University of Girona

Department of Electronics, Informatics and Automation

Research Project

Reinforcement Learning Policy Gradient Algorithms for Robot Learning

Work presented by

Andres El-Fakdi Sencianes
Supervisor: Dr. Marc Carreras Pérez

Girona, June 2005

To my family and all my friends.

Acknowledgements

In the next lines, I would like express my gratitude to all the people that has helped me to finish this research project. I give my special thanks to my family. Thanks to my parents for believing in me, even though they don't understand what the hell I am talking about (I hope reading this project will help, or not). Many thanks to my brother Raul, my best friend. I give my special thanks to my director Marc, for his good advices and for pushing me hard to do my best. Don't stop! The PhD is around the corner! Thanks to my friend David, always by my side, giving his help when needed, doesn't matter the time (one o'clock in the morning and still here! Buf!). I must also thank my colleagues of laboratory, for all the happy moments that we have spent together: Palo, Emili, Bet, Marc, Guillem, Eduard, Lluís, Carles, Ela, Jordi, thanks to everybody!! Also remember the friends that tried another road and are no more with us: Rossell, Tesh, Revengas, Sidrol, Joan, Ricard, Madrenas, Charlie, Freddie...guys! see you on Friday! I also mention my friend Javier, a PhD student from the University of the Balearic Islands for the nice days spent during our research collaboration. Come soon! or better say: See you this summer!

I don't want to forget our beloved captain, always taking care of all us, looking for grants, government projects, clusters, subventions, MCYTs, CYCITs and who knows!!! Thanks for all Pere "Little boy". My very special gratitude to the invaluable support of Jordi Freixenet, Xevi Cufí, Albert Figueras, Pep Forest, Josep Tomàs, and to Christine for revising my english.

I finally want to express my gratitude to Montse, for her patience and comprehension at every moment. Thank you very much!

Contents

List of Figures

List of Tables

Glossary

AI	Artificial Intelligence
ANN	Artificial Neural Networks
ASD	Action Selection Dynamics
AUV	Autonomous Underwater Vehicle
DOF	Degree Of Freedom
DP	Dynamic Programming
DPS	Direct Policy search
FMDP	Finite MDP
MC	Monte Carlo
MDP	Markov Decision Process
NGPA	Neural Policy Gradient Algorithm
PGA	Policy Gradient Algorithms
POMDP	Partially Observable MDP
QL	Q-Learning
RL	Reinforcement Learning
RLP	Reinforcement Learning Problem
ROV	Remotely Operated Vehicle
SGA	Stochastic Gradient Ascent
SMDP	Semi MDP
SONQL	Semi-Online Neural-Q_learning
TD	Temporal Difference
UUV	Unmanned Underwater Vehicle
VAPS	Single Value and Policy Search

Chapter 1

Introduction

Our society moves with the speed of sound. A lot of things have changed since 1917 when the writer Karel Kapek used for the first time the word robot referring to a machine with humanoid aspect. Even the well known ideas of Isaak Asimov about robots and his three famous laws of robotics, where we can see feared men worried for controlling machines against a possible revolution, seem to be far away from reality. Nowadays, different kinds of machines are present in our life, helping us in our most common daily activities. Some scientists understand robotics as an applied science born from the "marriage" between computer science and tool-machines. A robot should be able to process and manage information rationally and automatically without human help. However, reality suggests us that we are still very far from such a technology. Although being trivial for the human intelligence, the capability of a machine to perceive the environment (*sensing*) and take a decision (*acting*) is a very difficult task for a computer even in very simple applications. Thus, the field of *Artificial Intelligence* (AI) has been applied during the last decades to autonomous robots to solve this problem.

This research project is concerned with the field of autonomous robots and the problem of action-decision. This project is based on the topics of *Reinforcement Learning* (RL) theory and *Direct Policy Search (DPA) Algorithms*. In particular, this work surveys the field of *Policy Gradient Algorithms* as a way to solve a RL problem (RLP). Throughout this chapter the main aspects which have conditioned this research project will be overviewed. First, the research antecedents will be presented in the beginning of this research project. Then, the goal of the thesis will be pointed out. Finally, the chapter will finish with the outline of this dissertation.

1.1 Previous Research Work

This work has been fulfilled in the *Computer Vision and Robotics, VICOROB* research group of the University of Girona, Spain. This group has been doing research in underwater robotics since 1992 (supported by several programs of the Spanish commission MCYT). The main contribution throughout the past years is the development of two *Unmanned Underwater Vehicles* (UUV). The first prototype, called GARBI, was developed in collaboration with the Polytechnical University of Catalonia. This vehicle was first conceived as a *Remotely Operated Vehicle* (ROV), but at the moment it is being modified to convert it into an *Autonomous Underwater Vehicle* (AUV), adapting it to new requirements. The second prototype, URIS, was fully developed in the University of Girona and was designed as an *Autonomous Underwater Vehicle* (AUV). Smaller and cheaper than its older brother, was constructed with the objective of having a quick experimental benchmark for testing sensors and software for future implementation on GARBI.

The design of an autonomous vehicle requires a solution for the action decision problem. A *Control Architecture* is the part of the robot control system in charge of making these decisions. The control architecture used in URIS uses a set of behaviors to propose the best movement at any given moment. Each behavior has a particular goal ("go to", "avoid wall",...) and contains a mapping between the state of the environment and the action set to accomplish this goal. Previous research work consisted on investigating some implementation aspects of the behavior-based architecture in order to improve the performance of the control system.

In this way, the use of learning algorithms to improve the efficiency of the behaviors was explored. A learning theory, *Reinforcement Learning*, was overviewed. The development a RL based algorithm able to achieve simple tasks and exhibit real-time learning capabilities led to the PhD. thesis of Dr. Marc Carreras [Carreras, 2003]. The proposed RL method was called *Semi-Online Neural Q-Learning* (SONQL) and it is based on a value function.

The research project presented in this document is a continuation of the work started with the RL-based behaviors. This work surveys an special issue in RL called *direct policy search methods* and its application to robotics, with special attention to those algorithms able to work on-line. Also, this research project studies successful applications of policy search methods in real robot systems over the past few years.

1.2 Goal of this Research Project

After the description of the research antecedents, the goal of this research work can be stated. The general purpose is summarized as:

"The study of policy gradient algorithms as an alternative approach to value based algorithms and its suitability to robotics"

The work proposed continues the research line started with the development of the SONQL algorithm for on-line robot learning. On one hand, it overviews the main aspects of reinforcement learning and analyzes the main features of value and policy algorithms as alternatives approaches to solve RLP in robotics. On the other hand, surveys the most important policy gradient algorithms, successful applications in robotics and, finally, shows some implementation details with two simulated experiments. To study the suitability of direct policy search algorithms for on-line robot learning is certainly the most important purpose of this dissertation.

The general goal of the research project can be divided into three more specific points:

Reinforcement Learning. This part is a general overview of RL, its main features and its field of application. *Value Function* algorithms are first presented as a common methodology used to solve RLP. However, recent studies show the possibility of achieving better results applying direct policy search methodologies instead of value based, specially when our robot has to fulfill a task in unknown environment. Thus, this part's objective is to focus on the topic of policy gradient algorithms.

Policy Gradient Methods. A survey among several policy gradient methods has been presented and studied in this central part of the project. Moreover, the most relevant successful applications of this methodologies to solve real robot problems using RL are commented. The objective of this part is the comprehension of these algorithms.

Experimental Results. One of the algorithms studied in the previous part is selected to carry out simulated experiments. The chosen method is simple and it allows a deep understanding of policy gradient algorithms. The approach was implemented for two different RL problems. Results and conclusions are extracted from these experiments.

1.3 Outline of the Project

The contents of this work can be divided into three parts. The *first* part overviews the field of Reinforcement Learning and details a comparison be-

tween value based methods and direct policy search algorithms (Chapter 2). In the *second* part, several policy search algorithms are analyzed (Chapter 3) and successful robotics applications of these methods are commented (Chapter 4). The *third* part proposes a neural policy gradient algorithm for its application to robotics (Chapter 5) and shows the obtained experimental results (Chapter 6). Finally, the research project is concluded and the future work, that will be followed to continue the PhD thesis, presented (Chapter 7). A brief description of each chapter is next presented.

Chapter 2 *Reinforcement Learning.* This chapter presents the theory of Reinforcement Learning. Two different RL methodologies are discussed: value function algorithms and direct policy search methods. The suitability of policy gradient methods in robotics is pointed out in this chapter.

Chapter 3 *Policy Gradient Methods.* This chapter analyzes and classifies various gradient algorithms. The objective of this chapter is to emphasize the main ideas of each methodology, its weaknesses and its strong points.

Chapter 4 *Policy Gradient Methods in Robotics.* This chapter presents recent successful applications in robotics using the theoretic algorithms described in previous chapter.

Chapter 5 *Neural Policy Gradient Algorithm (NPGA).* In order to carry out the experiments, one of the studied algorithms was implemented. The algorithm uses a neural network as a function approximator.

Chapter 6 *Experimental Results.* The NPGA was applied in two different RL problems. First, an underwater robot having to navigate in a simulated 2D environment. Secondly, the NPGA performance is tested in the "mountain-car task" benchmark and the results are compared with the ones obtained with Q-learning.

Chapter 7 *Conclusions and Future Work.* This chapter concludes the research project by summarizing the work and pointing out the thesis proposal.

Chapter 2

Reinforcement Learning

A commonly used methodology in robot learning is Reinforcement Learning (RL). In RL, an agent tries to maximize a scalar evaluation (reward or punishment) obtained as a result of its interaction with the environment. This chapter overviews the field of RL. Once the main aspects are described, we compare different RL based algorithms and, to solve the RL problem, two methodologies are presented: Value Function algorithms and Policy Gradient methods. This chapter describes the principal advantages and defects when working on-line and off-line. Finally, some conclusions are presented, among which, Policy Gradient Algorithms (PGA) are shown to be suitable for our robot applications. This chapter represents only a general overview of RL theory. For a deeper understanding of this field, refer to the book "Reinforcement Learning. An Introduction" [Sutton and Barto, 1998].

2.1 The Reinforcement Learning Problem

The goal of an RL system is to find an optimal policy to map the state of the environment to an action which in turn will maximize the accumulated future rewards. Most RL techniques are based on *Finite Markov Decision Processes* (FMDP) causing finite state and action spaces. The main advantage of RL is that it does not use knowledge database, as does *supervised learning* when it uses a set of examples which show the desired input/output, so the learner is not told what to do, but instead must discover actions which yield the most reward. Therefore, this class of learning is suitable for online robot learning. The agent interacts with a new, undiscovered environment selecting the actions computed as the best for each state, receiving a numerical reward for every decision. These rewards will be "rich" for good actions and "poor" for bad actions. The rewards are used to teach the agent and in the end the

robot learns which action it must take at each state, achieving an optimal or sub-optimal policy (state-action mapping).

We find different elements when working with RL algorithms. The first is the *agent or learner* which interacts with the *environment*. The environment includes everything that is outside the learner. If we describe the interaction process step by step, the learner first observes the *state* of the environment and, as a result, the agent generates an action and the environment responds to it with a new state and a *reward*. The reward is a scalar value generated by a *reinforcement function* which evaluates the action taken related to the current state. The learner-environment relationship can be seen in Figure 2.1. Observing the diagram, the interaction sequence can be described: for each iteration step t, the learner observes the state s_t and receives a reward r_t. According to these inputs and the RL policy followed at that moment, the learner generates an action a_t. Consequently, the environment reacts to this action changing to a new state s_{t+1} and giving a new reward r_{t+1}. A sequence of states, actions and reward is shown in Figure 2.2. The most important features of the learner and environment are listed as follows:

Learner :

- Performs the learning and decides the actions.
- Input: the state s_t and reward r_t (numerical value).
- Output: an action a_t.
- Goal: to maximize the amount of rewards $\sum_{i=t+1}^{\infty} r_i$.

Environment :

- Everything outside the learner.
- Reacts to actions with a new state.
- Contains the Reinforcement Function which generates the rewards.

As stated before, the learner interacts with the environment in order to find correct actions. The action applied depends on the current state, so at the end of the learning process, the agent has a *mapping function* which relates every state with the best action taken. To get the best actions at every state, two common processes in RL are used: *exploitation and exploration*. Exploitation means that the learner always selects what is thought to be the best action at every current state, but, sometimes exploration is required to

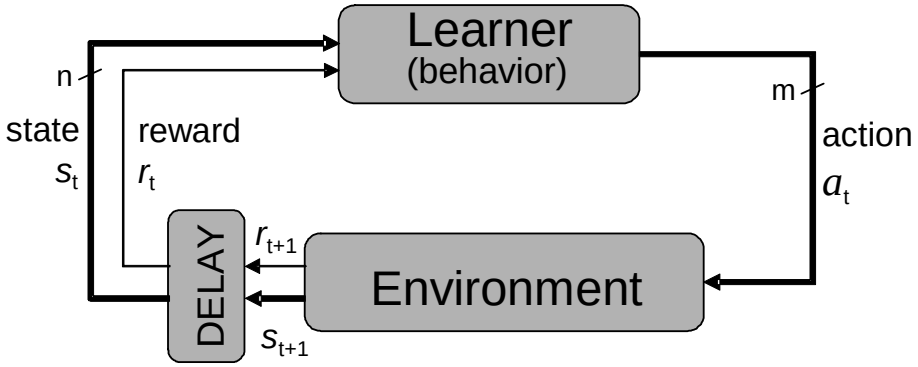

Figure 2.1: Diagram of the interaction Learner vs. Environment.

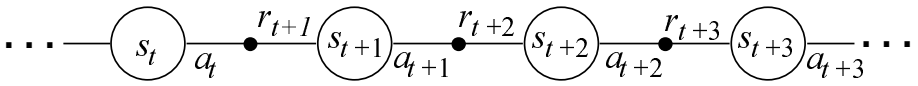

Figure 2.2: Schematic sequence of states, actions and rewards.

investigate the effectiveness of actions that have not been tried by the current state.

Once the learning period is through, the agent does not select the action that maximizes the immediate reward but the one that maximizes the sum of future rewards. In other words, an action can be the best one to solve the RLP, but high rewards will not come until after some iterations.

