GRADIENT-BASED REINFORCEMENT LEARNING TECHNIQUES FOR UNDERWATER ROBOTICS BEHAVIOR LEARNING

Andrés EL-FAKDI SENCIANES

ISBN: 978-84-694-2591-6
Dipòsit legal: GI-374-2011
http://hdl.handle.net/10803/7610

GRADIENT-BASED REINFORCEMENT LEARNING TECHNIQUES FOR UNDERWATER ROBOTICS BEHAVIOR LEARNING

ANDRES EL-FAKDI

PhD Thesis in Computer Engineering

University of Girona
Computer Engineering Department
Computer Vision and Robotics Group (VICOROB)

December 2010

SUPERVISOR:
Marc Carreras

LOCATION:
Girona

SUBMITTED:
December 2010

To my family and, specially, to Montse.

ABSTRACT

"Gradient-based reinforcement learning techniques for underwater robotics behavior learning."

A considerable interest has arisen around Autonomous Underwater Vehicle (AUV) applications. AUVs are very useful because of their size and their independence from human operators. However, comparison with humans in terms of efficiency and flexibility is often unequal. The development of autonomous control systems able to deal with such issues becomes a priority. The use of AUVs for covering large unknown dynamic underwater areas is a very complex problem, mainly when the AUV is required to react in real time to unpredictable changes in the environment.

This thesis is concerned with the field of AUVs and the problem of action-decision. The methodology chosen to solve this problem is Reinforcement Learning (RL). The work presented here focuses on the study and development of RL-based behaviors and their application to AUVs in real robotic tasks. The principal contribution of this thesis is the application of different RL techniques for autonomy improvement of an AUV, with the final purpose of demonstrating the feasibility of learning algorithms to help AUVs perform autonomous tasks. In RL, the robot tries to maximize a scalar evaluation obtained as a result of its interaction with the environment with the aim of finding an optimal policy to map the state of the environment to an action which in turn will maximize the accumulated future rewards. Thus, this dissertation is based on the principals of RL theory, surveying the two main classes of RL algorithms: Value Function (VF)-based methods and Policy Gradient (PG)-based techniques. A particular class of algorithms, Actor-Critic methods, born of the combination of PG algorithms with VF methods, is used for the final experimental results of this thesis: a real underwater task in which the underwater robot Ictineu AUV learns to perform an autonomous cable tracking task.

RESUM

"Tècniques d'aprenentatge per reforç basades en gradients per a l'aprenentatge de comportaments en robots submarins autònoms."

Darrerament, l'interès pel desenvolupament d'aplicacions amb robots submarins autònoms (AUV) ha crescut de forma considerable. Els AUVs són atractius gràcies al seu tamany i el fet que no necessiten un operador humà per pilotar-los. Tot i això, és impossible comparar, en termes d'eficiència i flexibilitat, l'habilitat d'un pilot humà amb les escasses capacitats operatives que ofereixen els AUVs actuals. L'utilització de AUVs per cobrir grans àrees implica resoldre problemes complexos, especialment si es desitja que el nostre robot reaccioni en temps real a canvis sobtats en les condicions de treball. Per aquestes raons, el desenvolupament de sistemes de control autònom amb l'objectiu de millorar aquestes capacitats ha esdevingut una prioritat.

Aquesta tesi tracta sobre el problema de la presa de decisions utilizant AUVs. El treball presentat es centra en l'estudi, disseny i aplicació de comportaments per a AUVs utilitzant tècniques d'aprenentatge per reforç (RL). La contribució principal d'aquesta tesi consisteix en l'aplicació de diverses tècniques de RL per tal de millorar l'autonomia dels robots submarins, amb l'objectiu final de demostrar la viabilitat d'aquests algoritmes per aprendre tasques submarines autònomes en temps real. En RL, el robot intenta maximitzar un reforç escalar obtingut com a conseqüència de la seva interacció amb l'entorn. L'objectiu és trobar una política òptima que relaciona tots els estats possibles amb les accions a executar per a cada estat que maximitzen la suma de reforços totals. Així, aquesta tesi investiga principalment dues tipologies d'algoritmes basats en RL: mètodes basats en funcions de valor (VF) i mètodes basats en el gradient (PG). Els resultats experimentals finals mostren el robot submarí Ictineu en una tasca autònoma real de seguiment de cables submarins. Per portar-la a terme, s'ha dissenyat un algoritme anomenat mètode d'Actor i Crític (AC), fruit de la fusió de mètodes VF amb tècniques de PG.

RESUMEN

"Técnicas de aprendizaje por refuerzo basadas en gradientes para el aprendizaje de comportamientos en robots submarinos autónomos."

Últimamente, el interés por el desarrollo de aplicaciones con robots submarinos autónomos (AUV) ha crecido de forma considerable. Los AUVs son atractivos por su tamaño y porque no necesitan un operador humano para su pilotaje. Aún y así, es imposible comparar, en términos de eficiencia y flexibilidad, la habilidad del pilotaje humano con las escasas capacidades operativas que ofrecen los AUVs actuales. El uso de AUVs para cubrir grandes áreas implica resolver problemas complejos, especialmente si se desea que el robot reaccione en tiempo real a cambios bruscos que pudieran producirse en las condiciones de trabajo. Por estas razones, el desarrollo de sistemas de control autónomo para mejorar estas capacidades se ha convertido en una prioridad.

Esta tesis trata sobre el problema de la toma de decisiones utilizando AUVs. El trabajo presentado se centra en el estudio, diseño y aplicación de comportamientos para AUVs utilizando técnicas de aprendizaje por refuerzo (RL). La contribución principal de la tesis consiste en la aplicación de varias técnicas que permiten mejorar la autonomía de los robots submarinos, con el objetivo final de demostrar la viabilidad de estos algoritmos para aprender tareas submarinas de forma autónoma en tiempo real. En RL, el robot intenta maximizar un refuerzo escalar obtenido como consecuencia de su interacción con el entorno. El objetivo es encontrar una política óptima que relaciona todos los estados posibles con las acciones a ejecutar para cada estado que maximizan la suma de refuerzos totales. Así, esta tesis investiga principalmente dos tipologias de algoritmos basados en RL: métodos basados en funciones de valor (VF) y métodos basados en el gradiente (PG). Los resultados experimentales finales muestran al robot submarino Ictineu en una tarea autónoma real de seguimiento de cables submarinos. Para llevarla a cabo, se ha diseñado un algoritmo llamado método del Actor y el Crítico (AC), fruto de la fusión de métodos VF con técnicas de PG.

We have seen that computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty.

— Donald E. Knuth [70]

ACKNOWLEDGMENTS

I would like to express my gratitude to the people who have supported me during the work on this PhD thesis. First and foremost to my advisor Marc Carreras, for introducing me into the magic of Reinforcement Learning, and for believing in me and encouraging me during this research with his unshakable optimism, inspiration and friendship. I give my special gratitude to my parents, Hassan and Maria Luisa, and my best friend and brother Raul. They deserve special gratitude for their comprehension and support of my decision to start a research career.

I also thank the members of the Underwater Robotics Lab for their camaraderie and willingness to help whenever it was necessary. I want to thank my colleagues David, Emili, Narcis, Enric, Tali, Lluis, Miki, Pere, Aggelos, Arnau and Sebas for always being there. I would also like to extend my gratitude to the rest of the members of the Computer Vision and Robotics Group and of the department of Computer Engineering, in particular to Xevi, Jordi, Quim, Sik, Quintana, Rafa, Tomas and Toni. I should not forget Javier and the rest of the colleagues from the University of the Balearic Islands for their help with some aspects of this thesis. I am also grateful to the people in the Robot Locomotion Group at the Computer Science and Artificial Intelligence Laboratory (CSAIL) of the Massachusetts Institute of Technology (MIT) for their warm reception and hospitality during my stay. I especially want to thank Russ Tedrake for his comments and advice on my work, and also to Alec, Rick, Stephen, Khash, Vanessa and Katie, for making me feel like a member of the group.

I also wish to extend my gratitude to all my friends, but in particular to Manolo, Edu, Tomas, Jaume, Felipe, Pum and Jordi. And finally, I want to express my gratitude to Montse, for her patience and comprehension at every moment.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DSTL	Defence Science and Technology Lab
AUV	Autonomous Underwater Vehicle
ROV	Remote Operated Vehicle
UUV	Unmanned Underwater Vehicle
MRU	Motion Reference Unit
SAUCE	Student Autonomous Underwater Challenge Europe
VICOROB	Computer Vision and Robotics Group
DoF	Degree of Freedom
DVL	Doppler Velocity Log
MSIS	Mechanically Scanned Imaging Sonar
CCD	Charge Coupled Device
LED	Light-Emitting Diode
ROI	Region of Interest

RL	Reinforcement Learning
CIRS	Centre d'Investigació en Robòtica Submarina
FMDP	Finite Markov Decision Process
RLP	Reinforcement Learning Problem
MDP	Markov Decision Process
TD	Temporal Difference
QL	Q-Learning
DP	Dynamic Programming
HMM	Hidden Markov Model
POFMDP	Partially Observable Finite Markov Decision Process
ANN	Artificial Neural Network
VAPS	Single Value and Policy Search
CPG	Central Pattern Generator
LS	Least Squares
MC	Monte Carlo
PG	Policy Gradient
VF	Value Function
NAC	Natural Actor-Critic
PD	Proportional Derivative
SPA	Sense, Plan and Act
MIT	Massachusetts Institute of Technology
AURA	Autonomous Robot Architecture
AI	Artificial Intelligence
CMAC	Cerebellar Model Articulation Controller
SVM	Support Vector Machine
SVR	Support Vector Regression

NTP	Network Time Protocol
MIMO	Multiple-Input-Multiple-Output
PID	Proportional-Integral-Derivative Controller
MCYT	Ministerio de Ciencia y Tecnologia
SONQL	Semi-Online Neural Q-Learning
LSTD	Least Squares Temporal Difference
AC	Actor-Critic
CSAIL	Computer Science and Artificial Intelligence Laboratory
GPOMDP	Gradient Partially Observable Markov Decision Process
EM	Expectation Maximization
PoWER	Policy Learning by Weighting Exploration with the Returns
DMP	Dynamic Movement Primitives

1 INTRODUCTION

Our society is constantly evolving. A lot of things have changed since 1917 when the writer Karel Čapek first used the word *robot* to refer to a machine with a humanoid aspect. Even the well-known ideas of Asimov about robots and his three famous laws of robotics seem to be far behind us today. 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 of the "marriage" between informatics science and tool-machines so now the tool is able to process and manage information rationally and automatically without human help, replacing the worker, without getting tired, without strikes and being 100% operational. However, reality suggests that we are still very far from such a technology. Although being trivial for humans, the capability to perceive the environment (*sensing*) and making decisions (*acting*) is a very difficult task for a computer. Thus, the field of *Artificial Intelligence* (AI) is needed for autonomous robots to solve such problems.

This thesis is concerned with the field of autonomous robots and the problem of action-decision. The methodology chosen is Reinforcement Learning (RL). In RL, an agent tries to maximize a scalar evaluation obtained as a result of its interaction with the environment with the aim of finding an optimal policy to map the state of the environment to an action which in turn will maximize accumulated future rewards. Thus, this dissertation is based on the principals of RL. It surveys two main classes of RL algorithms: Value Function (VF)-based methods and Policy Gradient (PG)-based techniques. A particular class of algorithms, Actor-Critic (AC) methods, born of the combination of PG algorithms with VF methods, is used for obtaining the final experimental results of this thesis: a real underwater task where the underwater robot Ictineu AUV learns to perform an autonomous cable tracking task.

This introduction continues with the main aspects which have conditioned this thesis. First, some background on the motivations and applicability will be provided, then a description of the objectives of this thesis and the outline of this document are given.

1.1 MOTIVATION

The research presented in this thesis was carried out in the Underwater Robotics Laboratory of the Computer Vision and Robotics Group (VICOROB) of the University of Girona. This group has been doing research in underwater robotics since 1992, supported by several National and European programs. A main contribution over the past few years has been the development of three Unmanned Underwater Vehicles (UUVs). The first prototype, called GARBI, was developed in collaboration with the Polytechnic University of Catalonia. This underwater robot was initially conceived as a Remote Operated Vehicle (ROV), but after successive modifications over the years, the robot evolved into its final configuration as an Autonomous Underwater Vehicle (AUV) in 2005. The second prototype, URIS (1999), was fully developed at the University of Girona and was designed as an small AUV, smaller and cheaper than its older brother. This underwater robot was constructed with the objective of having a quick experimental benchmark for testing sensors and software under laboratory conditions. The most recent robot is the Ictineu (2006), an AUV which brings together the broad sensorial capabilities of the GARBI and the small form factor of the URIS, which make this robot a perfect research platform for testing in both laboratory and real application environments.

The research efforts in the Underwater Robotics Laboratory focused on the technical development of autonomous underwater robots. The design of an autonomous robot requires a solution for the action-decision problem. The control system is the part of the robot in charge of making decisions. An autonomous robot is designed to accomplish a particular mission, and the control architecture must achieve this mission by generating the proper actions. The design of the autonomous robot URIS and previously, the adaptation of GARBI from a ROV to an AUV, led to the development of a new control architecture, the O^2CA^2 control architecture [34]. The main feature of this architecture was the breaking down of the whole robot control system into a set of objects. The parallel execution of these objects allowed real-time execution. In addition, some of these objects represented primitive behaviors of the robot. Also, as the complexity of the autonomous missions increased, advances were made towards the development of mission control systems [36]. Additional work on the identification of dynamic models of underwater robots [129] made the development of various research tools such as the Neptune simulator possible [128]. With respect to the application domain, preliminary work was carried out on the use of ROV technology for inspection of hydroelectric dams using image mosaics [21]. Later, the topic was readdressed using AUVs in the context of a research project supported by the Spanish commission Ministerio de Ciencia y Tecnologia (MCYT), made in collaboration with the University of the Balearic Islands and the Polytechnic University of Catalonia. The objective of the project was to improve AUVs performance for their use in industrial applications such as the inspection of hydroelectric dams, harbors and underwater cables and pipelines.

According to the achievements in those lines of research and application domains, the abilities of the robots have increased as well as their mission requirements. Following the work done with the O^2CA^2 control architecture, the study and development of better behaviors which constitute the action-decision methodology were the next objective. The investigation of some implementation aspects which influence the overall performance of the robot was carried out. In this way, the use of learning algorithms to improve the efficiency of the behaviors was explored. A learning framework, called Reinforcement Learning (RL), was studied. The research carried out on the RL principal succeeded with the development of the Semi-Online Neural Q-Learning (SONQL) method [34], an RL-based algorithm embedded inside a behavior-based control architecture for an AUV able to perform simple tasks such as learning how to follow an artificial target and exhibit real-time learning capabilities.

The work done during the elaboration of this thesis continues along the line started with the RL-based behaviors principal. This dissertation goes a step further and surveys novel RL-based methodologies which may offer important advantages in real robotic applications, paying special attention to the sort of algorithm able to perform on-line learning.

1.2 GOAL OF THE THESIS

As discussed in the motivation section, the goal of this thesis is the study and development of RL-based behaviors and their application to AUVs in real robotic tasks. Over the past few years, considerable interest has arisen around AUV applications. Industries around the world call for technology applied to several underwater scenarios, such as environmental monitoring, oceanographic research or maintenance/monitoring of underwater structures. AUVs are attractive for utilization in these areas because of their size and their non-reliance on human operators. However, comparison with humans in terms of efficiency and flexibility is often unequal. The development of autonomous control systems able to deal with such issues becomes a priority. The use of AUVs for covering large unknown dynamic underwater areas is a very complex problem, mainly when the AUV is required to react in real time to unpredictable changes in the environment. This thesis applies different RL techniques for autonomy improvement of an AUV. The purpose is to demonstrate the feasibility of learning algorithms to help AUVs perform autonomous tasks. In particular, this thesis concentrates on one of the fastest maturing, and probably most immediately significant, commercial application: *Cable and Pipeline Tracking*. In this way, this thesis presents Policy Gradient (PG) techniques, a particular class of RL method, as an alternative to classic Value Function (VF) algorithms. Most of the methods proposed in the RL community to date are not applicable to robotics as they do not scale beyond robots with more than one to three degrees of freedom. PG methods are a notable exception to this statement. The advantages of PG methods for robotics are numerous. Among the most important are that they have good generalization capabilities which allow them to deal with big state-spaces, that their policy representations can be chosen so that it is meaningful to the task and can incorporate previous domain knowledge and that often fewer parameters are needed in the learning process than in VF-based approaches. Also, there are various algorithms for PG estimation in the literature which have a rather strong theoretical underpinning. In addition, PG methods can be used model-free and therefore can also be applied to robot problems without an in-depth understanding of the problem or mechanics of the robot [115]. The final experiments demonstrate that the best results are obtained when combining both VF and PG techniques into a particular class of algorithms called Actor-Critic (AC) methods.

Low convergence speed is an ever present problem in real robotics. To solve this issue, this thesis combines RL methods with computer simulation techniques to speeding up the learning process. Parallel to the theoretical analysis, successful applications of these methods are presented together with experimental results of the final proposed algorithms.

1.2.1 *Objectives*

After reviewing the research motivations and describing the problem, the goal of this thesis is stated here as:

"The study and utilization of policy gradient-based reinforcement learning algorithms for the development of RL-based behaviors and its application to autonomous underwater robotics tasks."

The objectives of this dissertation have been oriented to continue along the research path of RL-based behaviors opened in this lab. Also, this thesis wants to give an insight into the need of the underwater technology community for solutions to increasing underwater vehicles' autonomy. This thesis reviews the main aspects of RL and analyzes the main features of VF-based algorithms and PG-based methods. It discusses them as different but complementary approaches to solving the Reinforcement Learning Problem (RLP) in real robotic tasks. Also, this dissertation extends the field of robot learning, which is one of the most active areas in robotics. The goal of this thesis can be divided into the following more specific objectives:

THE REINFORCEMENT LEARNING PROBLEM. A general overview of RL with the main objective of understanding its related theory, main features and field of applications in robotics. Also, particular classes of RL algorithms have been studied, some based on VF and others on PG techniques. These methodologies offer different solutions to solve the RLP, offering advantages and drawbacks. All of them have been analyzed, either from a theoretical or a practical point of view, giving examples of successful applications to real robotics. The objective of the first chapters is to build a theoretical basis to propose the most suitable class of RL algorithm for solving the RLP described in Chapter 5.

MODEL IDENTIFICATION. Although the theoretical side of the selected RL algorithms give them good convergence properties, once the learning is moved to a real environment and asked for on-line capabilities the whole process becomes slow. Due to the objective of achieving faster convergence, this dissertation introduces methods to speeding up the learning process. Among the different methods surveyed, simulated learning techniques were chosen for the final experiments of this thesis. In order to learn an initial policy with the aid of a simulator, an accurate model of the robot had to be approximated. Thus, the second objective of this dissertation is the study and utilization of a specially designed Least Squares (LS) model identification algorithm to obtain a reliable model of the Ictineu AUV, the robot used in the final tests and described in Chapter 5.

EXPERIMENTAL SETUP AND RESULTS. Evaluation of the various algorithms surveyed, either in simulation or in real experiments. The objective is to compare methods and extract the best features of each. For this purpose, common benchmarks were used for the simulated tests while the robot Ictineu AUV was being used for the real experiments. The final part of this thesis describes the experiments carried out with the various algorithms. They lead to the final learning methodology that solves the final experimental task: a real autonomous underwater cable tracking robot.

1.3 OUTLINE OF THE THESIS

The contents of this thesis can be divided into three main parts. The first part overviews the field of RL and details the most representative methods used to solve RL problems, either from the theoretical or the practical point of view (Chapter 2), paying special attention to their application to real robotic tasks and the issues related to them: generalization problems and convergence speeds. The most suitable gradient techniques were implemented inside an underwater robot control architecture (Chapter 3). The second part of this thesis prepares the whole setup for the final experiments. First, a mathematical model of the robot used for the tests, Ictineu AUV, is presented by means of a specially designed least squares identification method. The proposed LS algorithm itself represents another contribution of this thesis (Chapter 4). This model allowed us to compute policies by simulation and transfer them to the real robot to continue the learning on-line, greatly increasing convergence speeds. The underwater robot Ictineu AUV and the Computer Vision and Robotics Group facility CIRS where all the tests were performed is also described (Chapter 5). Finally, the third and last part of this document encloses the experimental results obtained (Chapter 6) and summarizes the contributions and further work (Chapter 7). A brief description of each chapter is presented next.

CHAPTER 2 *State Of The Art.* This chapter reviews the field of RL. Once the main aspects are described, the best known solutions to solving the RLP are presented, with special interest on those able to deal with real robotics. Among these are two kinds of algorithms, VF and PG algorithms, which are analyzed and compared, detailing the main advantages and drawbacks of each one. Finally, the combination of both methods, AC algorithms, demonstrate the best performance for solving real robotics tasks. Practical applications of the theoretical algorithms are discussed at the end of this chapter.

CHAPTER 3 *Policy Gradient Methods for Robot Control.* This chapter analyzes issues related with the application of the RL techniques described in the previous chapter to real robotic tasks. A brief discussion of the evolution of behavior-based control architectures for real robots is given at the beginning of the chapter. The generalization problem, which highly affects real robotics, is described next. The most common approaches to confront this problem and its application to robotics tasks are overviewed. Two algorithms have been chosen to obtain the final results of this thesis: Baxter and Bartlett's PG algorithm and Peters' NAC. This chapter describes the adaptations needed for the experiments of this thesis. Also, at the end of the chapter, various techniques to speeding up the learning process are briefly discussed.

CHAPTER 4 *Model Identification for Underwater Vehicles.* This chapter describes the identification procedure used to model the underwater robot Ictineu AUV. For this purpose, the dynamics equations of motion of an underwater vehicle are accurately described. After that, several common assumptions are given. These allow us to adapt the general equation to our particular robot and, therefore, simplify the problem. Finally, an LS identification method is proposed and applied to identify the vehicle's parameters. Some results regarding the performance of the identified model are given at the end of this chapter.

CHAPTER 5 *Experimental Setup.* The purpose of this chapter is to report the main characteristics of the different elements used to build the experimental setup. First, the most notable features of the Ictineu AUV are given. These include the design principles, the actuators and the on-board sensors. An insight into the control architecture of the robot is given next. Also, the problem of underwater pipeline and cable tracking is detailed. Various common approaches to performing this task are discussed and, among them, the one selected to carry out the experiments of this thesis, a vision based system developed by scientists at the University of Balearic Islands, is described. The algorithm used to estimate the position and orientation of the cable within the image plane is presented. Finally, the Computer Vision and Robotics Group facility CIRS where all the tests were performed is also described at the end of this chapter.

CHAPTER 6 *Results.* This chapter presents the experimental results of this thesis. The results are organized in a series of sections which show the experiments carried out with the various algorithms proposed. Both well known RL benchmarks and specific underwater tasks are described. The final experiments proposal presented in this thesis aims to reduce convergence times of an actor-critic algorithm combining it with one of the speeding up techniques discussed in Chapter 3. The idea is to build an initial policy quickly using a computer simulator which contains an approximate model of the environment. In a second step, the policy is transferred to the Ictineu AUV robot to continue the learning on-line.

CHAPTER 7 *Conclusion.* This chapter concludes the thesis by summarizing all the work done, pointing out contributions and future work. It also comments on the research evolution and the publications accomplished during this doctoral study.

2 STATE OF THE ART

A commonly used methodology in robot learning is Reinforcement Learning (RL) [152]. 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 reviews the field of RL. Once the main aspects are described, representative solutions to solving the RLP are presented, with special interest on those able to deal with real-world robots. Two kind of algorithms, Value Function (VF) algorithms and Policy Gradient (PG) algorithms, are analyzed and compared. The main advantages and drawbacks of each approach are described. Finally, a particular combination of both methods which exploits the advantages of VF and PG, called Actor-Critic (AC) algorithms, demonstrate the best performance for solving real robotic tasks. Practical applications of all theoretical algorithms are discussed at the end of this chapter, with conclusions and ideas extracted from them in order to design learning methodologies for underwater robots that represent the main contribution of this dissertation.

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. The robot 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 and 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 agent. If we describe the interaction process step by step, the agent 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 agent-environment relationship can be seen in Figure 1. Observing the diagram, the interaction sequence can be described: for each iteration step *t*, the agent observes the state s_t and receives a reward r_t. According to these inputs and the RL policy followed at that moment, the agent 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. The most important features of the agent and environment are listed as follows:

AGENT :

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

ENVIRONMENT :

- Everything outside the agent.
- Reacts to actions by transferring to a new state.
- Contains the reward function which generates the rewards.

As stated before, the agent 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 agent always selects what is thought to be the best action at every current state. However, exploration is sometimes required to investigate the effectiveness of actions that have not been tried

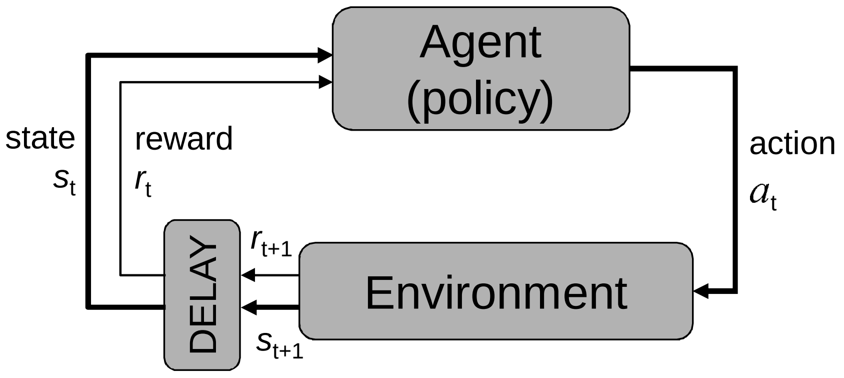

Figure 1: Diagram of the interaction Agent vs. Environment.

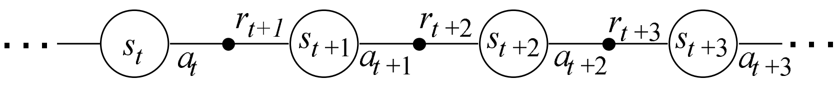

Figure 2: Schematic sequence of states, actions and rewards.

at the current state. Once the learning period is over, the agent does not select the action that maximizes the immediate reward but the one that maximizes the sum of future rewards.

If we take a deeper look, beyond the environment and the agent, four main sub-elements 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 function 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*, *deterministic* or *random*. Stochastic policies are represented by $a \sim \pi(a|s)$, where the probability of choosing action *a* from state *s* is contained. Deterministic policies $a = \pi(s)$, also know as *greedy* policies, contain the best mapping known by the agent, that is, the actions supposed to best solve the RLP. A *greedy action* is an action taken from a greedy policy. Finally, a policy can be *random*, in which case the action will be chosen randomly. An $\epsilon - greedy$ policy selects random actions with a probability ϵ and greedy actions with a probability (1 - ϵ). The policy that gets maximum accumulated rewards is called optimal policy and is represented as π^*.

REWARD FUNCTION. This function 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 agent is to maximize the total reward it receives in the long run. The reinforcement function gives an immediate evaluation to the agent. High immediate rewards are not the objective of an RL agent but only 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 agent 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 agent reaches the optimal value functions, we can obtain the optimal policy from it.

DYNAMICS FUNCTION. Also known as the model of the environment. This 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 RL algorithms use these functions to solve the RLP. As will be shown in the next sections, the learning process can be performed either over the value function, the state $V^{\pi}(s)$ or the action value function $Q^{\pi}(s,a)$, or over the policy π itself. These algorithms propose a learning update rule to modify the value or the policy function parameters in order to maximize the received rewards. If the algorithm converges after some iterations, an optimal policy π^* can be extracted and the RLP is solved.

2.2 FINITE MARKOV DECISION PROCESSES IN RL

In RL, the agent makes its decision according to the environment's information. This information is presented as the environment's *state* and *reward*. In this thesis, by *the state* we mean whatever information is available to the agent. This section discusses what is required of the state signal and what kind of information we should and should not expect it to provide. 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 that has 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. This property is sometimes referred to as *the independence of path* property because all that matters is in the current state signal; its meaning is independent of the path taken that led up to it. The response at time $t+1$ to an action taken at time t for a non-Markovian system is shown in Equation 2.1. For this case, the response depends on everything that has happened earlier. Hence, the conditional probability of achieving the next state s_{t+1} and obtaining the reward r_{t+1} when taking action a_t and knowing previous states and actions is defined as

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

for all s', r and all possible values of the past events represented by: $s_t, a_t, r_t, ..., r_1, s_0, a_0$. On the other hand,if an environment accomplishes 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

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

for all s', r, s_t and a_t. A state signal is considered to have the Markov property if, and only if, Equation 2.1 is equal to Equation 2.2 for all states and actions. If an environment accomplishes the Markov property, its one step dynamics allows us to predict the next state and the next reward given the current state and action. The Markov property is important in RL because the decisions taken are assumed to be functions only of the current state. Hence, although the state at each time step may not fully satisfy the Markov property, it will be approximated as a Markov state.

An RLP which accomplishes the Markov property is called a *Markov Decision Process (MDP)*. Moreover, if the state and action spaces are finite, the environment is considered a *Finite Markov Decision Process (FMDP)*. For a particular FMDP, 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 as

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

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 an FMDP. Most RL techniques are based on FMDPs causing finite state and action spaces. The RLP analyzed in this thesis assumes that the environment is an FMDP. Considering only Markov environments 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 [142]. Following this path, the most direct extension of FMDPs that hides a part of the state to the agent is known as Hidden Markov Models (HMMs).

The underlying environment continues to accomplish the Markov property but the information extracted does not seem Markovian to the agent. The analogy of HMMs for control problems are the Partially Observable Finite Markov Decision Processs (POFMDPs) [98]. The algorithms studied in this dissertation are based on POFMDPs because, in practice, it is impossible to get a complete state for any realistic robot system.

2.3 VALUE FUNCTIONS

Most RL algorithms are based on estimating a *value function* VF. The VF, located in the agent, is related to the MDP dynamics. For a particular learned policy π, the state-value function V^π or the action-value function Q^π can be expressed in terms of $P^a_{ss'}$ and $R^a_{ss'}$ and, as will be shown, the optimal VF (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 stated in Section 2.1, 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

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

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 robotic task, the agent may be continually activated. In this case, the tasks are called *continuous 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

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

By setting the discount factor between $0 \leqslant \gamma \leqslant 1$, 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 VFs, V^π and Q^π, can be expressed in terms of the expected future reward R_t. In the case of the state VF, 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 VF, denoted $Q^\pi(s,a)$, the value of taking action a in state s under policy π 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 as

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

and

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

These equations define the VFs 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 VFs indicate, the expected sum of future rewards for each state or state/action pair, an optimal VF will contain the maximum values. Therefore, from all the policies π, the one having a VF (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 VF can be found (V^* or Q^*), i. e.,

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

and

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

In order to find these optimal VFs, the Bellman equation [24] can be employed. 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 VFs are given as

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

and

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

The Bellman optimality equations offer a solution to the RLP by finding the optimal VFs V^* and Q^*. Although this solution only works for very small systems and not at all for most continuous systems 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, N being the number of states. This nonlinear system can be solved resulting in the V^* function. Similarly, the Q^* function can be found.

Once V^* or Q^* 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 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, i. e.,

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

This section has formulated the RLP using a FMDP model. A definition of optimal VFs and optimal policies has been given and the solution to the RLP when the dynamics of the environment is known has been detailed but this rarely happens in real robotics. Even if the model of a real environment is accurate enough, the computational power needed to deal with huge continuous space states and achieve good solutions in real tasks is too high and often unaffordable. Hence, real RLPs lead us to settle for approximations. The following section will describe different methodologies for finding good solutions to the RLP in real robotic scenarios.

2.4 SOLUTION METHODS FOR THE RLP

There are three fundamental methodologies to solve the RLP formulated as an FMDP. This section summarizes the main features of: Dynamic Programming (DP), Monte Carlo (MC) methods and TD learning. Each class has its strengths and weaknesses, which will be discussed in the following.

DYNAMIC PROGRAMMING. This methodology is able to compute the optimal policy π^* given a perfect model of the environment dynamics as an FMDP ($P^a_{ss'}$ and $R^a_{ss'}$). DP algorithms act iteratively to solve the Bellman optimality equations. DP algorithms are able to learn online, that is, while the agent is interacting with the environment in a step-by-step sense. At each iteration, they update the value of the current state based on the values of all possible successor states. The knowledge of the dynamics is used to predict the probability of the next state to occur. As the learning is performed online, the agent is able to learn the optimal policy while it is interacting with the environment. The general update equation for the state VF is given by

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]. \quad (2.14)$$

In this equation, the state VF at iteration $k+1$ is updated with the state VF at iteration k. Most DP algorithms compute the VFs for a given policy. Once this VF is obtained, they improve the policy based on it. Iteratively, DP algorithms are able to find the optimal policy which solves the RLP. The algorithm replaces the old value of s with a new value obtained from the successor states of s and the immediate reward expected from all one-step transitions. Hence, at each iteration step, a *full backup* of all state values is performed. The computational expenses of this process are prohibitive and, together with the perfect model requirements, represent the main drawback of DP in real robotic problems.

MONTE CARLO METHODS. MC do not need a model of the environment. Instead, they use the experience with the environment, sequences of states, actions and rewards, to learn the VFs. MC algorithms interact with the environment following a particular policy π. When the episode finishes, they update the value of all the visited states based on the rewards received. By repeating the learning over several episodes, the VF for a particular policy can be found. MC methods are incremental in an episode-by-episode sense, but not in a step-by-step sense, restricting them from some particular online applications. Equation 2.15 shows the general update rule to estimate the state-value function. At the end of an episode, the current prediction of the state-value $V_k^\pi(s)$ is modified according to the sum of rewards R_t received along the whole episode, not the immediate reward expected as DP algorithms do. There is also a learning rate α which averages the values obtained in different episodes. Hence, we have

$$V_{k+1}^\pi(s_t) = V_k^\pi(s_t) + \alpha[R_t - V_k^\pi(s_t)]. \tag{2.15}$$

After the evaluation of a policy, MC methods improve this policy based on the VF learnt. By repeating the evaluation and improving phases, an optimal policy can be achieved. The main advantage of MC methods over DP algorithms is that they do not need a model of the environment. As a drawback, MC methods are not suitable for most continuing tasks, as they can not update the VF until a terminal state is found.

TEMPORAL DIFFERENCE LEARNING. The advantages of the previous methods can be found in TD algorithms. Like DP, TD learning is able to update value estimates at every iteration step based in part on other estimates, without waiting for the final outcome (they *bootstrap*).

Also, similar to MC methods, they do not need the dynamics of the environment but the experience with the environment itself. The general update rule for the state-value function can be seen in Equation 2.16. Similar to MC methods, the updating of the state value $V^{\pi}_{k+1}(s_t)$ is accomplished by comparing its current value with the discounted sum of future rewards. However, in TD algorithms this sum is estimated with the immediate reward r_{t+1} plus the discounted value of the next state. Also, the update rule does not require the dynamics transition probabilities which allow TD algorithms to learn in an unknown environment.

$$V^{\pi}_{k+1}(s_t) = V^{\pi}_{k}(s_t) + \alpha[r_{t+1} + \gamma V^{\pi}_{k}(s_{t+1}) - V^{\pi}_{k}(s_t)]. \tag{2.16}$$

Although TD methods are mathematically complex to analyze, they do not require the dynamics to be known and are fully incremental, allowing them to be performed on-line. These reasons make TD methods the most suitable algorithms for solving the RLP for most robotic tasks, like the ones that will be solved in this dissertation. Therefore, the algorithms studied in this thesis will be based on TD methods.

When dealing with real-world robots, the dominant approach has been to apply different TD-based algorithms to learn a particular VF and then extract the optimal policy from it. These algorithms are called *Value Function algorithms*. The next section describes the mathematical foundation of this methodology, always focusing on those algorithms able to deal with real robotic tasks.

2.5 VALUE FUNCTION ALGORITHMS

Value Function (VF) methodologies find the optimal policy by first searching for the optimal VF and then deducing the optimal policy from the optimal VF. Figure 3 represents a block diagram which is usually followed by RL algorithms based on the VF approach. At every iteration step, the algorithm proposes a learning update rule to modify the current VF $V^{\pi}(s)$ or $Q^{\pi}(s, a)$ and then extracts a policy π to be followed by the agent. If the RL algorithm converges after some iterations, the VF changes to the optimal VF, $V^*(s)$ or $Q^*(s, a)$, from which the optimal policy π^* can be extracted. The solution of the RLP is accomplished by following the state-action mapping contained in the optimal policy π^*.

VF-based methodologies under Markov environments have been a platform for various RL algorithms. The next sections introduce the most representative algorithms using $V^\pi(s)$ and $Q^\pi(s, a)$ respectively.

2.5.1 *Temporal Difference (TD)(λ)*

The TD(λ) [152] is a common example of a state VF approach. The λ term refers to the use of a new concept, the *eligibility trace*. Eligibility traces are a basic tool used by several reinforcement learning algorithms that have been widely used to handle delayed rewards. They can be defined as a bridge between TD methods and MC techniques. As described in Section 2.4, TD methods do not need a model of the environment and update VF using immediate rewards r_{t+1} and value estimates. Similarly, MC methods do not need a model of the environment but they must wait till the episode ends and the total reward R_t is perceived to update the state values. Eligibility trace works as a *decay factor* which introduces a memory into our reward assignment as a temporary record of the occurrence of a particular event stored in a new variable associated to each state called $e_t(s)$ and defined as

$$e_t(s) = \begin{cases} \gamma\lambda\, e_{t-1}(s) & \text{if } s \neq s_t, \\ \gamma\lambda\, e_{t-1}(s) + 1 & \text{if } s = s_t. \end{cases} \tag{2.17}$$

Here, γ is a discount factor and λ is the decay factor defined at the beginning of this section. The range of the decay factor λ is $0 \leqslant \lambda \leqslant 1$. If $\lambda = 0$ all the credit given to previous states is zero except for the last one corresponding to s_t and the TD(λ) reduces to a simple TD method, called TD(0). If $\lambda = 1$ previous visits have full credit. Their value only decreases due to the discount factor γ and our algorithm behaves as an MC method. The TD(λ) algorithm update rule is detailed in Algorithm 1.

At any time, eligibility traces record which states have recently been visited, where "recently" is defined in terms of $\gamma\lambda$. Traces indicate the degree to which each state is *eligible*

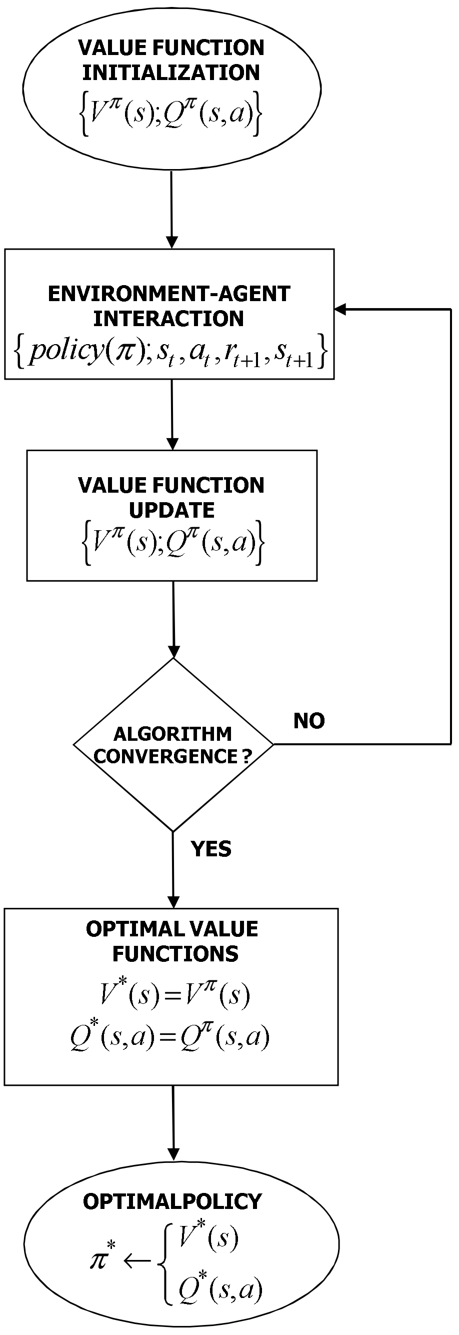
VALUE FUNCTION INITIALIZATION
$\{V^{\pi}(s);Q^{\pi}(s,a)\}$
ENVIRONMENT-AGENT INTERACTION
$\{policy(\pi);s_t,a_t,r_{t+1},s_{t+1}\}$
VALUE FUNCTION UPDATE
$\{V^{\pi}(s);Q^{\pi}(s,a)\}$
ALGORITHM CONVERGENCE ?
NO
YES
OPTIMAL VALUE FUNCTIONS
$V^*(s)=V^{\pi}(s)$
$Q^*(s,a)=Q^{\pi}(s,a)$
OPTIMALPOLICY
$\pi^* \leftarrow \begin{cases} V^*(s) \\ Q^*(s,a) \end{cases}$

Figure 3: Typical phase diagram of a VF-based RL algorithm, where the VF is updated according to the algorithm. Once the optimal VF is found, the optimal state-action policy π^* is extracted.

Algorithm 1: TD(λ) algorithm.

```
Initialize $V(s)$ arbitrarily and $e(s) = 0$ for all $s \in S$
Repeat until $V(s)$ is optimal:
  a) $s_t \leftarrow$ the current state
  b) $a_t \leftarrow$ action given by $\pi$ for $s_t$
  c) Take action $a_t$, observe reward $r_{t+1}$ and new state $s_{t+1}$
  d) $\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
  e) $e(s_t) = e(s_t) + 1$
  For all s:
    f) $V(s_t) = V(s_t) + \alpha\delta e(s_t)$
    g) $e(s_t) = \gamma\lambda e(s_t)$
  h) $s_t = s_{t+1}$
```

for undergoing learning changes. Those reinforcing events are the one-step TD error for the state-value prediction indicated by δ. If we take a look at Equation 2.16 in Section 2.4, the update rule of a simple TD(0) algorithm renews only the value of the current state at each iteration, whereas when introducing an eligibility trace, all the values of recently visited states are updated. The propagation of delayed rewards through recently visited states greatly increases the convergence of the algorithm, achieving an optimal policy in fewer iterations. Another on-policy TD control method, the *Sarsa* algorithm [152], uses essentially the same TD(0) method described above, but learns the action-value function rather than the state-value function. Theorems assuring the convergence of state values under TD(0) also apply to this algorithm for action values.

2.5.2 *Q-Learning (QL)*

The QL algorithm [163] is another TD algorithm. As distinguished from the TD(λ) algorithm, QL uses the action-value function $Q^\pi(s, a)$ to find an optimal policy π^*. The $Q^\pi(s, a)$ function has an advantage over the state-value function $V^\pi(s)$. Once the optimal function $Q^*(s, a)$ has been learnt, the extraction of an optimal policy π^* can be directly performed without the requirement of the environment dynamics. The optimal policy will be composed of a map relating each state s with any action a which maximizes the $Q^*(s, a)$ function.

Another important feature of QL is the *off-policy* learning capability. That is, in order to learn the optimal function Q^*, any policy can be followed. The only condition is that all the state/action pairs must be regularly visited and updated. This feature, together with the simplicity of the algorithm, makes QL very attractive for a lot of applications. In real systems, as in robotics, there are many situations in which not all the actions can be executed. For example, to maintain the safety of a robot, an action cannot be applied if there is any risk of colliding with an obstacle. Therefore, if a supervisor module modifies the actions proposed by the QL algorithm, the algorithm will still converge to the optimal policy.

The QL algorithm is described in Algorithm 2. Following the definition of the action-value function, the $Q^*(s, a)$ value is the discounted sum of future rewards when action a is executed from state s, and the optimal policy π^* is followed afterwards. To estimate this sum, QL uses the received reward r_{t+1} plus the discounted maximum value of the future state s_{t+1}.

