Proceedings of the 2002 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
EPFL, Lausanne, Switzerland • October 2002

Efficient Learning of Reactive Robot Behaviors with a Neural-Q_Learning Approach

Marc Carreras, Pere Ridao, Joan Batlle and Tudor Nicosevici

Institute of Informatics and Automation, University of Girona, Girona, Spain
{marcc,pere,jbatlle,tudor}@eia.udg.es

Abstract

The purpose of this paper is to propose a Neural-Q_learning approach designed for online learning of simple and reactive robot behaviors. In this approach, the Q_function is generalized by a multi-layer neural network allowing the use of continuous states and actions. The algorithm uses a database of the most recent learning samples to accelerate and guarantee the convergence. Each Neural-Q_learning function represents an independent, reactive and adaptive behavior which maps sensorial states to robot control actions. A group of these behaviors constitutes a reactive control scheme designed to fulfill simple missions. The paper centers on the description of the Neural-Q_learning based behaviors showing their performance with an underwater robot in a target following task. Real experiments demonstrate the convergence and stability of the learning system, pointing out its suitability for online robot learning. Advantages and limitations are discussed.

1. Introduction

A commonly used methodology in robot learning is Reinforcement Learning (RL) [9]. In RL, an agent tries to maximize a scalar evaluation (reward or punishment) 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, as do most forms of machine learning, making this class of learning suitable for online robot learning. The main disadvantages are a longer convergence time and the lack of generalization among continuous variables. The latter is one of the most active research topics in RL.

Many RL-based systems have been applied to robotics over the past few years. Most of them use classic RL algorithms combined with various methodologies which reduce the generalization problem. In [8], an instance-based learning algorithm was applied to a real robot in a corridor-following task. Also, for the same task, in [7] a hierarchical memory-based RL was proposed. A very common approach is the use of Q_learning [12] as the base of the learning algorithm due to the good learning capabilities in discrete domains: online and off-policy. Many generalization techniques have been applied to Q_learning. In [10] a memory-based implementation was proposed for vision-guided behavior acquisition and [11] shows a state/action quantification by a set of triangular patches. Also, many proposals combine Q_learning and Neural Networks (NN) (see [5] for an overview). Finally, several recent publications have pointed to the possibility of solving the RL problem by estimating the policy function instead of the value function. In [2], a practical application for controlling an autonomous helicopter was presented.

This paper proposes a *Neural-Q_learning* (NQL) approach designed for online learning of simple and reactive robot behaviors. Our approach differentiates from other NQL proposals in that we implement the Q-function into a NN directly instead of breaking the problem down into a finite set of actions, features or clusters. This NQL implementation, known as *direct Q_learning* [3], is the simplest and straightest way to generalize with a NN. This implies more learning capabilities and causes instability in the learning of the optimal state/action mapping. To avoid this problem, the proposed algorithm introduces a database of the most recent and representative learning samples from the whole state/action space. These samples are repeatedly used in the NN weight update phase, assuring the convergence of the NN to the optimal Q_function. To preserve the real time execution of the behavior, two different execution threads, one for learning and another for output generation, are used. This NQL algorithm was conceived to learn the internal state/action mapping of a reactive behavior. By combining several NQL-behaviors, a high level control scheme can be designed to achieve simple tasks with an autonomous robot. Behavior coordination is done

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.

through a hybrid coordinator [4] which does not need any tuning phase.

This paper demonstrates the feasibility of the proposed NQL algorithm with real experiments using the underwater vehicle URIS in a target following task, see figure 1. Experiments were carried out in a water tank. A reactive control system with four behaviors, wall avoidance, teleoperation, target following and target recovery, was used. The target following behavior was learnt using the proposed NQL algorithm. Results show the convergence of the NQL-based behavior into an optimal policy, and therefore, the achievement of the task.

The structure of this paper is as follows. In section 2, an overall description of the behavior-based control scheme is done. Section 3 describes the proposed Neural-Q_learning algorithm. In section 4, the experimental setup designed to carry out the target following task with URIS's AUV is presented. Section 5 shows and discusses the obtained results. And finally, conclusions are presented in section 6.