If we take a deeper look, beyond the environment and the agent, four main subelements of a reinforcement learning system can be identified: a *policy function*, a *reinforcement function*, a *value function* and a *model* of the environment. These functions define the RLP.

Policy Function. This defines the action to be taken by the agent at a particular state. The policy function represents a mapping between states and actions. In general, policies may be stochastic and are usually represented with the $\pi(s, a)$ function, where the probability of choosing action a from state s is contained. The policy that gets maximum accumulated rewards is called optimal policy and is represented as π^*.

Reinforcement Function. This defines the goal in a reinforcement learning problem. It is located in the environment and, roughly speaking, maps each perceived state of the environment to a single number called a *reward*. The main objective of an RL learner is to maximize the total

reward it receives in the long run. The reinforcement function gives an immediate evaluation to the learner, but high immediate rewards are not the objective of an RL learner but the total amount of rewards perceived at the end.

Value Function. As stated before, the reinforcement function indicates what is considered "good" in an immediate sense. The value function defines what will be good in the long run starting from a particular state. In other words, the value of a state is the total amount of estimated reward that the learner is going to receive, following a particular policy, starting from the current state. There are two kinds of value functions. The first is *State-Value* function $V^{\pi}(s)$, which contains the sum of expected rewards starting from state s and following a particular policy π. The second is the *Action-Value* function $Q^{\pi}(s, a)$, which contains the sum of rewards in the long run starting in state s, taking action a and then following policy π. The action-value function $Q^{\pi}(s, a)$ will be equal to the state-value function $V^{\pi}(s)$ for all actions considered greedy with respect to π. Once the learner reaches the optimal value functions, we can obtain the optimal policy.

Model of the environment. Also known as dynamics function. It describes the behavior of the environment and, given a state and an action, the model of the environment generates the next state and the next reward. This function is usually stochastic and unknown, but the state transitions caused by the dynamics are contained in some way in the value functions.

Most part of RL algorithms use these functions to solve the RLP. If we look deeper into algorithms that use a value function we notice that once the learning process is performed, the algorithm proposes a learning update rule to modify the value function, and then proposes a policy to be followed by the learner. If the RL algorithm converges after some iterations, the value function changes to the optimal value function from which the optimal policy can be extracted. The solution of the RLP is accomplished by following the state-action mapping contained in the optimal policy π^{*}. Figure 2.3 represents a phase diagram which is usually followed in RL algorithms based on the value function procedure.

The studies presented in this research project point out that other kinds of algorithms, in which attention has been growing recently, are able to solve the RLP without a value function. *Policy methods* or *Direct policy search methods* do not perform the learning process over value functions but over

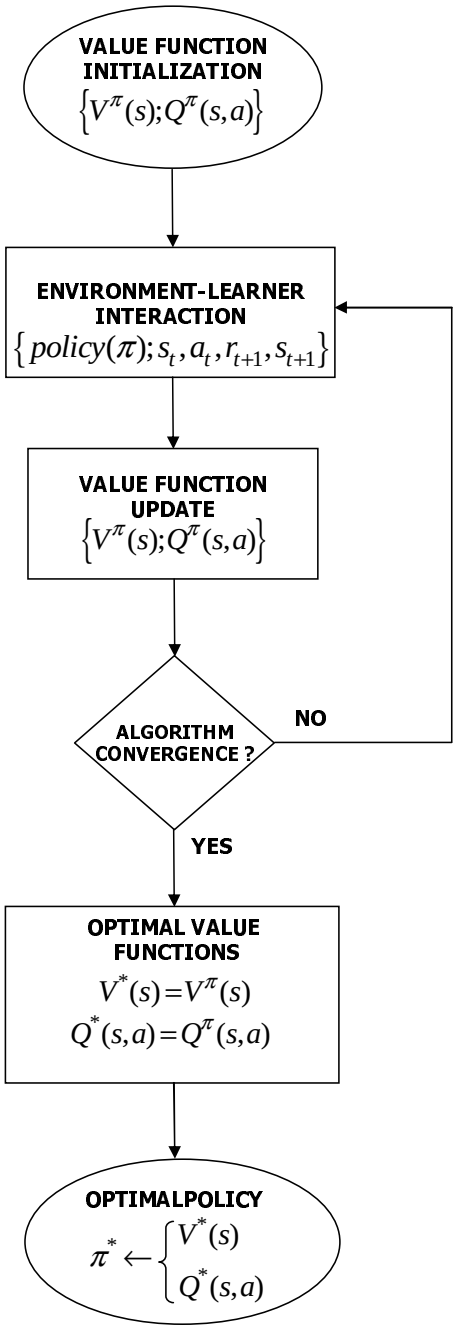

Figure 2.3: Typical phase diagram of a value function based RL algorithm, where the value function is updated according to the algorithm. Once the optimal value function is found, the optimal state-action policy is extracted.

the policy itself. These algorithms propose a learning update rule to directly modify the policy parameters. As will be seen in the following lines, these kinds of algorithms have advantages and disadvantages respecting value methods. Figure 2.4 represents the phase-diagram of policy based algorithms.

2.2 Finite Markov Decision Processes in RL

As mentioned before, in reinforcement learning the agent makes its decision according to the environment's information. This information is presented as the environment's *state* and *reward*. If the information contained in the state is sufficient for the agent to solve the RLP, it means that the state summarizes all relevant information. In this case the state will be called *complete* and the process is said to have accomplished the *Markov Property*. An environment which accomplishes the Markov property contains in its state all relevant information to predict the next state. Completeness entails that knowledge of past states, measurements or controls carry no additional information to help us predict the future more accurately. At the same time, future rewards do not depend on past states and actions. In Equation 2.1 we define the conditional probability of achieving next state s_{t+1} and obtaining reward r_{t+1}, when taking action a_t and knowing previous states and actions,

$$Pr\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., s_0, a_0\} \tag{2.1}$$

If an environment has the Markov property, the environment's new state and reward, s_{t+1} and r_{t+1}, will depend only on the state/action representation at time t. This statement can be defined mathematically as follows:

$$Pr\{s_{t+1} = s', r_{t+1} = r' | s_t, a_t\} \tag{2.2}$$

The environment and the whole process is considered to be a Markov Decision Process (MDP) if, and only if, Equation 2.1 is equal to Equation 2.2 for all states and actions. Moreover, if the state and action spaces are finite, the environment is considered a Finite MDP (FMDP). Therefore, for a particular finite Markov decision processes with finite states and actions, the stochastic dynamics of the environment can be expressed by a *transition probability* function $P^a_{ss'}$. This function represents the probability of reaching state s' from state s if action a is taken. Equation 2.3 defines this function.

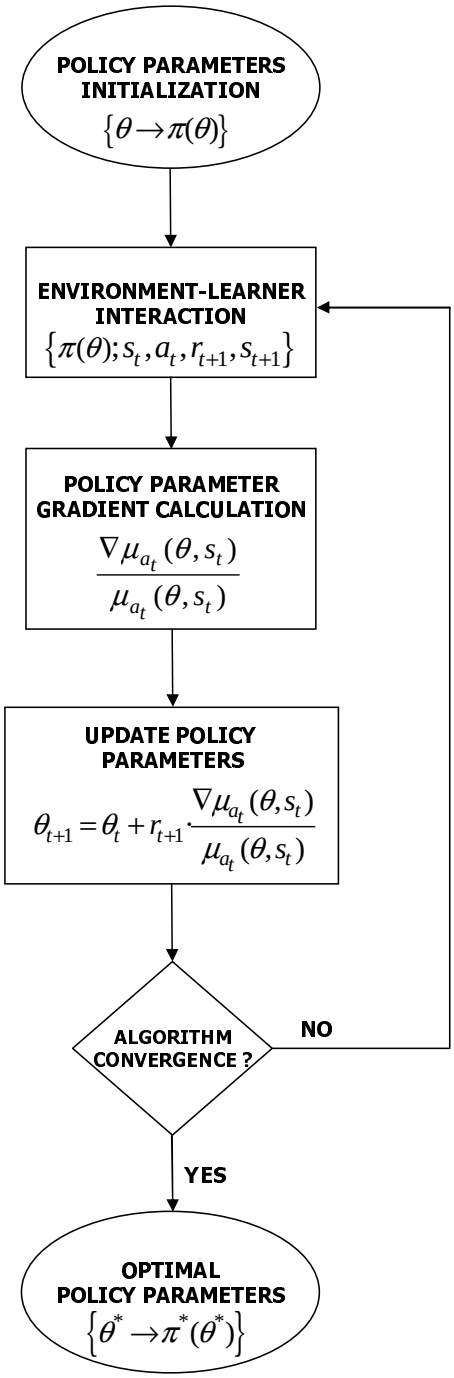

Figure 2.4: Diagram of a direct policy based RL algorithm. The policy parameters are updated according to the algorithm. Once the optimal policy parameters are found, the learning process finishes.

$$P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.3}$$

In the same way, the expected value of the next reward $R^a_{ss'}$ can be obtained. Given any current state s and action a, together with any state s', we have:

$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.4}$$

Both $P^a_{ss'}$ and $R^a_{ss'}$ represent the most important concepts for defining the dynamics of a FMDP. One of the learner's functions, the value function, is highly related to this dynamic. For a particular learner policy π, the value function (V^π or Q^π) can be expressed in terms of $P^a_{ss'}$ and $R^a_{ss'}$ and, as will be shown, the optimal value function (V^* or Q^*) can also be determined. Once this function is obtained, the optimal policy π^* can be extracted from it. Before reaching these expressions, a new function has to be defined.

As has been stated, the goal of RL is to maximize the sum of future rewards. A new function R_t is used in the FMDP framework to express this sum, as Equation 2.5 shows. This sum finishes at time T, when the task that RL is trying to solve finishes. The tasks, having a finite number of steps, are called *episodic tasks*. However, RL is also suitable for solving tasks which do not finish at a certain number of time steps. For example, in a robotics task, the learner may be continually activated. In this case, the tasks are called *continuing tasks* and can run to infinite. To avoid an infinite sum of rewards, the goal of RL is reformulated to the maximization of the *discounted sum* of future rewards. The future rewards are corrected by a discount factor γ as expressed in Equation 2.6.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{2.5}$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.6}$$

Setting the discount factor between $0 \leq \gamma \leq 1$, the infinite sum of rewards does not achieve infinite values and, therefore, the RLP can be solved. In addition, the discount factor allows the selection of the number of future rewards to be maximized. For $\gamma = 0$ only the immediate reward is maximized. For $\gamma = 1$ the maximization will take into account the infinite sum of rewards. Finally, for $0 < \gamma < 1$ only a reduced set of future rewards will be maximized.

The two value functions, V^π and Q^π, can be expressed in terms of the expected future reward R_t. In the case of the *state-value* function, the value of

a state s under a policy π, denoted $V^\pi(s)$, is the expected discounted sum of rewards when starting in s and following π thereafter. For the *action-value* function, the value of taking action a in state s under policy π, denoted $Q^\pi(s,a)$, is the expected discounted sum of rewards when starting in s, applying action a and following π thereafter. Equations 2.7 and 2.8 formally define these two functions.

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right\} \tag{2.7}$$

$$Q^\pi(s,a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} \tag{2.8}$$

The last two equations define the value functions obtained when following a particular policy π. To solve the RLP, the optimal policy π^* which maximizes the discounted sum of future rewards has to be found. As the value functions indicate the expected sum of future rewards for each state or state/action pair, an optimal value function will contain the maximum values. Therefore, from all the policies π, the one having a value function (V^π or Q^π) with maximum values in all the states or state/action pairs will be an optimal policy π^*. It is possible to have several policies $(\pi_1^*, \pi_2^*, ...)$ which fulfill this requirement, but only one optimal value function can be found (V^* or Q^*). Equations 2.9 and 2.10 reflect this statement.

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.9}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2.10}$$

In order to find these optimal value functions, the Bellman equation [Bellman, 1957] is applied. This equation relates the value of a particular state or state/action pair with the value of the next state or state/action pair. To relate the two environment states, the dynamics of the FMDP ($P^a_{ss'}$ and $R^a_{ss'}$) is used. The *Bellman optimality equations* for the state and action value functions are found in Equations 2.11 and 2.12.

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')] \tag{2.11}$$

$$Q^*(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')] \tag{2.12}$$

The Bellman optimality equations offer a solution to the RLP by finding the optimal value functions V^* and Q^*. If the dynamics of the environment

is known, a system of equations with N equations and N unknowns can be written using Equation 2.11, being N the number of states. This nonlinear system can be solved resulting in the V^* function. Similarly, the Q^* function can be found.

Once V^* is known, the optimal policy can be easily extracted. For each state s, any action a which causes the environment to achieve a state s' with maximum state value with respect to the other achievable states can be considered as an *optimal action*. The set of all the states with their corresponding optimal actions constitutes an optimal policy π^*. It is important to note that to find each optimal action, it is only necessary to compare the state value of the next achievable states. This is due to the fact that the state-value function V^* already contains the expected discounted sum of rewards for these states. In the case where the Q^* is known, the extraction of the optimal policy π^* is even easier. For each state s, the optimal action will be the action a, which has a maximum $Q^*(s, a)$ value, see Equation 2.13.

$$\pi^*(s) = arg \max_{a \in A(s)} Q^*(s, a) \tag{2.13}$$

This section has formulated the Reinforcement Learning problem using Finite Markov Decision Processes. It has also pointed out how to find the solution of the RLP when the dynamics of the environment is known.

Markovian environments have been a platform for various RL algorithms, like Sutton's temporal difference TD(λ) algorithm [Sutton, 1988] or Watkin's Q-learning(QL) method [Watkins, 1989]. The aim of dynamic programming(DP) and theory of stochastic approximation allowed these algorithms to be analyzed [Dayan, 1992], [Tsitsiklis, 1994], [Jaakkola et al., 1994] and [Watkins and Dayan, 1992]. Considering a Markovian environment is a risky assumption. The non-Markov nature of an environment can manifest itself in many ways, and the algorithm can fail to converge if the environment does not accomplish all its properties [Singh et al., 1994]. Following this path, the most direct extension of MDP that blinds part of the state to the learner is known as *Hidden Markov Models*(HMM). The underlying environment continues to be Markovian, but the information extracted does not seem Markovian to the learner. The analogy of HMM for control problems are the *Partially Observable Markov Decision Processes*(POMDP) [Monahan, 1982]. The algorithms presented in this research project are based on POMDP because, in practice, it is impossible to get a complete state for any realistic robot system.