The first step of the algorithm is to initialize the values of the Q function for all the states s and actions a randomly. After that, the algorithm starts interacting with the environment in order to learn the Q^* function. In each iteration, the update function needs an initial state s_t, the executed action a_t, the new state s_{t+1} which has been achieved, and the received reward r_{t+1}. The algorithm updates the value of $Q(s_t, a_t)$, comparing its current value with the sum of r_{t+1} and the discounted maximum Q value in s_{t+1}. The error is reduced with a learning rate α and added to $Q(s_t, a_t)$. When the algorithm has converged to the optimal Q^* function, the learning process can be stopped. The parameters of the algorithm are the discount factor γ, the learning rate α and the ϵ parameter for the random actions.

Algorithm 2: QL algorithm.

```
Initialize $Q(s, a)$ arbitrarily
Repeat until Q is optimal:
    a) $s_t \leftarrow$ the current state
    b) choose action $a_{max}$ that maximizes $Q(s_t, a)$ over all $a$
    c) $a_t \leftarrow (\epsilon - greedy)$ action, carry out action $a_{max}$ in the
    world with probability $(1 - \epsilon)$ (exploitation), otherwise apply
    a random action (exploration)
    d) Observe the reward $r_{t+1}$ and the new state $s_{t+1}$
    e) $Q(s_t, a_t) =$
    $Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
    f) $s_t \leftarrow s_{t+1}$
```

At each iteration, the algorithm perceives the state from the environment and receives the reward. After updating the Q value, the algorithm generates the action to be taken. The Q function is represented as a table with a different state in each row and a different action in each column. Both the state space and the action space can contain different variables with different values. As can be observed, the algorithm has to store the past state s_t and the past action a_t.

Other algorithms are able to solve the RLP by learning the policy directly. In such algorithms, the policy is explicitly represented by its own *function approximator*, independent of the VF and is updated according to the gradient of the expected reward with respect to the policy parameters. This approach comprises a wide variety of algorithms called *Policy Gradient algorithms*. The survey that follows is focused on PG algorithms that do not require the dynamics to be known and can be performed on-line.

2.6 POLICY GRADIENT ALGORITHMS

Most of the methods proposed in the reinforcement learning community are not applicable to high-dimensional systems as these methods do not scale beyond systems with more than three or four degrees of freedom and/or cannot deal with parameterized policies [114]. PG methods are a notable exception to this statement. Starting with the work in the early 1990s [25, 55], these methods have been applied to a variety of robot learning problems ranging from simple control tasks [26] to complex learning tasks involving many degrees of freedom [117]. The advantages of PG methods for robotics are numerous. Among the most important are that they have good generalization capabilities which allow them to deal with big state-spaces, that their policy representations can be chosen so that it is meaningful to the task and can incorporate previous domain knowledge and that often fewer parameters are needed in the learning process than in VF-based approaches. Also, there is a variety of different algorithms for PG estimation in the literature which have a rather strong theoretical underpinning. In addition, PG methods can be used model-free and therefore also be applied to robot problems without an in-depth understanding of the problem or mechanics of the robot [115]. Studies have shown that approximating a policy directly

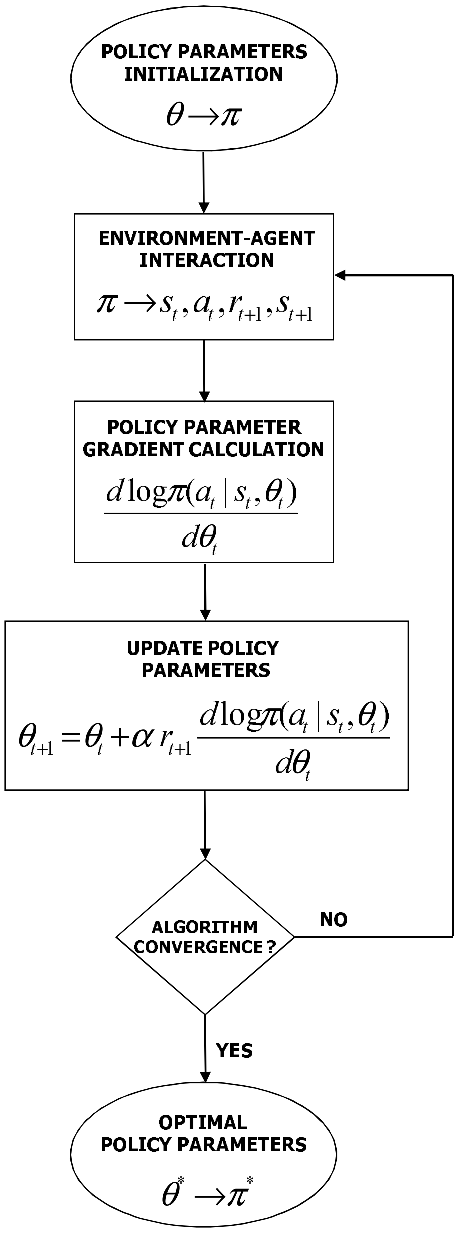
POLICY PARAMETERS INITIALIZATION
$\theta \rightarrow \pi$
ENVIRONMENT-AGENT INTERACTION
$\pi \rightarrow s_t, a_t, r_{t+1}, s_{t+1}$
POLICY PARAMETER GRADIENT CALCULATION
$\frac{d \log \pi(a_t \mid s_t, \theta_t)}{d\theta_t}$
UPDATE POLICY PARAMETERS
$\theta_{t+1} = \theta_t + \alpha \, r_{t+1} \frac{d \log \pi(a_t \mid s_t, \theta_t)}{d\theta_t}$
ALGORITHM CONVERGENCE ?
NO
YES
OPTIMAL POLICY PARAMETERS
$\theta^* \rightarrow \pi^*$

Figure 4: Diagram of a PG-based RL algorithm. The policy parameters are updated according to the gradient of the current policy. Once the optimal policy parameters are found, current policy π becomes optimal π^*.

can be easier than working with VFs [153, 9] and better results can be obtained. Informally, it is intuitively simpler to determine *how to act* instead of *value of acting* [1]. So, rather than approximating a VF, new methodologies approximate a stochastic policy using an independent *function approximator* with its own parameters, trying to maximize the future reward expected. In Equation 2.18 we can see that in the PG approach, the eligibility $e_t(\theta)$ is represented by the gradient of a given stochastic policy $\pi(a_t|s_t, \theta_t)$ with respect to the current parameter vector θ_t as

$$e_t(\theta) = \frac{d \log \pi(a_t|s_t, \theta_t)}{d\theta_t}. \tag{2.18}$$

In section 2.5.1 it is explained that, in the VF approach, the eligibility $e_t(s)$ is bound to the state. It represents a record of the number of visits of the different states while following current policy π. In the PG approach, $e_t(\theta)$ is bound to the policy parameters. It specifies a correlation between the associated policy parameter θ_i and the executed action a_t while following current policy π. Therefore, while the VF's eligibility indicates *how much* a particular state is *eligible* for receiving learning changes, PG's eligibility indicates *how much* a particular parameter of our policy is *eligible* to be improved by learning. The final expression for the policy parameter update is

$$\theta_{t+1} = \theta_t + \alpha r_{t+1} e_t(\theta). \tag{2.19}$$

Here, α is the learning rate of the algorithm and r_t is the immediate reward perceived. Figure 4 represents the phase-diagram of PG-based algorithms. At every iteration step, the algorithm proposes a learning update rule to modify a parameterized policy function $\pi(a|s, \theta)$. This rule is based on the policy derivative with respect to the policy parameters and the immediate reward r_{t+1}. Once the PG algorithm converges, the current policy π becomes the optimal policy π^*. The solution of the RLP is accomplished by following optimal policy π^*.

The next sections introduce several PG methods for learning algorithms. The algorithms presented hereafter detail the most important methodologies from over the last few years and constitute the basic foundation of most successful practical applications which solve the RLP using PG techniques.

2.6.1 *REINFORCE*

The first example of an algorithm optimizing the averaged reward obtained for stochastic policies working with gradient direction estimates is Williams' REINFORCE algorithm [171]. This algorithm learns much more slowly than other RL algorithms which work with a VF 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. The algorithm operates like the basic PG algorithm presented in Section 2.6 by adjusting policy parameters to a direction that lies along the gradient of expected reinforcement rewards following the expression

$$\Delta\theta_{t+1} = \alpha(r_{t+1} - b_{t+1})e_t(\theta). \tag{2.20}$$

Here, α is the learning rate of the algorithm and $e_t(\theta) = \frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$ is the eligibility of a particular parameterized stochastic policy $\pi(a_t|s_t, \theta_t)$. As a novelty, Williams introduces a b term called *reward baseline*. This term gives the algorithm a choice to work with immediate ($b = 0$) or delayed ($b \neq 0$) rewards. As can be seen in Equation 2.21, the reward baseline includes a discount factor γ ($0 \leqslant \gamma \leqslant 1$) that maintains an adaptive reward of the upcoming reinforcement based on past experience, i. e.,

$$b_{t+1} = \gamma r_{t+1} + (1 - \gamma)b_t. \tag{2.21}$$

Several researchers have previously shown that the use of a reward baseline does not bias the gradient estimate, but motivation to choose a particular baseline form has mainly been based on qualitative arguments and empirical success [148, 170, 41]. The optimal baseline which does not bias the gradient can only be a single number for all trajectories and can also depend on the time-step [117]. However, in the PG theorem it can depend on the current state and, therefore, if a good parameterization for the baseline is known, e.g., in a generalized linear form $b(x_t) = \phi(x_t)^T w$, this can significantly improve the gradient estimation process. However, the selection of the *basis functions* $\phi(x_t)$ can be difficult and often impractical in practice [114] since they may introduce more bias. Therefore, some theorems provide guidance in choosing a baseline [165, 52, 171, 166, 79]. The expression that gives us the update rule for every parameter of our policy is formulated as

$$\theta_{t+1} = \theta_t + \alpha(r_{t+1} - b_{t+1})e_t(\theta). \tag{2.22}$$

As can be easily noticed, the eligibility described in REINFORCE does not accumulate information about past actions, it just represents the correlation between parameters and actions taken at any time step t. Kimura and Kobayashi extended Williams' algorithm to the infinite horizon and modified the eligibility to become an *eligibility trace* [69]. A decay factor λ ($0 \leqslant \lambda \leqslant 1$) similar to the one used in Sutton's TD(λ) is added into the calculation of the new eligibility trace $z_t(\theta)$, i. e.,

$$z_{t+1}(\theta) = \lambda z_t(\theta) + e_t(\theta). \tag{2.23}$$

Hence, we obtain

$$\theta_{t+1} = \theta_t + \alpha(r_{t+1} - b_{t+1})z_{t+1}(\theta). \tag{2.24}$$

Williams' original eligibility $e(\theta)$ contained information about actions immediately executed. Kimura and Kobayashi's eligibility trace $z(\theta)$ acts as a discounted running average of the eligibility. As we move the decay factor λ close to 1, the memory of the agent concerning past actions increases. Adding eligibility traces considerably improves the performance of the algorithm but, as a drawback, this gradient version becomes biased and does not correspond to the correct gradient unlike [54]. The bias-variance tradeoff in gradient estimates are a critical factor in PG algorithms and nowadays it represents a hot topic for most researchers. A good balance between them directly affects the algorithm's performance. The REINFORCE with eligibility traces algorithm is described in Algorithm 3.

The easy structure of these kinds of algorithms, without any computational complexity, and their capability of mildly adapting to POFMDP, make REINFORCE-based methods a good startup for designing algorithms for real robot applications in unknown environments.

Algorithm 3: REINFORCE with eligibility traces algorithm.

```
Initialize parameter vector $\theta_0$ arbitrarily and set $z_0(\theta) = 0$ and $b_0 = 0$.
Repeat until $\pi(a|s,\theta)$ is optimal:
    a) $s_t \leftarrow$ the current state
    b) $a_t \leftarrow$ action given by $\pi(a_t|s_t,\theta_t)$
    c) Take action $a_t$, observe reward $r_{t+1}$
  Calculate eligibility $e_t(\theta)$ and eligibility trace $z_t(\theta)$
    d) $e_t(\theta) = \frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$
    e) $z_{t+1}(\theta) = \lambda z_t(\theta) + e_t(\theta)$
  Calculate reinforcement baseline and improve policy
    f) $b_{t+1} = \gamma r_{t+1} + (1-\gamma) b_t$
    g) $\theta_{t+1} = \theta_t + \alpha(r_{t+1} - b_{t+1}) z_{t+1}(\theta)$
```

2.6.2 *Gradient Partially Observable Markov Decision Process (GPOMDP)*

The variance of a gradient estimator remains a significant practical problem for PG applications. Although the eligibility traces introduced by Kimura and Kobayashi have proven effective, discounting rewards introduce the bias-variance balance problem: variance in the gradient estimates can be reduced by heavily discounting rewards, but the estimates will be biased. In the same way, the bias can be reduced by not discounting so much, but the variance will then be higher [23]. The GPOMDP algorithm proposed by Baxter and Bartlett aims to improve the gradient estimation. This algorithm demonstrates that a bad selection of a reinforcement baseline b only increases the bias of gradient estimates and does not even improve its variance. In fact, the bias-variance tradeoff can be controlled by just correctly selecting an appropriate value of the decay factor λ. As λ approaches 1, the bias of the estimates decreases, but its variance increases.

Let θ represent the parameter vector of an approximate function $\pi(a|s,\theta)$ that maps a stochastic policy. The gradient function $\frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$ computes approximations based on a continuous sample path of the Markov chain of the parameterized POFMDP. The accuracy of the approximation is controlled by the decay factor $\lambda \in [0,1)$ of the eligibility trace $z(\theta)$ which increases or decreases the agent's memory of past actions as in Kimura's algorithm. Furthermore, given a POFMDP and a randomized differentiable policy $\pi(a|s,\theta)$ for all observed states and actions with initial parameter values θ, the update of the eligibility trace $z(\theta)$ and the policy parameter vector θ are completed by

$$z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d \log \pi(a_t|s_t, \theta_t)}{d\theta_t}, \tag{2.25}$$

and

$$\theta_{t+1} = \theta_t + \alpha r_{t+1} z_{t+1}(\theta). \tag{2.26}$$

Here, α is the learning rate of the algorithm and r is the immediate reward received. As the value of λ increases, the memory of the agent increases, however, variance of the estimates $z_{t+1}(\theta)$ also rises with this parameter. The GPOMDP algorithm is detailed in Algorithm 4.

Algorithm 4: The GPOMDP algorithm.

```
Initialize parameter vector $\theta_0$ arbitrarily and set $z_0(\theta) = 0$.
Repeat until $\pi(a|s, \theta)$ is optimal:
    a) $s_t \leftarrow$ the current state
    b) $a_t \leftarrow$ action given by $\pi(a_t|s_t, \theta_t)$
    c) Take action $a_t$, observe reward $r_{t+1}$
  Calculate eligibility trace $z_{t+1}(\theta)$
    d) $z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$
  Policy improvement
    e) $\theta_{t+1} = \theta_t + \alpha r_{t+1} z_{t+1}(\theta)$
```

The low mathematical complexity shown by this algorithm together with its online feasibility and the improvements in variance vs. bias tradeoff make the GPOMDP approach a basic algorithm for real applications. At this point, there is no special mathematical difference between the GPOMDP and the REINFORCE approach. Since most of the literature uses the Baxter and Bartlett approach as a basic PG platform for testing, the GPOMDP algorithm was used in some real and simulated testing during the final results of this dissertation.

2.6.3 *Compatible Function Approximation*

As we have previously shown in Section 2.6.1, the natural alternative to using approximate VFs is problematic as these introduce bias in the presence of imperfect basis functions. However, as demonstrated in [73, 153] the term

$Q^\pi(x,u) - b^\pi(x)$ can be replaced by a *compatible function approximation* $f^\pi(s,a)$. The algorithm proposed in [74, 152] defines the VF as a linearly parameterized approximation with its own parameter vector. The main advantage of the proposed algorithm lies in the fact that, due to the small dimension of the parameter vector compared with the space dimension, it is not necessary to find the exact approximated VF, but what they named a reduced *projection* of the VF. Therefore, for the action-value function approach, a compatible function approximation is described by a linear feature-based parameterized approximation without affecting the unbiasedness of the gradient estimate and irrespective of the choice of the baseline $b^\pi(s)$ as

$$Q^\pi(s,a) - b^\pi(s) = f^\pi(s,a) = \phi(s,a)^T w. \tag{2.27}$$

Here w represents the parameter vector and $\phi(s,a)$ is the features function. If the algorithm includes an eligibility trace $z(w_t)$, the parameter update at any time step t is performed as

$$\begin{aligned} z_{t+1}(w) &= \lambda z_t(w) + \phi(s_{t+1}, a_{t+1}), \\ w_{t+1} &= w_t + \alpha\delta_t z_{t+1}(w). \end{aligned} \tag{2.28}$$

The variable α denotes the learning rate while λ is the decay factor of its eligibility. The calculation of TD-error δ_t is defined by

$$\delta_{t+1} = r_{t+1} + \phi(s_{t+1}, a_{t+1})^T w_t - \phi(s_t, a_t)^T w_t. \tag{2.29}$$

Similarly to the TD-error, a particular policy $\pi(s,a,\theta)$ can be approximated by

$$\pi(s,a,\theta) = Z_\theta \exp(\phi(s,a)^T w_t) \tag{2.30}$$

where $Z_\theta = \Sigma_a \exp(\phi(s,a))$, θ is the parameter vector and $\psi(s,a)$ is the features function. The policy updates its parameter vector θ according to the TD-error and the features function $\psi(s_{t+1}, a_{t+1})$ by

$$\theta_{t+1} = \theta_t + \beta\delta_{t+1}\psi(s_{t+1}, a_{t+1}), \tag{2.31}$$

where β denotes a learning rate. The algorithm procedure is summarized in Algorithm 5.

Algorithm 5: Compatible Function Approximation.

```
Initialize policy $\pi(s, a)$ and evaluation VF with parameter
vectors $\theta_0$ and $w_0$ respectively. Set feature vectors $\psi(s, a)$ and
$\phi(s, a)$ for parameterization. Set eligibility trace $z_0(w) = 0$.
Repeat until $\pi(s, a, \theta)$ is optimal:
    a) $a_t \leftarrow$ action given by $\pi(s_t, a_t, \theta_t)$
    b) Take action $a_t$, observe next state $s_{t+1}$ and reward $r_{t+1}$
    Policy evaluation:
    d) $\delta_{t+1} = r_{t+1} + \phi(s_{t+1}, a_{t+1})^T w_t - \phi(s_t, a_t)^T w_t$
    e) $z_{t+1}(w) = \lambda z_t(w) + \phi(s_{t+1}, a_{t+1})$
    f) $w_{t+1} = w_t + \alpha \delta_{t+1} z_{t+1}(w)$
    Policy update:
    g) $\theta_{t+1} = \theta_t + \beta \delta_{t+1} \psi(s_{t+1}, a_{t+1})$
```

2.6.4 *Natural Policy Gradients*

PG algorithms have often exhibited slow convergence. PG convergence speed is directly related to the direction of the gradient, and those found around plateau landscapes of the expected reward may be small and not point towards the optimal solution [114]. These poor performance results made Sham Kakade think whether we were obtaining the right gradient [66]. The GPOMDP approach described in the previous section offered the most straightforward approach of policy improvement, following the gradient in policy parameter space using the steepest gradient ascent,

$$\theta_{t+1} = \theta_t + \alpha \nabla \eta(\theta) \tag{2.32}$$

where $\nabla \eta(\theta)$ is the gradient of the averaged reward function of the POFMDP with the parameter vector θ. In [7], Amari pointed out that the natural PG may be a good alternative to the PG described above. A natural gradient is the one that looks for the steepest ascent with respect to the *Fisher information matrix* [7] instead of the steepest direction in the parameter space. The Fisher information is a way of measuring the amount of information that an observable random variable x carries about an unknown parameter θ upon which the *likelihood function* of θ, $L(\theta) = f(x, \theta)$, depends. The natural gradient of the averaged reward function $\tilde{\nabla} \eta(\theta)$ can be expressed as a function of the Fisher information matrix $F(\theta)$ and the gradient of the averaged reward $\nabla \eta(\theta)$ as

$$\tilde{\nabla} \eta(\theta) = F^{-1}(\theta) \nabla \eta(\theta). \tag{2.33}$$

By inserting a *compatible function approximation* [114] parameterized by the vector w into the PG we obtain

$$\tilde{\nabla}\eta(\theta) = F^{-1}(\theta)G(\theta)w. \tag{2.34}$$

Here the matrix $G(\theta)$ is known as the all-action matrix [66]. It has been demonstrated in [114] that it corresponds to the Fisher information matrix $G(\theta) = F(\theta)$ so the natural gradient can be computed as

$$\tilde{\nabla}\eta(\theta) = F^{-1}(\theta)F(\theta)w = w. \tag{2.35}$$

The policy parameter vector update is finally given by

$$\theta_{t+1} = \theta_t + \alpha w_t. \tag{2.36}$$

Here α is the learning rate of the algorithm. The procedures of a PG algorithm with natural gradient computation is detailed in Algorithm 6.

Algorithm 6: Natural Gradient algorithm.

```
Initialize parameter vector $\theta_0$ arbitrarily and set $z_0(\theta) = 0$ and
$b_0 = 0$.
Repeat until $\pi(s, a, \theta)$ is optimal:
        a) $s_t \leftarrow$ the current state
        b) $a_t \leftarrow$ action given by $\pi(s_t, a_t, \theta_t)$
        c) Take action $a_t$, observe reward $r_{t+1}$
    Calculate the natural gradient of the expected reward $\eta(\theta)$
        d) $\tilde{\nabla}\eta(\theta) = F^{-1}(\theta)F(\theta)w = w_t$
    Policy improvement
        e) $\theta_{t+1} = \theta_t + \alpha w_t$
```

Being easier to estimate than regular PGs, natural gradients are expected to be more efficient and therefore accelerate the convergence process. Also, a more direct path to the optimal solution in parameter space increases convergence speed and avoids premature undesirable convergence.

2.7 VALUE FUNCTION VS POLICY GRADIENT

In previous sections we presented the foundation of VF and PG techniques. Advantages and drawbacks of each one were also discussed when dealing with real robotic tasks. This section compares both methodologies directly with the aim of finding the most suitable solution for the particular experimental tasks where RL techniques are applied in this dissertation.

The dominant approach over the last decade has been to apply RL using the VF approach. As a result, many RL-based control systems have been applied to robotics. In [146], an instance-based learning algorithm was applied to a real robot in a corridor-following task. For the same task, in [59] a hierarchical memory-based RL was proposed, obtaining good results as well. In [35], an underwater robot learns different behaviors using a modified QL algorithm. VF methodologies have worked well in many applications, achieving great success with discrete lookup table parameterization but giving few convergence guarantees when dealing with high dimensional domains due to the lack of *generalization* among continuous variables [152].

RL is usually formulated using FMDPs. 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 ANNs, instance-based methods or decision-trees. In some cases, QL can fail to converge to a stable policy in the presence of function approximation, even in MDPs, and it may be difficult to calculate $max_{a \in A(s)} Q^*(s, a)$ when dealing with continuous space-states [104]. Another feature of VF methods is that these approaches are oriented to finding deterministic policies. However, stochastic policies can yield considerably higher expected rewards than deterministic ones as in the case of POFMDPs, selecting among different actions with specific probabilities [142]. Furthermore, some problems may appear when the state-space is not completely observable, small changes in the estimated value of an action may or may not cause it to be selected, resulting in convergence problems [27].

Rather than approximating a VF, policy search techniques approximate a policy using an independent function approximator with its own parameters, trying to maximize the future reward expected [115]. The advantages of PG methods over VF-based methods are various. The main advantage is that using a function approximator to represent the policy directly solves the generalization problem. 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 POFMDPs than those obtained with deterministic value-function methods [142]. In [9] a comparison between a policy-only algorithm [23] and a value QL method [164] is presented where the QL oscillates between the optimal and a suboptimal policy while the policy-only method converges to the optimal policy.

Attempts to apply the PG algorithms to real robotic tasks have shown slow convergence, in part caused by the small gradients around plateau landscapes of the expected return [114]. In order to avoid these situations, studies presented in [7] pointed that the natural PG may be a good alternative to the *typical* PG. Being easier to estimate than regular PGs, they are expected to be more efficient and therefore accelerate the convergence process. Natural gradient algorithms have found a variety of applications over the last few years, as in [125] with traffic-light system optimization and in [161] with gait optimization for robot locomotion.

PG applications share a common drawback, gradient estimators used in these algorithms may have a large variance [88, 74], learning much more slowly than RL algorithms using a VF (see [153]) and they can converge to local optima of the expected reward [96], making them less suitable for on-line learning in real applications. In order to decrease convergence time and avoid local optima, the newest applications combine PG search with VF techniques, adding the best features of both methodologies [20, 74]. These techniques are commonly known as Actor-Critic (AC) methods. In AC algorithms, the critic component maintains a VF, and

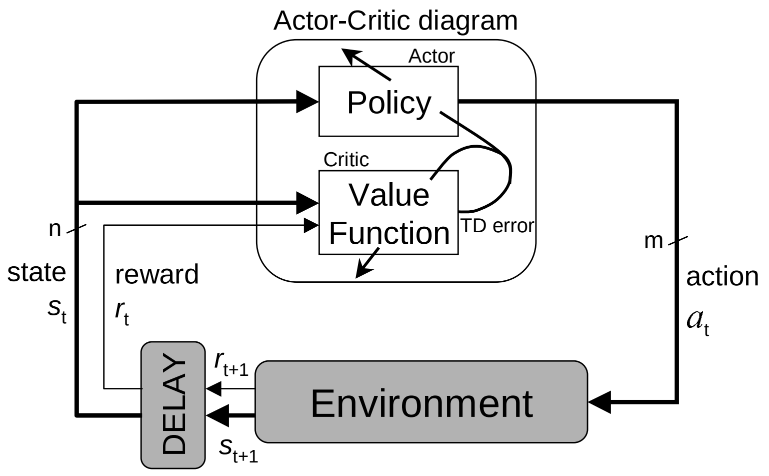

Figure 5: General diagram of the AC methods.

the actor component maintains a separate parameterized stochastic policy from which the actions are drawn. The Actor's PG gives convergence guarantees while the critic's VF reduces variance of the policy update improving the convergence rate of the algorithm [74]. AC basic procedures as well as its most representative algorithms are discussed in the next section.

2.8 ACTOR-CRITIC ALGORITHMS

Actor-Critic (AC) methods [172] combine some advantages of PG algorithms and VF methods. AC methods have two distinctive parts, the *actor* and the *critic*. The actor part contains the policy to be followed. It observes the state of the environment and generates an action according to this policy. On the other hand, the critic observes the evolution of the states and criticizes the actions made by the actor. The critic contains a VF which tries to learn according to the actor policy. A common property of AC methods is the fact that the learning is always *on-policy* which means that, at any time step, the action to be followed is the one indicated by the actor policy. If the agent does not follow this action, the algorithm cannot converge to the RLP solution. The critic will always learn the state-value function for this policy. Figure 5 shows the general diagram of an AC method.

In AC methods, the critic typically uses the state-value function $V^{\pi}(s)$. The actor is initialized to a policy which relates all the states with an action, and the critic learns the state-value function of this policy. According to the values of the visited states, the critic calculates the TD-error and informs the actor. Finally, the actor modifies its policy according to this value difference. If the value of two consecutive states increases, the probability of taking the applied action increases. On the contrary, if the value decreases, the probability of taking that action decreases. AC methods refine the initial policy until the optimal state-value function $V^*(s)$ and an optimal policy π^* are found. The next sections introduce basic AC techniques for training algorithms. The algorithms presented hereafter detail the most important methodologies from over the last few years and constitute the basic foundation of most successful practical applications which solve the RLP using AC techniques.

2.8.1 *Original Actor-Critic*

The main idea and foundation of an AC algorithm was first proposed in [152]. The method follows the structure previously described in Section 2.8. The critic takes the form of a TD error, it learns and critiques whatever policy currently being followed by the actor. Typically, the critic is a state-value function. After each action selection, it evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.37}$$

where $V(s)$ is the value function implemented by the critic, r_{t+1} is the immediate reward perceived and $0 \leqslant \gamma < 1$ is the discount factor. The algorithm's procedure is quite simple. The value of the TD error is used to evaluate the actor's current policy, i. e., the action a_t chosen in state s_t. If the value of the TD error is positive, it means that, from state s_t, taking action a_t should be enforced in the future. If the value of the TD error is negative, it means that the tendency to select a_t from s_t should be weakened. Suppose the actor is implemented by a stochastic policy $\pi(a|s,\theta)$ parameterized by vector θ. The critic's error estimate δ_t allows the actor to upgrade its policy according to

$$\theta_{t+1} = \theta_t + \alpha\delta_t \frac{d \log \pi(a_t|s_t, \theta_t)}{d\theta_t}. \tag{2.38}$$

Here α is the learning rate for the actor. The critic updates its VF according a common TD procedure given by

$$V(s_t) = V(s_t) + \beta\delta_t. \tag{2.39}$$

Here the β term corresponds to the learning rate for the critic. Many of the earliest RL systems that used TD methods were AC methods [172, 20]. Since then, the attention given to these methods has increased greatly, mainly because they have two significant apparent advantages: they require minimal computation to select actions and they can learn an explicitly stochastic policy [152]. The algorithm procedure is summarized in Algorithm 7.

Algorithm 7: Original Actor-Critic (AC).

```
Initialize policy $\pi(a|s,\theta)$ with initial parameters $\theta = \theta_0$, its
derivative $d\log\pi(a|s,\theta)$, an arbitrarily VF $V^{\pi}(s)$.
Repeat until $\pi(a|s,\theta)$ is optimal:
    a) $a_t \leftarrow$ action given by $\pi(a_t|s_t,\theta_t)$
    c) Take action $a_t$, observe next state $s_{t+1}$ and reward $r_{t+1}$
    Critic evaluation and update:
      b) $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
      e) $V(s_t) = V(s_t) + \alpha\delta_t$
    Actor update:
      d) $\theta_{t+1} = \theta_t + \alpha\delta_t \frac{d\log\pi(a_t|s_t,\theta_t)}{d\theta_t}$
```

2.8.2 *Traced Actor-Critic*

In Section 2.6.1, Kimura and Kobayashi introduced the eligibility trace $z(\theta)$ to the PG estimates proposed in REINFORCE. This addition improved the overall performance of the algorithm considerably. Good results encouraged them to apply trace calculation to AC methods. Kimura and Kobayashi's algorithm [68] proposes a classic AC method which follows the rules previously presented in Figure 5. As will be appreciated in the various AC methodologies discussed in this section, the utilization of the eligibility trace in AC algorithms can vary from one algorithm to another. Some methods use them as a part of the actor update, others use them in the critic and others apply them to both sides. Kimura and Kobayashi's proposed algorithm utilizes the eligibility traces for the actor parameter update. The next lines summarize the whole algorithm's procedure. The actor is implemented by a stochastic policy $\pi(a|s,\theta)$ parameterized by vector θ. The critic attempts to estimate the evaluation function for the current policy. The reinforcement used is the value function TD-error δ as shown in the following equation

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{2.40}$$

Here r_{t+1} is the immediate reward perceived and $0 \leqslant \gamma < 1$ is the discount factor. On the actor's side, the eligibility trace $z(\theta)$ is represented as the decayed sum of the eligibility of a stochastic policy $\pi(a|s,\theta)$ with respect to the current parameter vector θ as

$$z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}, \tag{2.41}$$

$0 \leqslant \lambda \leqslant 1$ being the decay factor of the eligibility trace. The critic's error estimate δ allows the actor to upgrade its policy according to

$$\theta_{t+1} = \theta_t + \alpha\delta_t z_{t+1}(\theta). \tag{2.42}$$

Here α is the learning rate for the actor. The critic updates its VF according to a common TD procedure given by

$$V(s_t) = V(s_t) + \beta\delta_t. \tag{2.43}$$

Here the β term corresponds to the learning rate for the critic. The algorithm procedure is summarized in Algorithm 8.

2.8.3 *Single Value and Policy Search (VAPS)*

The VAPS algorithm [18] allows PG and VF methods to be combined simultaneously. With the aid of a mixing term β, the VAPS algorithm computes the differential error value of a state $e_{va-po}(s)$ as a combination of a PG and a VF as

$$e_{va-po}(s_{t+1}) = (1-\beta)\delta_{va}(s_{t+1}) + \beta(b - \gamma r_{t+1}), \tag{2.44}$$

where δ_{va} is the TD-error function for the VF approach and $b - \gamma r_{t+1}$ corresponds to the averaged reward of a PG

Algorithm 8: Traced AC: Eligibility traces for actor update.

```
Initialize policy $\pi(a|s,\theta)$ with initial parameters $\theta = \theta_0$, its
derivative $d\log\pi(a|s,\theta)$, an arbitrarily VF $V^{\pi}(s)$ and set the
eligibility trace $z(\theta) = 0$.
Repeat until $\pi(a|s,\theta)$ is optimal:
    a) $a_t \leftarrow$ action given by $\pi(a_t|s_t,\theta_t)$
    c) Take action $a_t$, observe next state $s_{t+1}$ and reward $r_{t+1}$
    Critic evaluation and update:
      b) $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
      e) $V(s_t) = V(s_t) + \alpha\delta_t$
    Actor update:
      c) $z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d\log\pi(a_t|s_t,\theta_t)}{d\theta_t}$
      d) $\theta_{t+1} = \theta_t + \alpha\delta_t z_{t+1}(\theta)$
```

algorithm which uses a reinforcement baseline b and a discount factor γ, like the classic PG algorithms detailed in Section 2.6.1. Adjustments of parameter β between 0 and 1 allows us to go back and forth between both RL methods. When $\beta = 0$, the algorithm totally learns the action-value function $Q^{\pi}(s,a)$ that satisfies the Bellman equation

$$\delta_{va}(s_{t+1}) = r_{t+1} + \gamma max_{a_{t+1}} Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t). \tag{2.45}$$

On the other hand, if $\beta = 1$, the algorithm directly learns a policy that will minimize the expected total discounted reward. An eligibility trace is added for the stochastic policy update

$$z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d\log\pi(a_t|s_t,\theta_t)}{d\theta_t}. \tag{2.46}$$

Here λ is the decay factor of the eligibility trace. The current differential error value of a state $e_{va-po}(s)$ is finally used for updating the policy parameters as shown in

$$\Delta\theta_t = -\alpha\left[\frac{de_{va-po}(s_{t+1})}{d\theta} + e_{va-po}(s_{t+1})z_{t+1}(\theta)\right] \tag{2.47}$$

and

$$\theta_{t+1} = \theta_t + \Delta\theta_t. \tag{2.48}$$

The variable α represents a learning rate. The VAPS algorithm procedure is summarized in Algorithm 9. When $\beta = 0$, the algorithm behaves as a pure VF method, the new algorithm converges but it cannot learn the optimal policy as the reward function includes the Bellman residual. When $\beta = 1$, the algorithm behaves as a pure PG method. In this case the algorithm converges to optimality, but slowly since there is no VF catching the results in the long sequence of states near the end. By combining the two approaches, given a particular RLP, this algorithm can offer a much quicker solution than either of the two alone for the same problem. It is important to mention that this algorithm is known to be biased and it does not approximate a true gradient. Therefore, it has no convergence guarantees.

Algorithm 9: VAPS algorithm.

```
Initialize policy $\pi$, an arbitrarily $Q^{\pi}(s,a)$ and set $z(\theta) = 0$.
Repeat until $\pi(a|s,\theta)$ is optimal:
    a) $a_t \leftarrow$ action given by $\pi(a_t|s_t,\theta_t)$
    b) Take action $a_t$, observe next state $s_{t+1}$ and reward $r_{t+1}$
    Evaluate the policy mixing the action value error and the
    expected discounted reward:
    c) $\delta_{va}(s_{t+1}) = r_{t+1} + \gamma Q^{\pi}(s_{t+1},a_{t+1}) - Q^{\pi}(s_t,a_t)$
    d) $\delta_{va-po}(s_{t+1}) = (1-\beta)\delta_{va}(s_{t+1}) + \beta(b - \gamma r_{t+1})$
    Update eligibility trace and improve policy:
    e) $z_{t+1}(\theta) = \lambda z_t(\theta) + \frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$
    f) $\theta_{t+1} = \theta_t - \alpha \left[\frac{d e_{va-po}(s_{t+1})}{d\theta} + e_{va-po}(s_{t+1}) z_{t+1}(\theta)\right]$
```

2.8.4 *The Natural Actor-Critic (NAC)*

The NAC algorithm [117] was proposed by Jan Peters and Stefan Schaal. This algorithm combines AC techniques with natural gradient computation, detailed in Section 2.6.4, with the aim of obtaining advantages from both methodologies and achieve faster convergence. Stochastic natural PGs allow actor updates while the critic computes the natural gradient and the VF parameters by linear regression simultaneously. The actor's PG gives convergence guarantees under function approximation and partial observability while the critic's VF reduces variance of the estimates update improving the convergence process. The parameter update procedure starts on the critic's side. At any time step t, the features $\phi(s)$ of the designed basis function are updated according to the gradients of the actor's policy parameters θ as shown in

$$\tilde{\phi}(s_t) = [\phi(s_{t+1}), 0] \tag{2.49}$$

and

$$\hat{\phi}(s_t) = [\phi(s_t), \frac{d \log \pi(a_t|s_t, \theta_t)}{d\theta_t}]. \tag{2.50}$$

These features are then used to update the critic's parameters and find the natural gradient. This algorithm uses a variation of the Least Squares Temporal Difference (LSTD)(λ) [29] technique called LSTD-Q(λ). Thus, instead of performing a gradient descent, the algorithm computes estimates of matrix A and b and then solves the equation $b + A\tau = 0$ where the parameter vector τ encloses both, the critic parameter vector ν and the natural gradient vector w:

$$\begin{aligned} z(s_{t+1}) &= \lambda z(s_t) + \hat{\phi}(s_t), \\ A(s_{t+1}) &= A(s_t) + z(s_{t+1})(\hat{\phi}(s_t) - \gamma\tilde{\phi}(s_t)), \\ b(s_{t+1}) &= b(s_t) + z(s_{t+1})r_t, \\ [\nu_{t+1}, w_{t+1}] &= A_{t+1}^{-1} b_{t+1}. \end{aligned} \tag{2.51}$$

Here, λ is the decay factor of the critic's eligibility, r_t is the immediate reward perceived and γ represents the discount factor of the averaged reward. On the actor's side, the current policy is updated when the angle between two consecutive natural gradients is smaller than a given threshold, $\epsilon \ \angle(w_{t+1}, w_t) \leqslant \epsilon$ according to

$$\theta_{t+1} = \theta_t + \alpha w_{t+1}. \tag{2.52}$$

Here, α is the learning rate of the algorithm. Before starting a new loop, a forgetting factor β is applied to the critic's statistics as shown in

$$\begin{aligned} z(s_{t+1}) &= \beta z(s_{t+1}), \\ A(s_{t+1}) &= \beta A(s_{t+1}), \\ b(s_{t+1}) &= \beta b(s_{t+1}). \end{aligned} \tag{2.53}$$

The algorithm's procedure is summarized in Algorithm 10.

Algorithm 10: NAC algorithm with LSTD-Q(λ)

```
Initialize $\pi(a|s,\theta)$ with initial parameters $\theta = \theta_0$, its derivative
$\frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t}$ and basis function $\phi(s)$ for the VF $V^{\pi}(s)$. Draw
initial state $s_0$ and initialize $z = A = b = 0$.
for $t = 0$ to $n$ do:
    Generate control action $a_t$ according to current policy $\pi_t$.
    Observe new state $s_{t+1}$ and the reward obtained $r_t$.
    Critic Evaluation (LSTD-Q($\lambda$))
        Update basis functions
        $\tilde{\phi}(s_t) = [\phi(s_{t+1})^T, 0^T]$
        $\hat{\phi}(s_t) = [\phi(s_t)^T, (\frac{d \log \pi(a_t|s_t,\theta_t)}{d\theta_t})^T]^T$
        Update sufficient statistics:
        $z(s_{t+1}) = \lambda z(s_t) + \hat{\phi}(s_t)$
        $A(s_{t+1}) = A(s_t) + z(s_{t+1})(\hat{\phi}(s_t) - \gamma\tilde{\phi}(s_t))^T$
        $b(s_{t+1}) = b(s_t) + z(s_{t+1})r_t$
        Update critic parameters:
        $[v_{t+1}^T, w_{t+1}^T] = A_{t+1}^{-1} b_{t+1}$
    Actor Update
        If $\angle(w_{t+1}, w_{t-\tau}) \leqslant \epsilon$, then update policy parameters:
        $\theta_{t+1} = \theta_t + \alpha w_{t+1}$
        Forget sufficient statistics:
        $z(s_{t+1}) = \beta z(s_{t+1})$
        $A(s_{t+1}) = \beta A(s_{t+1})$
        $b(s_{t+1}) = \beta b(s_{t+1})$
end
```

2.9 APPLICATIONS OF POLICY GRADIENT METHODS IN ROBOTICS

A background of representative RL techniques has been presented. The survey developed along previous lines started with general solutions to solve the RLP and, as we continued, the interest moved to all those RL methodologies that show interesting advantages when solving real robotic tasks. Fast convergence, on-line capabilities and the power to deal with high dimensional domains are some of the special requirements this survey is looking for. Among all the algorithms reviewed, PG algorithms offer serious advantages for real robotic tasks. With the theoretical aspects clear, this sections aims to discuss practical contributions obtained in real robot learning using PG methods.

2.9.1 *Robot Weightlifting*

If we take a look back in time chronologically, the first successful attempts to apply gradient techniques to real robotics were very simple and barely worried about variance of the estimates, true gradients or convergence issues. Despite the simplicity of those first applications, they kept the essence of PG methods. An example is the algorithm designed by Rosenstein and Barto in 2001 [131]. The proposed method solves the RLP of a weightlifter automata that tries to perform a payload shift episodic task with a limited maximum torque at each of the three robot joints. The policy is represented by a Proportional Derivative (PD) controller whose parameters are the policy parameters to be improved though learning. The update rule is very simple: the algorithm performs a *simple random search* in the parameter space by adding a small amount to each parameter at every iteration step, this amount being normally distributed with zero mean and a variance equal to a particular search size. In order to assure exploration, the algorithm updates the parameters with a fixed probability β or keeps the best ones 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. Every time some new set of parameters is obtained, the PD controller tries them in an attempt to lift the weight. Such off-line configuration disconnects the learning algorithm from the real continuous state space and actions, leaving the real interaction to the PD controller. Although the results that Rosenstein and Barto presented were simulated, its weightlifter was able to learn the task, but the number of trials to achieve a good solution was very high.

2.9.2 *Wheeled Mobile Robot*

In looking for procedures to improve performance of RL methods for real tasks, and even though this one is not a PG technique, it is worth discussing here the ideas presented by Smart and Kaelbling around 2002. This algorithm uses VF techniques to solve an RLP were a wheeled mobile robot learns a control policy for corridor following and obstacle avoidance tasks [147]. The policy is extracted from a QL-based method which follows the procedures stated in Section 2.5.2, with the difference that, instead of applying the common tabular representation of $Q(s, a)$, a continuous function approximator for the Q function is used, allowing the algorithm to deal with continuous state-space representation encountered in the real world. The robot has 2 DoF, rotation around the Z axis and X translation forward and backward. The translation DoF is not learnt and the goal of the algorithm is to learn a good policy for controlling the rotation. The learning is formulated as an episodic task and the rewards are computed as a function of the steps taken to reach the goal at every episode. One of the main contributions of this algorithm is that, in order to overcome the long convergence times where simple RL techniques certainly fall, Smart and Kaelbling's proposal includes prior knowledge in the learning system. To do so, the learning procedure is split into two phases. In the first phase of learning, the robot is controlled by a supplied policy. This can either be a human directly controlling the robot or some kind of control code. During this phase, the learning algorithm is passively watching state transitions and updating the Q function. In the second phase, once the Q function is complete enough to adequately assume command, the learned policy takes control of the robot and continues improving itself. Results show that around 30 episodes are needed to achieve an initial policy in phase one and another 35 episodes more to improve it during phase two.

