion generated by the CPGs, as it will be presented in section 5.2. The schematic view of the leg joints, and the corresponding movements for which they are responsible for, is depicted in Figure 5.2, where the rolling of the cylinders indicate the direction of the joint's movements. Arms and neck will be ignored since they do not contribute to locomotion, the purpose of this thesis.

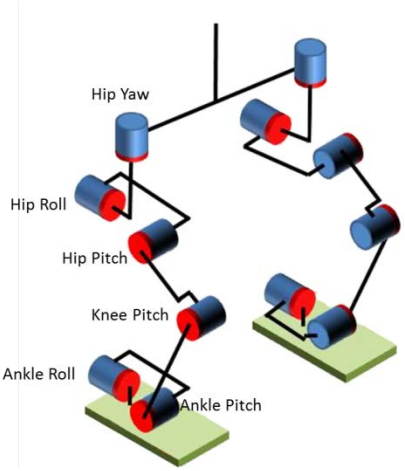


Figure 5.2 – Schematic view of legs joints

5.2 - Central Pattern Generators

The CPG approach to biped locomotion developed by the group work "Adaptative System Behaviour Group" (Vítor Matos and C. Santos 2012) is based on phase

oscillators, where there is a capability of incrementally adding basic motion primitives. This locomotion system is a bio-inspired architecture that is implemented as an organized network of unit generators as non-linear differential equations. Each unit generator is responsible for the activation of one joint and it is composed by motion pattern generators driven by a global rhythmic generator (Figure 5.3). The combined produced behaviors will model the locomotion pattern.

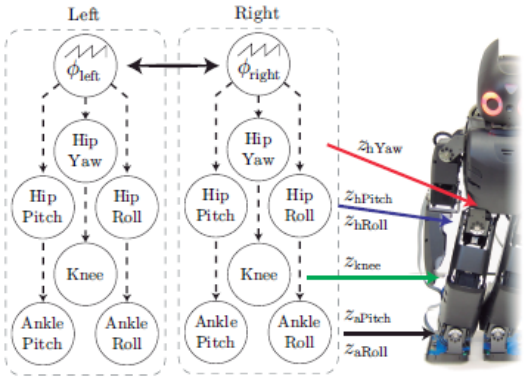


Figure 5.3 - CPGs and corresponding phase oscillators, motion generators and corresponding joints (Vítor Matos and C. Santos 2012)

It is considered that as a starting point in the life of the robot, there should be a basic motor repertoire of motion primitives capable of achieving basic, but sufficient, walking behavior. Afterwards, it is presented the functioning of the CPGs and the description of the general motions, translated by a set of differential equations that control the robot actions.

The rhythmic generator of the mentioned unit generators uses a phase oscillator (ϕ), increasing monotonically and linearly with rate ω, to produce the rhythmicity for the motion patterns, following Eq. 5.1. In this case, only two phases are generated in each time step, one for each robot leg, which means that joints placed in the same leg share the same phase oscillator.

$$\dot{\phi} = \omega \qquad (5.1)$$

Joint position ($z_i(t)$) is generated by the motion pattern generators according to the current phase of the CPG, through the following non-linear differential equation (Eq. 5.2), where each $f_{i,j}^{n}$ corresponds to a motion primitive to be included in the generated motion of each joint, j defines the unit generator of a joint in a leg (hip roll, ankle pitch,

etc), i defines the left or right leg, $O_{i,j}$ specifies the equation offset, and β is a relaxation parameter for the offset.

$$\dot{z}_{i,j} = \sum_{n=1}^{N} f_{i,j}^{n}\left(z, \dot{\phi}_{i,j}, \phi_{i,j}\right) - \beta(z_{i,j} - O_{i,j}) \quad (5.2)$$

By acting on the hip roll and ankle roll joints, the robot achieves a balancing movement, shifting horizontally the pelvis at the hip. As the biped steps alternately, it must correctly land over the supporting leg during a step cycle, allowing the contralateral leg to execute the swing phase. So this is a very important motion of bipedal walking, since an incorrect displacement over the supporting foot may lead to a fall. The designed movements for the hip and ankle roll joints are generated according to the following Eq. 5.3 and Eq. 5.4.

$$f_{i,hRoll}^{balancing} = A_{balancingHip}\dot{\phi}_i \sin(\phi_i) \quad (5.3)$$

$$f_{i,aRoll}^{balancing} = A_{balancingAnk}\dot{\phi}_i \sin(\phi_i) \quad (5.4)$$

In order to add some natural resemblance to human walking, a pelvic rotation must be performed at the hip yaw joints, as follows (Eq. 5.5).

$$f_{i,hYaw}^{pelvisRot} = A_{pelvisRot}\dot{\phi}_i \sin(\phi_i + \frac{\pi}{2}) \quad (5.5)$$

Leg flexion motion is performed by the non-supporting leg during swing phase, by actuating on the three pitch joints: hip (Eq. 5.6), knee (Eq. 5.7) and ankle (Eq. 5.8). Parameter σ specifies the duration of leg flexion during the step cycle and is set to $\frac{\pi}{6}$.

$$f_{i,hPitch}^{flexion} = \frac{A_{Hip}\dot{\phi}_i}{\sigma^2} \exp(-\frac{{\phi_i}^2}{2\sigma^2}) \quad (5.6)$$

$$f_{i,kPitch}^{flexion} = -\frac{A_{Knee}\dot{\phi}_i}{\sigma^2} \exp(-\frac{{\phi_i}^2}{2\sigma^2}) \quad (5.7)$$

$$f_{i,aPitch}^{flexion} = \frac{A_{AnkKnee}\dot{\phi}_i}{\sigma^2} \exp\left(-\frac{{\phi_i}^2}{2\sigma^2}\right) - \frac{A_{AnkHip}\dot{\phi}_i}{\sigma^2} \exp\left(-\frac{{\phi_i}^2}{2\sigma^2}\right) \quad (5.8)$$

To flatten the trajectory occurred when the knee supports the body weight during stance phase, a small flexion on the knee joint is added during the stance phase to the knee (Eq. 5.9) and ankle (Eq. 5.10) pitch joints. π provides for a phase shift to allow the flexion to occur at the middle of the stance phase.

$$f_{i,kPitch}^{flexKnee} = A_{flexKnee}\dot{\phi}_i \sin(\phi_i + \pi) \quad (5.9)$$

$$f_{i,aPitch}^{flexKnee} = A_{flexAnk}\dot{\phi}_i \sin(\phi_i + \pi) \quad (5.10)$$

The generation of body propulsion, allowing the legs to move forward and backward, occurs when actuating on the pitch joints of hip (Eq. 5.11) and ankle (Eq. 5.12).

$$f_{i,hPitch}^{compass} = A_{compassHip}\dot{\phi}_i \sin(\phi_i + \frac{\pi}{2}) \quad (5.11)$$

$$f_{i,aPitch}^{compass} = A_{compassAnk}\dot{\phi}_i \sin(\phi_i + \frac{\pi}{2}) \quad (5.12)$$

The combination of all these motions leads to the desired biped walking.

The motions performed by each joint are summarized in the following equations (Eq. 5.13 - 5.18).

$$\dot{z}_{i,hRoll} = f_{i,hRoll}^{balancing} - \beta(z_{i,hRoll} - O_{i,hRoll}) \quad (5.13)$$

$$\dot{z}_{i,hYaw} = f_{i,hYaw}^{pelvisRot} - \beta(z_{i,hYaw} - O_{i,hYaw}) \quad (5.14)$$

$$\dot{z}_{i,hPitch} = f_{i,hPitch}^{flexion} + f_{i,hPitch}^{compass} - \beta(z_{i,hPitch} - O_{i,Pitch}) \quad (5.15)$$

$$\dot{z}_{i,kPitch} = f_{i,kPitch}^{flexion} + f_{i,kPitch}^{flexKnee} - \beta(z_{i,kPitch} - O_{i,kPitch}) \quad (5.16)$$

$$\dot{z}_{i,aPitch} = f_{i,aPitch}^{flexion} + f_{i,aPitch}^{flexKnee} + f_{i,aPitch}^{compass} - \beta(z_{i,aPitch} - O_{i,aPitch}) \quad (5.17)$$

$$\dot{z}_{i,aRoll} = f_{i,aRoll}^{balancing} - \beta(z_{i,aRoll} - O_{i,aRoll}) \quad (5.18)$$

5.3 - Natural Actor-Critic

To successfully accomplish the task at hand, it is important to idealize a RL solution for the humanoid locomotion problem. This solution involves the investigation of the algorithms and methodologies used for the continuous cases, and the gathering of the state-of-the-art regarding to Reinforcement Learning with the use of CPGs (see 2.3). It is also important to understand the essentials behind the CPGs approaches.

Most of the works investigated made use of Artificial Neural Networks for their architecture of the neural oscillators. As the approach of the work group is considerably different, as it is based on differential equations, the adaptation of the solution was found to be a very difficult task. One of the most used algorithms based on the state-of-the-art, is the Actor-Critic with Policy Gradient methods. This appeared to be a good approach for the problem at hand, but the methodologies described were only applied for ANNs, as can be stated from 2.3.

DMPs share the aims of the CPGs, being capable of generating rhythmic behaviors (see section 2.3). Investigations on this mentioned approach led to better target the

idealization of the solution. From the investigation carried out, it was concluded that there are very few approaches to these CPGs representation that apply the Reinforcement Learning framework, and those are mostly implemented towards the goal of moving an arm (playing baseball, making pancakes…), in contrast to the present project.

The next step in the idealization of the solution, leads to a thorough investigation, attempting to understand the different algorithms used. Three have been found, suitable for dealing with DMPs: the Natural Actor-Critic, the Policy learning by Weighting Exploration with the Returns, and the Policy Improvement with Path Integrals. Explored all three, it was decided to use the NAC algorithm, due to the higher familiarity with the operations, in comparison with the other two algorithms, and because Actor-Critic is a widely used approach, even in the earlier implementations studied, which were addressed to ANN.

As already stated in the state-of-the-art review, Natural Actor-Critic was introduced by Jan Peters (Peters, Vijayakumar, and Schaal 2003) and further developed in (Peters 2007; Peters and Schaal 2008). NAC is a known sub-class of policy gradient methods, which is based on the simultaneous online estimation of two structures: the *actor*, which corresponds to a conventional action-selection policy, and the *critic*, which operates as a state-value function (Bhatnagar et al. 2009).

The actor is responsible for updating the policy, via natural policy gradients, and towards the direction of the average reward gradient. The critic obtains both the natural policy gradient and additional parameters of a value function by linear regression. It evaluates the current policy to provide a basis for an actor improvement (Jan Peters and Stefan Schaal 2008). Several methods can be used by the critic to address its task, but the ones based on TD learning have proved in the literature more efficiency in large applications (Bhatnagar et al. 2009).

Although distinguishable, actor and critic must be solved simultaneously to find an optimal policy. The interaction between these two structures is illustrated in Figure 5.4.

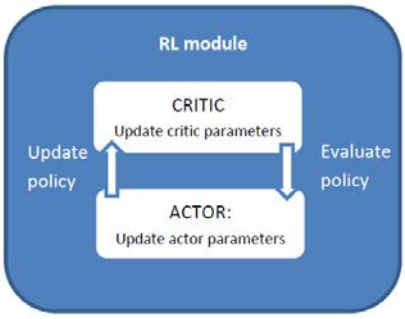


Figure 5.4 – Interaction of actor and critic structures

Sensed a particular state, an action is chosen by the actor from a stochastic and parameterized policy $\pi(u|x) = p(u|x,\theta)$, leading to a state transition and the achievement of a scalar reward, on the part of the critic, which will cause the actor to reconsider its actions. It is assumed that the policy $\pi(u|x)$ is linear and continuously differentiable with respect to its parameters θ.

It is also assumed the existence of a set of linearly independent basis functions $\phi(x)$ so that the state-value function can be approximated with linear function approximation. These basis functions reflect the important features used to describe an environment state, and which can be useful for an efficient decision making. There are no general rules to the design of these features. Therefore, experience, analysis and intuition are needed to provide a good selection (Roy 1998). This mentioned work states that "a good choice of basis functions is critical to success". Given a set of state variables, the simplest process of deriving basis functions is to use each variable directly as a basis function, plus a constant function, $\phi_0(x) = 1$ (Konidaris, Osentoski, and Thomas 2011).

The critic evaluation makes use of Least-Squares Temporal Difference (λ) (LSTD(λ)) policy evaluation algorithm (Bradtke and Barto 1996), which is a derivation from a TD-learning rule on least-square techniques. The algorithm makes use of a set of statistics, z, A and b (Boyan 1999), which support the calculation of the natural gradient w. This gradient serves in updating policy parameters, following $\Delta\theta_t = \alpha w_t$, where α is the learning rate. After the parameters update, the critic has to forget part of its accumulated statistics using a forgetting factor $\beta \in [0,1]$.

To adequately adapt LSTD(λ) in the current approach, new basis functions must be defined, reducing bias and variance of the learning process (Jan Peters and Stefan Schaal 2008): $\tilde{\phi}_t$, resorting to the current state features and $\hat{\phi}_t$, which uses the next state features.

The complete NAC algorithm, including details of the calculation of the set of statistics, parameter updates and the new basis functions, is depicted in Algorithm 5.1.

Algorithm 5.1 Natural Actor-Critic

Inputs: Parameterized policy $\pi(u|x) = p(u|x,\theta)$, basis functions $\phi(x)$, policy derivative $\nabla \log \pi(u|x)$, reward function, learning parameters α, β, γ and λ

1. Initialize basis functions and statistics
2. *Repeat*
3. Select and execute action: $u_t \sim \pi(u|x)$
4. Calculate reward r_t and observe next state
5. **Critic** (Evaluation)
6. Update:
7. Basis functions: $\tilde{\phi}_t = [\phi(x_{t+1})^T, 0^T]^T$;
8. $\hat{\phi}_t = [\phi(x_t)^T, \nabla \log \pi(u|x)]^T$
9. Statistics: $z_t = \lambda z_{t-1} + \hat{\phi}_t$;
10. $A_t = A_{t-1} + z_t(\hat{\phi}_t - \tilde{\phi}_t)^T$;
11. $b_t = b_{t-1} + z_t r_t$
12. Critic parameters: $[v_t^T, w_t^T]^T = A_t^{-1} b_t$
13. **Actor** (Updating)
14. If gradient estimate is accurate, update:
15. Policy parameters: $\theta_t = \theta_{t-1} + \alpha w_t$;
16. Forget statistics:
17. $z_t = \beta z_t$; $A_t = \beta A_t$; $b_t = \beta b$;
18. *Until convergence*

The information required for the algorithm described herein, which is not initialized or considered, will be discussed in the next section.