2.3 Value Function Methods vs Policy Search algorithms

The dominant approach over the last decade has been to apply reinforcement learning using the *value function* approach. As a result, many RL-based control systems have been applied to robotics over the past decade. In [Smart and Kaelbling, 2000], an instance-based learning algorithm was applied to a real robot in a corridor-following task. For the same task, in [Hernandez and Mahadevan, 2000] a hierarchical memory-based RL was proposed, obtaining good results as well. [Carreras et al., 2003] presented an underwater robot that learnt different behaviors using a modified Q-learning algorithm. Although value function methodologies have worked well in many applications, they have several limitations. The considerable amount of computational requirements that increase time consumption and the lack of generalization among continuous variables represent the two main disadvantages of "value" RL algorithms.

Reinforcement Learning is usually formulated using Finite Markov Decision Processes (FMDP). This formulation implies a discrete representation of the state and action spaces. However, in some tasks the states and/or the actions are continuous variables. A first solution can be to maintain the same RL algorithms and discretize the continuous variables. If a coarse discretization is applied, the number of states and actions will not be too high and the algorithms will be able to learn. However, in many applications the discretization must be fine in order to assure a good performance. In these cases, the number of states will grow exponentially, making the use of RL impractical. The reason is the high number of iterations necessary to update all the states or state/action pairs until an optimal policy is obtained. This problem is known as the *curse of dimensionality*. In order to solve this problem, most RL applications require the use of generalizing function approximators such as artificial neural-networks (ANNs), instance-based methods or decision-trees. In some cases, Q-learning can fail to converge to a stable policy in the presence of function approximation, even for MDPs and it may be difficult to calculate $max_{a \in A(s)} Q^*(s, a)$ when dealing with continuous space-states [Murphy, 2000].

Another feature of value function methods is that such approaches are oriented to finding deterministic policies. However, stochastic policies can yield considerably higher expected rewards than deterministic ones as in the case of POMDPs, selecting among different actions with specific probabilities [Singh et al., 1994]. Furthermore, some problems may appear when the state-space is not completely observable (POMDP), small changes in the es-

timated value of an action may or may not cause it to be selected; reslting in convergence problems [Bertsekas and Tsitsiklis, 1996].

Over the past few years, studies have shown that approximating a policy directly can be easier than working with value functions, and better results can be obtained [Sutton et al., 2000][Anderson, 2000]. Informally, it is intuitively simpler to determine *how to act* instead of *value of acting* [Aberdeen, 2003]. So, rather than approximating a value function, new methodologies approximate a policy using an independent function approximator with its own parameters, trying to maximize the future expected reward. Furthermore, scientists have developed different kinds of policy search algorithms obtaining good results. This survey cites the methods presented in [Sondik, 1978], [Bertsekas and Tsitsiklis, 1996], [Hansen, 1998] and [Meuleau et al., 1999]. Meuleau's work also proposed how gradient methods can be used to search in the space of stochastic policies. These particular kind of algorithms are the focus of our research project.

Policy gradient algorithms can be used to represent the policy. For example, an ANN whose weights are the policy parameters. The state would be the input of the network and as output we would have a distribution probability function for action selection. In Equation 2.14 we can see that if θ represents the vector of the policy parameters and ρ the performance of the policy (e.g., reward received), then the policy parameters are updated approximately proportional to the gradient [Sutton et al., 2000]:

$$\Delta\theta \approx \alpha \frac{\delta\rho}{\delta\theta} \tag{2.14}$$

where α is a positive step size. In comparison with the value function approach, small changes in θ can cause only small changes in the policy.

The advantages of policy gradient methods against value-function based methods are various. The main advantage is that using a function approximator to represent the policy directly solves the generalization problem. Besides, a problem for which the policy is easier to represent should be solved using policy algorithms [Anderson, 2000]. Working this way should represent a decrease in the computational complexity and, for learning systems which operate in the physical world, the reduction in time consumption would be enormous. Furthermore, learning systems should be designed to explicitly account for the resulting violations of the Markov property. Studies have shown that stochastic policy-only methods can obtain better results when working in POMDP than those obtained with deterministic value-function methods [Singh et al., 1994]. In [Anderson, 2000] a comparison between a policy-only algorithm [Baxter and Bartlett, 2000] and a value

Q-learning method [Watkins and Dayan, 1992] is presented; both algorithms use a simple neural network as function approximator. A 13-state Markovian decision process is simulated for which the Q-learning oscillates between the optimal and a suboptimal policy while the policy-only method converges to the optimal policy.

On the other hand, as disadvantage, policy gradient methods learn much more slower than RL algorithms using a value function [Sutton et al., 2000], they use to find local optima of the expected reward [Meuleau et al., 2001] and the gradient estimators used in these algorithms may have a large variance [Marbach and Tsitsiklis, 2000][Konda and Tsitsiklis, 2003].

The first example of an algorithm optimizing the averaged reward obtained for stochastic policies working with gradient direction estimates was Williams's REINFORCE algorithm [Williams, 1992]. This algorithm learns much more slower than other RL algorithms which work with a value function and, maybe for this reason, has received little attention. However, the ideas and mathematical concepts presented in REINFORCE were a basic platform for later algorithms.

A few years later, in [Kimura et al., 1997], Williams's algorithm was extended to the infinite horizon setting. Kimura's method is, as is REINFORCE, based on stochastic gradient ascent (SGA). The authors compared its algorithm with Jaakola's method [Jaakkola et al., 1995] and Watkin's Q-learning algorithm [Watkins and Dayan, 1992] in a robot control problem achieving good results.

The Baxter and Bartlett approach [Baxter and Bartlett, 1999] is the one selected in this research project to carry out the experiments. Its method calculates a parameterized policy that converges to an optimal by computing approximations of the gradient of the averaged reward from a single path of a controlled POMDP. The convergence of the method is proven with probability 1, and one of the most attractive features is that it can be implemented on-line. Baxter and Bartlett's approach is based on the fact that, given a state s, it searches for a policy that minimizes the expected reward. Moreover, in [Marbach and Tsitsiklis, 1998] and [Marbach, 1998] an algorithm similar to Baxter and Bartlett's approach was described and its convergence demonstrated. The algorithm is only suitable for finite MDP and can be implemented to work on-line.

Learning a value function and using it to reduce the variance of the gradient estimate appears to be essential for rapid learning. In order to improve learning speed, some "hybrid" approaches combine value function estimates with gradient estimation. Among these, this survey wishes to point out [Jaakkola et al., 1995] and more recently [Baird and Moore, 1999] and [Sutton et al., 2000]. In Jaakkola's method, the algorithm involves a Monte-

Carlo policy evaluation combined with a policy improvement algorithm; results guaranteed the convergence to local maximum. On the other hand, Baird and Moore presented a methodology that allows policy-search and value-based algorithms to be combined, unifying these two approaches into what they named Single Value and Policy Search (VAPS) algorithm. Sutton's method proposed a policy iteration algorithm with a general differentiable function approximator. Results demonstrate converge to local optima.

Other studies in this survey are certain variants of actor-critic algorithms or policy iteration architectures that obtained good results as well. The studies presented in [Kimura and Kobayashi, 1998] and, more recently, the work in [Konda and Tsitsiklis, 2003] can be considered a reference in this field.

As can be appreciated, this research project has a special interest in those model-free algorithms designed to perform practical tasks in unknown environments and on-line. However, some model-based approaches have also obtained good results by first learning the policy on a simulated model and, once the policy is considered learned, using it on the real agent. Among these methods, the PEGASUS algorithm [Ng and Jordan, 2000] obtained good results by simulating a "transformed" POMDP with deterministic transitions. The obtained results showed a considerable reduction of the variance. An important trade off is that it can introduce false local maxima [Aberdeen, 2003].

Close to the root of these theoretical variants of policy search methods, only a few but promising practical applications of these algorithms have appeared. Chronologically, this research project emphasizes the work presented in [Bagnell and Schneider, 2001], where an autonomous helicopter learns to fly using an off-line model-based policy search method. Also important is the work presented in [Rosenstein and Barto, 2001] where a simple "biologically motivated" policy gradient method is used to teach a robot in a weightlifting task. More recent is the work done by [Kohl and Stone, 2004] where a simplified policy gradient algorithm is implemented to optimize the gait of Sony's AIBO quadrupedal robot. Finally, in [Tedrake et al., 2004] and [Matsubara et al., 2005], a biped robot is trained to walk by means of a "hybrid" RL algorithm that combines policy search with value function methods.

All these methodologies and their practical applications in robotic systems will be detailed in the next two chapters.

Chapter 3

Policy Gradient Methods

This chapter introduces several policy-gradient methods for agent training algorithms. The survey presented here details the most important methodologies from over the last few years. These approaches have been classified into two main currents: methods which are strictly based on the gradient of the averaged reward (here considered as *pure* policy gradient algorithms) and those which, besides the gradient estimates, have the assistance of a value function (considered *hybrid* policy-valued algorithms). Algorithms stated here constitute the basic foundation of most successful practical applications which solve the RLP with direct policy search.

3.1 "Pure" policy gradient algorithms

The easy structure of these algorithms, without any computational complexity, and their capability of mildly adapting to non-Markov environments, make pure policy gradient methods the most suitable for real robot applications in unknown environments. The best representatives are enumerated in the following.

3.1.1 Williams' REINFORCE

As mentioned in the previous chapter, Williams' algorithm is a reference in policy search methods and its principles are an essential support for most RL policy gradient-based methods. The REINFORCE algorithm operates by adjusting weights in a direction that lies along the gradient of expected reinforcement (delayed or immediate) rewards. Let's consider a neural network (ANN) dealing with an immediate-reinforcement learning task. The weights of this network are updated following the expression:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij} \tag{3.1}$$

where α_{ij} is a *learning rate factor*, b_{ij} is a *reinforcement baseline* and $e_{ij} = \frac{\delta \ln g_i}{\delta w_{ij}}$ represents the *eligibility* of w_{ij}. g_i represents the probability mass function which determines the value at the output of the ANN as a function of the network weights and its input. Considering the special case of a stochastic network with its neurons of the logistic type, the eligibility for each weight w_{ij} can be computed, following the chain rule, as stated in Equation 3.2;

$$e_{ij} = \frac{\delta \ln g_i}{\delta w_{ij}} = \frac{y_i - p_i}{p_i(1 - p_i)} f_i'(s_i)x_j = (y_i - p_i)x_j \tag{3.2}$$

where y_i represents the output of the ith unit in the network, p_i is the probability of this output when the total input is s_i, $f'(s_i)$ is the derivative of that neuron's activation function and finally x_j is the jth input to the neuron, either provided by another neuron of the preceding layer or directly as a part of the state of the environment. Furthermore, the REINFORCE algorithm has the form:

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})(y_i - p_i)x_j \tag{3.3}$$

Williams' algorithm examples set the reinforcement baseline b_{ij} to 0, but its algorithm is also consistent with delayed rewards setting the baseline to a term $\bar{r}$ that acts as an adaptive estimate of past experience:

$$\bar{r}(t) = \gamma r(t-1) + (1 - \gamma)\bar{r}(t-1) \tag{3.4}$$

where $0 < \gamma < 1$. Subsequently, the expression that gives us the update rule for every parameter of our network is formulated in Equation 3.5;

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_{ij}(r(t) - \bar{r}(t))(y_i(t) - p_i(t))x_j(t) \tag{3.5}$$

For more information on this work refer to [Williams, 1992].

3.1.2 Kimura's extension

Williams' REINFORCE is unbiased. Hajime Kimura and coauthors extended Williams' algorithm to the infinite horizon by adding a *discount factor* γ

into the calculation of the eligibility as seen in Equation 3.7. This term considerably improves the performance of the algorithm but, as drawback, our gradient estimates will be biased:

$$e_i(t) = \frac{\delta \ln g_i}{\delta w_i} \tag{3.6}$$

$$D_i(t) = e_i(t) + \gamma D_i(t-1) \tag{3.7}$$

where $0 < \gamma < 1$. Williams' eligibility $e_i(t)$ contains information about immediate executed actions. $D_i(t)$ acts as a discounted running average of the eligibility. Moreover, as we move the discount factor γ closer to 1, we increase the memory of the agent concerning past actions. This new term is called *eligibility trace* [Singh and Sutton, 1996] and accumulates the agent's history on executed actions. Kimura's update weight procedure is computed as:

$$w_i(t+1) = w_i(t) + \alpha(1-\gamma)(r(t) - b)D_i(t) \tag{3.8}$$

where α_{ij} is a *learning rate factor*, γ represents a *discount factor*, b_{ij} is a *reinforcement baseline* and $D_i(t)$ concerns the *eligibility trace*. For more information on this work refer to [Kimura et al., 1997].

3.1.3 Baxter and Bartlett's approach

The algorithm proposed by Baxter and Bartlett is considered, like Kimura's, an extension of Williams' REINFORCE and very similar to other recent contemporary methodologies such as the one in [Marbach and Tsitsiklis, 1998] or [Marbach, 1998]. The main difference with the algorithm proposed by Kimura is that this method does not use a reward baseline. Also, differing from Marbach's, this algorithm is suitable for use in POMDP, as it does not need the knowledge of the transition probability function $P^a_{ss'}$, and only needs to know a randomized policy μ.

$\theta \in R^K$ represents the parameter vector of an approximate function that maps a stochastic policy. Let $\eta(\theta)$ be the averaged reward of the POMDP with the parameter vector θ. The gradient function computes approximations $\nabla\eta_\beta(\theta)$ to $\nabla\eta(\theta)$ based on a continuous sample path of the Markov chain of the parameterized POMDP. One of the alternative approaches considered by Baxter and Bartlett's algorithm is the possibility of being implemented on-line by adjusting the parameter vector at every iteration of

the partially-Markovian environment. The accuracy of the approximation is controlled by the discount factor $\beta \in [0, 1)$ which increases or decreases the agent's memory of past actions as in Kimura'a algorithm. It was proven in [Baxter and Bartlett, 2000] that

$$\nabla\eta(\theta) = \lim_{\beta \to 1} \nabla_\beta \eta(\theta) \tag{3.9}$$

As we increase the value of β, the memory of the agent increases, however the trade-off is that the variance of the estimates $\nabla\eta_\beta(\theta)$ also rises with this parameter.