2.9.3 *Gait Optimization for Quadruped*

Following the procedures of Rosenstein and Barto, in 2004 Kohl and Stone applied a simple random search PG algorithm to learn a task where the commercial quadrupedal robot created by Sony, AIBO, tries to find the fastest possible walking speed [72]. The robot's policy is defined by a set of parameters which refer to different aspects of AIBO's dynamics: front and rear locus, robot heights, etc. Tuning them modifies gait behavior and consequently, the walking speed. Starting from an initial parameter vector, the algorithm generates a set of random policies by adding a small amount to each parameter. Each of the randomly generated policies is evaluated and classified by means of an averaging score and the best adjustment vector from all the policies is extracted. In the next trial, the policy is modified with this adjustment and tested. The process is repeated until convergence. The learning algorithm finds the best policy after 3 hours of learning and, like Rosenstein and Barto's, the learning process of the algorithm proposed by Kohl and Stone is quite simple and performed off-line between each trial, sharing the same slowness found in the previous method. The results demonstrated that PG techniques were a promising application to solve the RLP, but improvements must be made to make it viable for real robotic tasks.

2.9.4 *Optimizing Passive Dynamics Walk*

Up to this point, the attempts to solve real RL tasks via PG methods have been rather simple. They were based on tuning an existing controller in open-loop trajectories with its final performance limited by the initial controller design. In 2004, Tedrake proposes a PG technique to improve a passive dynamics controller [155]. The resultant algorithm is an AC based method similar to the one discussed in Section 2.8.2 and is used to solve an RLP were a biped robot learns to walk from a blank-slate. In order to decrease the number of DoFs and actuators, the mechanical design of the robot was based on a passive dynamic walker [94], with a resultant biped of only 9 DoFs and 4 actuators. The actor is represented by a deterministic policy which is updated using discounted eligibilities. The critic provides an estimate of the VF to the actor by observing the robot's performance on a Poincaré map. Both actor and critic are represented by parameterized linear approximators using non-linear features. As input, the policy has the actual value of the 9 DoFs (6 internal and 3 for the robot's orientation). As output, a 4 dimension vector generates the desired torque for the actuators. The results show that the robot begins to walk after only one minute of learning from blank-slate and the learning converges to the desired trajectory in less than 20 minutes. Also, once the policy is learned, on-line learning capabilities allow the robot to adapt to small changes in the terrain easily. As a drawback, 9 DoFs and 4 actuators represent a high dimensional domain for the policy. Increasing the number of dimensions to upgrade the performance may result in scaling problems. As DoFs are added, the reward assignment to each actuator becomes more difficult, requiring more learning trials to obtain a good estimate of the correlation.

2.9.5 *Learning Planar Biped Locomotion*

Along the same lines as Tedrake's walker, in 2005 Matsubara designed an AC algorithm to make a biped robot learn to walk [91]. The AC method follows the lines described in Section 2.8.2. The actor's policy is represented by a Central Pattern Generator (CPG) composed of a neural oscillator [92], whose weights are the policy parameters to be learnt by the actor. The critic is represented by a VF with its own set of parameters, which estimates the value of the state as a function of the biped's maintenance of the upright position as the first requirement and the forward progress in a second term. Both policy and value approximators are modeled using an ANN. Mechanically, Matsubara's robot differs from Tedrake's in the fact that the design of this biped is not based on passive walker dynamics, thus, an incorrect gait will cause the robot to fall down and the learning process finishes without reaching the objective. Therefore, the aim for a fast convergence algorithm was a priority. In order to increase convergence speed, Matsubara proposes to reduce the dimensionality of the continuous state-space of the problem turning the MDP into a POFMDP. To do so, the input state to the actor includes only the DoFs corresponding to the two hip joints and their derivatives forming a 4 dimensional input vector. Therefore, the rest of the states and the state of the neural oscillator remain hidden for the learning algorithm. The reduction of the number of states make the problem easier to solve but, as a drawback, a more precise value of the known states is needed. For this purpose, a good sensory feedback is a must to achieve satisfactory results. Considering a trial as one attempt to walk without falling, results show that an appropriate policy is learned after around 3000 trials have been completed.

2.9.6 *Robot T-ball*

Successful applications of PG algorithms for real robotic tasks point towards a specific class of methods: AC algorithms. AC algorithms join special properties vital for RL algorithms to succeed in a real task. The actor's PG gives convergence guarantees while the critic's VF reduces variance of the policy update, improving the convergence rate of the algorithm. In 2003, Jan Peters and Stefan Schaal developed an AC algorithm to solve an RLP where a robotic arm learns to hit a ball properly so that it flies as far as possible, like a baseball player would try to do [115]. The algorithm uses Dynamic Movement Primitives (DMP) [138] as a compact representation of a movement. As a novelty, the actor's policy is represented by a parameterized function approximator which is updated with respect to the natural gradient estimates instead of the typical ones. The critic is represented by a VF approximation with its own set of parameters. Peters and Schaal called this algorithm NAC. The input state of the actor's policy is given by the arm's joint angles and derivatives corresponding to the 7 DoFs of the arm. As output, the actor's policy generates accelerations for each of the joints. The critic evaluates the rewards by means of a computer vision algorithm which tracks the whole system and evaluates the performance of every trial. This methodology, as with the wheeled mobile robot in Section 2.9.2, benefits from including prior knowledge in the learning system. This time, a human teaches a rudimentary stroke by taking the robotic hands between his and hitting the ball when it comes. With this information, the arm tries to hit the ball without help in subsequent trials. Results show that after approximately 200-300 trials the ball is correctly hit by the robot arm.

2.9.7 *Learning Motor Skill Coordination*

The results obtained in Section 2.9.6 with the combination of learning from demonstration and DMP techniques has inspired more research in this field. In [76], a robot arm acquires new motor skills by learning the couplings across motor control variables. The skills are first trained from demonstration and, in order to reduce the number of states, encoded in a modified version of DMP. In order to learn new values for the coordination matrices, the proposed method uses an Expectation Maximization (EM)-based RL algorithm called Policy Learning by Weighting Exploration with the Returns (PoWER) developed in [71]. The policy parameterization allows the algorithm to learn the couplings across the different motor control variables. One major advantage of this algorithm over other PG-based approaches is that it does not require a learning rate parameter, always a critical parameter to tune in PG algorithms. Also, the proposed algorithm can be combined with importance sampling to make better use of past experience to focus future exploration of the parameters space. Two learning experiments were performed: a reaching task, where the robot needs to adapt the learned movement to avoid an obstacle, and a dynamic pancake-flipping task. The proposed methodology successfully accomplished both tasks, demonstrating the fitness of the proposed approach in such highly-dynamic real-word tasks.

2.9.8 *Traffic Control*

The performance demonstrated by NAC class algorithms encouraged researchers to try this kind of methodology in other practical applications. In 2006, Richter and Aberdeen used an efficient on-line version of the NAC to solve an RLP where an agent tries to learn an optimal traffic signal control for a city [125]. Considering a street intersection as a POFMDP, the actor's policy is defined as a parameterized function approximator while the critic is represented by a VF approximation with its own set of parameters. The actor's state input variables are measurable traffic parameters related to sequences of cars and time cycles. As output, the agent operates the traffic light at the intersection. The critic's VF computes estimates of the averaged reward considering the immediate reward as the number of cars that entered the intersection over the last time step. Even though the results presented were simulated, the nature of the problem makes no substantial difference between the simulation and the real world. The learning algorithm does not depend directly on a model of the traffic flow, just the ability to interact with it, so the final controller learned can be plugged into a more accurate simulation without modification. Results show that the NAC algorithm outperforms existing traffic controllers, becoming a serious candidate for use in real cities.

2.9.9 *Summary*

A summary of the practical applications described in this section is presented in Table 1. Both, theoretical algorithms and practical applications point towards AC methods as the best for real robotics. Also, benefits from introducing initial reliable knowledge into the learning system are great.

Application	**On/Off line**	**Simulated or real?**	**Need a model?**	**Method used**
Robot Weightlifting	Off-line	Simulated	Yes	PG
Wheeled Mobile Robot	On-line	Real	No	VF
Gait Optimization	Off-line	Real	No	PG
Passive Dynamics Walk	On-line	Real	No	AC
Planar Biped Locomotion	Off-line	Real	No	AC
Robot T-ball	Off-line	Real	No	NAC
Motor Skill Coordination	On-line	Real	No	PoWER
Traffic Control	On-line	Simulated	Yes	NAC

Table 1: A summary of the practical applications described in this section.

This action decreases convergence times and is especially recommended for all those *risky* tasks where the robot needs to start interaction with a real environment from a *safe* position.

3 POLICY GRADIENT METHODS FOR ROBOT CONTROL

In this chapter, we analyze and propose the utilization of different PG techniques for AUV in real robotic tasks. When dealing with real robotics, most of the RL methodologies are implemented as reactive behaviors inside a control architecture. A brief discussion of the evolution of behavior-based control architectures [12] for real robots is given at the beginning of this chapter. It first reviews the history of control architectures for autonomous robots, starting with traditional methods of Artificial Intelligence (AI) and ending with the actual most used behavior-based architectures. The generalization problem, which highly affects real robotics, is described next. The most common approaches to confront this problem and their application to robotics tasks are overviewed. A simple PG algorithm is chosen to carry out the first experimental tests. The GPOMDP algorithm is one of the algorithms proposed in this thesis to build a policy for an RLP in a real autonomous underwater task. To solve the generalization problem, the policy is first approximated by means of an ANN and secondly by a barycentric interpolator. A more complex algorithm is proposed for the second set of results: the NAC. The great performance demonstrated by the NAC algorithm indicates it as the learning algorithm used for the final approach of this thesis. Also, in order to speed up the learning process various techniques are briefly discussed and, among them, a two-step learning process which shares simulated and real learning is chosen as the best option to reduce convergence time. The final proposal, which represents the main contribution of this thesis, is the application in a real underwater robotic task of a two step RL technique such as NAC. For this purpose, the NAC algorithm is first trained in a simulated environment where it can quickly build an initial policy. In the second step the policy is transferred to the real robot to continue the learning process on-line in a real environment. All the experimental results are shown in Chapter 6.

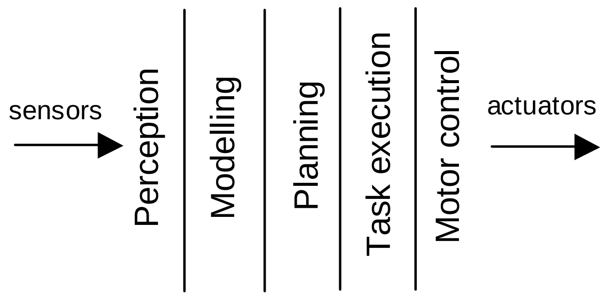

Figure 6: Phases of a classical deliberative control architecture [34].

3.1 EVOLUTION OF CONTROL ARCHITECTURES

The first attempt at building autonomous robots began around the mid-twentieth century with the emergence of Artificial Intelligence. The approach begun at that time was known as *Traditional AI*, *Classical AI* or *Deliberative approach*. Traditional AI relied on a centralized world model for verifying sensory information and generating actions in the world, following the Sense, Plan and Act (SPA) pattern [106]. Its main architectural features were that sensing flowed into a world model, which was then used by the planner, and that plan was executed without directly using the sensors that created the model. The design of the classical control architecture was based on a top-down philosophy. The robot control architecture was broken down into an orderly sequence of functional components and the user formulated explicit tasks and goals for the system [31]. The sequence of phases usually found in a traditional deliberative control architecture can be seen in Figure 6.

Real robot architectures and programming using an SPA deliberative control architecture began in the late 1960s with the Shakey robot at Stanford University [45, 107]. As sensors, the robot was equipped with a camera, a range finder and bump sensors that translated the camera image into an internal world model. A planner took the internal world model and a goal and generated a plan that would achieve this goal. The executor took the plan and sent the actions to the robot. The robot inhabited a set of especially prepared rooms. It navigated from room to room, trying to satisfy a given goal. Many other robotic systems have been built with the traditional AI approach [6, 61, 80, 38, 78], all of which shared the same kind of problems. Planning algo-

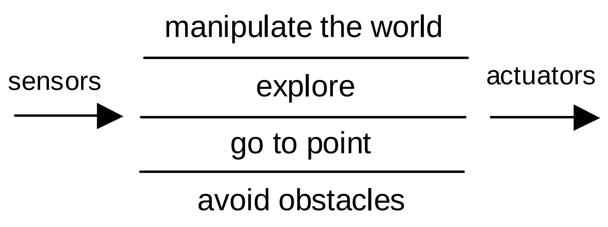

Figure 7: Structure of a behavior-based control architecture [34].

rithms failed with non-trivial solutions and the integration of the world representations was extremely difficult and, as a result, planning in a real world domain took a long time. Also, the execution of a plan without sensing was dangerous in a dynamic world. Only structured and highly predictable environments were proven to be suitable for classical approaches.

In the middle of the 1980s, due to dissatisfaction with the performance of robots in dealing with the real world, a number of scientists began rethinking the general problem of organizing intelligence. Among the most important opponents to the AI approach were Rodney Brooks [30], Rosenschein and Kaelbling [130] and Agre and Chapman [3]. They criticized the symbolic world which Traditional AI used and wanted a more reactive approach with a strong relation between the perceived world and the actions. They implemented these ideas using a network of simple computational elements, which connected sensors to actuators in a distributed manner. There were no central models of the world represented explicitly. The model of the world was the real one as perceived by the sensors at each moment. Leading the new paradigm, Brooks proposed the *Subsumption Architecture*. A subsumption architecture is built from layers of interacting finite-state machines. These finite-state machines were called *Behaviors*, representing the first approach to a new field called *Behavior-based Robotics*. The behavior-based approach used a set of simple parallel behaviors which reacted to the perceived environment proposing the response the robot must make in order to accomplish the behavior (see Figure 7). Whereas SPA robots were slow and tedious, behavior-based systems were fast and reactive. There were no problems with world modeling or real-time processing because they constantly sensed the world and reacted to it.

Since then, behavior-based robotic approaches have been explored in depth. The field attracted researchers from many disciplines such as biologists, neuroscientists, philosophers, linguists, psychologists and, of course, people working with computer science and artificial intelligence, all of whom found practical uses for this approach in their various fields of endeavor. Successful robot applications were built using the behavior-based approach, most of them at MIT [39, 60]. A well known example of behavior-based Robotics is Arkin's motor-control schemas [11], where motor and perceptual schemas were dynamically connected to one another.

Despite the success of the behavior-based models, they soon reached their limits in terms of capabilities. Limitations when trying to undertake long-range missions and the difficulty of optimizing the robot behavior were the most important difficulties encountered. Also, since multiple behaviors can be active at any given time, behavior-based architectures need an *arbitration* mechanism that enables higher-level behaviors to override signals from lower-level behaviors. Therefore, another difficulty has to be solved: how to select the proper behaviors for robustness and efficiency in accomplishing goals? In essence, robots needed to combine the planning capabilities of the classical architectures with the reactivity of the behavior-based architectures, attempting a compromise between bottom-up and top-down methodologies. This evolution was named *Layered Architectures* or *Hybrid Architectures*. As a result, most of today's architectures for robotics follow a hybrid pattern. Usually, a hybrid control architecture is structured in three layers: the reactive layer, the execution control layer and the deliberative layer (see Figure 8). The reactive layer takes care of the real time issues related to the interactions with the environment. It is built by the robot behaviors where sensors and actuators are directly connected. Each behavior can be designed using different techniques, ranging from optimal control to reinforcement learning techniques. The execution control layer interacts between the upper and lower layers, supervising the accomplishment of the tasks. This layer acts as an interface between the numerical reactive and the symbolic planning layers. It is responsible for translating high-level plans into low-level behaviors and for enabling/disabling the behaviors at the appropriate moment with the correct parameters. Also, the execution

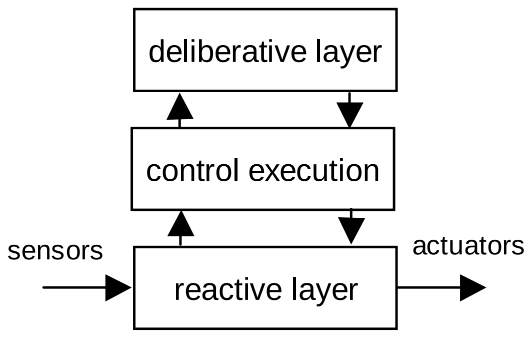

Figure 8: The hybrid control architecture structure [34].

control layer monitors the behaviors being executed and handles any exceptions that may occur. The deliberative layer transforms the mission into a set of tasks which make up a plan. It determines the long-range tasks of the robot based on high-level goals.

One of the first architectures which combined reactivity and deliberation was proposed by James Firby. In his thesis [46], the first integrated three-layer architecture is presented. From there, hybrid architectures have been widely used. One of the best known is Arkin's Autonomous Robot Architecture (AURA) [13], where a navigation planner and a plan sequencer were added to the initial behavior-based motor-control schemas architecture. The Planner-Reactor architecture [83] and the Atlantis [50] used in the Sojourner Mars explorer are well known examples of hybrid architectures.

Nowadays, hybrid architectures represent the basic foundation for control architectures on real robotic systems, but what about behavior programming? This section has described the behaviors as a set of *primitives* belonging to the reactive layer directly responsible for interaction with the environment. Therefore, in terms of AI, there is a key factor that must be taken into serious consideration when designing behaviors: *adaptation*. Intelligence cannot be realized without adaptation. If a robot requires autonomy and robustness it must adapt itself to the environment. The need for adaptation is obvious when dealing with real robotics. The programmer does not know all the parameters of the system and the robot must be able to perform in different and changing environments. At the moment there is no established methodology to develop adaptive behavior-based systems. The next section describes the applicability of RL techniques to design behaviors for control architectures.

3.2 REINFORCEMENT LEARNING BASED BEHAVIORS

RL is a very suitable technique for learning in unknown environments. As has been described in Chapter 2, RL learns from interaction with the environment and according to a scalar reward value. The reward function evaluates the environment state and the last taken action with respect to achieving a particular goal. The mission of an RL algorithm is to find an optimal state/action mapping which maximizes the sum of future rewards whatever the initial state is. The learning of this optimal mapping or policy is also known as the RLP.

The features of RL make this learning theory useful for robotics. There are parts of a robot control system which cannot be implemented without experiments. For example, when implementing a reactive robot behavior, the main strategies can be designed without any real test. However, for the final tuning of the behavior, there will always be parameters which have to be set with real experiments. A dynamics model of the robot and environment could avoid this phase, but it is usually difficult to achieve this model with reliability. RL offers the possibility of learning the behavior in real-time and avoid the tuning of behaviors with experiments. RL automatically interacts with the environment and finds the best mapping for the proposed task, which in this example would be the robot's behavior. The only necessary information which has to be set is an initial parameterization of the policy, the reinforcement function which gives the rewards according to the current state and the past action. It can be said that by using RL the robot designer reduces the effort required to implement the whole behavior, to the effort of designing the reinforcement function. This is a great improvement since the reinforcement function is much simpler and does not contain any dynamics. There is another advantage in that an RL algorithm can be continuously learning and, therefore, the state/action mapping will always correspond to the current environment. This is an important feature in changing environments.

RL theory is usually based on FMDPs. However, in a robotic system, it is usual to measure signals with noise or delays. If these signals are related to the state of the environment, the learning process will be damaged. In these cases, it would be better to consider the environment as a POFMDP (see Section 2.2). The dynamics of the environment is formulated as a POFMDP and the RL algorithms use the properties of these systems to find a solution to the RLP. TD techniques are able to solve the RLP incrementally and without knowing the transition probabilities between the states of the FMDP. In a robotics context, this means that the dynamics existing between the robot and the environment do not have to be known. As far as incremental learning is concerned, TD techniques are able to learn each time a new state is achieved. This property allows the learning to be performed online, which in a real system context like a robot, can be translated to a real-time execution of the learning process. The term *online* is here understood as the property of learning with the data that is currently extracted from the environment and not with historical data.

The combination of RL with a behavior-based system has already been used in many approaches. In some cases, the RL algorithm was used to adapt the coordination system [84, 48, 67, 90]. Moreover, some researchers have used RL to learn the internal structure of the behaviors [132, 158, 154, 140] by mapping the perceived states to control actions. The work presented by Mahadevan [85] demonstrates that breaking down a robot control policy into a set of behaviors simplifies and increases the learning speed. In this dissertation, PG techniques are designed to learn the internal mapping of a reactive behavior.

The main problem of RL when applied to a real system is the *generalization* problem. In a real system, the variables (states or actions) are usually continuous. However, RL theory is based on FMDPs, which uses discrete variables. Classic RL algorithms must be modified to allow continuous states or actions. The next section overviews common methodologies applied to solve the generalization problem from its theoretical point of view.

3.3 generalization methods

As previously stated, the generalization problem appears when dealing with real environments with continuous states and/or actions. In order to successfully adapt continuous variables to finite TD methods, the first solution might be to discretize the continuous space into a finite space. However, in order to solve real robotic tasks, accurate discretizations would require a high number of states or state/action pairs which makes such discretizations impractical for solving real tasks. The need for function approximators that closely *match* or approximate a target function in a task-specific way seems obvious.

The reasons why there are several techniques for solving the generalization problem is because none offer a perfect solution. Some techniques have a higher generalization while others are computationally faster, but their most important feature is their capability to converge into an optimal policy. Convergence proofs have only been demonstrated for algorithms using tabular representations [149, 42, 160, 143]. In addition, in order to maintain stability, the learning procedure must be performed on-policy. This means that off-policy methods using linear function approximators cannot profit from such convergence proof and may diverge [121, 17, 159]. Also, TD methods suffer from convergence problems if the function approximator is not selected properly, as they estimate the VF based on immediate rewards and on the function itself (boostrapping) which means that the VF will always be an approximation of the discrete function. Several methods have been developed to deal with divergence problems [17, 51, 159] either using special update rules or special function approximators to ensure convergence. However, their use is restricted in practice since their convergence times are slow and they have limited generalization capabilities. Despite the lack of convergence proofs, there are many successful algorithms including linear and non-linear approximations with both on and off-policy methodologies [81, 173, 151, 40]. Although it is not intended to be a survey, the next lines show common function approximation techniques, some of which have been used in this thesis to deal with the generalization problem.

Classical approaches for function approximation in RL are based on lookup table methods. However, lookup tables do not scale well with the number of inputs, and huge continuous spaces of robotic tasks make them impractical for the real world. Working in this direction, a very intuitive approach to solving the generalization is offered by *Decision Trees* [105]. This methodology discretizes the entire state or the state/action space, also called *root*, into a space with a different resolution distribution. Those regions of the space requiring more accuracy would be divided into more cells or *leaves* than others requiring less. This variable resolution substantially reduces the number of finite states or state/action pairs. Decision trees highly improve the generalization problem with respect to classical tabular RL methods which use a uniform discretization. In addition, the convergence of the algorithms is sometimes proved and the generalization capability has been shown to be very high [102]. However, in order to select greedy actions, the whole set must be tested, resulting in slow convergence times. Moreover, decision trees always use a finite set of actions and, therefore, the generalization is only carried out in the state space. Several proposals using decision trees to solve the generalization problem can be found. The *G-Learning* algorithm [37] applies a decision tree to represent the QL over a discrete space. In another approach, the *Continuous U-tree* applies ideas similar to the G-Learning to a simulated robot environment, but with a continuous representation of the state. Other proposals show their results in simulated tasks [123, 102].

A popular technique for solving the generalization problem in RL is the Cerebellar Model Articulation Controller (CMAC) [4, 5]. CMAC is a simple linear approximator based on a set of *tiles*. The input space is divided up into hyperrectangles, each of which is associated with a memory cell or tile. The contents of the memory cells are the parameters of *weights*, which are adjusted during training. Usually, more than one quantization of input space is used. Various layers or *tilings* may be superimposed so that any point in input space is associated with a number of tiles, and therefore with a number of memory cells. The output of a CMAC is the algebraic sum of the weights in all the memory cells activated by the input point. The generalization capability of CMAC depends on the number of tilings and the number of tiles within each tiling. The higher the number of tiles the better the resolution and the higher the number of tilings the better the generalization. Since CMAC is a linear approximator, convergence is guaranteed as long as it used with an on-policy algorithm. However, successful applications of both on and off-policy techniques can be found [151, 134, 133, 163]. Drawbacks of CMAC are similar to those presented by decision trees. Although a high generalization capability has been demonstrated by this technique, its application to real robotics shows slow convergence.

Other function estimation methodologies are *Memory-based* techniques, also known as *Instance-based* techniques [15]. Memory-based methods comprise a family of learning algorithms that, instead of performing explicit generalization, compare new problem instances with instances seen in training and stored in memory. Each element of the memory represents a visited state or state/action pair also called a *case*. Each case contains the value of the approximated function. The memory is initialized with zero elements and is dynamically filled according to the visited cases. If a new case which is not contained in the memory appears, neighbor cases are used to compute the value of the new case. There are different techniques to estimate the value of new cases: *nearest neighbors*, *weighted average* and *locally weighted regression* are some of them. The convergence of memory-based methods is not guaranteed, although its generalization capability is very high. Memory-based systems are a kind of *lazy learning* since the generalization beyond the training data is delayed until a query is made to the system. The main advantage gained in employing a lazy learning method is that the target function will be locally approximated and, therefore, lazy learning systems can simultaneously solve multiple problems and deal successfully with changing environments. The main disadvantage of these techniques is that they have high computational requirements. Each time a new approximation is required, the whole process must be performed with all its neighbor cases computation. Also, the space required to store the entire training data set tends to be high. Particularly noisy training data increases the case base unnecessarily because no abstraction is made during the training phase. In addition, as happened with decision trees, in order to find a greedy action, a finite set of actions must be evaluated, which again implies extra computation time. Some examples of their performance on simulated tasks can be found in [113, 93, 109]. Despite their inconvenience for real robotics, some approaches obtained good results [144, 97]. A comparison between memory-based methods and CMAC can be found in [134] where memory-based techniques show better performance.

Continuing along the path of linear approximators, a similar CMAC technique is the *basis function* [152]. The idea is to substitute the binary activation function of each tile by a continuous function. The approximated value is a linear function of the parameter vector to be identified. Corresponding to each state, there is a vector of *features* with the same number of components as the parameter vector. The resultant output of the function is the algebraic sum of the products between each parameter with its correspondent feature. Feature selection becomes critical for a good approximation and the later success of the learning algorithm. Choosing features appropriate to the task is an important way of adding prior domain knowledge to reinforcement learning systems. Intuitively, the features should correspond to the natural features of the task, those for which generalization is most appropriate. If we are valuing geometric objects, for example, we might want to have features for each possible shape, color, size, or function. If we are valuing states of a mobile robot, then we might want to have features for locations, degrees of remaining battery power, recent sonar readings, and so on. The features may be constructed from the states in many different ways, resulting in different kinds of well known basis function approximators: *coarse coding*, *tile coding* [152], *radial basis functions* [120, 118], *barycentric interpolators* [101, 102] and more recently *Proto-value Functions* [86] are some examples. The main advantage of using basis functions is that, if feature selection is appropriate for the task, they can be very efficient in terms of both data and computation for real robotic applications. Over the last few years, much work has been devoted to the topic of developing basis functions techniques. The methods presented in [145] and [122] are essentially heuristics that attempt to exploit the intuition that trajectories taken by the system may lie along a low-dimensional manifold. In [95] the basis functions and the parameters vector are adapted simultaneously, using either gradient descent or the cross-entropy method. In this case, though, the resulting approximator can no longer be considered linear. Also, some practical applications of radial basis functions in robotics can be found in [135, 75].

One of the most important breakthroughs was the introduction of Artificial Neural Networks (ANNs) as function approximators for learning algorithms [58]. An ANN is a parameterized function composed of a set of *neurons* which become activated depending on some input values. These neurons generate an output according to an *activation* *function*. Neuron outputs can be used as inputs for other neurons, forming a multi-layered structure. By combining a set of neurons and using non-linear activation functions, an ANN is able to approximate any non-linear function, making them a very powerful algorithm. As a drawback, the convergence of an RL algorithm using an ANN cannot be guaranteed due to its non-linear nature. In addition, ANNs suffer from *inference* problems [167, 15]. When an ANN-based learning algorithm updates its parameters to change the output value, that change affects the entire space, so a learning step in a particular region of the state space causes a step backward in another place. Solutions to avoid inference point towards using ANNs locally or uniformly updating the space. ANN-based algorithms have been successfully applied to approximate VFs in real robotic tasks [53, 56, 87, 49, 32] and higher generalization capabilities are demonstrated in [173, 27]. In [156] an ANN based system learns to play backgammon. The resultant algorithm was able to play at the same level as the best human players in the world. Successful applications like this motivated the application of function approximators to real robotics. Initial experiments presented in this dissertation use ANN as the function approximator to learn the policy.

Over the last few years, another class of algorithms have received special attention from the RL community: Support Vector Machines (SVMs) [162]. These algorithms have been used mainly for classification tasks. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Intuitively, an SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. Although these methods are known for their application in classification, they can also be used to approximate functions. The main idea is to apply a variation of SVM called SVM-Regression or Support Vector Regression (SVR) [139] to approximate the VF. SVR represents a powerful alternative: in principle unharmed by the dimensionality, trained by solving a well defined optimization problem, and with generalization capabilities presumably superior to local instance-based methods. Even though most of the experimental results are limited to off-line learning [43], some on-line algorithms have appeared recently with promising results [89, 65].

3.4 GPOMDP IN ROBOTICS

The GPOMDP algorithm is an on policy search methodology. Its aim is to obtain a parameterized policy that converges to an optimal by computing gradient approximations of the averaged reward from a single path of a controlled POFMDP. The theoretical aspects of the algorithm have been presented in Section 2.6.2. The GPOMDP algorithm is a TD method and, like all TD methods, the dynamics of the environment does not have to be known. Another important feature of TD algorithms is that the learning process can be performed on-line. The characteristics of the algorithm together with its mathematical simplicity make the GPOMDP approach a good startup to solve the RLP in real robotic tasks. This section analyzes the application of the on-line version of the algorithm in a real system such as a robot.

The algorithm works as follows: having initialized $T > 0$, the parameter vector θ_0 arbitrarily and setting the eligibility trace vector $z_0(\theta) = 0$, the learning procedure will be iterated T times. At every iteration, the system receives the current state from the environment s_t. The algorithm generates a control action a_t according to current stochastic policy $\pi(a_t|s_t, \theta_t)$ and a reward r_{t+1}. The reward function $r(s, a)$ is defined experimentally depending on the RLP. After that, the eligibility trace $z_t(\theta)$ is updated according to the gradient of the current policy with respect to its parameters. The eligibility's decay factor λ is set between $[0, +1)$ with the aim of increasing or decreasing the agent's memory of past actions. The immediate reward received r_{t+1} and the eligibility trace $z_{t+1}(\theta)$ allow us to finally compute the new vector of policy 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 to the problem. The learning rate factor α, tuned experimentally, controls the step size of the iteration process, trying to find a compromise between learning speed and convergence guarantees.

From the different generalization techniques described in Section 3.3, two different function approximators have

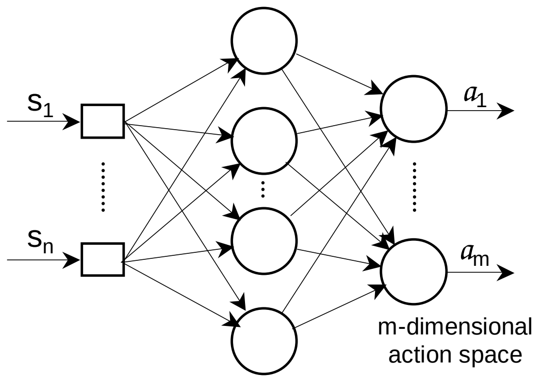

Figure 9: Implementation of a simple policy with an ANN.

been selected to represent the policy: ANNs and barycentric interpolators.

3.4.1 *Policy Approximation with ANN*

Together with the GPOMDP algorithm, a parameterized ANN function is one of the approximators selected to represent a stochastic policy $\pi(a_t|s_t, w_t)$. Its weights w_t represent the policy parameters to be updated at every iteration step. Advantages and drawbacks of various function approximators have been described in Section 3.3. The main reason for choosing an ANN as one of the approximators for these initial experiments is for its excellent ability to approximate any nonlinear function in comparison with other function approximators. Also, an ANN is easy to compute and the required number of parameters is very small, making it suitable for application with the GPOMDP approach. An ANN is a function approximator able to approximate a mathematical function which has a set of inputs and outputs. The input and output variables are real numbers and the approximated functions can be non-linear, according to the features of the ANN. ANN were inspired by the real neurons found in the human brain, although a simpler representation is used. As stated in Section 3.3, the basic theory of ANN was widely studied during the 1980s but there are still many active research topics.

The ANN model used to represent the policy in this first approach is depicted in Figure 9. The state vector is driven directly as the ANN input. The output of the network is the number of continuous variables which make up the action space. As can be seen, neurons are grouped in different *layers*. The first layer uses the state vector as neuron inputs $\{s_1, ..., s_n\}$. This set of inputs is also called *input layer*, although it is not a layer of neurons. The second and consecutive layers use as neuron inputs the neuron outputs from the preceding layer. Finally, the last layer of the ANN is the *output layer*, where each neuron generates an output of the network as actions $\{a_1, ..., a_m\}$ of the approximated policy. All the neuron layers preceding the output layer are also called *hidden layers* since the neuron output values are not seen from the outside nor are they significant.

Going deeper into the design of the network, Figure 10 takes a closer look into a single neuron. The neuron j located in the layer l has a set of inputs $\{y_1^{l-1}, y_2^{l-1}, ..., y_p^{l-1}\}$ and one output y_j^l. The value of this output depends on these inputs, on a set of weights $\{w_{j1}^l, w_{j2}^l, ..., w_{jp}^l\}$ and on an *activation function* φ^l. In the first computation, the induced *local field* v_j^l of the neuron j is calculated by adding the products of each input y_i^{l-1} by its corresponding weight w_{ij}^l. An extra input y_0^{l-1} is added to the computation of v_j^l. This input is called the *bias* term and has a constant value equal to 1. By adjusting the weight w_{j0}^l, the neuron j can be activated even if all the inputs are equal to 0. The local field v_j^l is then used to compute the output of the neuron $y_j^l = \varphi^l v_j^l$. The activation function has a very important role in learning efficiency and capability. The learning process of the ANN consists of adapting the parameters or weights of the network until the output is equal to a desired response. The GPOMDP algorithm has the goal of indicating the procedure to modify the values of these parameters.

The next lines will describe the parameter update procedure carried out by the GPOMDP algorithm for the particular case of an ANN as the policy approximator. Once the parameters of the ANN are initialized, the network receives an observation of the state as input s_t and gives a control action as output a_t. Then, the algorithm is driven to another state s_{t+1} and will receive a reward associated with this new state r_{t+1}. The first step in the parameter update procedure is to compute the gradient of the policy with respect to each parameter. As defined in Equation 3.1, in the ANN context, the gradient of the policy with respect to each parameter is

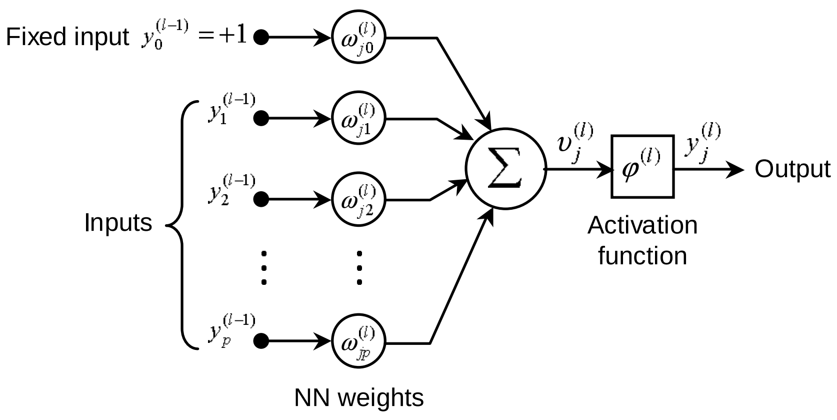

Figure 10: Diagram of a single neuron j located at layer l.

equal to the *local gradient* δ_t associated with a single neuron multiplied by the input to the neuron y_t.

$$\frac{d\pi(a_t|s_t, w_t)}{dw_t} = \delta_t y_t. \tag{3.1}$$

In order to calculate the local gradient of all the neurons in a particular ANN, the local gradient of the neurons at the output layer must be computed first and then propagated to the rest of the neurons in the hidden layers by means of *backpropagation*. For the local neuron j located in the output layer, we may express its local gradient as

$$\delta_j^o = e_j \varphi'(o_j). \tag{3.2}$$

Here e_j is the error in the output of neuron j, $\varphi'(o_j)$ corresponds to the derivative of the activation function associated with that neuron and o_j is the function signal in the output for that neuron. Therefore, for neuron j located in a hidden layer, the local gradient is defined as follows

$$\delta_j^h = \varphi'(o_j) \sum_k \delta_k w_{kj}. \tag{3.3}$$

As can be seen in Figure 11, when computing the gradient of a neuron located in a hidden layer, the previously obtained gradient of the following layers must be back propagated. $\varphi'(o_j)$ is the derivative of the activation function associated with that neuron and the summation term includes the different gradients of the following neurons

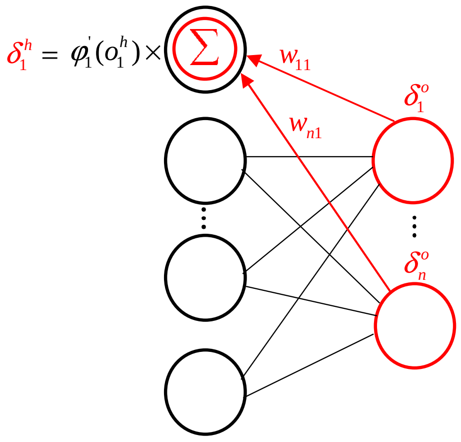

Figure 11: Local gradient computation for a hidden layer neuron.

back propagated by multiplying each gradient δ_k with its correspondent weight w_{kj}.

With the local gradients of all the neurons computed, the gradients of the policy with respect to each weight in the ANN can be obtained. As described in the GPOMDP procedures of Algorithm 4, eligibility traces can be calculated following Equation 3.4. Finally, the parameter vector is updated following Equation 3.5, i. e.,

$$z_{t+1}(w) = \lambda z_t(w) + \delta_t y_t, \tag{3.4}$$

$$w_{t+1} = w_t + \alpha r_{t+1} z_{t+1}(w). \tag{3.5}$$

Here the vector of parameters w represents the network weights to be updated and r_{t+1} is the reward given to the agent at every time step. The learning rate α controls the step size of the iteration process.

3.4.2 *Policy Approximation with Barycentric Interpolators*

A parameterized basis function represented by a barycentric interpolator is another approximator chosen to represent the policy $\pi(a_t|s_t, \theta_t)$ together with the GPOMDP approach. Barycentric interpolators, described in Section 3.3, are a specific class of basis function approximators. These basis functions are a popular representation for VFs because they provide a natural mechanism for variable resolution discretization of the function and the barycentric co-ordinates

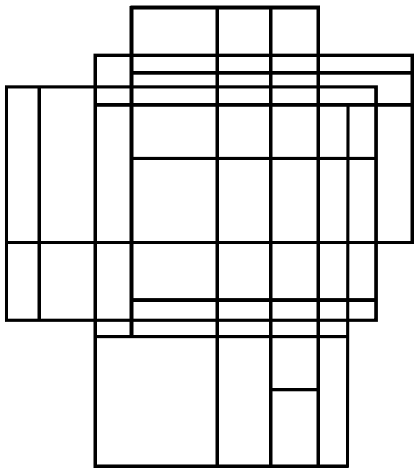

Figure 12: An N-dimension rectangular mesh. The mesh elements need not be regular, however, the boundary can vary in extent.

allow the interpolators to be used directly by value iteration algorithms. Another advantage in choosing linear function approximators is their convergence guarantees when used together with an on-policy method such as the GPOMDP algorithm. Barycentric interpolators apply an interpolation process based on a finite set of discrete points that form a mesh. This mesh does not need to be regular, but the method outlined here assumes that the state space is divided into a set of rectangular boxes. An example of an appropriate mesh is shown in Figure 12. In practice, we will often start with a grid that is iteratively remeshed and which would look more like Figure 13. Getting the barycentric interpolation itself to work only requires a mesh of the type shown in Figure 12, though. The key is that any particular point is enclosed in a rectangular box that can be defined by the 2^N nodes in our N-dimensional state space.

As shown in Figure 14, ξ_i being a set of nodes distributed in a rectangular mesh for any state s. Once it is enclosed in a particular box $(\xi_1, ..., \xi_4)$ of the mesh, the state s becomes the *barycenter* inside this box with positive coefficients $p(s|\xi_i)$ of sum 1 called the *barycentric coordinates* and, therefore,

$$s = \sum_{i=1..4} p(s|\xi_i)\xi_i. \tag{3.6}$$

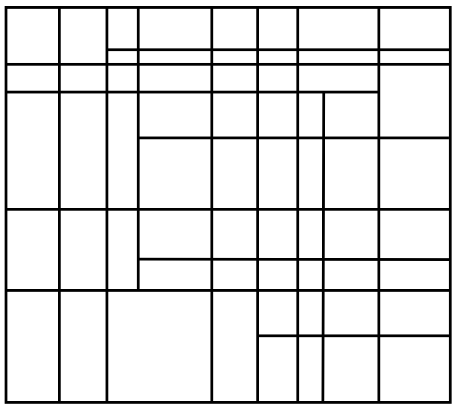

Figure 13: A more typical-looking mesh, where an initial grid has been refined through iterative splitting of some of the elements.

Thus, $V(\xi_i)$ is set as the value of the function at the nodes previously defined as ξ_i. Having defined the value of the function at the nodes of the mesh, Equation 3.7 shows how to compute the value of the function at state s, called the *barycentric interpolator* of state s and defined as $V(s)$. The value of state s is calculated as a function of the value $V(\xi_i)$ at the nodes of the box in which the state is enclosed and its barycentric coordinates $p(s|\xi_i)$. As depicted in Figure 15, we give

$$V(s) = \sum_{i=1..4} p(s|\xi_i)V(\xi_i). \tag{3.7}$$

The policy is approximated using a barycentric interpolator function where the nodes of the mesh $V(\xi_i)$ represent the policy parameters θ to be updated at each learning iteration. Therefore, given an input state s_t, the policy will compute a control action $V(s_t) = a_t$. The parameters of the approximator are updated following the procedures stated in Algorithm 4. As defined in Equation 3.8, in the barycentric approximator context, the gradient of the policy with respect to each parameter is equal to the derivative of the approximator with respect to the parameter multiplied by an error function e_t, hence,

$$\frac{d\pi(a_t|s_t, \theta_t)}{d\theta_t} = \frac{dV(s_t)}{dV(\xi_i)} e_t. \tag{3.8}$$

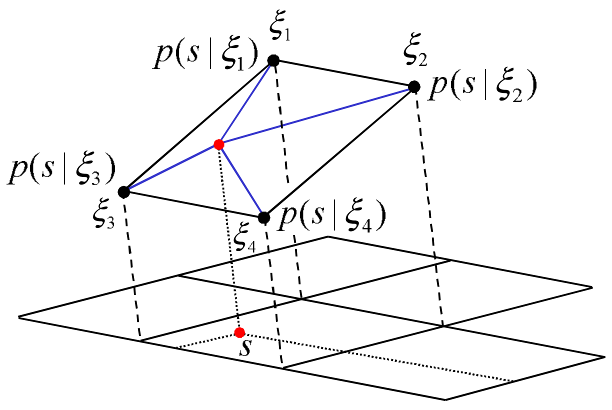

Figure 14: Graphic representation of the *barycentric coordinates* given a state s in a 2 dimensional mesh case.