NAC has been applied to several recent works to solve Reinforcement Learning tasks in the field of robotics and control. For example, NAC was applied with recursive least-squares method for optimizing the locomotion of a two-linked robot arm, similar to Crawler, achieving good performance results (Park, Kim, and Kang 2005). In (Ueno et al. 2006), RL is used to accelerate the automatic learning of a set of parameters, employing a modified natural actor-critic method, the off-policy NAC. Works approaching motor primitives have been developed, mainly for optimizing an arm movement, i.e., for playing baseball (Peters and Schaal 2007). In (Girgin and Preux 2008) it is presented a new approach to automatically construct a set of basis functions within the context of NAC algorithms, which allow the representation of more complex policies. Recently, a Switching Max-Plus-linear model was developed to apply episodic-NAC to a hexapod robot, with the aim of maximizing velocity in flat or irregular terrains (Knobel 2011).

5.4 - Solution Outline

Comprehended the RL methodology addressed to the continuous cases, as well as the organization of the CPGs implemented and the chosen algorithm, it is now possible to define all the variables required to the learning process in order to get closer to the proposed goal.

Abstracting from the issues related to the approaches' details, it is essential to comprehend the process of interaction between RL and CPGs, and to realize what are the duties associated to each module. The following diagram (Figure 5.5) illustrates the interaction between the two modules and the physical environment (Robot). CPG controller calculates joint velocities using pre-defined non-linear equations. The velocities will then be integrated in order to provide the joint positions to the robot, which will result in a smooth and rhythmic locomotion movement. The introduction of a RL module aims at optimizing the robot locomotion by updating particular parameters, resorting to the information about the state of the robot, so that the movement generated by the CPG is now optimized with respect to the basis movement. The RL module is translated by the Natural Actor-Critic algorithm, exposed in the previous Figure 5.4.

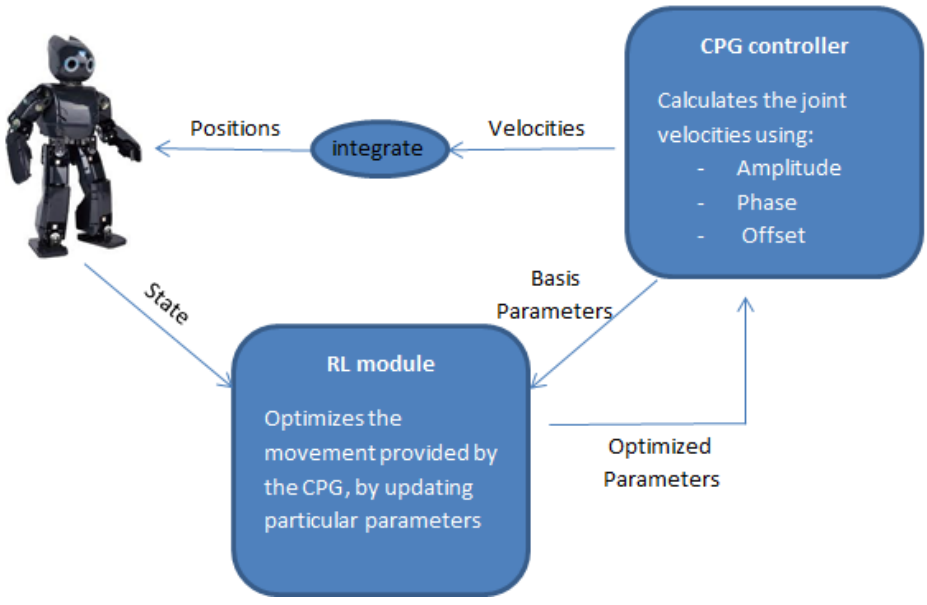


Figure 5.5 – The integration of RL and CPGs modules

After, it is summarized the necessary information to the implementation of the algorithm and the choices made for each component required as an input.

In the beginning of a project like this, it is imperative to define smaller goals to achieve in each phase. So, in a first attempt to develop the solution and test its feasibility, it is going to be addressed the RL framework only to two joints of each leg, performing the movement of balancing: hip and ankle roll joints. If good results are achieved, then it is time to move forward and tackle the other joints of the robot. Balancing movement seemed a good optimization target since only two joints are responsible for this sort of motion. The most intuitive reward, in terms of the distance traveled, is difficult to control targeting only two joints, since it is the result of the combination of all leg joints. Therefore, as it is intended to optimize the balancing movement, the reward to use by NAC is in terms of the distance from Center Of Pressure (COP) to foot center. Specifically, this distance shall be the nearest to 0, during stance phase (in swing phase there is no COP). This should encourage the robot to maintain the balance and stability. Additionally, a penalization should be included whenever the learning process causes the robot to fall down.

It is assumed in this concrete case that the hip roll and ankle roll joints perform exactly the same movement. As to the distinction of the two legs, the only difference is the phase, which leads to the adoption of the same trajectory for both legs (dephased). Note that although the phase-dependent kernel functions for each leg are different, the policy parameters should be the same for both legs.

Despite being based on similar principles, DMPs and the CPGs used in this project slightly differ. DMPs are composed of weighted kernel functions, whose weights are to be updated, while the equations used in this project are rhythmic *sin* or *Gaussian* functions, with all the common parameters of sinusoidal waves (amplitude, frequency, phase). As a basic movement is provided, in terms of a (or a set of) well-defined *sin/Gaussian* function(s), *how can the equations of DMPs be integrated with the current CPG approach*? The answer is not trivial, but as the CPGs equations are non-linear dynamical systems, they easily allow the sum of more components or motor primitives (e.g., Eq. 5.17 has 3 summed primitives). Therefore, the component of the DMPs (F) will be summed to the CPG original equations of hip and ankle roll, in an attempt to optimize the robot movement of balancing in locomotion. The value of this summed component corresponds to the action to take by the robot, as follows (Eq. 5.19).

$$Action = F = \frac{\sum \psi\theta}{\sum \psi} \qquad (5.19)$$

$$\psi_i = \exp(h(\cos(\phi_i - c_i) - 1)), \qquad h = 2.5N, \boldsymbol{c_i} \in [0, 2\pi] \qquad (5.20)$$

DMPs resort to kernel functions ψ (Eq. 5.20) (Gams et al. 2009), which are phase dependent and pre-defined, and which are weighted by a set of parameters θ, whose function is to determine the waveform of the produced periodic trajectories. So, the action selection is based on a parameterized policy $\pi(u|x) = p(u|x,\theta)$, as it was previously stated. The initial set of parameters, $\theta = \theta_0$, is populated with zeros, so that the initial movement that NAC attempts to optimize is equivalent to the basic movement provided by the original CPGs. Furthermore, this policy follows a normal distribution N(F,σ) to allow some exploration: $\pi(u|x) = p(u|x,\theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}(x - F)^2)$.

With the summed component of the DMPs, the new hip and ankle roll movement equations are given, respectively, by Eq. 5.21 and Eq. 5.22.

$$\dot{z}_{i,hRoll} = f_{i,hRoll}^{balancing} - \beta(z_{i,hRoll} - O_{i,hRoll}) + \frac{\sum \psi\theta}{\sum \psi} \qquad (5.21)$$

$$\dot{z}_{i,aRoll} = f_{i,aRoll}^{balancing} - \beta(z_{i,aRoll} - O_{i,aRoll}) + \frac{\sum \psi\theta}{\sum \psi} \qquad (5.22)$$

To approximate the state-value function with linear function approximation, as seen in section 5.3, it is necessary to define the set of basis functions ϕ(x), so as to more accurately describe an environment state. It can be difficult to set the important features of a state. In this context, the most intuitive state variables correspond to the 12 joint positions of robot legs, along with the 2 phases, one for each leg. So, 15 basis functions are initialized in all iterations:

$$\phi(X) = \{1, z_{left,hRoll}, z_{left,hYaw}, z_{left,hPitch}, z_{left,kPitch}, z_{left,aPitch}, z_{left,aRoll}, z_{right,hRoll}, z_{right,hYaw}, z_{right,hPitch}, z_{right,kPitch}, z_{right,aPitch}, z_{right,aRoll}, \phi_{left}, \phi_{right}\}.$$

These basis functions are initialized according to the initial state of the robot.

Note that, because of the number chosen for the basis functions, policy parameters are also 15 in total, so that all matrix calculations are possible. Consequently, it is required a set of 15 kernel functions, each one weighted by each policy parameter.

In critic evaluation stage, NAC makes use of LSTD(λ) algorithm, which in turn makes use of a set of statistics, *z*, *A* and *b*. The matrices *z* and *b* are populated with

zeros. As A cannot be a zero matrix, it must be initialized with a small positive multiple of the identity matrix, in order to allow the calculation of its inverse.

Finally, it is assumed that the problem is a Markov Decision Process in discrete time and with a continuous state set and a continuous action set.

It should be noticed that the algorithm presented in Algorithm 5.2 and further discussed in the current section, works for one degree-of-freedom (DOF), and it must be adapted to solve multiple-joint problems. Even though a single execution is needed in the present case, because there is only one set of parameters to update, the implementation should be prepared to parallel executions, for the further addition of more DOFs, where several parameter sets should be encountered.

It is intended to evaluate the solution and understand the changings caused by learning, when compared to the pre-learning phase (CPGs only), analyzing the variation of the rewards achieved and the changing of the joints trajectories.

5.5 - Results

The implementation described in the previous section didn't result in the expected outcome. First, it is important to highlight that the NAC algorithm presented several problems during its implementation. Due to matrix operations as inversions, numerical instabilities had emerged, causing large increments to the values of the parameters, which exponentially increased and led to infinite leg's movements. To solve this, the basis functions must be normalized by the standard deviation (except the "1", that is normalized by the number of samples). Another important step is to include and apply ridge regression method, that is, "a number δ is added to the elements on the diagonal of the matrix to be inverted" (Bjorkstrom 2001).

With the solution to solve inversion problems, the parameters did not increased exponentially as before, but they still do not stabilize in the long run, only for a short period of time. Also, the reward variation reveals no improvement, when compared to that before the introduction of RL, where the same function was achieved. The distance from COP to foot center was already close to 0, and the values after optimization remain the same.

The next figures present an overview of the results achieved, by means of the variation of F's sum (Figure 5.6 a)), and the consequent $\dot{z}_{Left,hRoll}$ trajectory (Figure

5.6 b)). It is observed that the trajectory achieved with the NAC algorithm is very similar to the one generated by the CPGs, in spite of existing an increase of the component F. This can be indicative that the parameters chosen are not very sensitive to slight changes, needing more abrupt modifications of their values to result in differences in the trajectories. But on the other hand, if parameters take too high values, consequently leading to a huge increasing of component F, the rewards will get worse, thus resulting in a robot fall.

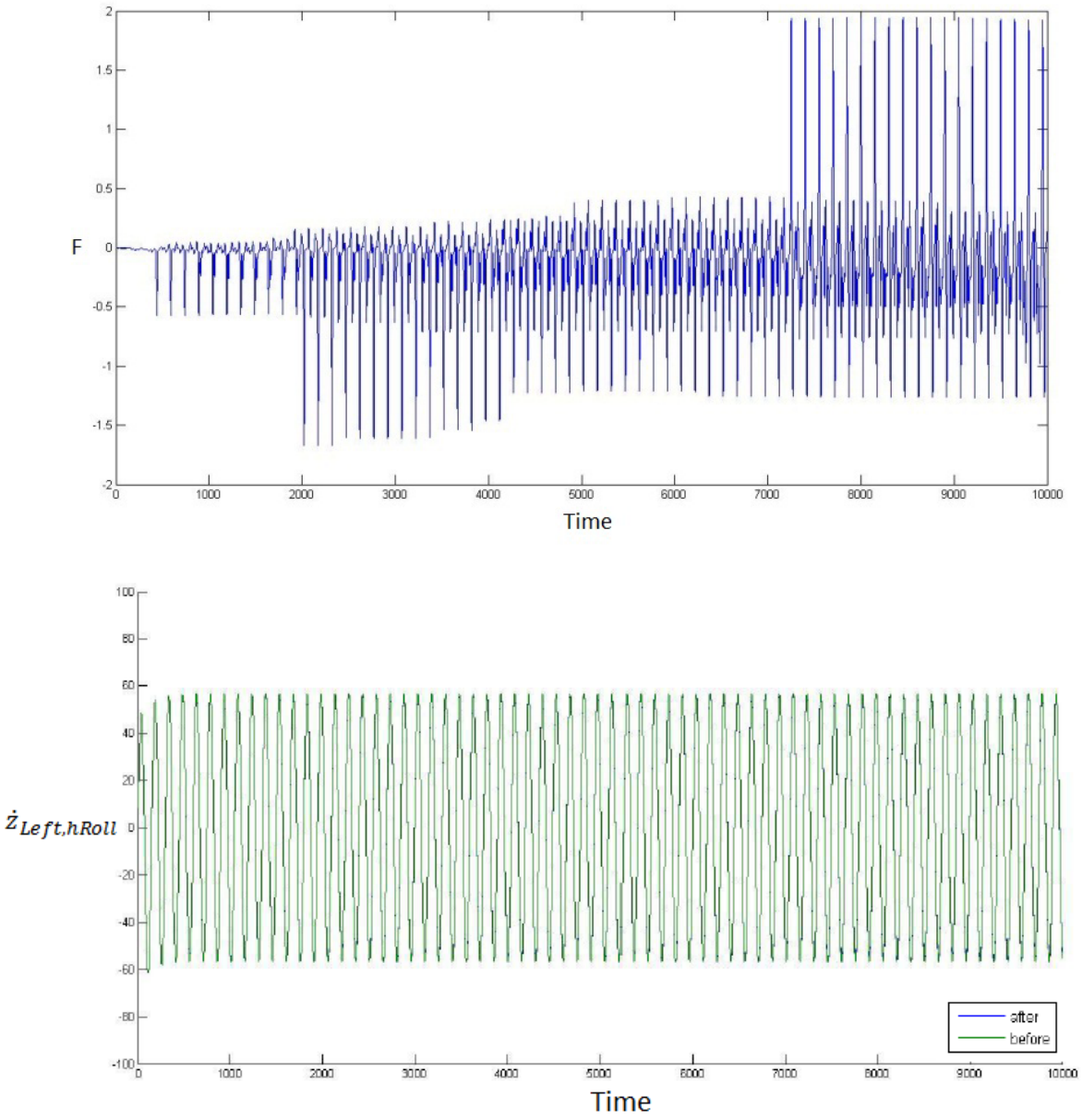


Figure 5.6 – a) F's variation **b)** $\dot{z}_{Left,hRoll}$ trajectory before learning (green) and after learning (blue)

Taking these results into account, a question arises: *why did not the NAC algorithm lead to some reasonable results*? Two hypotheses immediately arise to answer this question:

1- The algorithm is not appropriate to the problem exposed.

