Learning in real robots from environment interaction

P. Quintía, R. Iglesias, M.A. Rodríguez, C. V. Regueiro and F. Valdés

***Abstract*—This article describes a proposal to achieve fast robot learning from its interaction with the environment. Our proposal will be suitable for continuous learning procedures as it tries to limit the instability that appears every time the robot encounters a new situation it had not seen before. On the other hand, the user will not have to establish a degree of exploration (usual in reinforcement learning) and that would prevent continual learning procedures. Our proposal will use an ensemble of learners able to combine dynamic programming and reinforcement learning to predict when a robot will make a mistake. This information will be used to dynamically evolve a set of control policies that determine the robot actions.**

***Index Terms*—continuous robot learning, robot adaptation, learning from environment interaction, reinforcement learning.**

I. Introduction

On line robot learning and adaption is a key ability robots must have if we really want them working in everyday environments. Robots must become part of everyday life as assistants, be able to operate in standard human environments, automate common tasks, and collaborate with us. Robotic devices are meant to become a nearly ubiquitous part of our day-to-day lives. Despite the increasing demand for personal robots able to educate, assist, or entertain at home, or for professional service robots able to sort out tasks that are dangerous, dull, dirty, or dumb, there is still an important barrier between the latest developments on research robots and the commercial robotic applications available. To overcome the frontier amongst commercial and research robots we believe that, like humans, robots should be able to learn from their own experiences when emulating people or exploring an environment. The mistakes and successes the robot makes should influence its future behaviour rather than relying only on predefined rules, models or hardwire controllers. This will result in robots that are able to adapt and change according to the environment. These robots should not use pre-defined knowledge, on the contrary most of their competences should be learned through direct physical interaction with the environment and human observation.

From our experience working with robots, we think that it is true to say that no matter how perfect our robot controller is, there are always unexpected situations or different environments that will make our robot fail. We always promoted the use of reinforcement learning as an interesting paradigm that can be used to learn from robot-environment interaction [7]. Nevertheless, the application of reinforcement learning algorithms still suffer from the same problem just described. Generally, reinforcement learning is applied to get a robot learning a behaviour on simulation and, once the robot-controller is learnt it is placed on the real robot. Nevertheless, if the real robot misbehaves, it would be necessary to investigate the reasons behind the robot mistakes and to learn the behaviour once again trying to include situations similar to those that caused the failure. There are some previous publications that highlight the interest of real robot learning from reinforcement and for different applications [8], [9]. Nevertheless, most of these works achieve the desired behaviour through a careful parametrization of the action space or the reinforcement function but they do not re-design the classic reinforcement learning algorithms to get real-time learning processes. Some other works deal with the problem of real robot learning from scratch. One of them is [1]. In this work the robot explores the environment and it uses its own experiences to learn a model of the environment. This model is used to improve the control policy and thus achieve quasi-online reinforcement learning. In our case we will explore on-line learning without models, i.e. without probabilistic transition matrices.

We are interested in getting continuous learning procedures that are never stopped. The idea is that robot would move in the environment using a behaviour learnt on simulation, nevertheless, the robot would be able to adapt and modify the behaviour according to the environment where it is moving. The achievement of continuous learning requires the development of systems able to fulfil three characteristics:

1) The learning must be as fast as possible
2) Every time the robot encounters new problems, it will have to learn and improve the controller. Nevertheless, this should not cause important instabilities or make the robot forget important aspects of what had been learnt before
3) It should be possible to incorporate new knowledge or destroy old one, at any time, without causing important robot misbehaviours

II. Achieving Fast Learning processes

We need robots that are able to learn the suitable action they must carry out for every different situation (state) they might encounter, and thus reach a particular behaviour. Reinforcement Learning is suitable to get robots learning from their own experiences and environment interaction. Nevertheless, from our previous work [2], [3], [4], [5], [6] we know that reinforcement learning is too slow and requires too many trials to learn a particular task. This makes its application on a real robot almost impossible. Due to this, instead of building a learning system that needs to determine the suitable action

M.A. Rodríguez, R. Iglesias, P. Quintía are with the Centro de Investigación en Tecnologías de la Información de la Universidade de Santiago de Compostela (CITIUS). Campus Vida. Universidade de Santiago de Compostela. E-mail: roberto.iglesias.rodriguez@usc.es

C. V. Regueiro is with the Department of Electronics and Systems, of the Universidade de A Coruña.

F. Valdés is with the Electronics Department, Polytechnic, University of Alcalá.

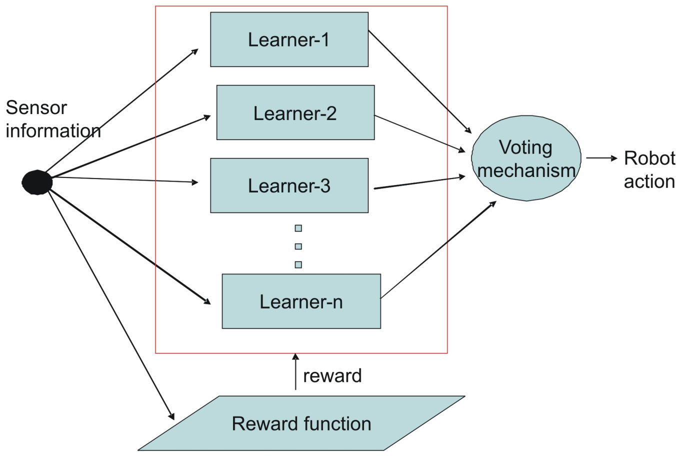

Fig. 1. Ensemble of independent learners to achieve fast learning processes

.

