

HIGH-LEVEL CONTROL OF AUTONOMOUS ROBOTS USING A BEHAVIOR-BASED SCHEME AND REINFORCEMENT LEARNING

M. Carreras¹, J. Yuh² and J. Batlle¹

¹ *Institut d'Informàtica i Aplicacions
Universitat de Girona
Edifici Politècnica II, Campus Montilivi,
17071 Girona, Spain*

² *Autonomous Systems Laboratory
University of Hawaii
2540 Dole St., Holmes 302
Honolulu, HI 96822, USA*

Abstract - This paper proposes a behavior-based scheme for high-level control of autonomous robots. Two main characteristics can be highlighted in the control scheme. Behavior coordination is done through a hybrid methodology, which takes in advantages of the robustness and modularity in competitive approaches, as well as optimized trajectories in cooperative ones. As a second feature, behavior state/action mapping is learnt by means of Reinforcement Learning (RL). A new continuous approach of the Q-learning algorithm, implemented with a multi-layer neural network, is used. The behavior-based scheme attempts to fulfill simple missions in which several behaviors/tasks compete for the vehicle's control. This paper is centered in the RL-based behaviors. In order to test the feasibility of the proposed Neural-Q-learning scheme, real experiments with the underwater robot ODIN in a target following behavior were done. Results showed the convergence of the behavior into an optimal state/action mapping. Discussion about the proposed approach is given, as well as an overall description of the high level control scheme. *Copyright © 2002 IFAC*

Keywords: Autonomous vehicles, decentralized control, learning algorithms, neural networks, robot navigation.

1. INTRODUCTION

The control of an autonomous vehicle to fulfill a mission in an unstructured and unknown environment is still a challenge. In the middle of 1980s the appearance of *Behavior-based Robotics* (Arkin, 1998) philosophy revolutionized the development of robots. Its principles of parallelism, modularity, situatedness/embeddedness and behavior emergence provided a more feasible approach than traditional top-down *deliberative* architectures. Behavior-based control architectures propose a bottom-up methodology in which several behaviors or tasks act independently generating set-points to be followed by the robot. A *coordination* module is in charge of choosing the final set-point to be followed. Two main coordination methodologies can be found. In *competitive* coordinators a single behavior is selected whereas in *cooperative* coordinators several behavior responses are superposed.

In the implementation of a behavior-based system, the design and tune-up of the behaviors is a hard task and requires a lot of experimentation. In these systems, there is also the need of performing in unknown and time-varying environments, which means that some kind of adaptation is needed. To solve these difficulties, many robotic systems include learning techniques. There is not yet an established methodology to develop adaptive behavior-based systems. However, a commonly used approach is *Reinforcement Learning* (Sutton and Barto, 1998), a class of learning algorithm in which 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 the optimal policy, which maps the state of the environment to an action that will maximize the accumulated future rewards. Most RL techniques are based on *Finite Markov Decision Processes* (FMDP) causing that state and action spaces are finite. The main advantage of RL is that it does not use any knowledge database, as in most forms of machine learning, making this class of

learning suitable for online learning. The main disadvantages are the large convergence time and lack of generalization among continuous variables. The latter one is one of the most active research topics in RL.

Many RL-based systems have been applied to robotics during last years. Most of them use classic RL algorithms combined with some methodologies that reduce the generalization problem. In (Smart and Kaebbling, 2000), an instance-based learning algorithm was applied to a real robot in a corridor-following task. Also for the same task, in (Hernandez and Mahadevan, 2000) a hierarchical memory-based RL was proposed. A very common approach is the use of Q-learning (Watkins and Dayan, 1992) 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 (Takahashi et al., 1999) a memory-based implementation was proposed for vision-guided behavior acquisition, and (Takeda et al., 2001) shows a state/action quantification by a set of triangular patches. Also, a lot of proposals combine Q-learning and Neural Networks (NN), see (Gaskett et al., 1999) 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 (Bagnel and Schneider, 2001), a practical application to control an autonomous helicopter was presented.