Furthermore, given a POMDP with the magnitudes of its rewards uniformly bounded for all possible states, an initial parameter values $\theta_0 \in R^K$ and a randomized differentiable and uniformly bounded policy $\mu(\theta, *)$ for all observed states and actions, the vector of parameter that represents the current policy can be updated following the expressions below. In Equation 3.10, the eligibility trace for each parameter is calculated according to the gradient approximation.

$$z_{t+1} = \beta z_t + \frac{\nabla \mu_{u_t}(\theta, y_t)}{\mu_{u_t}(\theta, y_t)} \tag{3.10}$$

z_t relates the eligibility trace, β is a discount factor, y_t is the observed state and u_t the generated control action for each time step t. In Equation 3.11 the update parameter procedure is completed,

$$\theta_{t+1} = \theta_t + \alpha_t r(i_{t+1}) z_{t+1} \tag{3.11}$$

where θ_t represents the parameter vector, α_t is the learning rate and $r(i_{t+1})$ as the immediate reward perceived. This algorithm requires $2K$ parameters to be stored, where K is the length of the parameter vector θ. For more information on this work refer to [Baxter and Bartlett, 1999].

3.1.4 Marbach's algorithm

The considerations taken in Marbach's methodology make its algorithm more complex mathematically than other pure policy gradient methods presented before. This algorithm is designed for Markov reward processes and needs to know the environment's transition probability function $P^a_{ss'}$. These requirements make this method unsuitable for robotic applications in real environments, but the mathematical concepts introduced in this work are a reference

in this field. The proposed on-line algorithm updates $2K + 1$ numbers every iteration step. As with other algorithms, it stores the vector of the policy parameters θ and a Z vector similar to Kimura's eligibility trace. Also, it computes and stores an estimate $\widetilde{\lambda}$ of the average reward under the current value of θ. Authors assure us that, as θ keeps changing, λ is not an accurate estimate of the averaged reward but a biased estimation.

At any time state k, being the current state i_k, $g_i(\theta)$ is the one stage reward depending on θ, and the values of θ_k, Z_k and $\widetilde{\lambda}_k$ are available from the previous iteration, the update of θ and $\widetilde{\lambda}$ is computed as follows:

$$\theta_{k+1} = \theta_k + \alpha_k(\nabla g_{i_k}(\theta_k) + (g_{i_k}(\theta_k) - \widetilde{\lambda}_k)z_k) \tag{3.12}$$

$$\widetilde{\lambda}_{k+1} = \widetilde{\lambda}_k + \eta\alpha_k(g_{i_k}(\theta_k) - \widetilde{\lambda}_k) \tag{3.13}$$

The updated policy parameters allow us to make a transition to next state i_{k+1} according to the transition probabilities $p_{ij}(\theta_{k+1})$, so the computation of the vector traces can be obtained as stated in Equation 3.14.

$$z_{k+1} = \left\{ \begin{array}{cc} 0, & if i_{k+1} = i^* \\ z_k + L_{i_k i_{k+1}}(\theta_{k+1}), & otherwise \end{array} \right\} \tag{3.14}$$

where $L_{i_k i_{k+1}}(\theta_{k+1})$ can be interpreted as a likelihood ratio derivative term [L'Ecuyer, 1990] related to the space of transition probabilities and defined as:

$$L_{i_k i_{k+1}}(\theta_{k+1}) = \frac{\nabla p_{i_k i_{k+1}}(\theta_{k+1})}{p_{i_k i_{k+1}}(\theta_{k+1})} \tag{3.15}$$

$p_{i_k i_{k+1}}$ represents the transition probability of, once being in state i_k, going to state i_{k+1} under current policy θ_{k+1}. For more information on this work refer to [Marbach and Tsitsiklis, 1998].

3.2 "Hybrid" policy-valued algorithms

High variances of policy gradient methods slow down the convergence speed of this algorithms. On the other hand, value methods maintain low variances but they do not guarantee convergence when dealing with POMDPs. The idea of hybrid algorithms is to combine the advantages of both methods, thus obtaining a fast algorithm able to converge in semi-Markov environments.

3.2.1 Jaakkola's proposed method

Jaakkola's algorithm is divided into two parts or different steps: First, the value Q-function is computed by means of a Monte-Carlo (MC) method [Sutton and Barto, 1998][Barto and Duff, 1994]. The obtained policy assures a higher reward as long as for some m:

$$\max_a[Q(m,a) - V(m)] > 0 \tag{3.16}$$

The averaged reward is not known by the learner and, in order to compute the Q-values, the true averaged reward is replaced by an incremental estimation of the expected reward. In a second step, as detailed in Equation 3.17, the policy is improved by increasing the probability of taking the best action as defined by $Q(m,a)$

$$\pi(a|m_n) \rightarrow \pi(a|m_n) + \epsilon[Q_n(m_n,a) - V_n(m_n)] \tag{3.17}$$

When the condition expressed in Equation 3.16 is not true, the algorithm has reached a local maximum and the learning process finishes. As aforementioned, the aim of this method is not to wait for the Monte-Carlo evaluation to converge, but to help the policy to find optimality. Also, the authors proposed an on-line version of the algorithm where the policy is changed at the same time with the calculation of the Q-values. For more information on this work refer to [Jaakkola et al., 1995].

3.2.2 Baird and Moore's VAPS

The VAPS (Value and Policy Search) algorithm offers the possibility of a flexible selection among different types of RL algorithms depending on the expression chosen to compute an error function. Furthermore, the VAPS algorithm includes Q-learning, SARSA and advantage learning. In addition, Baird and Moore's method allows policy search and value-based algorithms to be combined. This last possibility has been studied in this research work. The method procedure starts by computing the eligibility trace for the current policy as stated in Equation 3.18 and Equation 3.19;

$$\Delta e_t = \frac{\delta \ln P(u_{t-1}|s_{t-1})}{\delta w} \tag{3.18}$$

$$e_t = e_{t-1} + \Delta e_t \tag{3.19}$$

Starting from a general VAPS formulation, possibilities appear when selecting an RL methodology (QL, SARSA, advantage learning, value-iteration or policy-value), each one with a different error function $\epsilon(s_t)$ that calculates the error at each step time. The combination of policy search and value function has an error function as detailed in Equation 3.20;

$$\epsilon_{SARSA-Policy}(s_t) = (1-\beta)\epsilon_{SARSA}(s_t) + \beta(b - \gamma' R_t) \tag{3.20}$$

where ϵ_{SARSA} is the error function for the value-function approach, b is a reinforcement baseline, γ' represents a discount factor and β defines a mixing term. Baird's "hybrid" algorithm proposes a combination of a pure policy algorithm and a SARSA Q-valued method. Adjustments of parameter β between 0 and 1 allows us to go back and forth between both methods. When $\beta = 0$ the algorithm totally learns a Q-function that satisfies the Bellman equation. On the other hand, if $\beta = 1$, the algorithm is reduced to Williams' REINFORCE algorithm and directly learns a policy that will minimize the expected total discounted reward. The error function $\epsilon_{SARSA}(s_t)$ is obtained as defined in Equation 3.21:

$$\epsilon_{SARSA}(s_t) = \frac{1}{2}E^2[R_{t-1} + \gamma Q(x_t, u_t) - Q(x_{t-1}, u_{t-1})] \tag{3.21}$$

Finally, the update procedure is formulated in Equation 3.22 and Equation 3.23;

$$\Delta w_t = -\alpha[\frac{\delta \epsilon_{SARSA-Policy}(s_t)}{\delta w} + \epsilon_{SARSA-Policy}(s_t)e_t] \tag{3.22}$$

$$w_t = w_{t-1} + \Delta w_t \tag{3.23}$$

For more information on this work refer to [Baird and Moore, 1999].

3.2.3 Sutton's generalization

This approach considers the standard MDP framework and generalizes "hybrid" policy-valued methodologies in a mathematical sense. The work presented by Sutton's team is a theoretical convergence demonstration where the authors prove for the first time that a generalized version of a policy iteration with an arbitrary differentiable function approximator is convergent

to local optima. Therefore, the gradient can be aided by an approximate action-value or advantage function if the last one is compatible with the policy parametrization, accomplishing Equation 3.24, in order to speed up the convergence time of the algorithm.

$$\frac{\delta f_w(s,a)}{\delta w} = \frac{\delta \pi(s,a)}{\delta \theta}\frac{1}{\pi(s,a)} \tag{3.24}$$

where s and a represent the state-action pair. π and f_w are any differentiable function approximators for the policy and value function with parameters θ and w respectively. The policy parameters are updated according to Equation 3.25;

$$\theta_{k+1} = \theta_k + \alpha_k \sum_s d^{\pi_k}(s) \sum_a \frac{\delta \pi_k(s,a)}{\delta \theta} f_{wk}(s,a) \tag{3.25}$$

being α_k any step size sequence and $d^{\pi_k}(s)$ a discounted weighting of state transition probabilities encountered starting at s_0 and then following π: $d^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi\}$. $0 < \gamma < 1$ defines a discount rate. For more information on this work refer to [Sutton et al., 2000].

3.2.4 Actor-Critic solutions

Actor-Critic [Witten, 1977] is a methodology to solve RLP. These algorithms are considered "hybrid" algorithms. Actor-Critic algorithms have two distinctive parts. The *actor* and the *critic*. The actor part contains the policy to be followed. On the other hand, the critic observes the state evolution and criticizes the actions taken by the actor. The critic contains a value function which tries to learn according to the actor policy. Actor-critic methods try to combine the advantages of actor-only methods (policy search methods) with critic-only methods (value-based algorithms) similar to those done in the "hybrid" methodologies mentioned before. Actor-Critic algorithms have been successfully applied to various RL tasks: ASE/ACE architecture for pole balancing [Barto et al., 1983] [Gullapalli, 1992], RFALCON for pole balancing and for control of a ball-beam system [Lin and Lin, 1996].

Kimura's traced actor

The algorithm presented is an Actor-Critic, in which the actor updates a stochastic policy using eligibility traces of its parameters. The critic mission provides an appropriate baseline function to the actor using the value function. Studies carried out earlier, relating to Actor-Critic methods and the

use of eligibility traces focused on the critic. Kimura's algorithm proposes for the first time to apply them to the actor parameter update procedure. Therefore, the actor improves its policy using the gradient of its return.

Suppose the agent generates a control action according to its policy $\pi(a_t, W, s_t)$. In response, the environment generates a new state x_{t+1} and gives an immediate reward r_t. With this information, the critic is able to provide a TD-error to the actor following the equation:

$$(TD - error) = [r_t + \gamma \widetilde{V}(s_{t+1})] - \widetilde{V}(s_t) \tag{3.26}$$

where $0 \leq \gamma < 1$ is a discount factor and $\widetilde{V}(s)$ is an estimate of the value function computed by the critic. With the information offered by the critic, the actor upgrades its policy. First, it calculates the eligibility trace for every parameter of the policy:

$$e_i(t) = \frac{\delta}{\delta w_i} \ln(\pi(a_t, W, s_t)) \tag{3.27}$$

$$D_i(t) = e_i(t) + \beta D_i(t-1) \tag{3.28}$$

and then the actor policy parameters are updated following Equation 3.29 and Equation 3.30;

$$\triangle w_i(t) = (TD - error) D_i(t) \tag{3.29}$$

$$W \leftarrow W + \alpha_p \triangle W(t) \tag{3.30}$$

$0 \leq \beta < 1$ is a discount factor for the eligibility trace and α_p is the learning rate for the actor. Finally, the critic updates its value function according to TD methods. Taking a deeper look into the algorithm, reveals that this method is very similar to Kimura's policy search method presented in section 3.3.1. If $\gamma = \beta$ and the value function is replaced by a fixed reinforcement baseline; the two algorithms are identical. One of the main features of this algorithm is that both, the policy function and the value function, are completely independent, and it's possible that the actor learns the policy without the convergence of the critic's value function. If the critic approximates its value function, the actor's policy improvement will be accelerated. For more information on this work refer to [Kimura and Kobayashi, 1998].

Konda's projection

As in Kimura's Actor-Critic algorithm, Konda's actor is based on a parameterized function which is updated by gradient estimation and the critic is implemented using a TD algorithm, but of the Q-value function type. The author's main contribution is that, due to the small dimension of the parameters's vector updated by the actor compared with the space dimension, it is not necessary for the critic to find the exact approximated value function, but what they named a reduced "projection" of the function.

At any time step k, let r_k be the parameter vector of the critic, Z_k its eligibility trace and α_k a scalar estimate of the average cost. If (X_k, U_k) is the state-action pair at time k, the update procedure of the critic is stated as follows:

$$\alpha_{k+1} = \alpha_k + \gamma_k(c(X_k, U_k) - \alpha_k) \tag{3.31}$$

$$r_{k+1} = r_k + \gamma_k d_k Z_k \tag{3.32}$$

TD d_k is defined as:

$$d_k = c(X_k, U_k) - \alpha_k + r'_k \phi_{\theta_k}(X_{k+1}, U_{k+1}) - r'_k \phi_{\theta_k}(X_k, U_k) \tag{3.33}$$

where γ_k is a positive step size parameter. Besides, authors propose two different TD algorithms for updating the eligibility trace:

TD(1) *critic*:

$$z_{k+1} = \left\{ \begin{array}{cc} Z_k + \phi_{\theta_k}(X_{k+1}, U_{k+1}) & if X_{k+1} \neq x^* \\ \phi_{\theta_k}(X_{k+1}, U_{k+1}) & otherwise \end{array} \right\} \tag{3.34}$$

TD($0 < \lambda < 1$) *critic*:

$$Z_{k+1} = \lambda Z_k + \phi_{\theta_k}(X_{k+1}, U_{k+1}) \tag{3.35}$$

Finally, the actor updates its parameters according to:

$$\theta_{k+1} = \theta_k + \beta_k \Gamma(r_k) r'_k \phi_{\theta_k}(X_{k+1}, U_{k+1}) \psi_{\theta_k}(X_{k+1}, U_{k+1}) \tag{3.36}$$

where $\Gamma(*)$ is a scalar that controls the step size β_k of the actor, taking into account the current estimate r_k of the critic. For more information on this work refer to [Konda and Tsitsiklis, 2003].

