# Control of an Autonomous Robot using Multiple RL-based Behaviors

Andres El-Fakdi, Marc Carreras and Joan Batlle
Institute of Informatics and Applications
University of Girona
Campus Montilivi, Girona 17071, Spain
{aelfakdi,marcc,jbatlle}@eia.udg.es

## Abstract

This paper proposes a Behavior-based Control Architecture for solving the action selection problem of an autonomous robot. This architecture is characterized by the use of a new coordination system among the behaviors, and also by the use of Reinforcement Learning (RL) for learning the internal state/action mapping of some behaviors. The coordination system proposes a hybrid method between competitive and cooperative methodologies with the aim of taking the advantages of robustness and optimized robot trajectories from both approaches. This paper shows the feasibility of this hybrid coordinator with a real application of an Autonomous Underwater Vehicle (AUV) in a target following task. The second feature of the Behavior-based control architecture is the possibility of learning some of the behaviors. Reinforcement Learning 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 real results of the SONQL algorithm learning a target following behavior.

**Keywords** : Behavior-based Robotics, Behavior Coordination, Reinforcement Learning, Robot Learning

## 1 Introduction

Behavior-based Robotics [1] is a methodology for designing autonomous agents and robots. Since its appearance, in the middle of 1980s, a huge amount of robotic applications have used this methodology. Endless quantities of methods have been proposed to solve the common characteristics of a Behavior-based system: behavior expression, design, encoding and coordination. Behaviors are implemented as a control law using inputs and outputs. The basic structure consists of all behaviors taking inputs from the robot's sensors and sending outputs to the robot's actuators. Behavior coordination is the phase in which a coordinator module receives the responses of all the behaviors and generates a single output to be applied to the robots. If the output is the selection of a single behavior, the coordinator is classified as *competitive*. On the other hand, if the output is the superposition of several behavior responses, the coordinator is called *cooperative*.

According to the coordination system, some advantages and disadvantages appear in the control performance of an autonomous vehicle. After testing 4 well-known behavior-based architectures (Subsumption [4], Action Selection Dynamics [10], Schema-based approach [2] and Process Description Language [13] in a simulated 3D-navigation mission with an AUV some conclusions were extracted [5,6]. Competitive methods (subsumption and

action selection dynamics) show good robustness in the behavior selection and modularity when adding new behaviors. However, a bad trajectory is found when there is a continuous change of the dominant behavior. As far as cooperative methods are concerned, they have an optimal trajectory when parameters are properly tuned. However, they lack of robustness. A small change on the parameters can lead to control failures. In some circumstances, a set of behaviors can cancel the action of behaviors with a higher priority (i.e. obstacle avoidance behavior).

In this paper we use a *hybrid* approach between competitive and cooperative coordination systems with the aim of taking advantage of both. Coordination is done through hierarchical hybrid nodes. These nodes act as cooperative or competitive coordinators depending on an activation level associated to each behavior. This paper details the coordination methodology and presents an example in which an underwater robot must follow a target while avoiding obstacles. In this architecture, a set of behaviors were designed and coordinated to accomplish the task.

Making use of the high capability of Reinforcement Learning [15] for robot learning, behaviors were implemented using this technique. 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 [18] 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 [17], CMAC function approximator [11], memory-based methods [12] and Neural Networks (NN) [8]. 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 [14]. Neural Networks is a non-linear method, however, it offers a high generalization capability and demonstrated its feasibility in very complex tasks [16]. 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 [19].

In this paper the state/action mapping of each behavior is learnt with the Semi-Online Neural-Q_learning algorithm (SONQL) [7]. 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 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, see Figure 1. Results demonstrate the feasibility of the algorithm in a real-time system.

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 Semi On-Line Neural-Q_learning algorithm. In section 4, the experimental setup designed to carry out the target following task with URIS's AUV and the learning results are presented. And finally, conclusions are presented in section 5.