for every state of the robot, we prefer to build an ensemble of parallel learners able to determine, each one of them, the interval of actions most suitable for each state of the robot [3], [4], Figure 1. There is a clear example to illustrate this: Imagine a very diligent student who after three hours studying learns by heart the multiplication tables and makes no mistakes when the teacher asks him. On the other hand imagine a group of not-so-diligent students who prefer to have fun and prepare the exam at the last moment. In this case, given the lack of time, each student decides to learn only a random selection of times tables. These students can still get a good mark if they make the exam altogether. Whenever the teacher enquires about the result of a multiplication the answer is the number voted by the majority of the students. The qualification will be good provided that each student studied a random selection of the multiplication tables and those students who don't know the answer to the teachers question make it up (random guess and independent answers). In this case, given the question of the teacher, those students who didn't study the right times tables will not agree (most probably situation), while the rest of the students will agree with each other thus achieving majority.

Therefore, we have built an ensemble of independent learners like the one shown in Figure 1. This ensemble will use a voting mechanism to decide the action to be executed by the robot at every instant. Each one of the learners will have to learn a mapping between world states and actions. This mapping, also called policy, enables a robot to select an action base upon its current world state.

A. Using Fuzzy ART networks to building a representation of the world

Each learner of the ensemble shown in Figure 1 will have to build a map between world states and actions. This is a problem that lies at the heart of many robotic applications. This mapping, also called policy, enables a robot to select an action based upon its current world state. Therefore, the first problem to deal with is how to represent the world through a finite set of states. In our case, and as we can see in Figure 2, each learner will build a representation of the environment that will dynamically increase to include new situations that have not been seen before. We shall call to these new and distinguishable situations, detected in the stream of sensor inputs, states. This dynamic representation of the environment will be independent for each learner, i.e., each learner can see the world differently from the others. To quantify the sensor space we decided to use a The Fuzzy Adaptive Resonance Theory (Fuzzy ART) [10] to build the state representation for each learner. The FuzzyART clusters robot sensor readings into a finite number of distinguishable situations that we call *states*. Therefore, the FuzzyART network will achieve a sensor-state mapping that will dynamically increase to include new situations, detected in the stream of sensor inputs, and that have not been seen before.

Basically the FuzzyART will divide the sensor space into a set of regions (Vector Quantization). Each one of these regions will have a representative or prototype representing it. The FuzzyART works on the idea of making the input information resound with the representatives or prototypes of the regions into which the network has divided the sensor space so far. We call to these regions, states. If resonance occurs between the input and any of the states, this means that they are similar enough; the network will consider that it belongs to this state and will only perform a slight update of the prototype, so that it incorporates some characteristics of the input data. When the input does not resound with any of the stored states, the network creates a new one using the input pattern as its prototype.

The input of the Fuzzy ART will be an *M*-dimensional vector, where each of its components is in the interval [0, 1]. In our case, the input data we are dealing with comprise the information provided by a laser rangefinder and sonar sensors, but other sources of information are valid (e.g. grey levels of an image, or joint angles in a robotic arm). The prototypes of the states will be codified as arrays of M dimensions with values in [0, 1]: $w_j = (w_{j1}, , w_{jM})$. We shall use the letter N to refer to the number of states learnt by the network so far.

The behaviour of the Fuzzy ART is determined by two parameters: learning rate $\beta \in [0, 1]$; and a vigilance parameter $\rho \in [0, 1]$. The way the Fuzzy ART network operates can be summarised in the following steps (there are some important differences in comparison with the general proposal described in [10]):

1) After presenting an input **I** to the network, there will be a competitive process after which the states will be sorted from the lowest activation to the highest. For each input **I** and each state *j*, the activation function T_j is defined as

$$T_j(\mathbf{I}) = \frac{|\mathbf{I} \wedge w_j|}{|w_j|} \tag{1}$$

the fuzzy operator AND $\wedge$ is $(x \wedge y)_i \equiv min(x_i, y_i)$ and the norm $|\cdot|$ is defined as

$$|x| \equiv \sum_{i=1}^{M} |x_i|. \tag{2}$$

2) The state with the maximum activation value will be selected to see if it resounds with the input pattern **I**

$$J = arg\_max_j\{T_j : j = 1...N\}. \quad (3)$$

3) The Fuzzy ART network will enter in resonance if the matching between the input **I** and the winning state **J** is greater or equal than the vigilance parameter ρ:

$$|\mathbf{I} \wedge w_J| \geq \rho|\mathbf{I}| \quad (4)$$

If this relation is not satisfied, a new state will be created and the new prototype vector will be equal to the input **I**.

4) When the network enters in resonance with one input, the prototype vector w_J is updated:

$$w_J^{(new)} = \beta\mathbf{I} + (1-\beta)w_J^{(old)}. \quad (5)$$

A proliferation of states can be avoided if inputs are normalised:

$$|\mathbf{I}| \equiv \gamma, \forall\mathbf{I}, \gamma > 0 \quad (6)$$

The complement coding normalisation rule achieves normalisation while preserving amplitude information. The complement coded input *I* to the recognition system is the 2M-dimensional vector

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) \equiv (a_1, ..., a_M, a_1^c, ..., a_M^c), \quad (7)$$

where $a_n^c = 1 - a_n$. Using complement coding, the norm of the input vector will always be equal to the dimension of the original vector.

The vigilance parameter ρ is the most important parameter for determining the granularity of the classification. Low values for the vigilance parameter will create few classes. As the value of ρ approaches one, there will be almost one state for each sensor reading.