3.2.5 Ng and Jordan's PEGASUS

One of the most important drawbacks of applying policy gradient methods to POMDPs are high variances in estimations, caused in part by stochastic state transitions. Thus, executing a fixed number of transitions starting from the same initial state can lead to obtaining different rewards. In order to reduce variance estimates, the authors present PEGASUS (Policy Evaluation-of-Goodness and Search Using Scenarios), a model-based, off-line, policy gradient algorithm designed for searching policies in MDPs or POMDPs, which proposes to transform a common POMDP with stochastic state transitions into a deterministic one. This process is done by recording the random numbers generated between each observed state transition and "fixing" them, so the simulated world is no longer stochastic. This trick allows us to compute precisely the average reward obtained for each trial, and we are assure that differences in gradient estimates are due to changes between policies and not to the variance introduced by transition between states. Furthermore, given n different simulated trials $s_0^1, ..., s_0^n$, the value function can be computed as a deterministic function:

$$\widehat{V}(\pi) = \frac{1}{n} \sum_{i=1}^{n} R'(s_0^i) + \gamma R'(s_1^i) + ... + \gamma^f R'(s_f^i) \tag{3.37}$$

where $s_0^i, s_1^i, ..., s_f^i$ is the sequence of states visited deterministically by the policy π, the initial state being s_0. Thus, the problem is transformed into a Monte-Carlo [Sutton and Barto, 1998] approach with the difference that its randomization is fixed, so it need only generate n samples and observe the reward obtained to adjust the policy π. Since $\widehat{V}(\pi)$ is a deterministic function, the policy search process used could be a standard optimization method. For more information on this work refer to [Ng and Jordan, 2000].

3.3 Summary

The table below summarizes the main features of the algorithms studied in this chapter. The table classifies the algorithms according to some defined features: The function used to evaluate the performance of the algorithm, the requirement of a model of the environment, the possibility to implement the algorithm on-line and if the method is biased. The already mentioned classification between pure and hybrid methods has also been included in this table.

Method	**Performance analysis**	**Biased est.**	**Use of a Model**	**Hybrid or Pure**	**On-line possibility**
REINFORCE	$\nabla\eta(\theta)$ + Baseline	No	Model-free	Pure	Yes
Kimura's	$\nabla\eta_\beta(\theta)$ + Baseline	Yes	Model-free	Pure	Yes
B&B	$\nabla\eta_\beta(\theta)$	Yes	Model-free	Pure	Yes
Marbach's	Differential Rewards	Yes	Model-free	Pure	Yes
Jaakkola's	$Q^\pi(s,a)$	Yes	Model-free	Hybrid	Yes
VAPS	Combination	Yes	Model-free	Hybrid	Yes
Sutton's	General funct. approx.	Yes	Model-free	Hybrid	Yes
Kimura's A/C	$V^\pi(s)$	Yes	Model-free	Hybrid	Yes
Konda's	$Q^\pi(s,a)$	Yes	Model-free	Hybrid	Yes
PEGASUS	$V^\pi(s)$	Yes	Model-based	Hybrid	No

Table 3.1: A summary of the algorithms described in Chapter 3.

Chapter 4

Policy Gradient Methods in Robotics

This chapter presents the main contributions obtained in real robot applications using some of the policy gradient methodologies mentioned in the previous chapter.

4.1 Bagnell's Helicopter Flight

As mentioned before, direct policy search has better capabilities of finding solutions compared with value-based methods when the environment is not totally observable (POMDP). The solution does not need to be optimal, and sub-optimal convergence is usually enough for structured controllers to perform well. Following this idea, Bagnell's team described the successful application of the PEGASUS policy search algorithm (see Section 3.2.5 for details) to design a controller for an autonomous helicopter flight.

The identification of the controller is performed off-line, so the learning procedure is carried out previously using a simulated model of the helicopter. When the controller achieves a reliable confidence level, the authors transferred the learned controller to the real machine. The controller is represented by a simple neural network where a total of 10 weights are the parameters to be updated during the learning process. The reward function depends on the helicopter translational deviation in the x and y axis. According to the PEGASUS learning procedure, a set of trajectories with a fixed horizon was used to modify the controller parameters to the optimal ones.

The results obtained show the viability of the algorithm in a difficult control problem. Future approaches will try the challenging task of implementing the learning on-line, intending to respond to critical failures

of the helicopter in real time. For more information on this work refer to [Bagnell and Schneider, 2001].

4.2 Rosenstein and Barto's Weightlifter

Although the results presented here are simulated, the implementation of Rosenstein and Barto's algorithm, despite its simplicity, is the essence of direct policy methods based on gradient computation. The authors designed what they called a "biologically motivated algorithm" that easily adapted through learning. The RL problem deals with a simulated weightlifter automata that tries to perform a payload shift task with a limited maximum torque at each of the three robot joints.

The parameterized policy is represented by a total set of 24 parameters which characterize two proportional-derivative (PD) controllers. Starting at a base point θ_0, the algorithm performs a *simple random search* (SRS) in the parameter dimension by adding a perturbation amount $\Delta\theta$ to each parameter. This amount is normally distributed with zero mean with a variance equal to a search size σ. Through iterations, each new point $\theta_0 + \Delta\theta$ is evaluated by a function which computes the total amount of torque accumulated during a trial. In order to assure exploration, the algorithm updates a previous point θ with the new one $\theta + \Delta\theta$, with a fixed probability β or the best one found at that moment with probability $1 - \beta$. The search size is also updated every iteration, modified by a decay factor γ until a minimum is reached σ_{min}.

The results obtained, even though they are considered good, are attributed not only to the SRS algorithm itself, but to other aspects such as the controller implementation and some experimental issues. The authors propose as future work the use of more sophisticated policy gradient methods than Baxter and Bartlett's or Sutton's. For more information on this work refer to [Rosenstein and Barto, 2001].

4.3 Kohl's Quadrupedal Locomotion

This approach applies an RL policy gradient method for a learning task where the commercial quadrupedal robot created by Sony, AIBO, tries to find the fastest possible walking speed. The robot's behavior is defined by a set of 12 parameters which refer to different aspects of AIBO's dynamics: front and rear locus, different robot heights,... Since Kohl's team does not know the form of its policy, it is impossible to compute its gradient exactly.

The solution is a degenerated form of a policy gradient algorithm, inspired by the one proposed by Baxter and Bartlett (see Section 3.1.3 for details).

Starting from an initial parameter vector, the algorithm generates a set of random policies by adding $(+\epsilon, 0, -\epsilon)$ to each parameter. This amount is a constant value that is relatively small compared to the parameter itself. Then the method evaluates each of the generated policies and estimates the partial derivative in each of the 12 dimensions of the parameter vector. After classifying the obtained results, an average score for each of the classifications is calculated and an adjustment vector can be constructed. Next iteration's policy is modified with this adjustment. The process is repeated until convergence.

The algorithm commented on here is not of the form as the others analyzed in our survey, but Kohl's method is also based on the gradient of the reward perceived and its results demonstrate the feasibility of policy gradient methods in practicable robot applications. The simplicity of the algorithm used in the calculation of AIBO's gait parameters reinforce the use of policy gradient methods in real unknown environments, since its results show that they generated one of the fastest walks known for this quadrupedal robot. For more information on this work refer to [Kohl and Stone, 2004].

4.4 Tedrake's Simple 3D Biped

The learning problem studied here deals with a biped robot walking from a blank-state. The RL algorithm used to carry out the learning procedure is a policy gradient hybrid method originally proposed by Kimura (presented in Section 3.2.4). Small modifications of this algorithm improve the weight update step, guiding it in the direction of the performance gradient. The algorithm used in this application is an Actor-Critic, in which the actor updates an stochastic policy using eligibility traces of its parameters. The critic's mission is to provide an appropriate baseline for the actor using some kind of value function. The simplicity of the robot presented allows the authors to learn a policy easily with a single output which controlled a 9 DOF system.

As stated previously, policy gradient actor-critic algorithms use two function approximators, one implemented on the actor assigned to execute the policy, and another for the approximation of the value function used by the critic to judge the actions taken by the actor. Tedrake's algorithm uses in both functions parameterized linear approximators using non linear features.

The algorithm tested in this simplified robot seems to perform well, and once the policy is learned, the robot easily adapts to small changes in the

terrain. Moreover, the authors propose working further in the same direction in order to reduce biped constraints and study the possibility of applying the same algorithm to more sophisticated bipeds. For more information on this work refer to [Tedrake et al., 2004].

4.5 Matsubara's Biped Robot

The work presented here demonstrates the feasibility of policy gradient methods applied to a real biped locomotion problem. In particular, as in a previous section, the authors apply Kimura's actor-critic algorithm (presented in Section 3.2.4) to its RLP. The partial observability of its space dimension conceives the learning framework as a POMDP, thus, the impossibility of computing the exact value function directs the authors to using a policy gradient actor-critic algorithm. The learning procedure will be performed on-line. The characteristics of the environment justify the aid of a value function for rapid learning. If the biped does not learn or takes too long to do so, it will fall down and the learning process finishes without reaching the objectives.

The biped controller is a CPG (central pattern generator) composed of a neural oscillator, whose weights are the policy parameters to be learnt by the actor. The critic is represented by a value function with its own set of parameters. Both policy and value approximators are modeled using a normalized Gaussian network (NGnet). The reward function combines the maintenance of the upright position as the first requirement and the forward progress in a second term.

The results obtained show an efficient learning policy gradient method able to converge within a short number of trials. On the other hand, the authors noted the importance of a good sensory feedback controller in order to achieve satisfactory results. For more information on this work refer to [Matsubara et al., 2005].

Chapter 5

Neural Policy Gradient Algorithm (NPGA)

The objective of this research work is focused on the study of stochastic policy gradient methods as an alternative substitute to value-based algorithms in on-line robot learning. To this end, after comparing the advantages and drawbacks of different methodologies analyzed in previous sections, the selected algorithm for this initial approach to policy gradient methods is Baxter and Bartlett's direct gradient method. The small computational requirements of the method and its basic mathematical foundation make this algorithm an easy-to-follow methodology and a good test platform for the experiments proposed in the next chapter. Baxter and Bartlett's algorithm procedure is summarized in Algorithm 1.

Algorithm 1: Baxter and Bartlett's OLPOMDP algorithm

1. Initialize:
   $T > 0$
   Initial parameter values $\theta_0 \in R^K$
   Initial state i_0
2. Set $z_0 = 0$ ($z_0 \in R^K$)
3. $for\ t = 0$ to T do:
   (a) Observe state y_t
   (b) Generate control action u_t according to current policy $\mu(\theta, y_t)$
   (c) Observe the reward obtained $r(i_{t+1})$
   (d) Set $z_{t+1} = \beta z_t + \frac{\nabla \mu_{u_t}(\theta, y_t)}{\mu_{u_t}(\theta, y_t)}$
   (e) Set $\theta_{t+1} = \theta_t + \alpha_t r(i_{t+1}) z_{t+1}$
4. $end\ for$

The algorithm works as follows: having initialized the parameters vector θ_0, the initial state i_0 and the eligibility trace $z_0 = 0$, the learning procedure

will be iterated T times. At every iteration, the parameters' eligibility z_t will be updated according to the policy gradient approximation. The discount factor $\beta \in [0, 1)$ increases or decreases the agent's memory of past actions. The immediate reward received $r(i_{t+1})$, and the learning rate α allows us to finally compute the new vector of parameters θ_{t+1}. The current policy is directly modified by the new parameters becoming a new policy to be followed by the next iteration, getting closer to a final policy that represents a correct solution of the problem.

As aforementioned, the algorithm is designed to work on-line. The function approximator adopted to define our policy is an artificial neural network (ANN) whose weights represent the policy parameters to be updated at every iteration step. Thus, the *neural policy gradient algorithm* (NPGA) is defined.

For a better understanding, the next lines will relate closely to the update weight process done by the algorithm. Once the ANN is initialized at random, the network will be given an observation of the state and, as a result, a stochastic control action is computed. As a result, the learner will be driven to another state and will receive a reward associated with this new state. The first step in the parameter update procedure is to compute the ratio:

$$\frac{\nabla \mu_{u_t}(\theta, y_t)}{\mu_{u_t}(\theta, y_t)} \tag{5.1}$$

for every weight of the network. In artificial neural networks like the one used in the algorithm, the expression defined in step 3.d of Algorithm 1 can be rewritten as:

$$z_{t+1} = \beta z_t + \delta_t y_t \tag{5.2}$$

At any step time t, the term z_t represents the estimated gradient of the reinforcement sum with respect to the network's layer weights. In addition, δ_t refers to the local gradient associated with a single neuron of the ANN and is multiplied by the input to the neuron y_t. In order to compute these gradients, we evaluate the soft-max distribution for each possible future state exponentiating the real-valued ANN outputs $\{o_1, ..., o_n\}$, being n the number of neurons of the output layer [Aberdeen, 2003].

After applying the soft-max function, the outputs of the neural network give a weighting $\xi_j \in (0, 1)$ to each of the possible control actions. Finally, the probability of the ith control action is then given by:

$$Pr_i = \frac{\exp(o_i)}{\sum_{a=1}^{n} \exp(o_a)} \tag{5.3}$$

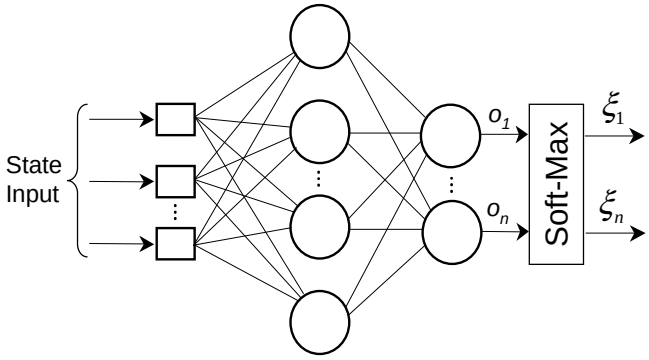

Figure 5.1: Schema of the ANN architecture adopted.

where n is the number of neurons at the output layer. Actions have been labeled with the associated control action and chosen at random from this probability distribution. Once we have computed the output distribution over all possible actions, the next step is to calculate the gradient for the action chosen by applying the chain rule. The whole expression is implemented similarly to error back propagation [Haykin, 1999]. Before computing the gradient, the error on the neurons of the output layer must be calculated. This error is given by Equation 5.4.

$$e_j = d_j - Pr_j \tag{5.4}$$

The desired output d_j will be equal to 1 if the action selected was o_j, and 0 otherwise (see Figure 5.2).