Here the error is given by

$$e_t = V(s_t)_{\text{desired}} - V(s_t). \tag{3.9}$$

The eligibility trace z_{t+1} is updated following Equation 3.10. Eligibility's decay factor λ controls the memory on past actions. The gradient of the policy with respect to a particular parameter contains the barycentric coordinate of the parameter $p(s|\xi_i)$,

$$z_{t+1} = \lambda z_t + p(s_t|\xi_i)e_t. \tag{3.10}$$

Finally, the new parameter vector is given by

$$V(\xi_i)_{t+1} = V(\xi_i)_t + \alpha r_{t+1} z_{t+1}. \tag{3.11}$$

The vector $V(\xi_i)$ represents the policy parameters to be updated, r_{t+1} is the reward given to the agent at every time step and α as the learning rate of the algorithm.

3.5 NATURAL ACTOR-CRITIC IN ROBOTICS

The experimental results obtained with both approximations described in Section 3.4 showed poor performance of the algorithm. Although the GPOMDP approach is able to converge to an optimal or near optimal policy, results suggest that the application of a *pure* PG algorithm in a real task is quite slow and rigid. Fast convergence and adaptation to an unknown changing environment is a must when

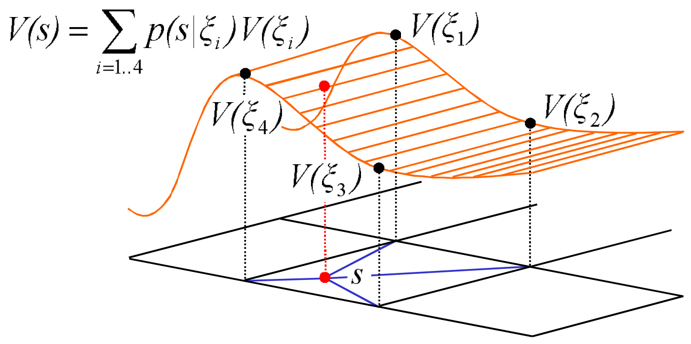

Figure 15: Calculation of the function approximator output given a particular state s.

dealing with real applications and a PG seems not able to do so alone. With the objective of finding a faster algorithm, the attention of this thesis moves to AC methodologies. As discussed in Section 2.8, AC methods try to combine the advantages of PG algorithms with VF methods. The actor's PG gives convergence guarantees while the critic's VF reduces variance of the policy update improving the convergence rate of the algorithm. Also, AC methods are on-policy algorithms and, therefore, the learning procedure benefits from convergence proofs. The main features of AC methods seem to have what real robotics is looking for.

The theoretical aspects of the algorithm have been presented in Section 2.8.4. The algorithm is divided into two main blocks, one concerning the critic and another the actor. The critic is represented by a VF $V^{\pi}(s)$ which is approximated by a linear function parameterization represented as a combination of the parameter vector v and a particularly designed basis function $\phi(s)$, the whole expression being $V^{\pi}(s) = \phi(s)v$. On the other hand, the actor's policy is specified by a normal distribution $\pi(a|s) = \mathcal{N}(a|Ks, \sigma^2)$. The mean μ of the actor's policy distribution is defined as a parameterized linear function of the form $a = Ks$. The variance of the distribution is described as $\sigma = 0.1 + 1/(1 + \exp(\eta))$. Therefore, the whole set of the actor's parameters is represented by the vector $\theta = [K, \eta]$. The actor's policy derivative has the form $\frac{d \log \pi(a|s,\theta)}{d\theta}$ and, as detailed in Section 2.6.4, in order to obtain the natural gradient, this function is inserted into a parameterized compatible function approximation, $f(s,a)^{\pi}_{w} = \frac{d \log \pi(a|s,\theta)}{d\theta} w$, with the parameter vector w as the true gradient .

At every iteration, action a_t is drawn from the current policy π_t generating a new state s_{t+1} and reward r_t. After updating the basis function and the critic's statistics by means of LSTD-Q(λ), the VF parameters ν and the natural gradient w are obtained. The actor's policy parameters are updated only if the angle between two consecutive natural gradients is small enough compared to an ϵ term. The learning rate of the update is controlled by the α parameter. Next, the critic has to forget part of its accumulated statistics using a forgetting factor, $\beta \in [0, 1]$. The current policy is directly modified by the new parameters becoming the new policy to be followed in the next iteration, getting closer to a final policy that represents a correct solution to the problem.

3.6 METHODS TO SPEEDING UP REINFORCEMENT LEARNING

Speeding up RL algorithms in real continuous high dimensional domains represents a key factor. The idea of providing initial high-level information to the agent is a topic that has received significant attention in the real robotic field over the last decade. The main objective of these techniques is to make learning faster in contrast to tedious reinforcement learning methods or trials-and-error learning. Also, it is expected that the methods, being user-friendly, would enhance the application of robots in human daily environments. This section briefly describes some common techniques for enhancing and accelerating learning in real robotic applications.

One form of speeding up the learning process is to have the robot learn a particular task by *watching* the task being performed by a human or an expert system. The approach is known as *Learning From Demonstration*, *Programming From Demonstration*, *Imitation Learning* or simply *Teaching* [28, 15, 81]. Teaching plays a critical role in human learning. The main idea is that teaching can shorten the learning process and even turn intractable learning tasks into tractable. If learning is resumed as a search problem, the teacher gives external guidance for this search. For this kind of learning, the robot can try to repeat the actions taken by the teacher or learn how the teacher acts and reacts to the different situations encountered, finally building its own policy. Although these kinds of techniques encompass a wide range of algorithms with thier own procedures, a general teaching process overview may be one like this: The teacher shows the learning agent how an instance of the target task can be achieved from some initial state. The sequence of actions, state transitions and rewards are recorded as a *lesson*. Several taught lessons can be collected and repeatedly replayed. Like experienced lessons, taught lessons should be replayed selectively; in other words, only policy actions are replayed. But if the taught actions are known to be optimal, all of them can be replayed all the time. The term lesson refers to both taught and experienced lessons. It is unnecessary for the teacher to demonstrate only optimal solutions in order for the agent to learn an optimal policy. In fact, the agent can learn from both positive and negative examples. This is an important property as it makes teaching techniques different from *supervised learning* approaches [100, 119]. Teaching is useful for two reasons: First, teaching can direct the agent to first explore the promising part of the search space which contains the goal states. This is important when the search space is large and a thorough search is infeasible. Second, teaching can help the agent to avoid being stuck in local maxima. The idea of providing initial high-level information to the agent has achieved great success in several applications. In [57], an outdoor mobile robot learns to avoid collisions by observing a human driver operate a vehicle equipped with sensors that continuously produce a map of the local environment. The work presented in [117] shows how imitation learning can be used to extract an initial policy to learn a control task where an anthropomorphic arm tries to hit a ball. First, a human teacher gives the agent an insight into the task by helping it hit the ball. Results show that this initial policy is not good enough to fulfil the task. Therefore, in a second phase, an NAC algorithm is applied to improve the policy obtaining a better policy which successfully accomplishes the task.

Other approaches to accelerate the learning process with real robots use *Supplied Control Policies* to feed prior knowledge to the agent [144]. The main idea is to split the learning into two phases. In the first phase, the robot is being controlled by a supplied control policy. This initial control policy can be represented either by a classic controller or by a human controlling the robot. During this period, the agent passively watches state transitions, actions and rewards that the supplied control policy generates and uses them to update its policy. The supplied control policy exposes the agent to those areas of the space-state where the reward is not zero, accelerating the learning process. Differing from previously described techniques, the agent does not mimic the trajectories generated by the supplied policy, it just uses them to update its policy. Also, these trajectories are not generated directly, as in teaching, but as a result of interaction with the environment. When the learned policy is considered good enough, in a second phase, the agent takes control of the robot and continues the learning by normal interaction with the environment. Results presented in [147] show the viability of supplied control policies for speeding up learning algorithms. In that paper, a mobile robot successfully learns control policies for two simple tasks: obstacle avoidance and corridor following.

Another way of reusing past experiences, which was investigated in the DYNA architecture [150], is to use experiences to build an *action model* and use it for planning and learning. An action model can be understood as a computer simulator which mimics the behaviors of the environment for a particular RLP. The action model contains the dynamics of the environment together with its reward function. Instead of direct interaction with the environment, the agent can experience the consequences of actions without physically being in the real world. As a result, the robot will learn faster as the iteration steps with the simulator would be faster than those with the real world and more importantly, fewer mistakes will be made when the robot starts running in the real world with a previously trained policy. Approximate policies learned in simulation represent a good startup for the real learning process, meaning that the agent will face the challenge of the real world with some initial knowledge of the environment, avoiding initial gradient plateaus and local maxima dead ends [81]. The help of computer simulators has been successfully applied in [16] where a PG algorithm designed to control a RC-helicopter is first trained off-line by means of a computer simulator. Once the agent's policy is considered to be *safe* enough it is transferred to the real vehicle where the learning process continues. Also, in [77], a technique called *Time Hopping* is proposed for speeding up reinforcement learning algorithms in simulation. Experiments on a simulated biped crawling robot confirm that this technique can greatly accelerate the learning process.

The success of such techniques is closely related to the accuracy of the simulation, being almost impossible to apply to real complex tasks where a model of the environment can not be obtained. In order to have a simulation phase that really represents an advantage, a good model has to be provided. For the particular RLP proposed in this thesis, most of the model identification work is based on the dynamic equations of motion derived from the Newtonian or Lagrangian mechanics [47], which are characterized by a set of unknown parameters. The application of system identification techniques to underwater vehicles is concerned with the estimation, on the basis of experimental measurements, of a number of parameters or of hydrodynamic derivatives that characterize the vehicle's dynamics [2]. Such measurements, collected during full-scale trials by the on-board sensors [33], or during captive testing in a planar motion mechanism [108], are processed by a parameter estimation routine [82]. Several methods have been proposed in the literature for this purpose like the use of genetic algorithms and simulated annealing [157]. Nevertheless, since the dynamic equation of motion can be formulated as an equation linear in the vector of unknown parameters, LS methods can be easily used.

The final proposal of this thesis is the application in a real underwater robotic task of RL-based behavior in a two step learning process. The RL method chosen is the NAC algorithm. For this purpose, the learning algorithm is first trained in a simulated environment where it can quickly build an initial policy. An approximated model of the environment emulates a robot with the same number of DoFs of the real one. Once the simulated results are accurate enough and the algorithm has acquired enough knowledge from the simulation to build an approximated policy, in a second step the policy is transferred to the real robot to continue the learning process on-line in a real environment. Since the simulated model and environment are just approximations, the algorithm will have to adapt to the real world. An important idea is that, once the algorithm converges in the real environment, the learning system will continue working forever, being able to adapt to any future change. The next chapter accurately describes the identification procedures carried out to obtain a model of the environment, which is applied to learn an initial policy during the first step of the learning process.

4 MODEL IDENTIFICATION FOR UNDERWATER VEHICLES

In order to decrease learning periods, the previous chapter proposed a learning methodology where a model of the environment is needed for the initial learning phase: the learning in simulation step. For this purpose, an approximated model of the underwater vehicle that will be used to test the learning methodologies proposed in the previous chapter must be computed. The following lines describe the dynamics equations of motion for an underwater vehicle. After that, some common assumptions are detailed to adapt the general equation to our particular vehicle and, therefore, simplify the problem. Finally, an LS identification method is proposed and applied to identify the vehicle's parameters. Some results regarding the performance of the identified model are given at the end of this chapter.

4.1 THE AUTONOMOUS UNDERWATER VEHICLE MODEL

As described in [47], the non-linear hydrodynamic equation of motion for an underwater vehicle with 6 DoFs can be conveniently expressed in the fixed body frame $\{B\}$ as

$$\tau^B + G(\eta)^B - D(\upsilon^B)\upsilon^B + \tau_p^B = (M^B)\dot{\upsilon}^B + C^B(\upsilon^B)\upsilon^B, \quad (4.1)$$

where we have the following variables:

- υ^B and $\dot{\upsilon}^B$ are the velocity and acceleration vectors.
- $\eta = (\phi, \theta, \psi)^T$ is the position and attitude vector.
- τ^B is the resultant force-moment vector exerted by thrusters.
- $G(\eta)^B$ is a vector of gravitational forces and moments.
- $D(\upsilon^B)$ corresponds to the linear and quadratic damping matrixes.
- M^B corresponds to the inertia matrix (including added mass).

- C^B represents the matrix of Coriolis and centripetal terms (including added mass).
- τ_p^B is a vector of unmodelled perturbations.

Equation 4.1 relates the forces exerted on the AUV with the acceleration and the velocity experienced by the vehicle. In other words, if the forces acting on the robot are known, Equation 4.1 allows for the estimation of the robot's acceleration, velocity and position.

4.1.1 *Common Simplifications*

Equation 4.1 represents a complex non-linear model with couplings among the different degrees of freedom dependent on a large set of physical parameters (the robot's mass and inertia, the linear and nonlinear friction coefficients, thrust coefficients, etc...). Nevertheless, after some simplifications, it is possible to carry out identification experiments to find the most relevant coefficients to obtain an approximate model:

- The fixed body frame is located at the vehicle's center of gravity ($(x_G \ y_G \ z_G)^B = (0 \ 0 \ 0)^B$). In the same way, the fixed body frame axes coincide with the principal axes of inertia or the longitudinal, lateral and normal symmetry axes of the vehicle, so the origin of the fixed body coordinate system can then be chosen so that the inertia tensor will be diagonal, that is $I_{3x3} = diag(I_{xx}, I_{yy}, I_{zz})$. Moreover, the vehicle is considered a neutrally buoyant underwater vehicle.
- For an underwater vehicle with 6 DoF, damping forces are highly nonlinear and coupled. In practical applications, it is difficult to determine off-diagonal terms in the general expression of the damping matrix $D(\upsilon^B)$. Since the vehicle will perform non-coupled movements, $D(\upsilon^B)$ can be assumed to be diagonal, with linear and quadratic damping terms on the diagonal. Forward and backward damping coefficients are considered equal due to the square shape of the vehicle which gives it three planes of symmetry.
- The vehicle moves at low speed and, due to the symmetry condition mentioned before, the contribution

from off-diagonal elements in M^B can be ignored. Also, decoupled movement assumption discards Coriolis terms.

Let us consider hereafter, the main forces acting on a AUV.

4.1.2 *Forces Exerted by Thrusters*

When a thruster propelled vehicle is considered, a thruster model has to be used for computing the axial force exerted by the propeller. Although the thruster has its own dynamics [169], when the propeller is speed-controlled the servo velocity loop can be designed to have a much smaller time constant than the time constant of the whole vehicle. Then, the thruster dynamics can be ignored and a steady-state model can be used. The thrust is better described by a bilinear model. A bilinear model can be used to estimate the thrust exerted by the thruster i as a function of the angular speed of the propeller (ω_i) and the linear velocity of the vehicle (υ_i) in the thruster direction, as shown by

$$\tau_i = b_{i,1}|\omega_i|\omega_i - b_{i,2}|\omega_i|\upsilon_i. \tag{4.2}$$

Nevertheless, for vehicles moving at slow speeds, the affine thruster model can be used (see Equation 4.3). This model is the same as the previous one in the special case of $\upsilon_i = 0$. The thruster constant parameter $b_{i,1}$ has been renamed to $C_{T,i}$, hence,

$$\tau_i = C_{T,i}|\omega_i|\omega_i. \tag{4.3}$$

If the thruster has not been specifically designed for symmetric thrust, different C_T must be expected for each direction as stated by

$$\tau_i = \begin{cases} C_{Tf,i}|\omega_i|\omega_i & \text{if} \quad \omega_i > 0, \\ C_{Tb,i}|\omega_i|\omega_i & \text{if} \quad \omega_i < 0. \end{cases} \tag{4.4}$$

Moreover, as stated in [33], a non-neglectable efficiency loss due to the interactions between the propeller and the hull is experienced when the thruster is mounted and operated in the AUV. For this reason, $C_{Tf,i}$ and $C_{Tb,i}$ must be identified while mounted on the vehicle. For the model identification of Ictinieu AUV, the steady-state thruster affine model will be used for computing the force exerted by the thrusters from the rotational speed of the propellers. Once the thrust exerted by each thruster is known, a thruster configuration matrix B depending on the thruster coefficients as well as on the geometry of their location can be defined as detailed in

$$\begin{gathered}\tau^B = Bu, \\ \begin{bmatrix} X \\ Y \\ Z \\ K \\ M \\ N \end{bmatrix}^B = B \begin{bmatrix} |\omega_1|\omega_1 \\ \dots \\ |\omega_n|\omega_n \end{bmatrix},\end{gathered} \tag{4.5}$$

where $(X, Y, Z, K, M, N)^T$ is the total force and moment vector and n represents the number of thrusters of the vehicle. This expression allows for computing the resultant force and torque exerted by all the thrusters.

4.1.3 *Restoring Forces and Moments*

The vector of gravitational forces and moments $G(\eta)^B$ is the resultant vector of combining gravity with the buoyancy forces on the vehicle. The gravity (F_W^B) and buoyancy (F_B^B) forces are constant forces in earth coordinate systems if the vehicle does not change its weight or volume. In such cases, these forces have only to be referenced to the vehicle's coordinate system depending on its orientation. The total gravity force magnitude (W) of a submerged body and the total buoyancy force magnitude (B) can be expressed as

$$\begin{aligned} W &= mg, \\ B &= \rho V g. \end{aligned} \tag{4.6}$$

Here m represents the mass of the vehicle including the water in free floating space, V is the volume of liquid displaced by the vehicle, g is the acceleration of gravity (positive downwards) and ρ is the liquid density. As stated in

Section 4.1.1, the fixed body frame is located at the vehicle's center of gravity $((x_G\ y_G\ z_G)^B = (0\ 0\ 0)^B)$ with the center of buoyancy defined as $(x_b\ y_b\ z_b)^B$, then the vector $G(\eta)^B$ can be computed as

$$G(\eta)^B = F_W^B + F_B^B,$$

$$G(\eta)^B = \begin{bmatrix} -Ws\theta \\ Wc\theta s\phi \\ Wc\theta c\phi \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} Bs\theta \\ -Bc\theta s\phi \\ -Bc\theta s\phi \\ Bz_b c\theta s\phi - By_b c\theta c\phi \\ Bx_b c\theta c\phi + Bz_b s\theta \\ -Bx_b c\theta s\phi - By_b s\theta \end{bmatrix}. \tag{4.7}$$

s and c being the sine and the cosine respectively, $\eta = (\phi, \theta, \psi)^T$ the vehicle's Roll, Pitch and Yaw angles, F_B^B the buoyancy forces and moments and F_W^B the gravity forces and moments. Considering that a neutrally buoyant underwater vehicle satisfies $W = B$, combining expressions of Equation 4.7 yields

$$G(\eta)^B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ Bz_b c\theta s\phi - By_b c\theta c\phi \\ Bx_b c\theta c\phi + Bz_b s\theta \\ -Bx_b c\theta s\phi - By_b s\theta \end{bmatrix}. \tag{4.8}$$

4.1.4 *Hydrodynamic Damping*

The damping forces are drag forces depending on the vehicle's speed. As shown in Equation 4.9, the two main components are the linear damping (D_L) and the quadratic damping (D_Q). Linear damping refers to the skin friction due to laminar boundary layers while the quadratic form contains skin friction due to turbulent boundary layers:

$$D(\upsilon^B) = D_L + D_Q. \tag{4.9}$$

Simplifications proposed in Section 4.1.1 suggest that the vehicle performs non-coupled movements, so the damping matrixes can be assumed diagonal, with linear and quadratic damping terms on the diagonal. Hence, the equation that describes these forces is expressed as follows;

$$D(\upsilon^B) = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_w & 0 & 0 & 0 \\ 0 & 0 & 0 & K_p & 0 & 0 \\ 0 & 0 & 0 & 0 & M_q & 0 \\ 0 & 0 & 0 & 0 & 0 & N_r \end{bmatrix} +$$

$$\begin{bmatrix} X_{u|u|}|u| & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{v|v|}|v| & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{w|w|}|w| & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{p|p|}|p| & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{q|q|}|q| & 0 \\ 0 & 0 & 0 & 0 & 0 & N_{r|r|}|r| \end{bmatrix}. \tag{4.10}$$

Here $(X_u\ Y_v\ Z_w\ K_p\ M_q\ N_r)$ correspond to the linear terms, $(X_{u|u|}\ Y_{v|v|}\ Z_{w|w|}\ K_{p|p|}\ M_{q|q|}\ N_{r|r|})$ are the quadratic terms and $(u\ v\ w\ p\ q\ r)$ is the linear and angular velocity vector. Adding matrixes of Equation 4.10 yields

$$\begin{aligned} D(\upsilon^B) = \mathrm{diag}(&X_u + X_{u|u|}|u|, Y_v + Y_{v|v|}|v|, Z_w + Z_{w|w|}|w|, \\ &K_p + K_{p|p|}|p|, M_q + M_{q|q|}|q|, N_r + N_{r|r|}|r|). \end{aligned} \tag{4.11}$$

4.1.5 *Added Mass, Inertia and Coriolis Terms*

As described in Equation 4.1, both the inertia and Coriolis matrixes include an added mass term. Added mass refers to the inertia added to a system because an accelerating or decelerating body must displace some volume of surrounding liquid as it moves through it, since the object and liquid cannot occupy the same physical space simultaneously. For simplicity, this has been modeled as some volume of liquid moving with the object, though in reality all the liquid will be accelerated to various degrees.

To fit with rigid body dynamics, the added mass term is divided in two terms: the added mass inertia matrix (M_A^B)

and the added mass matrix of hydrodynamic Coriolis and centripetal terms (C_A^B). Therefore, as shown in Equation 4.12, both matrixes are combined with the rigid-body inertia matrix (M_{RB}^B) and the rigid-body matrix of hydrodynamic Coriolis and centripetal terms (C_{RB}^B), given by

$$\begin{aligned} M^B &= M_{RB}^B + M_A^B, \\ C^B &= C_{RB}^B + C_A^B. \end{aligned} \tag{4.12}$$

The parametrization of the inertia matrix M^B of Equation 4.12 is described by

$$M^B = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_{xx} & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yx} & I_{yy} & -I_{yz} \\ -my_G & mx_G & 0 & -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} + \begin{bmatrix} -X_{\dot{u}} & -X_{\dot{v}} & -X_{\dot{w}} & -X_{\dot{p}} & -X_{\dot{q}} & -X_{\dot{r}} \\ -Y_{\dot{u}} & -Y_{\dot{v}} & -Y_{\dot{w}} & -Y_{\dot{p}} & -Y_{\dot{q}} & -Y_{\dot{r}} \\ -Z_{\dot{u}} & -Z_{\dot{v}} & -Z_{\dot{w}} & -Z_{\dot{p}} & -Z_{\dot{q}} & -Z_{\dot{r}} \\ -K_{\dot{u}} & -K_{\dot{v}} & -K_{\dot{w}} & -K_{\dot{p}} & -K_{\dot{q}} & -K_{\dot{r}} \\ -M_{\dot{u}} & -M_{\dot{v}} & -M_{\dot{w}} & -M_{\dot{p}} & -M_{\dot{q}} & -M_{\dot{r}} \\ -N_{\dot{u}} & -N_{\dot{v}} & -N_{\dot{w}} & -N_{\dot{p}} & -N_{\dot{q}} & -N_{\dot{r}} \end{bmatrix}. \tag{4.13}$$

Here $(x_G\ y_G\ z_G)^B$ are the coordinates of the vehicle's center of gravity, I_{3x3} is the inertia tensor with respect to the origin of the fixed body frame $\{B\}$, m is the mass of the vehicle and $A_{ij} = \{M_A^B\}_{ij}$ are the hydrodynamic added mass derivative terms. Due to the simplifications carried out in Section 4.1.1, simple expressions of M_{RB} and M_A are

$$M^B = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz} \end{bmatrix} +$$

$$\begin{bmatrix} -X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & -Y_{\dot{v}} & 0 & 0 & 0 & 0 \\ 0 & 0 & -Z_{\dot{w}} & 0 & 0 & 0 \\ 0 & 0 & 0 & -K_{\dot{p}} & 0 & 0 \\ 0 & 0 & 0 & 0 & -M_{\dot{q}} & 0 \\ 0 & 0 & 0 & 0 & 0 & -N_{\dot{r}} \end{bmatrix}, \tag{4.14}$$

$$M^B = diag(m - X_{\dot{u}}, m - Y_{\dot{v}}, m - Z_{\dot{w}}, \\ I_{xx} - K_{\dot{p}}, I_{yy} - M_{\dot{q}}, I_{zz} - N_{\dot{r}}).$$

The parametrization of the Coriolis matrix (C^B) of Equation 4.12 is detailed in Equation 4.15. For a rigid-body moving through an ideal liquid, both the rigid-body Coriolis and centripetal matrix (C^B_{RB}) and the added mass matrix of hydrodynamic Coriolis and centripetal terms (C_A^B), can be parameterized as

$$C^B = \begin{bmatrix} 0 & 0 & 0 & a_1 & -a_2 & -a_3 \\ 0 & 0 & 0 & -a_4 & a_5 & -a_6 \\ 0 & 0 & 0 & -a_7 & -a_8 & a_9 \\ -a_1 & a_4 & a_7 & a_{10} & -a_{11} & a_{12} \\ a_2 & -a_5 & a_8 & a_{11} & a_{10} & -a_{13} \\ a_3 & a_6 & -a_9 & -a_{12} & a_{13} & a_{10} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & -b_3 & b_2 \\ 0 & 0 & 0 & b_3 & 0 & -b_1 \\ 0 & 0 & 0 & -b_2 & b_1 & 0 \\ 0 & -b_3 & b_2 & 0 & -b_6 & b_5 \\ b_3 & 0 & -b_1 & b_6 & 0 & -b_4 \\ -b_2 & b_1 & 0 & -b_5 & b_4 & 0 \end{bmatrix}. \tag{4.15}$$

Where a_i terms of the C_{RB}^B are given by

$$\begin{aligned} a_1 &= m(y_G q + z_G r), & a_7 &= m(z_G p - v), \\ a_2 &= m(x_G q - w), & a_8 &= m(z_G q + u), \\ a_3 &= m(x_G r + v), & a_9 &= m(x_G p - y_G q), \\ a_4 &= m(y_G p + w), & a_{10} &= 0, \\ a_5 &= m(z_G r + x_G p), & a_{11} &= I_{yz} q + I_{xz} p - I_z r, \\ a_6 &= m(y_G r - u), & a_{12} &= I_{yz} r + I_{xy} p - I_y q, \\ a_{13} &= I_{xz} r + I_{xy} q - I_x p, \end{aligned}$$

and b_i terms of the C_A^B are defined as

$$\begin{aligned} b_3 &= X_{\dot w} u + Y_{\dot w} v + Z_{\dot w} w + Z_{\dot p} p + Z_{\dot q} q + Z_{\dot r} r, \\ b_2 &= X_{\dot v} u + Y_{\dot v} v + Y_{\dot w} w + Y_{\dot p} p + Y_{\dot q} q + Y_{\dot r} r, \\ b_1 &= X_{\dot u} u + X_{\dot v} v + X_{\dot w} w + X_{\dot p} p + X_{\dot q} q + X_{\dot r} r, \\ b_4 &= X_{\dot p} u + Y_{\dot p} v + Z_{\dot p} w + K_{\dot p} p + K_{\dot q} q + K_{\dot r} r, \\ b_5 &= X_{\dot q} u + Y_{\dot q} v + Z_{\dot q} w + K_{\dot q} p + M_{\dot q} q + M_{\dot r} r, \\ b_6 &= X_{\dot r} u + Y_{\dot r} v + Z_{\dot r} w + K_{\dot r} p + M_{\dot r} q + N_{\dot r} r. \end{aligned}$$

Here $(u\ v\ w\ p\ q\ r)$ is the linear and angular velocity vector. Again, assumptions of Section 4.1.1 give us a simple expression for C^B_{RB} and C^B_A, hence C^B is defined as

$$C^B = \begin{bmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & mw & -mv & 0 & I_{zz}r & -I_{yy}q \\ 0 & 0 & 0 & -I_{zz}r & 0 & I_{xx}p \\ 0 & 0 & 0 & I_{yy}q & -I_{xx}p & 0 \end{bmatrix} +$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -Z_{\dot{w}}w & Y_{\dot{v}}v \\ 0 & 0 & 0 & Z_{\dot{w}}w & 0 & -X_{\dot{u}}u \\ 0 & 0 & 0 & -Y_{\dot{v}}v & X_{\dot{u}}u & 0 \\ 0 & -Z_{\dot{w}}w & Y_{\dot{v}}v & 0 & -N_{\dot{r}}r & M_{\dot{q}}q \\ Z_{\dot{w}}w & 0 & -X_{\dot{u}}u & N_{\dot{r}}r & 0 & -K_{\dot{p}}p \\ -Y_{\dot{v}}v & X_{\dot{u}}u & 0 & -M_{\dot{q}}q & K_{\dot{p}}p & 0 \end{bmatrix}. \tag{4.16}$$

With all the equation terms clear, the next section uncouples the model and describes the LS identification method taken for a particular DoF.

4.2 DECOUPLED MODEL

During the identification procedure, the vehicle will be actuated in a single DoF. After all the simplifications proposed in Section 4.1.1 are applied to Equation 4.1, the resultant dynamic equation for one DoF, the X movement in this case, can be conveniently expressed in the fixed body frame $\{B\}$ as

$$\begin{gathered} X - (X_u + X_{u|u|}|u|)u + \tau_{p_x} = \\ (m - X_{\dot{u}})\dot{u} + (m - Z_{\dot{w}})wq - (m - Y_{\dot{v}})vr. \end{gathered} \tag{4.17}$$

Since the vehicle will perform decoupled movements during the identification process, Coriolis terms can be discarded. Hence, the final dynamic equation for the X DoF can be obtained as

$$X - (X_u + X_{u|u|}|u|)u + \tau_{p_x} = (m - X_{\dot{u}})\dot{u}. \tag{4.18}$$

By isolating the acceleration we obtain

$$\frac{1}{(m - X_{\dot{u}})}(X - X_{\dot{u}}u - X_{u|u|}u|u| + \tau_{p_x}) = \dot{u}. \tag{4.19}$$

If we assume that

$$\begin{aligned} \alpha &= -\tfrac{X_u}{(m-X_{\dot{u}})}, \\ \beta &= -\tfrac{X_{u|u|}}{(m-X_{\dot{u}})}, \\ \gamma &= +\tfrac{1}{(m-X_{\dot{u}})}, \\ \delta &= +\tfrac{\tau_p}{(m-X_{\dot{u}})}, \end{aligned} \tag{4.20}$$

we can give the decoupled equation as

$$\dot{u} = \alpha u + \beta u|u| + \gamma X + \delta. \tag{4.21}$$

The same procedure can be applied to each DoF. Therefore, a generic decoupled equation of motion for the i-degree can be considered as

$$\dot{x}_i = \alpha_i x_i + \beta_i x_i|x_i| + \gamma_i \tau_i + \delta_i. \tag{4.22}$$

Identification becomes easier if Equation 4.22 is used. The variable x represents speed and τ the force or moment for a particular DoF. The same procedure can be applied to each DoF. By running decoupled experiments, α_i, β_i, γ_i and δ_i can be estimated by means of LS techniques. Then, as the mass m of the vehicle is known since it has been measured off-line, the physical parameters X_u, $X_{u|u|}$, $X_{\dot{u}}$ and τ_p can be easily extracted as shown in Equation 4.20. Note that these are the unknown parameters of the coupled equation of motion. Hence, although a decoupled experiment is being run, the whole model can be estimated. Details of the LS identification algorithm, the procedures and the identification results are given in the next sections.

4.3 IDENTIFICATION METHODOLOGY

Identification methods based on discrete-time models [82] have been extensively used in the area of engineering applications, owing to the fact that the related algorithms can be easily implemented. It is worth noting, however, that in applications involving the identification of an intrinsically continuous-time physical system, such as an AUV, it is generally preferable to use a continuous-time method. In fact, given a model described by a set of differential equations, it is more direct and much easier to use that directly into an identification algorithm rather than using an intermediate discrete-time form, that can sometimes be error prone. It is also important to note that a continuous time model has a global validity in the sense that it can be used for generating a variety of discrete-time models to be applied to the design of control or fault detection systems by simply selecting a suitable sampling interval. The LS method used for the model identification of Ictineu AUV is based on the integral of the dynamics equations. The transformation of differential equations into integral equations is used to provide a better numerical performance of the LS estimation algorithm. Let us consider the dynamics Equation 4.22 which can be rewritten as follows

$$\dot{x}_i = \begin{bmatrix} x_i & x_i|x_i| & \tau_i & \eta_i \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \\ \gamma_i \\ \delta_i \end{bmatrix} + \varepsilon_i. \tag{4.23}$$

As can be easily recognized, the AUV decoupled dynamics, as expressed by Equation 4.23, is a particular case of a more general class of non-linear systems that are linear with respect to the system's parameter vector. The system's dynamics can be expressed by

$$\begin{aligned} \dot{x}(t) &= \phi(x(t),\tau(t))\theta + \varepsilon(t), \\ y(t_k) &= x(t_k) + e(t_k), \end{aligned} \tag{4.24}$$

where $\phi \in \mathbb{R}^{n \times l}$, n being the number of measurements and l the number of parameters to identify, is a matrix valued function depending only on state and control vectors, while $\theta \in \mathbb{R}^l$ is a constant and unknown parameter vector that characterizes the system's dynamics. The identification problem consists of estimating the unknown parameter vector θ according to the output prediction error method

[82]. Identification of parameter vector θ is equivalent to the minimization of a scalar cost function

$$J(\theta) = \frac{1}{2}\sum_{k=1}^{n} \varepsilon^T(t_k)W^{-1}(t_k)\varepsilon(t_k). \tag{4.25}$$

The cost function is made up of a weighted sum of squares of prediction errors $\varepsilon(t_k)$, which are the difference between the observed output $y(t_k)$ and the one-step prediction of the output $\hat{y}(t_k)$ where the error is given by

$$\varepsilon(t_k) = y(t_k) - \hat{y}(t_k). \tag{4.26}$$

The positive definite matrices $\{W^{-1}(t_k)\}_{k=1}^{n}$ consist of weights that should take into account the reliability of measurements at each discrete time instant. If the measurement noise vector $\varepsilon(t_k)$ is zero-mean, then

$$\hat{y}(t_k) = \hat{x}(t_k), \tag{4.27}$$

where $\hat{x}(t_k)$ denotes the expected state vector at time t_k. In order to determine a solution to the minimization of the cost function expressed by Equation 4.25, it is necessary that an estimate of the one-step predicted output $\hat{y}(t_k)$ is available. For this purpose, let us formally integrate both sides of the state equation in Equation 4.24 with two subsequent time instants t_{k-1} and t_k obtaining

$$x(t_k) - x(t_{k-1}) = [\int_{t_{k-1}}^{t_k} \phi(x(s), \tau(s))\delta s]\theta. \tag{4.28}$$

If, taking into account Equation 4.27, it is assumed that $x(t_{k-1}) = \tilde{y}(t_{k-1})$, where $\tilde{y}(t_{k-1})$ is a properly filtered version of the output vector $y(t_{k-1})$, i.e. if we assign to the unknown state vector a corresponding filtered output, then we obtain the following estimate for the state vector at time t_k,

$$\begin{gathered} \hat{x}(t_k) = \tilde{y}(t_{k-1}) + F_k\theta, \\ F_k = \int_{t_{k-1}}^{t_k} \phi(\hat{(x)}(s), \tau(s))\delta s. \end{gathered} \tag{4.29}$$

Thus, the one-step prediction error of Equation 4.26 can be computed as

$$\varepsilon(t_k) = \tilde{y}(t_k) - \tilde{y}(t_{k-1}) - F_k\theta. \tag{4.30}$$

As far as the considered AUV dynamics has been taken into account, it has been found out that a simple low pass filter was adequate enough to remove the noise measurement errors. By reordering Equation 4.30, we have

$$\begin{gathered}
\tilde{y}(t_k) - \tilde{y}(t_{k-1}) = F_k \cdot \theta + \varepsilon(t_k), \\
\begin{bmatrix} \tilde{x}_1 - \tilde{x}_0 \\ \tilde{x}_2 - \tilde{x}_1 \\ \dots \\ \tilde{x}_n - \tilde{x}_{n-1} \end{bmatrix} = \\
\begin{bmatrix} \int_{t_0}^{t_1} \tilde{x}\delta t & \int_{t_0}^{t_1} \tilde{x}|\tilde{x}|\delta t & \int_{t_0}^{t_1} \tau\delta t & \int_{t_0}^{t_1} \delta t \\ \int_{t_1}^{t_2} \tilde{x}\delta t & \int_{t_1}^{t_2} \tilde{x}|\tilde{x}|\delta t & \int_{t_1}^{t_2} \tau\delta t & \int_{t_1}^{t_2} \delta t \\ \dots & \dots & \dots & \dots \\ \int_{t_{n-1}}^{t_n} \tilde{x}\delta t & \int_{t_{n-1}}^{t_n} \tilde{x}|\tilde{x}|\delta t & \int_{t_{n-1}}^{t_n} \tau\delta t & \int_{t_{n-1}}^{t_n} \delta t \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \\
+ \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_n \end{bmatrix}.
\end{gathered} \tag{4.31}$$

A linear equation in the vector of unknowns is obtained. Using the above equation, the LS estimation techniques [82] can be applied to estimate the vector of unknowns θ and the related estimation error covariance matrix through the application of the following equations

$$\begin{aligned} \theta_{LS} &= (F^T \cdot W^{-1} \cdot F)^{-1} \cdot F^T \cdot W^{-1} \cdot Y, \\ P(\theta) &= (F^T \cdot W^{-1} \cdot F)^{-1}. \end{aligned} \tag{4.32}$$

The next section describes the experimental identification process used for each of the identified DoFs.

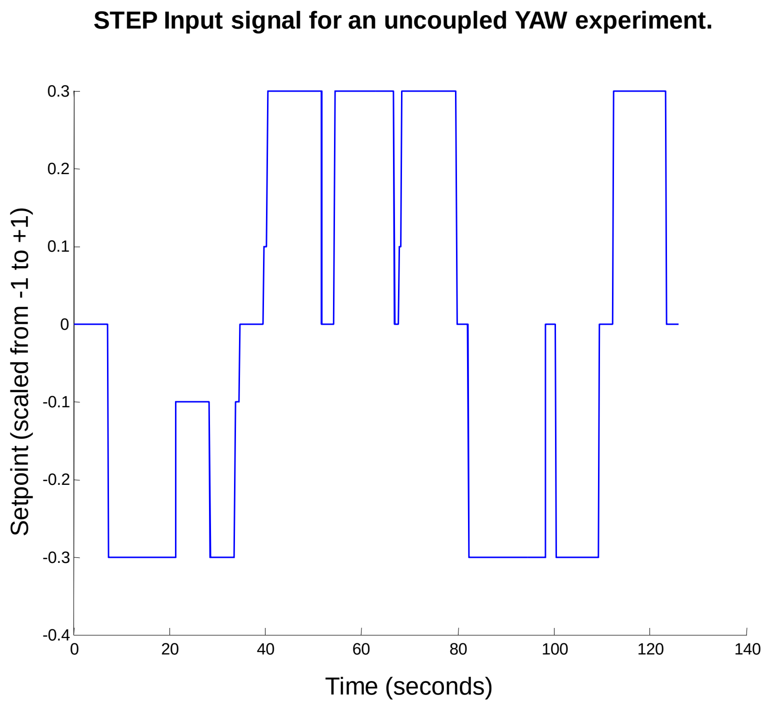

Figure 16: Example STEP input used for the identification of the Yaw DoF.

4.4 IDENTIFICATION PROCESS AND RESULTS

The identification method was carried out with real data measured with Ictineu AUV. The number of DoFs required to perform an underwater cable tracking task, like the one described in this thesis, is 3: Surge (movement along the vehicle's X axis), Sway (movement along the vehicle's Y axis) and Yaw (rotation around the vehicle's Z axis). Also, following the cable at different distances from the seabed requires the Heave DoF (movement along the vehicle's Z axis) to be identified but, as will be explained later, the shape of the vehicle suggests that the Sway parameters can also be used for the Heave. The next lines describe the identification process carried out and the results obtained for each identified DoF.

4.4.1 *Phase 1: Decoupled experiments*

The underwater robot Ictineu AUV is actuated in a single DoF through several independent tests for each DoF that has to be identified. During each test, all the navigation data measured by the sensors (position, velocity and acceleration in the 6 DoFs) as well as the input setpoints is recorded in

Table 2: Parameter identification results.

	Parameters		
DoF	D_L	M_A^B	τ_p
X	10,974 N/(m/s)	26,472 Kg	0,326 N
Y	12,286 N/(m/s)	29,235 Kg	-0,033 N
Z	3,183 N/(rad/s)	7,674 Kg.m^2	-0,013 N
Mass		55 Kg	
B		55 Kg	
I_{zz}		4,583 Kg.m^2	

log files. The setpoints signals used to actuate the robot are STEP signals of different lengths and amplitudes. The objective of this initial phase is to generate a rich data set of values around the robot's working point, insuring that the final identified model approximates the vehicle's dynamics over a wide range around this point. Figure 16 shows an example of a multi STEP input used in one of the identification tests.

4.4.2 *Phase 2: Data validation*

Here we aim to discard bad experiments, or at least part of them, such as when the force exerted by the umbilical cable cannot be ignored. In order to assume a decoupled identification, we must be sure that speeds in the rest of the DoFs are zero or almost zero. For this purpose, the data set from the experiments performed in Phase 1 is plotted and analyzed, eliminating all those tests which do not accomplish all the previously specified conditions.

4.4.3 *Phase 3: Filtering*

To reduce the effects of measurement noise, the data was filtered. Position and force were filtered using a Hamming filter and the velocity was computed through position differentiation by means of a Savitzky-Golay filter [136]. The acceleration was computed through velocity differentiation using a Savitzky-Golay filter as well. Once we got the ve-

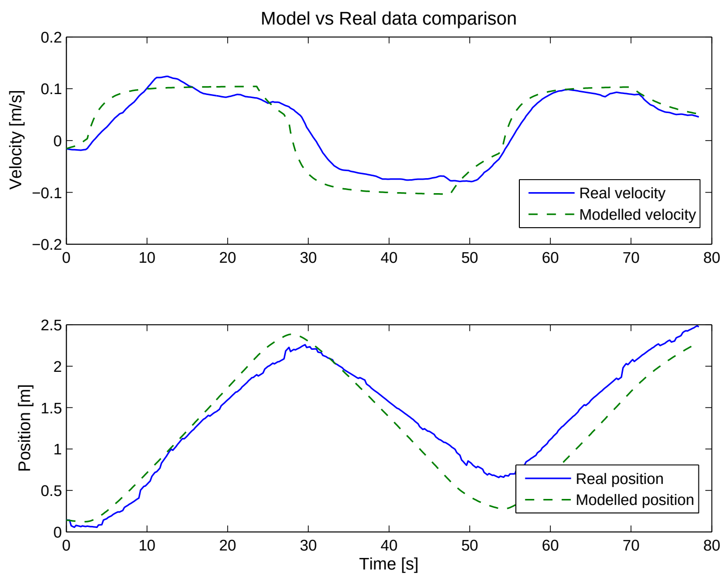

Figure 17: Velocity and position response for the Surge DoF.

locity, we integrated this signal and compared it with the original position to be sure that the signals obtained with the filter were not delayed.

4.4.4 *Phase 4: Off-line model identification*

At this point, a final data set has been collected for each DoF that has to be identified. Therefore, the identification methodology described in Section 4.3 is followed to estimate the model parameters of Ictineu AUV.