2- The solution conceived to implement Reinforcement Learning to the tuning of CPGs is not the ideal one:

a. Is the sin function of the basic movement good enough to model the desired movement? As the rewards values are too close from the maximum of rewards function, this could be a strong hypothesis. In this case, the NAC algorithm (or other) would not be capable of improving it, taking into account the current solution.

b. Is the reward model unsuitable? This is also a very strong hypothesis, that later became more obvious, throughout the development of the entire project. No matter how the solution is worsened, the rewards achieved are always around the same values, which reveals that they do not reflect enhancements or worsenings. So, *how to pick a good reward model, appropriate for this case*? This is a question that remains unanswered, and is further evidence that the reward model has a huge influence in RL problems.

Analyzing the conceived hypotheses, it is important to identify the possibilities of tackling the problem in a different way. One potential solution passes through the implementation of other algorithm, which implies further study and investigation. However, as the solution seems not very suitable, either due to reward function, either due to the addition of the DMPS to the basis movement performed by the *sin* functions, also the solution had to be improved. This suggests a different investigation target, leaving the concept of DMPs aside.

An additional possibility consists in implementing DMPs as they are presented in the literature, removing the *sin* functions of the used CPGs, and providing a basic movement with some initial hand-tuned parameters. This hypothesis does not seem to be viable since it suggests a major change to the existing CPGs, which is not a purpose of this thesis.

There is also a possibility of changing the target joints to better idealize a reward model, but this issue revealed to be very difficult to conceive, as the use of only few joints is recommended to verify the viability of the solution, due to the increasing number of parameters.

5.6 - Summary

In this phase it was developed an integrated solution of CPGs and RL, which has revealed to be a nontrivial task, due to the differences observed in the provided CPGs and those commonly used in the literature. The solution was idealized as an adaptation of the known DMPs, widely applied in robotic control. However, it did not result in any optimization, whether due to the unsuitable reward model, or even to the solution itself, as it was hypothesized that the *sin* function was already very good and well-defined. Natural Actor-Critic revealed some numerical instabilities and sensibility to particular parameters. This last detail was not further explored due to the several difficulties encountered and the absence of satisfactory results.

Although it was not achieved the desired results, this stage provided more experience and knowledge, which has allowed to convincingly state about the viability or not of following some conjectured paths. Due to the complexity of the problem at hand, further investigation and study are required, to properly design a solution capable of handling with the integration of the two proposed approaches.

Chapter 6

A Cross-Entropy Solution

In the previous chapter it was verified that the use the NAC algorithm, along with the solution designed, did not result in any optimization. This chapter presents the design of a different solution, as well as the implementation of a different algorithm which shares similar concepts with some classic RL methods: the known Cross Entropy Method (CEM). This is a general algorithm, widely applied in many contexts of robot control.

6.1 - The Cross Entropy Method

From the state-of-the-art review, it is realized that Policy Search methods are providing more interesting results and are the core of the most recent investigations in solving continuous RL problems. One of the most popular approaches is the use of gradient-based methods. NAC and PoWER algorithms are examples of this general approach. But despite the good results in the literature, such gradient-based methods are in general complex and still difficult to apply to many continuous state and action problems, since they are sensitive to parameterization. Moreover, they may comprise numerical instabilities.

Gradient-free policy search represents a set of several gradient-free optimization techniques that can be used rather than gradient-based methods, allowing a richer policy parameterization, which can be non-differentiable. One of these alternative approaches that has been a study target lately is based on trajectory rollouts, e.g., PI^2. Many works have shown that these methods are able to outperform the previously mentioned, in terms of speed of convergence, quality of the solution, and even the need for less

parameter tuning (E. Theodorou et al. 2010). Instead of using gradient estimates, this algorithm is based on the concept of "probability-weighted averaging", which reveals similarity with some other methods, like Covariance Matrix Adaptation – Evolution Strategy (CMA-ES) (Hansen and Ostermeier 2001) and Cross Entropy Method (CEM) (R. Y. Rubinstein 1997). These algorithms seem to share very similar concepts, as well as identical parameter update rules, and this relationship can be examined in (Stulp and Sigaud 2012). CMA-ES was even efficiently applied for solving RL problems in the optimization of the weights of neural networks (Igel 2003). CEM was used in similar problems, for optimizing a set of weights of basis functions in a Reinforcement Learning task (Menache, Mannor, and Shimkin 2005).

Afterwards, comparisons between policy gradient methods, e.g., NAC, and CMA-ES have arisen, as two different methods of tackling Reinforcement Learning tasks (Heidrich-meisner and Igel 2008a, 2008b). The experiments described to compare the two methodologies suggest that CMA-ES outperforms the NAC algorithm. Despite these results, Heidrich-meisner and Igel argue that these approaches are quite similar, and even refer to CMA-ES as an evolutionary RL algorithm, as it can be applied to the optimization of RL parameterized policies and to MDPs, which are the basic formalism to describe RL problems. Furthermore, in (Whitley et al. 1993) it is stated that "although not often thought of in this way, genetic algorithms are, in a sense, inherently a reinforcement learning technique", a statement which can also be generalized to CMA-ES and CEM. However, although in some problems this assertion may be closer to the truth, it must be taken into account that one distinguishing feature of RL is the possibility of sequential decision-making.

(Stulp and Sigaud 2012) states that CEM is a special case of CMA-ES, by setting some parameters of the latter to specific values. Because of the similarity encountered between CEM and RL methods like PI^2, and given the great theoretical complexity that PI^2 has demonstrated, CEM appeared to be a good way of tackling the problem at hand, since it has been evidenced to be suitable for solving MDPs. The purpose of this section is then to introduce the Cross Entropy Method as a starting point for the design and implementation of a possible solution.

The Cross Entropy Method is a simple, efficient and general Monte Carlo approach for solving continuous optimization problems (R. Y. Rubinstein 1999), suitable for problems of robot control. Its aim consists in achieving a solution that optimizes some objective (cost/reward) function, by testing a distribution of solutions, instead of just

one solution iterated over time. That distribution is intended to be approximated towards the proposed goal, until convergence to the optimal solution. The term "cross-entropy" denotes the distance between distributions.

The basic principle, which is very similar with PI^2 principle, is illustrated in Figure 6.1.

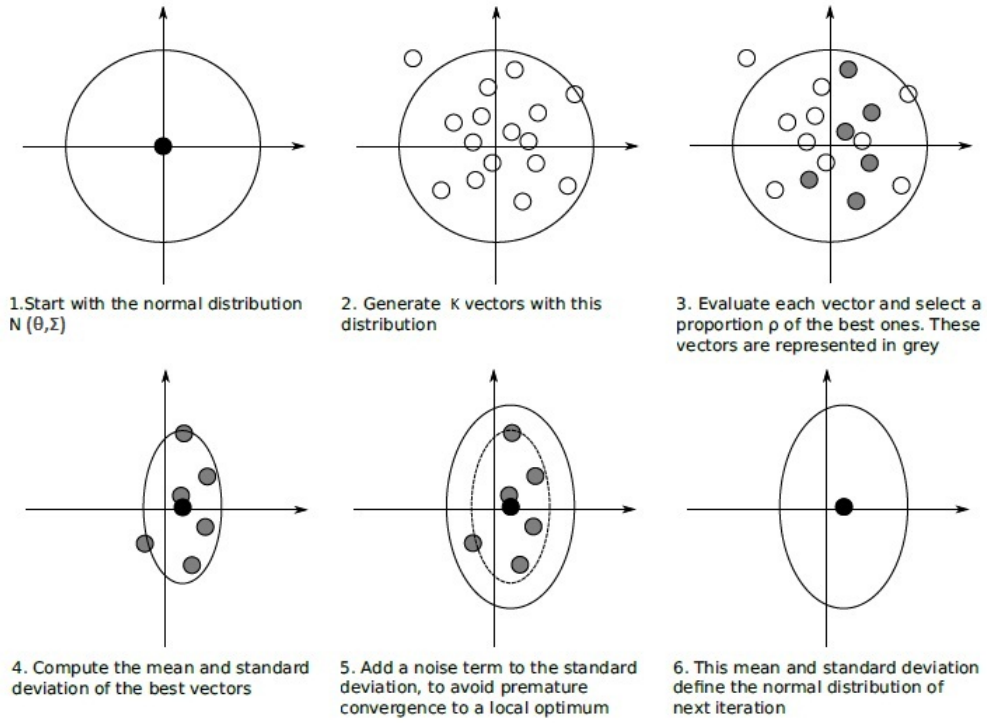


Figure 6.1 - Schematic view of CEM (Marin et al. 2011)

Four simple steps sequentially organized can be used to synthesize this algorithm: *Sample*, by taking K samples from a given distribution; *Sort and Select*, by sorting the samples with respect to a cost calculated for each one and selecting the first *elite* ones; *Update*, using the chosen best vectors to calculate the new distribution, and finally, *Iterate*, returning to the first step with the last distribution calculated (Stulp and Sigaud 2012). Resorting to Figure 6.1, stages 1 and 2 constitutes *Sample* phase; stage 3 represents *Sort and Select* phase; stages 4 and 5 make up *Update* phase; and *Iterate* is represented by stage 6.

The similar PI^2 algorithm also attempts to approximate a distribution towards the direction of lower cost. However, in this case all vectors are chosen to calculate the new distribution, in terms of weighted average, instead of using just the best ones.

The distribution most commonly used is the Normal Distribution, $N(\theta,\Sigma)$, where θ and Σ denote, respectively, the mean and the covariance of that distribution. Note that the mean θ may denote a set of parameters, rather than just one, which are represented by a vector and thus leading to a matrix representation of the covariance of the distribution. The K samples derived from $N(\theta,\Sigma)$, which are represented by θ_k, are in this case vectors, whose parameter values differ slightly from the ones of the distribution's mean.

The representation of the complete CEM algorithm is depicted in Algorithm 6.1.

Algorithm 6.1 Cross Entropy Method

Inputs: Initial distribution $N(\theta,\Sigma)$; Cost/reward function; Parameters: K and *elite* (number of chosen vectors)

1. *Repeat*
2. From the chosen initial distribution $N(\theta,\Sigma)$, compute K samples:
3. $\theta_k \sim N(\theta,\Sigma)$
4. For each sample K
5. Run the execution for N time steps, and calculate the cost of the sample
6. Sort the samples in ascending order with respect to the cost and choose the first *elite* ones
7. Attribute probabilities of *P = 1/elite* to each chosen vector
8. Compute the weighted mean and covariance of the set of elite samples, according to the Normal distribution
9. *Until convergence*

The information required for the algorithm described herein, within the context of biped locomotion, will be discussed in the next section.

To better understand the functioning and the details of CEM, and in order to conceive a suitable solution in the subsequent section, taking into account the setting of all inputs required to the algorithm implementation, a simple case study is hereafter demonstrated, based on (Stulp and Sigaud 2012).

Considering a parameter space with just two dimensions, represented by a Cartesian space, a vector θ_k symbolizes a point in the referential, with coordinates (x,y). The aim of the application of CEM in this case study is to approximate an initial point to the origin of the referential. Starting with an original multivariate normal distribution, with $N(\begin{bmatrix}8\\8\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 9\end{bmatrix})$, K = 10 samples are taken in each round of the optimization process. As they are points in Cartesian space, the cost calculation is immediate, with no need of running each sample during N time steps. After calculating the chosen cost of each sample in terms of distance to the origin, elite = 5 vectors are chosen to

participate in the calculation of the next distribution, each one with probability of $\frac{1}{5}$. Vectors with a great distance from the origin are eliminated, in favor of those which achieve a distance closer to 0. The mean of the distribution, which before took values (8,8), assuming a cost of approximately 11.31, is now taking x and y-values closer to the origin (0,0), and consequently, lower costs as desired. Initially, no amount of noise is added to the covariance matrix, i.e., stage 5 of Figure 6.1 is not considered.

After a number of CEM updates, the algorithm converges to a mean with values (7,0), achieving a reward of 7, which is closer to 0 than the 11.31 of the first iteration was. Despite this improvement, the objective of approximating the point to the origin was not accomplished. So, as no value noise was added to the covariance matrix, it was verified premature convergence.

An amount of 0.8 was added as the noise value, and after some updates, CEM is now able to achieve a solution with a reward very close to 0. Due to the noise introduced in the implementation, an error of about 0.1 was found taking into account the origin (0,0). CEM was found efficient in solving this simple problem, but the achievement of a better or worse solution is dependent on the decisions made by the programmer, i.e., the amount of noise included, the elite number chosen, etc.

Apart from these simpler optimization problems, this algorithm has been applied to different applications, like the traveling salesman problem (R. Y. Rubinstein 1999), simulated helicopters (Kobilarov 2001), DNA sequence alignment (Keith and Kroese 2002), etc.

CEM for policy optimization, such as in this work, was introduced in (Mannor, R. Rubinstein, and Gat 2003) for solving MDPs with large state spaces. The extension to MDPs with continuous action spaces can be found in (Stulp and Sigaud 2012). There were not encountered works with similar aims of this thesis, e.g., robot locomotion using CPGs.

6.2 - Solution Outline

The current chapter aims to expose a possible and appropriate solution for the tuning of particular CPGs parameters, to optimize biped locomotion. Both CEM and

NAC (Chapter 5) solutions are intended to address the main purpose of the thesis, described in 1.3.

The CEM algorithm described in the previous section requires the setting of some inputs, e.g., the actions and rewards, as well as considerations whether about attributes initializations, or the iterations itself. The necessary information to the implementation of the algorithm is carried out below.

It is desired to formulate this problem as a RL problem, so it is important the setting of particular features which are characteristic of RL methods. The states are represented by all important environment features, or more specifically in this case, they comprise the set of the 12 leg joints' positions. The combination of the appropriate set of positions is crucial to a good performance, thus states play a very important role in the optimization process despite they are not directly required for the algorithm inputs or calculations.