This paper presents a behavior-based scheme for high-level control of autonomous robots. Such scheme has been conceived to accomplish the *tasks* given by a higher level deliberative layer. Each task, i.e.: “target following”, “go to a position”, “environment exploration”, is a simple sub-mission of the whole mission. The control scheme is structured as a set of behaviors and a hybrid coordinator between competitive and cooperative methodologies. This hybrid approach keeps the robustness and modularity of competitive approaches as well as the optimized paths of cooperative ones. As a second important feature, behaviors are learnt with a *Neural-Q-learning* (NQL) approach. Our approach differentiates from other NQL proposals in that we implement directly the Q-function into a NN, instead of decomposing the problem into a finite set of actions, features or clusters. This NQL implementation, known as *direct Q-learning* (Baird, 1995), is the most simple and straight way to generalize with a NN. This implies more learning capabilities, causing instability in the learning of the optimal state/action mapping. To avoid this problem, a database of the most recent learning samples taken from the whole state/action space is repeatedly used in the NN weight update phase.

To test the feasibility of the Neural-Q-learning algorithm, real experiments were carried out with the Autonomous Underwater Vehicle (AUV) ODIN

(Choi et al., 1995), see figure 1. The experiments consisted in learning a “target following” behavior with the vehicle in a pool environment. Real results showed the convergence of the NQL behaviors into an optimal policy, and therefore, the achievement of the behavior.

The structure of this paper is as follows. Section 2 describes the behavior-based control scheme. Section 3 introduces the Neural-Q-learning algorithm used for behavior learning. In section 4, the target following task using ODIN’s AUV is detailed. In section 5, real results and some discussion is given. And finally, conclusions and future work are presented in section 6.



Fig.1. ODIN executing the target following behavior.

2. BEHAVIOR-BASED CONTROL SCHEME

A set of simple behaviors and a coordinator constitute the behavior-based control scheme. 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. An *hybrid coordinator* (Carreras et al., 2001) 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 how important it is for the behavior to take control of 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. The nodes have two inputs and generate also 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 one is completely activated ($a_i=1$). However, when the dominant behavior is 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 slightly modify the responses of dominant behaviors 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.

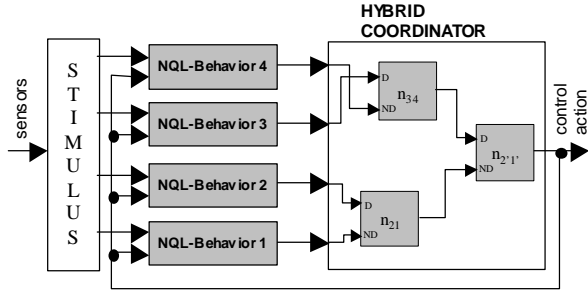


Fig. 2. Behavior-based architecture with the hybrid coordination system.

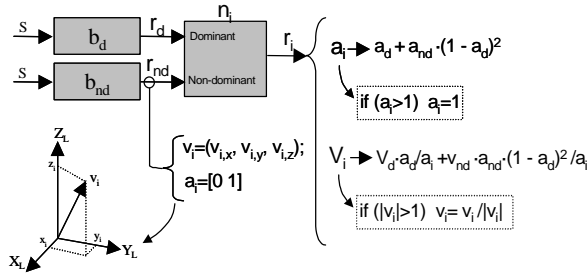


Fig. 3. Normalized output of a behavior and equations of the hierarchical hybrid node.

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 needed by the behavior in order to accomplish its goal. And the action space is the velocity set-points that the robot should follow.

3.1 Q_learning

Q-learning (Watkins and Dayan, 1992) is a temporal difference algorithm, see (Sutton and Barto, 1998), 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 phase like “on-site manual tuning” or “data collection + offline learning”. Another

important characteristic about 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]. Concerning the maximization of future rewards. If $\gamma=0$, the agent maximizes only immediate rewards.
- α : learning rate [0 1].
- \hat{I} : random action probability [0 1]. Exploitation versus exploration. The final action is called \hat{I} -greedy action.

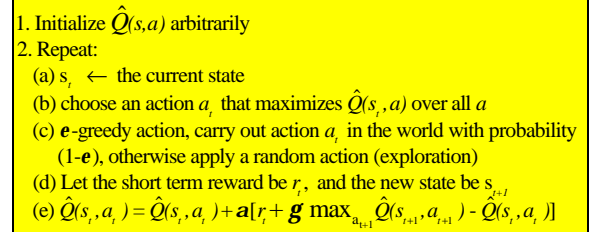


Fig. 4. Q_learning algorithm.