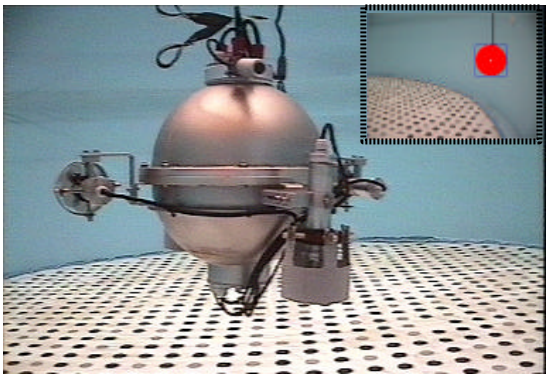

Figure 1: URIS's underwater vehicle during the target following task in a water tank.

## 2 Behavior Based control Architecture

The proposed behavior based coordinator algorithm was designed to coordinate a set of independent behaviors without the need of a complex designing phase or tuning phase. The addition of a new behavior only implies the assignment of its priority with reference to other behaviors. The BBCA uses a hybrid coordinator between competitive and cooperative methodologies; the coordinator uses this priority and a behavior activation level to calculate the resultant control action. Therefore, the response $r_i$ of each behavior is composed of the activation level $a_i$ and the desired robot control action $v_i$, as illustrated in Figure 2. The activation level indicates the degree to which the behavior wants to control the robot. This degree is expressed by a numerical value from 0 to 1.

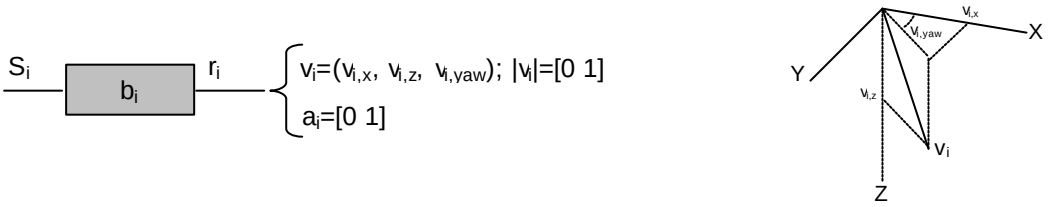


Figure 2: The normalized robot control action $v_i$ and the behavior activation level $a_i$ constitute the behavior response $r_i$

The robot control action is the movement to be followed by the robot. There is a different movement for each degree of freedom (DOF). By movement, we mean the velocity the robot will achieve for a particular DOF. In the case of the underwater robot URIS, which has 3 controllable DOFs, the control action is a vector with three components. This vector is normalized and its magnitude cannot be greater than 1. Therefore, the units of the vector $v_i$ do not correspond to any real units. After the coordination phase, this normalized vector will be re-escalated to the velocities of the vehicle.

The hybrid coordinator uses the behavior responses to compose a final control action. This process is executed at each sample time of the high-level controller. The coordination system is composed of a set of nodes $n_i$. Each node has two inputs and generates a response which also has an activation level and a control action. The response of a node cannot be discerned from one of a behavior. By using these nodes, the whole coordination process is accomplished. After connecting all the behavior and node responses with other nodes, a final response will be generated to control the robot.

Each node has a *dominant* and a *non-dominant* input. The response connected to the dominant input will have a higher priority than the one connected to the non-dominant. When the dominant behavior is completely activated, $a_d = 1$, the response of the node will be equal to the dominant behavior. Therefore, in this case, the coordination node will behave competitively. However, if the dominant behavior is partially activated, $0 < a_d < 1$, the two responses will be combined. The idea is that non-dominant behaviors can modify the responses of dominant behaviors slightly when these are not completely activated. In this case, the node will behave cooperatively. Finally, if the dominant behavior is not activated, $a_d = 0$, the response of the node will be equal to

the non-dominant behavior. These nodes are called *Hierarchical Hybrid Coordination Nodes* (HHCN) as its coordination methodology changes depending on the activation level of the behaviors and the hierarchy between them.

