Studying the feasibility of policy reinforcement learning methods for autonomous agents

Andrés El-Fakdi, Marc Carreras, Pere Ridao, Emili Hernández
Institute of Informatics and Applications
University of Girona
Edifici Politecnica 4, Campus Montilivi
17071 Girona
aelfakdi@eia.udg.es

Abstract

Autonomous Underwater Vehicles (AUV) represent a challenging control problem with complex, noisy, dynamics. Nowadays, not only the continuous scientific advances in underwater robotics but the increasing number of sub sea missions and its complexity ask for an automatization of submarine processes. This paper proposes a high-level control system for solving the action selection problem of an autonomous robot. The system is characterized by the use of Reinforcement Learning Direct Policy Search methods (RLDPS) for learning the internal state/action mapping of some behaviors. We demonstrate its feasibility with simulated experiments using the model of our underwater robot URIS in a target following task.

1. Introduction

A commonly used methodology in robot learning is Reinforcement Learning (RL) [1]. In RL, an agent tries to maximize a scalar evaluation (reward or punishment) obtained as a result of its interaction with the environment. The goal of a RL system is to find an optimal policy which maps the state of the environment to an action which in turn will maximize the accumulated future rewards. Most RL techniques are based on *Finite Markov Decision Processes* (FMDP) causing finite state and action spaces. The main advantage of RL is that it does not use any knowledge database, so the learner is not told what to do as occurs in most forms of machine learning, but instead must discover actions yield the most reward by trying them. Therefore, this class of learning is suitable for online robot learning. The main disadvantages are a long convergence time and the lack of generalization among continuous variables.

In order to solve such problems, most of RL applications require the use of generalizing function approximators such artificial neural-networks (ANNs), instance-based methods or decision-trees. As a result, many RL-based control systems have been applied to robotics over the past decade. In [2], an instance-based learning algorithm was applied to a real robot in a corridor-following task. For the same task, in [3] a hierarchical memory-based RL was proposed. Also, some RL applications on autonomous helicopter flights [17], optimization of robot locomotion movements [19] and robot weightlifting task [18] have obtained good results as well.

The dominant approach has been the value-function approach, and although it has demonstrated to work well in many applications, it has several limitations, too. If the state-space is not completely observable (POMDP), small changes in the estimated value of an action cause it to be, or not be, selected; and this will detonate in convergence problems [4].

Over the past few years, studies have shown that approximating directly a policy can be easier than working with value functions, and better results can be obtained [5,6]. Instead of approximating a value function, new methodologies approximate a policy using an independent function approximator with its own parameters, trying to maximize the expected reward. Examples of direct policy methods are the REINFORCE algorithm [7], the direct-gradient algorithm [8] and certain variants of the actor-critic framework [9].

The advantages of policy methods against value-function based methods are various. A problem for which the policy is easier to represent should be solved using policy algorithms [6]. Working this way should represent a decrease in the computational complexity and, for learning control systems which operate in the physical world, the reduction in time-consuming would be notorious. Furthermore, learning systems should be designed to explicitly account for the resulting violations of the Markov property. Studies have shown that stochastic policy-only methods can obtain better results when working in POMDP than those ones obtained with deterministic value-function methods [10]. On the other side, policy methods learn much more slowly than RL algorithms using value function [5] and they typically find only local optima of the expected reward [11].

In this paper we propose an on-line direct policy search algorithm based on Baxter and Bartlett's direct-gradient algorithm OLPOMDP [12] applied to a real learning control system in which a simulated model of the AUV URIS [13] navigates a two-dimensional world. The policy is represented by a neural network whose input is a representation of the state, whose output is action selection probabilities, and whose weights are the policy parameters. The proposed method is based on a stochastic gradient descent with respect to the policy parameter space, it does not need a model of the environment to be given and it is incremental, requiring only a constant amount of computation step. The objective of the agent is to compute a stochastic policy [10], which assigns a probability over each action. Results obtained in simulation show the viability of the algorithm in a real-time system.

The structure of the paper is as follows. In section II the direct-policy search algorithm is detailed. In section III a description of all the elements that affect our problem (the world, the robot and the controller) are commented. The simulated experiment description and the results obtained are included in section IV and finally, some conclusions and further work are included in section V.