Each learner of the ensemble that we suggest (Figure 1) will use a near-random vigilance value to build a state representation from the sensor inputs. Since the vigilance parameter is different for each learner, so will be the partition of the sensor space into regions; the size of the regions into which the sensor space is divided will change from learner to learner, Figure 2. This helps to get a better generalization during the learning process.

Other artificial neural networks, such as the *Echo State Networks* [11] have been used in the past to learn from robot-environment interaction. Nevertheless, these networks are most appropriate to learn from demonstrative processes in which a user teaches the robot the desired control policy. In our case we need to use unsupervised techniques able to quantify the sensor space in a set of regions according to how similar the values coming from the sensors are, the best action for every one of these states will have to be discovered by the robot.

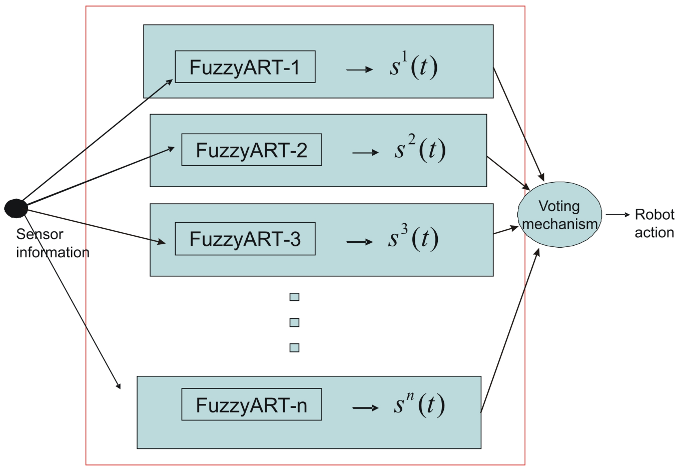

Fig. 2. Each learner of the ensemble builds a its own representation of the sensor space, i.e. the current state of the world is not described in the same way for all learners .

B. Policy derivation and performance

As mentioned before, each learner of the ensemble (Figure 1) will achieve a control policy that maps states into actions. Therefore, besides the problem representing the environment (described in the previous section), we also need to deal with the problem of how to represent the robot actions. In our case, and similarly to what happened with the states, each control policy divides the action space A in a set of intervals. Nevertheless, this set of intervals is different for each control policy ($A^1, A^2, ..., A^N$), Figure 3. We decided to use different partitions of the action space for each learner in response to the necessity of improving the generalization ability of our proposal. Each partition A^l, $\forall l = 1, ..., N$ is built randomly, but it must verify the following properties:

$$\begin{array}{l} A^l = ... \\ ... = \{A^l(1) = [a_1^l, b_1^l), A^l(2) = [a_2^l, b_2^l), ... \\ ..., A^l(P) = [a_P^l, b_P^l]\}, \forall l = 1, ..., N \end{array} \quad (8)$$

- A^l covers all the action space, A, $\forall l = 1, ..., N$:

$$\cup_j A^l(j) = A$$

- A^l is a pairwise disjoint collection of intervals, $\forall l = 1, ..., N$:

$$A^l(j) \bigcap A^l(k) = \phi$$

The cardinality (number of intervals) is the same for all partitions:

$$cardinal(A^i) = cardinal(A^j), \quad \forall i, j = 1, ..., l$$

Therefore, in our case, a control policy π is a function that determines for every possible state of the robot, the interval of actions that seems to be suitable for the task. Since we used a different FuzzyART network for every learner, i.e., the sensor-state mappings are different for each learner, and so are the actions intervals, it is true to say that:

$$\begin{array}{lcl} \pi^l : \quad S^l & \rightarrow & A^l \\ s \in S^l & \rightarrow & \pi^l(s) \in A^l \end{array} \quad (9)$$

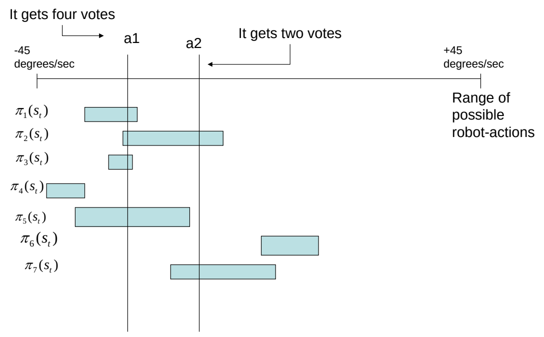

Fig. 4. Example of a voting procedure. Each learner suggests an interval of actions. The action most voted is the one the robot finally executes.

The final action the robot performs is selected after a voting procedure that considers these control policies (Figures 3 and 4). The action most voted is the one that the robot finally performs.Although there are a large variety of techniques to combine different sources of information, we selected *majority vote* since it is one of the simplest and most known strategies.

There is still an unanswered question: how does each learner know which interval of actions vote at each state?. The answer to this question is a utility function of states and action-intervals called Q. Each learner l from the ensemble will learn a utility function Q^l. These utility functions $Q^1, ..., Q^N$, are functions of states an action-intervals: $Q^l(S^l \times A^l)$. Basically each value $Q^l(s \in S^l, A^l(j))$ represents how good is executing any action a, included in the interval $A^l(j)$, when the robot is in state s. To learn these functions we have used the algorithm *Improving Time before a Robot Failure* [3], [4]. In this algorithm $Q^l(s^l, A^l(j))$ represents the expected time interval before a robot failure when the robot starts moving in $s \in S^l$, performs an action of the interval $A^l(j)$, and follows an optimal control policy thereafter:

$$Q(s, A^l(j)) = E[-e^{(-Tbf(s_0=s^l \in S^l, a_0=a \in A^l(j))/50T)}], \quad (10)$$

where $Tbf(s^l, A^l(j))$ represents the expected time interval (in seconds) before the robot does something wrong, when it performs any action $a \in A^l(j)$ in s^l, and then it follows the best possible control policy. T is the control period of the robot (expressed in seconds). The term $-e^{-Tbf/50T}$ in Eq. 10 is a continuous function that takes values in the interval $(-1, 0)$, and varies smoothly as the expected time before failure increases.