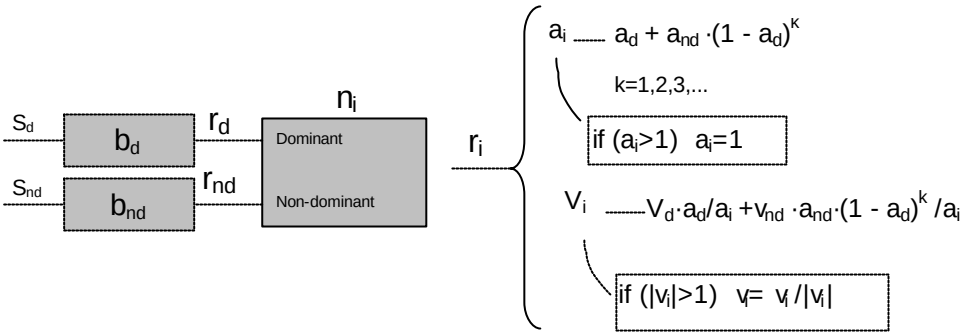


Figure 3: Hierarchical Hybrid Coordination Node. The equations used to calculate the response of the node are shown.

Figure 3 shows the equations used to calculate the response of an HHCN. The activation level will be the sum of the activation levels of the input responses, in which the non-dominant activation level has been multiplied by a reduction factor. This factor, $(1-a_d)^k$, depends on the activation of the dominant behavior and on the value of the integer parameter $k$. If $k = 1$, the activation level will linearly decrease as $a_d$ increases. If more drastic reduction is desired, the value of $k$ can be set at 2, 3, 4,... This parameter does not have to be tuned for each node. The same value, for example a quadratic reduction $k=2$, can be applied to all the coordination nodes. Finally, if the new activation level is larger than 1, the level is saturated to 1.

The control action is calculated in the same way as the activation level. Vector $v_i$ will be the sum of $v_d$ and $v_{nd}$, applying the corresponding proportional factors. Therefore, each component of $v_d$ will be taken in the proportion of the activation level, $a_d$ with respect to $a_i$. And, each component of $v_{nd}$ will be taken in the proportion of the reduced activation level, $a_{nd}$ with respect to $a_i$. If the module of $v_i$ is larger than 1, the vector will be resized to a magnitude equal to 1.

An example of the use of the hierarchical hybrid coordination node is seen in Figure 4. In this figure, two different situations are depicted. In the first situation, the node acts cooperatively, generating an action which mainly follows the dominant response, but it is also affected by the non-dominant response. In the second situation, the dominant behavior is completely activated and the node acts *competitively*.

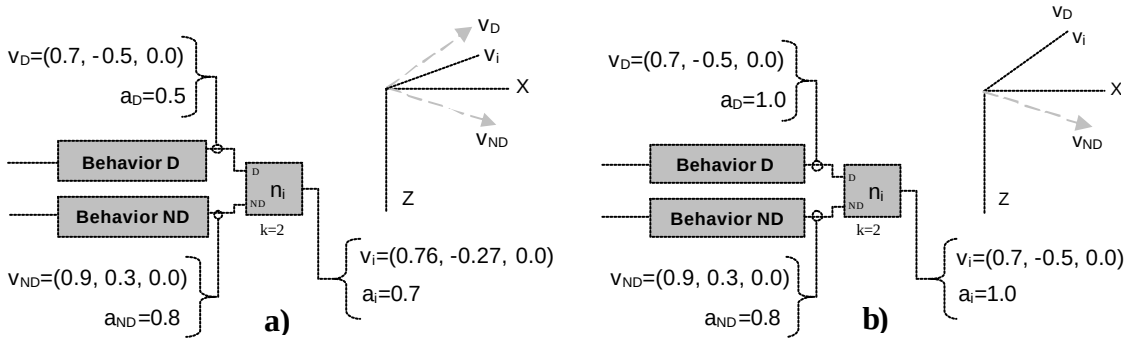


Figure 4: Response of the HHCN. Only two-dimensional actions are represented for a better understanding. The node acts cooperatively a), when the dominant behavior is not fully activated. In case the dominant behavior is completely activated, a competitive action is generated b).