The action selection, as in the previous chapter and other policy search methods, is based on a parameterized policy $\pi(u|x) = p(u|x,\theta)$, which will lead to a state transition and the achievement of a scalar reward. The action chosen for the NAC solution was related to the DMPs component to sum to the original CPGs equations, due to the parameterization sensibility that the method appears to have.

"For PI^2 learning, the linear parameterization in θ is among the most important features" (Stefan Schaal et al. 2010). This requirement of linearity criterion of NAC or PI^2 algorithm, especially conceived for solving DMPs, is a quite strong limitation regarding the project at hand. Recalling Eq. 5.1 - 5.18, it can be realized that the used CPGs are very well defined, as well as the adjustable parameters. As opposed to the known DMPs in the literature, the parameters are clear and distinct, and the meaning of each one is acknowledged. For example, changing the amplitude parameters will influence the maximum and minimum angular positions that each robot joint can reach. It would be really interesting to study this influence of a particular parameter, directly controlling it. That is an advantage of using CEM, as it has no linearity restrictions (Mannor et al. 2003). The algorithm can be applied directly to the parameters, requiring a simpler solution than in Chapter 5.

Therefore, in the present case, a single action will be performed during a trial. Due to the advantages stemming from the used CPGs, i.e., the generation of continuous and rhythmic movements, the parameters to be tuned only need to be set at the beginning of

the execution. The joint positions observed during a robot step will be repeated throughout all steps.

The action then corresponds to the test of a single parameter vector, i.e., each of the K generated vectors. The parameters chosen for the implementation and test of the outlined solution are the CPGs amplitudes, and so, each action coincides to the vector:

$$\boldsymbol{\theta_k} = \{A_{balancingHip}, A_{balancingAnk}, A_{Hip}, A_{Knee}, A_{AnkHip}, A_{AnkKnee}, A_{flexKnee}, A_{flexAnk}, A_{pelvisRot}, A_{compassHip}, A_{compassAnk}\}$$

The mean vector $\boldsymbol{\theta}$ of the initial distribution, also representing the initial set of parameters of the parameterized policy, is defined by the initial hand-tuned amplitudes, which allow the robot to perform the basis movement to be optimized.

This parameter selection will allow to control how large should be a robot step, how high must be lifted the knee, how much the hip should swing, etc. Because of this large control and as it is now possible to address all leg joints as opposed to what occurred in the previous chapter, the balancing problem is no longer the target of optimization. With the control lying on all robot joints, rather than just two, it is now possible to improve or decrease the distance traveled, the new optimization target. It has to be noticed that as the reward chosen is in terms of the distance traveled in a certain period of time, it is only necessary to gather a single reward, at the end of each complete trial.

Unlike the case studies of Chapter 4, or even the NAC solution of Chapter 5, CEM does not make use of a discount factor, so the influence of future rewards cannot be measured. In this specific case, this turns out to be irrelevant, since only a single reward is gathered, with no need of considering future rewards.

With this, all RL features are set out. However, CEM has its specific features to be set. Defined the mean vector, it is possible to initialize the covariance matrix. As there is no rule to attribute values to the covariance, as it depends on the problem at hand, different options have to be tested. It has to be taken into account that high values lead to high exploration, and slight changes of joint positions can lead to major changes in the resulting movement. Therefore, it is then initialized as an identity matrix, as to allow a properly tradeoff between exploration and exploitation.

Premature convergence is a very common problem, and in order to address it, a noise value needs to be added to the covariance matrix during the calculation of the new distribution. This corresponds to stage 5 in Figure 6.1. There is no rule to define this amount of noise, so several values need to be verified.

Another specific feature of CEM is the elite number, namely, the number of best vectors to be chosen in the end of each round to take part in the calculation of the new distribution. As with the previous mentioned parameter, a set of tests is required to determine the best option.

Each vector from the set of vectors initially generated has to be executed for a given period of time, to assess the effect of the action and to be later evaluated in terms of the cost or reward achieved. Therefore, for each round, it was generated 10 vectors (K=10), each of which runs during 1000 iterations/timesteps, to allow the robot to take a few steps and evaluate the distance travelled. This value could not be chosen taking into account only one step, because different parameters lead to different step lengths, which influence the number of time steps needed to perform a robot step. Those trials that result in the fall of the robot are given a high penalization that prevents them from being chosen by the CEM algorithm.

The first of the K generated vectors was set to be the mean vector itself, used for evaluation purposes, which means that each set of parameter vectors includes the mean vector and K-1 others.

The algorithm was then implemented with the decisions described above, resorting to Webots simulator.

Assays were performed in order to meet a quite adequate solution for this optimization problem, taking into account several parameters: the number of generated vectors K, the elite number, and the noise value. This last factor seems to have a lot of influence on the convergence of the learning process, exactly as occurred with the exploration rate of the previous cases. Hence, it is faced again the exploration/exploitation problem, which seems to persist regardless of the circumstances. Note that while the covariance matrix remains not null (i.e., in the beginning of the execution where it is equal to an identity matrix), some exploration is always included, which is translated in the generation of different vectors. However, it will sooner or later converge to a null matrix, and the K generated vectors will be all the same. That is why the noise value is so important, propelling the continuity of the execution.

To address these problems, it is intended to answer the following set of questions: *what is the best way of managing the noise value*? *Is the addition of this noise value*

worthwhile to achieve better results? How many vectors should be generated in each round? And how many should participate in the calculation of the new distribution?

6.3 - Results

Throughout the implementation and analysis processes, it was noticed that there are two parameters that can influence a lot the results: the noise value and elite number. For this, the algorithm was tested with and without a noise value, to confirm the problem of premature convergence, and to what extent the solution achieved with exploration is better and remote from the noise-free one. It was also tested a set of elite numbers, for each of the two sets of noise tests.

The following describes the results for the various tests carried out. The results are presented in terms of a cost to minimize rather than a reward to maximize as occurred in the previous implemented solutions. In fact, a cost corresponds to a negative reward ($cost = -distance$).

Without noise

As foreseen, it was verified premature convergence. In all evaluated testes, what happens is that the covariance matrix will soon stabilize and converge to zero, which leads to no exploration. As a consequence, the execution reaches a point where the K vectors generated are all the same, and the parameters stabilize.

If only the best vector is chosen for the elite group, after one round the algorithm has converged, because the covariance of the new distribution is 0. This fact makes this number a poor choice, when no noise is added to the covariance matrix. The distance attained by this approach was near 0.07 unities of distance (u.d.), while with the original parameters it was only around 0.0036 u.d..

Values {2,3,4} were also subject of study, and the costs achieved are exposed in Figure 6.2. The results evaluation is made resorting to the minimum cost achieved.

The cost of the set of elite numbers {3,4} is minimized to a certain point, but soon these two approaches lead to consecutive falls of the robot, from round 3 and 25, respectively, not achieving a result better than 0.16 (round 11) and 0.07 u.d. (round 3). Using an elite number of 2, CEM was capable of accomplishing better results, performing a trial with the achievement of approximately 0.33 u.d. (round 6), with no falls. The cost of this approach, as well as the parameters, tends to converge.

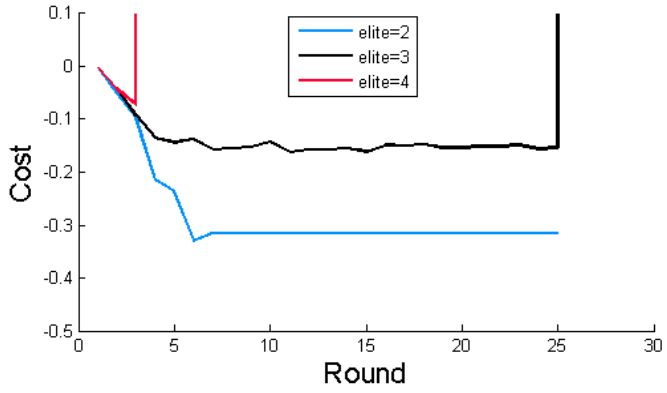


Figure 6.2 - Costs variation: without noise and elite numbers: 2 (in blue), 3 (in black) and 4 (in red). Vertical lines (round 3 and 25) represent a robot fall.

With the experiments carried out, it is straightforward to conclude one of the disadvantages of the CEM algorithm. As the parameter update is a vector of the mean of the best vectors, one cannot be assured that this set of parameters is going to achieve a good result, since it was not tested before. Only the elite vectors used to calculate the mean were tested and with guaranteed results. As a result, the robot can travel a shorter distance, or it may even fall if the mean parameters do not result as a good and operational set. This last graphic (Figure 6.2), for example, translates this situation, where the mean of 4 different good sets results in the fall of the robot, in round 3. As it was mentioned before, slight changes of joint positions can lead to major changes in the resulting movement, which can cause these disparities of the mean vector results. The parameters of the policy are dependent among them. As they are not independent, they are combined to work as a whole, and the change of values of one parameter no longer ensures the correct functioning of the chosen set.

It is verified that the selection of elite number = 1, does not allow an exploration and the consequent achievement of different sets of parameters, that could accrue in better results. On the other hand, choosing the elite number of 4 vectors is also not a good alternative. It includes excessive exploration, which may cause the deviation from the main purpose, in contrast to what is desired. It should be recalled one of the great issues of Reinforcement Learning, also valid for CEM, that is the tradeoff between exploration and exploitation. And as one parameter update is reflected in the mean vector of the chosen vectors, which was not tested before, it is convenient not to include too much diversification, or more likely it would fail the experiments that make use of

the new mean vector. Note that it can also result in better outcomes that could not have been reached otherwise. Nevertheless, from the carried out experiments, the bad outcomes are more frequent, and the greater the variability introduced in the mean vector (elite number = {3,4} in contrast with {1,2}), more distorted the results can be.

Despite the results and handicaps described herein, the CEM algorithm has found to be able to enhance the hand-tuned solution, in view of the chosen reward function. The best value achieved for the distance traveled was around 0.33 u.d. (for elite number = 2), against the value of 0.0036 u.d. from the initial solution, which is nearly 100 times more than the hand-tuned solution. A summary table (Table 6.3) is displayed after analysis of all tests carried out, to allow the comparison of the several obtained results.

Subsequently, it will be checked whether this best solution has come from a premature convergence. An exploration noise was added to the diagonal of the covariance matrix, in order to experiment a wider range of possible solutions.

With noise

After some tests to the noise value, a value of 0.8 was included to allow more exploration and in order to overcome the premature convergence problem. The chosen amount of noise also avoids the great apartness from the mean values, in order to not include further variability.

As with the experiments without noise, the set of elite numbers to test consists in the values {1,2,3,4}.

The solution with elite = 1 is now feasible, comparatively to the solution without noise, since both the cost and the parameters do not stabilize immediately, due to the noise added. A value of approximately 0.23 u.d. was reached for the distance traveled. However, when comparing to the best solution achieved so far (elite number = 2, no noise, distance = 0.32), this approach does not seem a good option in terms of optimization. A very similar cost was accomplished with elite number = 4. This value for the elite has revealed to be a poor alternative, since it provides excessive exploration arising from noise introduction plus the high elite number chosen. So, the optimization process leads to robot falls. It is interesting to notice that in the previous implementations (without noise), these two approaches (elite = 1 and 4) also achieved similar minimum costs.

A reasonable distance was performed by the robot when CEM algorithm applies the value 3 to elite number, of approximately 0.39 u.d.. But as in the approach using value 4, excessive exploration is also included, converging to consecutive falls. Note that either value 3 or value 4 accomplished better results with noise addition than they achieved with the absence of this rate.

Both implementations, with/without noise, achieved the best solution with elite number = 2. The problem of premature convergence was solved, and it is now possible to attain a distance of about 0.55 u.d. (Figure 6.3), which is the best achieved so far (in round 42). The problem of the mean among the best vectors remains in this implementation, as expected. Therefore, and as it is included more exploration due to the noise value, some rounds result in robot falls, unlike what occurred with the same elite number, without noise. Table 6.3 at the end of this section displays the summary of the results achieved.

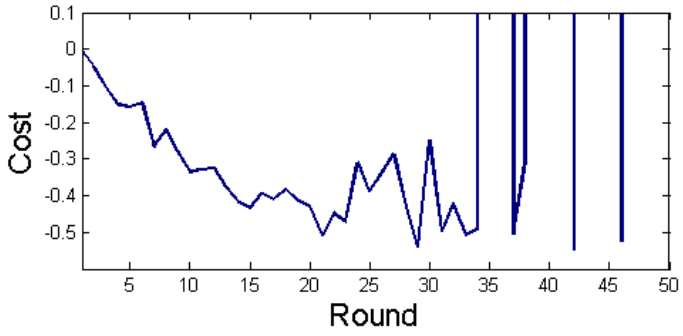


Figure 6.3 - Costs variation: with noise = 0.8 and elite = 2. Vertical lines (e.g., round 34) represent a robot fall.

For the elite number = 2, and for evaluation purposes, it was tested different exploration noise values. The value 1 is not as convenient as 0.8, because it introduces an excessive amount of exploration, which leads to worse results. The noise value of 0.2 slightly improves the rewards achieved during the execution process, but the noise introduced was unable to exceed the 0.8-noise rewards. Furthermore, the convergence to its best value was slower than with a noise value of 0.8.

So, for the best solution and vector achieved (noise = 0.8; elite = 2; distance = 0.55), it is next presented the set of parameters achieved, as well as the hand-tuned initial values (Table 6.1).

Table 6.1 — Achieved parameters: noise = 0.8; elite = 2; distance = 0.55

Parameters	Hand-tuned values	CEM values
$A_{balancingHip}$	11.0	2.979604
$A_{balancingAnk}$	11.0	4.339130
A_{Hip}	20.0	29.370636
A_{Knee}	40.0	35.984702
A_{AnkHip}	20.0	15.705181
$A_{AnkKnee}$	40.0	23.451423
$A_{flexKnee}$	2.0	2.894878
$A_{flexAnk}$	2.0	5.565523
$A_{pelvisRot}$	0.0	23.346823
$A_{compassHip}$	0.0	6.059359
$A_{compassAnk}$	0.0	1.057788

It is straightforward to verify that some values differ considerably from the ones hand-tuned. But once more it is highlighted that the parameters set has to work as a whole. If one parameter is changed, the solution's functioning is not guaranteed.