If we are able to predict the consequences of performing any action of a particular interval of actions in a state (time that will elapse before the robot makes a mistake), it is straightforward that we can achieve the best control policy by simply selecting the interval with the highest Q-value for every state. This will give us the control policy that maximizes the time interval before any robot failure; this is called greedy policy π^*:

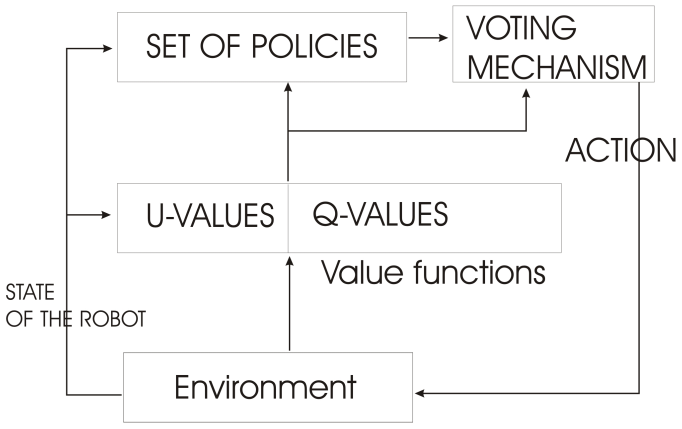

Fig. 5. General schema of our proposal

$$\pi^{l*}(s^l) = arg\_max_k\{Q^l(s^l, A^l(k))\}, \quad ,\forall s^l \in S^l \quad (11)$$

There is a greedy policy for every control policy in our system. Since the $Q^l(,)$ values and $Tbf(,)$ are not known, we can only refer to their current estimations $Q^l_t(,)$ and $Tbf_t(,)$. Starting from the Q-values we can determine the expected time before a robot failure for an action-state pair:

$$Tbf^l_t(s, a) = -50 * T * Ln(-Q^l_t(s, A^l_a)), \quad (12)$$

where A^l_a is the interval of A^l that contains action a. We need to use the super-index l for Tbf^l_t since it is estimated from the Q^l values. We can also determine the expected time before a robot failure for a state:

$$Tbf^l_t(s^l) = max_k\{-50 * T * Ln(-Q^l_t(s^l, A^l(k)))\}, \quad (13)$$

We can notice that the time before failure, determined for a given state, considers the best expectation, i.e., when the robot follows the greedy policy.

Initially, each learner will build a control policy that coincides with its corresponding greedy policy:

$$\pi^l(s \in S^l) = \pi^{1*}(s \in S^l), \quad \forall l = 1, ..., N \quad (14)$$

These greedy policies $\pi^{1*}, ..., \pi^{N*}$ are got from the Q-values following the straightforward process indicated in Eq 11. The final action the robot performs would be selected after a voting procedure (Figure 4). Nevertheless, to increase the robustness of our proposal, the voting procedure considers a new value-function (Figure 5) that determine how good is a particular policy for a given state, i.e., whether the interval of actions that particular policy suggests for the state seems to be right for the task or not. We represent this new value function with the letter *U*. Thus $U^j(s) = 1$ means that the j-control policy is invalid for state s, and hence its vote can be discarded. On the contrary $U^j(s) = 0$, means that the control policy j seems to be right for the state s, and therefore its vote must be taken into account. Eq. 15 summarizes the voting procedure:

$$a_t = argmax_{\forall a \in A}\{\sum_1^N \delta[a \notin \pi^l(s^l(t))] * (1 - U^l(s^l(t))\} \quad (15)$$

