Learning on Real Robots from Experience and Simple User feedback

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

***Abstract*—In this article we describe a novel algorithm that allows fast and continuous learning on a physical robot working in a real environment. The learning process is never stopped and new knowledge gained from robot-environment interactions can be incorporated into the controller at any time. Our algorithm lets a human observer control the reward given to the robot, hence avoiding the burden of defining a reward function. Despite the highly-non-deterministic reinforcement, through the experimental results described in this paper, we will see how the learning processes are never stopped and are able to achieve fast robot adaptation to the diversity of different situations the robot encounters while it is moving in several environments.**

***Index Terms*—Autonomous robots, reinforcement learning**

I. Introduction

Although the industrial sector has been the main user of robots for many years, nowadays there is a clear shift towards the service sector. The unquestionable success of the Roomba robotic vacuum is enough to prove that the new robotics companies would be able to sell millions of robots instead of tens of thousands. This view of a growing market is shared by all national and international robotic organizations. However, if we go back to the example of the Roomba robotic vacuum, the limited intelligence of this robot, together with its low perception ability, or its inexistent adaptability, is the reason why sometimes people need to create obstacles so that the Roomba is confined to a space and thus, purportedly, cleans it more thoroughly. The lack of intelligence in the robots is quite often replaced with the presence of a person. However, if we consider the main scenarios where future robots are expected to move, or the tasks they are expected to carry out (assisting with the housework, security and vigilance, rehabilitation, collaborating in the care-entertainment, etc), we immediately realize that this new generation of robots must be able to learn on their own. They can not rely on an expert programmer, on the contrary, once they are bought they should be *trainable*.

This learning or robot-adaptation can not only consist on a demonstration process, in which the user shows the robot what to do. The limited nature of the human patience, the ambiguous nature of the information provided by the robot owners, the advanced age or impaired mobility of the robot owners, deem it necessary that not only should robots be able to learn from what the user does, but also from their interaction with the physical and social environment. Like humans, the mistakes and successes the robot makes should influence its future behaviour. Furthermore, this adaptation should not be constrained to a time interval, but on the contrary it should be continuous, i.e., during the life of the robot.

We promote the use of reinforcement learning [1] as an interesting paradigm that can be used to learn from robot-environment interaction. There are several publications that point out the interest of real robot learning by direct interaction with the environment [14] [15]. Unfortunately, most reinforcement learning algorithms are rarely applied to get physical robots learning from scratch on real environments. Very often the desired robot-behaviour is learnt on simulation and, once the process is finished, the robot-controller is placed on the real robot. This is due to the fact that the learning process is very slow and costly. There are two common alternatives to avoid these limitations: on one hand there are authors that manage to break the task in a sequence of sub-tasks that can be learnt separately (thus reducing the learning time for each one of them). On the other hand, another alternative consists on combining the reinforcement learning with the learning from demonstration [13]. Nevertheless, in all these cases if the real robot misbehaves, it will be still 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.

We are interested in getting continuous learning procedures that are never stopped and that can happen in the real robot. The achievement of continuous learning requires the development of systems able to fulfill 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 misbehaviors.

II. Giving user feedback to robots

One way of getting a robot to learn from its interaction with the environment is through reinforcement: according to psychology theories, learning is strengthened if it is followed by positive reinforcement "pleasure" and weakened if it is followed by punishment "pain" [2]. This is something that is clearly described in Thorndike's Law of Effect [3]. Reinforcement learning algorithms grew inspired by these

P. Quintía, R. Iglesias and M. Rodríguez are with Information Technologies Research Centre (CITIUS). University of Santiago de Compostela. Spain.
{roberto.iglesias.rodriguez; miguel.rodriguez.gonzalez; pablo.quintia}@usc.es

C.V. Regueiro is with the Department of Electronics and Systems, University of A Coruña, Spain.
cvazquez@udc.es

psychological theories, therefore reinforcement learning is a machine learning paradigm that determines how an agent ought to take actions in an environment so as to maximise the amount of reward received in the long term.

Nevertheless, getting this feedback is still an important drawback: we cannot ask people without knowledge of robots to provide a set of rules which specify when the robot is doing right or wrong. A key aspect to achieve successful service robots lies in the possibility that any person (not necessarily specialist in robotics) can teach robots new tasks, or can do something that will alter the behaviour of the robot. An article published in Artificial Intelligence regarding *how the humans want to teach the machines* [4], comes to the conclusion that the processes of *teaching and learning* must be closely linked. A good *instructor* must hold a mental model of the state of the learner (trying to guess what the robot has understood, what the robot knows and what the robot is still ignorant of, etc). On the other hand the robot must help the instructor doing the learning process as clear as possible. In short, the process of teaching/learning must be bi-directional.

In [5] the authors examine the hypotheses that reinforcement provided by humans is compatible with the traditional reinforcement signal of the reinforcement learning. They perform several experiments in a simulated environment where a human must interact, using a graphical interface, with an agent implementing a Q-learning algorithm.