With the soft-max output error calculation completed, the next phase consists of computing the gradient at the output of the ANN and back propagate it to the rest of the neurons of the hidden layers. For a local neuron j located in the output layer, we may express the local gradient as:

$$\delta_j^o = e_j \varphi_j'(o_j) \tag{5.5}$$

where e_j is the soft-max error at the output of neuron j, $\varphi_j'(o_j)$ corresponds to the derivative of the activation function associated with that neuron, and o_j is the function signal at the output for that neuron. So we do not back propagate the gradient of an error measure, but instead back propagate the soft-max gradient of this error. Therefore, for a neuron j located in a hidden layer, the local gradient is defined as follows:

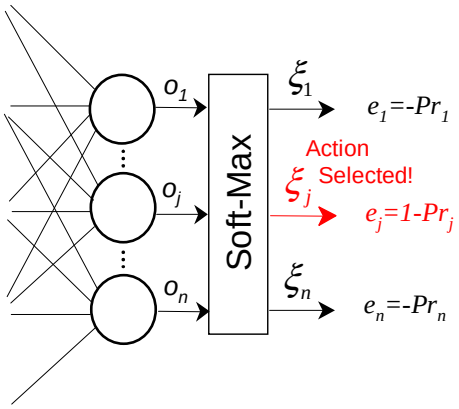

Figure 5.2: Soft-Max error computation for every output.

$$\delta_j^h = \varphi'_j(o_j) \sum_k \delta_k w_{kj} \tag{5.6}$$

When computing the gradient of a hidden-layer neuron, the previously obtained gradient of the following layers must be back propagated. In Equation 5.6 the term $\varphi'_j(o_j)$ represents the derivative of the activation function associated to that neuron, o_j is the function signal at the output for that neuron and finally the summation term includes the different gradients of the following neurons back propagated by multiplying each gradient to its corresponding weighting (see Figure 5.3).

Having all the local gradients of all the neurons calculated, the expression in Equation 5.2 can be obtained. Finally, the old parameters are updated

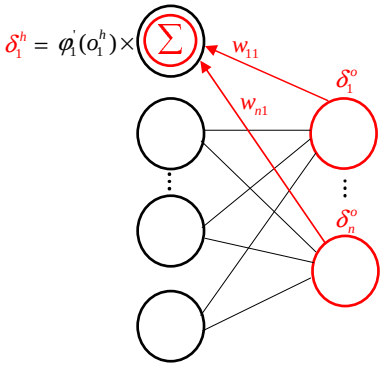

Figure 5.3: Gradient computation for a hidden layer neuron.

following expression 3.(e) of Algorithm 1:

$$\theta_{t+1} = \theta_t + \alpha r(i_{t+1}) z_{t+1} \tag{5.7}$$

The vector of parameters θ_t represents the network weights to be updated, $r(i_{t+1})$ is the reward given to the learner at every time step, z_{t+1}describes the estimated gradients mentioned before and, at last, we have α as the learning rate of the algorithm.

Chapter 6

Experimental Results

This chapter presents the experiments carried out in this research project applying the *neural policy gradient algorithm* (NPGA) detailed in Chapter 5. The chapter is organized into two main sections. In the first section, details of a simulated robot task are presented. In the second, results on a RL benchmark will point out the suitability of policy methods in problems influenced by the generalization problem.

6.1 Target-following task

The experiment consisted of learning a target following behavior using the underwater robot URIS. Experiments were performed in simulation. The next lines will describe the different elements that took place in our problem. First, the simulated world will be detailed. Secondly, the underwater vehicle URIS is presented. Section 6.1.3 describes the neural-network controller. Finally, some results and conclusions are given.

6.1.1 The world

As previously mentioned, the problem deals with the simulated model of the autonomous underwater vehicle (AUV) URIS navigating a two-dimensional world constrained in a plane region without boundaries. The vehicle can be controlled by two degrees of freedom (DOF), surge (X movement) and yaw (rotation with respect to z-axis) by applying 4 different control actions: a force in either the positive or negative surge direction, and another force in either the positive or negative yaw rotation. The simulated robot was given a reward of 0 if the vehicle reaches the objective position (if the robot enters a circle with a 1 unit radius, the target is considered reached) and a reward

equal to -1 in all other states. To encourage the controller to learn how to control the robot and reach the target independently of the starting state, the AUV position was reset every 50 (simulated) seconds to a random location in x and y between [-20, 20], and, at the same time, the target position was set to a random location within the same boundaries. The sample time was set to 0.1 seconds.

6.1.2 URIS AUV description

*U*nderwater *R*obotic *I*ntelligent *S*ystem is indicated by the acronym URIS. This Unmanned Underwater Vehicle (UUV) is the result of a project started in 2000 at the University of Girona. The main purpose of this project was to develop a small-sized underwater robot with which to experiment easily in various research areas such as control architectures, dynamics modeling and underwater computer vision. Another goal of the project was to develop an Autonomous Underwater Vehicle (AUV) with the required systems, hardware and software as the word *autonomous* implies. Other principles considered in the design were flexibility in the tasks to be accomplished and generalization in the developed systems.

The design of this vehicle was clearly influenced by its predecessor Garbi UUV [Amat et al., 1996], although some mechanical features were redesigned. The shape of the vehicle is compounded of a spherical hull surrounded by various external elements (the thrusters and camera sensors). The hull is made of stainless steel with a diameter of 350mm, designed to withstand pressures of 3 atmospheres (30 meters depth). On the outside of the sphere there are two video cameras (forward and down looking) and 4 thrusters (2 in X direction and 2 in Z direction). Figure 6.1 shows a picture of URIS and its body fixed coordinate frame. Referred to this frame, the 6 degrees of freedom (DOFs) in which a UUV can be moved are: *surge*, *sway* and *heave* for the motions in X, Y and Z directions respectively; and *roll*, *pitch* and *yaw* for the rotations about the X, Y and Z axes respectively.

URIS weighs 30 Kg., which is approximately equal to the mass of the water displaced and, therefore, the buoyancy of the vehicle is almost neutral. Its gravity center is in the Z axis, at some distance below the geometrical center. The reason for this is the distribution of the weight inside the sphere. The heavier components are placed at the bottom. The difference between the two centers provides stability in both *pitch* and *roll* DOFs. The further down the gravitational center is, the higher the torque which has to be applied to the X or Y axes to incline the robot a certain degree in *roll* or *pitch*, respectively.

The movement of the robot is accomplished by its 4 thrusters. Two

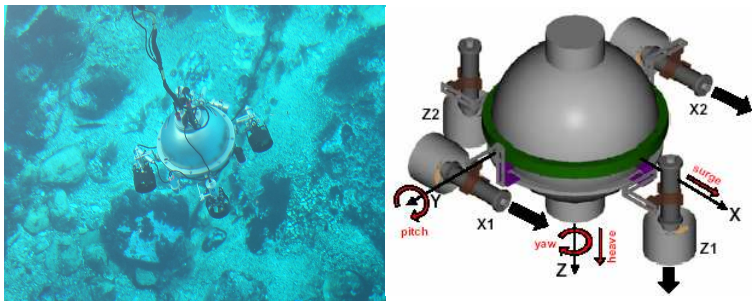

Figure 6.1: URIS' AUV, (Left) URIS in experimental test. (Right) Robot reference frame.

of them, labeled *X1* and *X2* in Figure 6.1(Right), exert a force in the X axis and a torque in the Z axis. The resultant force of both trusters is responsible for the *surge* movement of the vehicle, and the resultant torque is responsible for the *yaw* movement. Analogously, the other two thrusters, Z1 and Z2, exert a force in the Z axis and a torque in the Y axis. The resultant force is responsible for the *heave* movement of the vehicle, and the resultant torque is responsible for the *pitch* movement. In this case, the *pitch* movement is limited to only a few degrees around the stable position, since the gravity and buoyancy forces cause a high stabilization torque compared to that of the thruster. Therefore, only 4 DOFs can be actuated leaving the *sway* and *roll* movements without control. Like the *pitch* DOF, the *roll* DOF is stabilized by the gravity and buoyancy forces. The *sway* movement is neither controlled nor stabilized by any force, which makes it sensitive to perturbations like water currents or the force exerted by the umbilical cable. Hence, URIS is a nonholonomic vehicle.

The mathematical model of URIS has been computed using parameter identification methods [Ridao et al., 2004]. The whole model has been adapted to the problem so the hydrodynamic equations of motion for an underwater vehicle with 6 DOFs [Fossen, 1995] have been uncoupled and reduced to moderate a robot with two DOFs. Let us consider the dynamic equations for the surge and yaw DOFs:

$$\dot{u} = \underbrace{\frac{X}{(m - X_{\dot{u}})}}_{\gamma} - \underbrace{\frac{X_u}{(m - X_{\dot{u}})}}_{\alpha} \cdot u - \underbrace{\frac{X_{u|u|}|u|}{(m - X_{\dot{u}})}}_{\beta} \cdot u + \underbrace{\frac{\tau_p}{(m - X_{\dot{u}})}}_{\delta} \tag{6.1}$$

	α	β	γ	δ
UNITS	$\frac{N*s}{Kg*m}$	$\frac{N*s^2}{Kg*m^2}$	Kg^{-1}	$\frac{N}{Kg}$
SURGE	-0.3222	0	0.0184	0.0012
YAW	1.2426	0	0.5173	-0.050

Table 6.1: URIS model parameters for Surge and Yaw.

$$\dot{r} = \underbrace{\frac{N}{(m - N_{\dot{r}})}}_{\gamma} - \underbrace{\frac{N_r}{(m - N_{\dot{r}})}}_{\alpha} \cdot r - \underbrace{\frac{N_{r|r|}|r|}{(m - N_{\dot{r}})}}_{\beta} \cdot r + \underbrace{\frac{\tau_p}{(m - N_{\dot{r}})}}_{\delta} \tag{6.2}$$

Then, due to the identification procedure [Ridao et al., 2004], expressions in Equation 6.1 and Equation 6.2 can be rewritten as follows:

$$\dot{v}_x = \alpha_x v_x + \beta_x v_x |v_x| + \gamma_x \tau_x + \delta_x \tag{6.3}$$

$$\dot{v}_\psi = \alpha_\psi v_\psi + \beta_\psi v_\psi |v_\psi| + \gamma_\psi \tau_\psi + \delta_\psi \tag{6.4}$$

where $\dot{v}_x$ and $\dot{v}_\psi$ represent the acceleration in both the surge and yaw DOFs, v_x is the linear velocity in surge and v_ψ is the angular velocity in the yaw DOF. The force and torque excerted by the thrusters in both DOFs are indicated as τ_x and τ_ψ. The model parameters for both DOFs are stated as follows: α and β coefficients refer to the linear and the quadratic damping forces, γ represents a mass coefficient and the bias term is introduced by δ. The values of the identified parameters are indicated in Table 6.1

6.1.3 The controller

A one-hidden-layer neural-network with 4 input nodes, 3 hidden nodes and 4 output nodes was used to generate a stochastic policy. One of the inputs corresponds to the distance between the vehicle and the target location, another represents the yaw difference between the vehicle's current heading and the desired heading to reach the objective position. The other two inputs represent the derivatives of the distance and yaw difference at the current time-step. Each hidden and output layer has the usual additional bias term. The activation function used for the neurons of the hidden layer is the hyperbolic tangent type (see Equation 6.5 and Figure 6.2), while the output layer nodes are linear. The four output neurons have been exponentiated and

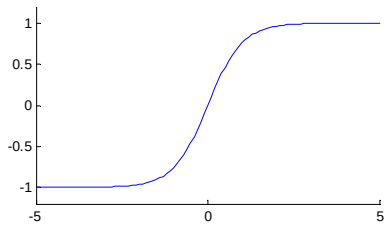

Figure 6.2: The hyperbolic tangent function.

normalized as explained in Chapter 5 to produce a probability distribution. Control actions are selected at random from this distribution.

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} \tag{6.5}$$

6.1.4 Learning Results

The controller was trained in an episodic task. Robot and target positions were reset every 50 seconds so the total amount of reward per episode perceived varies depending on the episode. Even though the results presented have been obtained, as explained in Section 6.1.1, in order to clarify the graphical results of time convergence of the algorithm, some constraints have been applied to the simulator: Target initial position was fixed to $(0, 0)$ and robot initial location was set to four random locations, $x = \pm 20$ and $y = \pm 20$. Therefore, the total amount per episode when converged was the same.

The number of episodes was set to 100.000. For every episode, the total amount of reward perceived was calculated. Figure 6.3 represents the performance of the NPGA as a function of the number of episodes. The episodes were averaged over bins of 50 episodes. The experiment was repeated over 100 independent runs, and the results presented are the mean of these runs. The simulated experiments were repeated and compared for different values of α and β.

As can be appreciated in Figure 6.4, the optimal performance (within the neural network controller used here) is around -100 for this simulated problem due to the fact that the puck and target locations are reset every 50 seconds and for this reason the vehicle must be away from the target a fraction of the time. The best results are obtained when $\alpha = 0.00001$ and $\beta = 0.999$, see Figure 6.4.