2. The RLDPS Algorithm

A partially observable Markov decision process (POMDP) consists of a state space *S*, an observation space *Y* and a control space *U*. For each state $i \in S$ there is a deterministic reward *r(i)*. As mentioned before, the algorithm applied is designed to work on-line so at every time step, the learner (our vehicle) will be given an observation of the state and, according to the policy followed at that moment, it will generate a control action. As a result, the learner will be driven to another state and will receive a reward associated to this new state. This reward will allow us to update the controller's parameters that define the policy followed at every iteration, resulting in a final policy considered to be optimal or closer to optimal. The algorithm procedure is summarized in Table 1.

TABLE 1
Algorithm: Baxter & Bartlett's OLPOMDP

```
Given:
    • $T > 0$
    • Initial parameter values $\theta_0 \in \mathbb{R}^K$
    • Arbitrary starting state $i_0$
Set $z_0 = 0$ ( $z_0 \in \mathbb{R}^K$ )
for $t = 0$ to $T$ do
    Observe state $y_t$
    Generate control action $u_t$ according to current policy $\mu(\theta, y_t)$
    Observe the reward obtained $r(i_{t+1})$
    Set $z_{t+1} = \beta z_t + \frac{\nabla \mu_{u_t}(\theta, y_t)}{\mu_{u_t}(\theta, y_t)}$
    Set $\theta_{t+1} = \theta_t + \alpha r(i_{t+1}) z_{t+1}$
end for
```

The algorithm works as follows: having initialized the parameters vector θ_0, the initial state i_0 and the gradient $z_0 = 0$, the learning procedure will be iterated T times. At every iteration, the parameters gradient z_t will be updated. According to the immediate reward received $r(i_{t+1})$, the new gradient vector z_{t+1} and a fixed learning paramenter α, the new paramenter vector θ_{t+1} can be calculated. The current policy μ_t is directly modified by the new parameters becoming a new policy μ_{t+1} that will be followed next iteration, getting closer, as $t \to T$ to a final policy μ_T that represents a correct solution of the problem.

In order to clarify the steps taken, the next lines will relate the update parameter procedure of the algorithm closely. The controller uses a neural network as a function approximator that generates a stochastic policy. Its weights are the policy parameters that are updated on-line every time step. The accuracy of the approximation is controlled by the parameter $\beta \in [0,1)$.

The first step in the weight update procedure is to compute the ratio:

$$\frac{\nabla \mu_{u_t}(\theta, y_t)}{\mu_{u_t}(\theta, y_t)} \tag{1}$$

for every weight of the network. In AANs like the one used in the algorithm the expression defined in step 7 of Table 1 can be rewritten as:

$$z_{t+1} = \beta z_t + \delta_t y_t \tag{2}$$

At any step time *t*, the term z_t represents the estimated gradient of the reinforcement sum with respect to the network's layer weights. In addition, δ_t refers to the local gradient associated to a single neuron of the ANN and it is multiplied by the input to that neuron y_t. In order to compute these gradients, we evaluate the soft-max distribution for each possible future state exponentiating the real-valued ANN outputs $\{o_1, \ldots, o_n\}$ being *n* the number of neurons of the output layer [14].

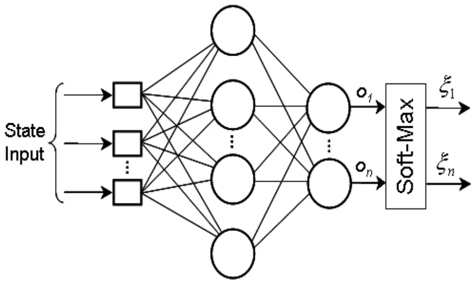

Figure 1. Schema of the ANN arquitecture used.

After applying the soft-max function, the outputs of the neural network give a weighting, $\xi_j \in (0,1)$ to each of the vehicle's thrust combinations. Finally, the probability of the i^{th} thrust combination is then given by:

$$\Pr_i = \frac{\exp(o_i)}{\sum_{z=1}^{n} \exp(o_z)} \tag{3}$$

Actions have been labeled with the associated thrust combination, and they are chosen at random from this probability distribution.