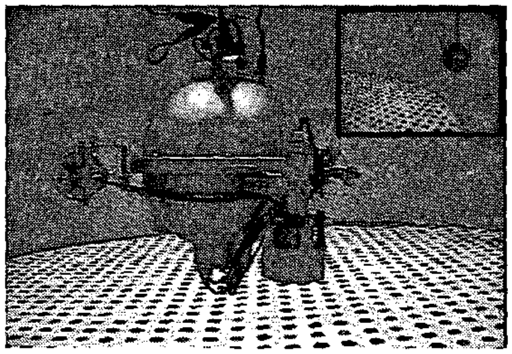

Figure 1: URIS's underwater vehicle during the target following task in a water tank.

2. Behavior-based Control Scheme

A set of simple behaviors and a coordinator constitute the behavior-based control scheme [1]. For a given task, the behaviors, with the corresponding priorities among them, must be determined. Each behavior has an independent goal which tries to accomplish by perceiving the state of the environment and proposing a control action.

The coordinator is in charge of choosing the final action to be followed. An *hybrid coordinator* [4] between competitive and cooperative methodologies is used. The general structure is shown in figure 2. This coordinator is based on normalized behavior outputs. Each output contains a three-dimensional vector "v_i" which represents the velocity proposed by the behavior and, associated with this vector, an activation level "a_i" indicates the level of need of controlling the robot. This value is between 0 and 1, see figure 3.

The hybrid coordination system is implemented with a set of *hierarchical hybrid nodes,* see figure 3. These nodes have two inputs and generate a merged normalized control response. One of the inputs is used by a *dominant* behavior which suppresses the responses of the *non-dominant* behavior when the first is completely activated (a_i=1). However, when the dominant behavior is only partially activated ($0<a_i<1$), the final response will be a combination of both inputs. The basic idea is to use the optimized paths from cooperation when the predominant behavior is not completely active. Non-dominant behaviors can modify the responses of dominant behaviors slightly when they aren't completely activated. When non-decisive situations occur, cooperation between behaviors is allowed. Nevertheless, robustness is present when dealing with critical situations. The hybrid nodes do not need any tuning phase. The coordination of a set of behaviors is defined hierarchically, classifying each behavior depending on its priority.

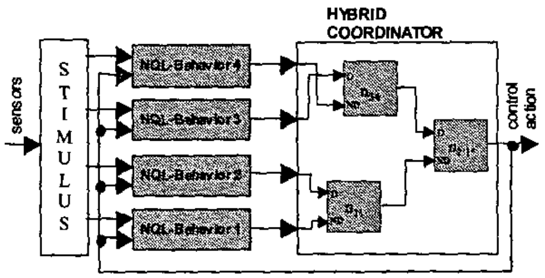

Figure 2: Behavior-based architecture with the hybrid coordination system.

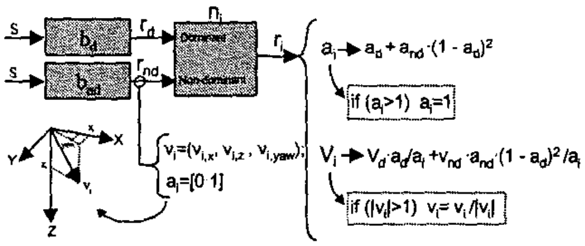
s
s
b_d
b_nd
r_d
r_nd
n_i
Dominant
Non-dominant
r_i
a_i → a_d + a_nd·(1 - a_d)²
if (a_i>1) a_i=1
v_i=(v_i,x, v_i,z, v_i,yaw);
a_i=[0 1]
V_i → V_d·a_d/a_i + v_nd·a_nd·(1 - a_d)²/a_i
if (|v_i|>1) v_i= v_i/|v_i|
X
Y
Z
x
v_i

Figure 3: Normalized output of a behavior and equations of the hierarchical hybrid node.

3. Neural-Q_learning Based Behaviors

A Neural-Q_learning approach is used to learn the mapping between the state and action spaces (policy). The state space is the sensor information perceived by the robot and is needed by the behavior in order to accomplish its goal. The action space is the velocity set-points the robot should follow.