3.2 Neural Q_learning

When working with continuous states and actions, as it is usual in robotics, the Q-function table becomes huge for the required state/action resolution. In these cases, tabular Q-learning has a considerably large learning time and memory requirements, which makes impractical the implementation of the algorithm in a real-time control architecture. The use of a Neural Network (NN) to generalize among states and actions decreases 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 (Haykin, 1999) is known as *direct Q_learning* (Baird, 1995).

Direct Q-learning algorithm has no convergence proofs and demonstrated to be unstable when trying to learn a behavior. The instability was caused by the lack of weight updating in the whole state/action space. 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 by 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 showed in figure 5. The Q -function approximated by the NN is:

$$\hat{Q}(s_t, a_t) = r_t + \mathbf{g} \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) \quad (1)$$

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 a lineal activation functions for the first and second layers respectively. Weights are initialized randomly. To find the action that maximizes the Q -value, the network evaluates all the possible actions that could be applied. Although actions are continuous, a finite set, which guarantees enough resolution, is used.

3.3 Reinforcement function

The reinforcement function determines the policy learnt by the behavior. The definition of this function requires the knowledge of 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. TARGET FOLLOWING TASK

To test the feasibility of the NQL based behaviors, a target following behavior with ODIN’s AUV was carried out. ODIN (Choi et al., 1995) is the testbed AUV developed at the Autonomous Systems Laboratory of the University of Hawaii. ODIN is a sphere shaped vehicle with eight thrusters (4 horizontal and 4 vertical). It is capable of maneuvering with six degrees-of-freedom (DOF). ODIN has various sensors such as 8 sonar transducers, a pressure sensor, and an inertial navigation system. It also has an on-board CPU with VxWorks OS in VMEbus.

The task consisted in following a target by means of a colour camera attached to the vehicle towards X-axis, see figure 6. This application was designed to be carried out in a diving pool where light absorption

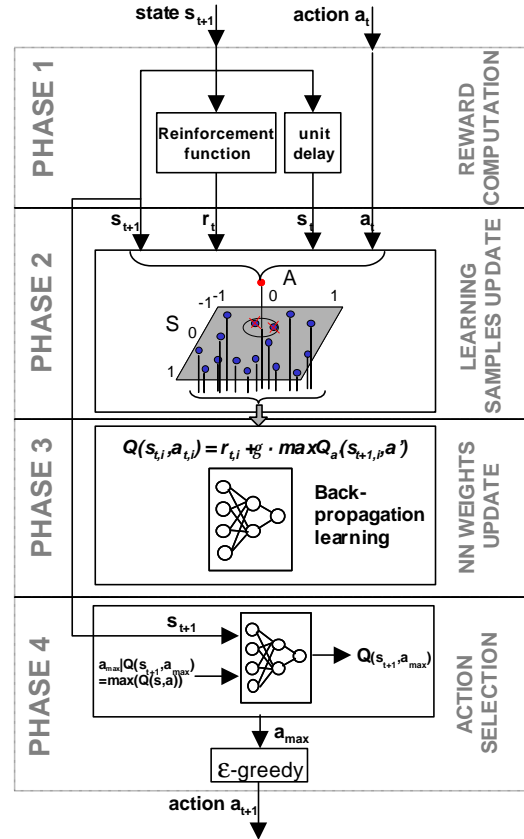


Fig. 5. Neural Q-learning algorithm structure.

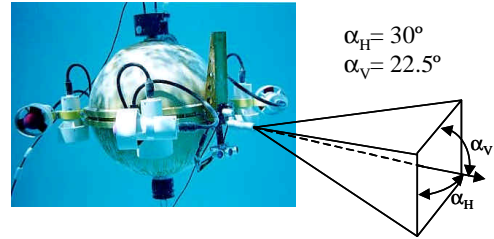


Fig. 6. Colour video camera layout

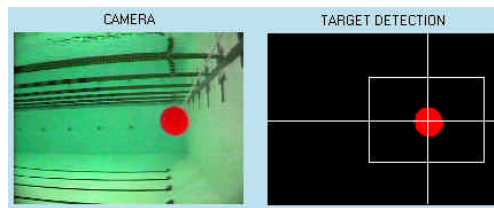


Fig. 7. Real and segmented images used by the target following behavior.