They conclude that new reinforcement learning algorithms should be developed to incorporate the main observations taken from their experiments:

- In addition to providing a feedback to the robot, the users want to guide the robot to the action they consider correct. On the other hand, in general humans want to give anticipated reinforcements. Although the effect of the anticipated reinforcement in the reinforcement learning has been studied in depth [6], the use of anticipated reinforcement is not part of the classical reinforcement learning model.
- Users prefer to give positive feedback than negative feedback, probably reflecting their opinions about the motivation in human learning, or due to the fact that they feel that the robot ignores their negative reinforcements.
- As the learning advances the users create a mental model of the agent and changes its teaching strategy. Classical reinforcement learning strategies do not take into account the fact that a benevolent teacher will adjust its teaching behavior to the learner characteristics.

The work of Knox and Stone [9] [10] is focused in the transfer of knowledge from humans to machines, with the goal of accelerating the learning and reducing its cost. With this purpose they created the TAMER framework (*Training an Agent Manually via Evaluative Reinforcement*). TAMER is based in modeling the reinforcement provided by the human in order to allow the system to choose those actions that provide the highest reinforcement. This strategy can be considered as inverse reinforcement learning [11].

We decided to adapt our learning algorithm so that the reinforcement should come from a human observer that is seeing what the robot does. This observer will be able to punish the robot by simply pressing a button in a wireless joystick (Figure 1). This action will be enough to tell the robot that what it is doing is unsatisfactory. It is important to be aware of the fact that this way of providing the reinforcement is highly not deterministic, i.e., the same user can give the robot negative reinforcements in certain situations but remain impassive in other scenarios that are very similar. Moreover, the human observer can change his mind about what is right or wrong while the robot is still learning. When the user presses the button of the wireless joystick to give the robot negative reinforcement, the robot learns from it and transfers the control to the joystick, so that the user will be able to move the robot and place it in a suitable position to go on learning. Once this manual control is over, the user will press a second button to continue the learning process.

Merging learning from demonstration and reinforcement learning is out of the scope of this work, therefore our algorithm will not learn from what the robot does when it is being controlled by the user. We decided to proceed in this manner since we want to highlight the ability of our algorithm to reach fast learning procedures from robot-environment interaction when the robot starts from scratch. Nevertheless, as part of our future research, we plan to allow the user to guide the movement of the robot and thus speed up the learning procedures. But in this work the user will only use the joystick to provide reinforcement and move the robot a short distance to place it in a new position, where user considers that the robot is safe and can continue the learning.

III. Evaluating a control policy

Let us say that there is a control policy π that determines what the robot does at every instant, i.e., this policy π is a mapping from relevant and distinguishable states to actions:

$$\begin{array}{lcl} \pi^l : S & \rightarrow & A \\ s \in S & \rightarrow & \pi^l(s) \in A \end{array} \quad (1)$$

where S is the set of states that represent the environment around the robot, and A is the set of possible actions the robot can carry out.

Our first interest is to evaluate this policy, i.e. to quantify how long this policy will be able to move the robot before it makes something wrong and the robot receives negative reinforcement. To carry out this task we will use an algorithm that we have published in the past and which is called *increasing the time interval before a robot failure* [8]. Using this algorithm our system will not learn the expected discount reward the robot will receive – as is habitual in reinforcement learning–, rather the expected time before failure (*Tbf*). This will make it easier to assess the evolution of the learning process as a high discrepancy between the time interval before failure predicted and what is actually observed on the real robot is a clear sign of an erroneous learning.

To assess our control policy we will build an utility function of the states the robot might encounter, termed Q-function. Thus, $Q^\pi(s)$ is a function of the expected time interval before a robot failure when the robot starts moving in s, performs

the action determined by the policy for that state $\pi(s)$, and follows the same policy π thereafter:

$$Q^{\pi}(s) = E[-e^{(-Tbf^{\pi}(s_0=s)/50T)}], \quad (2)$$

where $Tbf^{\pi}(s_0)$ represents the expected time interval (in seconds) before the robot does something wrong, when it performs action $\pi(s)$ in s, and then it follows the control policy π. T is the control period of the robot (expressed in seconds). The term $-e^{-Tbf/50T}$ in Eq. 2 is a continuous function that takes values in the interval $[-1, 0]$, and varies smoothly as the expected time before failure increases.

Since $Q^{\pi}(s)$ and $Tbf^{\pi}(s)$ are not known, we can only refer to their current estimations $Q^{\pi}_t(s)$ and $Tbf^{\pi}_t(s)$:

$$Tbf^{\pi}_t(s) = -50 * T * Ln(-Q^{\pi}_t(s)), \quad (3)$$

Therefore, according to Eq. 3, if the robot performs an action every 250 milliseconds (value of T in Eq. 3), a Q-value equal to -0.8 (for example) will clearly mean that the robot will probably receive a negative reinforcement after 2.8 seconds. The definition of $Q^{\pi}(s)$, Tbf^{π}, determine the relationship between consecutive states:

$$Tbf^{\pi}_t(s_t) = \begin{cases} T & \text{if } r_t < 0 \\ T + Tbf^{\pi}_t(s_{t+1}) & \text{otherwise} \end{cases} \quad (4)$$

r_t is the reinforcement the robot receives when it executes action $\pi(s_t)$ in state s_t, T is the control period of the robot. If we combine Eq. 3 and Eq. 4, it is true to say:

$$Q^{\pi}_{t+1}(s_t) = \begin{cases} -e^{-1/50} & \text{if } r_t < 0 \\ Q^{\pi}_t(s_t) + \delta & \text{otherwise} \end{cases} \quad (5)$$

where,

$$\delta = \beta(e^{\frac{-1}{50}} * Q^{\pi}(s_{t+1}) - Q^{\pi}(s_t)). \quad (6)$$

$\beta \in [0, 1]$ is a learning rate.