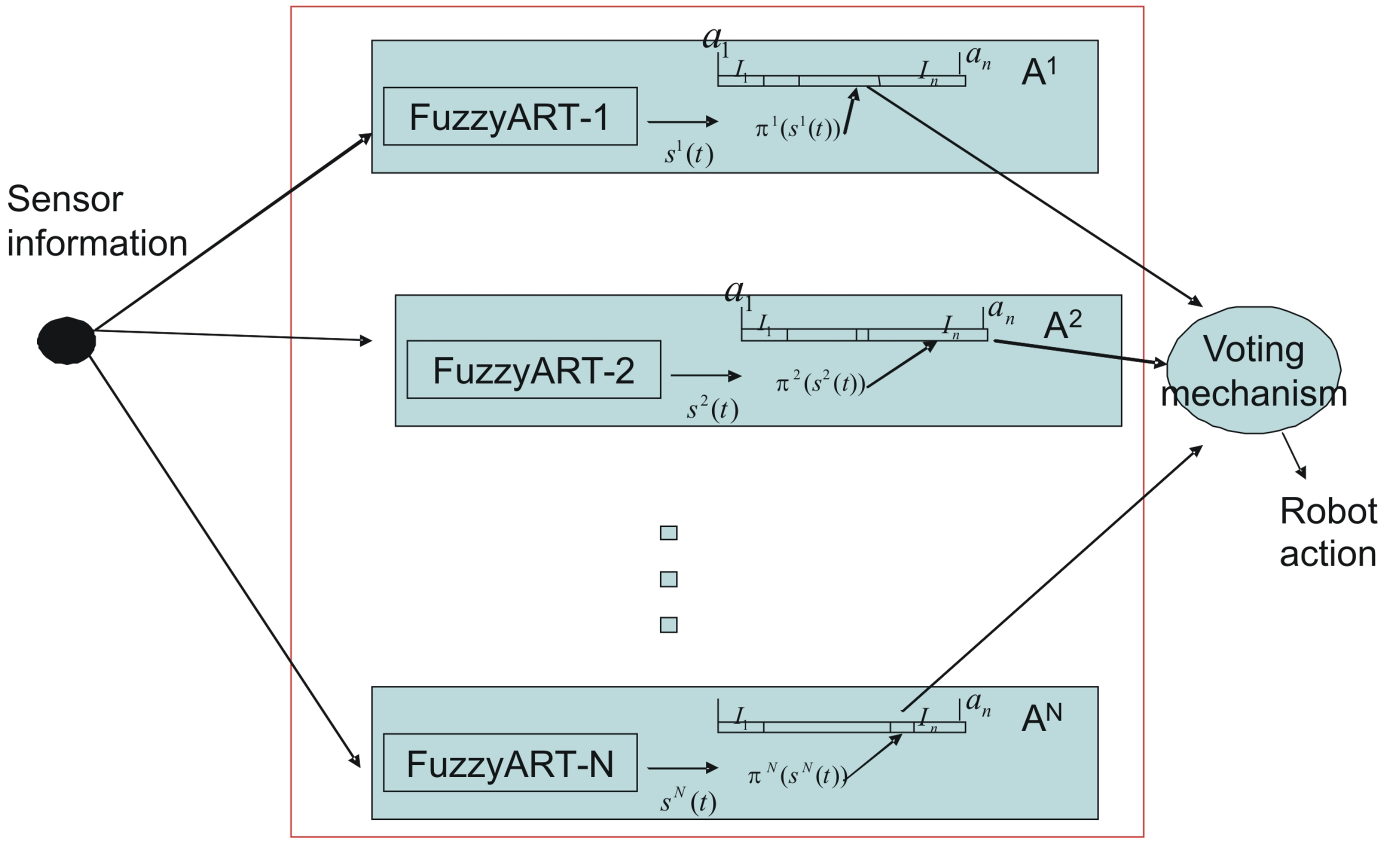

Fig. 3. In our proposal each learner of the ensemble builds a quantization of the sensor space and the action set that is different and independent from the rest of the learners.

δ is the kronecker delta, i.e:

$$\delta[a \notin \pi^l(s^l(t))] = \begin{cases} 1 & if a \in \pi^l(s^l(t)) \\ 0 & if a \notin \pi^l(s^l(t)) \end{cases} .$$

As we will see in the next subsection, the new utility function U will also determine when the control policies $\pi^1, ..., \pi^N$ are updated using the greedy policies $\pi^{1*}, ..., \pi^{N*}$.

C. Learning of the Utility Functions

As we have described in the previous section, the action the robot should execute at each instant is determined using a voting procedure that is highly influenced by two utility functions: Q and U (Figures 5 and 6). Both utility functions are independent for each learner of the ensemble: $Q^1, ..., Q^N$ and $U^1, ..., U^N$. In this section we will describe how the values of all these utility functions are inferred from the robot experiences when it interacts with the environment.

When the robot is moving interacting with the environment performing different actions, we can dynamically update the Q-values. Thus, if we consider a robot that has performed the action a_t in the current state s_t, and as an outcome the robot has moved to state s_{t+1} and has received the reinforcement r_t, the Q^l values corresponding to every policy, can be updated taking only into account the relationship amongst consecutive states:

$$Q^l_{t+1}(s^l_t, A^l_a) = \begin{cases} -e^{-1/50} & \text{if } r_t < 0 \\ Q^l_t(s^l_t, A^l_a) + \delta & \text{otherwise} \end{cases} \quad (16)$$

where,

$$\delta = \beta_L(e^{\frac{-1}{50}} * Q^l_{max}(s_{t+1}) - Q^l(s_t, A^l_{a_t})). \quad (17)$$

r_t is the reinforcement the robot receives when it executes action a_t in state s_t, $\beta_L \in [0, 1]$ is a learning rate, and it is the only parameter whose value has to be set by the user. Finally, $A^l_{a_t}$ is the interval of A^l that contains action that has been performed by the robot, a_t.

Equations 16 and 17 allow the updating of the Q-values using the experiences of the robot. Basically the robot would move in the environment, trying the execution of different actions and, at every instant, the Q-value of the last action performed by the robot would be updated according to the current state of the robot and the reinforcement that the robot received. Nevertheless, this would lead to very slow learning processes. A common solution to speed up this learning consists on updating the Q-value of each action performed by the robot considering not only the immediate consequences of the execution of this action, but also what the robot does and the reinforcement it receives during a short interval after the execution of the action which is being considered. Thus, to obtain the utility values $Q^l(s, A^l(k))$, the robot begins with an initial set of random values, $Q^l(s, A^l(k)) \in [-1, -0.95]$, and then it initiates a stochastic exploration of its environment. The robot will move and collect data during a maximum period of time or until it makes an error. The data collected will later be used to update the Q-values:

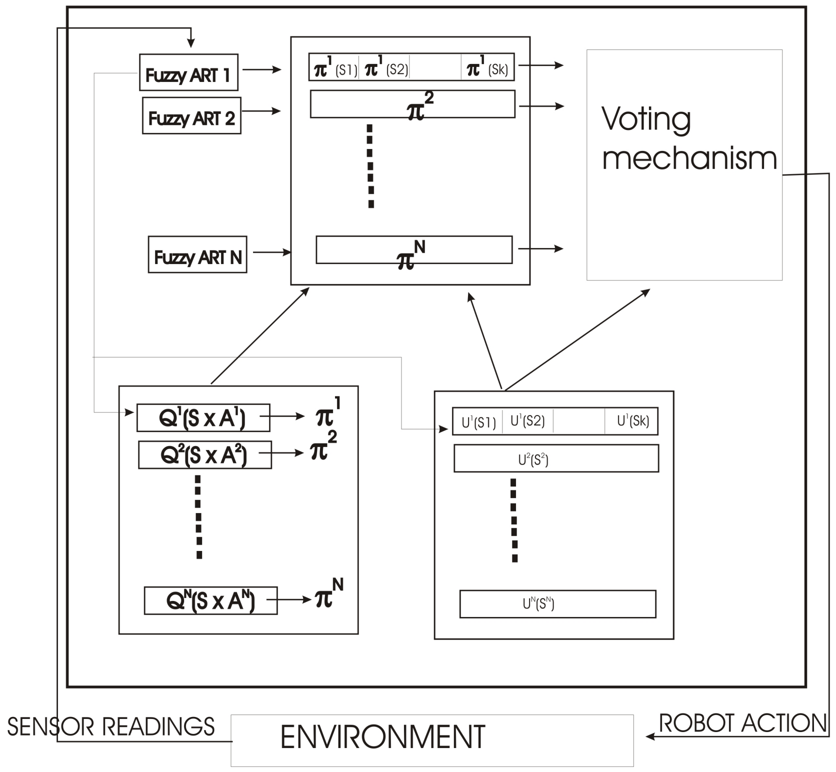
Fuzzy ART 1
Fuzzy ART 2
Fuzzy ART N
$\pi^1(S1)$ $\pi^1(S2)$ $\pi^1(Sk)$
π^2
π^N
Voting mechanism
$Q^1(S \times A^1)$ π^1
$Q^2(S \times A^2)$ π^2
$Q^N(S \times A^N)$ π^N
$U^1(S1)$ $U^1(S2)$ $U^1(Sk)$
$U^2(S^2)$
$U^N(S^N)$
SENSOR READINGS
ENVIRONMENT
ROBOT ACTION

Fig. 6. Detailled schema of our proposal

First stage, collecting data

1) $m = 0$
2) At each instant t and while the robot does not receive a negative reinforcement or moves for a maximum period of time do:
   a) Observe the current state in every learner of the ensemble, $s^l(t)$.
   b) Select an action a_t to be executed by the robot (through a voting procedure).
   c) Perform action a_t, observe new states s^l_{t+1} and reinforcement value.
   d) If $r_t >= 0$ then $m \leftarrow m + 1$ and go back to the first step, otherwise stop collecting data and do not shift the m-index

Second stage, updating the Q-values:

1) Update *time before failure* for every learner of the ensemble:
   a) for $k = 0, 1, \ldots, m$ do:
      if $k = 0$ then:

$$Tbf^l = \begin{cases} T & \text{if } r_{t-k} < 0 \\ Tbf^l(s_{t-k}) & \text{otherwise} \end{cases}$$

      else $Tbf^l \leftarrow \lambda * (Tbf^l + T) + (1 - \lambda) * Tbf^l(s_{t-k})$.
2) Update the Q-values of the first record in the experience set:
   - $\Delta Q^l_t(s_{t-m}, A^l_{a_{t-m}}) = \beta_L(-e^{-Tbf^l/50T} - Q^l_t(s_{t-m}, A^l_{a_{t-m}})), \forall l = 1, .., N$
3) Shift the index m which represents the number of states recorded in the experience set and which have not been updated: $m \leftarrow m - 1$
4) if $m = 0$ exit, otherwise go to the first step, 1)

The parameter λ (second stage, first step) is a parameter that determines the relative importance of the last robot-environment interactions. The higher the values of lambda, the more the Q-values are altered considering the last robot-environment interactions. Low values of lambda are recommended when either the rewards or the sensor readings are noisy.

The updating of the new utility function, *U*, is straightforward: those policies that voted for the actions the robot executed far from any failure will see their *U* values decreased. On the contrary, those policies that voted for the actions executed just before a robot failure will see their *U* values increased, thus reducing their future consideration for the same states. Finally, those policies that voted for actions that finally weren't executed will not see their *U* values altered. This way of updating the *U* values means that the robot will tend to repeat the same sequences of actions that it has already tried in the past and did not cause any trouble, but the robot will change its behaviour whenever the actions it carried out lead the robot to some kind of failure.

Figures 5 and 6 give an overview of our system. Each time the robot makes a mistake and it receives negative reinforcements, the set of policies used to control the robot will evolve trying to improve robot's behaviour. Basically, each policy will mutate in those actions that have lost their vote ($U^j(s)$ is not null), in this case the new interval of actions will coincide with the greedy policy:

- for i=1,...,N
  - for j=0,...,maximum number of states in S^i
    - if $U^i(j) > 0$ $\pi^i(s^i(j)) = \pi^{i*}(s^i(j))$

The use of the greedy policies to update the control policies, allows faster and faster recoveries from robot misbehaviours due to the use of the past experiences learnt by the robot. On the other hand, we must be aware of the fact that any time the robot makes a mistake the use of the function U limits the number of mutations and prevents the system from falling into heavy instabilities. This is something new that allows us to improve the generalization ability of our system.

To get independent and uncorrelated policies, only a subset of policies will take part in the decision making at each instant. This subset of M control policies ($M < N$) that takes part on the decision making, is determined randomly, and it will be the same until the end of the learning process. Nevertheless, every time the robot makes a mistake this subset is increased with some new randomly selected control policies. Adding new control policies during the learning process represent a way of incorporating new knowledge into the system (this was the third characteristic mentioned in the introduction and that continual learning would require). These new policies are initially random but they start learning and evolving from the moment they are incorporated into the system.