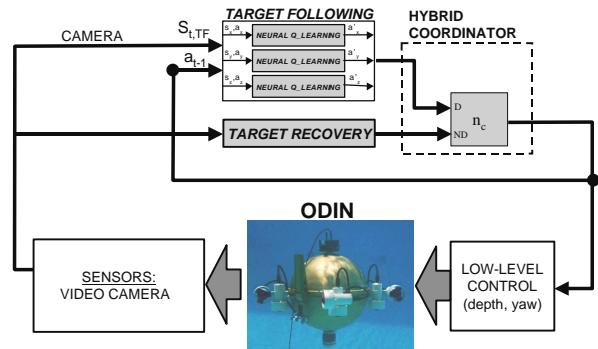


Fig. 8. ODIN's target following task control system.

does not apply, therefore a simple segmentation algorithm was used to detect the position of a tinted target, see figure 7. The two-dimensional position of the target in the image and its area were used as the input state of the NQL behavior. Each DOF was learnt with an independent NQL algorithm. Aside from the position of the target, the euler derivative of it was used also as input. A reinforcement function was designed to give positive rewards ($r=1$) when the target was around the vehicle's relative position $x=2$, $y=0$ and $z=0$. Otherwise, values like $r=0$ or $r=-1$ were given. The general control scheme for the target following behavior can be seen in figure 8. As the figure shows, an additional behavior, the "target recovery", was used. This behavior moved the vehicle in the direction in which the target was last seen when this disappeared from the image.

5. RESULTS

The described "target following" task was carried out with ODIN's AUV in a diving pool. Several phases were accomplished in order to test the communication interface, the onboard sensors, the low-level controllers, the computer vision algorithm and the NQL algorithm. An external computer was in charge of processing the video images, as well as executing the behavior-based layer. As stated above, the experiments were designed to test the feasibility of the NQL-based behaviors. The target following behavior was implemented with 3 different NQL algorithms (one for each DOF). In these preliminary experiments, only one NQL algorithm was learnt at each time, no simultaneous learning between different DOF was tested.

During initial experimentation phases some problems were found. One of these problems was the choice of the sample time of the behavior-based layer. A small sample time was required to ensure the control performance. However in order to learn properly with the NQL algorithm, a bigger period was needed to ensure an identifiable state variation compared with the inherent noise of the data. Finally, we adopted an intermediate value of 0.8 seconds. Another problem was related to the NN generalization approach. We tried to update the weights using only the current learning sample, but the NN was not able to maintain the Q-value for all the visited points. For this reason, we added the database in which the newest and most indicative learning samples of all the visited space were stored. This drastically solved the instability problem we had before.

Despite of these problems, we successfully were able to learn the NQL behavior. The robot was able to learn how to move in the 3 DOFs achieving the target following behavior. The internal parameters and configuration we used in this behavior can be seen in table 1. Also, the learnt state/action mappings for each DOF is shown.

The NQL behavior showed a very good robustness and small convergence time. Figure 9 shows a typical online learning evolution of the X DOF in which the behavior was learnt in less than 350 iterations (280 seconds). During this learning phase, aleatory actions caused the exploration of the space until the optimal policy was learnt and the algorithm found the maximum reward. Finally, once the 3 DOF were learnt, the target following behavior was tested. Figure 10, 11 and 12 show the performance of the X, Y and Z DOF respectively. In these figures it can be seen how after the suppression of the target following behavior by a manual behavior, driving away the vehicle from the target, the target following behavior took the control and reached the target again.

6. CONCLUSIONS

This paper has proposed a behavior-based scheme for high-level control of autonomous robots. The scheme is compound by a hybrid coordination system and several Reinforcement Learning-based behaviors. In particular, a Neural-Q_learning approach has been proposed and evaluated in a target following behavior with ODIN's AUV. Real results showed the feasibility of the algorithm demonstrating its convergence, robustness and efficiency. In the presented work, each DOF was learnt independently. Future work will concentrate on simultaneously learning different DOFs and behaviors.