An iterative and straightforward process applying Eq. 5 will make it possible to obtain the utility values $Q^{\pi}(s)$. Basically, the robot begins with an initial set of random values, $Q^{\pi}(s) \in [-1, -0.95]$, $\forall s$, and then it initiates a exploration of its environment executing the control policy π. As the robot moves performing the control policy π, it continually makes predictions about when it will receive negative reinforcements, in such a way that later comparisons of the predictions and the rewards the robot actually received will allow the updating of the utility values $Q^{\pi}(s)$.

IV. Achieving fast learning procedures

Since the I_Tbf algorithm is able to predict when a robot mistake will occur, it would be possible to iterate a control policy in an attempt to *increase the time before a robot failure*. We already did this experiment in the past with nice results [8], but the learning procedures were still too slow to be applied in real robots. We need robots that are able to learn in a short period of time. Due to this, instead of building a learning system that needs to determine the suitable action for every state of the robot, we prefer to build an ensemble of policies that determine, each one of them, the interval of actions most suitable for each state of the robot, (Figure 1, *ensemble of decision policies*). This ensemble will use a voting mechanism

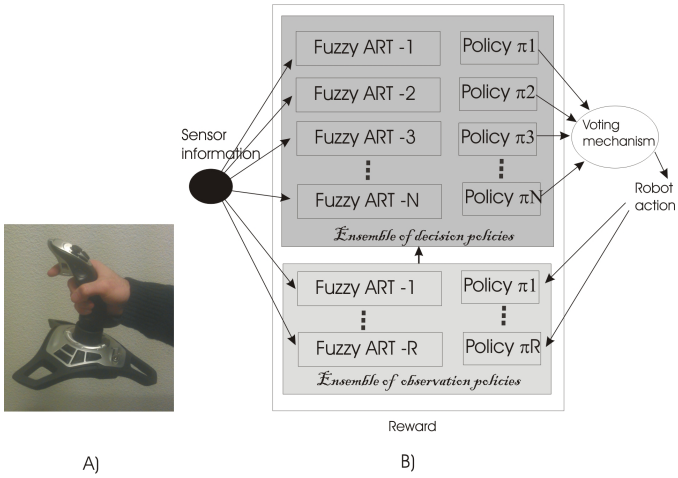

Fig. 1. a) Wireless joystick used by the human observer to provide punishment to the robot. B) General scheme of our proposal to get fast and continuous learning procedures. The *ensemble of decision policies* determine what the robot does at every instant.

to decide the action to be executed by the robot at every instant. Each policy of the ensemble π_l, is a mapping between world states and intervals of actions:

$$\begin{array}{lcl} \pi^l : S^l & \rightarrow & A \\ s \in S^l & \rightarrow & \pi^l(s) \in A = [a, b), a \in A, b \in A, a < b \end{array} \quad (7)$$

There are two important aspects that are necessary to consider: each policy of the ensemble is built randomly, i.e. the interval of actions that each policy π^l suggests for every state $s \in S^l$ is determined randomly. On the other hand, each policy builds its own representation of the world around the robot using a Fuzzy ART Network, S^l, $\forall l = 1, ..., N$. Therefore, there will be N Fuzzy ART Networks working in parallel (Figure 1). The use of the Fuzzy ART will be explained later in this paper.

A. Evaluating the policies of the ensemble

The robot will use the ensemble of policies to determine the action that needs to be carried out at every instant, but these policies have been built randomly. Therefore, we will need to evaluate the suitability of each l policy of the ensemble to the task being learned by the robot. To do this, we will use the I_Tbf algorithm described in the previous section, and the utility functions to represent how long each policy will be able to move the robot without making a mistake. Therefore, we will use a set of N utility functions $Q_1, ..., Q_l, ...Q_N$ to evaluate the policies of the ensemble.

As it was described in the previous section, the robot begins with an initial set of random values, $Q^l(s) \in [-1, -0.95]$, $\forall s \in S^l$, and then it initiates a exploration of its environment executing the actions determined by a voting procedure. As the robot moves the Q-values corresponding the different policies are updated. Nevertheless, in this case the policy used to generate the *behaviour* of the robot will not necessarily be the policy that is being evaluated. In consequence the definition of the utility function is modified slightly, now $Q^l(s)$ will be the expected time interval before a robot