As commented on above, the hybrid coordinator is composed of a set of HHCNs which connect each pair of behaviors or nodes until a final response is generated. In order to build up this network of nodes, it is necessary to set up the hierarchy among the behaviors. This hierarchy will depend on the task to be performed. Once the priorities have been set, usually by the mission designer, the hybrid coordinator will be ready to use. Figure 5 shows an example of a set of three nodes which coordinate four behaviors.

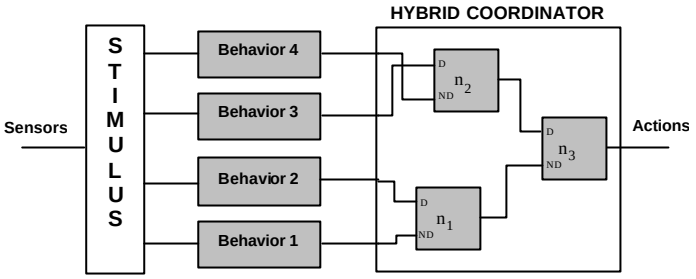


Figure 5: Example of a reactive layer with four behaviors and the hybrid coordinator. The priority among the behaviors depends on the input connections of the HHCNs.

The advantage of the hybrid coordination system is that the coordinator has good modularity. Each time a new behavior is added, the priority of the new behavior with respect to the others is the only aspect which has to be chosen. The tuning time is also a very good property since after implementing a new behavior, only the *k* parameter has to be changed. Therefore, these advantages, together with the advantage of robustness from competition and optimized trajectories from cooperation, point out the suitability of the proposed hybrid coordinator.

# 3 SONQL – Based Behavior

The SONQL algorithm is used to learn the internal mapping between state and action spaces that each behavior must contain. 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. 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.

## 3.1 Q_learning

Q_learning [18] is a temporal difference algorithm, see [15], 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 6 shows a diagram of the Q_learning algorithm. The parameters of the algorithm are the discount factor ?, the learning rate a and the e parameter for the random actions.

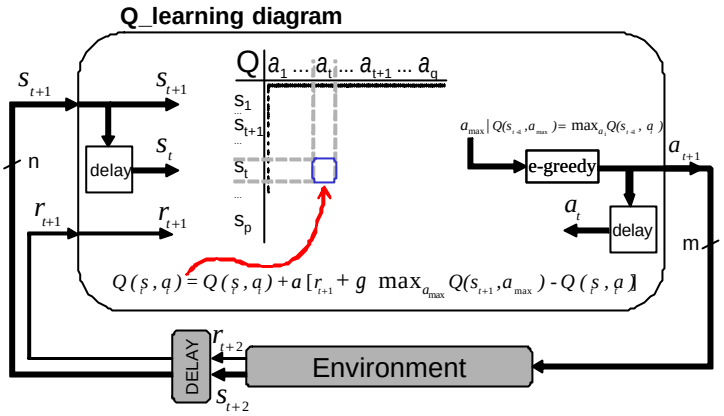


Figure 6: Diagram of the Q_learning algorithm

## 3.2 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 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 backpropagation algorithm [9] is known as direct Q_learning [3]. 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.

$$Q(s_t, a_t) = r_t + g \; max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{1}$$

The direct Q_learning appro ach 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.

## 3.3 Semi-Online Learning

Neural Networks have a high generalization capability; however they suffer from the interference problem [19]. 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 SONQL 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 SONQL 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.