Once we have computed the output distribution over the possible control actions, next step is to calculate the gradient for the action chosen by applying the chain rule; the whole expression is implemented similarly to *error back propagation* [15]. Before computing the gradient, the error on the neurons of the output layer must be calculated. This error is given by expression (4).

$$e_j = d_j - \Pr_j \tag{4}$$

The desired output d_j will be equal to 1 if the action selected was o_j, and 0 otherwise (see Fig. 2).

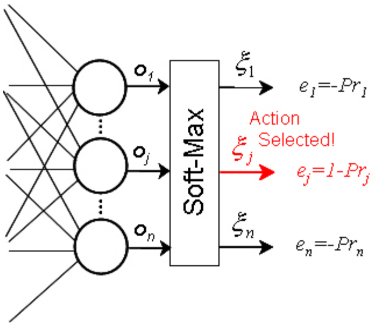

Figure 2. Soft-Max error computation for every output.

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

$$\delta_j^o = e_j \cdot \varphi_j^{'}(o_j) \qquad (5)$$

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

$$\delta_j^h = \varphi_j^{'}(o_j) \sum_k \delta_k w_{kj} \qquad (6)$$

When computing the gradient of a hidden-layer neuron, the previously obtained gradient of the following layers must be back propagated. In (6) the term $\varphi_j^{'}(o_j)$ represents de derivative of the activation function associated to that neuron, o_j is the function signal at the output for that neuron and finally the summation term includes the different gradients of the following neurons back propagated by multiplying each gradient to its corresponding weighting (see Fig. 3).

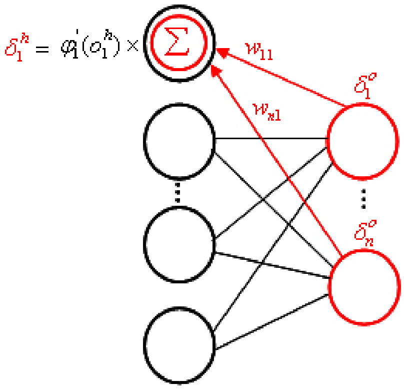

Figure 3. Gradient computation for a hidden-layer neuron.

Having all local gradients of the all neurons calculated, the expression in (2) can be obtained and finally, the old parameters are updated following the expression:

$$\theta_{t+1} = \theta_t + \gamma r(i_{t+1}) z_{t+1} \qquad (7)$$

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

3. Case Study: Target Following

The following lines are going to describe the different elements that take place in our problem. First, the simulated world will be detailed, in a second place we will present the underwater vehicle URIS and its model used in our simulation. At last, a description of the neural-network controller is presented.

3.1. The world

As mentioned before, the problem deals with the simulated model of the AUV URIS navigating a two-dimensional world constrained in a plane region without boundaries. The vehicle can be controlled in two degrees of freedom (DOFs), surge (X movement) and sway (Y movement) by applying 4 different control actions: a force in either the positive or negative x direction, and another force in either the positive or negative y direction.

The simulated robot was given a reward of 0 if the vehicle reaches the objective position (if the robot enters inside a circle of 2 units radius, the target is considered reached) and a reward equal to -1 in all other states. To encourage the controller to learn to navigate the robot to the target independently of the starting state, the AUV position was reset every 50 (simulated) seconds to a random location in x and y between [-20, 20], and at the same time target position was set to a random location within the same boundaries. The sample time is set to 0.1 seconds.

3.2. URIS AUV description

The Autonomous Underwater Vehicle URIS (Fig. 4) is an experimental robot developed at the University of Girona with the aim of building a small-sized UUV. The hull is composed of a stainless steel sphere with a diameter of 350 mm, designed to withstand pressures of 4 atmospheres (30 meters depth).

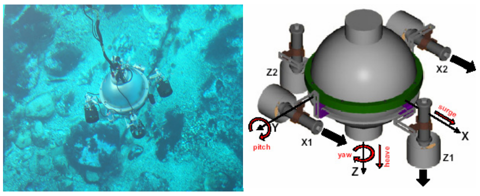

Figure 4. (Left) URIS in experimental test. (Right) Robot reference frame

The experiments carried out use the mathematical model of URIS computed by means of parameter identification methods [13]. The whole model has been adapted to the problem so the hydrodinamic equation of motion of an underwater vehicle with 6 DOFs [16] has been uncoupled and reduced to modellate a robot with two DOFs. Let us consider the dynamic equation for the *surge* DOF:

$$\dot{u} = \underbrace{\frac{X}{(m-X_{\dot{u}})}}_{\gamma} - \underbrace{\frac{X_u}{(m-X_{\dot{u}})}}_{\alpha} \cdot u - \underbrace{\frac{X_{u|u|}\,|u|}{(m-X_{\dot{u}})}}_{\beta} \cdot u + \underbrace{\frac{\tau_p}{(m-X_{\dot{u}})}}_{\delta} \qquad (8)$$

$$\dot{v} = \underbrace{\frac{Y}{(m-Y_{\dot{v}})}}_{\gamma} - \underbrace{\frac{Y_v}{(m-Y_{\dot{v}})}}_{\alpha} \cdot v - \underbrace{\frac{Y_{v|v|}\,|v|}{(m-Y_{\dot{v}})}}_{\beta} \cdot v + \underbrace{\frac{\tau_p}{(m-Y_{\dot{v}})}}_{\delta} \qquad (9)$$

Then, due to identification procedure [13], expressions in (8) and (9) can be rewritten as follows:

$$\dot{v}_x = \alpha_x v_x + \beta_x v_x |v_x| + \gamma_x \tau_x + \delta_x \qquad (10)$$

$$\dot{v}_y = \alpha_y v_y + \beta_y v_y |v_y| + \gamma_y \tau_y + \delta_y \qquad (11)$$

Where $\dot{v}_x$ and $\dot{v}_y$ represent de acceleration in both surge and sway direction, v_x and v_y are the linear velocity in surge and sway. The forces excerted by the thrusters in both DOFs are indicated as τ_x and τ_y. The model parameters for both DOFs are stated as follows: α and β coeficients refer to the linear and the quadratic damping forces, γ represent a mass coeficient and the bias term is introduced by δ. The identified parameters values of the model are indicated in Table 2.

TABLE 2
URIS Model Parameters for Surge and Sway

	α	β	γ	δ
Units	$\left(\frac{N\cdot s}{Kg\cdot m}\right)$	$\left(\frac{N\cdot s^2}{Kg\cdot m^2}\right)$	Kg^{-1}	$\left(\frac{N}{Kg}\right)$
Surge	-0.3222	0	0.0184	0.0012
Sway	-0.3222	0	0.0184	0.0012

3.3. The Controller

A one-hidden-layer neural-network with 4 input nodes, 3 hidden nodes and 4 output nodes has been used to generate a stochastic policy. Two of the inputs correspond to the x and y vehicle's distance to target and the other two represent the x and y velocities at the current time-step. Each hidden and output layer has the usual additional bias term. The activation function used for the neurons of the hidden layer is the hyperbolic tangent type (12, Fig. 5), while the output layer nodes are linear. The four output neurons have been exponentiated and normalized as explained in section 2 to produce a probability distribution. Control actions are selected at random from this distribution.

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} \qquad (12)$$

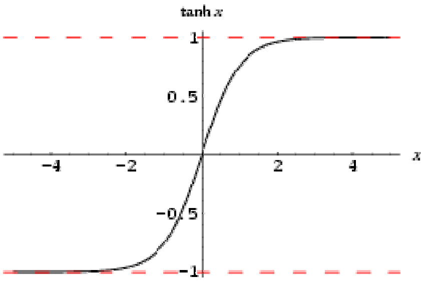

Figure 5. The hyperbolic tangent function.

4. Simulated Results

The controller was trained, as commmented in section 3, in an episodic task. Robot and target positions are reseted every 50 seconds so the total amount of reward per episode percieved varies depending on the episode. Even though the results presented have been obtained as explained in section 3, in order to clarify the graphical results of time convergence of the algorithm, for the plots below some constrains have been applied to the simulator: Target initial position is fixed to (0,0) and robot initial location has been set to four random locations, $x = \pm 20$ and $y = \pm 20$, therefore, the total amount per episode when converged to minima will be the same.

The number of episodes to be done has been set to 100.000. For every episode, the total amount of reward percieved is calculated. Figure 6 represents the performance of the neural-network vehicle controller as a function of the number of episodes, when trained using OLPOMDP. The episodes have been averaged over bins of 50 episodes. The experiment has been repeated in 100 independent runs, and the results presented are a mean over these runs.

The simulated experiments have been repeated and compared for different values of α and β.