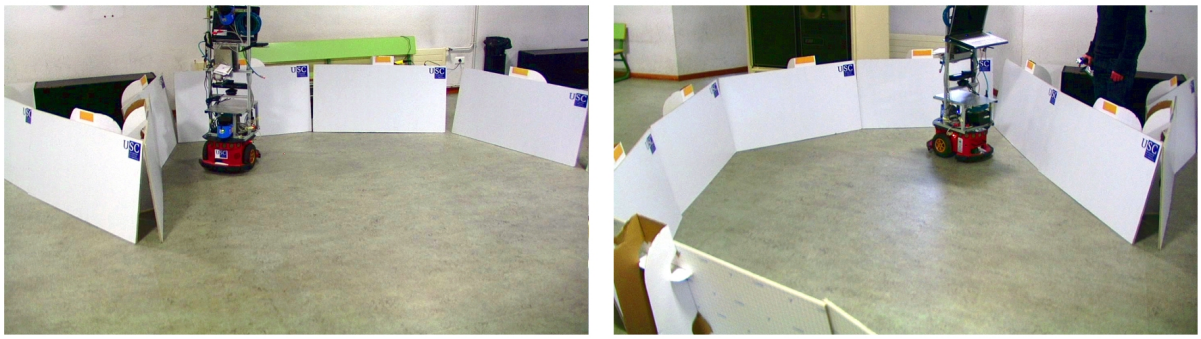

Fig. 2. In the first experiment the robot will learn to follow the wall on its right side. The movement of the robot was confined in a small area.

failure when the robot starts moving in s, performs $\pi^l(s)$, and follows an optimal policy thereafter. On the other hand Eq. 5 describes the simplest way of learning the Q-values. Nevertheless, when rewards occur infrequently, the learning process can take too long. One option for speeding up this process is by adding more *memory* into the system. In our case, the robot will move collecting data that will later be used to update the Q-values. This updating is done according to the algorithm described below:

1) $m = 0$
2) Observe the current state in every learner and store that information in a *experience set*: $s[m] = \{s_t^1, , s_t^l, , s_t^N\}$, so that $s[m,1] = s_t^1, s[m,l] = s_t^l, s[m,N] = s_t^N$
3) Perform the action a_t that seems to be the best according to the ensemble: $a[m] = a_t$
4) Obtain the *estimated time before failure* for this state according to the ensemble:

$$u[m-1] = -50\ T\ ln\left(-\frac{\sum_{l=1}^{l=N}\delta(a_t \in \pi^l(s_t^l)).Q^l(s_t^l)}{\sum_{l=1}^{l=N}\delta(a_t \in \pi^l(s_t^l))}\right)$$

where δ is the Kronecker delta, i.e., if a_t is within the interval determined by $\pi^l(s_t^l)$ then $\delta(a_t \in \pi^l(s_t^l)) = 1$, being zero otherwise.
5) After performing a_t observe the new state s_{t+1}^1, $s_{t+1}^2 ... s_{t+1}^N$ and the reinforcement value r_t, $r[m] = r_t$,
6) Shift the m-index: $m \leftarrow m + 1$
7) **If $r_t = 0$ and $m < M$**
   a) return to step 2
8) **If $r_t = 0$ and $m >= M$**
   a) Update *time before failure*:
      - for $k = m-1, m-2, \ldots, 1$ do:
        - if $k = m-1$ then $Tbf = u[k]$
        - else $Tbf \leftarrow \lambda_1(Tbf + T) + (1-\lambda_1)u[k]$.
   b) Update the Q-values of the first state in the *experience set*:
      $\Delta Q_t^l(s[0,l]) =$
      $\beta_1 \delta(a[0], \pi^l(s[0,l]))(-e^{-Tbf/50T} - Q_t^l(s[0,l]))$,
      $\forall l = 1, \ldots, N$
   c) Delete the information related with the fist state in the *experience set*, $(s[0], u[0], r[0])$, and shift the number of items in the experience set:$m \leftarrow m-1$
   d) return to step 2
9) **If $r_t < 0$**,
   a) for $j = 0, 1, \ldots, m-1$ do:
      i) Update *time before failure*:
         - for $k = m-1, m-2, \ldots, j$ do:
           - if $k = m-1$ then $Tbf = T$
           - else $Tbf \leftarrow \lambda_2(Tbf + T) + (1-\lambda_2)u[k]$.
      ii) Update the Q-values:
      $\Delta Q_t^l(s[j,l]) =$
      $\beta_2 \delta(a[j], \pi^l(s[j,l]))(-e^{-Tbf/50T} - Q_t^l(s[j,l]))$
      $\forall l = 1, \ldots, N$
      iii) Delete the information related with the j state in the *experience set*, $(s[j], u[j], r[j])$

Observing the previous algorithm we can distinguish three different situations:

- At the beginning the robot starts moving but it does not update the Q-values until it has moved for certain interval (M control cycles).
- After that period the robot keeps moving updating the Q-values of every state considering what has happened in the next M control cycles – this update corresponds to the step 8 of the algorithm.
- Finally, when the robot receives a negative reinforcement it updates the Q-values of those states stored in the experience set taking into account the actual time interval to the error – step 9 – and the process starts again.

It is important to notice that only those policies which voted for the action that the robot finally executed, see their Q-values changed. On the other hand, the expected time before a robot failure at each instant is approximated as the average value of the expected times (Q-values) for all policies. It is also important to be aware of the existence of two sets of parameters $\{\beta_1, \lambda_1\}$ and $\{\beta_2, \lambda_2\}$. This is due to the fact that the negative reinforcements are infrequent and therefore its influence in the Q-values must be higher than in those cases in which the reinforcement is null.