## 3.4 Algorithm phases

The proposed Semi-Online Neural-Q_learning algorithm can be divided in four different phases, as shown in Figure 7. In the first phase, the current learning sample is assembled. The state $s_{t+1}$ and the last taken action at 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.

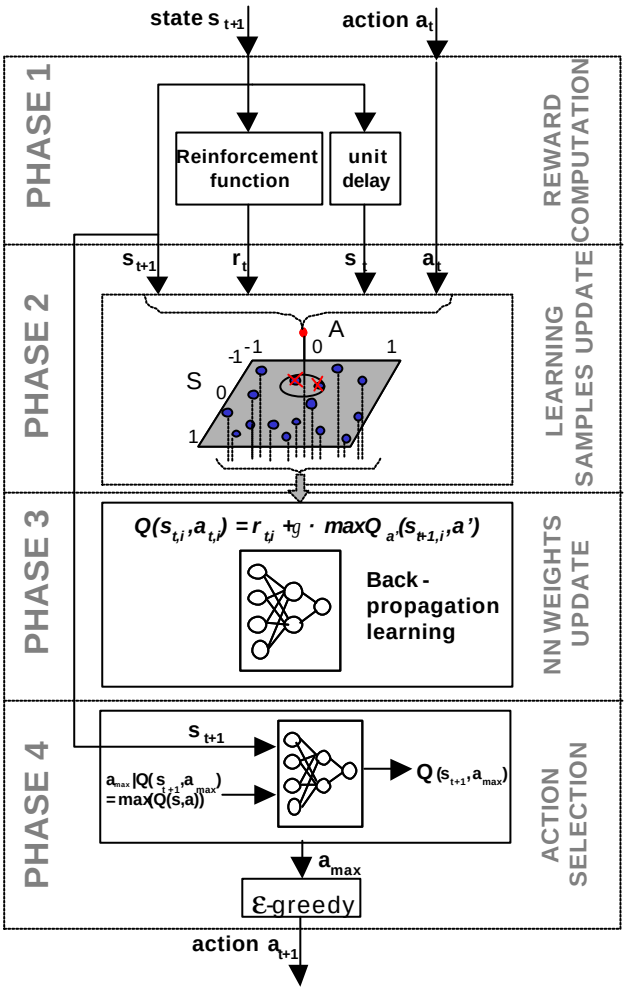


Figure 7: Phases of the semi-Online Neural-Q_learning algorithm.

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 *e-greedy* policy. With probability *(1- e)*, the action will be the one that maximizes the Q-function in the current state $s_{t+1}$. Otherwise, a 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.

# 4 Results

## 4.1 URIS's Experimental Setup

The proposed target following task, to test the hybrid coordination system and the SONQL-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.

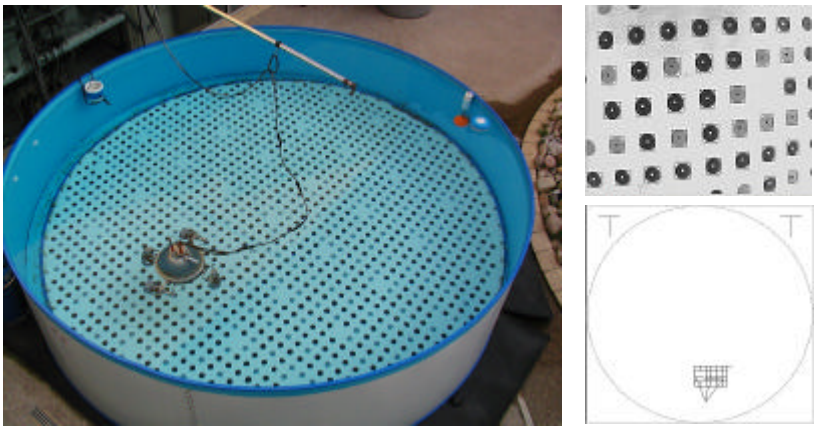

Figure 8: Water tank with the positioning pattern used in the experiments with URIS.