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.

3.1 Q_learning

Q-learning [12] is a temporal difference algorithm, see [9], designed to solve the reinforcement learning problem (RLP). Temporal difference algorithms solve the RLP without knowing the transition probabilities between the states of the Finite Markov Decision Problem (FMDP), and therefore, in our context, the dynamics of the robot environment does not have to be known. Temporal difference methods are also suitable for learning incrementally, or online robot learning. The importance of online learning resides in the possibility of executing new behaviors without any previous phases such as "on-site manual tuning" or "data collection + offline learning". Another important characteristic of Q_learning is that it is an off-policy algorithm. The optimal state/action mapping is learnt independently of the policy being followed, which is very important in our approach because all the behaviors can be learnt even if they are not controlling the vehicle.

The original Q_learning algorithm is based on FMDPs. It uses the perceived states (*s*), the taken actions (*a*) and the received reinforcements (*r*) to update the values of a table, denoted as *Q(s,a)* or Q-function. If state/action pairs are continually visited, the Q values converge to a *greedy policy*, in which the maximum Q value for a given state points to the optimal action. Figure 4 shows the Q_learning algorithm. There are several parameters which define the learning evolution:

- γ: discount rate [0 1]. Maximization of future rewards. For γ=0, only immediate rewards are used.
- α: learning rate [0 1].
- ϵ: random action probability [0 1]. Exploitation versus exploration. ϵ-*greedy* action.

```
1. Initialize $\hat{Q}(s,a)$ arbitrarily
2. Repeat:
   (a) $s_t$ ← the current state
   (b) choose an action $a_t$ that maximizes $\hat{Q}(s_t,a)$ over all $a$
   (c) $\varepsilon$-greedy action, carry out action $a_t$ in the world with probability
       $(1-\varepsilon)$, otherwise apply a random action (exploration)
   (d) Let the short term reward be $r_t$, and the new state be $s_{t+1}$
   (e) $\hat{Q}(s_t,a_t) = \hat{Q}(s_t,a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t)]$
```

***Figure 4:** Q_learning algorithm.*

3.2 Neural Q_learning

When working with continuous states and actions, as is usual in robotics, the Q-function table becomes too large for the required state/action resolution. In these cases, tabular Q-learning needs a very long learning time and memory requirements which makes the implementation of the algorithm in a real-time control architecture impractical. The use of a Neural Network (NN) to generalize among states and actions reduces the number of values stored in the Q-function table to a set of NN weights. The implementation of a feed-forward NN with the backpropagation algorithm [6] is known as *direct Q_learning* [3].

The Direct Q-learning algorithm has no convergence proofs and turned out to be unstable when we tried to learn a behavior. The instability was caused by the lack of weight updating in the whole state/action space. The optimal Q-function was only learnt in the current state zone. The Q-values learnt in past states were not maintained and therefore had to be learnt each time causing the instability.

To solve this limitation the proposed Neural-Q_learning based behaviors maintain a database of the most recent *learning samples*. All the samples of this database are used at each iteration to update the weights of the NN. This assures a generalization in the whole visited state/action space instead of a local generalization in the current visited space. Each learning sample is composed of the initial state s_t, the action a_t, the new state s_{t+1} and the reward r_t. The action used by the NQL behaviors, is the one sent by the coordinator to the low-level control system. For this reason, a feedback of the last generated control action is needed, see figure 2. Finally, in order to prevent a huge database, each new learning sample substitutes old samples closer than a threshold. The distance between samples is geometrically computed from both $\{s_t, a_t, r_t\}$ vectors. It is important to maintain a database with the most recent samples to keep the current dynamics of the environment.

The structure and phases of the proposed *neural Q_learning* algorithm is shown in figure 5. The Q_function approximated by the NN is:

$$\hat{Q}(s_t,a_t) = r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}) \qquad (1)$$

Therefore, its inputs are the continuous state and actions, and the output is the Q_value. According to the output value, the error is found and the weights are updated using the standard backpropagation algorithm. A two layer NN has been used with a hyperbolic tangent and lineal activation functions for the first and second layers respectively. Weights are initialized randomly. To find the action which maximizes the Q_value, the network evaluates all the possible actions which could be applied. Although actions are continuous, a finite set, which guarantees sufficient resolution, is used.