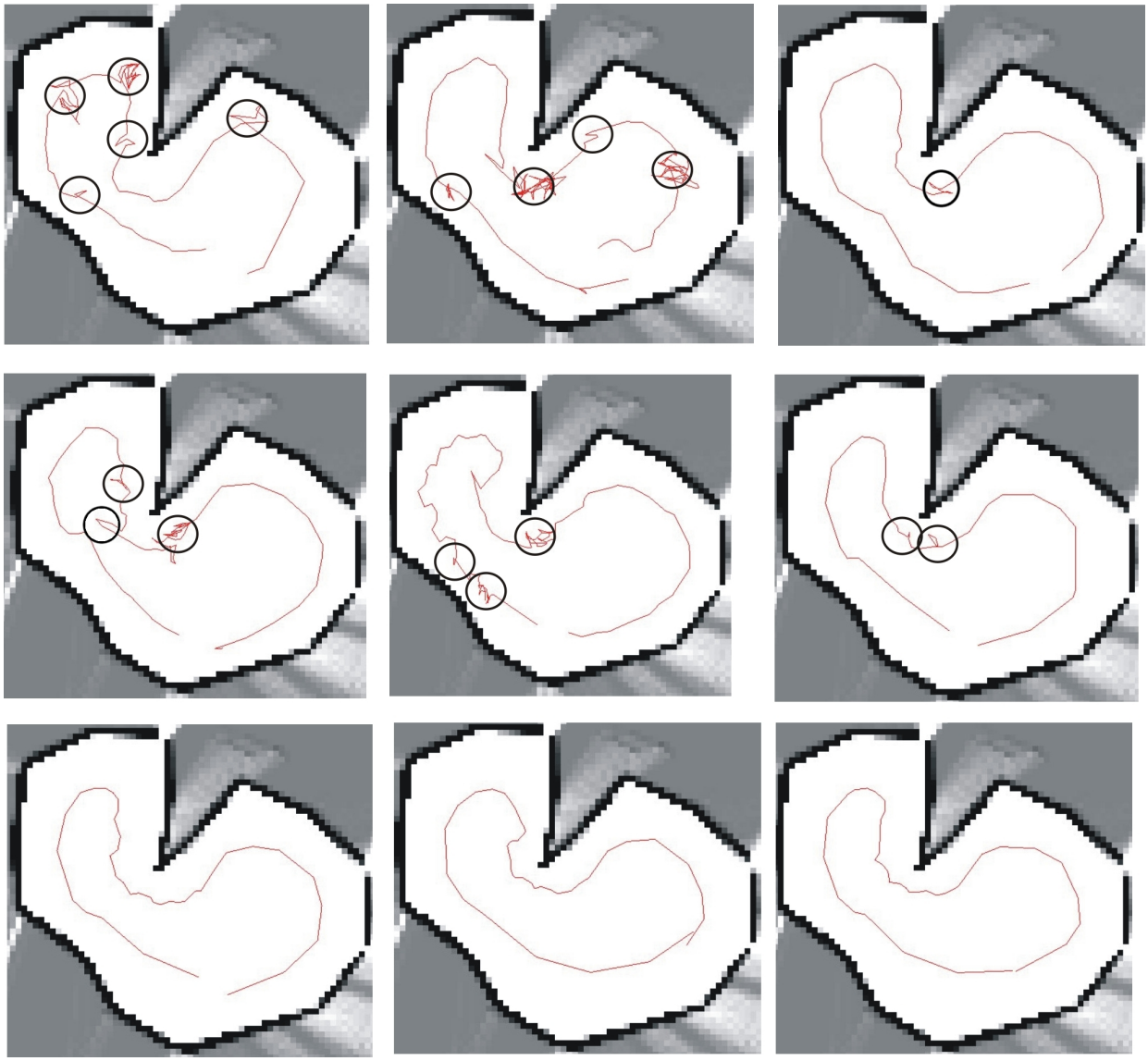

Fig. 3. First laps of the experiment where the robot had to learn a wall following behavior in the environment shown in Figure 2. The continuous line in the figure shows the trajectory of the robot. The circles show the areas where the robot received negative reinforcement from the human user. After 6 laps the robot is capable of performing the task without errors.

B. Voting procedure

The action the robot executes at each instant is the one that seems to be the optimum according to what the robot knows so far, i.e., the robot will perform the action with the highest average Q-value:

$$a_t = arg\_max_{a \in A} \frac{\sum_{l=1}^{l=N} \delta(a \in \pi^l(s_t^l)).(1 + Q^l(s_t^l)}{\sum_{l=1}^{l=N} \delta(a \in \pi^l(s_t^l))} \quad (8)$$

From the previous equation we can observe that each policy l votes for the actions suggested by the corresponding policy π^l, this vote will be weighted by the Q^l value.

C. Incorporating new knowledge

Our system is still unable to get rid of unsuitable policies and to include new ones. Because of that we decided to incorporate a second pool of policies (Figure 1), we call this second pool *ensemble of observation policies*. Every time the robot reaches a certain amount of mistakes the oldest policy in the observation pool is transferred to the *decision pool*. The policies in the observation ensemble will use the algorithm described below to observe the sensor readings and the action performed by the robot at each instant. They will use this information to build their state representation and to focus the the attention on those actions that seem to be right for every state.

