Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada · October 2003

Semi-Online Neural-Q_learning for Real-time Robot Learning

M. Carreras, P. Ridao and A. El-Fakdi
Institute of Informatics and Applications
University of Girona
Campus Montilivi, Girona 17071, Spain
marcc@eia.udg.es

Abstract— **Reinforcement Learning (RL) is a very suitable technique for robot learning, as it can learn in unknown environments and in real-time computation. The main difficulties in adapting classic RL algorithms to robotic systems are the generalization problem and the correct observation of the Markovian state. This paper attempts to solve the generalization problem by proposing the Semi-Online Neural-Q_learning algorithm (SONQL). The algorithm uses the classic Q_learning technique with two modifications. First, a Neural Network (NN) approximates the Q_function allowing the use of continuous states and actions. Second, a database of the most representative learning samples accelerates and stabilizes the convergence. The term semi-online is referred to the fact that the algorithm uses the current but also past learning samples. However, the algorithm is able to learn in real-time while the robot is interacting with the environment. The paper shows simulated results with the "mountain-car" benchmark and, also, real results with an underwater robot in a target following behavior.**

I. INTRODUCTION

A commonly used methodology in robot learning is Reinforcement Learning (RL) [8]. 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 that 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 drawbacks are the lack of *generalization* among continuous variables and the difficulties in observing the Markovian state in the environment. A very used RL algorithm is the Q_learning [12] algorithm due to its good learning capabilities: online and off-policy.

Many RL-based systems have been applied to robotics over the past few years and most of them have attempted to solve the generalization problem. To accomplish this, classic RL algorithms have been usually combined with other methodologies. The most commonly used methodologies are decision trees [11], CMAC function approximator [6], memory-based methods [7] and Neural Networks (NN) [3]. These techniques modify the RL algorithms breaking in many cases their convergence proofs. Only some algorithms which use a linear function approximator have maintained these proofs [9]. Neural Networks is a non-linear method, however, it offers a high generalization capability and demonstrated its feasibility in very complex tasks [10]. The drawback of Neural Networks is the *interference* problem. This problem is caused by the impossibility of generalizing in only a local zone of the entire space. Interference occurs when learning in one area of the input space causes unlearning in another area [13].

This paper proposes the Semi-Online Neural-Q_learning algorithm (SONQL). This algorithm attempts to solve the generalization problem combining the Q_learning algorithm with a NN function approximator. This approach implements the Q-function directly into a NN. This NQL implementation, known as direct Q_learning [1], is the simplest and straightest way to generalize with a NN. In order to solve the interference 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 and, also, accelerating the learning process. The algorithm was designed to work in real systems with continuous variables. To preserve the real time execution, two different execution threads, one for learning and another for output generation, are used.

The SONQL algorithm was conceived to learn the internal state/action mapping of a reactive robot behavior. This paper demonstrates its feasibility with real experiments using the underwater robot URIS in a target following task. Results demonstrate the feasibility of the algorithm in a real-time system. The paper also demonstrates the convergence of the algorithm in a well-known generalization benchmark. Simulation results of the "Mountain-Car" task [5] are presented.

The structure of this paper is as follows. In Section II, the SONQL algorithm is detailed. The experimental results are included in Section III. And the conclusions and future work are presented in Section IV.

0-7803-7860-1/03/$17.00 © 2003 IEEE

II. SEMI-ONLINE NEURAL-Q_LEARNING

Next subsections describe the basic features of the Semi-Online Neural-Q_learning algorithm (SONQL). First of all, the original Q_learning technique is introduced. Then, the modifications found in the SONQL are presented. And finally, the phases of the algorithm and some implementation aspects are detailed.

A. Q_learning

Q_learning [12] is a temporal difference algorithm, see [8], 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 real-time robot learning. The importance of real-time learning resides in the possibility of executing new behaviors without any previous phase 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. This is a very important feature in a robotic domain, since sometimes, the actions that are proposed by the learning algorithm can not be carried out. For example, if the algorithm proposes an action that would cause a collision, another behavior with more priority will prevent it, and the learning algorithm will use the real executed action.