To maintain the real time execution of the behavior, two execution threads are used. The more priority thread is in charge of acquiring new learning samples and generating control actions at the frequency of the behavior-based control system. Phases 1,2 and 4 are computed in this thread, see figure 5. The second thread is used to continually update the weights of the NN, phase 3. It has less priority than the rest of the control processes and therefore it will use the remaining computational power to learn the NQL-based behaviors.

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.

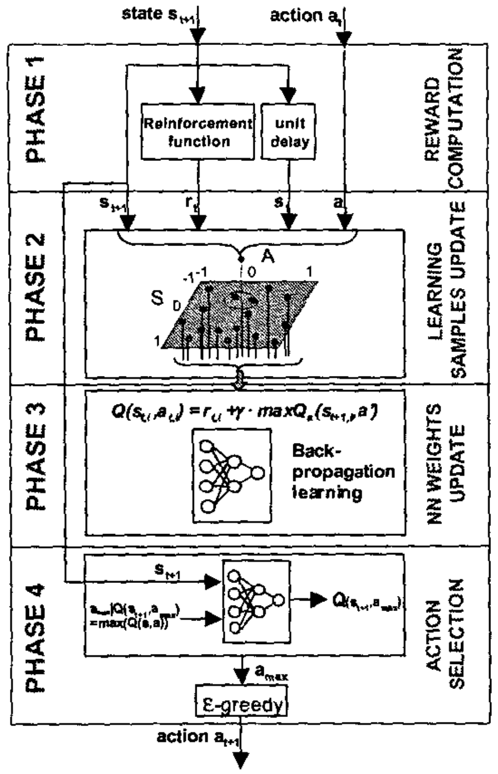

Figure 5: Neural Q_learning algorithm structure.

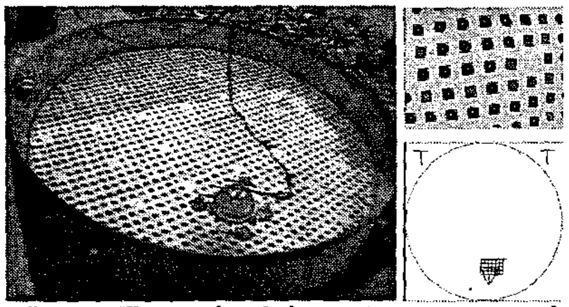

Figure 6: Water tank with the positioning pattern used in the experiments with URIS.

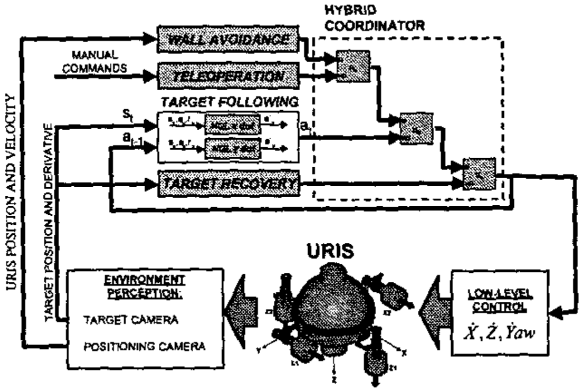

Figure 7: URIS's target following task scheme.

3.3 Reinforcement Function

The reinforcement function determines the policy learnt by the behavior. The definition of this function requires knowledge from a human designer. The function associates each state with a reward "*r*". In our approach, we have reduced the possible values to three : {-1, 0, 1}. By associating the desired states with "*r=1*" and the undesired with "*r = -1*", the algorithm learns how to act.

4. URIS's Experimental Setup

The proposed target following task, to test the NQL-based behaviors, consisted of following a target by means of a color camera. Experiments were carried out with the small-sized underwater robot URIS (hull ∅ 0.35m), an Autonomous Underwater Vehicle (AUV) developed at the University of Girona. URIS is a non-holonomic robot stable in *pitch* and *roll*, where only x, z and yaw degrees of freedom (DOF) can be controlled.