Figure 6.6 represents the behavior of the trained robot controller. For the purpose of the illustration, only the target location was reset to a random

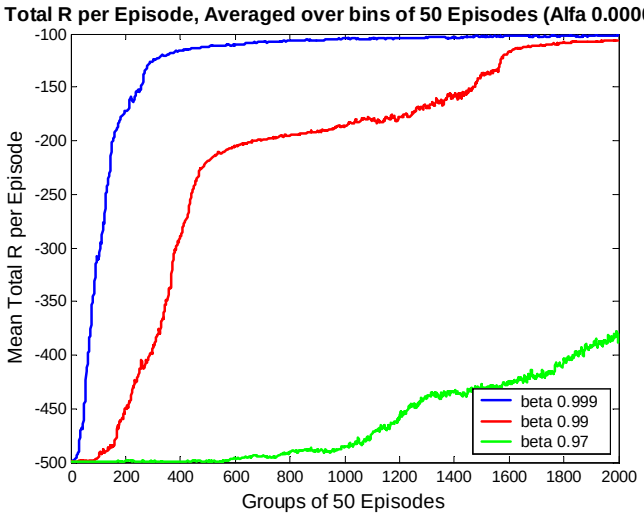

Figure 6.3: Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generated by simulating 100.000 episodes, averaging them over bins of 50 episodes. The process was repeated over 100 independent runs. The results are a mean of these runs. $\alpha = 0.000001$, for different values of $\beta = \{0.999, 0.99, 0.97\}$.

location, not the robot location.

6.1.5 Conclusions and discussion

The results obtained show a convergence of the NPGA. As can be seen Figure 6.6, once learning period finishes, the trained robot is able to fulfill the target following task. The algorithm's implementation is easy and simple. Although the network used to generate the policy is small the learning results are satisfactory. In order to obtain definitive results related to convergence time, the NPGA must be compared to a value-based or a hybrid algorithm in the same task. Thus, the results present policy Methods as an alternative approach to Value Methods to solve Reinforcement Learning problems.

6.2 “Mountain-Car” task

This section presents the application of the NPGA to the “mountain-car” benchmark. This problem is widely accepted by the RL research community as a convenient benchmark to test the convergence and generalization capabilities of an RL algorithm. The ”mountain-car” benchmark is not a

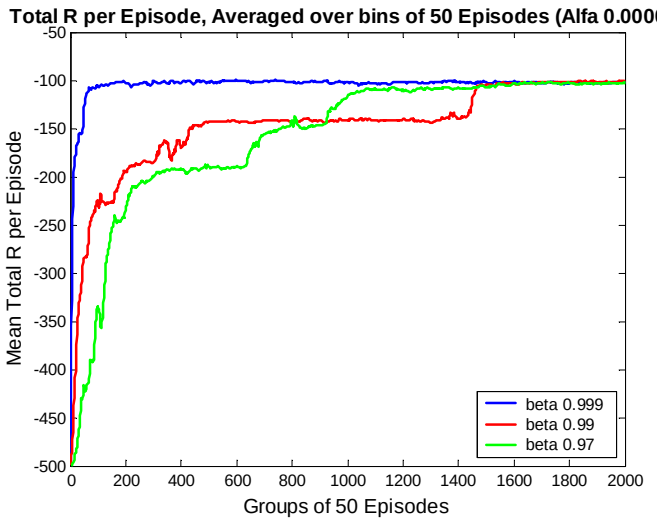

Figure 6.4: Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generated by simulating 100.000 episodes, averaging them over bins of 50 episodes. The process was repeated over 100 independent runs. The results are a mean of these runs. $\alpha = 0.00001$, for different values of $\beta = \{0.999, 0.99, 0.97\}$.

continuous task like a robot behavior, but an episodic task. Moreover, contrary to a robot behavior, the environment is completely observable without noise. These qualities make this problem an excellent platform for result analysis. In order to have a performance baseline, the Q_learning algorithm was also applied to the problem. The results obtained by both algorithms are compared and some conclusions are extracted at the end of this section.

6.2.1 “Mountain-Car” task definition

The ”mountain-car” task [Moore, 1991, Singh and Sutton, 1996] was designed to evaluate the generalization capability of RL algorithms. In this problem, a car has to reach the top of a hill. However, the car is not powerful enough to drive straight to the goal. Instead, it must first reverse up the opposite slope in order to accelerate, acquiring enough momentum to reach the goal. The states of the environment are two continuous variables: the position p and the velocity v of the car. The bounds of these variables are $-1.2 \leq p \leq 0.5$; and $-0.07 \leq v \leq 0.07$. Action a is a discrete variable with two values $\{-1, +1\}$, which correspond to reverse thrust and forward thrust respectively. The mountain geography is described by the equation:

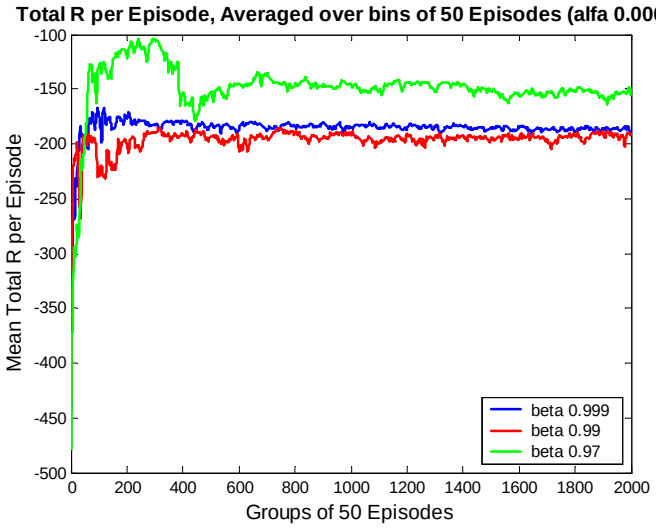

Figure 6.5: Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generated by simulating 100.000 episodes, averaging them over bins of 50 episodes. The process was repeated over 100 independent runs. The results are a mean of these runs. $\alpha = 0.0001$, for different values of $\beta = \{0.999, 0.99, 0.97\}$.

$altitude = sin(3p)$. Figure 6.7 shows the ”mountain-car” scenario. The dynamics of the environment, which determines the state evolution, is defined by these two equations:

$$v_{t+1} = bound[v_t + 0.001\, a_t - 0.0025\, cos(3\, p_t)] \tag{6.6}$$

$$p_{t+1} = bound[p_t + v_{t+1}] \tag{6.7}$$

in which the *bound* operation maintains each variable within its allowed range. If p_{t+1} is smaller than its lower bound, then v_{t+1} is reset to zero. On the other hand, if p_{t+1} achieves its higher bound, the episode finishes since the task is accomplished. The reward is -1 everywhere except at the top of the hill, where it is 1. New episodes start at random positions and velocities and run until the goal has been reached or a maximum of 200 iterations have elapsed. The optimal state/action mapping to solve the ”mountain-car” task is not trivial since, depending on the position and the velocity, a forward or reverse action must be applied.

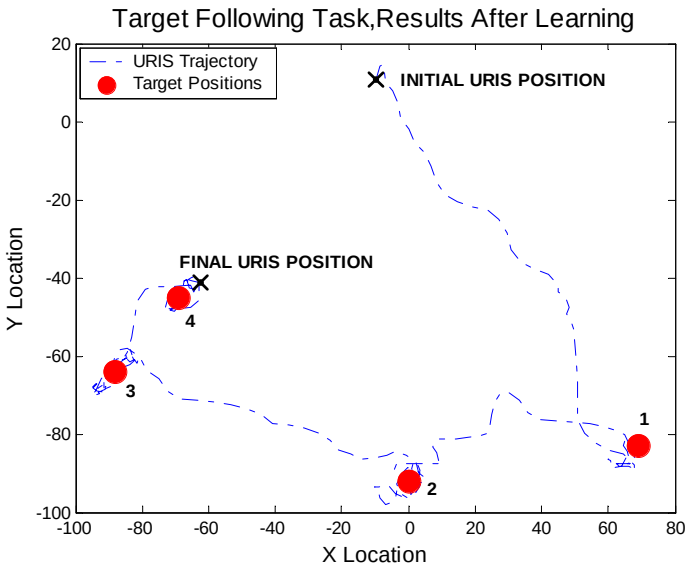

Figure 6.6: The behavior of a trained robot controller. Results of a target following task after the learning period has been completed.

6.2.2 Results with the Q_learning algorithm

To provide a performance baseline, the classic Q_learning algorithm was applied. The state space was finely discretized, using 180 states for the position and 150 for the velocity. The action space contained only three values, -1, 0 and 1. Therefore, the size of the Q table was 81000 positions. The exploration strategy was an $\epsilon - greedy$ policy with ϵ set at 30%. The discount factor was $\gamma = 0.95$ and the learning rate $\alpha = 0.5$, which were found experimentally. The Q table was randomly generated at the beginning of each experiment. In each experiment, a learning phase and an evaluation phase were repeatedly executed. In the learning phase, a certain number of iterations were executed, starting new episodes when it was necessary.

In the evaluation phase, 500 episodes were executed. The policy followed in this phase was the *greedy* policy, since only exploitation was desired. In order to numerically quantify the *effectiveness* of the learning, the average time spent in each episode is used. This time is measured as the number of iterations needed by the current policy to achieve the goal. After running 100 experiments with Q_learning, the average episode length in number of iterations once the optimal policy had been learnt was 50 iterations with 1.3 standard deviation. The number of learning iterations to learn this optimal policy was approximately 10^7. Figure 6.8 shows the effectiveness evolution

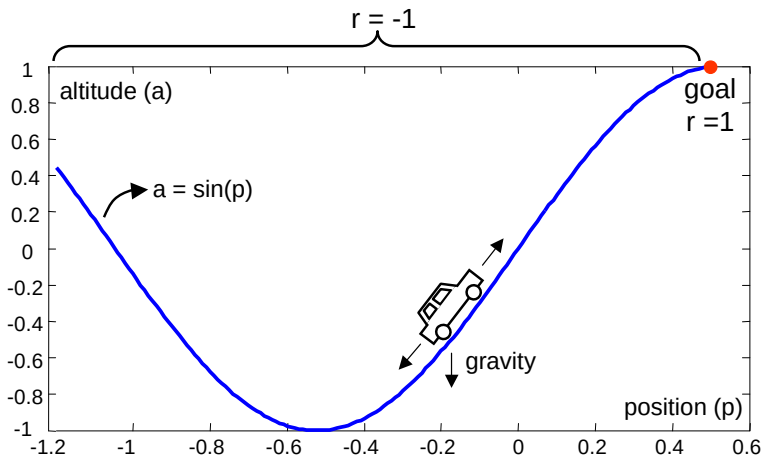

Figure 6.7: The ”mountain-car” task domain.

of the Q_learning algorithm after different learning iterations.

It is interesting to compare this mark with other state/action policies. If a forward action ($a = 1$) is always applied, the average episode length is 86. If a random action is used, the average is 110, see Figure 6.8. These averages depend highly on the fact that the maximum number of iterations in an episode is 200, since in many episodes these policies do not fulfill the goal.

6.2.3 Results with the policy gradient algorithm

A one-hidden-layer neural-network with 2 input nodes, 10 hidden nodes and 2 output nodes was used to generate a stochastic policy. One of the inputs corresponded to the vehicle's position, the other represented the vehicle's velocity. Each hidden and output layer had the usual additional bias term. The activation function used for the neurons of the hidden layer was the hyperbolic tangent type, see Section 6.1.3 for details. The output layer nodes were linear. The two output neurons were exponentiated and normalized as explained in Chapter 5 to produce a probability distribution. Control actions were selected at random from this distribution.

In each experiment, a learning phase and an evaluation phase were repeatedly executed. In the learning phase, 500 iterations were executed, starting new episodes when it was necessary. In the evaluation phase, 200 episodes were executed. The *effectiveness* of learning was evaluated by looking at the averaged number of iterations needed to finish the episode. After running 100 experiments with the NPGA, the average number of iterations, when the

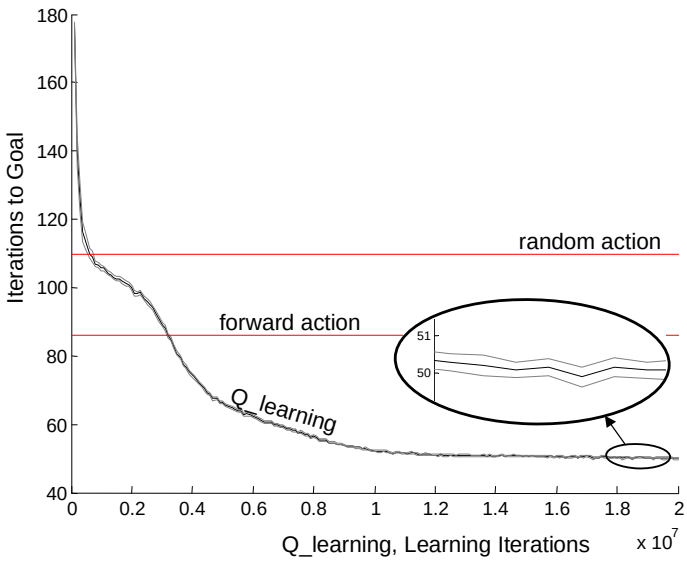

Figure 6.8: Effectiveness of the Q_learning algorithm with respect to the learning iterations. During the first iterations the efficiency was very low, requiring many iterations to reach the goal. The graph was obtained by averaging 100 trials. In each trial, the effectiveness was calculated by averaging the number of iterations to reach the goal in 500 episodes. After converging, the effectiveness was maximum, requiring only 50 iterations to accomplish the goal. The 95% confidence intervals are also shown. Finally, the effectiveness levels of random and forward policies can be observed.

optimal policy had been learnt, was 52.5. And the number of learning iterations to learn this optimal policy was 40.000 learning iterations. Figure 6.9 shows the effectiveness evolution of the NPGA over the learning iterations.

6.2.4 Conclusions and discussion

The results obtained reinforce Policy Methods as an alternative to Value Methods to solve Reinforcement Learning problems. After performing the experiments with the Q_learning algorithm and the policy gradient algorithm, it can be concluded:

Simplicity A very simple ANN configuration was able to learn the necessary policy. However, Q_learning, which was affected by the generalization problem, required 81000 cells to obtain a similar policy.