Reward with penalization

It was verified that some of the parameter sets which result in better outcomes in terms of the distance traveled, led the robot to deviate from the frontal axis that it was supposed to follow (z axis). In spite of the high distance value that can be achieved with this approach, the fact of the robot deviating sideways may be evidence of clumsy movements. Hence, it is introduced a penalization for any lateral displacement in the cost calculation. Figure 6.4 presents the results achieved with this new reward model, for the best solution achieved previously (noise = 0.8 and elite = 2).

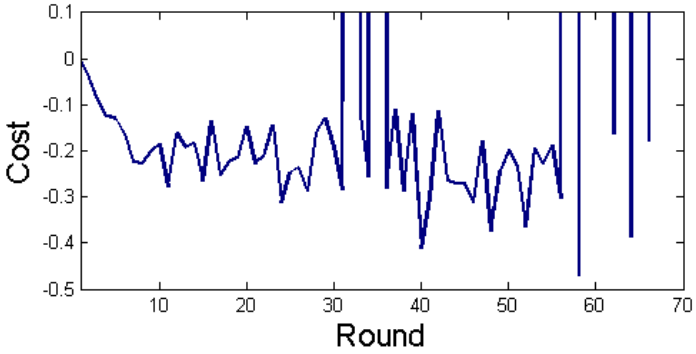


Figure 6.4 - Costs variation: with penalization, noise = 0.8 and elite = 2. Vertical lines represent a robot fall.

The hand-tuned solution led to a very small distance, about 0.0009 u.d.. The best displacement achieved was approximately 0.47 u.d. (round 58), therefore it is safe to

state that the optimization process was quite efficient. Once more, due to the introduced exploration and as there is no guarantee of mean vector functioning, some rounds result in the fall of the robot.

Generating more vectors

One way to accomplish an improvement in the optimization process is to increase the number of generated vectors, K, in the first step of the CEM algorithm, which would require far more computational resources. Note that runtime is proportional to the vectors chosen to be tested. Increasing the number of vectors for each round, the execution will last more. Thus, for the elite number 2 and noise value 0.8, and with no lateral displacement penalization due to evaluation purposes, different vector numbers were experimented.

For 20 vectors generated in each round, the algorithm soon found very good cost values. Although the execution takes longer to meet its optimal displacement, due to the need of testing 20 trials in each round, fewer rounds are required to meet reasonably good cost values. For example, to accomplish the same displacement as in K=10, 0.55 u.d., here it is only required a total of 8 rounds, as opposed to the previous 42 rounds. As each round with 20 vectors takes twice the time of each round with 10, it can be stated that less time was now needed to achieve those 0.55 u.d.. Furthermore, this approach was able to reach higher values than previously, getting a maximum distance of approximately 1 unity of distance (round 40). So even spending more resources, this option has revealed more reasonable than the former. The costs evolution is exposed in Figure 6.5, where it is verified that despite the high values of distance, this approach also resulted in more falls than the original.

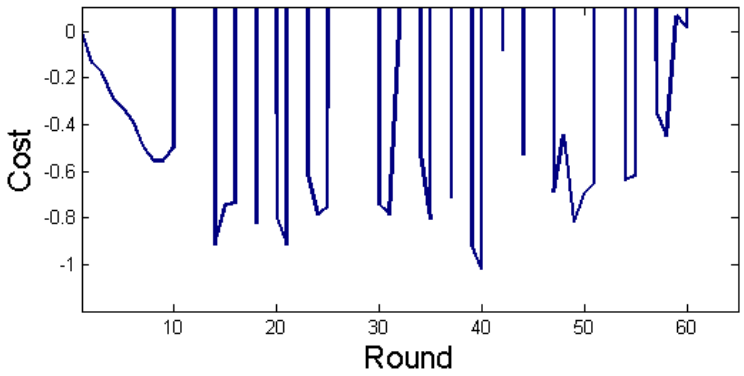


Figure 6.5 - Costs variation: with K = 20, noise = 0.8 and elite = 2. Vertical lines represent a robot fall

Towards these results, it would be interesting to comprehend how the algorithm would behave if the value of produced vectors was further enlarged. For this propose, an assessment of 50 trials per round was executed, each round requiring a 5 times longer runtime than in the original solution. Therefore, the execution took even more time (see Table 6.3) due to this higher number of trials to test, but a best displacement was achieved, of about 1.78 u.d. (in round 42). This minimum cost differs significantly from the one achieved in 20 trials, thus it turns out to compensate the expenditure of the additional resources required. These results can be checked in Figure 6.6.

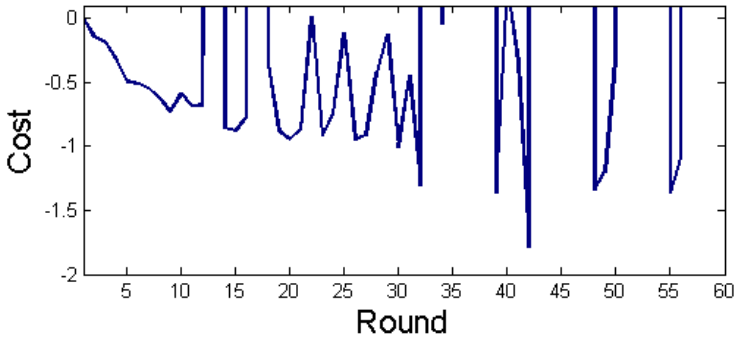


Figure 6.6 - Costs variation: with K = 50, noise = 0.8 and elite = 2. Vertical lines represent a robot fall

The set of parameters that achieved this great displacement is shown in the following table (Table 6.2).

Table 6.2 — Achieved parameters: K = 50; noise = 0.8; elite = 2; distance = 1.78

Parameters	Hand-tuned values	CEM values
$A_{balancingHip}$	11.0	3.886218
$A_{balancingAnk}$	11.0	12.091291
A_{Hip}	20.0	28.796737
A_{Knee}	40.0	33.628418
A_{AnkHip}	20.0	29.245331
$A_{AnkKnee}$	40.0	38.766492
$A_{flexKnee}$	2.0	7.986102
$A_{flexAnk}$	2.0	0.801668
$A_{pelvisRot}$	0.0	5.145981
$A_{compassHip}$	0.0	25.679888
$A_{compassAnk}$	0.0	18.227246

The robot steps are now larger and it is easy to observe the movement differences achieved with this approach, in comparison to the hand-tuned solution, or even to the K = 20 approach. (see videos (Duarte 2012)).

Figure 6.7 shows the robot walking forward, where it can be verified how large the steps are. In order to compare the differences of step length, and consequently, the displacement achieved, Figure 6.8 shows the robot at the end of the swing phase, for these three approaches with elite = 2 and noise = 0.8: K = 10 (Figure 6.8 b)), K = 20 (Figure 6.8 c)) and K = 50 (Figure 6.8 d)), along with the original hand-tuned solution (Figure 6.8 a)).

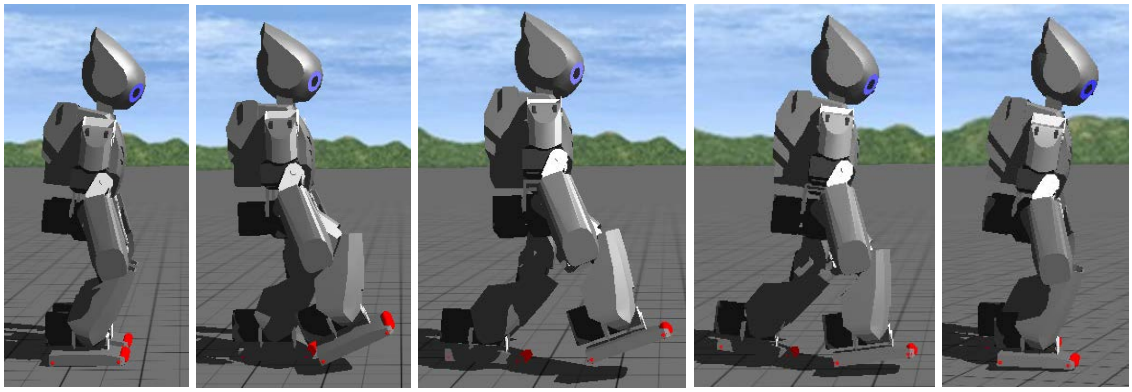


Figure 6.7 – DARwIN-OP robot walking forward, with K = 50, noise = 0.8 and elite = 2

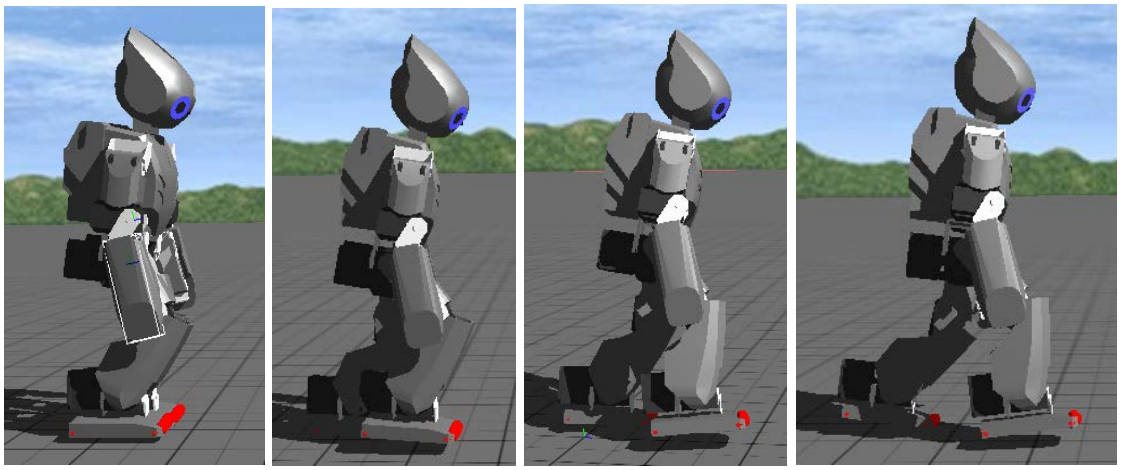


Figure 6.8 – Comparison between the various approaches: **a)** original hand-tuned. **b)** K = 10, noise = 0.8 and elite = 2. **c)** K = 20, noise = 0.8 and elite = 2. **d)** K = 50, noise = 0.8 and elite = 2.

But *what is the impact these visual differences have in joints trajectory*? This is an important point to analyze, and the Figure 6.9 illustrates the 6 left leg joints, for the hand-tuned (in blue) and the best solution achieved in the K = 50 approach (in green). The trajectories of the right leg are exactly the same, but dephased.

As verified in Table 6.2, the optimized amplitude of ankle roll (middle left panel) is very close to the hand-tuned one. This fact is reflected in Figure 6.9, when analyzing ankle roll trajectory. The same happens with hip roll joint (top left panel), where the

amplitude found takes lower values than the originals, or with hip yaw joint (bottom right panel), whose amplitude is now not null. The remaining trajectories translate the sum of various motion primitives, in which the ankle (top right panel) and hip pitch (bottom left panel) joints achieve very different results.

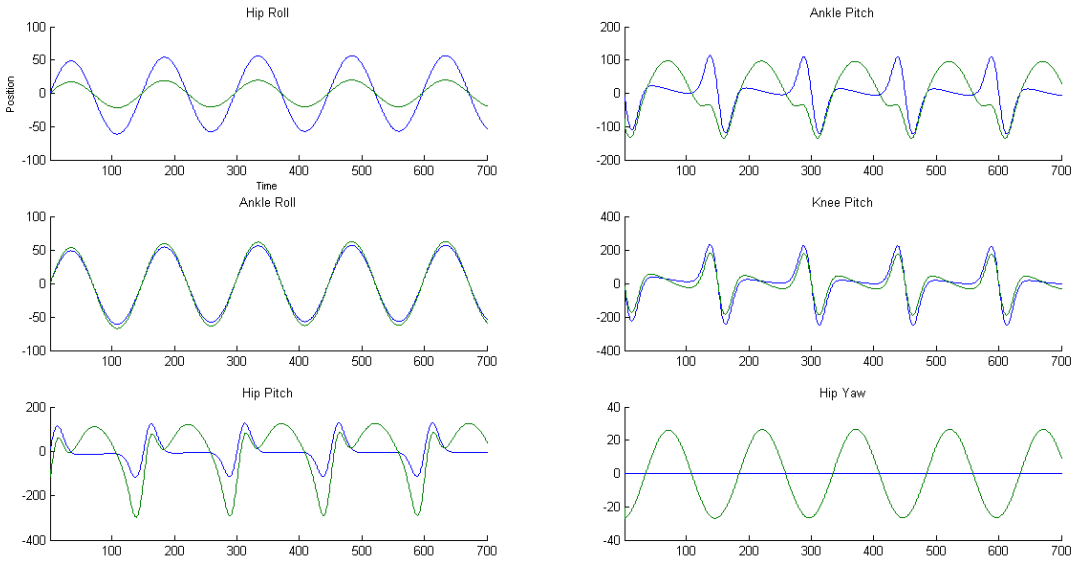


Figure 6.9 – Differences in joints trajectories between the optimized (in green) and the original (in blue) CPGs

As the results seem to improve with the increased number of trials, further tests were added to the set of experimented ones. However, increasing the number of generated vectors beyond 50 revealed not as good solutions. Testing 100 or even 70 trials leads to a great consumption of resources which unveiled to not compensate the results achieved.

The vectors are generated within a very restrict interval. Furthermore, the generation of solutions is much guided since the algorithm follows a normal distribution based in the best vectors achieved. However this point leads to a faster convergence to the solution, this can also mean a premature convergence in a local minimum, quite depending on the amount of noise added to the covariance matrix. Still, CEM attested to be an algorithm that quickly provides good solutions and a great optimization for this specific case, being a good choice despite the few disadvantages aforementioned. A great advantage of this method is the automatic learning and adaptation of the covariance matrix, as its definition is only necessary in the first round.

Table 6.3 synthesizes the results attained in the implemented set of tests. First row of the table represents the original hand-tuned solution, so the CEM parameters are not set. The results are presented for K = {10,20,50} for the approaches with (0.8) or without noise, and elite numbers = {1,2,3,4}. The optimal distance achieved in each of the executed tests is shown in the 4th column of the table, and the round in which this distance occurs is also depicted in order to provide a comparison of the several approaches, regarding the runtime of the executions. For example, if 10 vectors are being tested (K = 10), each round of tests lasts a time of *t* (Time/Round), which means that achieving an optimal solution in round R (e.g., 42) makes a total time (Runtime) of R**t* (e.g., 42*t*).

