A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics
May 23 – 25, 2002, Cluj-Napoca, Romania

Learning reactive robot behaviors with Neural-Q_leaning

M. Carreras, P. Ridao, J. Batlle, T. Nicosebici and Z. Ursulovici
Institute of Informatics and Applications, University of Girona
Edifici Politècnica IV, Campus Montilivi, 17071 Girona, Spain
{ marcc,pere,jbatlle,tudor,zoranu }@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 autonomous underwater vehicle (AUV) in a target following mission. Simulated experiments demonstrate the convergence and stability of the learning system, pointing out its suitability for online robot learning. Advantages and limitations are discussed.

Keywords: Robot Learning, Reinforcement Learning, Behavior-based Robotics, Autonomous Underwater Vehicles.

1 INTRODUCTION

A commonly used methodology in robot learning is Reinforcement Learning (RL) [6]. 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.

This paper proposes a *Neural-Q_learning* (NQL) [4] approach, a kind of RL algorithm, 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* [2], 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. 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 missions with an autonomous robot. Behavior coordination is done through a hybrid coordinator [3] which does not need any tuning phase. This paper demonstrates the feasibility of the proposed NQL algorithm with simulated experiments using the underwater robot URIS, see figure 1. A three-dimensional target following mission with two behaviors, target following and obstacle avoidance, were learnt using NQL. The results obtained show the convergence of the NQL-based behaviors into an optimal policy, and therefore, the achievement of the mission. The structure of this paper is as follows. In section 2, an overall description of the behavior-based control scheme is presented. Section 3 introduces the proposed Neural-Q_learning approach for behavior learning. In section 4, the simulated experiments with URIS's AUV are detailed and discussed. And finally, conclusions and future work are presented in section 5.

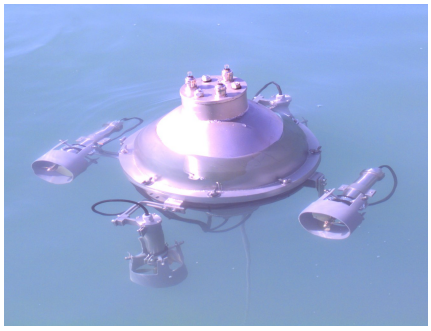

Figure 1. URIS's AUV.

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 behavior action 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. An *hybrid coordinator* [3] 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.

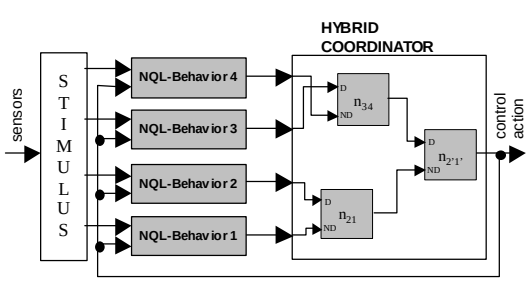

Figure 2: Behavior-based architecture with the hybrid coordination system.

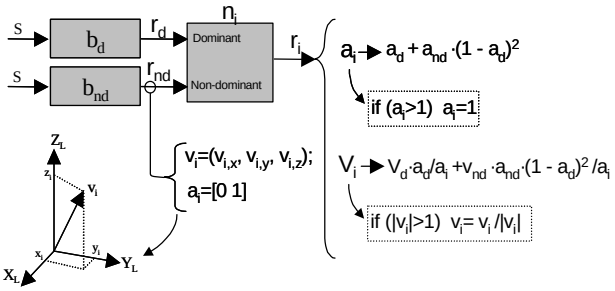

Figure 3: Normalized output of a behavior and equations of the hierarchical hybrid node.

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.

3.1 Q_learning

Q-learning [7] is a temporal difference algorithm, see [6], 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. 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.

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

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 [5] is known as *direct Q_learning* [2]. The Direct Q-learning algorithm has no convergence proofs and has proved 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 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 3. 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 $\hat{Q}(s_t, a_t) = r_t + g \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$. 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.