4.4.5 *Phase 5: Test selection and mean values*

From the previous phase, outliers in the experiments are detected and discarded. The results from the best ones are averaged to get the final values for the estimation. The identified parameters for Ictineu AUV can be seen in Table 2. Parameters for the Surge, Sway and Yaw are the mean of the results for the validated experiments. Note that the quadratic damping term D_Q has been ignored due to the low speeds achieved by Ictineu AUV during the tests, $< 0.5m/s$. The Heave DoF (movement around the vehicle's Z axis) has not been identified, nevertheless, due to the square shape of the vehicle, Heave is assumed to behave like the Sway DoF.

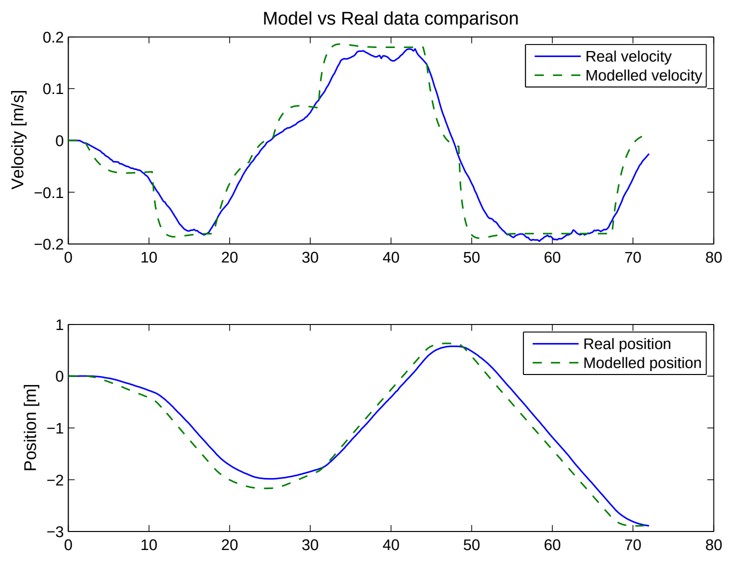

Figure 18: Velocity and position response for the Sway DoF.

4.4.6 *Phase 6: Simulation*

The final identified model is used to simulate some true experiments. The real and simulated results, when the real system and the model are subject to the same initial conditions and control inputs, are plotted for comparison. Thus, the same sample setpoint input is used first on the real vehicle and then on the simulator while measurements of the velocity and position are extracted. Figure 17 shows the results obtained with the final identified model for the Surge DoF. The graph depicts the long-term simulation capability of the model with respect to the real measured values of velocity and position. As can be seen in the bottom image of Figure 17, as time goes on, the simulated position diverges with respect to the real measured one. At the end of the test (around 80 seconds), the difference between both positions is less than 30cm. On the other hand, as illustrated in the top image of Figure 17 representing the real and the simulated velocity, these ones stay very close to each other during the whole test. These results correspond to the expected ones, since the identification method performs an LS error minimization with respect to the velocity and not the position. The results for the Sway and Yaw DoFs are also depicted in Figure 18 and Figure 19. The results for these DoFs are better than those obtained with the Surge DoF. The difference of the final position of the model with respect

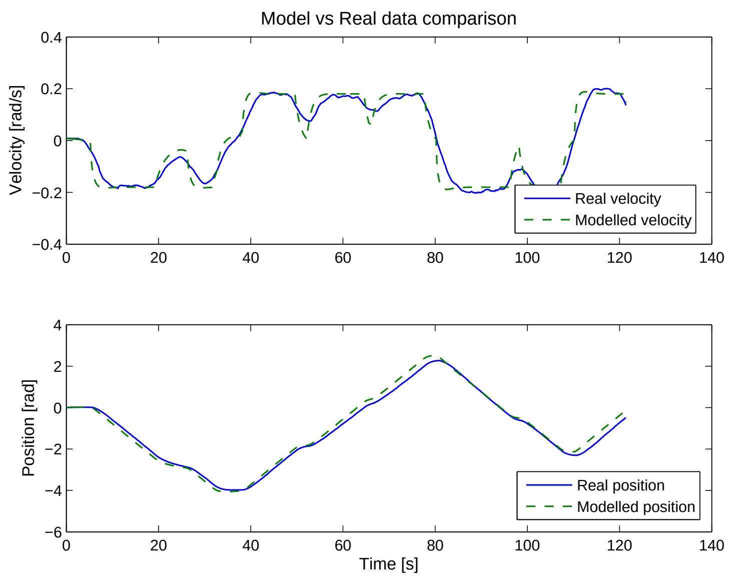

Figure 19: Velocity and position response for the Yaw DoF.

to the real values is very small compared to the difference experienced by the Surge. The main reason for such a difference in the results is the fact that the range of velocities used to identify the Surge DoF has more amplitude, since the robot moves faster in Surge, than the ranges used for Sway and Yaw. As the ranges move away from the working point, the accuracy of the model falls for a particular test.

5 EXPERIMENTAL SETUP

The purpose of this chapter is to report the main characteristics of the different elements used to build the experimental setup. First, the most notable features of the Ictineu AUV are given. These include the design principles, the actuators and the on board sensors. An insight of the control architecture of the robot is given next. As stated in Section 3.2, the RL algorithms chosen to carry out the tests are implemented as behaviors inside this architecture. Also, the problem of underwater pipeline/cable tracking is detailed. Different common approaches to carrying out this task are discussed and among them the one selected to carry out the experiments of this thesis, a vision based system, is especially analyzed. The algorithm used to estimate the position and orientation of the cable within the image plane is presented. Finally, the Computer Vision and Robotics Group facility CIRS where all the tests have been performed is also described at the end of this chapter. The correct operation of all these systems in real-time computation allows the experimentation and, therefore, the evaluation of the proposed RL algorithms for the particular task of autonomous underwater cable tracking.

5.1 ICTINEU AUV

This section describes the Ictineu AUV (see Figure 20), the research vehicle built at the VICOROB of the University of Girona which constitutes the experimental platform of this thesis. Its name is a tribute to the Catalan engineer Narcís Monturiol (1819 – 1885). He was the inventor of the first combustion engine-driven submarine, *Ictineu*, which was propelled by an early form of air-independent propulsion.

The Ictineu AUV is the result of a project started in 2006. During the summer of that year, the Defence Science and Technology Lab (DSTL), the Heriot-Watt University and the National Oceanographic Center of Southampton organized the first Student Autonomous Underwater Challenge Europe (SAUCE), a European wide competition for students to foster research and development in underwater technol-

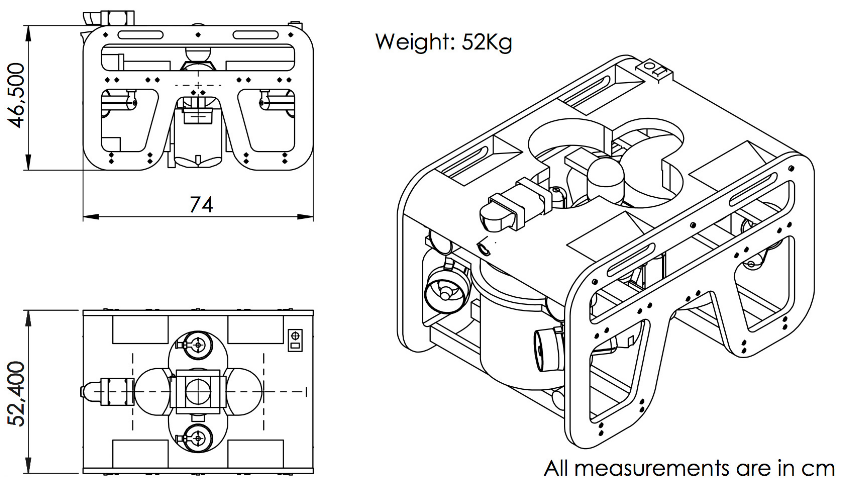

Figure 20: Ictineu AUV weight and dimensions.

ogy. The Ictineu AUV was originally conceived as an entry for the SAUCE competition by a team of students collaborating with the VICOROB [124]. Although the competition determined many of the vehicle's specifications, Ictineu was also designed keeping in mind its posterior use as an experimental platform for various research projects in our laboratory. The experience obtained from the development of previous vehicles by the group, Garbi ROV [8], Uris [22] and Garbi AUV, made it possible to build a low-cost vehicle of reduced weight (52 Kg) and dimensions (74 x 46.5 x 52.4 cm) with remarkable sensorial capabilities and easy maintenance.

The Ictineu AUV was conceived using a typical open frame design. This configuration has been widely adopted by commercial ROVs because of its simplicity, toughness and reduced cost. Although the hydrodynamics of open frame vehicles is known to be less efficient than that of closed hull vehicles, they are suitable for applications not requiring movements at high velocities or traveling long distances. The robot's chassis is made of Delrin, an engineering plastic material which is lightweight, durable and resistant to liquids. Another aspect of the design is the modular conception of its components which simplifies upgrading the vehicle and makes it easier to carry out maintenance tasks. Some of the modules (the thrusters and the major part of the sensors) are watertight and therefore, are mounted directly onto the vehicle's chassis. On the other hand, two cylindrical pressure containers made of aluminum house the power and computer modules while a smaller one made of Delrin contains the Motion Reference Unit (MRU). Their end caps are sealed with conventional O-ring closures while the electrical connections with other hulls or external sensors are made with plastic cable glands sealed with epoxy resin. The Ictineu is propelled by six thrusters that allow it to be fully actuated in *Surge* (movement along the X axis), *Sway* (movement along the Y axis), *Heave* (movement along the Z axis) and *Yaw* (rotation around the Z axis) achieving maximum speeds of 3 knots. The Ictineu AUV is passively stable in both *pitch* and *roll* DoFs as its center of buoyancy is above the center of gravity. This stability is the result of an accurate distribution of the heavier elements in the lower part of the chassis combined with the effect of technical foam placed on the top, which provides a slightly positive buoyancy.

One of the main objectives of the laboratory was to provide the underwater robot with a complete sensor suite. This sensor suite was created adding new sensors to correct some limitations of the old prototypes. The robot includes a Tritech Miniking Mechanically Scanned Imaging Sonar (MSIS) designed for use in underwater applications such as obstacle avoidance and target recognition. Also, the robot is equipped with a SonTek Argonaut Doppler Velocity Log (DVL) especially designed for applications which measure ocean currents, vehicle speed over ground and altimetry using a precise 3-axis measurement system. The particular spatial distribution chosen to place the acoustic sensors within the vehicle frame to avoid dead zones improve their overall performance. Moreover, the Ictineu AUV has a compass which outputs the sensor heading (angle with respect to the magnetic north), a pressure sensor for water column pressure measurements and a Xsens MTi low cost miniature MRU which provides a 3D orientation (attitude and heading), 3D rate of turn as well as 3D acceleration measurements. Finally, the robot is equipped with two cameras. The first is a forward-looking color camera mounted on the front of the vehicle and is intended for target detection and tracking, inspection of underwater structures and to provide visual feedback when operating the vehicle in ROV mode. The second camera is a Tritech Super SeaSpy colour Charge Coupled Device (CCD) Underwater Camera, located in the lower part of the vehicle and downward-looking. This camera is mainly used to capture images of the seabed for research on image mosaicking.

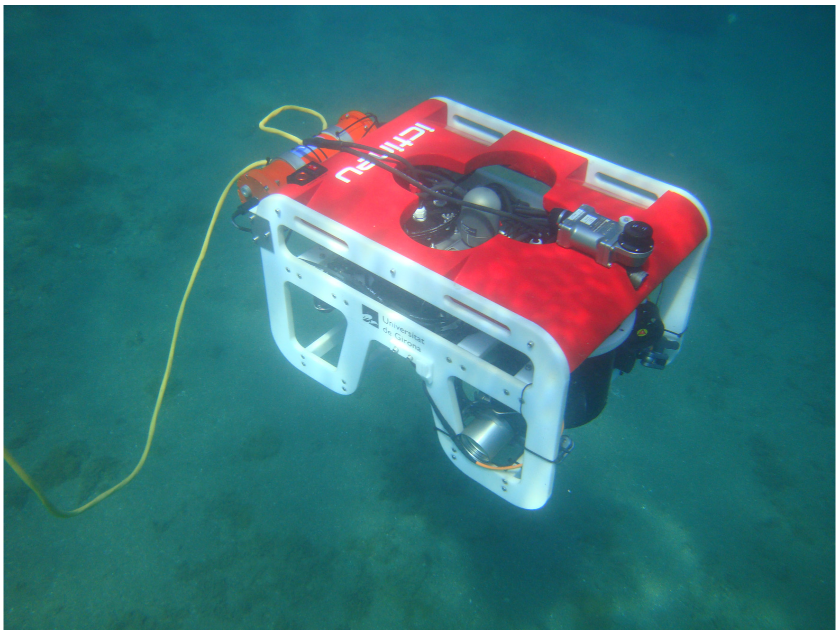

Figure 21: Ictineu AUV during a mission in a real environment.

The downward-looking camera is the main sensor used for the experiments developed in this dissertation. Nowadays, the Ictineu AUV is being used as a research platform for different underwater inspection projects which include dams, harbors, shallow waters and cable/pipeline inspection (see Figure 21).

5.2 THE O2CA2 CONTROL ARCHITECTURE

The O2CA2 Control Architecture was first proposed in [126] with the aim of developing a control strategy for an autonomous vehicle. Nowadays, an evolution of that control architecture can be found in all the operating vehicles of the VICOROB, including the Ictineu AUV. The control architecture has the task of guaranteeing the robot's functionality. The real-time POSIX, together with the ACE/TAO CORBA-RT ORB, have been extensively used to develop the architecture as a set of distributed objects with soft real time capabilities. These objects are distributed between the two onboard PCs and, when operating in tethered mode, the external PC. The architecture is composed of a base system and a set of objects customized for each particular robot, which makes possible the sharing of the same software architecture with all the vehicles in the laboratory. There are classes providing soft real-time capabilities that guarantee

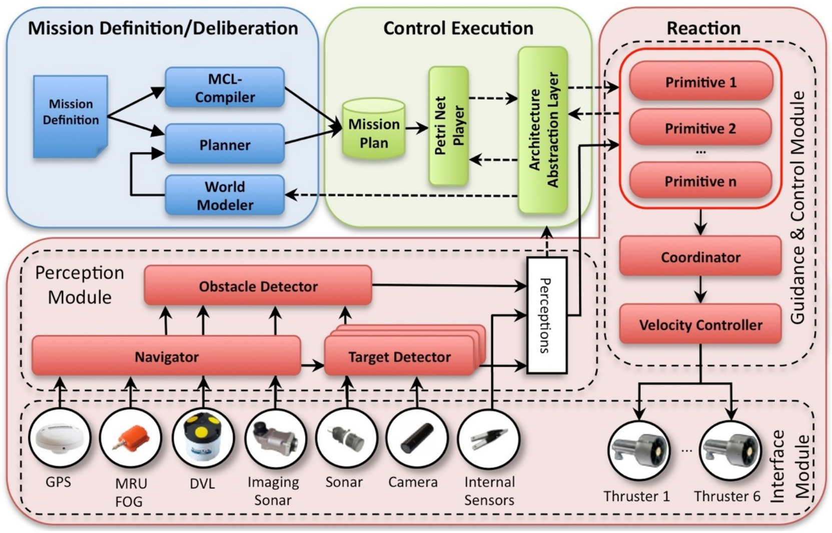

Figure 22: Schematic of the Ictineu AUV control architecture.

the period of execution of the periodic tasks such as the controllers or the sensors. Another important part of the base systems are the loggers. A logger system is used to log data from sensors, actuators or any other object component. Loggers do not execute in real time, they are background processes which receive the data from real time objects. Their role consists of packing the data and saving them into files. It is worth noting that, although loggers do not run in real time, the data has a time-stamp corresponding to the gather time. Moreover, all the computers in the network are synchronized by means of the Network Time Protocol (NTP) and hence, all the data coming from different sensors can be time related. The software architecture is divided into three modules as represented in Figure 22: interface module, perception module and control module.

5.2.1 *Interface Module*

This is the only module containing software objects which dialog with the hardware. There are basically two types of objects: sensor objects responsible for reading data from sensors and actuator objects responsible for sending commands to the actuators. As detailed in Section 5.1, sensor objects for the Ictineu AUV include a DVL, an imaging sonar, a MRU, two cameras and a depth sensor. There are also objects for the safety sensors like water leakage detectors and internal temperature and pressure sensors that allow for the monitoring of the conditions within the pressurized containers. Actuator objects for the Ictineu AUV include the thrusters.

5.2.2 *Perception Module*

This module contains two basic components: the *navigator* and the *obstacle detector*. The navigator has the goal of estimating the position of the robot. To accomplish this task, there is an interface called *navigation sensor* from which all the localization sensors (DVL, MRU and depth sensor) inherit. This interface provides all these sensors with a set of methods to return the position, velocity and acceleration in the 6 DoF together with an estimation of the quality of these measurements. The navigator can be dynamically connected to any navigation sensor and, using the quality factor, fuses the data to obtain a more accurate position, velocity and acceleration. The control module uses the navigation data provided by the navigator keeping the behaviors independent of the physical sensors being used for the localization. The *obstacle detector* uses the same philosophy to provide obstacle position in the fixed world frame. The obstacle detector is also used to detect the distance between the vehicle and the bottom of the pool. Detecting frontal obstacles is possible using the the imaging sonar while pool bottom obstacles can be detected by means of the DVL sensor.

5.2.3 *Control Module*

The control module receives sensor inputs from the perception module. This sensory information is driven into the active behaviors and each one generates a response. The *coordinator* mixes all the responses with its respective priorities and sends command outputs to the actuators residing in the interface module. Since *task* and *behavior* are words that can be interpreted in different ways depending on the authors in the literature, here we describe how they are interpreted within this thesis. A behavior is a function that maps the sensor input space, *stimuli*, into a set point (reference point for the controller) *behavior response*, for the robot's low level controller. The behavior response is chosen in a way that drives the robot towards its corresponding goal. As an example, the goal corresponding to the *Keep-Depth* behavior is considered to be achieved when the robot is within an interval around the desired depth. On the other hand, a task is a set of behaviors that are enabled/disabled together to achieve a more complex goal. For instance, *Keep-Depth* and *FollowCable* can work together to allow for planar navigation while following a submerged cable at a constant depth. The control module follows the principles of the hybrid control architectures, described in Section 3.1, organized in three layers: vehicle level, task level and mission level.

VEHICLE LEVEL. This level is composed of a Multiple-Input-Multiple-Output (MIMO) Proportional-Integral-Derivative Controller (PID) for each DoF. This object reads the vehicle's velocity from the navigator and sends the velocity setpoints to the coordinator. This level also includes a simple control allocator strategy based on the pseudo inverse of the thruster configuration matrix [47].

TASK LEVEL. The task level is a conventional behavioral layer which includes a library of behaviors that can run alone or in parallel. Each behavior has a particular goal. The input of a behavior comes from the perception module. As output, the behavior response contains the setpoints and the activation level for every DoF. Also, the behavior response contains an integer term which gives a priority for each of the active behaviors. During the execution of a mission, more than one behavior can be enabled simultaneously. Hence, a coordinator module is used to fuse all the responses corresponding to the enabled behaviors into a single response to be sent to the low level controller located at the vehicle level. The coordinator output, after combining all the active behaviors, is a vector as large as the number of the robot's DoFs [112]. The RL-based behaviors proposed in this thesis have been programmed at this level.

MISSION LEVEL. The mission level is responsible for the sequencing of the mission tasks, selecting for each mission phase the set of behaviors that must be enabled. The mission controller is built with a *Petri network* [103] in which the sequence of tasks is defined. Since the vehicle can move in an unstructured environment, unexpected situations have to be taken into account

> by the mission designer. A Petri network is made up of *nodes* which can be active or not active. In the Petri network developed for the Ictineu AUV [36], each node represents a behavior with a particular configuration to be executed on the task controller. A library of behaviors is used to define a mission. Each one has a simple goal such as *move to point*, *keep depth*, *search a target* or *follow a cable*. Therefore, the mission controller has the work of defining the task that the robot is accomplishing at each moment by activating or deactivating behaviors with the final goal of fulfilling the mission. The mission controller does not determine the actions that guide the robot, it only determines the active behaviors and their configuration which, through the task controller, will be coordinated to guide the robot.

Section 5.1 and Section 5.2 have described the hardware and software elements which compose the Ictineu AUV. Since its creation, the vehicle has experienced extensive usage in many different research fields. It has proved to be a very reliable platform, requiring only minor maintenance tasks. We expect the Ictineu AUV to become a reference for all the future prototypes developed in our laboratory.

5.3 CABLE TRACKING IN UNDERWATER ENVIRONMENTS

Nowadays, oil companies have a huge net of pipes spread under the oceans. Most offshore platforms are located on the continental shelf, though with advances in technology and increasing crude oil prices, drilling and production in deeper waters has become both feasible and economically viable. Also, power transmission and communications companies share the same problems for their cable lines. Wired communications are possible thanks to cables properly designed which, during a costly deployment, are laid out over the seabed. Due to the particularly aggressive conditions to which these cables are exposed, the feasibility of such installations can only be guaranteed by means of a suitable inspection programme. It must provide timely information on the current state of the installation or about potentially hazardous situations or damages caused by the movement of the seabed, corrosion, or human activities such as marine traffic or fishing [62] [168]. The use of professional

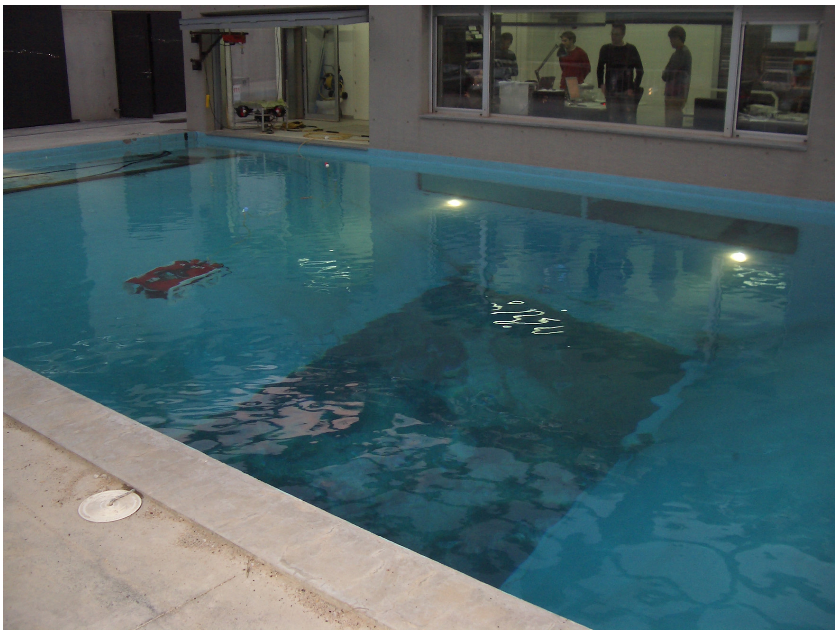
Figure 23: Ictineu AUV during experimental tests at CIRS facility.

divers for the inspection and maintenance of underwater cables/pipelines is limited by depth and time. Also, a manual visual control is a very tedious job and tends to fail if the operator loses concentration. ROVs represent an alternative to human operators. The main drawback of using ROVs for surveillance missions resides in the cost, since it increases rapidly with depth because of the requirements for longer umbilicals and a support ship. All these reasons point towards AUVs as an alternative solution for such missions. An AUV can be deployed from the coast without the help of a ship, perform the whole tracking mission by itself gathering all useful data from sensors and surface at the desired location for recovery.

Several systems have been developed for underwater cable/pipeline inspection purposes. Basically, the technology applied classifies the methodologies in three large groups depending on the sensing device used for tracking the cable/pipeline: magnetometers, sonar and vision based methods. Magnetometer systems are limited to tracking metallic or electric lines. A magnetometer is a measurement instrument used to measure the strength and/or direction of the magnetic field in the vicinity of the instrument. With the installation of a magnetometer, an AUV can detect the magnetic field induced by a cable, calculate its position and follow it. The main drawback of this technique is the low accuracy of the measurements caused by misalignment of the magnetic sensor's axes, waving of laid cables and magnetic noise. The advantage is that these methods can detect the cable even if it is buried below the seabed and no artificial illumination is required. Some results with magnetometers can be found in [63, 14]. Over the last decade, sonar-based technologies have received a lot of interest due to the great advances in the field. For cable/pipeline tracking purposes, most of the experimental results obtained with sonar-based methods combine two kind of sensors: *multibeam echo sounders* and *sidescan sonars*. A multibeam echo sounder is a sensor specifically designed to produce bathymetric maps of large areas of the seabed. It is composed of an array of hydrophones which can emit fan shaped beams towards the bottom and measure the range of a strip of points placed perpendicularly to the direction of the vehicle's movement. A sidescan sonar allows large areas of the seabed to be searched quickly by producing acoustic images. Its mode of operation is similar to that of multibeam echo sounders, but oriented to imaging tasks. While the sonar moves along a survey path, it emits fan shaped pulses down towards the seabed over a wide angle perpendicular to the direction of the movement, producing a strip of echo intensity measurements. Initially, the cable/pileline is located from a high altitude over the seabed using the sidescan sonar and, once it has been detected, it is tracked at low altitude by means of the multibeam echo sounder. Sonar systems can operate with no illumination and the accuracy and resolution of the measurements obtained with the multibeam echo sounder is very high. As a disadvantage, sidescan sonar images are generally very noisy, making analysis of the raw image very difficult. To overcome this, some data pre-processing is required. Some results with sonar-based systems can be found in [44, 64].

Vision based systems represent a good alternative to sonar methods. Although sonar sensors are becoming smaller and require less power to operate, vision cameras, apart from being a passive sensor, provide far more information with a larger frequency update, are inexpensive, much less voluminous and can be powered with only a few watts. LED technology is also contributing to reducing the size of the lighting infrastructure and the related power needs, which also matters in this case. For performing a camera vision-based tracking, the AUV control architecture must command

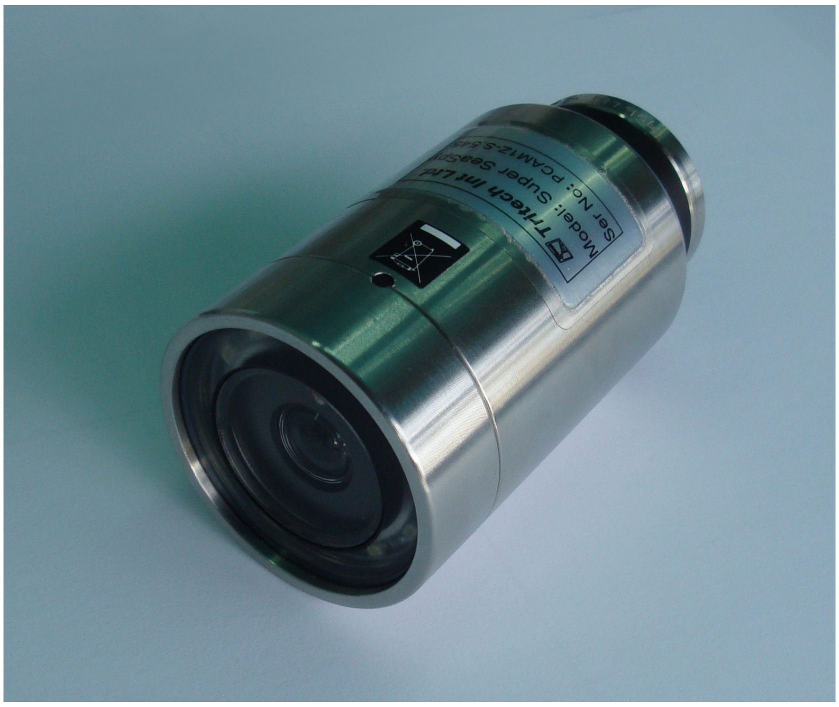

Figure 24: The Tritech Super SeaSpy camera used by the vision system to track the pipe/cable.

the vehicle to travel over the cable/pipeline, which leads to the frame by frame tracking of the position and the orientation of the cable/pipeline. Over the last few years, several research groups have showed the suitability of vision cameras as a short range sensor for underwater environments either for navigation or for mission tasks [111, 19]. For the final experiments of this dissertation, a vision-based system developed at the University of the Balearic islands has been chosen to track a submerged cable in a controlled environment. The VICOROB laboratory has been collaborating with the Computer Science and Mathematics Department of the University of the Balearic Islands in several National and European projects for years, obtaining successful results. All the tests have been carried out in the Computer Vision and Robotics Group facility, CIRS, at the University of Girona. Although the conditions in the pool are far from real undersea conditions, it allows comprehensive testing by a single operator and represents a unique opportunity to test RL methods in underwater conditions (Figure 23). The next section gives details of the vision system used and the procedures to obtain the position and orientation of the cable.

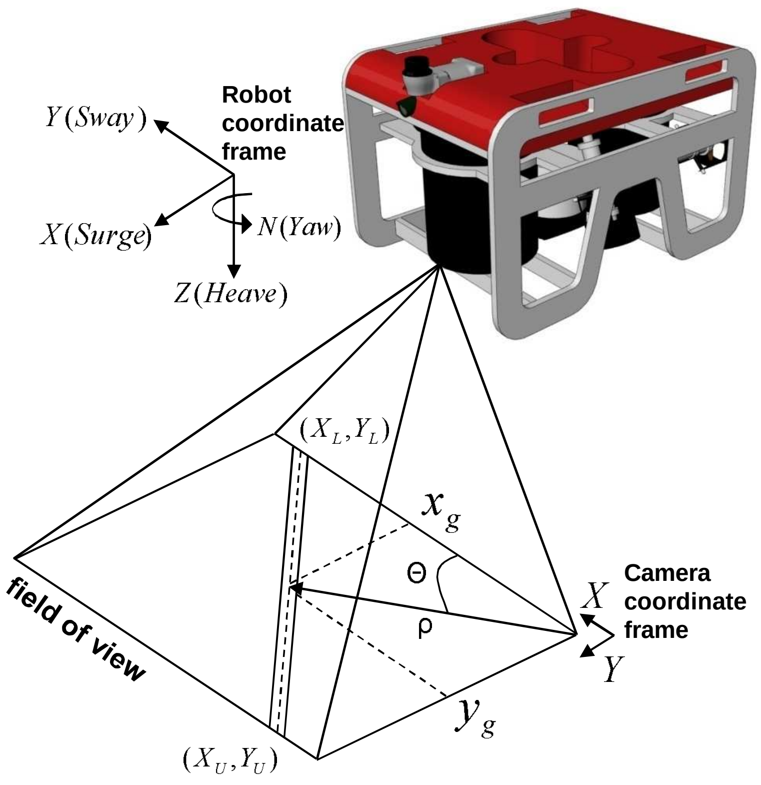

Figure 25: Coordinates of the target cable with respect to the Ictineu AUV.

5.4 THE CABLE TRACKING VISION-BASED SYSTEM

The Tritech Super SeaSpy colour CCD downward-looking camera installed on the Ictineu AUV will be used for the vision algorithm to track the cable. The camera has integrated white ring Light-Emitting Diodes (LEDs) providing uniform illumination across the viewing area. A feedback loop automatically adjusts the lighting level so that optimum picture quality is achieved, regardless of the reflectivity of the work surfaces. This integral lighting provides an underwater camera that is well suited for close proximity inspection work where little or no lighting is available. The camera has internal focus adjustment that allows the focal range to be set between 50mm and infinity with a diagonal field of view of 70 degrees (see Figure 24). This sensor does not provide us with absolute localization but it gives us relative information about the position and orientation of a particular object with respect to our vehicle: if we are too close/far or if we should move to the left/right in order to center the object in our image plane.

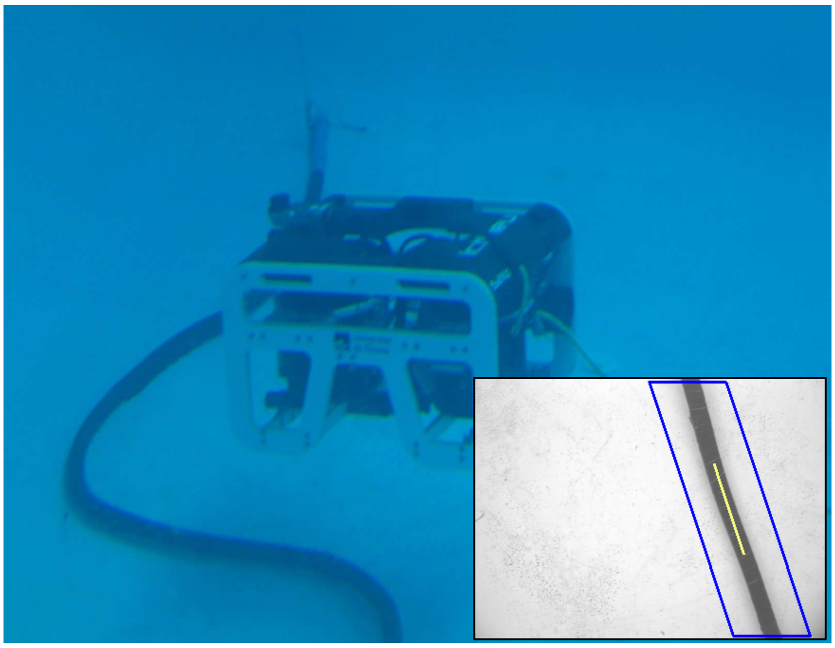

Figure 26: Ictineu AUV in the test pool. Small bottom-right image: Detected cable.

The vision-based algorithm used to locate the cable was first proposed in [110] and later improved in [10]. It exploits the fact that artificial objects present in natural environments usually have distinguishing features; in the case of a pipe or a cable, given its rigidity and shape, strong alignments can be expected near its sides. In order to obtain the pipe/cable parameters, an optimized segmentation step is executed. Given the contours of the resultant regions, alignments of contour pixels are determined. If among those alignments there is strong evidence of the location of the cable (mainly two alignments with a great number of pixels lined up and with a high degree of parallelism, even without discounting the perspective effect), then the cable is considered to have been located and its parameters are computed. Otherwise, the image is discarded and the next one is analyzed. Once the cable has been detected, its location and orientation in the next image are predicted by means of a Kalman filter, which allows reducing the pixels to be processed to a small Region of Interest (ROI). In this way, the computation time is considerably lowered together with the probability of misinterpretations of similar features appearing in the image. If low or nil evidence of the pipe/cable presence in the ROI is obtained, a transient failure counter is increased after discarding the image. If this anomalous situation continues throughout too many images, then it is attributed to a failure in the prediction of the ROI, resulting in two special actions: the Kalman filter is reset and the ROI is widened to the whole image. In other words, when faced with a persistent error, the system will no longer make use of the knowledge acquired from the last processed images.

As can be seen in Figure 25, the algorithm computes the polar coordinates (ρ, Θ) of the straight line corresponding to the detected cable in the image plane, (ρ, Θ) being the parameters of the cable line, the cartesian coordinates (x,y) of any point along the line must satisfy

ρ: orthogonal distance between the origin of the image frame and the cable line. Θ: angle between X axis of the image frame and the orthogonal line passing through the origin of the image frame.

$$\rho = x\cos(\Theta) + y\sin(\Theta). \tag{5.1}$$

As shown in Figure 25, Equation (5.1) allows us to obtain the coordinates of the cable intersections with the image boundaries (X_u,Y_u) and (X_L,Y_L), thus the mid point of the straight line (x_g,y_g) can be easily computed $(x_g, y_g = \frac{X_L+X_u}{2}, \frac{Y_u+Y_L}{2})$. The computed parameters (ρ, Θ, x_g, y_g) together with its derivatives are sent to the control module in order to be used by the various behaviors. Figure 26 shows a real image of the Ictineu AUV while detecting a cable.

6 RESULTS

This chapter presents the experimental results of this thesis. The results are organized in a series of sections which intend to show the experimental progression carried out with the various algorithms proposed, using them to solve some well known RL benchmarks and specific underwater tasks. In the first section, a comparison of a VF method and a PG technique is presented. This section compares and analyzes the results obtained by QL and GPOMDP algorithms when trying to solve a common benchmark; the mountain-car problem. In the second section, GPOMDP, demonstrating a better performance than the QL, is tested in a specific simulation. This time, the PG method is programmed using different function approximators to deal with a simulated underwater cable tracking task. The model of the underwater robot Ictineu AUV identified in Section 4 is used by the simulator together with a simulated underwater world. At this point, the results obtained, although being quite good, suggest a search for faster PG algorithms. Therefore, the third section compares the GPOMDP algorithm with an AC; the NAC. This section compares both algorithms when trying to solve another famous benchmark, the cartpole balancing problem. Finally, with the evidence of the results pointing to the NAC as the most suitable algorithm for the kind of underwater tasks presented in this dissertation, the NAC algorithm faces the final experiments. The robot Ictineu AUV performs a real cable tracking task at the CIRS facility. The final experiments presented in this thesis aim to reduce the convergence time of an AC algorithm like NAC combining it with one of the speed up techniques discussed in Section 3.6. The idea is to build a fast, initial policy using a computer simulator which contains an approximation model of the environment for, in the second step, transfer to the Ictineu AUV robot and continuing the learning on-line.

6.1 VF VERSUS PG, THE MOUNTAIN-CAR PROBLEM

This section presents a comparison of a VF method and a PG algorithm in the *mountain-car* benchmark. This simulated

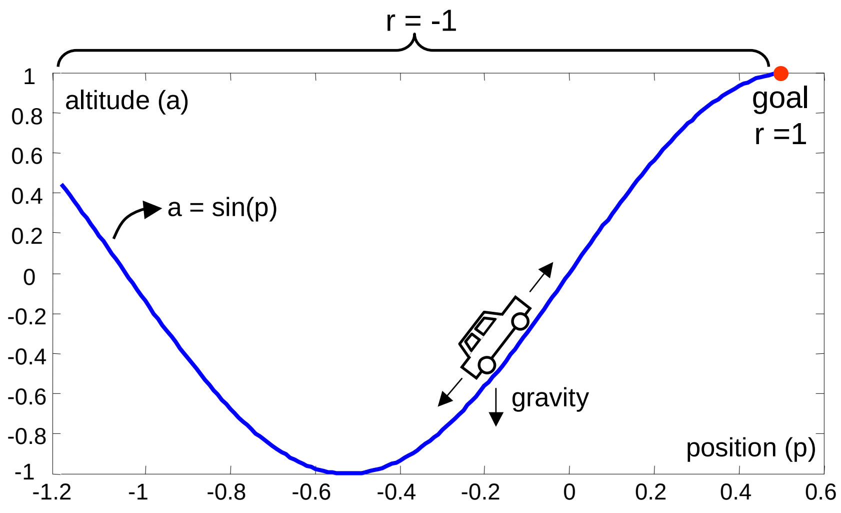

Figure 27: The mountain-car task domain.

problem is well known 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 continuous task like a robot behavior, but an episodic task. Moreover, contrary to a robot behavior, the environment is completely observable and without noise. Hence, this task is highly suitable to test the generalization capabilities of both approaches. The VF method used for these initial experiments is the tabular QL described in Section 2.5.2. The PG method used is the GPOMDP approach described in Section 3.4. This section first describes the "mountain-car" task, then the QL algorithm is applied to the problem. After showing the performance of the QL, the GPOMDP is applied to the same task. Finally, a comparison of the performance of both approaches is given.

6.1.1 *The Mountain-Car task definition*

The mountain-car task [99, 141] was designed to evaluate the convergence and generalization capabilities 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 state of the environment is represented by a two-dimensional vector containing two continuous variables; the position p and the velocity v of the car. The bounds of these variables are $-1.2 \leqslant p \leqslant 0.5$

and $-0.07 \leqslant v \leqslant 0.07$. Action a is a discrete variable with three values $\{-1, 0, +1\}$, which correspond to reverse thrust, no thrust and forward thrust respectively. The mountain geography is described by the equation: $altitude = sin(3p)$. Figure 27 shows the mountain-car scenario. The dynamics of the environment, which determines the state evolution, is defined by these two equations

$$v_{t+1} = \text{bound}[v_t + 0.001\, a_t - 0.0025 \cos(3\, p_t)], \qquad (6.1)$$

and

$$p_{t+1} = \text{bound}[p_t + v_{t+1}], \qquad (6.2)$$

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 the reward 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.

6.1.2 *Results obtained with the QL algorithm*

For the tabular QL approach, the state space has been finely discretized, using 180 states for the position and 150 for the velocity. The action space contained three values $\{-1, 0, +1\}$. Therefore, the *Q table* has 81000 cells. In order to ensure exploration, the policy chosen is an $\epsilon - greedy$ one with ϵ set to 0.3. The discount factor is set to $\gamma = 0.95$ and the learning rate to $\alpha = 0.5$, which have been found experimentally. The experiments have been repeated in 100 independent runs. At the beginning of each run, the Q table is randomly initialized and the learning procedure starts. Since the mountain-car task is an episodic task, the learning is divided into episodes. 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 *effectiveness* of each episode is measured as the number of

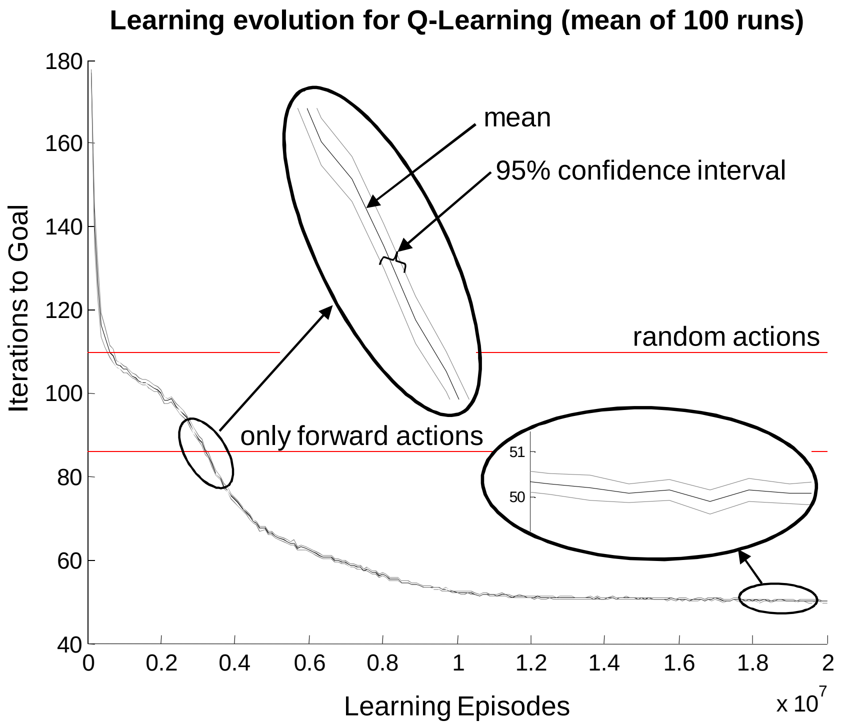

Figure 28: Average learning curve of the QL algorithm with respect to the number of episodes. Results averaged over 100 independent runs. Efficiency levels of random and forward policies can be observed.

iterations needed by the current policy to achieve the goal. Figure 28 shows the learning evolution of the QL algorithm as a mean of 100 independent runs. The 95% confidence intervals are also shown. During the first episodes the efficiency is very low, requiring a lot of iterations to reach the goal. As can be seen, at the end of the learning period, the average number of iterations needed to reach the goal is set around 50 iterations. The average number of episodes needed to achieve this performance is approximately 10^7 episodes. These results show one of the main disadvantages of tabular discretization: each state/action pair must be updated several times until a homogeneous policy is obtained, resulting in a high number of learning iterations.

It is also interesting to compare these results with the ones obtained 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 28). 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.