III. Experimental results

We have performed an experimental study of the strategy described in the previous section. In particular we have implemented it on a Pioneer 3DX robot. This robot is equipped with a SICK LMS-200 laser rangefinder, a ring of 16 ultrasound sensors and bumpers. In all the experiments the linear velocity of the robot was kept constant (15.24 cm/s $\equiv$ 6 inch/s), and the robot received the motor commands every 300ms (value of T in Eq. 10). Through the experiments the robot had to learn a common reactive task: wall following. To teach the robot how to follow a wall located on its right at a certain distance interval, we used a reinforcement signal that is negative whenever the robot goes too far from or too close to the wall being followed. We tested the algorithm in both, real and simulated environments. We must emphasize that the behaviour itself is not the objective of this work, but a benchmark for all the tests we want to carry out.

A. Simulation results

Fig.7 shows the simulated environment where the robot learnt the wall following behaviour. In this figure we can also see the trajectory of the robot once the task has been learnt. To simulate the movement of the robot we have used Player&Stage [14]. We consider that the behaviour has been learnt when the robot is able to move without making any mistake during ten minutes (several laps in the same environment). The learning time is the time elapsed since the

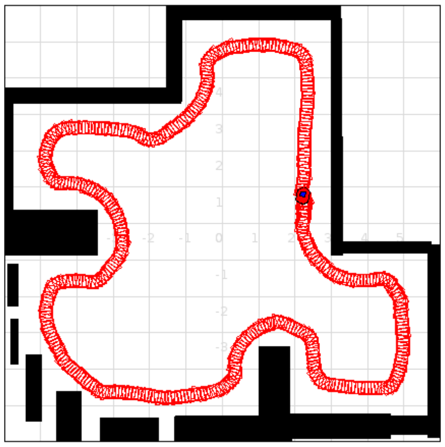

Fig. 7. A robot learning how to follow a wall located on its right. This graph was obtained during the last stages of the learning process

robot starts moving until it achieves a valid control policy (the policy that is able to move the robot without mistakes for at least ten minutes).

To learn the behaviour we used all the sensor information available. The laser sensor we used on simulation provides 177 measures – one measure per degree from 0^o to 176^o –. On the other hand we also used the information provided by 16 ultrasound sensors surrounding the robot. We then scaled the values in the interval [0,8] to the interval [0, 1):

$$I_i = \frac{1}{(1 + input_i)} - \frac{1}{9} \tag{18}$$

The learning parameters we used are: learning coefficient $\beta_l = 0.25$, $\lambda = 0.5$, the vigilance parameter for each FuzzyART neural network was selected randomly in the interval $[0.86, 0.92]$. and, finally the learning coefficient for the FuzzyART neural network was $\beta = 0$. The average learning time after 30 experiments was 14.55 minutes and the standard deviation was 8.8 minutes.

After this first set of experiments we carry out a second set of 30 experiments, but this time the processing of the sensor information was different. In this second case we scaled the sensor values using Eq. 19:

$$I_i = \frac{(8 - input_i)^2 * e^{\frac{-input_i}{1.5}}}{64} \tag{19}$$

In this second case, the vigilance parameters were selected randomly in the interval $[0.8, 0.92]$, the rest of the parameters kept the same values as in the first set of experiments. After 30 experiments we got an average learning time of 6.61 minutes and the standard deviation was 4.45 minutes.

In general, through the simulated experiments we noticed the importance of the U-function described in sections (B and C). This utility function works as a gate enabling or disabling learners in the voting mechanism. We noticed that the use of this function allows achieving the desired control policies sooner that if we do not use it.

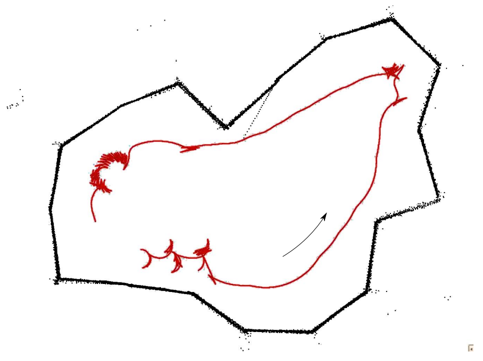
Fig. 8. First lap of a real robot learning the wall following behaviour in a real environment. The robot moves a bit backwards every time it makes a mistake and receives negative reinforcement, this can be appreciated in the robots trajectory.

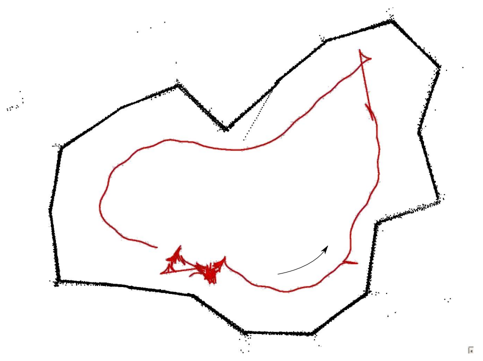
Fig. 9. Third lap of a real robot learning the wall following behaviour in the same real environment as in Figure 8.

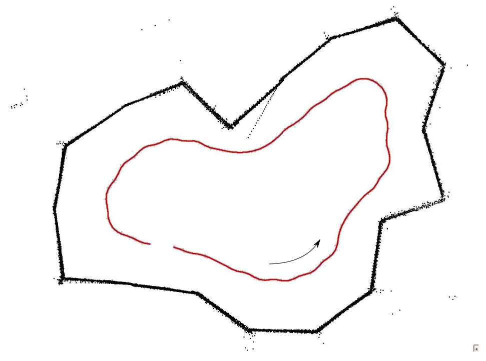
Fig. 10. Fourthlap of a real robot learning the wall following behaviour in the same real environment as in Figure 8.