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 1 shows a diagram of the Q_learning algorithm. The parameters of the algorithm are the discount factor γ, the learning rate α and the ϵ parameter for the random actions.

B. Generalization with Neural Networks

When working with continuous states and actions, as it is usual in robotics, the Q-function table becomes too large for the required state/action resolution. In these cases, tabular Q_learning requires a very long learning time, making the implementation of the algorithm in a real-time control architecture impractical. This problem is known as the generalization problem.

The use of a Neural Network to generalize among states and actions reduces the number of values stored in the Q-function table to a set of NN weights. The approximation of the Q-function using a feed-forward NN with the back-propagation algorithm [4] is known as direct Q_learning

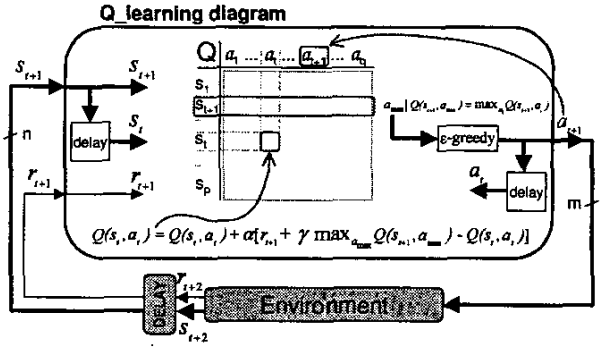

Fig. 1. Diagram of the Q_learning algorithm.

[1]. In this implementation, the NN has as inputs the state and the action, and as output, the one-dimensional Q value. The function that must be approximated is the one shown in Equation 1. Other implementations of the Q-function with a NN can also be found, although this is the most straightforward one.

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

The direct Q_learning approach has been taken to solve the generalization problem. In particular, the NN configuration is composed by one or two hidden layers containing a set of neurons, and the output layer, which has one neuron. The number of layers and neurons, depends on the complexity and the number of dimensions of the Q_function to be approximated. For the hidden layers, an hyperbolic tangent function is used as the activation function. This function is antisymmetric and accelerates the learning process. The output layer has a lineal activation function, which allows to the NN to approximate any real value. The initialization of the NN weights is done randomly.

C. Semi-Online Learning

Neural Networks have a high generalization capability, however they suffer from the *interference problem* [13]. Interference occurs when learning in one zone of the input space causes loss of learning in other zones. To solve the interference problem, the SONQL uses a *database* of *learning samples*. The main goal of the database is to include a representative set of visited learning samples, which is repeatedly used to update the NQL algorithm. The immediate advantage of the database, is the stability of the learning process and its convergence even in difficult problems. Due to the representative set of learning samples, the Q_function is regularly updated with samples of the whole visited state/action space, which is one of the conditions of the original Q_learning algorithm. A consequence of the database is the acceleration of the

learning. This second advantage is most important when using the algorithm in a real system. The updating of the NQL is done with all the samples of the database and, therefore, the convergence is achieved with less iterations. The term *semi-online* is therefore referred to the fact that the current and also past samples are used in the learning.

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+1}. During the learning evolution, the learning samples are added to the database. Each new sample replaces older samples previously introduced. The replacement is based on the geometrical distance between vectors (s_t, a_t, r_{t+1}) of the new and old samples. If this distance is less than a *density parameter* t for any old sample, the sample is removed from the database. The size of the database is, therefore, controlled by this parameter which has to be set by the designer. Once the algorithm has explored the reachable state/action space, a homogeneous, and therefore, representative set of learning sample is contained in the database.

D. Algorithm phases

The proposed Semi-Online Neural-Q_learning algorithm can be divided in four different phases, as shown in Figure 2. In the first phase, the current learning sample is assembled. The state s_{t+1} and the last taken action a_t are received from the environment. The reward r_t is computed according to s_{t+1}, and, the past state s_t is extracted from a unit delay. In the second phase, the database is updated with the new learning sample. As has been commented on, old samples similar to the new one will be removed.