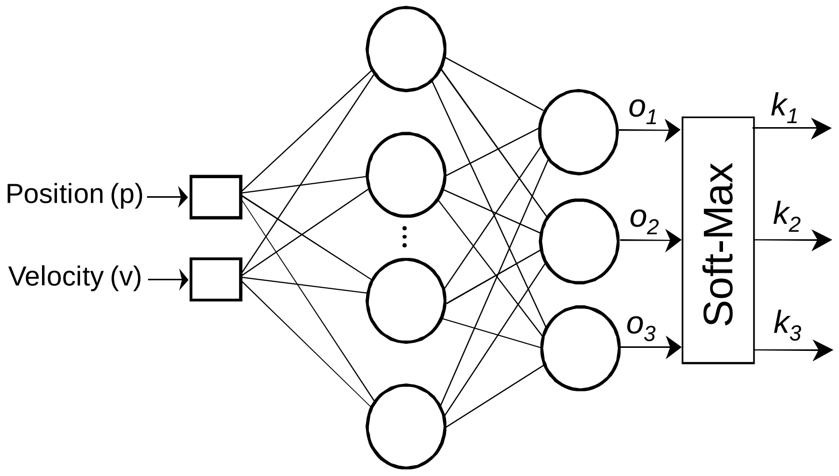

Figure 29: Schematic of the ANN used for the mountain-car task.

6.1.3 *Results obtained with the PG algorithm*

As mentioned at the beginning of this section, the PG algorithm chosen for this comparison is the GPOMDP. The policy is approximated by means of a one-hidden-layer ANN with 2 input nodes, 10 hidden nodes and 3 output nodes (see Figure 29). The state values corresponding to the position p and the velocity v of the car are the inputs of the network. The activation function chosen for the neurons of the hidden layer is of the *hyperbolic tangent* type [58], while the output layer nodes are linear. To ensure exploration, the two output neurons have been exponentiated and normalized to produce a probability distribution. For this purpose, a *soft-max distribution* is evaluated for each possible future state exponentiating the real-valued ANN outputs $\{o1, o2, o3\}$ [1].

After applying the soft-max function, the outputs of the neural network give a weighting $k_i \in (0, 1)$ to each one of the three possible control actions $\{-1, 0, +1\}$. The probability of selecting the *ith* control action is then given by:

$$\Pr_i = \frac{\exp(o_i)}{\sum_{z=1}^{n} \exp(o_z)} \tag{6.3}$$

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. The decay factor of the eligibility is set to $\lambda = 0.98$ and the learning rate to $\alpha = 0.1$, which have been found experimentally. The experiments have been repeated in 100

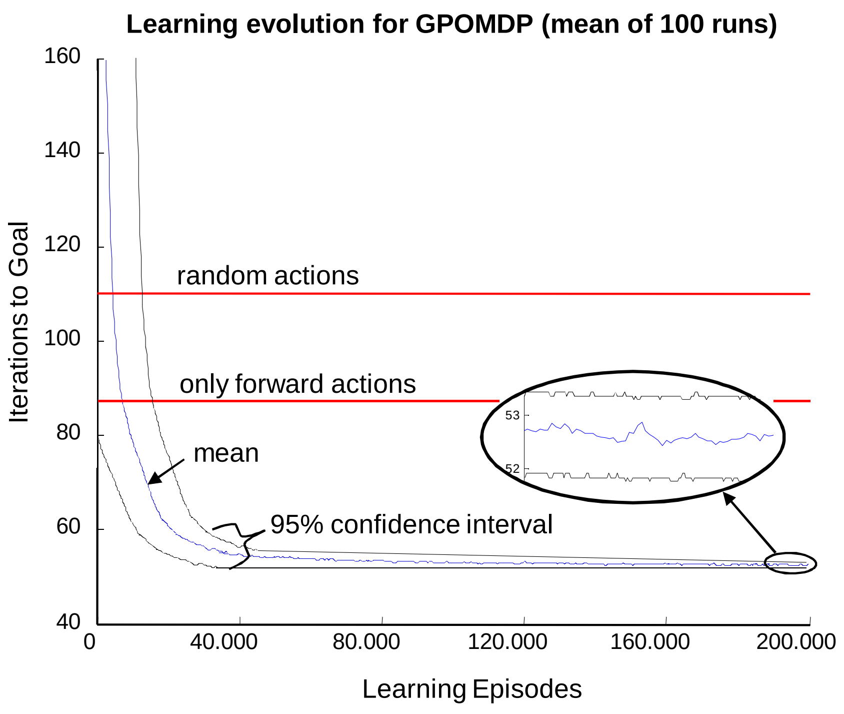

Figure 30: Average learning curve of the PG algorithm with respect to the number of episodes. Results averaged over 100 independent runs. Efficiency levels of random and forward policies can be observed.

independent runs. At the beginning of each run, the ANN weights are randomly initialized and the learning procedure starts. Similar to QL experiments, the effectiveness of each episode is measured as the number of iterations needed by the current policy to achieve the goal. Figure 30 shows the learning evolution of the GPOMDP algorithm as a mean of 100 independent runs. The 95% confidence intervals are shown. Also, the effectiveness of random and forward policies is depicted.

After running 100 independent runs with the PG algorithm, the average number of iterations needed to reach the goal once the optimal policy is learnt is set around 53 iterations, 3 iterations more than the optimal reached with the QL method. The average number of episodes needed to learn this optimal policy is approximately $6x10^4$ episodes, 3 orders of magnitude smaller compared to QL. Observing the amplitude of the 95% confidence intervals, it is important to note the high variability along the runs observed when applying the PG algorithm, especially during the initial episodes of each run.

6.1.4 *Comparative results*

Although the performance of the PG algorithm (average of 53 iterations to goal) is slightly worse than the one obtained with QL (average of 50 iterations to goal), the convergence speed of the GPOMDP method is by far superior to the QL which was heavily affected by the generalization problem. Whereas the PG algorithm achieves a near optimal policy in approximately $6x10^4$ episodes, the QL algorithm needs 3 orders of magnitude more to reach similar results, around 10^7 episodes. In the PG approach, the simplicity of the function approximator chosen to represent the policy is one of its main advantages. A very simple one-hidden-layer ANN configuration with 2 input nodes, 10 hidden nodes and 3 output nodes stores 50 weights. Adding the eligibility trace parameter for each weight, the low total amount of stored data of the algorithm is equal to 100 parameters, which compared to the 81000 cells required by the QL represents a huge difference. The variability of both methods during the initial episodes of each run is worth some attention. The QL method learns slowly but with safe steps towards its optimal policy. As a tabular method, the evolution of the policy from one episode to another is small, achieving uniformly higher effectiveness as the learning goes on and, in the end, obtain an even better policy than the PG. On the other hand, at the beginning of each run, the PG algorithm shows big differences between two consecutive episodes, denoting a highly changing policy which tries to evolve as fast as possible towards the solution, sometimes at the cost of not finding it or getting stuck in a local optima.

The initial simulation results obtained in the mountain-car task encourage more experimentation. A PG technique like that proposed by Baxter and Bartlett definitely outstrips the VF algorithm based on the QL proposed by Watkins. The next section continues with more testing on the PG algorithm and different function approximators dealing with a simulation problem of a cable tracking task. The main objective of this new set of experiments aims to get one step closer to the real robotic task.

6.2 PG METHODS, SIMULATION OF A CABLE TRACKING TASK

This section again presents a simulated application of the GPOMDP algorithm described in Section 3.4. This time, this PG technique deals with an underwater cable tracking task. The simulator has been developed for this thesis with the aim of having a fast platform for testing RL algorithms for robotics in structured underwater conditions. The model of the underwater robot Ictineu AUV identified in Chapter 4 is used in the simulator together with a simulated world corresponding to the pool where all the tests have been carried out in the Computer Vision and Robotics Group facility, CIRS, at the University of Girona (see Section 5.3). The results presented in this section compare two different function approximators applied to the same RLP. First, the policy is approximated using an ANN schema similar to the one used in previous experiments of Section 6.1.3. Next, the same RLP is solved using a barycentric interpolator as the function approximator (see Section 3.3). Comparative results are shown at the end of the section. The next lines give a brief description of the simulator.

6.2.1 *The Ictineu AUV simulated environment*

The model of the underwater robot Ictineu AUV identified in Chapter 4 is used to simulate an underwater robot autonomously navigating a two dimensional world at a predefined constant height above the seafloor. The simulated cable line can be placed at the bottom in any position, allowing the user to work with different cable curvatures and thicknesses. A downward-looking camera model has been used to emulate the vision system of the robot. The objective of this simulation is to center the cable in the image as fast as possible and follow it at a constant predefined forward speed. The state of the environment is represented by a four-dimensional vector containing 4 continuous variables, $(\Theta, \frac{\delta\Theta}{\delta t}, x_g, \frac{\delta x_g}{\delta t})$ where, as can be seen in Figure 31, Θ is the angle between the Y axis of the image plane and the cable, x_g is the X coordinate of the mid point of the cable in the image plane and, finally, $\frac{\delta\Theta}{\delta t}$ and $\frac{\delta x_g}{\delta t}$ are Θ and x_g derivatives respectively (see Section 5.4 for more details). The bounds of these variables are $\frac{-\pi}{2} rads \leqslant \Theta \leqslant \frac{\pi}{2} rads$;

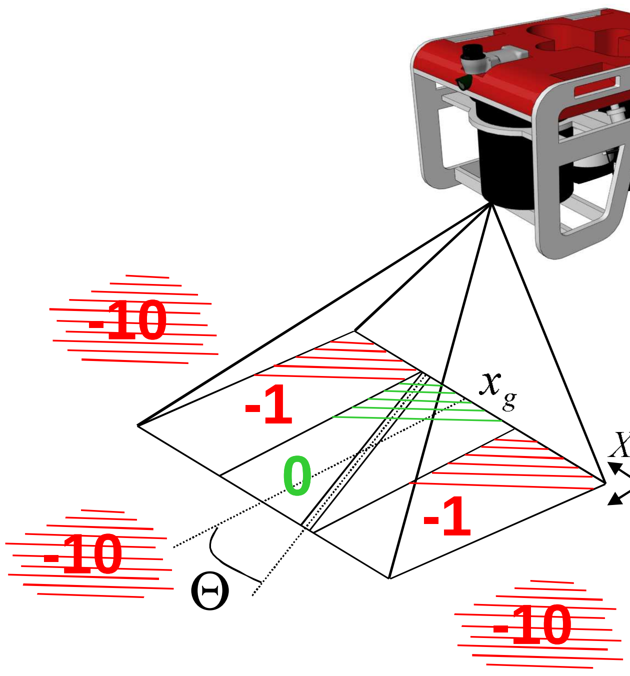

Figure 31: Reward map for the cable tracking task.

$-1\text{rads/s} \leqslant \frac{\delta\Theta}{\delta t} \leqslant +1\text{rads/s}$; $-0.539 \leqslant x_g \leqslant 0.539$ (adimensional bound scaled from the size of the image in pixels) and $-0.5 \leqslant \frac{\delta x_g}{\delta t} \leqslant +0.5$ (adimensional bound derived from x_g). With this input state, the robot makes decisions concerning two DoFs: the Y movement (Sway) and the Z axis rotation (Yaw). Therefore, the continuous action vector is defined as (a_{sway}, a_{yaw}) with boundaries $-1 \leqslant a_{sway} \leqslant +1$ and $-1 \leqslant a_{yaw} \leqslant +1$. The X movement or *Surge* of the vehicle is not learned. A simple controller has been implemented to control the X DoF; whenever the cable is centered in the image plane (the cable is located inside the 0 reward boundaries), the robot automatically moves forward $a_{surge} = 0.3\text{m/s}$. If the cable moves outside the *good* limits or the robot misses the cable, it stops moving forward $a_{surge} = 0\text{m/s}$.

Although the cable tracking task can be understood as a continuous task, for a clear comparison of the results obtained, the robot has been trained in an episodic task. At the beginning of an episode, the robot is placed at a random position and orientation in the vicinity of the cable, assuring any location of the cable within the image plane. An episode ends either every 150 iterations or when the robot loses visual contact with the cable in the image plane. When an episode ends, the robot position is reset to a

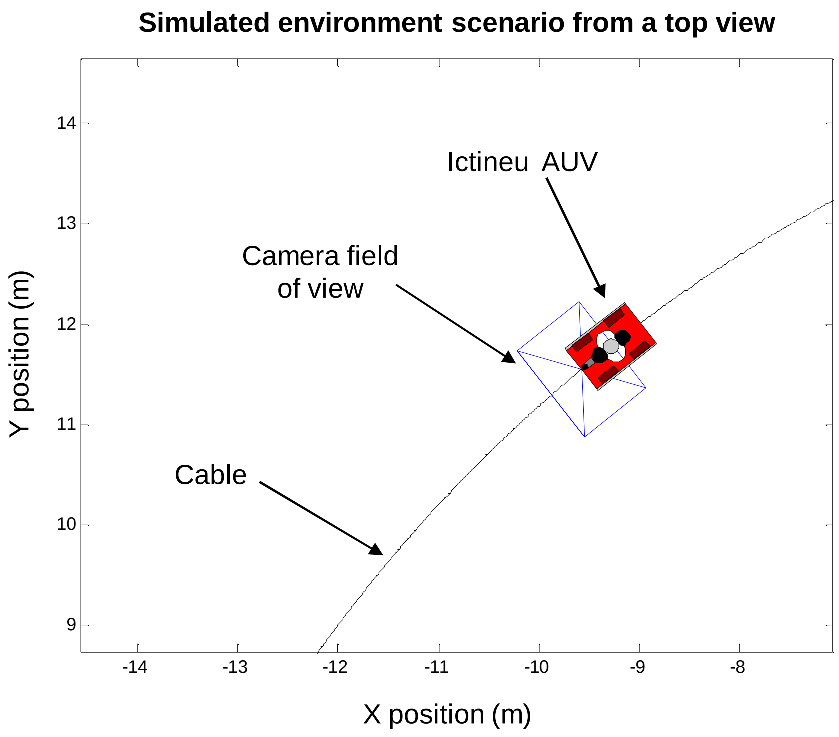

Figure 32: Example situation of the underwater scenario offered by the simulator.

random position and orientation with the cable in the image plane. As can be seen in Figure 31, three discrete values have been chosen for the rewards: $\{-10, -1, 0\}$. A reward of 0 is given whenever the position and the orientation of the cable are inside the boundaries $-0.15 \leqslant x_g \leqslant +0.15$ and $-15^\circ \leqslant \Theta \leqslant +15^\circ$. A reward of -1 is given whenever either the position or the orientation of the cable are in any other location outside the previous bounds but still in the image plane. Finally, a reward of -10 is given when the vehicle misses the target and the episode prematurely ends before 150 iterations are reached. The next lines present the results obtained with the different function approximators. Figure 32 shows the simulated environment scenario.

6.2.2 *Results obtained with the ANN in the cable tracking task*

The update procedures of a PG algorithm aided by an ANN as function approximator have been detailed in Section 3.4.1. For the particular task of cable tracking, the policy is approximated by means of a one-hidden-layer ANN with 4 input nodes, 3 hidden nodes and 2 output nodes (see Figure 33). As stated in the previous section, the state values

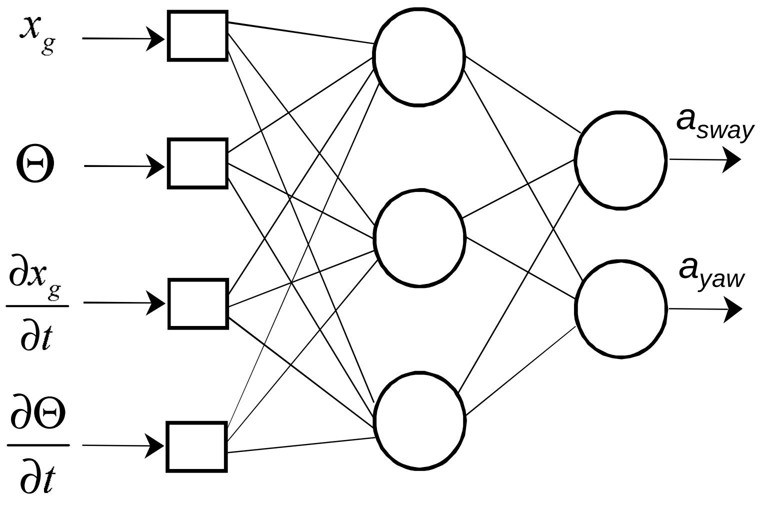
x_g
Θ
$\frac{\partial x_g}{\partial t}$
$\frac{\partial \Theta}{\partial t}$
a_{sway}
a_{yaw}

Figure 33: Schematic of the ANN used for the simulated cable tracking task.

corresponding to the position x_g and the orientation Θ of the cable together with its derivatives $\frac{\delta x_g}{\delta t}$, $\frac{\delta \Theta}{\delta t}$ are the inputs of the network. The activation function chosen for the neurons of the hidden layer is the *hyperbolic tangent* [58], while the output layer nodes are linear. The two output neurons match the continuous action vector (a_{sway}, a_{yaw}) which corresponds to each controlled DoF: Sway (Y movement) and Yaw (Z axis rotation). Both neuron outputs are normalized with boundaries $-1 \leqslant a_{sway} \leqslant +1$ and $-1 \leqslant a_{yaw} \leqslant +1$.

The learning experiments have been repeated in 100 independent runs. At the beginning of each run, the ANN weights are randomly initialized and the learning procedure starts. The effectiveness of each episode is measured as the total reward per episode perceived by the current policy. As described in the previous section, an episode ends either every 150 iterations or when the robot loses visual contact with the cable in the image plane. When an episode ends, the robot position is reset to a random position and orientation with the cable in the image plane. The decay factor of the eligibility is set to $\lambda = 0.98$ and the learning rate to $\alpha = 0.001$, which have been found experimentally. Figure 34 shows the learning evolution of the GPOMDP algorithm using an ANN as function approximator. These results are the mean of 100 independent runs. The 95% confidence intervals are also shown. For every independent run, the maximum number of episodes has been set to 2000. For every episode, the total amount of reward perceived is calculated.

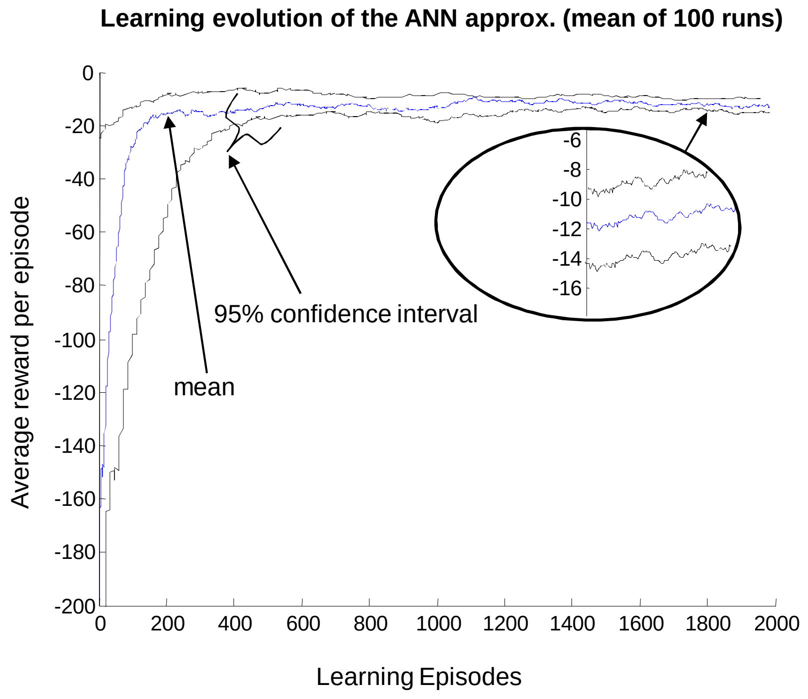

Figure 34: Learning evolution with respect to the number of episodes of the GPOMDP algorithm with the ANN function approximator. Results averaged over 100 independent runs.

After running 100 independent runs with the ANN approximator, the average reward per episode once the optimal policy is learnt is set around -12. The average number of episodes needed to learn this optimal policy is approximately 10^3 episodes. These results show a fast convergence of the algorithm, specially if we compare them to the ones obtained with the same algorithm in the mountain-car task ($6x10^4$ episodes), denoting that this task is, in simulation, easier to learn. The next section explains the performance of the barycentric interpolator approximator on the same task.

6.2.3 *Results obtained with the barycentric interpolator in the cable tracking task*

The update procedures of a PG algorithm aided by a barycentric interpolator as function approximator have been detailed in Section 3.4.2. For the cable tracking task, the observed state is the same used in the previous ANN approach, a 4 dimension vector $(x_g, \Theta, \frac{\delta x_g}{\delta t}, \frac{\delta \Theta}{\delta t})$ corresponding to the position x_g and the orientation Θ of the cable together with

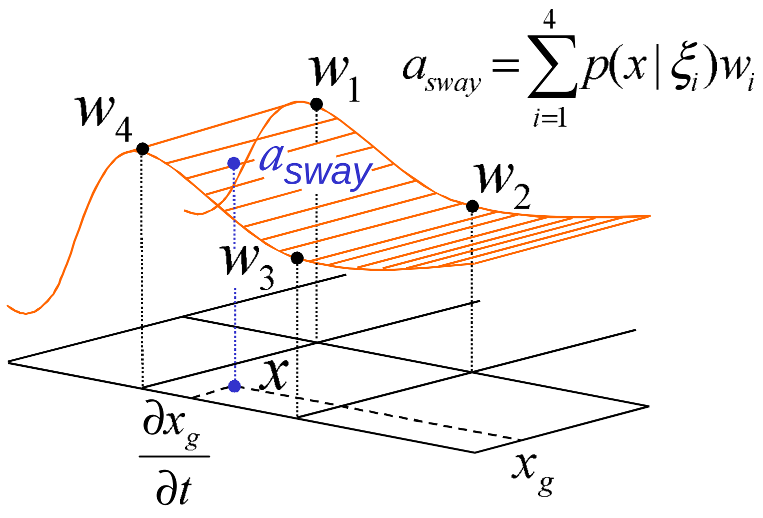
$a_{sway} = \sum_{i=1}^{4} p(x \mid \xi_i) w_i$
w_1
w_2
w_3
w_4
a_{sway}
x
$\frac{\partial x_g}{\partial t}$
x_g

Figure 35: Barycentric interpolator schema for the Sway policy.

its derivatives $\frac{\delta x_g}{\delta t}, \frac{\delta \Theta}{\delta t}$. Also, the continuous output vector of the function are the two control actions, one for each controlled DoF, (a_{sway}, a_{yaw}). With the aim of decreasing the interpolator complexity, the main policy has been split into two subpolicies, each one represented by a 2-dimension barycentric interpolator. a_{sway} actions cause Y displacements on the robot and, therefore, X displacements on the image plane. Then, it can be easily noticed that a_{sway} actions directly affect the position of x_g along the X axis of the image plane together with its derivative. In the same way, a_{yaw} causes rotation around the Z axis and, therefore, Θ angle variations in the image plane. Although learning uncoupled policies certainly reduces the overall performance of the robot, this method greatly reduces the total number of stored parameters, making it easier to implement.

In Figure 35, the observed values of the state x_g and its derivative $\frac{\delta x_g}{\delta t}$ are the input for the Sway policy. The output of this policy is the action a_{sway}. In the same way, as can be seen in Figure 36, the state value Θ and its derivative $\frac{\delta \Theta}{\delta t}$ are the input for the Yaw policy, a_{yaw} being its action output. The state space density of the mesh for both grids has been experimentally set to 10 equal divisions for each axis, therefore the mesh for each policy has 100 cells.

In order to obtain comparative results, the learning procedures are identical to those followed with the ANN approach. This way, the experiments have been repeated in 100 independent runs. At the beginning of each run, the values w_i of the interpolator are randomly initialized and the learn-

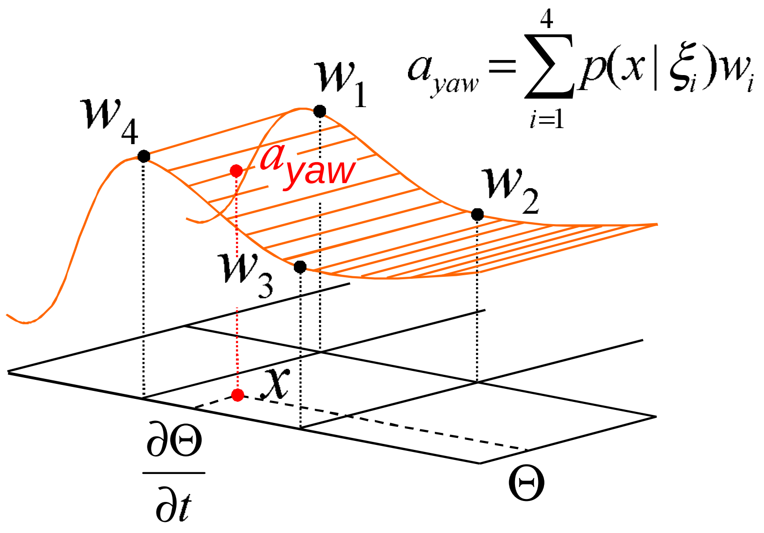
$a_{yaw} = \sum_{i=1}^{4} p(x|\xi_i)w_i$
w_1
w_2
w_3
w_4
a_{yaw}
x
$\frac{\partial\Theta}{\partial t}$
Θ

Figure 36: Barycentric interpolator schema for the Yaw policy.

ing procedure starts. The effectiveness of each episode is measured as the total reward per episode perceived. Even though the barycentric algorithm uses two policies, one for the Sway and another for the Yaw, there is only one reward function, defined in Figure 31, which gives the same discrete values to both policies. As described in the previous section, an episode ends either every 150 iterations or when the robot loses visual contact with the cable in the image plane. When an episode ends, the robot position is reset to a random position and orientation with the cable in the image plane. The decay factor of the eligibility is set to $\lambda = 0.98$ and the learning rate to $\alpha = 0.001$, which have been found experimentally. Figure 37 shows the learning evolution of the GPOMDP algorithm using a barycentric interpolator as the function approximator. These results are the mean of 100 independent runs. The 95% confidence intervals are also shown. For every independent run, the maximum number of episodes has been set to 2000. For every episode, the total amount of reward perceived is calculated.

After running 100 independent runs with the barycentric interpolator approximator, the average reward per episode once the optimal policy is learnt is set around -10, slightly better than the average reward reached by the ANN (best value about -12). The average number of episodes needed to learn this optimal policy is set around 10^3 episodes, approximately equal to that of the ANN. These results show a convergence speed similar to the one achieved by the

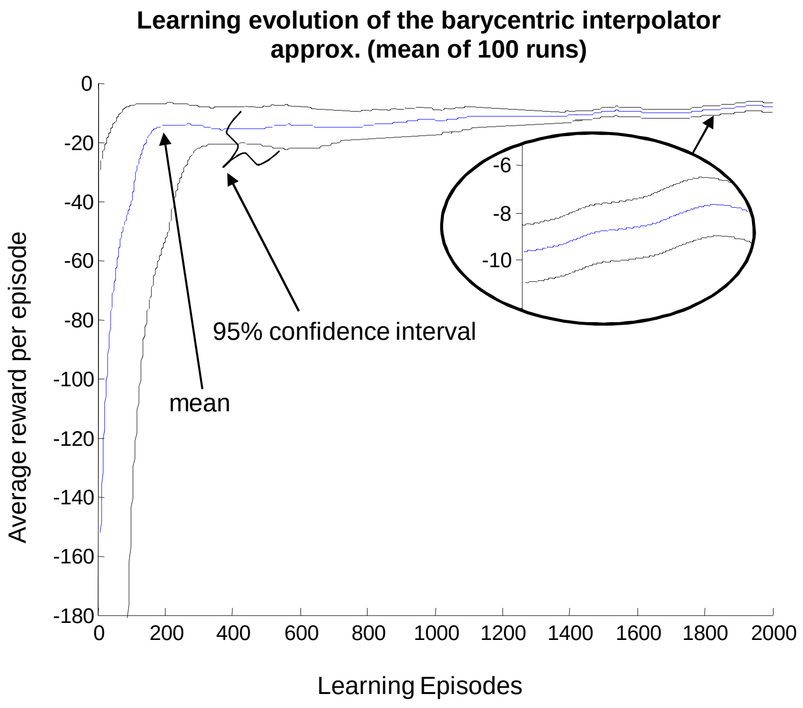

Figure 37: Learning evolution with respect to the number of episodes of the GPOMDP algorithm with the barycentric interpolator function approximator. Results averaged over 100 independent runs.

ANN approximator. The next section compares the results obtained with both methodologies.

6.2.4 *Comparative results*

Figure 38 compares the results obtained with both approximators. As can be seen, the performance reached by the barycentric interpolator approximator (average reward of -10 per episode) is slightly better than the one obtained with the ANN approximator (average reward of -12 per episode) while the convergence speed of both methodologies is similar, around 10^3 episodes. Although the results achieved with the barycentric interpolator are quite better, the simplicity and the total amount of stored data of the ANN function approximator favor the last one. The ANN used is a very simple one-hidden-layer ANN configuration with 4 input nodes, 3 hidden nodes and 2 output nodes stores 18 weights. Adding the eligibility trace parameter for each weight, the total amount of stored data for the ANN approach is equal to only 36 parameters. On the other hand, the barycentric approach uses two independent policies, one mesh of 100

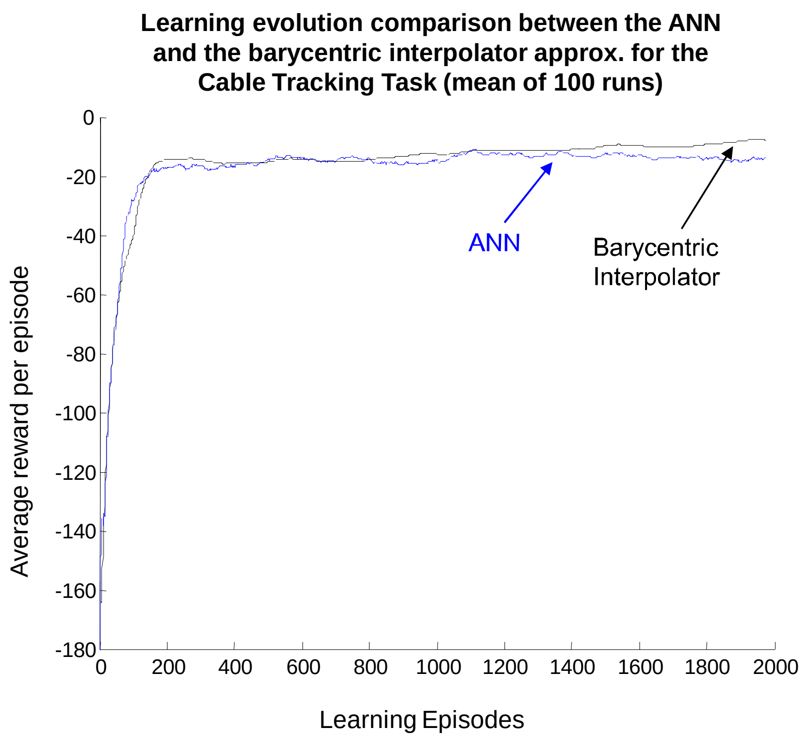

Figure 38: Comparison of the simulated results obtained when using an ANN and as barycentric interpolator with the GPOMDP in a Cable Tracking Task. Results averaged over 100 independent runs.

cells for the Yaw DoF and another for the Sway DoF. As detailed in Section 3.4.2, the approximator parameters to be updated at each iteration are located at the intersections of the mesh, having a total of 121 parameters for each mesh, 242 needed for both policies. Adding the eligibility trace for each parameter, the total amount of stored data for the barycentric interpolator approach is equal to 484 parameters, which compared to the 36 parameters required by the ANN represents a huge difference.

The simulation results obtained in the cable tracking task using two different function approximators do not show great differences between them. A general overview of the results obtained demonstrate that the GPOMDP approach is able to converge to an optimal or near optimal policy for this particular simulation of a cable tracking task. Going one step further, a deep analysis of the data suggests that the application of a *pure* PG algorithm in a real task can be slow, unstable and rigid, with low capabilities of fast adaptation to changing conditions in the real world.

6.3 PG VERSUS AC, THE CARTPOLE PROBLEM

With the aim of finding faster and more flexible algorithms, this section presents another comparison. This time, the GPOMDP algorithm faces the NAC algorithm in the *Cartpole Balancing* benchmark. As discussed in Section 2.8, AC methods try to combine the advantages of PG algorithms with VF methods. The actor's PG gives convergence guarantees while the critic's VF reduces variance of the policy update improving the convergence rate of the algorithm. The result is a very fast algorithm with high adaptation capabilities. Aiming for the best comparison, both approaches use the same function approximator, a linear parameterized basis function, to represent the policy. This section first describes the cartpole task. Then, the GPOMDP algorithm is applied to the problem. After showing the performance of a pure PG method, the NAC is applied to the same task. Finally, a comparison of the performance of both approaches is given.

6.3.1 *The Cartpole Balancing definition*

The cartpole balancing problem, like the mountain-car task, is another well-known benchmark in RL [152, 137]. As with the mountain-car task, it has been designed to evaluate the the convergence and generalization capabilities of RL algorithms. In the cartpole balancing task, a pole with a point-mass on its upper end is mounted on a cart acting as an inverted pendulum. Control actions move the car forward and backward in order to stabilize the pole on its top. Whereas a normal pendulum is stable when hanging downwards, an inverted pendulum is inherently unstable and must be actively balanced in order to remain upright, either by applying a torque at the pivot point or, as in this case, by moving the pivot point horizontally as part of a feedback system. Figure 39 shows the cartpole scenario.

The cartpole dynamics have been linearized along the vertical point of the pole [116]. The state vector x is represented by $x = (p, \dot{p}, \theta, \dot{\theta})$ containing four continuous variables: the position p of the cart and its derivative $\dot{p}$ together with the pole angle θ and its derivative $\dot{\theta}$. The bounds of the position p of the cart is $-1.5 \leqslant p \leqslant 1.5$ while the pole angle limits are $-\pi/6 \leqslant \theta \leqslant \pi/6$. Action a is a continuous variable

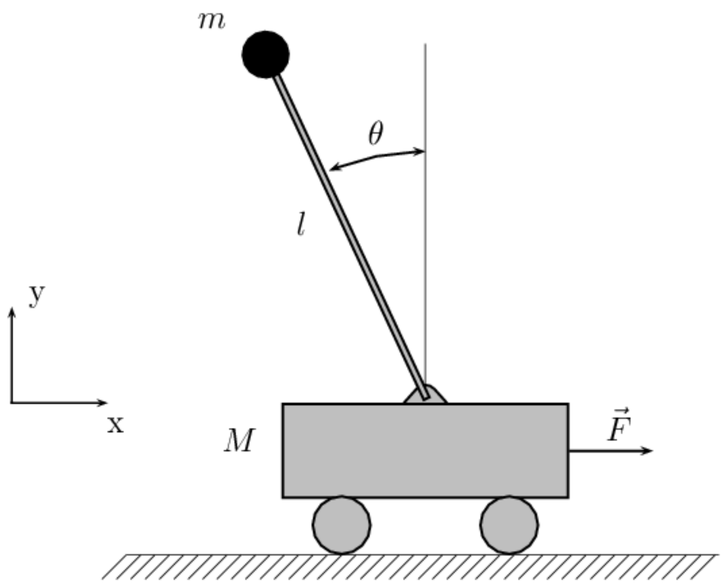

Figure 39: The cartpole balancing task domain.

with bounds $-1 \leqslant a \leqslant 1$. The cartpole dynamics, which determines the state evolution, is defined by

$$\begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & \tau & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \tau \\ 0 & 0 & \upsilon\tau & 1 \end{bmatrix} \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix}_{t} + \begin{bmatrix} 0 \\ \tau \\ 0 \\ \upsilon\tau/g \end{bmatrix} a. \quad (6.4)$$

Here $\tau = 1/60s$, $\upsilon = 13.2s^{-2}$ and $g = 9.81m/s^2$. The reward function is a continuous negative one, depending on the state and the action taken, given by the expression $r(x,a) = -x^TQx - a^TRa$, Q and R being scalar matrixes experimentally tuned to adjust the weight of the state and action values in the final reward contribution. The learning algorithm is trained in an episodic task. An episode ends either when the cart or the pole leave its boundaries or when a maximum of 200 iterations have elapsed. Whenever this happens, a new episode is started at a random position and velocity around the zero vector. This problem is sufficiently difficult to serve as a reference for this comparison.

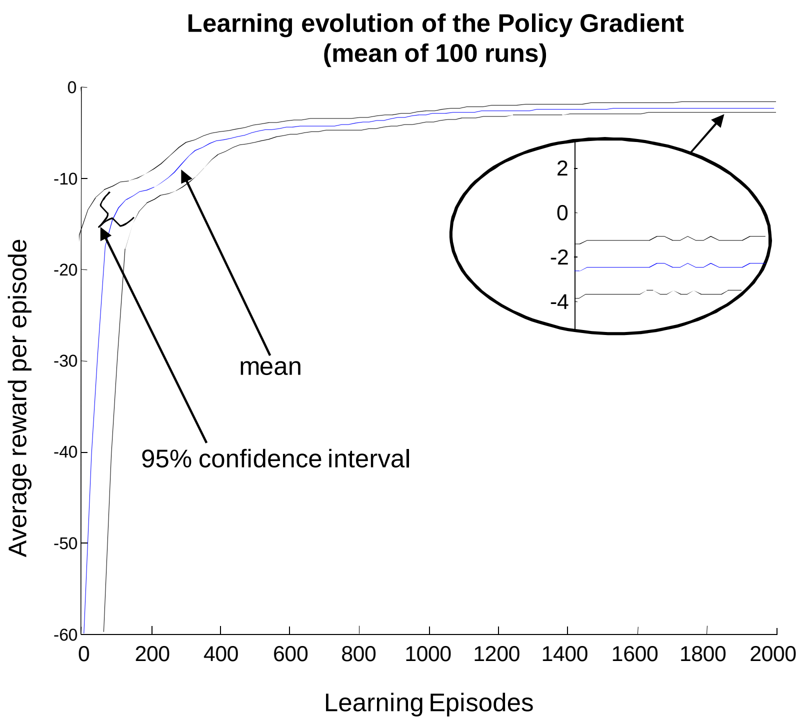

Figure 40: Learning evolution with respect to the number of episodes of the GPOMDP algorithm in the cartpole balancing task. Results averaged over 100 independent runs.

6.3.2 *Results obtained with the PG algorithm*

The PG algorithm chosen for this comparison is again the GPOMDP algorithm. As stated at the beginning of this section, the policy is approximated by means of a parameterized *basis function*. The policy is specified as $\pi(x, a, K) = N(a|Kx, \sigma^2)$, where its output is the result of a linear combination of the state vector values and the learned parameters vector K, following the expression $a = k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4$. A Gaussian distribution with the basis function output as mean and a fixed variance of 0.1 is used to insure some stochasticity around a chosen action.

The decay factor of the eligibility is set to $\lambda = 0.9$ and the learning rate to $\alpha = 0.01$, which have been found experimentally. Weight matrixes for the reward calculation have been set to $Q = diag(1.25, 1, 12, 0.25)$ and $R = 0.01$ as in [116]. The experiments have been repeated in 100 independent runs. At the beginning of each run, the policy parameter vector K is randomly initialized and the learning procedure starts. The effectiveness of each episode is measured as the total reward per episode perceived by the

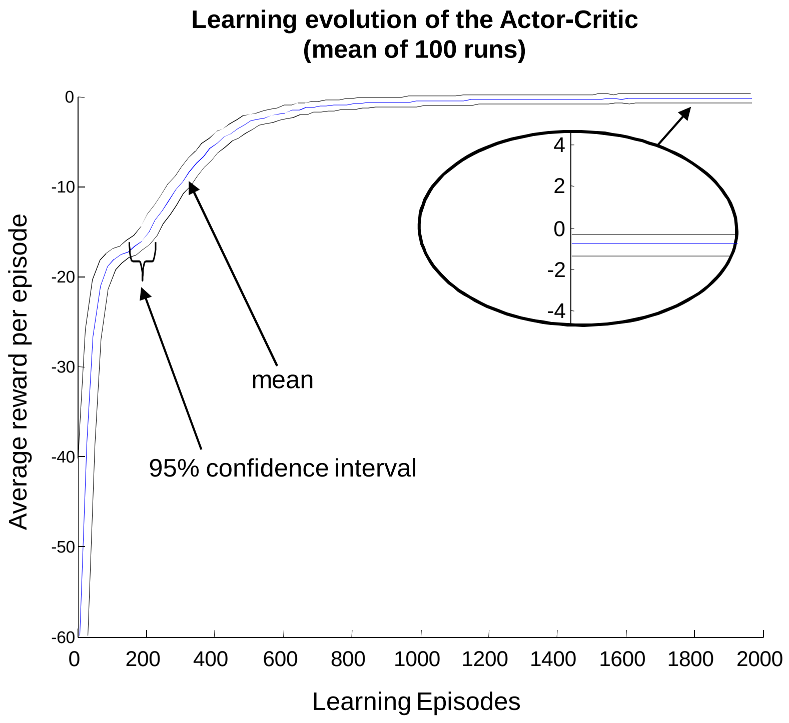

Figure 41: Learning evolution with respect to the number of episodes of the AC algorithm in the cartpole balancing task. Results averaged over 100 independent runs.

current policy. As described previously, an episode ends either every 200 iterations or when the pole falls down and the episode resets. Figure 40 shows the learning evolution of the GPOMDP as a mean of 100 independent runs. The 95% confidence intervals are shown.

After running 100 independent runs with the PG algorithm, the average reward per episode once the optimal policy is learnt is set around -3. The average number of episodes needed to learn this optimal policy is approximately 10^3 episodes.

6.3.3 *Results obtained with the AC algorithm*

The AC algorithm chosen for this comparison is the NAC algorithm described in Section 3.5. This algorithm has already been applied in simulation to solve the cartpole task in [116]. Those experiments are reproduced in this dissertation with minor changes in order to obtain comparative results between AC techniques and pure gradient algorithms. The part of the algorithm corresponding to the actor is represented by the same Gaussian distribution described in the previous approach $\pi(x, a, K) = N(a|Kx, \sigma^2)$. The basis function approximator combines the state vector values and the learned parameters vector K, following the expression $a = k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4$. The variance σ^2 of the distribution is fixed at 0.1. Additional basis functions are chosen to represent the critic. These basis functions selected for the comparison are the same ones used in [116] $\phi(x) = [x_1^2, x_1x_2, x_1x_3, x_1x_4, x_2^2, x_2x_3, x_2x_4, x_3^2, x_3x_4, x_4^2, 1]$.

The decay factor of the eligibility is set to $\lambda = 0.8$ and the learning rate to $\alpha = 0.01$, which have been found experimentally. A discount factor for the averaged reward is set to $\gamma = 0.95$ and a forgetting factor for the matrixes A and b has been set to $\beta = 0.9$. Weight matrixes for the reward calculation have been set to $Q = diag(1.25, 1, 12, 0.25)$ and $R = 0.01$ as in [116]. For every iteration, the policy is updated only when the angle $\angle(w_{t+1}, w_t) \leqslant \epsilon = \pi/180$. The experiments have been repeated in 100 independent runs. At the beginning of each run, the policy parameter vector K is randomly initialized and the learning procedure starts. The effectiveness of each episode is measured as the total reward per episode perceived by the current policy. As described previously, an episode ends either every 200 iterations or when the pole falls down and the episode resets. Figure 41 shows the learning evolution of the GPOMDP algorithm as a mean of 100 independent runs. The 95% confidence intervals are shown.

After running 100 independent runs with the NAC algorithm, the average reward per episode, once the optimal policy is learnt, is set around -1, evidencing slightly better results than the ones obtained with the GPOMDP, which got their best mark at -3. The average number of episodes needed to learn this optimal policy is approximately 8x10^2 episodes, around 200 episodes faster than the PG. The next section extracts some conclusions about both methods for this particular task.