For $\alpha = 0.000001$:

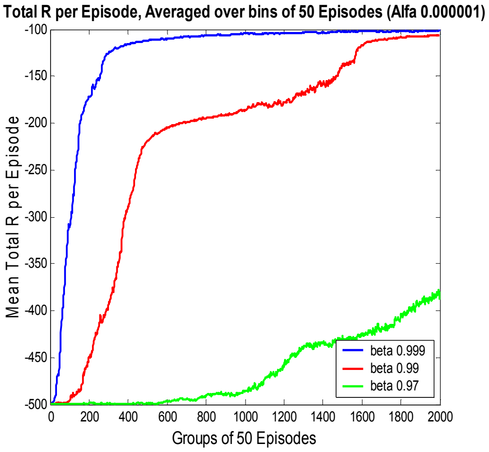

Figure 6. Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generatedby simulating 100.000 episodes, and averaging them over bins of 50 episodes. Process repeated in 100 independent runs. The results are a mean of these runs. Fixed $\alpha = 0.000001$, for different values of $\beta = 0.999$, $\beta = 0.99$ and $\beta = 0.97$

For $\alpha = 0.00001$:

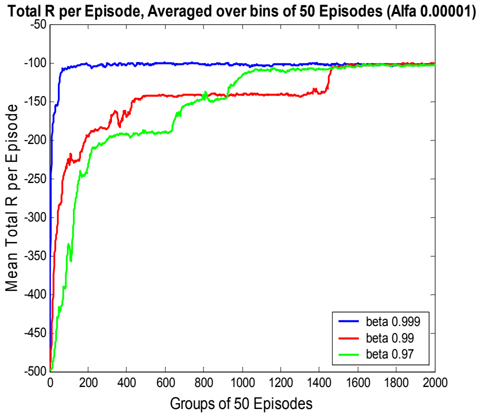

Figure 7. Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generatedby simulating 100.000 episodes, and averaging them over bins of 50 episodes. Process repeated in 100 independent runs. The results are a mean of these runs. Fixed $\alpha = 0.00001$, for different values of $\beta = 0.999$, $\beta = 0.99$ and $\beta = 0.97$.

For $\alpha = 0.0001$:

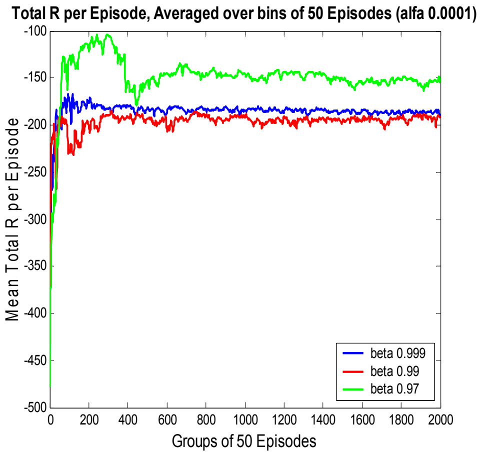

Figure 8. Performance of the neural-network puck controller as a function of the number of episodes. Performance estimates were generatedby simulating 100.000 episodes, and averaging them over bins of 50 episodes. Process repeated in 100 independent runs. The results are a mean of these runs. Fixed $\alpha = 0.0001$, for different values of $\beta = 0.999$, $\beta = 0.99$ and $\beta = 0.97$.

As it can bee apreciated in Fig. 7, the optimal performance (within the neural network controller used here) is around -100 for this simulated problem, due to the fact that the puck and target locations are reset every 50 seconds and for this reason the vehicle must be away from target a fraction of the time. The best results are obtained when $\alpha = 0.00001$ and $\beta = 0.999$, see Fig. 7.

Figure 9 represents the behavior of the trained robot controller. For the purpose of the illustration, only target location has been reseted to random location, not the robot location.

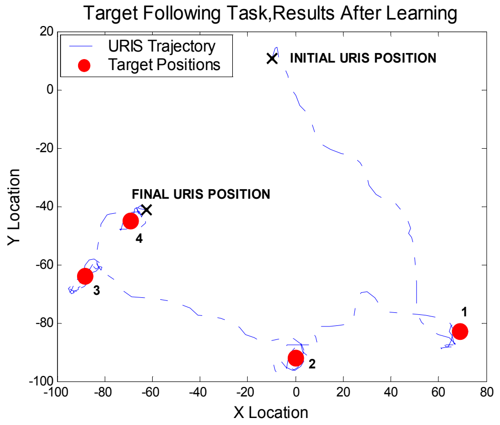

Figure 9. Behavior of a trained robot controller, results of target following task asfter learning period is completed.