A water tank (4.5 m. diameter and 1.2 m. depth) was used to test the control system and to perform the learning experiments, see figure 6. Due to the shallowness of the tank, the vehicle was only moved in the horizontal plane maintaining an intermediate depth. The position and the velocity of the vehicle was computed by a computer vision system which used a down looking camera attached to URIS and a coded pattern placed on the bottom of the tank. The pattern contains several dots with two different colors, gray and black. The vision system estimates the six-dimensional position of the vehicle by tracking the detected dots and solving the equations of the projective geometry. The pattern also contains some marks to reset the estimated position to an absolute position inside the water tank. The vehicle's velocity is also estimated. The main utility of this positioning system was to provide velocity feedback to the low level controller ($\dot{x},\dot{z},y\dot{a}w$). Also the absolute position was used by the behavior-based high level control system.

The control architecture of the robot was based on a real time distributed object oriented framework which assures the deadlines of all the task. Three different computers were used in these experiments, the onboard PC-104 which runs QNX and contains all the control objects, an external PC running Windows which computes the vision objects, and finally, another external PC running QNX used as HMI.

The control scheme of the target following task can be seen in figure 7. Four behaviors were used. The wall avoidance behavior had the highest priority and prevented the vehicle to collide with the walls of the water tank. The teleoperation behavior was used to move manually the robot. The target following behavior was the one learnt with the proposed NQL algorithm. And finally, the target recovery behavior spun the robot in order to find the lost target. The output action of each

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.

behavior was a 2D-speed vector (X and Yaw velocities) and an activation level, which determined the final output according to the hierarchy of behaviors.

5. Results

As previously stated, the experiments were designed to test the feasibility of the NQL-based behaviors. The target following behavior was implemented with 2 different NQL algorithms (one for each DOF, X and Yaw). In these experiments, only one NQL algorithm was learnt at a time, no simultaneous learning between different DOFs or behaviors was tested. Each DOF received information about the current state and the last action taken. The state was the position of the target in the image as well as its derivative. A reinforcement function gave different rewards (-1, 0 or 1) depending on the distance at which the target was detected. Activation was 1 when the target was seen.

Several trials were carried out in order to find the best learning performance. The sample time of the NQL-based behaviors was set at 0.3 [s], while the one of the low level controllers was 0.1 [s]. The robot was able to learn how to move in both DOFs achieving the target following task. The parameters we used in the NQL algorithms can be seen in table 1, and figure 8 shows the normalization used for the target states. The learnt state/action optimal mappings can be seen in figures 9 and 10. Note the obtained non-lineal mappings.

The NQL behaviors showed a good robustness and presented a small convergence time (only 40 [s]). Figure 11 shows six consecutive learning attempts by the target following behavior in the Yaw DOF. The figure also shows that the averaged reward increased, demonstrating that the behavior was being learnt. In this experiment, the robot was learning how to turn in order to keep the target in the center of the image. It can be seen that the algorithm starts exploring the state in order to find maximum rewards. Once the whole state had been explored, the algorithm exploited the learnt Q_function and obtained the maximum rewards.

Figures 12 and 13 show a typical online learning evolution of the X and Yaw DOFs respectively. It can be seen how the vehicle explored the state in the learning phase. For the X DOF, the learning time was 100 seconds and for the Yaw DOF was 60 seconds approximately. This difference was due to the fact that the optimal mapping for the X DOF is more nonlinear than for the Yaw DOF, see figures 9 and 10. Immediately after the learning phase, the behavior was tested applying with the teleoperation behavior, an action that moved the vehicle away from the target. It can be seen that when the target following behavior took control, the target was reached again.

6. Conclusions

A proposal of Neural-Q_learning based behaviors have been presented and tested with a real robot in a target following task. The approach proved suitable for learning behaviors from a reactive control scheme. The Neural Network generalization of the Q_function was able to map an optimal state/action policy in a short time. The use of a database with the most significant learning samples assures the convergence of the algorithm. The paper has demonstrated the algorithm's suitability showing real results with the underwater robot URIS. In the presented work, each DOF was learnt independently. Future work will concentrate on simultaneously learning different DOFs and behaviors.