Effectiveness The *minimum iterations to goal* achieved by NPGA (52.5)

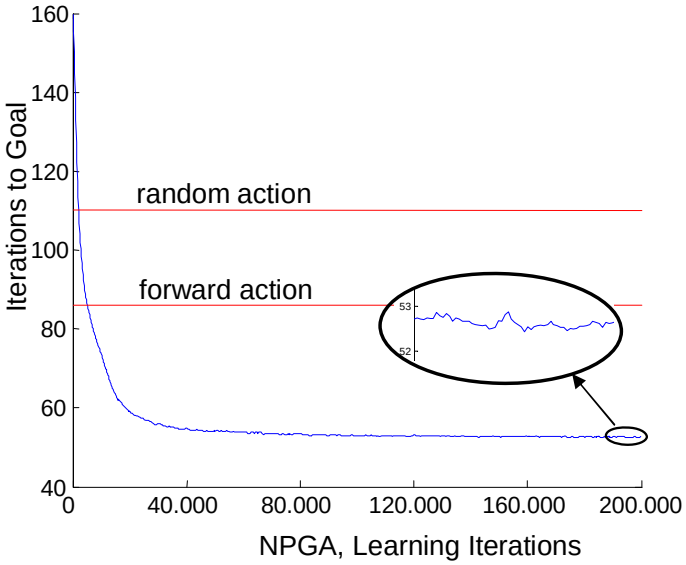

Figure 6.9: Effectiveness of the NPGA with respect to the learning iterations. The graph was obtained by averaging 100 independent trials. In each one, the effectiveness was calculated by averaging the number of iterations to finish the episode. In the learning phase, 500 number of iterations were executed, starting new episodes when it was necessary. In the evaluation phase, 200 episodes were executed.

was practically equal to those achieved by Q_learning (50).

Swiftness Although policy methods usually learn slower than value methods, in this case the NPGA algorithm was much faster than Q_learning (affected by the generalization problem).

Chapter 7

Conclusions and Future Work

This chapter concludes this research project. It first summarizes the main conclusions that have been extracted from the studies, analysis and implementations presented in this dissertations. Then, it points out the directions that will be followed in order to propose a novel policy gradient algorithm to be used on-line in a real robotic application. This work should finally contribute with policy gradient methods and, therefore, conduct to the PhD thesis. Finally, the expected planning is also presented.

7.1 Conclusions

The fundamental theory of policy methods has been surveyed. In this research project several direct policy gradient algorithms have been presented. Some of them showed theoretical demonstrations of convergence and others were applied to real robot systems. The main features have been analyzed and compared.

The methodologies commented can be classified into two main groups: *Pure policy gradient algorithms* and *hybrid policy-valued methods*. Pure policy gradient algorithms use a function approximator to represent the policy. Gradient estimates of the current's policy performance with respect to the function parameters lead the algorithm to convergence. High variances of these estimations slow down the convergence time of these algorithms. In order to reduce variance estimates and increase learning speed, hybrid policy-valued algorithms combine both policy gradient methods with value function approaches. Pure value methods do not guarantee convergence when dealing with POMDPs but, on the other hand, pure policy algorithms have the capability of mildly adapting to non-Markov environments. Thus, the resulting combined algorithm takes advantages of the strong points of both methods.

Two simulated experiments were carried out for this dissertation. The objective of this second part was to show the implementation details of a policy gradient algorithm and to point out its suitability in two applications. The selected method was Baxter and Bartlett's algorithm. This method is a pure policy gradient algorithm. Due to small computational requirements and its basic mathematical foundation, this algorithm constitutes an easy-to-follow methodology and was considered a good test platform for obtaining initial results in this field. An ANN was used as a function approximator to represent the policy. Its weights represent the policy parameters to be updated at every iteration step. Thus, the *neural policy gradient algorithm* (NPGA) has been clearly defined. This algorithm was applied in two different simulated problems. First, the NPGA controlled an underwater robot performing a target following task. Secondly, the algorithm was tested in the "mountain-car task" benchmark. The obtained results in both simulations showed a good performance of the algorithm. The convergence times were not too long if we consider that a classical value method would have been affected by the generalization problem and, therefore, spent much much more iterations to converge. It is also important to note the reduced dimensions of the ANN used in both tests. The results about the underwater robot have been published in [El-Fakdi et al., 2005]

7.2 Future Work

As aforementioned, all the experimentation was carried out in simulation. Desirable future steps would be directed as enumerated in the following:

STEP 1. Simulated learning + Real testing. The objective of this initial step is to transfer an accurate learned policy (in a simulated environment) to the real robot and test the behavior of the policy in real conditions.

STEP 2. Simulated learning + Real learning + Real testing. A basic learning will be performed in simulation with a simple model. The learnt policy will be transferred to the robot and will be refined on-line while testing.

STEP 3. Real learning + Real testing. The learning process will be carried out on-line with the robot navigating in a real underwater environment.

The goal pursued in our research is to achieve at least STEP 2. We are not sure that STEP 3 can be achieved or, if it is worth this achievement

in relation with the price that could imply: long convergence time, very simple applications, structured environments, high computational requirements,... In any case, the convergence time will always be important and NPGA may not be as fast as hybrid algorithms like Matsubara's biped or the work presented in by Tedrake's team. On the other hand, pure policy gradient algorithms obtain better results in POMDPs. Therefore, the future work will consist on:

PHASE A Testing the NPGA in STEP 1 and 2, and considering the possibility os achieving STEP 3.

PHASE B Testing a hybrid algorithm in STEP 1 and 2, and considering the possibility of achieving STEP 3.

After this implementations some analysis and conclusions will be extracted. Depending on the results and the problems appeared during this period, we expect to find out a novel methodology that can improve the results for our underwater robotics task domain. The theoretical work together with real experimentation should contribute the topic of policy gradient methods applied to robotics, and should conduct to the fulfillment of the PhD thesis.

7.3 Planning

The next table details the expected planning of work for the next months:

	2005							2006											2007		
	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
Experimental work	E1	E1	E1	E2	E2	E2	E2				E3	E3	E3			E4	E4	E4				
Theoretical work			T1	T2		T2	T2	T3	T3	T4	T4		T5	T5	T5	T5		T6	T6	T6		
Dissemination of results		D1	D1	D1	D2				D3					D4					D5	D6	D6	D6

E1 → Experiments concerning phase A.
E2 → Experiments concerning phase B.
E3 → Preliminary PhD results.
E4 → Final PhD results.

T1 → Analysis of phase A.
T2 → Analysis of phase B.
T3 → Conclusions phase A and B.
T4 → Improved Policy Methods hypothesis
T5 → PhD contributions formulation
T6 → Conclusions of PhD. Final results

D1 → Survey on policy methods
D2 → Paper about phase A
D3 → Paper about phase A and B. Conclusions
D4 → Paper about preliminary PhD results
D5 → Paper about final PhD results
D6 → PhD dissertation

Bibliography

[Aberdeen, 2003] Aberdeen, D. A. (2003). *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australian National University.

[Amat et al., 1996] Amat, J., Batlle, J., Casals, A., and Forest, J. (1996). GARBI: a low cost ROV, constrains and solutions. In *6ème Seminaire IARP en robotique sous-marine*, pages 1–22, Toulon-La Seyne, France.

[Anderson, 2000] Anderson, C. (2000). Approximating a policy can be easier than approximating a value function. Computer science technical report, University of Colorado State.

[Bagnell and Schneider, 2001] Bagnell, J. and Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Korea.

[Baird and Moore, 1999] Baird, L. and Moore, A. (1999). Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems*, Vol 11, MIT Press.

[Barto and Duff, 1994] Barto, A. and Duff, M. (1994). *Monte-Carlo matrix inversion and reinforcement learning*, volume 6. Morgan Kaufmann.

[Barto et al., 1983] Barto, A., Sutton, R., and Anderson, C. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13:835–846.

[Baxter and Bartlett, 1999] Baxter, J. and Bartlett, P. (1999). Direct gradient-based reinforcement learning: I. gradient estimation algorithms. Technical report, Australian National University.

[Baxter and Bartlett, 2000] Baxter, J. and Bartlett, P. (2000). Direct gradient-based reinforcement learning. In *International Symposium on Circuits and Systems*, Geneva, Switzerland.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princenton University Press.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

[Carreras, 2003] Carreras, M. (2003). *A Proposal of a Behavior-Based Control Architecture with Reinforcement Learning for an Autonomous Underwater Robot*. PhD thesis, University of Girona.

[Carreras et al., 2003] Carreras, M., Ridao, P., and El-Fakdi, A. (2003). Semi-Online Neural-Q-learning for real-time robot learning. In *IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS'03)*, Las Vegas, USA.

[Dayan, 1992] Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362.

[El-Fakdi et al., 2005] El-Fakdi, A., Carreras, M., Palomeras, N., and Ridao, P. (2005). Autonomous underwater vehicle control using reinforcement learning policy search methods. In *IEEE Conference and Exhibition Oceans'05 Europe*.

[Fossen, 1995] Fossen, T. I. (1995). *Guidance and Control of Ocean Vehicles*. John Wiley and Sons.

[Gullapalli, 1992] Gullapalli, V. (1992). *Reinforcement Learning and its Application to Control*. PhD thesis, University of Massachusetts, Amherst, COINS Technical Report 92-10.

[Hansen, 1998] Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *8th Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, WI.

[Haykin, 1999] Haykin, S. (1999). *Neural Networks, a comprehensive foundation*. Prentice Hall, 2nd ed. edition.

[Hernandez and Mahadevan, 2000] Hernandez, N. and Mahadevan, S. (2000). Hierarchical memory-based reinforcement learning. In *Fifteenth International Conference on Neural Information Processing Systems*, Denver, USA.

[Jaakkola et al., 1994] Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201.

[Jaakkola et al., 1995] Jaakkola, T., Singh, S., and Jordan, M. (1995). *Reinforcement Learning algorithms for partially observable Markov decision problems*, volume 7, pages 345–352. Morgan Kaufman.

[Kimura and Kobayashi, 1998] Kimura, H. and Kobayashi, S. (1998). An analisis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions. In *ICML*, pages 278–286.

[Kimura et al., 1997] Kimura, H., Miyazaki, K., and Kobayashi, S. (1997). Reinforcement learning in pomdps with function approximation. In Fisher, D. H., editor, *Fourteenth International Conference on Machine Learning (ICML'97)*, pages 152–160.

[Kohl and Stone, 2004] Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Konda and Tsitsiklis, 2003] Konda, V. and Tsitsiklis, J. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42, number 4:1143–1166.

[L'Ecuyer, 1990] L'Ecuyer, P. (1990). A unified view of IPA, SF and LR gradient estimation techniques. *Management Science*, 36(11):1364–1383.

[Lin and Lin, 1996] Lin, C. J. and Lin, C. T. (1996). Reinforcement learning for an ART-based fuzzy adaptive learning control network. *IEEE Transactions on Neural Networks*, 7(3):709–731.

[Marbach, 1998] Marbach, P. (1998). Simulation-based methods for markov decision processes. PhD Thesis, Laboratory for Information and Decision Systems, MIT.

[Marbach and Tsitsiklis, 1998] Marbach, P. and Tsitsiklis, J. N. (1998). Simulation-based optimization of markov reward processes. Technical report LIDS-P-2411, Massachussets Institute of Technology.

[Marbach and Tsitsiklis, 2000] Marbach, P. and Tsitsiklis, J. N. (2000). Gradient-based optimization of Markov reward processes: Practical variants. Technical report, Center for Communications Systems Research, University of Cambridge.

[Matsubara et al., 2005] Matsubara, T., Morimoto, J., Nakanishi, J., Sato, M., and Doya, K. (2005). Learning sensory feedback to CPG with policy gradient for biped locomotion. In *Proceedings of the International Conference on Robotics and Automation ICRA*, Barcelona, Spain.

[Meuleau et al., 1999] Meuleau, N., Kim, K. E., Kaelbling, L. P., and Cassandra, A. R. (1999). Solving POMDPs by searching the space of finite policies. In Kaufmann, M., editor, *15th Conference on Uncertainty in Artificial Intelligence*, pages 127–136, Computer science Dep., Brown University.

[Meuleau et al., 2001] Meuleau, N., Peshkin, L., and Kim, K. (2001). Exploration in gradient based reinforcement learning. Technical report, Massachusetts Institute of Technology, AI Memo 2001-003.

[Monahan, 1982] Monahan, G. (1982). A survey of partially observable markov decision processes. *Management Science*, 28:1–16.

[Moore, 1991] Moore, A. (1991). Variable resolution dynamic programming: Efficiently learning action maps on multivariate real-value state-spaces. In *Proceedings of the Eighth International Conference on Machine Learning*.

[Murphy, 2000] Murphy, K. P. (2000). A survey of pomdp solution techniques.

[Ng and Jordan, 2000] Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *16th Conference of Uncertainty in Artificial Intelligence*.

[Ridao, 2001] Ridao, P. (2001). *A hybrid control architecture for an AUV*. PhD thesis, University of Girona.

[Ridao et al., 2004] Ridao, P., Tiano, A., El-Fakdi, A., Carreras, M., and Zirilli, A. (2004). On the identification of non-linear models of unmanned underwater vehicles. *Control Engineering Practice*, 12:1483–1499.

[Rosenstein and Barto, 2001] Rosenstein, M. and Barto, A. (2001). Robot weightlifting by direct policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Singh et al., 1994] Singh, S., Jaakkola, T., and Jordan, M. (1994). Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning*, New Jersey, USA.

[Singh and Sutton, 1996] Singh, S. and Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

[Smart and Kaelbling, 2000] Smart, W. D. and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *International Conference on Machine Learning*.

[Sondik, 1978] Sondik, E. J. (1978). The optimal control of partially observable Markov decision processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304.

[Sutton, 1988] Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44.

[Sutton and Barto, 1998] Sutton, R. and Barto, A. (1998). *Reinforcement Learning, an introduction*. MIT Press.

[Sutton et al., 2000] Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063.

[Tedrake et al., 2004] Tedrake, R., Zhang, T. W., and Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS'04*, Sendai, Japan.

[Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 3:185–202.

[Watkins, 1989] Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University.

[Watkins and Dayan, 1992] Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

[Williams, 1992] Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

[Witten, 1977] Witten, I. (1977). An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34:286–295.