The third phase consists of updating the weights of the NN according to the back-propagation algorithm and Equation 1. If the SONQL algorithm is used in a real-time system, such as a robot, this phase is executed in a separate thread. This thread has a lower priority and uses all the non-used computational power to learn the optimal Q_function. Using these two execution threads, the real-time execution of the control system can be accomplished.

The fourth phase consists of proposing a new action a_{t+1}. The policy that is followed is the $\varepsilon - greedy$ policy. With probability $(1-\varepsilon)$, the action will be the one that maximizes the Q_function in the current state s_{t+1}. Otherwise, an random action is generated. Due to the continuous action space in the Neural-Q_function, the maximization is accomplished by evaluating a finite set of actions. As this evaluation is very fast, the action space can be discretized with the necessary resolution without an important computational cost.

III. EXPERIMENTAL RESULTS

This section shows experimental results of the SONQL algorithm in two different domains. The first is the well-known "mountain-car" task, in which an extensive set of

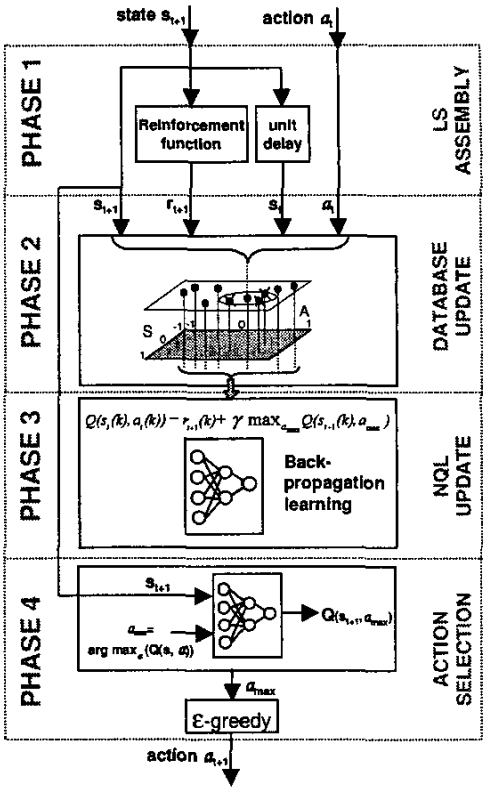

Fig. 2. Phases of the Semi-Online Neural-Q_learning algorithm.

simulations demonstrate the effectiveness of the algorithm. The second domain, is a real task with an underwater robot. The algorithm learns how to move the robot in order to follow a target.

A. The Mountain-Car Task

The "mountain-car" benchmark [5] was designed to evaluate the generalization capability of RL algorithms. In this problem, a car has to reach the top of a hill, see Figure 3. However, the car is not powerful enough to drive straight to the goal. Instead, it must first reverse up the opposite slope in order to accelerate, acquiring enough momentum to reach the goal. The states of the environment are two continuous variables, the position p and the velocity v of the car. The action a is the force of the car, which can be positive and negative. The reward is -1 everywhere except at the top of the hill, where it is 1. The dynamics of the system can be found in [5]. The episodes in the mountain-car task start in a random position and velocity, and they run for a maximum of 200 iterations or until the goal has been reached. The optimal state/action mapping is not trivial since depending on the position and the velocity, the action has to be positive or negative.

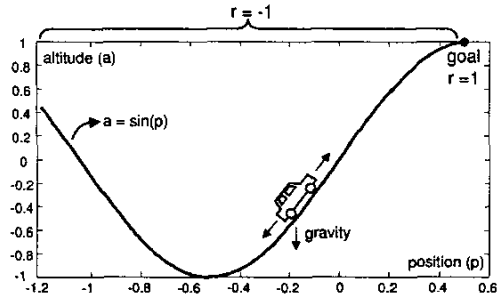

Fig. 3. The "mountain-car" task domain.