A water tank (4.5 m. diameter and 1.2 m. depth) was used to test the control system and to perform the experiments, see Figure 8. 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 ( $\dot{x}, \dot{z}, \dot{yaw}$ ). 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 the entire 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.

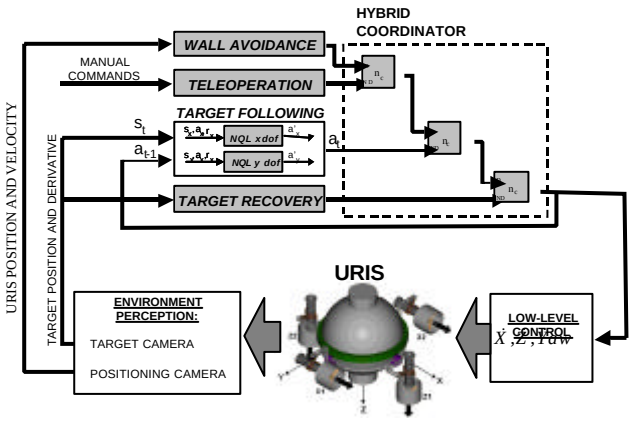


Figure 9: URIS's target following task scheme.

The control scheme of the target following task can be seen in Figure 9. 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 SONQL algorithm. And finally, the target recovery behavior spun the robot in order to find the lost target. The output action of each 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.

## 4.2 Experimental Results

As previously stated, the experiments were designed to test the feasibility of the SONQL-based behaviors. 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 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.

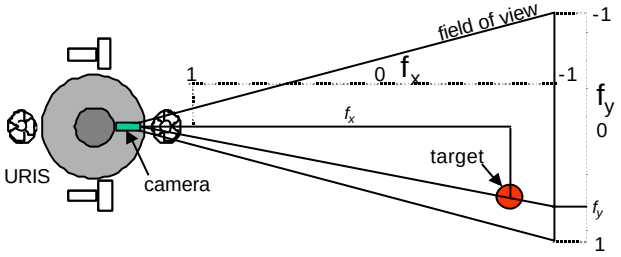


Figure 10: Coordinates of the target respect URIS.

Several trials were carried out in order to find the best learning performance. The sample time of the SONQL-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 SONQL algorithms can be seen in Table 1, and Figure 10 shows the normalization used for the target states. The learnt state/action optimal mappings can be seen in figures 11.a) and 11.b). Note the obtained non-lineal mappings.

| 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<br>$fv_x$ : derivative | $f_y$ : position<br>$fv_y$ : derivative |
| Action | $a_{fx}$ : lineal speed (X) | $a_{fy}$ :angular speed (yaw) |
| Reinfor-cement function | If $0.4>f_x>0.1$: $r_{fx} = 1$<br>if $0.5>f_x>0.4$: $r_{fx} = 0$<br>if $0.1>f_x>0.0$: $r_{fx} = 0$<br>else $r_{fx} = -1$ | If $\|f_y\| < 0.2$ : $r_{fy} = 1$<br>else if $\|f_y\| < 0.4$ : $r_{fy} = 0$<br>else $r_{fy} = -1$ |
| NQL parameters | $\alpha$=0.1; $\gamma$=0.9; $\epsilon$=0.3<br>database size = 50 samples | $\alpha$=0.1; $\gamma$=0.9; $\epsilon$=0.3<br>database size = 50 samples |
| NQL structure | inputs=3: $[f_x, fv_x, a_{fx}]$<br>output: Q_value<br>layer 1: 6 neurons<br>layer 2: 1 neuron | inputs= 3 : $[f_y, fv_y, a_{fy}]$<br>output: Q_value<br>layer 1: 6 neurons<br>layer 2: 1 neuron |