6.3.4 *Comparative results*

Figure 42 compares the results obtained with the NAC and the GPOMDP algorithm in the cartpole balancing task. As can be seen, the performance reached by the NAC (average reward of -1 per episode) is slightly better than the one obtained with the GPOMDP algorithm (average reward of -3 per episode). Also, the convergence speed of the NAC

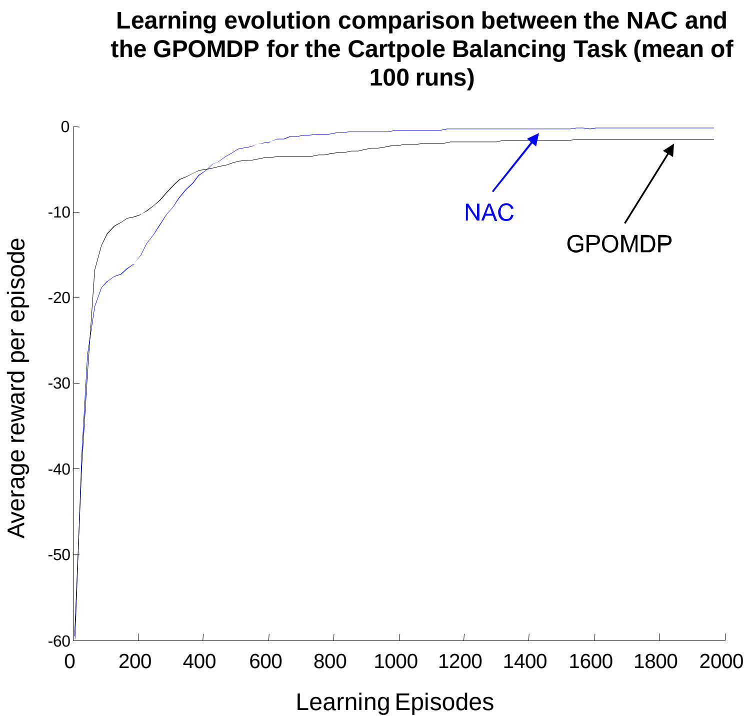

Figure 42: Learning evolution comparison with respect to the number of episodes between the NAC and the GPOMDP algorithm. Results averaged over 100 independent runs.

(around 8x10^2 episodes) is higher than the one obtained with the GPOMDP algorithm (around 10^3 episodes). These performance results clearly favor the NAC. With regard to data storage and computational requirements, independently from the function approximator chosen, AC methods require more storage space and calculation than a simple PG algorithm because, apart from the actor's parameters, the critic has its own set of parameters to be updated at each iteration step. Even so, the difference between the number of parameters used by both methods is not very high and do not represent a drawback for the final on-line learning purpose of this dissertation.

Paying attention to the 95% confidence intervals illustrated in Figures 40 and 41, the variability of both methods over 100 runs again favors the NAC. The AC method has smaller variance and presents solid learning curves, all very similar, from the first run to the last. On the other hand, the GPOMDP algorithm shows great differences between the different runs, especially at the beginning of each one, denoting a fragile and unstable learning curve with a high risk of getting stuck in a local optima.

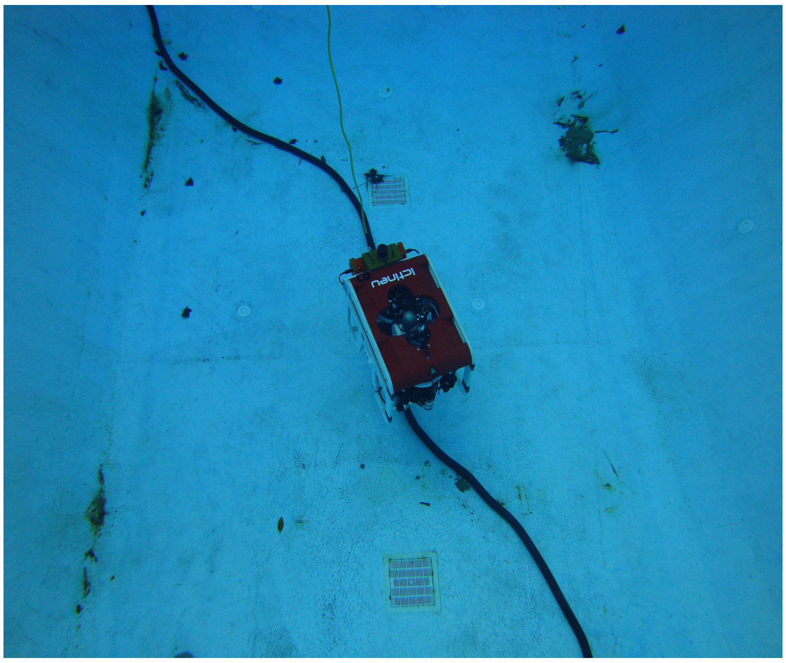
Figure 43: Image of Ictineu AUV in the CIRS facility while attempting a cable tracking task. The cable is placed diagonally to exploit the maximum length of the bottom of the pool.

6.4 ictineu auv learns a real underwater task

Definitively, the experimental results obtained in previous trial after applying the different methodologies described in Section 3.4 and Section 3.5, give the AC methods a great advantage over pure PG techniques. Although the convergence speed achieved by the AC algorithm is notably higher than the one showed by the GPOMDP approach, for the kind of RLPs solved in this dissertation, it may not be enough. The final experimental proposal presented in this thesis aims to reduce the convergence times of an AC algorithm like the NAC algorithm combining it with one of the speed up techniques discussed in Section 3.6. The idea is to build a fast initial policy using a computer simulator which contains an approximation model of the environment for, in the second step, transfer it to the Ictineu AUV to continue the learning on-line. The RLP to solve is the underwater cable tracking task. The artificial underwater scenario has been built at the CIRS facility at the University of Girona (see Figure 43), described in Section 5.3.

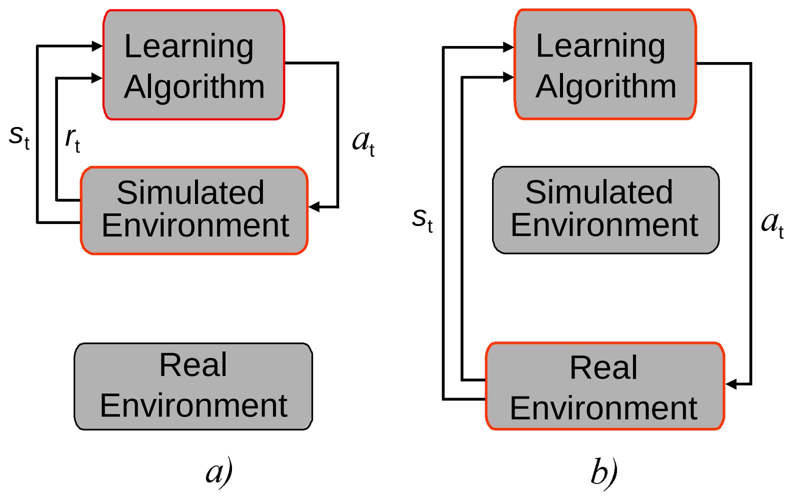

Figure 44: Learning phases.

6.4.1 *The "two-step" learning approach*

As can be seen in Figure 44, the learning process of the proposed method is split into two steps. First, as shown in Figure 44(a), the learner interacts with the simulator, building an initial policy. During this phase, the NAC algorithm is trained in the simulator. The simulated experiments begin by setting parameterized initial basis functions for both, the actor and the critic. An approximated model of the environment emulates a robot with the same DoFs as the real one. The sampling time of the simulator is set equal to the one offered by the real robot. Actions, forces, accelerations and velocities are scaled to match the real ones. As shown in Figure 44(b), once the simulated results are accurate enough and the algorithm has acquired enough knowledge from the simulation to build a secure policy, in the second step, the learned-in-simulation policies are transferred to the robot and step two starts. The actor and critic parameters are transferred to the real robot to continue the learning process on-line in a real environment, improving the initial policy. During the second step, the NAC algorithm continues its execution, this time, in the software architecture of the robot. Since the simulated model and environment are just approximations, the algorithm will have to adapt to the real world. An important idea is that, once the algorithm converges in the real environment, the learning system will continue working forever, being able to adapt to any future change.

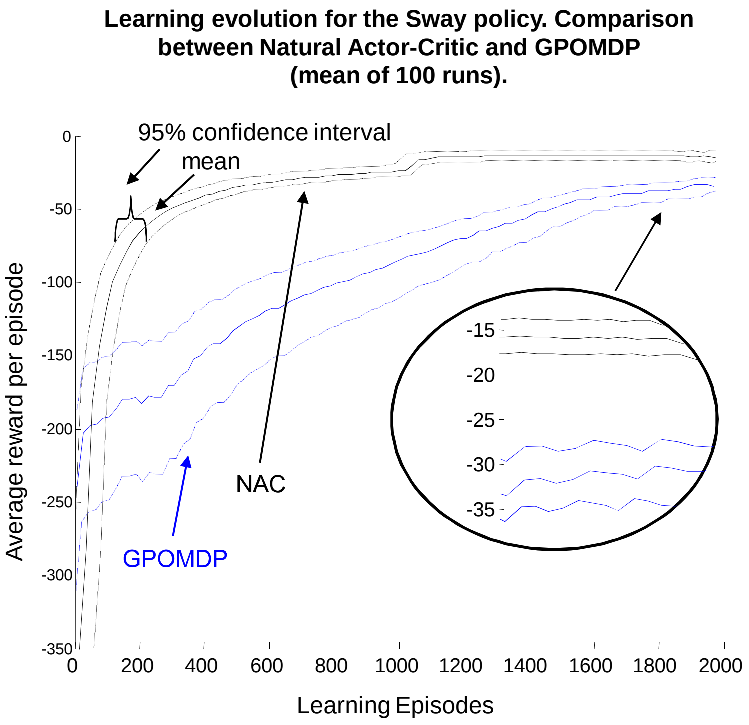

Figure 45: Learning evolution for the Sway policy. Comparison with respect to the number of episodes between the NAC and the GPOMDP algorithms. Results averaged over 100 independent runs.

6.4.2 *NAC algorithm configuration*

For both steps, the simulated and the real one, the state of the environment is represented by a four-dimensional vector containing 4 continuous variables, $(\Theta, \frac{\delta\Theta}{\delta t}, x_g, \frac{\delta x_g}{\delta t})$ where Θ is the angle between the Y axis of the image plane and the cable, x_g is the X coordinate of the mid point of the cable in the image plane and, finally, $\frac{\delta\Theta}{\delta t}$ and $\frac{\delta x_g}{\delta t}$ are Θ and x_g derivatives respectively. The bounds of these variables are $\frac{-\pi}{2}rads \leqslant \Theta \leqslant \frac{\pi}{2}rads$; $-1rads/s \leqslant \frac{\delta\Theta}{\delta t} \leqslant +1rads/s$; $-0.539 \leqslant x_g \leqslant 0.539$ (adimensional bound scaled from the size of the image in pixels) and $-0.5 \leqslant \frac{\delta x_g}{\delta t} \leqslant +0.5$ (adimensional bound derived from x_g). From this input state, the robot makes decisions concerning two DoFs: the Y movement (Sway) and the Z axis rotation (Yaw). Therefore, the continuous action vector is defined as (a_{sway}, a_{yaw}) with boundaries $-1 \leqslant a_{sway} \leqslant +1$ and $-1 \leqslant a_{yaw} \leqslant +1$. The X movement or *Surge* of the vehicle is not learned. A simple controller has been implemented to control the X DoF; whenever the cable is centered in the image plane (the

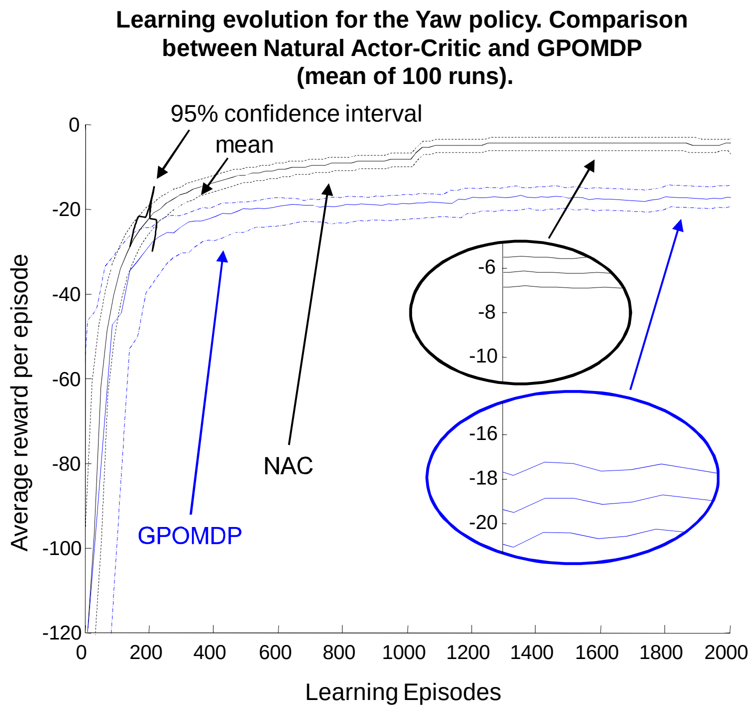

Figure 46: Learning evolution for the Yaw policy. Comparison with respect to the number of episodes between the NAC and the GPOMDP algorithms. Results averaged over 100 independent runs.

cable is located inside the 0 reward boundaries) the robot automatically moves forward $a_{surge} = 0.3m/s$. If the cable moves outside the *good* limits or the robot misses the cable, it stops moving forward $a_{surge} = 0m/s$.

The main policy has been split into two subpolicies. a_{sway} actions cause Y displacements on the robot and, therefore, X displacements on the image plane. Then, it can be easily noticed that a_{sway} actions directly affect the position of x_g along the X axis of the image plane together with its derivative. In the same way, a_{yaw} causes rotation around the Z axis and, therefore, Θ angle variations in the image plane. Although learning uncoupled policies certainly reduces the overall performance of the robot, this method greatly reduces the total number of stored parameters, making it easier to implement. The actor policies for Yaw and Sway are described as normal gaussian distributions of the form $\pi(x, a, K) = N(a|Kx, \sigma^2)$, where the variance σ^2 of the distribution is fixed at 0.1. The basis function approximator combines the state vector values and the learned parameters vector K, following the expression $a = k_1 x_1 + k_2 x_2$. The policy parameter vectors K for Sway and Yaw policies are

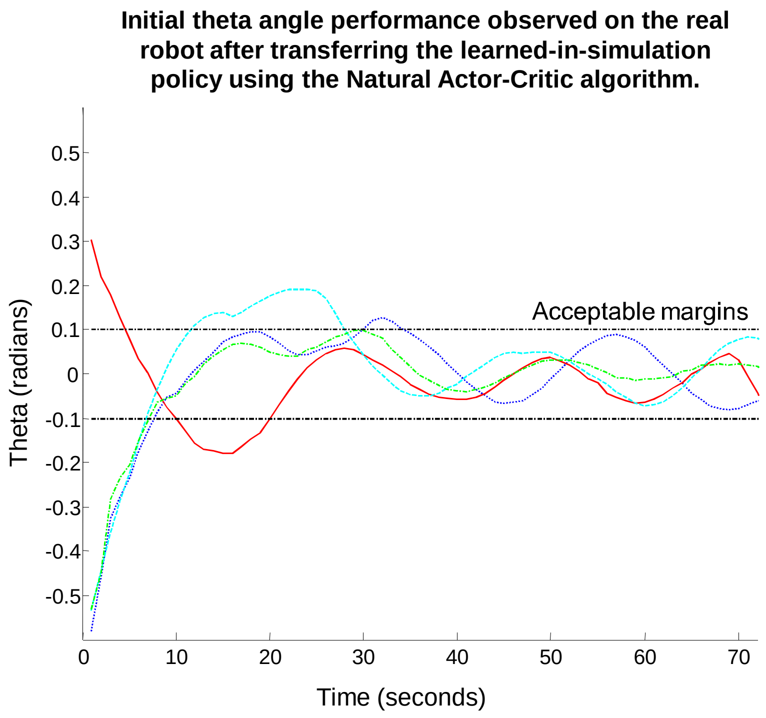

Figure 47: Real Θ angle measurements extracted from various independent samples. These results show the initial performance of the learned-in-simulation policies transferred to the real robot.

randomly initialized at the beginning of the experiment. Additional basis function for Sway and Yaw critic parameterization are chosen as $\phi(x) = [x_1^2, x_1x_2, x_2^2, 1]$, which experimentally have proven to be good enough to represent both functions.

A continuous function has been designed to compute specific rewards for each learned policy. Rewards are given by $r(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$ with $Q_{sway} = diag(1.1, 0.2)$ and $R_{sway} = 0.01$ for Sway DoF and $Q_{Yaw} = diag(2, 0.25)$ and $R_{Yaw} = 0.01$ for the Yaw DoF. Q's and R's values have been tuned experimentally. The learning parameters have the same value for both policies. The learning rate has been fixed to $\alpha = 0.01$ and the decay factor of the eligibility has been set to $\lambda = 0.8$. A discount factor for the averaged reward is set to $\gamma = 0.95$ and a forgetting factor for the matrixes A and b has been set to $\beta = 0.9$. For every iteration, the policy is updated only when the angle between to consecutive gradient vectors accomplishes $\angle(w_{t+1}, w_t) \leqslant \epsilon = \pi/180$.

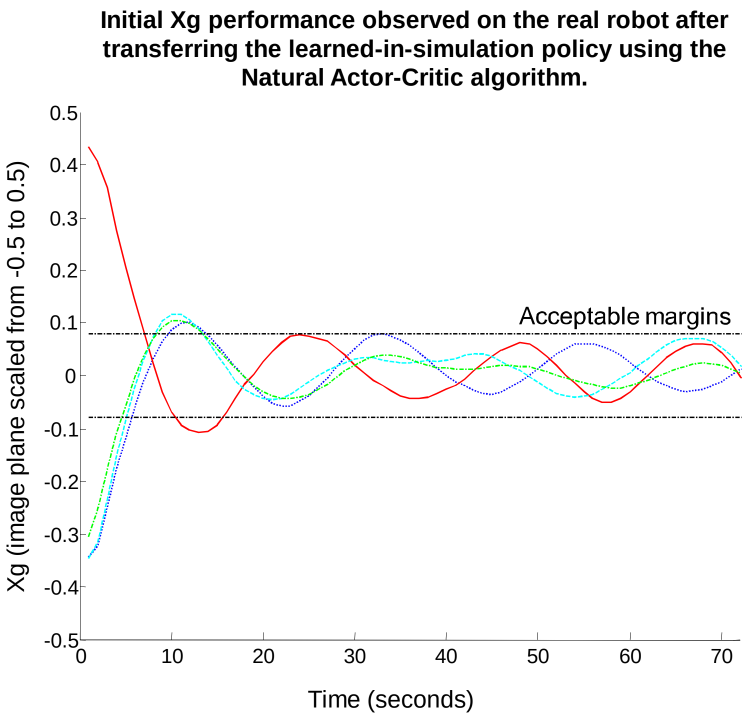

Figure 48: Real x_g measurements extracted from various independent samples. These results show the initial performance of the learned-in-simulation policies transferred to the real robot.

6.4.3 *GPOMDP algorithm configuration*

For performance comparison purposes, the results obtained with the NAC algorithm during the simulated step 1 are compared with the GPOMDP tested in the same conditions. The GPOMDP algorithm has been implemented using barycentric interpolators as function approximators, and, similarly to the NAC, the main policy has been split into two subpolicies. The state vector input of the Sway policy is $(x_g, \frac{\delta x_g}{\delta t})$ while the state vector input of the Yaw policy is $(\Theta, \frac{\delta\Theta}{\delta t})$. Each policy outputs a continuous scalar value which represents the action to be performed, a_{sway} for the Sway policy output and a_{yaw} for Yaw. Differing from the experiments in Section 6.2.3, where the algorithm performs the same simulated task with two different function approximators, the reward function used this time is not discrete. Instead, the PG method uses the same continuous reward function implemented in the NAC, given by $r(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$ with $Q_{sway} = diag(1.1, 0.2)$ and $R_{sway} = 0.01$ for the Sway DoF and $Q_{Yaw} = diag(2, 0.25)$ and $R_{Yaw} = 0.01$ for the Yaw DoF. The utilization of the same reward function allows

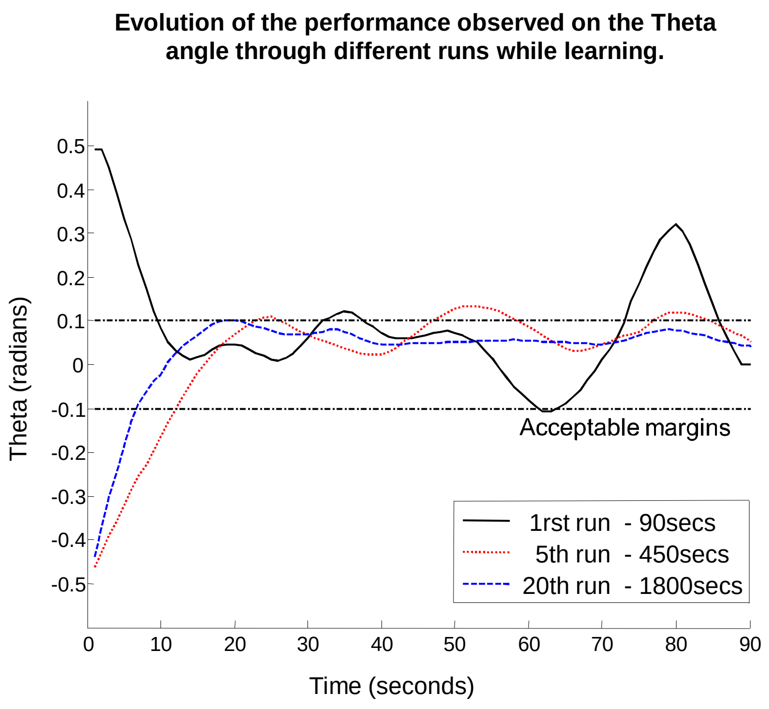

Figure 49: Policy improvement over different runs of real learning. Evolution of Θ angle performance.

comparative results between these two algorithms but gives different final results from those obtained in Section 6.2.3.

6.4.4 *Step 1, simulated learning results*

The computed model of the underwater robot Ictineu AUV navigates a two dimensional world at a 0.8 meter height above the seafloor. The simulated cable is placed on the bottom in a fixed position. The robot has been trained in an episodic task. An episode ends either after 150 iterations or when the robot misses the cable in the image plane, whatever comes first. When the trial ends, the robot position is reset to a random position and orientation around the cable's location, allowing any location of the cable within the image plane at the beginning of each trial. According to the value of the state and action taken, a scalar immediate reward is given at each iteration step.

The experiments in the simulation step have been repeated in 100 independent runs. At the beginning of each run, the policy parameters are randomly initialized for each one of the policies and the learning procedure starts. The effectiveness of each episode is measured as the total reward per episode perceived by the current policy. Figure 45

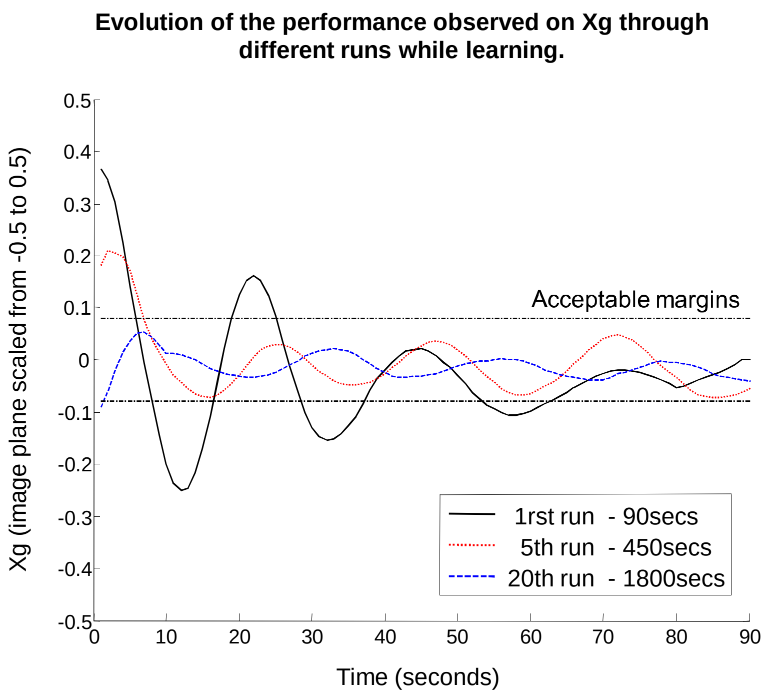

Figure 50: Policy improvement over different runs of real learning. Evolution of x_g angle performance.

shows the learning evolution of the Sway policy applying the NAC algorithm as a mean of 100 independent runs. The learning curve obtained under the same conditions with the GPOMDP algorithm is also depicted. The 95% confidence intervals for both curves are shown. After running 100 independent runs with the NAC algorithm, the average reward per episode once the optimal Sway policy is learnt is set around -15, notably better results than the ones obtained with the GPOMDP, which got their best mark at -35. The average number of episodes needed by the NAC to learn this Sway optimal policy is approximately $11x10^2$ episodes, while the GPOMDP needs twice the number of episodes to obtain its best result, which is still worse than the best one obtained by the NAC.

Figure 46 shows the same learning comparison, but this time for the learning evolution of the Yaw policy. The average reward per episode obtained by the AC algorithm once the optimal Sway policy is learnt is set around -6. Again these results are better than the ones obtained with the PG, which got their best mark at -19. The average number of episodes needed by the NAC to learn this Yaw optimal policy is approximately $14x10^2$ episodes, while the GPOMDP reaches average reward values close to its best mark at rel-

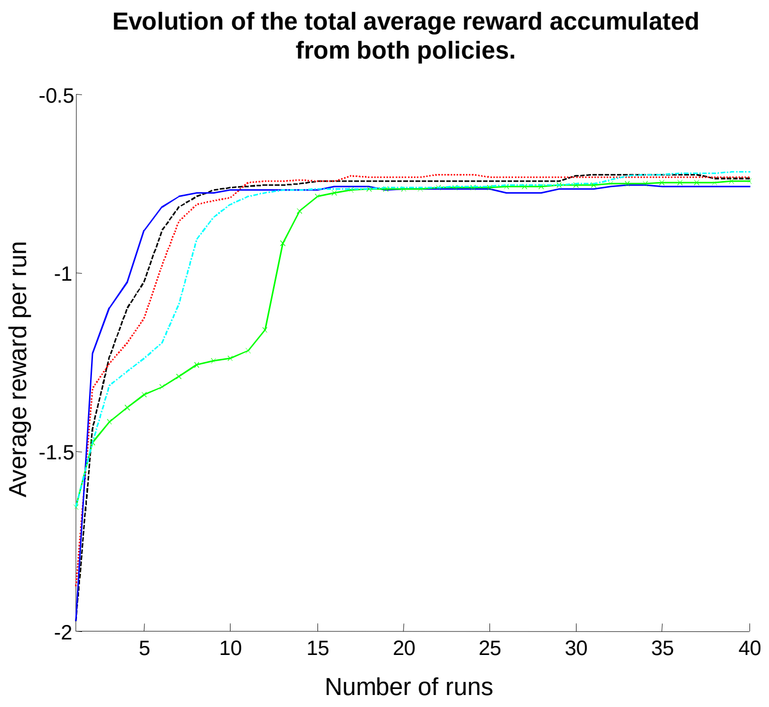

Figure 51: Accumulated reward progression for 5 independent tests. Reward shown is the result of adding the accumulated reward for both Sway and Yaw policies.

atively early episodes $6x10^2$. Once the learning process is considered finished, resultant policies obtained with the NAC algorithm with its correspondent parameters are transferred to the underwater robot Ictineu AUV, ready to be tested in the real world.

6.4.5 *Step 2, real learning results*

At the beginning of step 2, the learned-in-simulation policies are evaluated on the real robot. These policies, one for the Sway DoF and another for the Yaw DoF, are transferred to the Ictineu AUV and, before continuing with the policy improvement on-line, initial performance samples are extracted during various attempts by the robot to try to center the cable within the image plane. Figure 47 shows real data concerning the measured values of the Θ angle. In the same way, Figure 48 shows real samples of data regarding x_g.

The results obtained with the learned-in-simulation policies on the real robot show, despite some major oscillations at the beginning of each sample, a good performance of both simulated policies, allowing the robot to drive *safely* into the underwater scenario. Since this basic knowledge

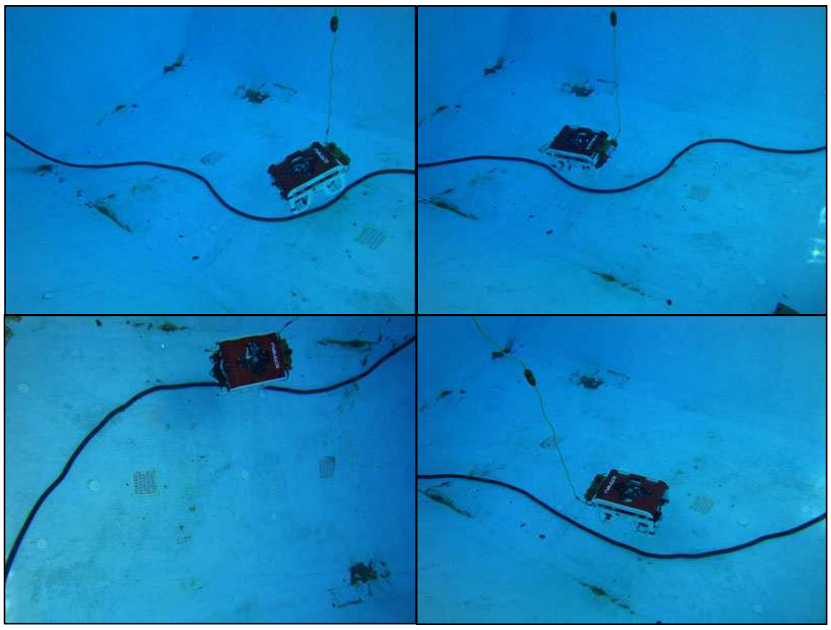

Figure 52: A particular learned policy confronts different cable configurations. Upper-left image: Configuration 1, Upper-right image: Configuration 2, Bottom-left image: Configuration 3, Bottom-right image: Configuration 4.

acquired in simulation has passed the tests, the transferred policies are ready to start the real learning on the robot. The main objective of the on-line real learning step is to improve the learned-in-simulation policies while interacting with the real environment. As policies improve, the oscillations of Θ and x_g observed in Figure 47 and Figure 48 should be reduced, smoothing the robot's movements while tracking the cable.

Figure 49 shows the evolution of the state variable Θ along different learning runs. The first run, depicted with black color, illustrates the performance of the Θ angle after 90 seconds of on-line learning. Each run lasts approximately 90 seconds, which corresponds to the time needed by the robot to make a full run from one corner of the pool to the opposite, navigating around 9 meters of cable. Once the robot reaches the corner of the pool, it automatically disconnects the learning behavior and enables another behavior which makes the robot turn 180° around the Z axis, facing the cable in the opposite direction. Then it enables the learning behavior again and a new run starts. Figure 49 shows how, as the learning process goes on, policies improve significantly. After 20 trials (around 1800 seconds from the beginning) the blue line shows a smoother Θ evo-

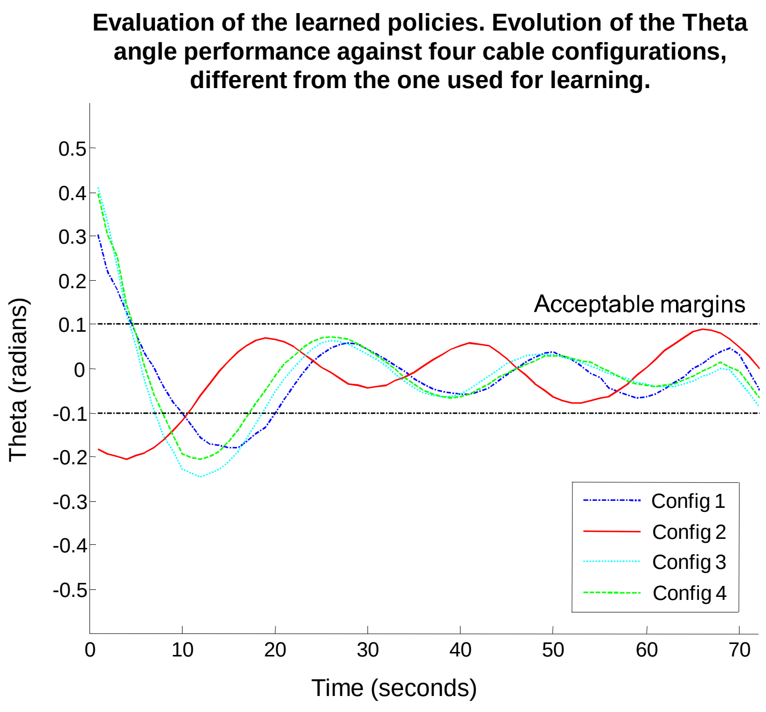

Figure 53: A particular Θ policy performance over the different cable configurations showed in Figure 52.

lution along the run, denoting a significant improvement of the trajectories. Figure 50 depicts the results regarding the state variable x_g along the same learning runs.

Figure 51 shows the accumulated reward evolution during the real learning phase. This figure is the result of adding the total accumulated reward per trials of both policies, Sway and Yaw, over five independent tests of 40 runs each, remembering that each run lasts approximately 90 seconds, and represents a full length run from one side of the cable to the other, navigating along 9 meters. It can be observed that in all 5 tests, the algorithm converges to a better policy as shown by the increasing evolution of the averaged rewards.

6.4.6 *Algorithm adaptation capabilities*

The last section of these experimental results looks into the adaptation capabilities of the NAC algorithm. Up to this point, the various tests performed demonstrate that this algorithm is able to solve various RLPs in short amounts of time, clearly outperforming other algorithms like QL or simple PG techniques. This section aims to test the response of the algorithm to sudden changes in the environmen-

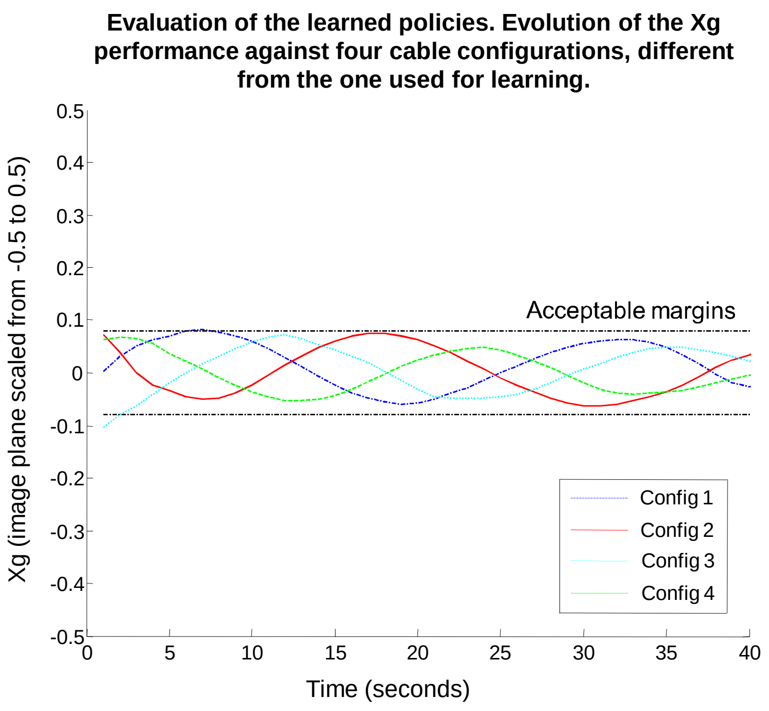

Figure 54: A particular x_g policy performance over the different cable configurations showed in Figure 52.

tal conditions once a particular policy has been learned. These changes will require a fast policy readaptation in order to successfully continue developing the task. The first adaptation experiment consists of modifying the cable's position and trajectory along the pool floor. Therefore, once the on-line real learning period is considered finished and a particular policy for both the Sway and the Yaw has been learned, the cable position is changed 4 different times. Figure 52 shows the 4 different cable configurations adopted. As can be seen in the figure, the top images correspond to a chicane configuration with close cable turns; the upper-left configuration starts with a left turn, whereas the upper-right one starts with a right turn. The bottom images correspond to a cable configuration with a big wide turn; the bottom-left configuration makes a wide right turn, whereas the bottom-right configuration makes a wide left turn.

Figure 53 shows the evolution of the state variable Θ for each of the 4 different configurations described above. Figure 54 shows the evolution of the state variable x_g for the same configurations. Results show that the policy learned for a particular situation is also a good one for tracking

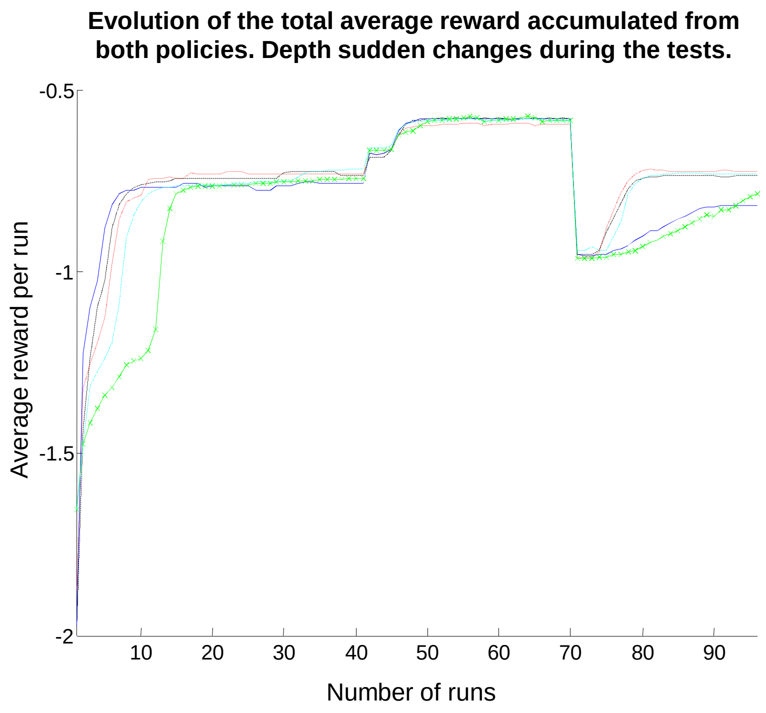

Figure 55: Total averaged reward evolution over 5 independent tests. Initially the setpoint is 0.8 meters. At run 40 the setpoint changes to 1.2 meters before finally recovering its original value of 0.8 at run 70.

a cable positioned in a wide range of different curvature angles that may differ from the original one.

After proving that the learned policies work with different cable configurations, a second set of experiments aims to evaluate if the NAC algorithm is able to adapt itself to unexpected changes in the environmental conditions and continue the learning process. In order to do so, a variation in the altitude with respect to the cable has been introduced in the middle of the learning process. As detailed in previous sections, the underwater robot autonomously navigates at a predefined constant height above the pool floor. For this set of tests, during the learning process, instead of tracking the cable at 0.8 meters with respect to the pool floor, the altitude will suddenly change to 1.2 meters. Since the altitude is not a part of the state input of the algorithm, a variation of this term causes a variation of the RL dynamics and, therefore, current policies are no longer optimal.

Figure 55 shows the accumulated reward evolution of the 5 sample tests initially presented in Figure 51. As can be seen in this figure, once the learning curves are stabilized, a sudden change in the robot's altitude is introduced (at around the 40th run the robot's altitude is changed from 0.8 meters to 1.2 meters). This change can be seen in the graph as a sudden change in the reward value. At this point, rewards become better because the robot navigates at a higher altitude and tracking the cable becomes easier, as it appears smaller in the image plane. From run 40 to run 50, the robot readapts itself to the new environmental conditions and the policies improve. From run 50 to run 70 the learning has stabilized itself and robot navigates at 1.2 meters altitude with its new, improved policies. It can be observed how the 5 independent tests converged to a better policy and achieved similar accumulated rewards. At around run 70, the altitude changes again, recovering its initial value of 0.8 meters. 15 runs later, the averaged rewards stabilize at the same values before the altitude changes were introduced, denoting good on-line adaptation capabilities of the algorithm.

7 CONCLUSION

This chapter concludes the work presented throughout this dissertation. It first summarizes the thesis by reviewing the contents described in each chapter. It then points out the research contributions extracted from the proposals and the experiments. In addition, all aspects which have not been accomplished as well as some interesting future research issues are commented on in the future work section. Then, the research framework in which this thesis was achieved is described. Finally, the publications related to this work are listed.

7.1 SUMMARY

This thesis is concerned with the field of autonomous underwater robots and the problem of action-decision. The methodology chosen to solve such a problem is Reinforcement Learning (RL). Throughout this dissertation, different RL techniques for autonomy improvement of Autonomous Underwater Vehicles (AUVs) have been applied to the Ictineu AUV, one of the research platforms available in the Computer Vision and Robotics Group (VICOROB) at the University of Girona. Moved by the considerable interest that, over the past few years, has arisen around AUV applications, this thesis demonstrates the feasibility of learning algorithms to help AUVs perform autonomous tasks. Particularly, this thesis concentrates on one of the fastest maturing, and probably most immediately significant, commercial application: Cable and Pipeline Tracking. In this way, this thesis presents Policy Gradient (PG) techniques, a particular class of RL methods, as an alternative to classic Value Function (VF) algorithms. The final experiments demonstrate that the best results are obtained when combining both VF and PG techniques into a particular class of algorithms called Actor-Critic (AC) methods.

Chapter 2 has overviewed the field of RL and presented its main principles. The different functions that intervene in the learning process have been described and the most representative solutions for solving the Reinforcement Learning Problem (RLP) have been presented. Among all the RL methodologies, this chapter has given special interest to those techniques especially designed to work in the real domain. Two kinds of algorithms, VF algorithms and PG algorithms, have been analyzed and compared, and the main advantages and drawbacks of each one when dealing with real robotic tasks have been detailed. This theoretical and practical survey concludes with a particular combination of both methods which exploits the advantages of VF and PG, called AC algorithms. These kinds of algorithms demonstrate the best performance for solving real robotic tasks. Practical applications of all theoretical algorithms are discussed at the end of this chapter. Some conclusions and ideas have been extracted in order to design learning methodologies for underwater robots that represent the main contribution of this dissertation.

Chapter 3 has analyzed and proposed the utilization of different PG techniques for AUVs in a real robotic task: underwater cable tracking. The first lines of this chapter have focused on a description of control architectures and how RL methodologies merge to them. A brief discussion of the evolution of behavior-based control architectures for real robots has been detailed, reviewing the history of control architectures for autonomous robots. The generalization problem, which highly affects real robotics, has also been described. The most common approaches to confront this problem and their application to robotic tasks have been overviewed. The GPOMDP algorithm is the first algorithm proposed in this thesis to build a policy for an RLP in a real autonomous underwater task. Two different function approximators have been used to solve the problem: first an Artificial Neural Network (ANN) and secondly a barycentric interpolator. The second algorithm proposed for the second set of results is an AC method, called Natural Actor-Critic (NAC). In order to speed up the learning process, various techniques have been briefly discussed and, among them, a two step learning process which shares simulated and real learning has been chosen as the best option to reduce convergence time. The final proposal, which represents the main contribution of this thesis, is made at the end of the chapter. The proposal consists of the application, in a real underwater robotic task, of an RL technique such as NAC in a two step learning process. For this purpose, the NAC algorithm is first trained in a simulated environment where it can build a fast initial policy. In the second step the policy is transferred to the real robot to continue the learning process on-line in a real environment.