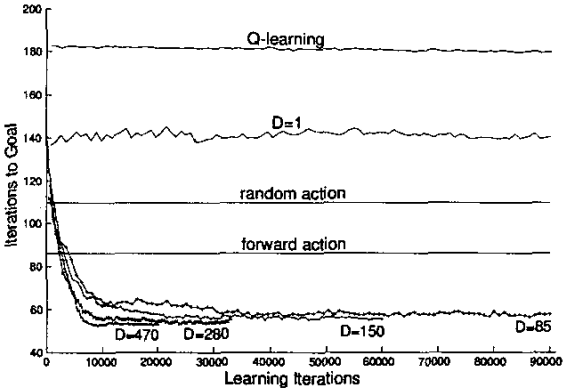

Fig. 4. Results of the SONQL algorithm with the Mountain Car task. Each learning iteration is a complete SONQL iteration. Different experiments, with different database sizes (D), are shown. The effectiveness of the Q_learning and the random and forward policies are also shown.

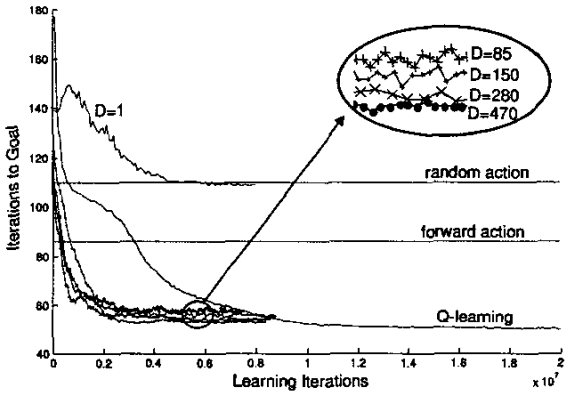

Fig. 5. Results of the SONQL algorithm with the Mountain Car task. For the SONQL graphs, each learning iteration is a NN update.

To provide a performance baseline, the classic Q_learning algorithm was applied. The state space was finely discretized, with 180 states for the position and 150 for the velocity. The action space contained only three values, -1, 0 and 1. Therefore, the size of the Q table was 81000 cells. The exploration strategy was an $\varepsilon - greedy$ policy with ε set at 30%. The discount factor was $\gamma = 0.95$ and the learning rate $\alpha = 0.5$, which were found experimentally. The Q table was randomly generated at the beginning of each experiment. In each experiment, a learning phase and an evaluation phase were repeatedly executed. In the learning phase, a certain number of iterations were executed, starting new episodes when it was necessary. In the evaluation phase, 500 episodes were executed. The *effectiveness* of learning was evaluated by looking the averaged number of iterations needed to finish the episode. After running 100 experiments with discrete Q_learning, the average number of iterations when the optimal policy had been learnt was 50 with 1.3 standard deviation. And the number of learning iterations to learn this optimal policy was $1x10^7$ learning iterations.

Since the state space had been finely discretized, it was assumed that with only three actions, the minimum number of iterations to fulfill the goal is 50. The SONQL algorithm cannot improve this mark, as it is based on the Q_learning algorithm. However, it is expected that it can reduce the number of iterations required to learn. It is interesting to compare this mark with other state/action policies. If a forward action ($a = 1$) is always applied, the average episode length is 86. If a random action is used, the average is 110. These averages depend highly on the fact that the maximum number of iterations in an episode is 200, since in a lot of episodes these policies do not fulfill the goal.