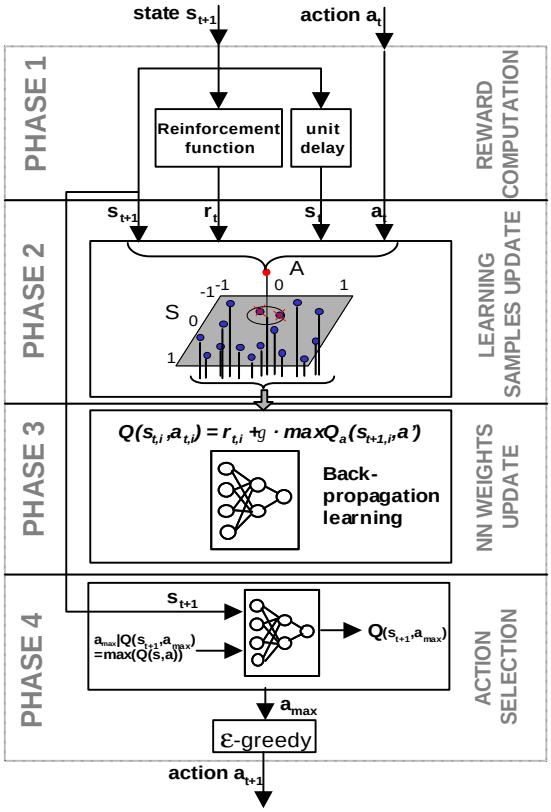

Figure 5: Neural Q_learning algorithm

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 EXPERIMENTATION WITH AN AUV

The kind of robot which we are working on is an Autonomous Underwater Vehicle (AUV) called URIS. The proposed application consists of following a target by means of a camera and avoiding obstacles using a set of sonar sensors. This application was designed to be carried out in a swimming pool where light absorption does not apply. URIS, see figure 1, is a small-sized non-holonomic AUV designed and built at the University of Girona. To accomplish this mission a Behavior-based architecture with three behaviors was designed. Each behavior has its own input from sensors and generates a 3D-speed vector defined by (*u*, *w*, *r*). Figure 6 shows the schema of the architecture. The three behaviors are:

- *Obstacle avoidance.* The goal is to avoid any obstacles perceived by means of 7 sonar sensors, see figure 7. The behavior is learnt using a NQL algorithm for each DOF (x,y,z). A reinforcement function gives negative rewards depending on the distance at which obstacles are detected.
- *Target following*. The behavior follows the target using a video camera pointed towards X-axis, see figure 7. A real-time tracking board based on chromatic characteristics gives the relative position of the target. The behavior is learnt using a NQL algorithm for each DOF (x,y,z). The reinforcement function gives negative rewards when the target moves away from the position X=5, Y=0 and Z=0.
- *Target recovery. The goal of this behavior is to recover* the target when it disappears from the camera view. When the tracking system loses the target, the behavior spins and moves the vehicle vertically in the direction last seen. This behavior is not learned but preprogrammed.

A simulated environment was developed in order to realistically experiment with the vehicle, see figure 8. The simulation contained the hydrodynamics model of the vehicle and the identified noise and delays of the sensors and actuators. A moving target was introduced carrying out a 3D closed path repeatedly. The velocity of the target changed between 0 and 0.17 m/s (60% of the maximum velocity of URIS). "Target following" and "Obstacle avoidance" behaviors were implemented using the neural Q_learning algorithm. Each degree of freedom was implemented independently with its inputs/outputs and rewards. At the beginning, each behavior was learnt alone, without the influence of the other. The number of iterations required to learn each DOF was approximately 150 (Sample time=1s). Figure 9 shows the evolution of the "x" DOF of the "target following" behavior during its training. It can be seen how the algorithm explores the action space and learns how to track the target. Once the 3

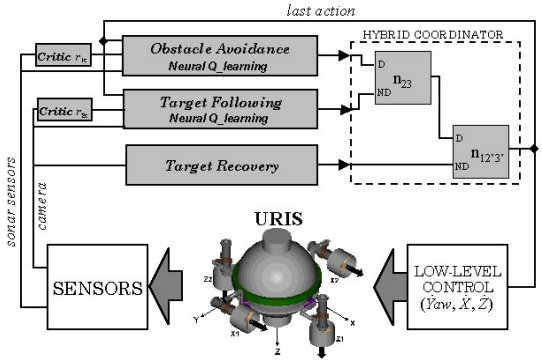

Figure 6. Schema of the architecture.

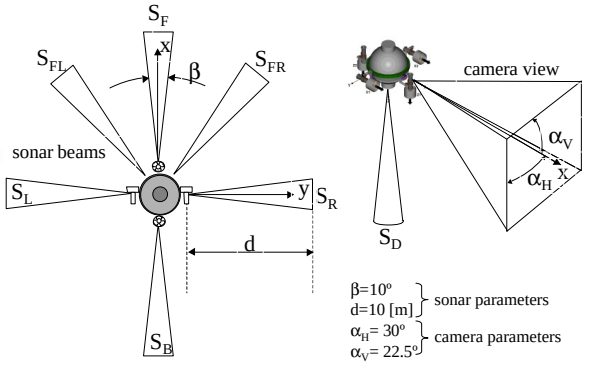

Figure 7. Sonar transducer and video camera layout.