Table 6.3 — Results synthesis of the several tested approaches. First row represents the original hand-tuned solution.

K	Noise	Elite	Optimal Distance	Time/round	Round	Runtime
-	-	-	0.003596	-	-	-
10	No	1	0.068321	t	2	2t
		2	0.328687	t	6	6t
		3	0.161517	t	11	11t
		4	0.072173	t	3	3t
	0.8	1	0.234469	t	7	7t
		2	0.549083	t	42	42t
		3	0.390968	t	23	23t
		4	0.235189	t	6	6t
20	0.8	2	1.019418	2t	40	80t
50	0.8	2	1.782815	5t	42	210t

In conclusion, there are no rules that dictates which are the best values and parameters that should be used, as they are highly dependent on the circumstances in which they operate. To find out the best way of obtaining optimal results it is required an exhaustive testing of different parameters or values.

It is important to notice that as the equations that provide the CPGs' basic motion remain the same along all execution, except for the phase, one single action is performed in the beginning of each trial. This means that no sequential decision-making is required, and thus CEM is suitable for solving this Reinforcement Learning problem.

The chosen reward has allowed a faster locomotion, since the robot can now travel a greater distance for unit of time than before. However, some additional features to the

reward function could lead to better results, concerning the naturalness of the movements, and the higher similarity to human locomotion.

Answering the questions that remained outstanding in the previous section, the noise value is very important to avoid premature convergence and the results achieved with this addition were far better than without any addition. 50 revealed to be a good number for the generation of vectors and the selection of just 2 was always the best option obtained, independently from the approach used. For the noise value, supervision was required to observe the behavior of the algorithm with several values tested. After a few experiments, it was encountered a good value, capable of handling the tradeoff of exploration/exploitation.

At this point of the thesis is now possible to conclude about some intriguing issues exposed in section 1.2. The hand-tuned parameters provide a very good movement for a starting point in the learning process. However, considering the distance traveled, this solution could be further optimized. So, the optimization proposed evolved from the previous original one, and proved to be very worthwhile when examining the robot's evolution. CEM was found to be very suitable to solve the problem at hand, particularly, with the used CPGs, which revealed a few differences with the literature approaches. In contrast to the Natural Actor-Critic solution, this integrated approach resulted as desired, approaching the proposed goals.

Strong advantages arise from the use of this integrated approach, CEM with CPGs, since few parameters are used for the optimization (only 11) and there is a well-defined movement already supplied by the CPGs themselves. If no CPG was used, the possibility of generating rhythmic and continuous movements would become harder and the movements would not be natural and defined. This is very important, as a set of hand-tuned parameters is essential for the beginning of the optimization, to provide a starting point for the learning algorithm. Moreover, if a comparison with human learning is made, it can be verified that babies already have a pre-defined and intuitive movement of their legs, which needs to be enhanced to perform walking.

Another advantage is the possibility of existing non-linearity in the policy parameters, in contrast to the majority of algorithms suitable for CPGs optimization. When using DMPs, there is a larger amount of parameters to update. For example, in NAC solution, there were 15 parameters for each joint of the robot, which can lead to a heavier and slower process.

As in all approaches, not everything is an advantage. In this case, a parameter change can be very sensible and can result in a very different movement (or a fall). In DMPs approach, these changes seem not so sensible and more noise is needed to explore all possibilities. Though, this was considered a drawback of minor importance.

The results presented can be seen via video in (Duarte 2012).

With regard to Webots, this phase was helpful to notice an apparent disadvantage of this simulator, as it seems to not be capable of resetting both the robot and environment in order to execute all roll-outs reliably, without noisy trials. The reset had to be complete, including the controller, which required a lot of writing and reading of files, to guarantee that the values of all variables needed throughout the execution would not be lost. Apart from this flaw, the simulation revealed to be very advantageous taking into account the amount of trials performed to achieve the desired goal. Such experiments on the part of a real robot would take much longer and require further assistance, quite apart the possibility of damaging the robot with so many falls.

6.4 - Summary

In this chapter it was demonstrated the functioning of the Cross Entropy Method and results were evaluated. The solution discussed proved to be capable of achieving a quite efficient optimization, since the maximum distance attained was vastly superior than the one traveled by the robot with a CPG-only approach. It is expected that the experience gained in the implementation and testing of this algorithm will be useful and capable of guiding future experiments.

This phase was very important, although it did not contribute directly with a solution of classic Reinforcement Learning. Nevertheless, it achieved an optimization and provided a familiarization with some useful concepts shared by RL methods, like PI^2. In fact, these two algorithms revealed to be very similar, as stated in 6.1, and CEM was even successfully applied to RL tasks, both in the literature and in the current thesis work. Furthermore, it contributes with a solution whose application was not encountered in the literature.

A conclusion can be extended to PI^2, since it can also be expected to achieve parameter vectors that could not work, due to the weighted average used for their calculation.

Chapter 7

Conclusions

Throughout the previous chapters, Reinforcement Learning was studied and implemented, providing knowledge and experience for carrying out the latest and the future experiments, and deciding about the viability of following some conjectured paths. In this chapter a results discussion is presented, focusing on the main conclusions drawn from this thesis work, including the advantages and drawbacks of the used RL techniques along with the designed techniques.

Despite the achievement of a good solution at the end of the project, some issues remained to be further explored. These issues will be discussed in the present chapter, for a potential inclusion in future works.

7.1 - Discussion

7.1.1 Objectives discussion

From the carried out study and investigation, it is concluded that Reinforcement Learning is a very useful technique, suitable for learning problems that require a lot of interaction with the environment, and which do not support the provision of a complete supervision. It can be applied in a wide range of applications, including locomotion in Robotics, a field which has been widely addressed in recent years.

Recalling the first intermediate goal defined in 1.3, a state-of-the-art was successfully gathered, covering the problem of learning and optimization of robotic locomotion with the use of Reinforcement Learning, that despite being derived from ancient principles and making use of not so recent frameworks, it is currently the

subject of study of many investigators, leading to the frequently emergence of developments and new approaches and techniques.

Also, knowledge and experience were achieved, in order to provide a greater comprehension of RL framework and functioning, which revealed much utility in the latest project stages. However, as some RL issues were found to be very dependent of the circumstances, further experience will be required in order to more easily make decisions accordingly, in future case studies.

As proposed, experiments based on the state-of-the-art were carried out in order to better understand RL. Thus, the Crawler and Inverted Pendulum case studies were successfully solved by Reinforcement Learning techniques, allowing a greater familiarization to the algorithms and the Webots environment. With this familiarization, it was possible to idealize a RL solution considering the use of Central Pattern Generators, which were studied and compared to other state-of-the-art approaches. Due to the non-desired results of the application of Natural Actor-Critic, a second solution was designed, based on CEM, an algorithm that shares very similar concepts to some RL methods. Hereupon, the objectives were all addressed and successfully solved. Furthermore, the two case studies resulted in a publication in ICNAAM conference.

7.1.2 Conclusions from the developed work

Webots simulator revealed advantages against the use of real robots, whose tests would take much longer and which would require further assistance, quite apart the possibility of damaging the robot with so many falls.

Another advantage from using Webots is related to the possibility of simulating environments as close as possible to reality, where physic issues must be taken into account. In Inverted Pendulum case study this advantage was strongly perceived, due to the major role that gravity has. Such issues are all automatically included in the simulation, which facilitates a process that otherwise would have to be addressed, lying outside the scope of this dissertation. Furthermore, for implementing the solution of biped locomotion in DARwIN-OP, Webots revealed great utility, featuring a complete dynamics model of the robot, similarly to the real one. However, the simulator has also a weakness as it seems not capable of resetting both the robot and environment in order to execute all roll-outs reliably, without noisy trials.

From the case studies and solutions implemented, it can be concluded that there are several factors that influence the achievement of good results. First, it is very important

to have an appropriate reward function, or otherwise, the results could be contrary of the desired ones. Sometimes, rewards are very hard to achieve, and that is one of the great challenging issues in Reinforcement Learning area. For example, an agent whose task is related to playing a game will receive a good reward if it wins the game, and a bad one in case of losing it. In these cases, it is important to understand which particular moves lead to the loss of the game, taking into account every single action performed by the agent. Experiments related to the Crawler case study demonstrate that a big penalization of bad moves may take the robot to stand still, opting for the actions that lead to a reward of zero, and not resulting in the desired displacement. Also, the first conceived solution based on Natural Actor-Critic, revealed, upon the results interpretation, that the reward model was not correctly chosen. What occurred in this case was that no matter how the solution is worsened, the rewards achieved are always around the same values, which reveals that they do not reflect enhancements or worsenings. It was found a difficult task to select an appropriate reward to address the balancing joints of the robot.

The discount factor γ, and the learning and exploration rate, α and ε, are three other influence factors in the results achieved, and are also considered as challenging issues in RL problems.

If γ value is low and little influence is exerted by the future rewards, the algorithm converges faster, but on the other hand, it may not reach the optimal cycle. A quite acceptable value valid for all the experiments performed is 0.9. While Crawler experiments revealed much sensibility to this parameter, not allowing a value less than 0.8 in order to achieve an optimal policy, Inverted Pendulum works with a wider range of values. The latter does not require as much information on future rewards as the former, because in Crawler it was found greater dependence on subsequent actions.

The learning rate α, which is not a requirement for all algorithms, translates the changes in Q-values. Low values for the parameter α, reflected minor changes in the values of Q, while high values (α=1) reflect more abrupt changes. This point revealed influence on the oscillations and stability of the pendulum in the long term. However, it must be noticed that the learning period should be extensive enough so that it is possible to verify the differences within the range of learning rate values.

Regarding the exploration rate, this was handled differently, according to the problem under study. The Crawler results showed that the exploration rate must be

managed in such a way that it is ensured that all possibilities are tried out in order to wisely select the greediest actions. So, a high learning rate was assigned in the beginning of the execution, decreasing over time. The Cross-Entropy-based solution also revealed sensibility to this added noise, as it prevents premature convergence. But if it is assigned a high value to this rate, too much deviation from the original solution is verified. In this case this high deviation is a drawback due to the sensibility of the CPGs parameters: minor changes of joint positions can lead to major changes in the resulting movement.

This last statement translates a disadvantage of CEM, as the resulting parameters are the mean of two or more vectors, which does not guarantee the properly displacement of the robot, with the possibility of resulting in a fall.

Apart from the mentioned challenging issues, Reinforcement Learning reveals some other bottlenecks, which are still difficult to solve, and which have been approached in recent works. In the real world, the Markov property is not always met and spaces and actions are almost always continuous and not discrete. With regard to robot locomotion and Central Pattern Generators, there are very few approaches suitable to address the problem at hand. In fact, the available techniques for the integration of RL and CPGs proved to be unsuitable, due to the differences encountered between the provided CPGs and the ones implemented in the literature.

The DMPs approach was considered in Chapter 5, by applying a Natural Actor-Critic technique to update the DMPs parameters. This required an adaptation of the basic CPGs, which do not result in any optimization. For this reason, the proposed problem was tackled differently, using both a different solution and a different algorithm.

CEM was then very capable of achieving a quite efficient and quick optimization, since the maximum distance attained was vastly superior than the one traveled by the robot with a CPG-only approach. This integrated approach is a good method of tackling robot locomotion problems, as strong advantages were verified. First, a well-defined movement is already supplied by the basic CPGs which allow the generation of rhythmic and continuous natural movements. Second, only 11 parameters are used in the optimization process, against the large amount of parameters in the DMPs approach, which in the implemented solution, included 15 parameters for each joint. Another advantage of applying CEM to CPGs is the possibility of existing non-linearity in the policy parameters, in contrast to the majority of algorithms suitable for CPGs

optimization. However, it must be noticed that CEM did not consider sequential decision-making. In this specific problem, this was not required, as the phase-varying equations that provide the CPGs' basic motion remain the same along all execution. So, one single action is performed in the beginning of each trial, as opposed, for example, to the Crawler, where at least two sequential actions are required to achieve a forward step.

This solution was a major contribute of this thesis, as it was not encountered works with similar aims of this thesis, e.g., robot locomotion using CPGs.

Apart from these continuous states and actions approaches, some RL algorithms were compared for evaluation purposes. In Crawler case study, Value Iteration showed great potential, as the robot was able to found the desired movement with no information provided except the reinforcement signal. It was very suitable for this small dimension problem, because all value states are updated in each step of the simulation, in contrast with the other approaches. This fact can turn out to be a drawback in large state and action spaces problems, due to the heavy computation. In terms of convergence, it also showed great performance. On the other hand, TD(λ) has also shown very interesting results, providing a smoother and more natural locomotion, in which the Crawler enlarges its steps.

From all experiments carried out, it can be concluded that there are no rules that dictates which are the best values and parameters that should be used, as they are highly dependent on the circumstances in which they operate. To find out the best way of obtaining optimal results it is required a sound planning of a set of different tests. Nevertheless, experience is a very important factor in designing solutions, as it may contribute to accelerate and facilitate both the process of this first phase of conception, and the results interpretation.

7.2 - Future work

After the successful implementation of the solution in Webots simulator, it would be interesting to transfer to the real DARwIN-OP the achieved optimal policy, analyzing the differences between the real and simulated environments.

As stated when interpreting CEM results, additional features to the reward function could lead to better results, concerning the naturalness of the movements, and the higher similarity to human locomotion. This is a point that should be addressed, in order

to find suitable rewards to further optimize the human-like locomotion. Also, to invest in this successful approach, other parameters could be added to the optimization target, i.e., the offset and sigma of the CPGs equations.

Also concerning the rewards, it would be curious to validate the hypothesis raised upon the results analysis of Natural Actor-Critic solution. Namely, the viability of the solution should be verified, finding a more suitable reward function, even having to select other leg-joints for the target of optimization. If the solution turned out to be feasible, other algorithms could be tested for evaluation purposes, e.g., PI^2. In case of failure, DMPs could be used to approximate the provided CPGs to the state-of-the-art approaches, by replacing the original *sin* functions by the DMPs equations, with a properly set of hand-tuned parameters.

A paper is being worked out for a Journal submission concerning the final results of Cross-Entropy Method.

Appendix

Value Iteration Step-by-Step

This section pretends to provide a detailed explanation on Value Iteration algorithm. Namely, a step-by-step will be presented for the Crawler case study, in order to show how this algorithm proceeds, i.e., how the actions are selected or even how the values are calculated.

In section 4.1.1, it was conceived a first solution of learning resorting to a complete reward model, where Value Iteration shall be implemented with Table 4.1 as input.