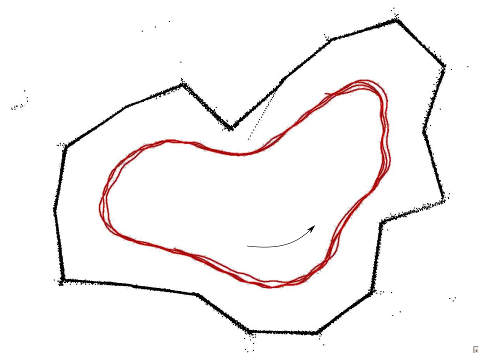
Fig. 11. In this figure we can observe several laps of a real robot following the wall located on its right in a real environment. We can appreciate the robots trajectory during several laps (from the fourth onwards) once the behaviour has been practically learnt.

B. Learning on the real robot

Since our learning algorithm proved to be very efficient and very fast on simulation, we decided to apply it to learn the behaviour on a real robot. It is really relevant to be aware of the fact that the learning process starts from scratch (i.e., our robot does not have any kind of prior knowledge about the task or how to solve it). This can be easily appreciated in the first lap of the robot in the environment, shown in Figure 8. Every time the robot receives a sequence of negative reinforcements, it stops moving and it goes back until it reaches a position where it does not receive negative reinforcements. Figures 8,9, 10 and 11 show the progress of the learning procedure on a real robot working on a real environment. We must emphasize that these results represent a huge achievement since little work has been done with learning processes on a real robot interacting with the real environment. Moreover, these learning processes on the real robot overcome the limitations of the simulation, i.e., when the simulator is not realistic enough the behaviour learnt might not work on the real robot.

IV. Conclusion

In this paper we have described a system that moves us close to continuous reinforcement learning procedures in a real robot operating in real environments. Basically our system combines a set of control policies that is being evolved considering two utility functions. These two utility functions learn the experiences accumulated by the robot when it interacts with environment, and prevents too unstable behaviours every time the robot encounters a situation it had not seen before. It is also important to mention the fact that our system incorporates new knowledge dynamically, as the learning process progresses (this new knowledge is included as new learners in the ensemble). This not only does not cause any instability but, on the contrary, helps the robot to behave better and better improving both, robustness and generalization. The experimental results achieved, show the achievement of extremely fast learning process able to work even on real robots learning continuously in a real environment.

It is important to be aware of the fact that our system learns the representation of the environment (states created by the Fuzzy ART neural networks) and the control policy (i.e. the actions the robots must execute for each state) simultaneously. This obviously increases the complexity of the problem, although a quantification of the increase of the complexity is still part of our future work.

Acknowledgment

This work was supported by the research grants TIN2009-07737 and INCITE08PXIB262202PR

References

[1] B. Bakker, V. Zhumatiy, G. Gruener, J. Schmidhuber. *Quasi-Online Reinforcement Learning for Robots*. Proceedings of the International Conference on Robotics and Automation (ICRA-06), Orlando, Florida, 2006.

[2] M. Rodriguez, R. Iglesias, C. V. Regueiro J. Correa and S. Barro, *Autonomous and fast robot learning through motivation*, Robotics and Autonomous Systems, vol. 55, pages: 735–740, 2007.

[3] Pablo Quintia, Roberto Iglesias, Carlos V. Regueiro, Miguel A. Rodriguez, *Simultaneous learning of perception and action in mobile robots*, Robotics and Autonomous Systems, vol 58, pages: 1306–1315, 2010.

[4] M. A. Rodriguez, R. Iglesias, P. Quintia C. V. Regueiro, *Parallel robot learning through an ensemble of predictors able to forecast the time interval before a robot failure*, XI Workshop of Physical Agents, 2010.

[5] T. Kyriacou, R. Iglesias, M. Rodriguez, P. Quintia. *Unsupervised Complexity Reduction of Sensor Data for Robot Learning and Adaptation*, 11th Towards Autonomous Robotic Systems (TAROS'2010). 2010

[6] Pablo Quintia, Roberto Iglesias, Miguel Rodriguez, Carlos Vazquez Regueiro, *SIMULTANEOUS LEARNING OF PERCEPTIONS AND ACTIONS IN AUTONOMOUS ROBOTS*, 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2010), 2010.

[7] R. S. Sutton, *Reinforcement learning: An introduction*, MIT Press, 1998.

[8] Thomas Kollar, Kicholas Roy, *Using reinforcement learning to Improve Exploration Trajectories for Error Minimization*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2006), 2006.

[9] Andrea L. Thomaz, Guy Hoffman, Cynthia Breazeal, *Real-Time Interative Reinforcement Learning for Robots*, AAAI 2005 Workshop on Human Comprehensible Machine Learning, 2005.

[10] G. A. Carpenter, S. Grossberg,D. B. Rosen, *Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system*, Neural Networks, volume 4, pages: 759–771,1991.

[11] M. Oubbati, B. Kord, and G. Palm, *Learning Robot-Environment Interaction Using Echo State Networks*, SAB 2010, LNAI 6226, pp. 501-510, 2010

[12] Amanda J.C. Sharkey, *Combining Artificial Neural Nets: Ensemble and Modular Multini-Net Systems*. Springer. 1999.

[13] Lior Rokach, *Pattern classification using ensemble methods*. World Scientific. 2010.

[14] *Player & Stage Project*. http://playerstage.sourceforge.net. (Accessed on 26 February 2012).