5. Conclusions

An on-line direct policy search algorithm for AUV control based on Baxter and Bartlett's direct-gradient algorithm OLPOMDP has been proposed. The method has been applied to a real learning control system in which a simulated model of the AUV URIS navigates a two-dimensional world in a target following task. The policy is represented by a neural network whose input is a representation of the state, whose output is action selection probabilities, and whose weights are the policy parameters. The objective of the agent was to compute a stochastic policy, which assigns a probability over each of the four possible control actions.

Results obtained confirm some of the ideas presented in section 1. The algorithm is easier to implement compared with other RL methodologies like value function algorithms and it represents a considerable reduction of the computational time of the algorithm. On the other side, simulated results show a poor speed of convergence towards minimal solution.

In order to validate the performance of the method proposed, future experiments are centered on obtaining empirical results: the algorithm must be tested on real URIS in a real environment. Previous investigations carried on in our laboratory with RL value functions methods with the same prototype URIS [20] will allow us to compare both results. At the same time, the work is focused in the development of a methodology to decrease the convergence time of the RLDPS algorithm.

Acknowledgments

This research was esponsored by the spanish commission MCYT (DPI2001-2311-C03-01). I would like to give my special thanks to Mr. Douglas Alexander Aberdeen of the Australian National University for his help.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*. MIT Press, 1998.

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

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

[4] D.P. Bertsekas and J.N. Tsitsiklis, Neuro-Dynamic Programming. Athena Scientific, 1996.

[5] R. Sutton, D. McAllester, S. Singh and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation" in Advances in Neural Information Processing Systems 12, pp. 1057-1063, MIT Press, 2000.

[6] C. Anderson, "Approximating a policy can be easier than approximating a value function" *Computer Science Technical Report*, CS-00-101, February 10, 2000.

[7] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning" in Machine Learning, 8, pp. 229-256, 1992.

[8] J. Baxter and P.L. Bartlett, "Direct gradient-based reinforcement learning" IEEE *International Symposium on Circuits and Systems*, May 28-31, Geneva, Switzerland, 2000.

[9] V.R. Konda and J.N. Tsitsiklis, "On actor-critic algorithms", in SIAM Journal on Control and Optimization, vol. 42, No. 4, pp. 1143-1166, 2003.

[10] S.P. Singh, T. Jaakkola, M.I. Jordan, "Learning without state-estimation in partially observable Markovian decision processes", in *Proceedings of the 11th International Conference on Machine Learning*, pp. 284-292, 1994.

[11] N. Meuleau, L. Peshkin and K. Kim, "Exploration in gradient-based reinforcement learning", Technical report AI Memo 2001-003, April 3, 2001.

[12] J. Baxter and P.L. Bartlett, "Direct gradient-based reinforcement learning I: Gradient estimation algorithms" Technical Report. Australian National University, 1999.

[13] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras, A. Zirilli, "On the identification of non-linear models of unmanned underwater vehicles" in Control Engineering Practice, vol. 12, pp. 1483-1499, 2004.

[14] D. A., Aberdeen, *Policy Gradient Algorithms for Partially Observable Markov Decision Processes*, PhD Thesis, Australian National University, 2003.

[15] S. Haykin, *Neural Networks, a comprehensive foundation*, Prentice Hall, Upper Saddle River, New Jersey, USA, 1999.

[16] T.I., Fossen, *Guidance and Control of Ocean Vehicles*, John Wiley and Sons, New York, USA, 1994.

[17] J. A. Bagnell, J. G. Schneider, "Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Korea, 2001.

[18] M. T. Rosenstein, A. G. Barto, "Robot Weightlifting by Direct Policy Search", in Proceedings of the International Joint Conference on Artificial Intelligence, 2001.

[19] N. Kohl, P. Stone, "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion", in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[20] M. Carreras, P. Ridao, A. El-Fakdi, "Semi-Online Neural-Q-Learning for Real-Time Robot Learning", in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, USA, 2003.