REFERENCES

- Arkin, R. Behavior-based Robotics. MIT Press, 1998.
- Bagnell, J.A. and Schneider, J.G., Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods, *IEEE International Conference on Robotics & Automation*, Korea, 2001.
- Baird, K. Residual Algorithms: Reinforcement Learning with Function Approximation. *Machine Learning: Twelfth International Conference*, San Francisco, USA, 1995.
- Carreras, M., Batlle, J. and Ridao, P. Hybrid Coordination of Reinforcement Learning-based Behaviors for AUV control, In: *IEEE/RSJ IROS*, Hawaii, USA 2001.
- Choi, S.K., Yuh, J. and Takashige, G.Y.. Development of the Omni-Directional Intelligent Navigator. *IEEE Robotics and Automation Magazine*, pp. 44-53, 1995.
- Gaskett, C., Wettergreen, D. and Zelinsky, A. Q-learning in continuous state and action spaces. *Proc. of the 12th Australian Joint Conference on Artificial Intelligence*, Sydney, Australia, 1999.
- Haykin, S. Neural Networks, a comprehensive foundation. Prentice Hall, 2nd ed., 1999.
- Hernandez, N. and Mahadevan, S., Hierarchical Memory-Based Reinforcement Learning, *15th Intern. Conference on Neural Information Processing Systems*, Denver, USA, 2000.

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

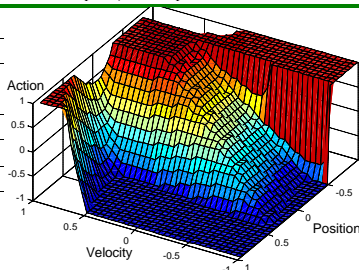
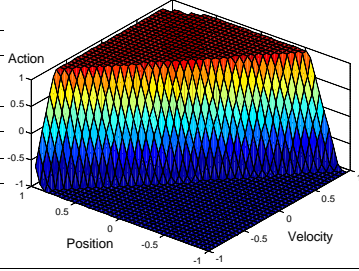
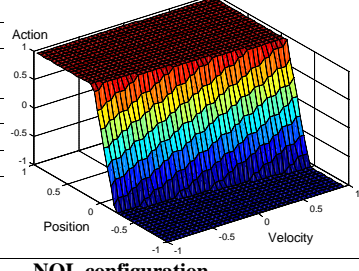
Sutton, R. and Barto, A. *Reinforcement Learning, an introduction*. MIT Press, 1998.

Takahashi, Y., Takada, M. and Asada, M. Continuous Valued Q-learning for Vision-Guided Behavior Acquisition. In *International Conference on Multisense Fusion and Integration for Intelligent Systems*, pages 716-721, 1999.

Takeda, M., Nakamura, T. and Ogasawara, T., Continuous Valued Q-learning Method Able to Incrementally Refine State Space, *IEEE/RSJ IROS*, Hawaii, USA 2001.

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

Table 1. Target following behavior specifications

Sensors	color camera + target detection algorithm using color segmentation	
Codification	[f_x, f_y, f_z] : target position + normalization [-1, 1]	
Activation	if target detected: $a_i = 1$; else $a_i = 0$	
X DOF		
state		
f_x :	position	
fv_x :	euler derivative	
action		
a_{fx} :	desired speed	
reinforcement function		
If $f_x < 0.15$: $r_{fx} = 1$		
else if $f_x < 0.4$: $r_{fx} = 0$		
else $r_{fx} = -1$		
NQL parameters		
$\alpha = 0.05; \gamma = 0.9; \epsilon = 0.4$		
Database size = up to 50 learning samples		
NQL configuration		
inputs: 3 : [f_x, fv_x, a_{fx}]		
output: Q_value		
layer 1: 6 neurons (hyperbolic tangent a.f.)		
layer 2: 1 neuron (lineal a.f.)		
		
Y DOF		
state		
f_y :	position	
fv_y :	euler derivative	
action		
a_{fy} :	desired speed	
reinforcement function		
If $f_y < 0.2$: $r_{fy} = 1$		
else if $f_y < 0.5$: $r_{fy} = 0$		
else $r_{fy} = -1$		
NQL parameters		
$\alpha = 0.05; \gamma = 0.9; \epsilon = 0.4$		
Database size = up to 50 learning samples		
NQL configuration		
inputs: 3 : [f_y, fv_y, a_{fy}]		
output: Q_value		
layer 1: 3 neurons (hyperbolic tangent a.f.)		
layer 2: 1 neuron (lineal a.f.)		
		