Along with Table 4.1 for the reward model, it is also provided the discount factor γ, which was set to 0.9, the initial robot state (0,0), and the state-values matrix V, initialized with zeros (Table A.1).

Table A.1 — Initialized matrix of the state values

	0	1	2	3	4
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000	0.000000

Having the essential information for the execution of Value Iteration, the first two iterations are detailed:

Iteration 1:

- Crawler is in state (0,0).

- Possible actions: {*down, right*}. Which one is the greediest, i.e., has the greatest state value?
- Values Calculation for the actions of (0,0) (see Eq. 3.1):

  V(0,0) for action *down* = $R^{down}_{(0,0)(1,0)} + \gamma V(1,0)$ = 0 + 0.9*0.0 = 0.0

  V(0,0) for action *right* = $R^{right}_{(0,0)(0,1)} + \gamma V(0,1)$ = 0 + 0.9*0.0 = 0.0

  V(1,0) and V(0,1) can be consulted in the last state-values matrix, in Table A.1.
- As the values of the possible actions are equal, any action can be chosen, e.g., *down*, which causes a transition to state (1,0) with a reward of 0.
- As the value for, e.g., *down* was the greatest,

  V(0,0) = V(0,0) for action *down* = 0.0
- Values of the subsequent states are calculated in a similar way. Note that the entire matrix has to be updated. The new V matrix is shown in Table A.2.

Table A.2 — Matrix of the state values at the end of Iteration 1

	0	1	2	3	4
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000
2	1.000000	10.000000	9.000000	8.100000	7.290000
3	46.000000	59.400000	85.460000	105.914000	96.322600
4	-14.600000	-7.140000	28.914000	72.022600	107.820340

Iteration 2:

- Crawler is in state (1,0).
- Possible actions: {*up, down, right*}. Which one is the greediest, i.e., has the greatest state value?
- Values Calculation for the actions of (1,0):

  V(1,0) for action *up* = $R^{up}_{(1,0)(0,0)} + \gamma V(0,0)$ = 0 + 0.9*0.0 = 0.0

  V(1,0) for action *down* = $R^{down}_{(1,0)(2,0)} + \gamma V(2,0)$ = -1 + 0.9*1.0 = -0.1

  V(1,0) for action *right* = $R^{right}_{(1,0)(1,1)} + \gamma V(1,1)$ = 0 + 0.9*0.0 = 0.0

  V(0,0), V(2,0) and V(1,1) can be consulted in the last state-values matrix, in Table A.2.
- As the values of the two best actions (*up* and *right*) are equal, any action can be chosen, e.g., *up*, which causes a transition to state (0,0) with a reward of 0.

- As the value for, e.g., *up* was the greatest,

  V(1,0) = V(1,0) for action *up* = 0.0

- The new V matrix is shown in Table A.3.

Table A.3 — Matrix of the state values at the end of Iteration 2

	0	1	2	3	4
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	9.000000	8.100000	7.290000	6.561000
2	42.400000	63.460000	82.914000	94.322600	86.690340
3	35.160000	49.644000	76.679600	104.820340	113.038306
4	-24.356000	-1.977400	26.220340	69.598306	105.638475

The following iterations are executed in a similar way. The values are updated until they converge and the optimal policy is found.

References

Abbeel, Pieter, Morgan Quigley, and Andrew Y. Ng. 2006. "Using inaccurate models in reinforcement learning." *International Conference on Machine learning* 1–8.

Abramson, M., and H. Wechsler. 2003. "Tabu search exploration for on-policy reinforcement learning." Pp. 2910–2915 in *International Joint Conference on Neural Networks*, vol. 4. Ieee.

Arena, Paolo. 2001. "A Mechatronic Lamprey controlled by Analog Circuits." in *IEEE mediterannean conference on control and automation*.

Bagnell, J. Andrew, and Jeff G Schneider. 2001. "Autonomous Helicopter Control Using Reinforcement Learning Policy Search Methods." in *International Conference on Robotics and Automation*.

Barto, A. G., R. S. Sutton, and C. W. Anderson. 1983. "Neuronlike adaptive elements that can solve difficult learning control problems." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13(5):834–846.

Beitelspacher, Josh, Jason Fager, Greg Henriques, and Amy McGovern. 2006. *Policy Gradient vs. Value Function Approximation: A Reinforcement Learning Shootout.*

Bellman, Richard. 1957. *A Markovian Decision Process*.

Benbrahim, Hamid, and Judy A Franklin. 1997. "Biped dynamic walking using reinforcement learning." *Robotics and Autonomous Systems* 22:283–302.

Bhatnagar, Shalabh, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. 2009. "Natural actor–critic algorithms." *Automatica* 45(11):2471–2482.

Bjorkstrom, Anders B J. 2001. *Ridge regression and inverse problems.*

Boyan, Justin A. 1999. "Least-Squares Temporal Difference Learning." Pp. 49–56 in *Machine Learning: Proceedings of the Sixteenth International Conference*.

Bradtke, Steven J., and Andrew G. Barto. 1996. "Linear Least-Squares algorithms for temporal difference learning." *Machine Learning* 22(1-3):33–57.

Branislav, Borovac, Raković Mirko, and Nikolić Milutin. 2012. "Use of Support Vector Machine for Humanoid Robot Motion Synthesis." *The IPSI BgD Transactions on Internet Research* 18–25.

Buck, S., M. Beetz, and T. Schmitt. 2002. "Approximating the value function for continuous space reinforcement learning in robot control." Pp. 1062–1067 in *IEEE/RSJ International Conference on Intelligent Robots and System*. Ieee.

Busoniu, L., D. Ernst, B. De Schutter, and R. Babuska. 2011. "Approximate Reinforcement Learning: An Overview." in *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning*.

Busoniu, Lucian, Damien Ernst, Bart De Schutter, and Robert Babuska. 2005. "Continuous-State Reinforcement Learning with Fuzzy Approximation." in *European conference on Adaptive and learning agents and multi-agent systems: adaptation and multi-agent learning*.

Busoniu, Lucian, Damien Ernst, Bart De Schutter, and Robert Babuska. 2009. "Policy Search with Cross-Entropy Optimization of Basis Functions." in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

Chang, Hyeong Soo, Michael C Fu, Jiaqiao Hu, and Steven I Marcus. 2006. "Simulation-Based Algorithms for Markov Decision Processes."

Chernova, Sonia, and Manuela Veloso. 2004. "An Evolutionary Approach To Gait Learning For Four-Legged Robots." in *International Conference on Intelligent Robots and Systems*.

Ciancio, Anna Lisa, Loredana Zollo, Eugenio Guglielmelli, Daniele Caligiore, and Gianluca Baldassarre. 2011. "Hierarchical Reinforcement Learning and Central Pattern Generators for Modeling the Development of Rhythmic Manipulation Skills." in *IEEE International Conference on Development and Learning*.

Crespi, Alessandro, and Auke Jan Ijspeert. 2006. "AmphiBot II: An Amphibious Snake Robot that Crawls and Swims using a Central Pattern Generator." Pp. 19–27 in *International Conference on Climbing and Walking Robots*.

Crites, Robert H., and Andrew G. Barto. 1996. "Improving Elevator Performance Using Reinforcement Learning." *Advances in Neural Information Processing Systems* 8:1017–1023.

Duarte, Ana Filipa. 2012. "'Using Reinforcement Learning in the tuning of Central Pattern Generators' Videos." Retrieved (http://www.youtube.com/user/roboticsRL).

Duarte, Ana Filipa, Pedro Silva, and Cristina Peixoto Santos. 2012. "Reinforcement Learning: Solving Two Case Studies." in *International Conference of Numerical Analysis and Applied Mathematics*.

Fukuoka, Yasuhiro, Hiroshi Kimura, and Avis H Cohen. 2003. "Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain Based on Biological Concepts." *The International Journal of Robotics Research* 22(3-4):187–202.

Gams, Andrej, Auke J. Ijspeert, Stefan Schaal, and Jadran Lenarčič. 2009. "On-line learning and modulation of periodic movements with nonlinear dynamical systems." *Autonomous Robots* 27(1):3–23.

Gaskett, Chris, David Wettergreen, and Alexander Zelinsky. 1999. "Q-Learning in Continuous State and Action Spaces." in *Australian Joint Conference on Artificial Intelligence*.

Girgin, Sertan, and Philippe Preux. 2008. "Basis Expansion in Natural Actor Critic Methods." in *European Workshop on Reinforcement Learning*.

Gomez, Faustino, Jurgen Schmidhuber, and Risto Miikkulainen. 2006. "Efficient Non-Linear Control through Neuroevolution." in *European Conference on Machine Learning*.

Ha, Seungsuk, Youngjoon Han, and Hernsoo Hahn. 2007. "Adaptive Gait Pattern Generation of Biped Robot based on Human ' s Gait Pattern Analysis." *Proceedings of World Academy of Science, Engineering and Technology* 23:406–411.

Hansen, N, and A Ostermeier. 2001. "Completely derandomized self-adaptation in evolution strategies." *Evolutionary computation* 9(2):159–195.

Hasselt, Hado Van. 2012. "Reinforcement Learning in Continuous State and Action Spaces." in *Reinforcement Learning: State of the Art*, edited by Marco Wiering and Martijn van Otterlo. Springer.

Hasselt, Hado van, and Marco a. Wiering. 2007. "Reinforcement Learning in Continuous Action Spaces." Pp. 272–279 in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. Ieee.

Heidrich-meisner, Verena, and Christian Igel. 2008a. "Evolution Strategies for Direct Policy Search." Pp. 428–437 in *10th international conference on Parallel Problem Solving from Nature*.

Heidrich-meisner, Verena, and Christian Igel. 2008b. "Similarities and differences between policy gradient methods and evolution strategies." Pp. 149–154 in *16th European Symposium on Artificial Neural Networks*.

Hester, Todd, Michael Quinlan, and Peter Stone. 2010. "Generalized model learning for Reinforcement Learning on a humanoid robot." Pp. 2369–2374 in *IEEE International Conference on Robotics and Automation*. Ieee.

Hornby, G S, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. 1999. "Autonomous Evolution of Gaits with the Sony Quadruped Robot." in *Genetic and Evolutionary Computation Conference*.

Igel, Christian. 2003. "Neuroevolution for Reinforcement Learning Using Evolution Strategies." *Congress on Evolutionary Computation* 4:2588–2595.

Ijspeert, A.J., J. Nakanishi, and S. Schaal. 2002. "Movement imitation with nonlinear dynamical systems in humanoid robots." in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, vol. 2. Ieee.

Ijspeert, Auke Jan. 2008. "Central pattern generators for locomotion control in animals and robots: a review." *Neural Networks* 21(4):642–53.

Irodova, Marina, and Robert H Sloan. 2005. "Reinforcement Learning and Function Approximation." in *Florida Artificial Intelligence Research Society Conference*.

Ito, Kazuyuki, and Fumitoshi Matsuno. 2002. "A Study of Reinforcement Learning for the Robot with Many Degrees of Freedom." Pp. 3392–3397 in *International Conference on Robotics and Automation*.

Jacob, David, Daniel Polani, and Chrystopher L Nehaniv. 2005. "Legs that can walk : Embodiment-Based Modular Reinforcement Learning applied." Pp. 365–372 in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*.

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore. 1996. "Reinforcement Learning : A Survey." *Journal of Artificial Intelligence Research* 4:237–285.

Kalyanakrishnan, Shivaram, and Peter Stone. 2009. "An Empirical Analysis of Value Function-Based and Policy Search Reinforcement Learning." Pp. 749–756 in *International Conference on Autonomous Agents and Multiagent Systems*.

Kamio, Shotaro, Hideyuki Mitsuhashi, and Hitoshi Iba. 2003. "Integration of genetic programming and reinforcement learning for real robots." Pp. 470–482 in *International Conference on Genetic and Evolutionary Computation*.

Keith, Jonathan, and Dirk P Kroese. 2002. "Sequence alignment by rare event simulation." Pp. 320–327 in *Proceedings of the 2002 Winter Simulation Conference*.

Klaassen, Bernhard, Ralf Linnemann, Dirk Spenneberg, and Frank Kirchner. 2002. "Biomimetic walking robot SCORPION: Control and modeling." *Robotics and Autonomous Systems* 41(2-3):69–76.

Klopf, A. Harry. 1972. *Brain Function and Adaptive Systems- A Heterostatic Theory*.

Knobel, G P A. 2011. "Learning optimal gait parameters using the episodic Natural Actor-Critic method." Delft University of Technology.

Kober, Jens, and Jan Peters. 2008. "Policy search for motor primitives in robotics." *Advances in Neural Information Processing Systems*.

Kober, Jens, and Jan Peters. 2009. “Learning motor primitives for robotics.” *IEEE International Conference on Robotics and Automation* 2112–2118.

Kobilarov, Marin. 2001. “Cross-Entropy Randomized Motion Planning.” in *Proceedings of Robotics: Science and Systems*.

Kohl, N., and P. Stone. 2004. “Policy gradient reinforcement learning for fast quadrupedal locomotion.” *IEEE International Conference on Robotics and Automation* 2619–2624.

Konidaris, George, Sarah Osentoski, and Philip Thomas. 2011. “Value Function Approximation in Reinforcement Learning Using the Fourier Basis.” Pp. 380–385 in *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Kormushev, Petar, and Darwin G Caldwell. 2012. “Direct Policy Search Reinforcement Learning based on Particle Filtering.” in *European Workshop on Reinforcement Learning*.

Kormushev, Petar, Sylvain Calinon, and Darwin G Caldwell. 2010. “Robot motor skill coordination with EM-based Reinforcement Learning.” Pp. 3232–3237 in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee.

Lachat, Daisy, Alessandro Crespi, and Auke Jan Ijspeert. 2006. “BoxyBot: a swimming and crawling fish robot controlled by a central pattern generator.” Pp. 643–648 in *International Conference on Biomedical Robotics and Biomechatronics*.

Lizotte, Daniel J, Michael Bowling, and Susan A Murphy. 2010. “Efficient Reinforcement Learning with Multiple Reward Functions for Randomized Controlled Trial Analysis.” in *International Conference on Machine Learning*.

Mahadevan, Sridhar, and Jonathan Connel. 1991. “Automatic Programming of Behaviour-based Robots using Reinforcement Learning.” Pp. 768–773 in *AAAI Conference on Artificial Intelligence*.