As introduced in Chapter 3, to decrease the overall learning period, a model of the environment is needed for the initial learning phase. For this purpose, Chapter 4 has described how to compute an approximated model of the underwater vehicle Ictineu AUV that will be used to test the learning algorithms proposed in Chapter 3. This chapter has described the dynamics equations of motion of an underwater vehicle as well as the assumptions made to adapt the general equation to our particular vehicle. A Least Squares (LS) identification method has been proposed and applied to identify the vehicle's parameters.

Once the proposal has been presented, Chapter 5 has reported the main characteristics of the different elements used to build the experimental setup. First, a description of the Ictineu AUV and its control architecture has been detailed. The problem and the motivations which lead to the selection of an underwater cable tracking task have been described. The algorithm used to estimate the position and orientation of the cable within the image plane has been presented. The Computer Vision and Robotics Group facility at the Centre d'Investigació en Robòtica Submarina (CIRS) where all the tests have been performed is described at the end of this chapter. Finally, in Chapter 6, the results have been presented, organizing them in a series of sections which intend to show the experimental progression carried out with the different algorithms proposed. First, a comparison between the Q-Learning (QL) method and the GPOMDP algorithm trying to solve the mountain-car benchmark has been presented. In a second set of tests, the Baxter and Bartlett algorithm has been tested using different function approximators in a simulated cable tracking task. A third round of tests has compared the Baxter and Bartlett PG with an AC. The results have proven Peter's NAC algorithm as the most suitable method for the underwater cable tracking task and has been chosen for the final tests. The final experiments presented in this thesis have reduced convergence time by adding an initial computer simulator step to the learning process and, in the second step, the transferred policy has continued the learning process on-line.

The purpose of this application is not to demonstrate that RL performs better than other learning techniques, simple controllers or a human operator. Our purpose is to demonstrate the feasibility of RL techniques to learn autonomous underwater tasks. The cable tracking task can be considered an easy one, but, to the best of the author's knowledge, it is the first underwater application where online learning takes place. RL algorithms demonstrate high adaptation capabilities, and the two-step NAC method has also proven to be so when performing this particular cable tracking task. The algorithm has successfully adapted itself to different cable curvatures, different distances from the cable and different tracking speeds. Of course, there are limits and probably beyond them new methodologies will show a better response but, as stated before, this work represents our first step. As future work, we plan to apply RL to other tasks: more challenging missions, open waters with unexpected and changing currents, introduce random motor failures, different sensor information or enhanced simulation algorithms for faster learning.

7.2 CONTRIBUTIONS

This thesis work has accomplished the proposed goal of studying and using of policy gradient-based reinforcement learning algorithms for the development of RL-based behaviors and their application to autonomous underwater robotic tasks. In the development of this goal, various research contributions were achieved. These contributions are listed below.

ONLINE LEARNING IN UNDERWATER ROBOTICS. Use of RL in robotics is very common nowadays. However, there are not many approaches which perform online learning and, to the best of the author's knowledge, this thesis presents one of the first online robot behavior learning in the underwater domain. It is, therefore, an important contribution to demonstrate the feasibility of PG methods in a real-time task, especially in a complex domain such as underwater robotics, with a two-step method proposal which greatly reduces the overall convergence time.

INTEGRAL LS MODEL IDENTIFICATION. The integral LS identification procedure carried out to identify the dynamic equations of the Ictineu AUV also represents a strong contribution to the underwater vehicles modeling field. The use of the integral of the LS error instead of the error itself gives a much more accurate model which, although being identified as an uncoupled model by independent DoFs, can be later used in the coupled model with minimal error in the final performance.

LEARNING UNDERWATER TASKS. Nowadays, there are a few underwater robots that execute some simple tasks autonomously. In fact, the cable tracking task, which the Ictineu AUV has *learnt* to perform, can be easily programmed using common control theory. This dissertation aims to introduce the application of learning techniques to the development of more challenging underwater tasks. Having a robot continuously learning to perform a task can be very valuable, especially for complex operations difficult to reprogram with control algorithms when adapting to new scenarios. The challenges of learning in such a changing environment like underwater domains are huge.

UNDERWATER DOMAINS AND RL. Most of the work published on RL real applications to robotic tasks are not concerned with underwater scenarios. To the best of the author's knowledge, the approach presented in this thesis is the first work on the utilization of PG algorithms for behavioral learning of autonomous underwater robots. This contribution opens the door to the use of PG methods in new application domains.

7.3 FUTURE WORK

During the development of this thesis, new problems and topics of interest for future research have arisen. The following points are considered the most logical lines to continue this research:

REAL SCENARIOS AND HUMAN DEMONSTRATION. The learning algorithms used in this thesis should be tested in real scenarios and in more, different underwater tasks. Operating in real scenarios like coastal waters will introduce new challenges: unpredictable water currents, irregular seafloor and changing light conditions to name a few. Also, in the real world, using human demonstration of the cable following task could significantly speed up the learning process.

GATHER ALL AVAILABLE DATA FROM SENSORS. All the final tests presented in this thesis use visual information provided by the downward-looking camera to build the state needed by the algorithm to track the cable. The main advantage of such a visual sensor is that, if the image is good enough to detect the cable, the state information will be accurate and almost without noise, insuring a good observation of the state which guarantees a correct learning process. The main drawback is that most commercial applications of underwater vehicles require depths which do not guarantee a clear and uniform illumination of the scene even with artificial light. Thinking about such applications would require learning algorithms capable of getting state information from multiple sensors, acoustic and visual, and deal with noisy state information, especially if it comes from the acoustic sensors.

MORE CHALLENGING TASKS. One of the objectives of this thesis was to demonstrate the feasibility of RL techniques to learn autonomous underwater tasks. Since the complexity of the cable tracking task presented in the final experiments did not represent a problem for algorithms like NAC, the development of more complex tasks may offer these algorithms the opportunity to prove themselves in more complex underwater operations.

INTRODUCE LEARNING AT THE MISSION LEVEL. As described in Section 5.2.3, the library of behaviors which include the RL-based behavior presented in this thesis are enclosed inside the control architecture at the task level. Above this level there is the mission level, responsible for the sequencing of the mission tasks, selecting the set of behaviors that must be enabled for each mission phase. At this time, the mission level is composed of a sequence of fixed behaviors which are triggered one after the other. Introducing RL techniques at this level will make the whole architecture more flexible and able to reorganize the tasks depending on the particular situations that may happen during the development of the mission.

7.4 RESEARCH FRAMEWORK

The results and conclusions presented in this thesis have been possible after the realization of countless tests and experiments, which were the fruit of numerous efforts made during the development of the different research robots and the necessary software and equipment. All the work done during the evolution of this thesis is summarized here with references to the most relevant research publications made by the author. The complete list of publications can be consulted in the next section.

At the beginning of this thesis, there were two research robots in the Computer Vision and Robotics Group (VICOROB) at the University of Girona. The first was the GARBI AUV, a large robot for operation in a real environment. The second robot was the URIS AUV [AMI'04], a robot of reduced dimensions designed to operate under laboratory conditions in a small tank. For simulation and navigation purposes, the model of the GARBI AUV had been previously identified following very rudimentary techniques [127]. For the URIS AUV, a new identification process based on the integral of the dynamics equation was carried out. The methodology was developed in collaboration with professor Antonio Tiano of the University of Pavia. The first initial results, where the performance of the proposed technique was compared with the old one using the GARBI AUV, were presented in [GCUV'03], [MMAR'03] and **[MCMC'03]**. The final identification procedures for fast identification of underwater robots concluded with the publication of **[CEP'04]**. This methodology has been used in Chapter 4 to identify the dynamics model of the Ictineu AUV, used for the final experiments of this dissertation. The performance of the identified models empowered the development of an underwater graphical simulator [ISCCSP'04]. This simulator allows the user to load any of the identified underwater robots in the lab, even working with multiple vehicles, as well as working under different environmental conditions depending on the task: dam inspection, cable tracking and obstacle avoidance. Also, the models obtained have been used to improve navigation algorithms developed for the vehicles [WESIC'03]. Parallel to the work done with modeling of underwater robots, this identification methodology has been used in collaboration with other research groups to the identification of wheeled mobile robots with successful results [IAV'04]. The field of application of dynamic models was very wide. In those days, the research frame of Dr. Marc Carreras focused on the development of RL-based behaviors for autonomous navigation of underwater vehicles. Several conversations suggested the possibility of applying simulators to solve one of the key problems of learning in real robotics: slow convergence speeds.

The first research work done in RL focused on VF methods [IROS'03] and the development of behavioral control architectures for underwater robots **[WAF'04]**. Due to the low performance demonstrated by *pure* VF algorithms such as QL when dealing with real environments, the interest of this thesis moved to other RL techniques. Another class of methods, called PG methods, used a different approach to learn a policy. Such algorithms were easier to program than VF algorithms and, although in general being less accurate, they showed great capabilities when dealing with real environments. The first results obtained with PG techniques studied the feasibility of these methods for the underwater robots' framework **[WAF'05]**. The first comparative results between a VF method and a PG algorithm were presented in **[CCIA'05]**, where the QL algorithm is compared with the GPOMDP method in the mountain-car task benchmark. Results obtained with a PG were good and the next step was to apply them to the underwater domain performing various simulated tasks published in **[OCEANS'05]**, **[ICINCO'05]**, **[IROS'06]** and **[ICAR'07]**. With the simulated results clear, experiments with PG techniques started to work with the real robot, Ictineu AUV. Therefore, the tests focused on learning an initial policy in simulation using a PG algorithm and transfer it to the real robot to test its performance, still without improving learning on-line. Different function approximators (ANN and barycentric interpolators) were studied and utilized during these tests, looking for the one which offered the best results for the particular task of underwater cable tracking **[NGCUV'08]**, **[ICINCO'08]**, **[ECAI'08]** and **[IROS'08]**.

The results obtained in these publications, although being quite good, showed low adaptation capabilities of the GPOMDP algorithm which would represent a problem when attempting real online learning with the Ictineu AUV. A new class of RL methods, called AC algorithms, which combine good properties of VF and PG techniques, were studied and applied to the final experiments of this thesis. Thus, a variation of an AC proposed by Jan Peters called NAC was applied to the final round of experiments. In these tests, the robot Ictineu AUV learned to perform an underwater cable tracking task in a two step learning process. First an initial policy is approximated by means of a simulator and, in the second step, it is transferred to the real robot where the learning continues on-line, improving the initial policy and adapting itself to sudden changes in the environment that may appear in the future **[ICRA'10]**.

7.5 RELATED PUBLICATIONS

Model identification of underwater robots

[GCUV'03] M. Carreras, A. Tiano, A. El-Fakdi, A. Zirilli and P. Ridao. On the identification of non linear models of unmanned underwater vehicles. In *1st IFAC Workshop on Guidance and Control of Underwater Vehicles GCUV*, Wales, UK, April 2003.

[MMAR'03] A. Tiano, M. Carreras, A. El-Fakdi, P. Ridao and M. Cuneo. Nonlinear Identification of Underwater Vehicles. In *9th International Conference on Methods and Models in Automation and Robotics*, Miedzyzdroje, Poland, August 2003.

[MCMC'03] A. El-Fakdi, A. Tiano, P. Ridao and J. Batlle. Identification of non linear models of unmanned underwater vehicles: comparison between two identification methods. In *6th Conference on Maneuvering and Control of Marine Crafts MCMC*, Girona, Spain, September 2003.

[IAV'04] B. Innocenti, P. Ridao, N. Gascons, A. El-Fakdi, B. Lopez and J. Salvi. Dynamical Model Parameters Identification of a Wheeled Mobile Robot. In *5th IFAC Symposium on Intelligent Autonomous Vehicles IAV*, Lisboa, Portugal, July 2004.

[ISCCSP'04] P. Ridao, M. Carreras, D. Ribas and A. El-Fakdi. Graphical Simulators for AUV Development. In *1rst International Symposium on Control, Communications and Signal*

Processing, Hammamet, Tunisia, March 2004.

[CEP'04] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras and A. Zirilli. On the identification of non linear models of unmanned underwater vehicles. *Control Engineering Practice*, 12:1483-1499, 2004.

PG methods applied to underwater robots

[IROS'03] M. Carreras, P. Ridao and A. El-Fakdi. Semi-Online Neural-Q- learning for Real-time Robot Learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, Las Vegas, USA, October 2003.

[WAF'04] A. El-Fakdi, M. Carreras and J. Batlle. Control of an Autonomous Robot using Multiple RL-based Behaviors. In *V Workshop de Agentes Físicos WAF*, Girona, Spain, March 2004.

[WAF'05] A. El-Fakdi, M. Carreras, P. Ridao and E. Hernández. Studying the feasibility of policy reinforcement learning methods for autonomous agents. In *VI Workshop de Agentes Físicos WAF*, Granada, Spain, September 2005.

[CCIA'05] A. El-Fakdi, M. Carreras and N. Palomeras. Direct policy search reinforcement learning for robot control. In *8é Congrés Català de Intel·ligència Artificial CCIA*, Algero, Italy, October 2005.

[OCEANS'05] A. El-Fakdi, M. Carreras, N. Palomeras and P. Ridao. Autonomous underwater vehicle control using reinforcement learning policy search methods. In *IEEE Oceans Europe*, Brest, France, June 2005.

[ICINCO'05] A. El-Fakdi, M. Carreras and P. Ridao. Direct-gradient based reinforcement learning for robot behavior learning. In *2nd International Conference on Informatics in Control, Automation and Robotics ICINCO*, Barcelona, Spain, September 2005.

[IROS'06] A. El-Fakdi, M. Carreras and P. Ridao. Towards Direct Policy Search Reinforcement Learning for Robot Control. In *IEEE/RSJ International Conference on Intelligent Robots* *and Systems IROS*, Beijing, China, October 2006.

[ICAR'07] A. El-Fakdi, M. Carreras and P. Ridao. Direct gradient-based reinforcement learning for robot behavior learning. *Informatics in Control Automation and Robotics II*, pp:175-182, 2007. J. Filipe, J. Ferrier, J.A. Cetto and M. Carvalho, first edition, 2007. ISBN 978-1-4020-5625-3.

[NGCUV'08] A. El-Fakdi, M. Carreras and J. Batlle. Direct Policy Search Reinforcement Learning for Autonomous Underwater Cable Tracking. In *IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles NGCUV*, Killaloe, Ireland, April 2008.

[ICINCO'08] A. El-Fakdi, M. Carreras, J. Antich and A. Ortiz. Learning by Example: Reinforcement Learning Techniques for Real Autonomous Underwater Cable Tracking. In *5th International Conference on Informatics in Control, Automation and Robotics ICINCO*, Funchal, Madeira, Portugal, May 2008.

[ECAI'08] A. El-Fakdi, M. Carreras and E. Hernández. Gradient Based Reinforcement Learning for Autonomous Underwater Cable Tracking. In *18th European Conference on Artificial Intelligence ECAI*, Patras, Greece, July 2008.

[IROS'08] A. El-Fakdi and M. Carreras. Policy Gradient Based Reinforcement Learning for Real Autonomous Underwater Cable Tracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, Nice, France, September 2008.

[ICRA'10] A. El-Fakdi, M. Carreras and E. Galceran. Two steps natural actor critic learning for underwater cable tracking. In *IEEE International Conference on Robotics and Automation ICRA*, Anchorage, Alaska, May 2010.

Related work in underwater robotics

[WESIC'03] D. Ribas, P. Ridao, X. Cufí and A. El-Fakdi. Towards a DVL-based navigation system for an underwater robot. In *4th Workshop on European Scientific and Industrial Collaboration WESIC*, Miskolc, Hungary, May 2003.

[AMI'04] J. Batlle, P. Ridao, R. Garcia, M. Carreras, X. Cufí, A. El-Fakdi, D. Ribas, T. Nicosevici, E. Batlle, G. Oliver, A. Ortiz and J. Antich. *URIS: Underwater Robotic Intelligent System. Automation for the Maritime Industries*, chapter 11, pp: 177-203. Instituto de Automática Industrial, Consejo Superior de Investigaciones Científicas, first edition, 2004. ISBN 84-609-3315-6.

BIBLIOGRAPHY

[1] D.A. Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australian National University, April 2003. (Cited on pages 29 and 119.)

[2] M.A. Abokowitz. System identification techniques for ship manoeuvring trials. In *Symposium on control theory and navy applications*, pages 337–393, Monterey, USA, 1980. (Cited on page 79.)

[3] P.E. Agre and D. Chapman. Pengi: an implementation of a theory of activity. In *6th Annual Meeting of the American Association for Artificial Intelligence*, pages 268–272, Seattle, Washington, 1987. (Cited on page 57.)

[4] J.S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1971. (Cited on page 63.)

[5] J.S. Albus. *Brain, behavior and robotics*. Byte Books, Peterborough, NH, 1981. (Cited on page 63.)

[6] J.S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3): 473–509, May-June 1991. (Cited on page 56.)

[7] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998. (Cited on pages 35 and 38.)

[8] J. Amat, J. Batlle, A. Casals, and J. Forest. GARBI: a low cost ROV, constrains and solutions. In *6ème Seminaire IARP en robotique sous-marine*, pages 1–22, Toulon-La Seyne, France, 1996. (Cited on page 102.)

[9] C. Anderson. Approximating a policy can be easier than approximating a value function. Computer science technical report, University of Colorado State, 2000. (Cited on pages 29 and 38.)

[10] J. Antich and A. Ortiz. Underwater cable tracking by visual feedback. In *First Iberian Conference*

on Pattern recognition and Image Analysis, IbPRIA, Port d'Andratx, Spain, 2003. (Cited on page 113.)

[11] R.C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research, August 1989*, 8(4):92–112, 1989. (Cited on page 58.)

[12] R.C. Arkin. *Behavior-based Robotics*. MIT Press, 1998. (Cited on page 55.)

[13] R.C. Arkin and T. Balch. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189, 1997. (Cited on page 59.)

[14] K. Asakawa, J. Kojima, Y. Kato, S. Matsumoto, N. Kato, T. Asai, and T. Iso. Design concept and experimental results of the autonomous underwater vehicle AQUA EXPLORER 2 for the inspection of underwater cables. *Advanced Robotics*, 16(1):27–42, 2002. (Cited on page 110.)

[15] C.G. Atkenson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11: 11–73, 1997. (Cited on pages 64, 66, and 76.)

[16] J.A. Bagnell and J.G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *IEEE International Conference on Robotics and Automation, ICRA*, Korea, 2001. (Cited on page 78.)

[17] K. Baird. Residual algorithms: Reinforcement learning with function approximation. In *12th International Conference on Machine Learning, ICML*, San Francisco, USA, 1995. (Cited on page 62.)

[18] L. Baird and A. Moore. Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems, NIPS*, Vol 11, MIT Press, 1999. (Cited on page 42.)

[19] A. Balasuriya and T. Ura. Vision based underwater cable detection and following using AUVs. In *MTS/IEEE Oceans*, Biloxi, Mississippi, October 2002. (Cited on page 111.)

[20] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike elements that can solve difficult learning control

problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13:835–846, 1983. (Cited on pages 38 and 41.)

[21] J. Batlle, T. Nicosevici, R. Garcia, and M. Carreras. ROV-aided dam inspection, practical results. In *6th IFAC Conference on Manoeuvring and Control of Marine Crafts*, Girona, Spain, 2003. (Cited on page 3.)

[22] J. Batlle, P. Ridao, R. Garcia, M. Carreras, X. Cufi, A. El-Fakdi, D. Ribas, T. Nicosevici, and E. Batlle. *URIS: Underwater Robotic Intelligent System*, chapter 11, pages 177–203. Instituto de Automatica Industrial, Consejo Superior de Investigaciones Cientificas, 1rst edition, 2004. (Cited on page 102.)

[23] J. Baxter and P.L. Bartlett. Direct gradient-based reinforcement learning. In *International Symposium on Circuits and Systems*, Geneva, Switzerland, May 2000. (Cited on pages 32 and 38.)

[24] R.E. Bellman. *Dynamic Programming*. Princenton University Press, 1957. (Cited on page 18.)

[25] H. Benbrahim and J. Franklin. Biped dynamic walking using reinforcement. *Robotics and Autonomous Systems*, 22:283–302, 1997. (Cited on page 27.)

[26] H. Benbrahim, J. Doleac, J. Franklin, and O. Selfridge. Real-time learning: A ball on a beam. In *International Joint Conference on Neural Networks (IJCNN)*, Baltimore, MD, USA, 1992. (Cited on page 27.)

[27] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996. (Cited on pages 38 and 66.)

[28] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. In *Handbook of Robotics*, Handbook of Robotics, chapter chapter 59. MIT Press, 2008. (Cited on page 76.)

[29] J.A. Boyan. Least-squares temporal difference learning. In *16th International conference on Machine Learning, ICML*, pages 49–56, 1999. (Cited on page 45.)

[30] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, April 1986. (Cited on page 57.)

[31] R.A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *12th International Joint Conference on Artificial Intelligence, IJCAI*, pages 569–595, Sydney, Australia, August 1991. Morgan Kaufmann publishers Inc. ISBN 1-55860-160-0. (Cited on page 56.)

[32] S. Buck, M. Beetz, and T. Schmitt. Approximating the value function for continuous space reinforcement learning in robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Lausanne, Switzerland, 2002. (Cited on page 66.)

[33] M. Caccia, G. Indiveri, and G. Veruggio. Modelling and identification of open-frame variable configuration underwater vehicles. *IEEE Journal of Ocean Engineering*, 25(2):227–240, 2000. (Cited on pages 79 and 83.)

[34] M. Carreras. *A Proposal of a Behavior-Based Control Architecture with Reinforcement Learning for an Autonomous Underwater Robot*. PhD thesis, University of Girona, May 2003. (Cited on pages xiv, 2, 3, 56, 57, and 59.)

[35] M. Carreras, P. Ridao, and A. El-Fakdi. Semi-Online Neural-Q-learning for real-time robot learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Las Vegas, USA, October 27-31 2003. (Cited on page 37.)

[36] M. Carreras, N. Palomeras, P. Ridao, and D. Ribas. Design of a mission control system for an AUV. *International Journal of Control*, 80:993–1007, 2007. (Cited on pages 3 and 108.)

[37] D. Chapman and L.P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and preformance comparisons. In *12th International Joint Conference on Artificial Intelligence, IJCAI*, pages 726–731, San Mateo, CA, 1991. (Cited on page 63.)

[38] R. Chatila and J. Laumond. Position referencing and consistent world modelling for mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 138–170, 1985. (Cited on page 56.)

[39] J.H. Connell. Sss: A hybrid architecture applied to robot navigation. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 2719–2724, 1992. (Cited on page 58.)

[40] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems, NIPS*, Cambridge, MA, 1996. MIT Press. (Cited on page 62.)

[41] P. Dayan. Reinforcement comparison. In editors Touretzky, D. S. et al., editor, *1990 Connectionist Models Summer School*, San Mateo, CA, 1990. Morgan Kaufmann. (Cited on page 30.)

[42] P. Dayan. The convergence of TD(λ) for general λ. *Journal of Machine Learning*, 8:341–362, 1992. (Cited on page 62.)

[43] T.G. Dietterich and X. Wang. Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems, NIPS*, pages 1491–1498. MIT Press, 2002. (Cited on page 67.)

[44] J. Evans, Y. Petillot, P. Redmond, M. Wilson, and D. Lane. AUTOTRACKER: AUV embedded control architecture for autonomous pipeline and cable tracking. In *MTS/IEEE Oceans*, pages 2651–2658, San Diego, California, September 2003. (Cited on page 110.)

[45] R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971. (Cited on page 56.)

[46] R.J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, New Haven, Connecticut, 1989. (Cited on page 59.)

[47] T.I. Fossen. *Guidance and Control of Ocean Vehicles*. John Wiley and Sons, 1995. (Cited on pages 79, 81, and 107.)

[48] D. Gachet, M. Salichs, L. Moreno, and J. Pimental. Learning emergent tasks for an autonomous mobile

robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 290–97, Munich, Germany, September 1994. (Cited on page 61.)

[49] C. Gaskett. *Q-learning for Robot Control*. PhD thesis, Australian National University, 2002. (Cited on page 66.)

[50] E. Gat. *Reliable Goal-directed Reactive Control for Real-World Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic and State University, Blacksburg, Virginia, 1991. (Cited on page 59.)

[51] G.J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie-Mellon University, 1999. (Cited on page 62.)

[52] E. Greensmith, P.L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5: 1471–1530, 2004. (Cited on page 31.)

[53] H.M. Gross, V. Stephan, and M. Krabbes. A neural field approach to topological reinforcement learning in continuous action spaces. In *IEEE World Congress on Computational Intelligence, WCCI and International Joint Conference on Neural Networks, IJCNN*, Anchorage, Alaska, 1998. (Cited on page 66.)

[54] V. Gullapalli. Associative reinforcement learning of real-value functions. In *IEEE International Conference on Systems, Man and Cybernetics*, Charlottesville, VA, USA, 1991. (Cited on page 31.)

[55] V. Gullapalli, J J. Franklin, and H. Benbrahim. Aquiring robot skills via reinforcement. *IEEE Control Systems Journal, Special Issue on Robotics: Capturing Natural Motion*, 4(1):13–24, 1994. (Cited on page 27.)

[56] S.T. Hagen and B. Krose. Neural q_learning. Technical Report IAS-UVA-00-09, Computer Science Institute, University of Amsterdam, The Netherlands, 2000. (Cited on page 66.)

[57] B. Hammer, S. Singh, and S. Scherer. Learning obstacle avoidance parameters from operator behavior. *Journal of Field Robotics, Special Issue on Machine*

Learning Based Robotics in Unstructured Environments, 23 (11/12), December 2006. (Cited on page 77.)

[58] S. Haykin. *Neural Networks, a comprehensive foundation*. Prentice Hall, 2nd ed. edition, 1999. (Cited on pages 65, 119, and 125.)

[59] N. Hernandez and S. Mahadevan. Hierarchical memory-based reinforcement learning. In *Advances in Neural Information Processing Systems, NIPS*, Denver, USA, 2000. (Cited on page 37.)

[60] I. Horswill. Polly: A vision-based artificial agent. In *IEEE National Conference on Artificial Intelligence, AAAI*, 1993. (Cited on page 58.)

[61] H.M. Huang. An architecture and a methodology for intelligent control. *IEEE Expert: Intelligent Systems and their applications*, 11(2):46–55, 1996. (Cited on page 56.)

[62] L. Iovenitti, M. Venturi, G. Albano, and E.H. Touisi. Submarine pipeline inspection: the 12 years experience of TRANSMED and future developments. In *International Conference on Off-shore Mechanics and Arctic Engineering*, pages 149–161, 1994. (Cited on page 108.)

[63] Y. Ito, N. Kato, J. Kojima, S. Takagi, K. Asakawa, and Y. Shirasaki. Cable tracking for autonomous underwater vehicle. In *IEEE Symposium on AUV technology*, pages 218–224, 1994. (Cited on page 110.)

[64] M.D. Iwanowski. Surveillance unmanned underwater vehicle. In *IEEE Oceans*, pages 1116–1119, 1994. (Cited on page 110.)

[65] T. Jung and T. Uthmann. Experiments in value function approximation with sparse support vector regression. In *15th European Conference on Machine Learning, ECML*, 2004. (Cited on page 67.)

[66] S. Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems, NIPS*, 14, 2001. (Cited on pages 35 and 36.)

[67] Z. Kalmar, C. Szepesvari, and A. Lorincz. Module-Based Reinforcement Learning: Experiments with a Real Robot. In *Proceedings of the 6th European Workshop on Learning Robots*, 1997. (Cited on page 61.)

[68] H. Kimura and S. Kobayashi. An analisis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions. In *15th International Conference on Machine Learning, ICML*, pages 278–286, 1998. (Cited on page 41.)

[69] H. Kimura, K. Miyazaki, and S. Kobayashi. Reinforcement learning in pomdps with function approximation. In D. H. Fisher, editor, *14th International Conference on Machine Learning, ICML*, pages 152–160, 1997. (Cited on page 31.)

[70] Donald E. Knuth. Computer Programming as an Art. *Communications of the ACM*, 17(12):667–673, December 1974. (Cited on page ix.)

[71] J. Kober and J. Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation ICRA*, pages 2112–2118, Kobe, Japan, May 2009. (Cited on page 51.)

[72] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, ICRA*, 2004. (Cited on page 48.)

[73] V.R. Konda and J.N. Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems, NIPS*, 12, 2000. (Cited on page 33.)

[74] V.R. Konda and J.N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42, number 4:1143–1166, 2003. (Cited on pages 34, 38, and 39.)

[75] T. Kondo and K. Ito. A Reinforcement Learning with Adaptive State Space Recruitment Strategy for Real Autonomous Mobile Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Lausanne, Switzerland, 2002. (Cited on page 65.)

[76] P. Kormushev, S. Calinon, and D.G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3232–3237, Taipei, Taiwan, October 2010. (Cited on page 51.)

[77] P. Kormushev, K. Nomoto, F. Dong, and K. Hirota. Time hopping technique for faster reinforcement learning in simulations. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Submitted 2009. (Cited on page 78.)

[78] J.E. Laird and P.S. Rosenbloom. Integrating, execution, planning, and learning in Soar for external environments. In T. S. W. Dietterich, editor, *8th Annual Meeting of the American Association for Artificial Intelligence, AAAI*, pages 1022–1029, Hynes Convention Centre, July–August 1990. MIT Press. (Cited on page 56.)

[79] G. Lawrence, N. Cowan, and S. Russell. Efficient gradient estimation for motor control learning. In *International Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico, 2003. (Cited on page 31.)

[80] D. Lefebvre and G. Saridis. A computer architecture for intelligent machines. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 245–250, Nice, France, 1992. (Cited on page 56.)

[81] L.J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Journal of Machine Learning*, 8(3/4):293–321, 1992. (Cited on pages 62, 76, and 78.)

[82] L. Ljung. *System identification: Theory for the user*. Prentice Hall, 1987. (Cited on pages 79, 91, 93, and 94.)

[83] D. Lyons. Planning, reactive. In *Encyclopedia of Artificial Intelligence*. John Wiley and Sons, New York, 1992. (Cited on page 59.)

[84] P. Maes and R. Brooks. Learning to coordinate behaviors. In *Proceedings of Eighth AAAI*, pages 796–802. Morgan Kaufmann, 1990. (Cited on page 61.)

[85] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992. (Cited on page 61.)

[86] S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Jour-*

nal of Machine Learning, 8:2169–2231, 2007. (Cited on page 65.)

[87] F. Maire. Bicephal reinforcement learning. In *7th International Conference on Neural Information Processing*, Taejon, Korea, 2000. (Cited on page 66.)

[88] P. Marbach and J.N. Tsitsiklis. Gradient-based optimization of Markov reward processes: Practical variants. Technical report, Center for Communications Systems Research, University of Cambridge, March 2000. (Cited on page 38.)

[89] M. Martin. On-line support vector machines for function approximation. Technical report, Software Department, Universitat Politecnica de Catalunya, 2002. (Cited on page 67.)

[90] E. Martinson, A. Stoytchev, and R. Arkin. Robot Behavioral Selection Using Q_learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Lausanne, Switzerland, 2002. (Cited on page 61.)

[91] T. Matsubara, J. Morimoto, J. Nakanishi, M. Sato, and K. Doya. Learning sensory feedback to CPG with policy gradient for biped locomotion. In *IEEE International Conference on Robotics and Automation ICRA*, Barcelona, Spain, April 2005. (Cited on page 50.)

[92] K. Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52:367–376, 1985. (Cited on page 50.)

[93] R.A. McCallum. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems, NIPS*, 1995. (Cited on page 64.)

[94] T. McGeer. Passive dynamic walking. *International Journal of Robotics Research*, 9:62–82, 1990. (Cited on page 49.)

[95] I. Menache, S. Mannor, and N. Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134:215–238, 2004. (Cited on page 65.)

[96] N. Meuleau, L. Peshkin, and K. Kim. Exploration in gradient based reinforcement learning. Technical report, Massachusetts Institute of Technology, AI Memo 2001-003, April 2001. (Cited on page 38.)

[97] J.R. Millan, D. Posenato, and E. Dedieu. Continuous-action q_learning. *Journal of Machine Learning*, 49:247–265, 2002. (Cited on page 64.)

[98] G. Monahan. A survey of partially observable markov decision processes. *Management Science*, 28:1–16, 1982. (Cited on page 17.)

[99] A.W. Moore. Variable resolution dynamic programming: Efficiently learning action maps on multivariate real-value state-spaces. In *8th International Conference on Machine Learning, ICML*, 1991. (Cited on page 116.)

[100] M.C. Mozer. Rambot: A connectionist expert system that learns by example. Institute for Cognitive Science Report 8610, University of California, San Diego, California, 1986. (Cited on page 77.)

[101] R. Munos and A. Moore. Barycentric interpolators for continuous space and time reinforcement learning. *Advances in Neural Information Processing Systems, NIPS*, 11:1024–1030, 1998. (Cited on page 65.)

[102] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Journal of Machine Learning*, 49:291–323, 2002. (Cited on pages 63 and 65.)

[103] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989. (Cited on page 107.)

[104] K.P. Murphy. A survey of pomdp solution techniques, 2000. (Cited on page 37.)

[105] K. Murthy. *On growing better decision trees from data*. PhD thesis, John Hopkins University, Baltimore, Maryland, 1996. (Cited on page 62.)

[106] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980. (Cited on page 56.)

[107] N.J. Nilsson. Shakey the robot. Technical Report 323, SRI International, Menlo Park, California, 1984. (Cited on page 56.)

[108] M. Nomoto and M. Hattori. A deep rov "dolphin 3k": Desing and performance analysis. *IEEE Journal of Oceanic Engineering*, 11(3):373–391, 1986. (Cited on page 79.)

[109] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Journal of Machine Learning*, 49:161–178, 2002. (Cited on page 64.)

[110] A. Ortiz, M. Simo, and G. Oliver. A vision system for an underwater cable tracker. *International Journal of Machine Vision and Applications*, 13 (3):129–140, 2002. (Cited on page 113.)

[111] A. Ortiz, J. Antich, and B. Oliver. A particle filter-based approach for tracking undersea narrow telecommunication cables. *International Journal of Machine Vision and Applications*, 2009. (Cited on page 111.)

[112] N. Palomeras, P. Ridao, M. Carreras, and C. Silvestre. Using petri nets to specify and execute missions for autonomous underwater vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Sant Louis, Missouri, October 2009. (Cited on page 107.)

[113] J. Peng. Efficient Memory-based Dynamic Programming. *12th International Conference on Machine Learning, ICML*, 1995. (Cited on page 64.)

[114] J. Peters. *Machine Learning of Motor Skills for Robotics*. PhD thesis, Department of Computer Science, University of Southern California., 2007. (Cited on pages 27, 30, 35, 36, and 38.)

[115] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, Beijing, China, October 9-15 2006. (Cited on pages 4, 27, 38, and 51.)

[116] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *IEEE-RAS International Conference on Humanoid Robots*, Karlsruhe, Germany, Sept. 29-30 2003. (Cited on pages 131, 133, 134, and 135.)

[117] J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. In *ECML*, pages 280–291, 2005. (Cited on pages 27, 30, 44, and 77.)

[118] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990. (Cited on page 65.)

[119] D.A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report cmucs-89-107, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1989. (Cited on page 77.)

[120] M.J.D. Powell. Radial basis functions for multivariate interpolation: A review. In *Algorithms for Approximation*, pages 143–167. Clarendon Press, Oxford, 1987. (Cited on page 65.)

[121] D. Precup, R.S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In *18th International Conference on Machine Learning, ICML*, 2001. (Cited on page 62.)

[122] B. Ratitch and D. Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *European Conference on Machine Learning, ECML*, 2004. (Cited on page 65.)

[123] S.I. Reynolds. *Reinforcement Learning with Exploration*. PhD thesis, University of Birmingham, United Kingdom, December 2002. (Cited on page 63.)

[124] D. Ribas, N. Palomeras, P. Ridao, M. Carreras, and E. Hernandez. Ictineu AUV wins the first SAUC-E competition. In *IEEE International Conference on Robotics and Automation, ICRA*, 2007. (Cited on page 102.)

[125] S. Richter, D. Aberdeen, and J. Yu. Natural actor-critic for road traffic optimisation. In *Neural Information Processing Systems, NIPS*, pages 1169–1176, 2006. (Cited on pages 38 and 52.)

[126] P. Ridao. *A hybrid control architecture for an AUV*. PhD thesis, University of Girona, 2001. (Cited on page 104.)

[127] P. Ridao, M. Carreras, and J. Batlle. Model identification of a low-speed UUV. In *Control Applications in*

Marine Systems, CAMS, Scotland (UK), 2001. (Cited on page 157.)

[128] P. Ridao, E. Batlle, D. Ribas, and M. Carreras. NEPTUNE: A HIL simulator for multiple UUVs. In *MTS/IEEE Oceans*, pages 524–531, Kobe, Japan, 2004. (Cited on page 3.)

[129] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras, and A. Zirilli. On the identification of non-linear models of unmanned underwater vehicles. *Control Engineering Practice*, 12:1483–1499, 2004. (Cited on page 3.)

[130] S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties. *TARK: Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, 1986. (Cited on page 57.)

[131] M.T. Rosenstein and A.G. Barto. Robot weightlifting by direct policy search. In *International Joint Conference on Artificial Intelligence, IJCAI*, 2001. (Cited on page 47.)

[132] M.R.K. Ryan and M.D Pendrith. RL-TOPs: An Architecture for Modularity and Re-Use in Reinforcement Learning. In *15th International Conference on Machine Learning, ICML*, Madison, Wisconsin, 1998. (Cited on page 61.)

[133] F. Saito and T. Fukuda. Learning architecture for real robot systems- extension of connectionist Q-learning for continuous robot control domain. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 27–32, 1994. (Cited on page 63.)

[134] J.C. Santamaria, R.S. Sutton, and A. Ram. Experiments with reinofrcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–218, 1998. (Cited on pages 63 and 64.)

[135] J.M. Santos. *Contribution to the study and design of reinforcement functions*. PhD thesis, Universidad de Buenos Aires, Universite d'Aix-Marseille III, 1999. (Cited on page 65.)

[136] A. Savitzky and M.J.E. Golay. *Analytical Chemistry*, volume 36, pages 1627–1639. 1964. (Cited on page 96.)

[137] S. Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems, NIPS*, pages 1040–1046. MIT Press, 1997. (Cited on page 131.)

[138] S. Schaal, P. Mohajerian, and A.J. Ijspeert. Dynamics systems vs. optimal control, a unifying view. *Progress in Brain Research*, 165:425–445, 2007. (Cited on page 51.)

[139] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, 1st edition, December 2001. ISBN 0262194759. (Cited on page 66.)

[140] J. Shackleton and M. Gini. Measuring the Effectiveness of Reinforcement Learning for Behavior-based Robotics. *Adaptive Behavior*, 5 (3/4):365–390, 1997. (Cited on page 61.)

[141] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. *Journal of Machine Learning*, 22:123–158, 1996. (Cited on page 116.)

[142] S.P. Singh, T. Jaakkola, and M.I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *11th International Conference on Machine Learning, ICML*, pages 284–292, New Jersey, USA, 1994. Morgan Kaufmann. (Cited on pages 16, 37, and 38.)

[143] S.P. Singh, T. Jaakkola, and M.I. Jordan. Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems, NIPS*, 7, 1995. (Cited on page 62.)

[144] W.D. Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science at Brown University, Rhode Island, May 2002. (Cited on pages 64 and 77.)

[145] W.D. Smart. Explicit manifold representations for value-functions in reinforcement learning. In *8th Symposium on Artificil Intelligence and Mathematics*, 2004. (Cited on page 65.)

[146] W.D. Smart and L.P. Kaelbling. Practical reinforcement learning in continuous spaces. In *International Conference on Machine Learning, ICML*, 2000. (Cited on page 37.)

[147] W.D. Smart and L.P. Kaelbling. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA*, 2002. (Cited on pages 47 and 78.)

[148] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, 1984. (Cited on page 30.)

[149] R.S. Sutton. Learning to predict by the method of temporal differences. *Journal of Machine Learning*, 3: 9–44, 1988. (Cited on page 62.)

[150] R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *7th International Workshop on Machine Learning*, pages 216–224, 1990. (Cited on page 78.)

[151] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparce coarse coding. *Advances in Neural Information Processing Sustems, NIPS*, 9:1038–1044, 1996. (Cited on pages 62 and 63.)

[152] R.S. Sutton and A. Barto. *Reinforcement Learning, an introduction*. MIT Press, 1998. (Cited on pages 11, 23, 25, 34, 37, 40, 41, 64, 65, and 131.)

[153] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems, NIPS*, 12:1057–1063, 2000. (Cited on pages 29, 33, and 38.)

[154] Y. Takahashi and M. Asada. Vision-guided behavior acquisition of a mobile robot by multi-layered reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2000. (Cited on page 61.)

[155] R. Tedrake, T.W. Zhang, and H.S. Seung. Stochastic policy gradient reinforcement learning on a simple 3D biped. In *IEEE/RSJ International Conference*

on Intelligent Robots and Systems, IROS, Sendai, Japan, September 28 - October 2 2004. (Cited on page 49.)

[156] G.J. Tesauro. Practical issues in temporal difference learning. *Journal of Machine Learning*, 8(3/4):257–277, 1992. (Cited on page 66.)

[157] A. Tiano, G. Magenes, R. Sutton, and P. Crafen. Identification methods applied to an unmanned underwater vehicle. In *4th IFAC Conference on manoeuvring anc control of marine craft, MCMC*, pages 36–41, Brijuni, Croatia, 1997. (Cited on page 79.)

[158] C. Touzet. Neural reinforcement learning for behavior synthesis. *Robotics and Autonomous Systems*, 22:251–281, 1997. (Cited on page 61.)

[159] J.N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Journal of Machine Learning*, 22:59–94, 1996. (Cited on page 62.)

[160] J.N. Tsitsiklis and B. Van Roy. Average cost temporal-difference learning. *IEEE Transactions on Automatic Control*, 42:674–690, 1997. (Cited on page 62.)

[161] T. Ueno, Y. Nakamura, T. Shibata, K. Hosoda, and S. Ishii. Stable learning of quasi-passive dynamic walking by an unstable biped robot based on off-policy natural actor-critic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2006. (Cited on page 38.)

[162] V.N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc, New York, NY, USA, 1995. (Cited on page 66.)

[163] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989. (Cited on pages 25 and 63.)

[164] C.J.C.H. Watkins and P. Dayan. Q-learning. *Journal of Machine Learning*, 8:279–292, 1992. (Cited on page 38.)

[165] L. Weaver and N. Tao. The optimal reward baseline for gradient-based reinforcement learning. In *17th Conference on Uncertainty in Artificial Intelligence*, pages 538–545. Morgan Kaufman Publishers, 2001. (Cited on page 31.)

[166] L. Weaver and N. Tao. The variance minimizing constant reward baseline for gradient-based reinforcement learning. Technical report no. 30, Australian National University, 2001. (Cited on page 31.)

[167] S. Weaver, L. Baird, and M. Polycarpou. An Analytical Framework for Local Feedforward Networks. *IEEE Transactions on Neural Networks*, 9(3), 1998. (Cited on page 66.)

[168] L.L. Whitcomb. Underwater robotics: out of the research laboratory and into the field. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 85–90, 2000. (Cited on page 108.)

[169] L.L. Whitcomb and D.R. Yoerger. Development, comparison and preliminary experimental validation of nonlinear dynamic thruster models. *IEEE Journal of Oceanic Engineering*, 24:481–494, 1999. (Cited on page 83.)

[170] R.J. Williams. Toward a theory of reinforcement learning. Technical report nu-ccs-88-3, College of Computer Science, Northeastern University, 360 Huntingdon Avenue, Boston, MA, 1988. (Cited on page 30.)

[171] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Journal of Machine Learning*, 8:229–256, 1992. (Cited on pages 30 and 31.)

[172] I.H. Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34:286–295, 1977. (Cited on pages 39 and 41.)

[173] W. Zhang and T.G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *International Joint Conference on Artificial Intelligence, IJCAI*, 1995. (Cited on pages 62 and 66.)