An extensive number of experiments were executed with the SONQL algorithm in order to find the best configuration. The NN had three layers with 15 neurons in the two hidden layers. Only three actions were used, as with the Q_learning experiments. The optimal learning rate and discount factor were $\alpha = 0.0001$ and $\gamma = 0.95$. And the ε parameter was set at 30%. The averaged number of learning samples at the end of the experiment, was approximately 470. As with the Q_learning algorithm, each experiment had a learning and a evaluation phases. In the evaluation phase 500 episodes were tested. For each experiment with a SONQL configuration, a total number of 100 trials were run. The average episode length for the parameters detailed above was 53 with 2.1 standard deviation. The number of learning iterations were only 20000 approximately. This result demonstrates that the SONQL is able to converge much faster than the Q_learning algorithm, although it is not able to learn the optimal policy (53 iterations instead of 50). The convergence of the algorithm has also proved to be very high, in all the experiments the optimal policy was learnt.

Figure 4 shows the effectiveness evolution respect the number of learning iterations. In the graph labelled

"D=470", after 20000 learning iterations the number of iterations required to accomplish the goal are 53. It is also observed how the SONQL maintains the effectiveness without diverging until the end of the experiments, at 20000 learning iterations. Figure 5 shows another representation of the same experiments. In this case, the graph "D=470" is drawn respect the total number of NN updates. Therefore, considering each sample of the database as a learning iteration. Evidently, the number of iterations is much higher than before, since for each SONQL iteration all the samples of the database are learnt. However, the number of iterations are also significantly less than with the Q_learning algorithm, which is also represented.

The influence of the database has been analyzed using the same SONQL configuration and changing the database size. Four sets of experiments with 280, 150, 85 and 1 samples were executed. The total number of NN updates was fixed to $8x10^6$ iterations approximately, as it can be seen in Figure 5. Consequently, having an smaller database the number of SONQL iterations increased, see Figure 4. Two important conclusions can be extracted. First, the database is necessary to ensure the convergence. In the graph "D=1" only the current sample was used and, even for the same number of NN updates, the algorithm did not converge. Second, with a larger database a better learning is achieved. The averages of the "D=470", "D=280", "D=150" and "D=85" experiments are 53, 54, 56 and 58 respectively. A large database implies a large set of learning samples uniformly distributed in the visited space. This representative set of samples improves the learning capability of the direct Q_learning. These results empirically demonstrate the benefit of using the learning samples database.

B. Target Following Behavior

The use of the SONQL algorithm in a real system was also tested. In particular, the task consisted in following an artificial target with the Autonomous Underwater Vehicle URIS. This small underwater robot was designed to carry out experiments in a water tank. The robot has four propellers which allow the movements in X, Z and Yaw Degrees Of Freedom (DOF). However, due to the shallowness of the water tank, the robot can only move in X and Yaw. The position and velocity of the robot is estimated by a vision system. And the position of the artificial target respect the robot is detected by an onboard forward looking camera. A picture of URIS and its onboard coordinate frame can be seen in Figure 6. For a complete description of the vehicle and the positioning system refer to [2].

The target following behavior was implemented with 2 different SONQL algorithms (one for each DOF, X and Yaw). In these experiments, only one SONQL algorithm was learnt at a time, no simultaneous learning between different DOFs was tested. Each DOF received information about the current state and the last taken action. The state was the position of the target in the image as well as its velocity. A reinforcement function gave different rewards (-1, 0 or 1) depending on the position at which the target was detected.

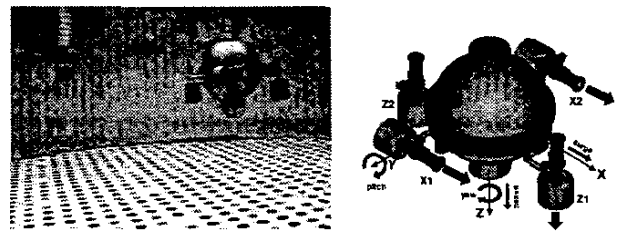

Fig. 6. URIS' AUV. a) Picture of the robot in the water tank while it was following the artificial target. b) Schema of the robot.