Table 1: Specifications of the target following NQL behavior

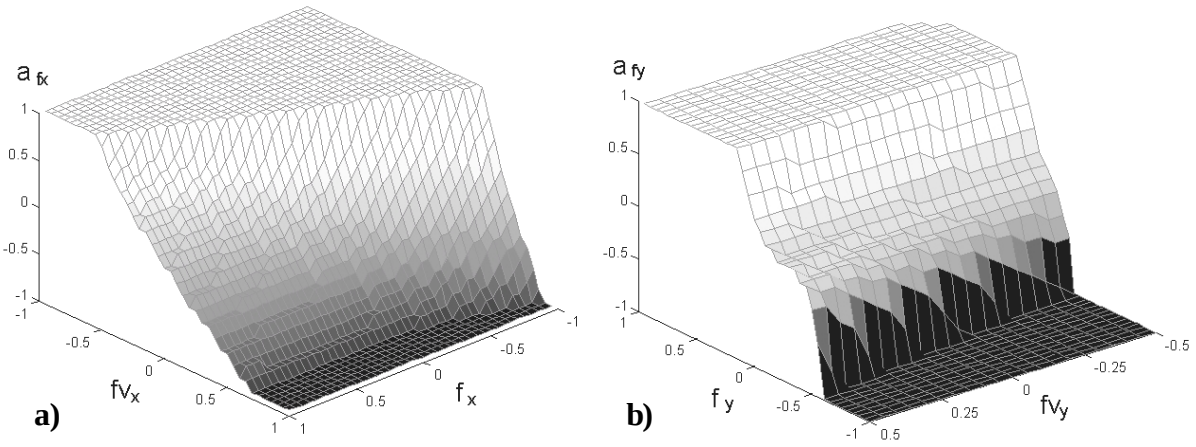


Figure 11: a) State/action mappings learnt for the target following behavior in the X DOF. b) State/action mappings learnt for the target following behavior in the Yaw DOF.

The SONQL behaviors showed a good robustness and presented a small convergence time (only 40 [s]). Figure 12 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. And in Figure 13, it can be seen how the robot followed the target at a certain distance. The target was moved manually around the water tank and, therefore, the robot trajectory was also a circumference. Note that the position of the target is approximate since there is no system to measure it.

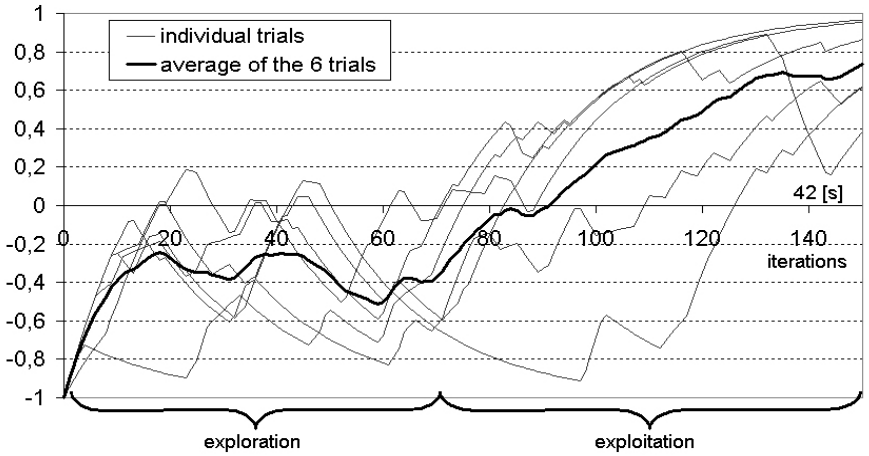


Figure 12: Behavior convergence proof. Results of six different learning trials of the target following behavior in the Yaw DOF. The averaged rewards over the last 20 iterations are shown. The average of the six experiments can also be seen. Accumulated rewards increased in measure as the behavior was learnt.