Mannor, Shie, Reuven Rubinstein, and Yohai Gat. 2003. “The Cross Entropy method for Fast Policy Search.” in *International Conference on Machine Learning*.

Marin, Didier, Jérémie Decock, Lionel Rigoux, and Olivier Sigaud. 2011. “Learning Cost-Efficient Control Policies with XCSF : Generalization Capabilities and Further Improvement.” in *Genetic and evolutionary computation*.

Mataric, Maja J. 1994. “Reward Functions for Accelerated Learning.” in *International Conference on Machine Learning*.

Matos, Vitor, Cristina P. Santos, and Carla M.a. Pinto. 2009. “A brainstem-like modulation approach for gait transition in a quadruped robot.” *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* 2665–2670.

Matos, Vítor. 2009. “Bio-inspired quadruped omnidirectional locomotion: A dynamical system approach.” Universidade do Minho.

Matos, Vítor, and Cristina Santos. 2012. "Central Pattern Generators with Phase Regulation for the Control of Humanoid Locomotion." in *IEEE-RAS International Conference on Humanoid Robots*.

Matsubara, Takamitsu, Jun Morimoto, Jun Nakanishi, Masa-Aki Sato, and Kenji Doya. 2007. "Learning a dynamic policy by using policy gradient: application to biped walking." *Systems and Computers in Japan* 38(4):25–38.

Matsubara, Takamitsu, Jun Morimoto, Jun Nakanishi, Masa-aki Sato, and Kenji Doya. 2006. "Learning CPG-based biped locomotion with a policy gradient method." *Robotics and Autonomous Systems* 54(11):911–920.

Matsuoka. 1985. "Sustained Oscillations Generated by Mutually Inhibiting Neurons with Adaptation." *Biological Cybernetics* 52:367–376.

Menache, Ishai, Shie Mannor, and Nahum Shimkin. 2005. "Basis Function Adaptation in Temporal Difference Reinforcement Learning." *Annals of Operations Research* 134(1):215–238.

Michel, Olivier. 2004. "Webots TM : Professional Mobile Robot Simulation." *International Journal of Advanced Robotic Systems* 1:39–42.

Minsky, Marvin. 1961. "Steps towards artificial intelligence." *Computers and Thought* 406–450.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill.

Mori, Takeshi, Yutaka Nakamura, Masa-aki Sato, and Shin Ishii. 2004. "Reinforcement Learning for a CPG-driven Biped Robot." Pp. 623–630 in *National Conference on Artificial Intelligence*.

Moriarty, David E, Alan C Schultz, and John J Grefenstette. 1999. "Evolutionary Algorithms for Reinforcement Learning." *Journal of Artificial Intelligence Research* 11:241–276.

Morimoto, Jun, Gordon Cheng, Christopher G Atkeson, and Garth Zeglin. 2004. "A Simple Reinforcement Learning Algorithm For Biped Walking." Pp. 3030–3035 in *IEEE International Conference on Robotics and Automation*.

Muller, Heiko et al. 2007. "Making a Robot Learn to Play Soccer Using Reward and Punishment." *Advances in Artificial Intelligence* 220–234.

Nakamura, Yutaka, Takeshi Mori, Masa-aki Sato, and Shin Ishii. 2007. "Reinforcement learning for a biped robot based on a CPG-actor-critic method." *Neural networks* 20(6):723–35.

Nakamura, Yutaka, Takeshi Mori, Yoichi Tokita, Tomohiro Shibata, and Shin Ishii. 2005. "Off-Policy Natural Policy Gradient Method for a Biped Walking Using a CPG Controller." *Robotics and Mechatronics* 17(6):636–644.

Nakanishi, Jun et al. 2004. “Learning from demonstration and adaptation of biped locomotion.” *Robotics and Autonomous Systems* 47:79–91.

Ng, A. Y., and S. Russell. 2000. “Algorithms for inverse reinforcement learning.” Pp. 663–670 in *International Conference on Machine Learning*.

Niekum, Scott, Andrew G Barto, and Lee Spector. 2010. “Genetic Programming for Reward Function Search.” *IEEE Transactions on Autonomous Mental Development* 2(2):83–90.

Ogino, Masaki, Yutaka Katoh, Masahiro Aono, Minoru Asada, and Koh Hosoda. 2004. “Reinforcement learning of humanoid rhythmic walking parameters based on visual information.” *Advanced Robotics* 18(7):677–697.

Park, Jooyoung, Jongho Kim, and Daesung Kang. 2005. “An RLS-Based Natural Actor-Critic Algorithm for Locomotion of a Two-Linked Robot Arm.” Pp. 65–72 in *Iternational Conference on computational intelligence and security*.

Peters, J., and S. Schaal. 2006. “Reinforcement Learning for Parameterized Motor Primitives.” Pp. 73–80 in *IEEE International Joint Conference on Neural Network Proceedings*. Ieee.

Peters, Jan. 2007. “Machine Learning of Motor Skills for Robotics.” Uiniveristy of Southern California.

Peters, Jan, and Stefan Schaal. 2006. “Policy Gradient Methods for Robotics.” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* 2219–2225.

Peters, Jan, and Stefan Schaal. 2007. “Applying the Episodic Natural Actor-Critic Architecture to Motor Primitive Learning.” in *15th European Symposium on Artificial Neural Networks*.

Peters, Jan, and Stefan Schaal. 2008. “Natural Actor-Critic.” *Neurocomputing* 71(7-9):1180–1190.

Peters, Jan, Sethu Vijayakumar, and Stefan Schaal. 2003. “Reinforcement Learning for Humanoid Robotics.” in *Third IEEE-RAS International Conference on Humanoid Robots*.

Picado, Hugo. 2008. “Development of Behaviors for a Simulated Humanoid Robot.” Universidade de Aveiro.

Picado, Hugo, Marcos Gestal, Nuno Lau, Luis P Reis, and Ana M Tomé. 2009. “Automatic Generation of Biped Walk Behavior Using Genetic Algorithms.” Pp. 805–812 in *International Work-Conference on Artificial Neural Networks, Part I*.

Poole, David L., and Alan K. Mackworth. 2010. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press Retrieved (http://artint.info/).

Puterman, Martin L. 1994. *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Randløv, Jette, and Preben Alstrøm. 1998. "Learning to Drive a Bicycle using Reinforcement Learning and Shaping." in *International Conference on Machine Learning*.

Ravindran, B. 1996. "Solution of delayed reinforcement learning problems having continuous action spaces." Indian Institute of Science.

Rennie, Jason, and Andrew Kachites McCallum. 1999. "Using Reinforcement Learning to Spider the Web Efficiently." *International Conference on Machine Learning*.

Roy, Benjamin Van. 1998. "Learning and Value Function Approximation in Complex Decision Processes." Massachusets Institute of Technology.

Rubinstein, Reuven Y. 1997. "Optimization of computer simulation models with rare events." *European Journal of Operational Research* 99(1):89–112.

Rubinstein, Reuven Y. 1999. "The Cross-Entropy Method for Combinatorial and Continuous Optimization." *Methodology and Computing in Applied Probability* 1(2):127–190.

Rummery, G. A., and M. Niranjan. 1994. *On-line Q-learning using connectionist systems*.

Russell, Stuart J, and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc.

Rutishauser, Simon, Alexander Sprowitz, Ludovic Righetti, and Auke Jan Ijspeert. 2008. "Passive compliant quadruped robot using central pattern generators for locomotion control." Pp. 710–715 in *International Conference on Biomedical Robotics and Biomechatronics*.

Sammut, Claude, and Geoffrey I. Webb. 2011. *Encyclopedia of Machine Learning*. Springer.

Samuel, A. L. 1959. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development* 3(3):210–229.

Santos, Cristina Peixoto. 2004. "Generating timed trajectories for an autonomous vehicle: A non-linear Dynamical Systems Approach." Pp. 3741–3746 in *IEEE International Conference on Robotics and Automation*.

Sato, M., Y. Nakamura, and S. Ishii. 2002. "Reinforcement learning for biped locomotion." Pp. 777–782 in *International Conference on Artificial Neural Networks*.

Sato, T, K.Watanabe, and H Igarashi. 2010. “Acquisition of adaptive walking behaviors using machine learning with Central Pattern Generator.” in *International Joint Conference on Neural Networks*.

Schaal, S. 1999. “Is imitation learning the route to humanoid robots?” *Trends in cognitive sciences* 3(6):233–242.

Schaal, Stefan. 2003. “Dynamic Movement Primitives – A Framework for Motor Control in Humans and Humanoid Robotics.” in *International Symposium on Adaptive Motion of Animals and Machines*.

Schaal, Stefan, Evangelos Theodorou, Jonas Buchli, and Freek Stulp. 2010. “An Example Application of Policy Improvement with Path Integrals (PI2).”

Scherffig, Lasse. 2002. “Reinforcement Learning in Motor Control.” University of Osnabruck.

Shan, Jiang, and Fumio Nagashima. 2002. “Neural Locomotion Controller Design and Implementation for Humanoid Robot HOAP-1.” in *Annual Conference of the Robotics Society of Japan*.

Shieh, Ming-Yuan, Ke-Hao Chang, Chen-Yang Chuang, and Yu-Sheng Lia. 2007. “Development and Implementation of an Artificial Neural Network based Controller for Gait Balance of a Biped Robot.” Pp. 2778–2782 in *Annual Conference of the IEEE Industrial Electronics Society*.

Singh, Satinder, and Dimitri Bertsekas. 1997. “Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems.” Pp. 974–980 in *Advances in Neural Information Processing Systems*.

Smart, W.D., and L. Pack Kaelbling. 2002. “Effective reinforcement learning for mobile robots.” Pp. 3404–3410 in *EEE International Conference on Robotics and Automation*, vol. 4. Ieee.

Smart, William D, and Leslie Pack Kaelbling. 2000. “Practical Reinforcement Learning in Continuous Spaces.” Pp. 903–910 in *International Conference on Machine Learning*.

Snel, Matthijs, Shimon Whiteson, and Yasuo Kuniyoshi. 2011. “Robust central pattern generators for embodied hierarchical reinforcement learning.” *2011 IEEE International Conference on Development and Learning* 1–6.

Strosslin, Thomas, and Wulfram Gerstner. 2003. “Reinforcement Learning in Continuous State and Action Space.” in *International Conference on Artificial Neural Networks*.

Stulp, Freek, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. 2010. “Reinforcement learning of full-body humanoid motor skills.” *IEEE-RAS International Conference on Humanoid Robots* 405–410.

Stulp, Freek, and Stefan Schaal. 2011. “Hierarchical reinforcement learning with movement primitives.” *IEEE-RAS International Conference on Humanoid Robots* 231–238.

Stulp, Freek, and Olivier Sigaud. 2012. “Path Integral Policy Improvement with Covariance Matrix Adaptation.” in *29th International Conference on Machine Learning*.

Sutton, R.S., and A.G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, Richard S. 1988. “Learning to Predict by the Methods of Temporal Differences.” *Machine Learning* 3:9–44.

Sutton, Richard S. 1997. “On the Significance of Markov Decision Processes.”

Sutton, Richard S. 1999. “Reinforcement Learning.” *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*.

Sutton, Richard S, David McAllester, Satinder Singh, and Yishay Mansour. 2000. “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” *Advances in Neural Information Processing Systems* 12:1057–1063.

Szepesvári, Csaba. 2010. “Algorithms for Reinforcement Learning.” *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4(1).

Tesauro, Gerald. 1992. “Practical issues in temporal difference learning.” *Machine Learning* 8(3-4):257–277.

Theodorou, E., J. Buchli, and S. Schaal. 2010. “A Generalized Path Integral Control Approach to Reinforcement Learning.” *Machine Learning Research* 11:3137–3181.

Theodorou, Evangelos, Jonas Buchli, and Stefan Schaal. 2010. “Reinforcement learning of motor skills in high dimensions: A path integral approach.” Pp. 2397–2403 in *IEEE International Conference on Robotics and Automation*. Ieee.

Thrun, Sebastian B. 1992. *Efficient Exploration In Reinforcement Learning*.

Tokic, Michel. 2010. “Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences.” *Advances in Artificial Intelligence*.

Tokic, Michel, Wolfgang Ertel, and Joachim Fessler. 2009. “The Crawler, A Class Room Demonstrator for Reinforcement Learning.” Pp. 160–165 in *International FLAIRS Conference*.

Tokic, Michel, Arne Usadel, Joachim Fessler, and Wolfgang Ertel. 2010. "On an educational approach to behavior learning for robots." Pp. 171–176 in *International Conference on Robotics in Education*.

Tomoyuki, Takita, Yoshiyuki Azuma, and Tomoshiro Shibata. 2009. "Acquisition of energy-efficient bipedal walking using CPG-based reinforcement learning." *IEEE/RSJ International Conference on Intelligent Robots and Systems* 827–832.

Ueno, Tsuyoshi et al. 2006. "Fast and Stable Learning of Quasi-Passive Dynamic Walking by an Unstable Biped Robot based on Off-Policy Natural Actor-Critic." *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* 5226–5231.

Wang, Yongjia, and John E Laird. 2007. "The Importance of Action History in Decision Making and Reinforcement Learning." in *International Conference on Cognitive Modeling*.

Watkins, C. J. C. H. 1989. "Learning from delayed rewards." Kings College.

Watkins, Christopher J. C. H., and Peter Dayan. 1992. "Q-learning." *Machine Learning* 8(3-4):279–292.

Whitehead, Steven D, and Long Ji Lin. 1992. "Reinforcement Learning in Non-Markov Environments." *Artificial Intelligence* 8.

Whiteson, Shimon, and Peter Stone. 2006. "Evolutionary Function Approximation for Reinforcement Learning." *Journal of Machine Learning Research* 7:877–917.

Whitley, Darrell, Stephen Dominic, Rajarshi Das, and Charles W Anderson. 1993. "Genetic Reinforcement Learning for Neurocontrol Problems." *Machine Learning* 13:259–284.

Wierstra, Daan, Tom Schaul, Jan Peters, and Juergen Schmidhuber. 2008. "Natural Evolution Strategies." Pp. 3381–3387 in *2008 IEEE Congress on Evolutionary Computation*. Ieee.

Witten, Ian. 1977. "An Adaptive Optimal Controller for Discrete-Time Markov Environments." *Information and Control* 34(4):286–295.

Zhang, Wei, and Thomas G. Dietterich. 1995. "A Reinforcement Learning Approach to Job-shop Scheduling." in *International Joint Conference on Artificial Intelligence*.