Several trials were carried out in order to find the best learning performance. The sample time of the SONQL algorithm was set at 0.3 [s]. The NN had two layers with 6 neurons in the hidden layer. 21 different actions were used for each DOF. The learning rate was $\alpha = 0.1$, the discount factor $\gamma = 0.9$, and the ε parameter was set at 30%. The number of learning samples was approximately 50, having an independent execution thread to update the NN (phase 3). The robot was able to learn in real-time how to move in both DOFs achieving the target following task. The main difficulty in achieving a robust learning was the correct estimation of the environment state (the target position and velocity). The learnt state/action optimal mappings can be seen in figures Figure 7 and Figure 8. Note the obtained non-lineal mappings.

The SONQL algorithm showed a good robustness and presented a small convergence time (less than 2 minutes). Figure 9 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.

IV. CONCLUSIONS

This paper has proposed the Semi-Online Neural-Q_learning algorithm (SONQL), which attempts to solve the generalization problem with a Neural Network an a learning samples database. The approach has been detailed an tested in two different domains. In the "mountain-car" task, it demonstrated to almost converge to the optimal

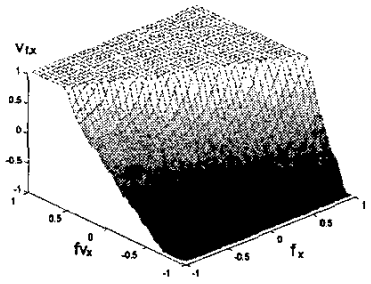

Fig. 7. State/action mapping learnt for the target following behavior in the X DOF. The state is the target position f_x and velocity fv_x with respect to X axis. The action $v_{f,x}$ is the robot velocity in the X DOF.

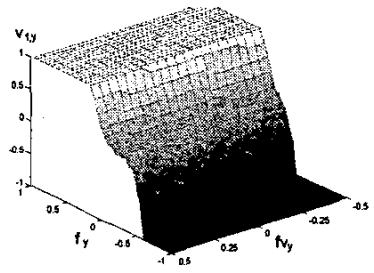

Fig. 8. State/action mapping learnt for the target following behavior in the Yaw DOF. The state is the target position f_y and velocity fv_y with respect to Y axis. The action $v_{f,y}$ is the robot velocity in the Yaw DOF.

policy. The benefit of the approach was the substantial reduction of the iterations required to converge, compared to the classic Q_learning algorithm. It has been demonstrated the important role of the database to ensure the convergence and performance. The second test domain has been a real robotics task. In this case, the fast convergence of the algorithm was used to learn a target following behavior. The algorithm was able to learn in real time the optimal state/action policy. The most important drawback in the robotics task was the correct observation of the state. Future work will concentrate on learning in Non-Markovian environments.

V. ACKNOWLEDGMENTS

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

VI. REFERENCES

[1] K. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning: Twelfth International Conference*, San Francisco, USA, 1995.

[2] M. Carreras, P. Ridao, R. Garcia, and T. Nicosevici. Vision-based localization of an underwater robot in a structured environment. In *IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[3] C. Gaskett. *Q_learning for Robot Control*. PhD thesis, Australian National University, 2002.

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

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

[6] 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.

[7] W.D. Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science at Brown University, Rhode Island, May 2002.

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

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

[10] G.J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, 1992.

[11] W.T.B. Uther and M.M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

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

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

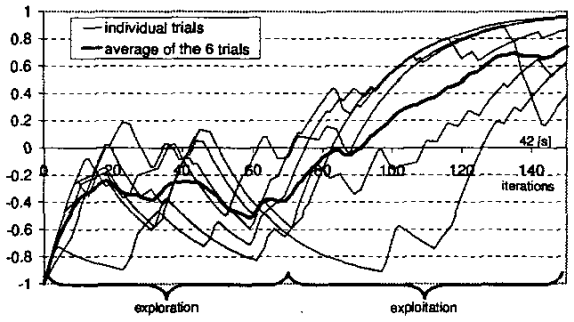

Fig. 9. Behavior convergence. Results of six different learning trials of the target following behavior in the Yaw DOF. The averaged rewards over the last 20 iterations are shown. The average of the six experiments can also be seen. Accumulated rewards increased in the measure that the behavior was learnt.