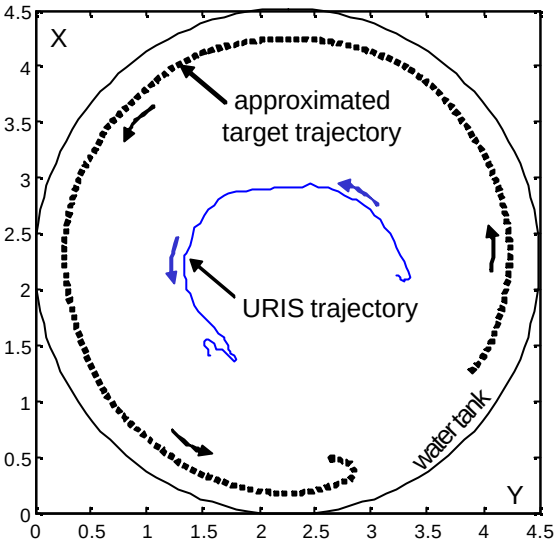


Figure 13: Trajectory of URIS while following the target in the water tank.

# 5 Conclusions

This paper has proposed a hybrid coordination method for Behavior-based control architectures. The methodology is able to coordinate a set of behaviors cooperatively when priority behaviors are partially activated, but also competitively when they are fully activated. As a second contribution, the paper has proposed the use of the Semi-Online Neural-Q_learning algorithm to learn the internal state/action mapping of the behaviors. The architecture has been tested with the underwater robot URIS in a target following task. Results showed the feasibility of the hybrid approach as well as the convergence of the learning algorithm. The proposed hybrid coordination demonstrated as behaving with the robustness of competitive coordinators and with the optimized trajectories of cooperative ones. The neural network implementation of the Q_learning algorithm also demonstrated the convergence to the optimal policy, obtaining maximum rewards. 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.

# Acknowledgments

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

# References

[1] Arkin, R.C., (1998). Behavior-based Robotics, MIT Press.

[2] Arkin, R.C., (1989). Motor schema-based mobile robot navigation. *International Journal of Robotic Research*, vol. 8, is. 4, pp. 92-112.

[3] Baird, K., (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning: 12th International Conference*, San Francisco, USA.

[4] Brooks, R., (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, vol. RA-2, is.1, pp.14-23.

[5] Carreras, M., Batlle, J., Ridao, P. and Roberts, G.N., (2000). An overview on behavior-based methods for AUV control. *MCMC2000, 5th IFAC Conference on Maneuvering and Control of Marine Crafts*. Aalborg, Denmark.

[6] Carreras, M., (2000). An Overview of Behavior-based Robotics with simulated implementations on an Underwater Vehicle. *University of Girona, Spain. Informatics and Applications Institute.* Research report: IIiA 00-14-RR.

[7] Carreras, M., Ridao, P. and El-Fakdi, A., (2003). Semi-Online Neural-Q- learning for Real-time Robot Learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, USA*.

[8] Gaskett, C., (2002). *Q_learnig for Control*. PhD thesis, Australian National University.

[9] Haykin, S., (1999). Neural Networks, a comprehensive foundation. Prentice Hall, 2nd edition.

[10] Maes, P., (1990). Situated Agents Can Have Goals. *Robotics and Automation Systems*, vol. 6, pp. 49-70.

[11] Santamaria, J.C., Sutton, R.S. and Ram, A., (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6:163-218.

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

[13] Steels, L., (1993). Building agents with autonomous behavior systems. The artificial route to artificial intelligence. *Building situated embodied agents*. Lawrence Erlbaum Associates, New Haven.

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

[15] Sutton, R.S. and Barto, A., (1998). Reinforcement Learning, MIT Press.

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

[17] Uther, W.T.B. and Veloso, M.M., (1998). Tree based discretization for continuous space state reinforcement learning. In *Proceedings of the 15th National Conference on Artificial Intelligence*.

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

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