7. Acknowledgments

This research was sponsored by the Spanish commission MCYT (DPI2001-2311-C03-01).

8. References

[1] R. Arkin, *Behavior-based Robotics*. MIT Press, 1998.

[2] J.A. Bagnell and J.G. Schneider, *'Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods'*, IEEE International Conference on Robotics & Automation, Korea, 2001.

[3] K. Baird, 'Residual Algorithms: Reinforcement Learning with Function Approximation', *Machine Learning: Twelfth International Conference*, San Francisco, USA, 1995.

[4] M. Carreras, J. Batlle and P. Ridao, 'Hybrid Coordination of Reinforcement Learning-based Behaviors for AUV control', *IEEE/RSJ IROS*, Hawaii, USA 2001.

[5] C. Gaskett, D. Wettergreen and A. Zelinsky, 'Q-learning in continuous state and action spaces', *Proc. of the 12th Australian Joint Conference on Artificial Intelligence*, Sydney, Australia, 1999.

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

[7] N. Hernandez and S. Mahadevan, 'Hierarchical Memory-Based Reinforcement Learning', *Fifteenth International Conference on Neural Information Processing Systems*, Denver, USA, 2000.

[8] W.D. Smart and L.P. Kaelbling, 'Practical Reinforcement Learning in Continuous Spaces', *Intern. Conference on Machine Learning*, 2000.

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

[10] Y. Takahashi, M. Takada, and M. Asada, 'Continuous Valued Q-learning for Vision-Guided Behavior Acquisition'. *Intern. Conference on Multisensor Fusion and Integration for Intelligent Systems*, 1999.

[11] M. Takeda, T. Nakamura and T. Ogasawara, 'Continuous Valued Q-learning Method Able to Incrementally Refine State Space', *IEEE/RSJ IROS*, Hawaii, USA 2001.

[12] C.J.C.H. Watkins and P. Dayan, 'Q-learning', *Machine Learning*, 8:279-292, 1992.

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.

Table 1: Specifications of the target following NQL behavior

Sensors	color camera + target detection by color segmentation + object tracking	
Codification	$[f_x, f_y]$: normalized target position [-1, 1]	
Activation	if target detected $a_f = 1$; else $a_f = 0$	
DOF	**X**	**YAW**
State	f_x : position fv_x : derivative	f_y : position fv_y : derivative
Action	a_{fx} : lineal speed (X)	a_{fy}:angular speed (yaw)

Reinforcement function	If $0.4>f_x>0.1$: $r_{fx} = 1$
	if $0.5>f_x>0.4$: $r_{fx} = 0$
	else $r_{fx} = -1$
		If $\|f_y\| < 0.2$: $r_{fy} = 1$
		else if $\|f_y\| < 0.4$: $r_{fy} = 0$
		else $r_{fy} = -1$

NQL parameters	$\alpha=0.1$; $\gamma=0.9$; $\epsilon=0.3$ database size = 50 samples	$\alpha=0.1$; $\gamma=0.9$; $\epsilon=0.3$ database size = 50 samples
NQL structure	inputs=3: $[f_x, fv_x, a_{fx}]$ output: Q_value layer 1: 6 neurons layer 2: 1 neuron	inputs= 3 : $[f_y, fv_y, a_{fy}]$ output: Q_value layer 1: 6 neurons layer 2: 1 neuron