1) Observe the action selected by the policies in the *decision pool*, a_t
2) Determine the current state for every learner in the *observation pool*, $s^l(t)$, $\forall l = 1, ..., R$
3) If $s^l(t)$ is new, initialize the control policy: $\pi^l(t) = [a, b]$, where $a = a_t$ and $b = a_t$
4) if $s^l(t)$ is not new update the control policy:

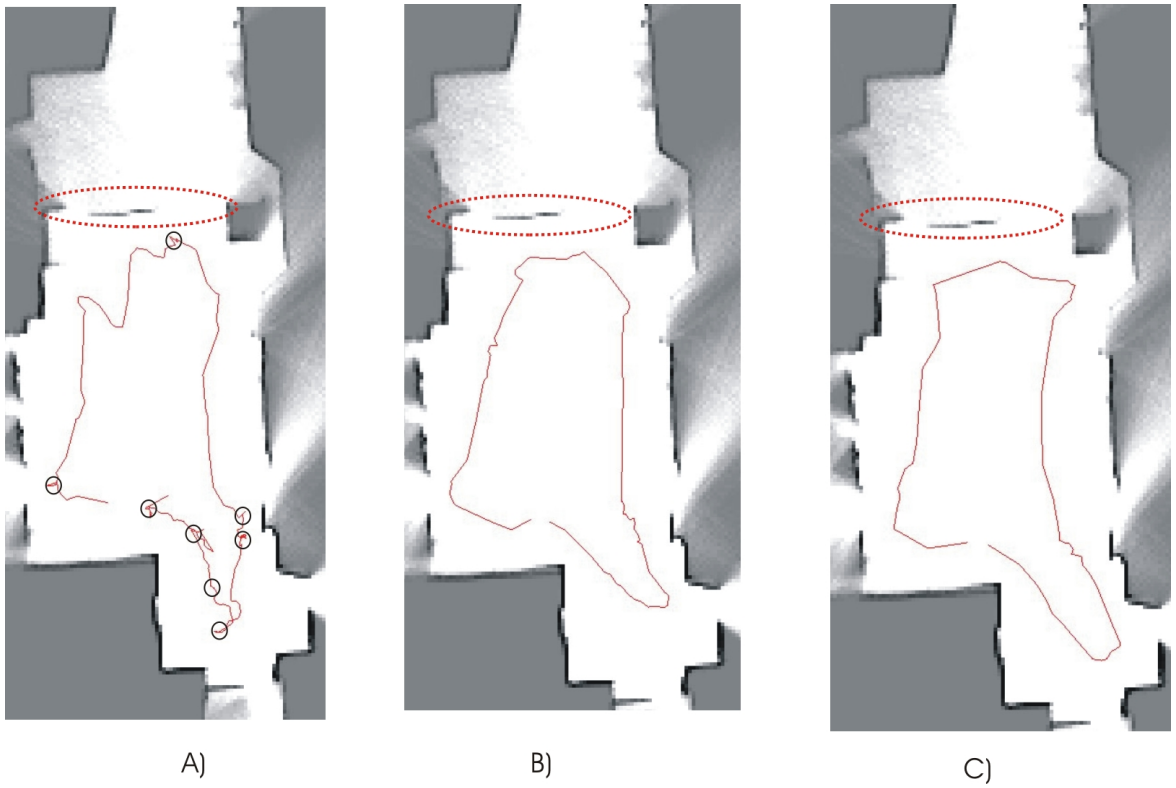

Fig. 4. First part of the experiment carried out to teach a robot how to navigate in the Department of Electronics and Computer Science. In this first part of the experiment the movement of the robot was confined to the hall of the department shown in Figure 2 (to limit the movement of the robot carton boxes were placed in the area rounded with a dashed line). The continuous line in the figure shows the trajectory of the robot. The circles show the areas where the robot received negative reinforcements from the human user. The robot only received negative reinforcements in the first lap.

$$\text{if } a_t \notin \pi^l(s_t^l) = [a, b), \text{ shift the interval to include } a_t$$

$$a_t \begin{cases} a = a_t & \text{if } a_t < a \\ b = a_t & \text{if } a_t > b \end{cases}$$

V. Dynamic creation of the state space

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 1, 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 the Fuzzy Adaptive Resonance Theory (Fuzzy ART [7], [8]) to build the state representation for each learner. We have chosen the Fuzzy ART because this artificial neural network is able to perform an unsupervised online clustering of the robot sensor readings into a finite number of distinguishable situations that we call *states*. Another advantage of the Fuzzy ART is that those states that represent situations which rarely appears during the robot lifetime are not shadowed by more common states. Therefore, the Fuzzy ART 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, while preserving the representation of situations visited earlier during the learning.

Basically the Fuzzy ART will divide the sensor space into a set of regions. Each one of these regions will have a representative or prototype representing it. The Fuzzy ART 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 comprise the information provided by a laser rangefinder and sonar sensors, but other sources of information are valid (e.g. gray 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]: $w_j = (w_{j1}, \cdots, w_{jM})$. We shall use the letter N to refer to the number of states learnt by the network so far.