DOFs of both behaviors were completely learnt, the mission was tested. Figure 10 shows the tracking error evolution during the mission. When the vehicle was close to an obstacle, the obstacle avoidance behavior took partial control of the vehicle, and therefore, the tracking error increased. However, the hybrid coordination system generated a cooperative response between both behaviors, and the target was not lost.

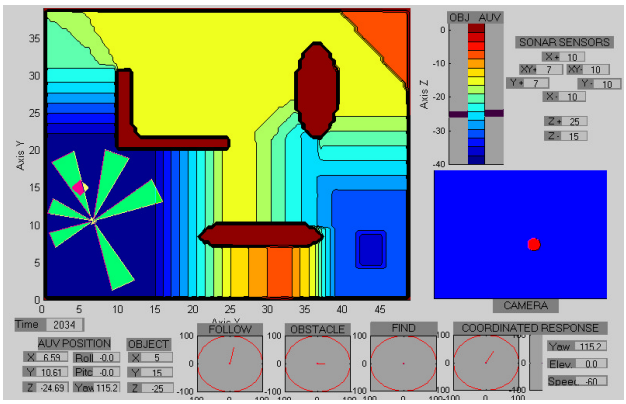

Figure 8, MMVVE running a simulation.

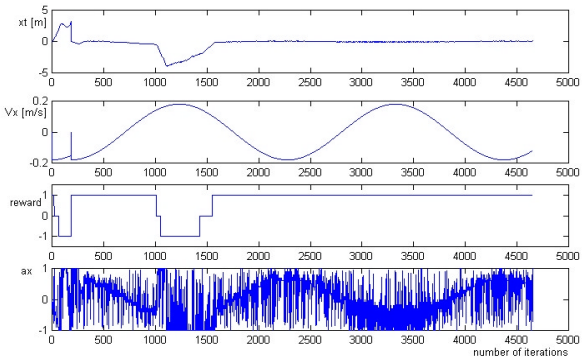

Figure 9. Learning evolution of the "target following" behavior (x axis).

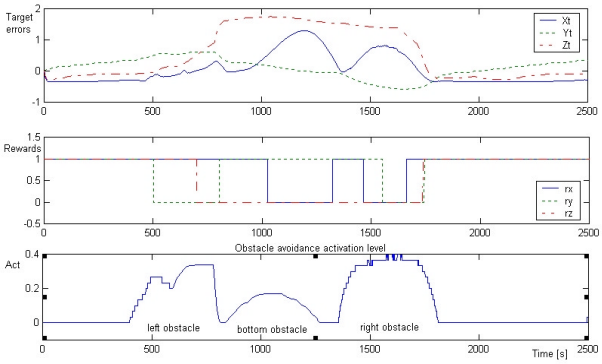

Figure 10. Tracking error evolution during a mission.

5 CONCLUSIONS AND FUTURE WORK

A proposal of Neural-Q_learning based behaviors have been presented. The simulated results showed the feasibility of the hybrid approach as well as the convergence of the learning algorithm. 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. In the presented work, each DOF was learnt independently. on the realization of real experiments and on the improvement of the RL-based behaviors in order to learn simultaneously all the behaviors.

REFERENCES

[1] R. Arkin, *Behavior-based Robotics*. MIT Press, 1998.
[2] K. Baird, 'Residual Algorithms: Reinforcement Learning with Function Approximation', *Machine Learning: Twelfth International Conference*, San Francisco, USA, 1995.
[3] M. Carreras, J. Batlle and P. Ridao, 'Hybrid Coordination of Reinforcement Learning-based Behaviors for AUV control', *IEEE/RSJ IROS*, Hawaii, USA 2001.
[4] C. Gaskett, D. Wettergreen and A. Zelinsky, 'Q-learning in continuous state and action spaces', *12th Australian Joint Conference on Artificial Intelligence*, Australia, 1999.
[5] S. Haykin, *Neural Networks, a comprehensive foundation*. Prentice Hall, 2nd ed., 1999.
[6] R. Sutton, and A. Barto, *Reinforcement Learning, an introduction*. MIT Press, 1998.
[7] C.J.C.H. Watkins and P. Dayan, 'Q-learning', *Machine Learning*, 8:279-292, 1992.