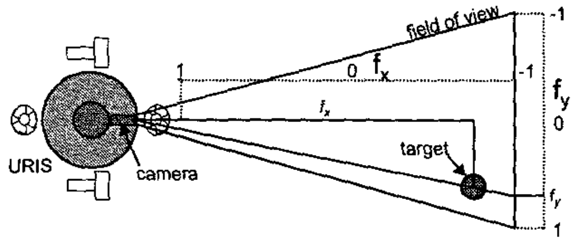

***Figure 8:** Coordinates of the target respect URIS.*

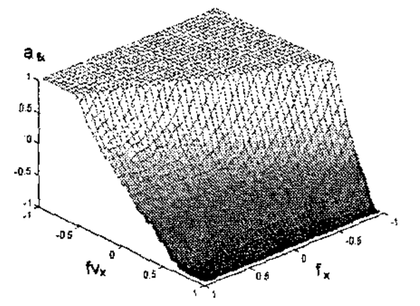

***Figure 9:** State/action mappings learnt for the target following behavior in the X DOF.*

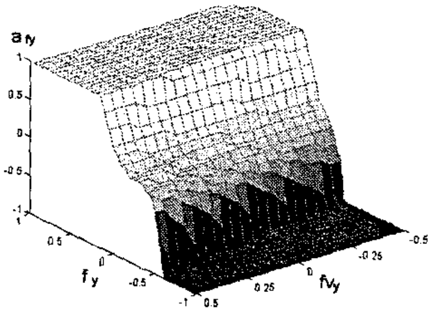

***Figure 10:** State/action mappings learnt for the target following behavior in the Yaw DOF.*

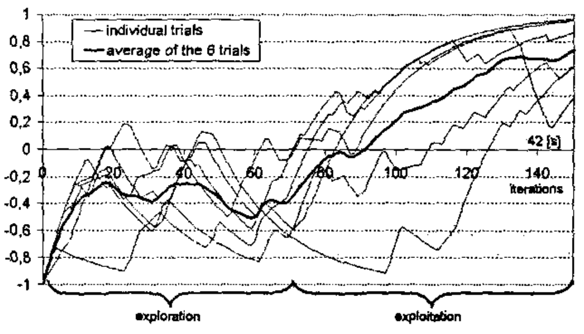

***Figure 11:** Behavior convergence proof. Results of six different learning trials of the target following behavior in the Yaw DOF. The averaged rewards over the last 20 iterations are shown. The average of the six experiments can also be seen. Accumulated rewards increased in measure as the behavior was learnt.*

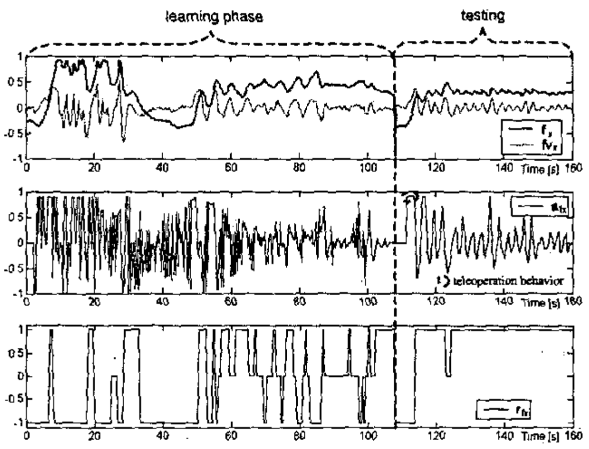

***Figure 12:** Online learning evolution and behavior testing of the target following behavior in the X DOF. The states, rewards and actions are shown above.*

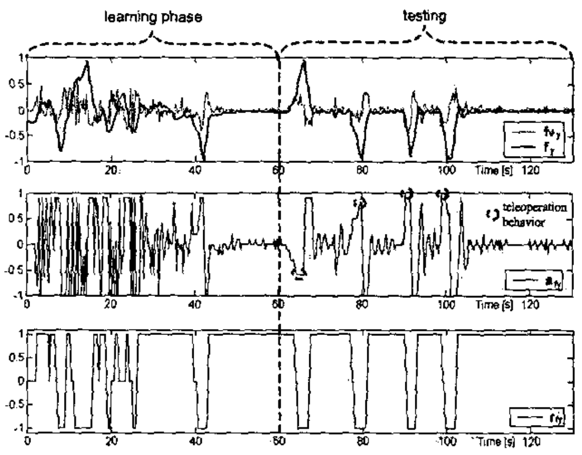

***Figure 13:** Online learning evolution and behavior testing of the target following behavior in the Yaw DOF. The states, rewards and actions are shown above.*

Authorized licensed use limited to: UNIVERSITAT DE GIRONA. Downloaded on April 26,2010 at 12:19:52 UTC from IEEE Xplore. Restrictions apply.