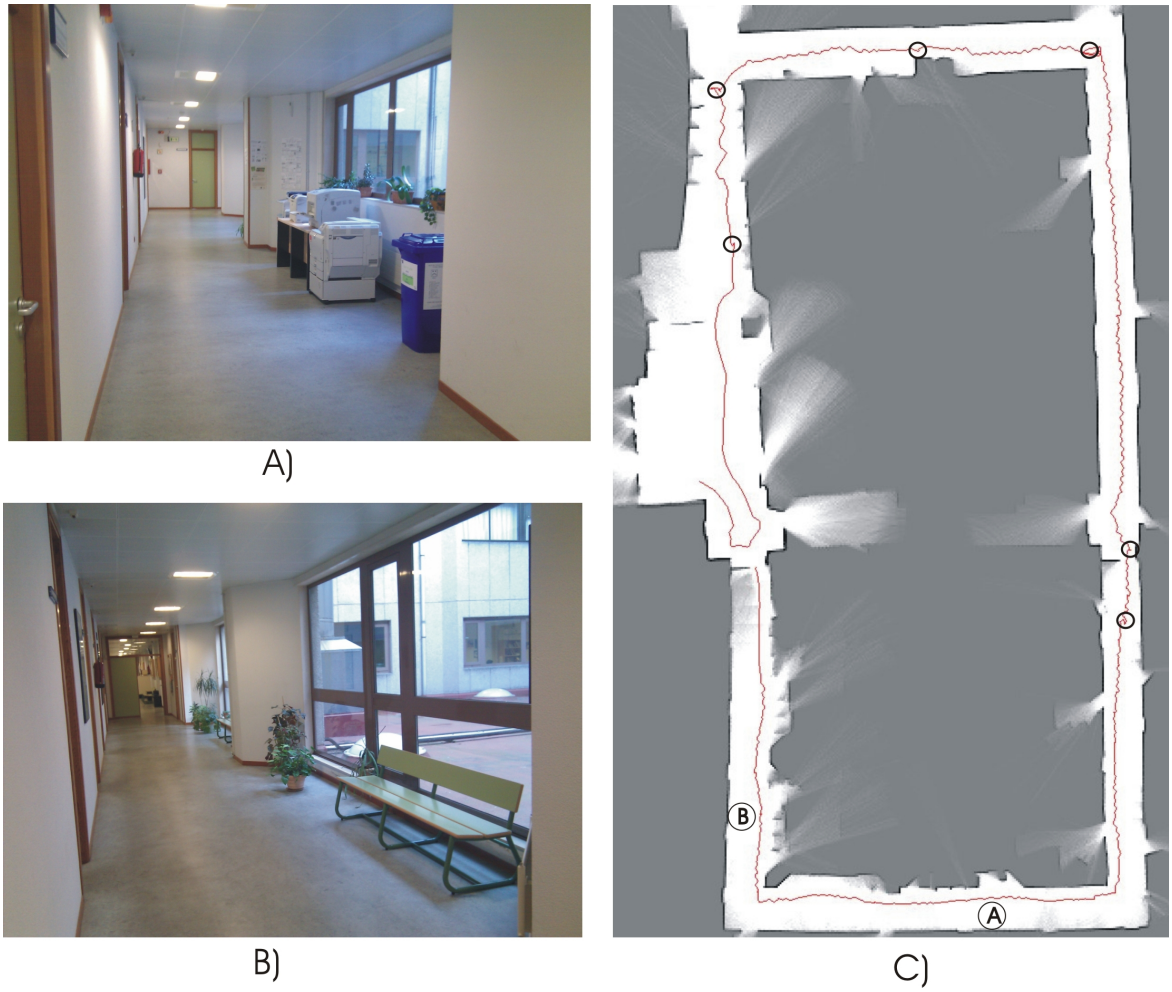

Fig. 5. Second part of the experiment shown in Figure 4. Once the robot completed the three laps in the hall of the department it was freed and allowed to move along the rest of the Department. The continuous line show the trajectory of the robot and the circles point out areas where the user provided negative reinforcement. We can see how very little reinforcement was provided despite the fact that the robot encountered completely new corridors with benches and plants (A), or even corridors with copying machines and printers (B). Most of the negative reinforcements were due to the fact that the robot had to traverse doors.

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 summarized in the following steps (there are some important differences in comparison with the general proposal described in [7]):

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

$$T_j(\mathbf{I}) = \frac{|\mathbf{I} \wedge w_j|}{|w_j|} \tag{9}$$

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

$$|x| \equiv \sum_{i=1}^{M} |x_i|. \tag{10}$$

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

$$J = arg\_max_j\{T_j : j = 1...N\}. \tag{11}$$

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:

$$|\mathbf{I} \wedge w_J| \geq \rho|\mathbf{I}| \tag{12}$$

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:

$$w_J^{(new)} = \beta\mathbf{I} + (1 - \beta)w_J^{(old)}. \tag{13}$$

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

$$|\mathbf{I}| \equiv \gamma, \forall \mathbf{I}, \gamma > 0 \tag{14}$$

Fig. 6. These pictures show the Department of Electronics and Computer Science where the robot will learn how to follow the wall situated on its right. It contains corridors (A) and open spaces (B).

Fig. 7. Trajectory followed by a robot along the first two laps of the environment shown in Figure 6. The robot starts with no prior knowledge. After the first lap the robot almost receives no negative reinforcement (marked as circles).

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

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) \equiv (a_1, ..., a_M, a_1^c, ..., a_M^c), \tag{15}$$

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 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. This helps to get a better generalization during the learning process.