Z DOF		
state		
f_z :	position	
fv_z :	euler derivative	
action		
a_{fz} :	desired speed	
reinforcement function		
If $f_z < 0.2$: $r_{fz} = 1$		
else if $f_z < 0.4$: $r_{fz} = 0$		
else $r_{fz} = -1$		
NQL parameters		
$\alpha = 0.05; \gamma = 0.9; \epsilon = 0.4$		
Database size = up to 50 learning samples		
NQL configuration		
inputs: 3 : [f_z, fv_z, a_{fz}]		
output: Q_value		
layer 1: 3 neurons (hyperbolic tangent a.f.)		
layer 2: 1 neuron (lineal a.f.)		
		

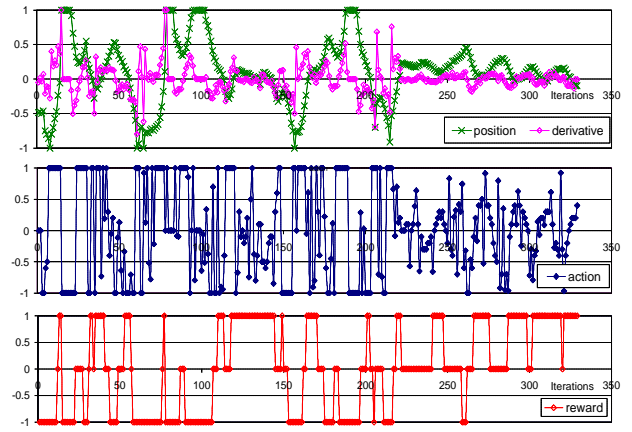


Fig. 9. Online learning evolution of the target following behavior in the X DOF. The states, actions and rewards are shown.

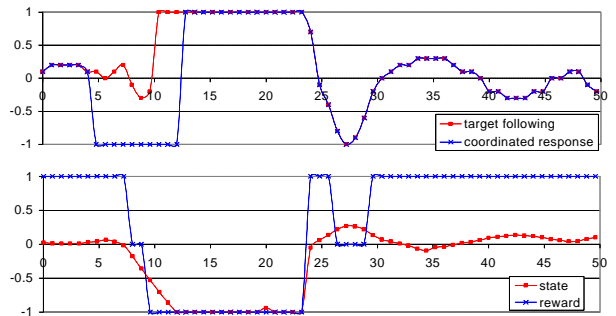


Fig. 10. Performance of the state target following behavior in the X DOF. Once the behavior took ODIN's control, the target was reached.

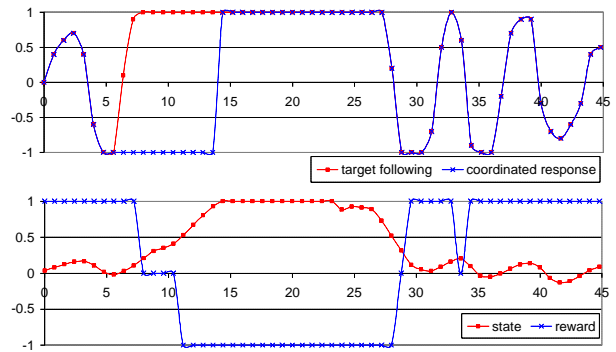


Fig. 11. Performance of the target following behavior in the Y DOF. Once the behavior took ODIN's control, the target was reached.

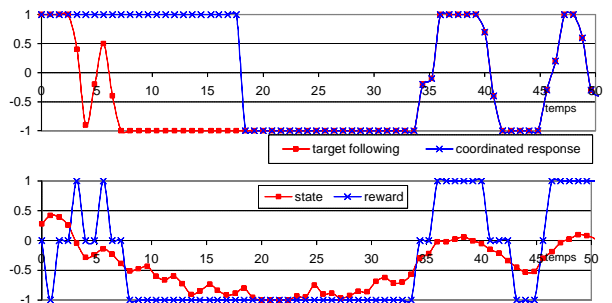


Fig. 12. Performance of the target following behavior in the Z DOF. Once the behavior took ODIN's control, the target was reached.