Other artificial neural networks, such as the *Echo State Networks* [12] 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.

VI. Experimental results

We have implemented our learning proposal on a Pioneer 3DX robot. This robot is equipped with a SICK LMS-100 laser scanner, 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. 2). The set of actions were the angular velocities in $[-0.8, 0.8]$ rads/s. $\lambda 1 = 0.99$, $\beta 1 = 0.05$, $\lambda 2 = 0.15$, $\beta 2 = 0.95$. The sensory input of the system is a vector containing 541 laser readings – one reading every half degree in $[-135°, 135°]$ – and the measurements provided by the 16 ultrasound sensors.

We present here several experiments performed with a real robot in real environments. In these experiments the robot had to learn from scratch how to follow the wall located on its right. The robot starts each experiment with no prior knowledge about the environment or the actions it ought to take. The initial Q-values are selected randomly in the range $[-1, -0.95]$, hence the first actions performed by the robot will be random. Figures 2 and 3 show the results we got in a first experiment in which the robot moved along a confined space. As we can observe in Figure 3, most of the errors were committed in the first laps of the environment.

Figures 4 and 5 show the results we got in the second experiment in which the robot had to learn from scratch how to navigate along a real environment. We divided this experiment in two parts: In the first part we confined the movement of the robot to the hall of the Department of Electronics and Computer Science of the University of Santiago de Compostela, Spain. As we can observe in Figure 4 the robot learnt very fast how to move in this area. Then, on a second part we removed the boxes that were limiting the movement of the robot and we allowed the robot to move along the whole department (Figure 5). It is important to notice that despite of the fact that the robot had to traverse complete new areas with corridors, doors, and a very different disposition of obstacles, it received very little punishment. It is also important to be aware of the fact that the robot was learning continuously, even in absence of negative reinforcements.

On the third experiment the robot started the learning in a corridor of the Department (Figures 6 and 7). In this experiment we tested again the capability of learning from scratch, but in this case the robot was never confined in a restricted area. The system behaves as expected, and after committing most of the errors on the first lap the robot is able to move safely.

VII. Conclusions

A truly useful personal robot must have the ability to learn form its own interactions with the physical environment. The robots must be able to adapt to changing conditions. These changes can be on the environment or on the own robot. Most reinforcement learning research has been made in simulation because real-environments require large computation costs as well as a lot of time. In this article we describe a strategy able to achieve continuous learning procedures on real robots that interact with the environment. Continual learning allows the robot to face and adapt to unexpected situations.

With the strategy described in this paper a robot is able to: first, learn to perform a task starting from scratch; and second, to incorporate new knowledge at any time and thus correct its own behaviour. Combining several learner agents our system is fast and stable. Each learner dynamically creates its own representation of the environment using a Fuzzy ART neural network.

The experimental results we achieved so far confirm that we are moving in the *right direction*, since the learning procedures were fast, reliable, and continuous. Finally, our proposal is able to incorporate real-time human feedback (any person can use a wireless joystick to punish the robot whenever it does something wrong). Despite the highly undeterministic character of the reinforcement provided in this way, our robot is still able to learn the desired behaviour in a short period of time.

Acknowledgments

This work was supported by the research grant TIN2009-07737 of the Spanish Ministerio de Economa y Competitividad, and María Barbeito program of the Xunta de Galicia.

References

[1] R. S. Sutton, *Reinforcement learning: An introduction*, MIT Press, 1998
[2] M.A. Bozarth, *Pleasure: The politics and the reality*, Springer Netherlands, pp.5–14,1994.
[3] E.L. Thorndike, *Animal Intelligence*,Hafner, Darien, CT, 1911.
[4] A. L. Thomaz and C. Breazeal, *Teachable robots: Understanding human teaching behaviour to build more effective robot learners*, Artificial Intelligence, volume 172, pages: 716–737,2008.
[5] A. L. Thomaz, G. Hoffman and C. Breazeal, *Reinforcement learning with human teachers: Understanding how people want to teach robots*, in Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 2006.
[6] L.P. Kaelbling, M.L. Littman and A.W. Moore, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research, 4:237–285, 1996.
[7] G. A. Carpenter, S. Grossberg and 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.
[8] Pablo Quintia, Roberto Iglesias, Carlos V. Regueiro and Miguel A. Rodriguez, *Simultaneous learning of perception and action in mobile robots*, Robotics and Autonomous Systems, vol 58, pages: 1306–1315, 2010.
[9] W. Bradley Knox and Peter Stone, *Interactively shaping agents via human reinforcement: The TAMER framework*, in Fifth International Conference on Knowledge Capture, California (USA), September 2009.
[10] W. Bradley Knox and Peter Stone, *Reinforcement learning with human feedback in mountain car*, in AAAI Spring 2011 Symposium on Bridging the Gaps in Human-Agent Collaboration, 2011.
[11] Andrew Y. Ng and Stuart Russell, *Algorithms for inverse reinforcement learning*, in Proceedings of the Seventeenth International Conference on Machine Learning, 2000.
[12] Maass W., Natschlaeger T. and Markram H. *Real-time computing without stable states: A new framework for neural computation based on perturbations*, in Neural Computation, 14(11):2531-2560, 2002.
[13] Brenna D. Argall, Sonia Chernova, Manuela Veloso and Brett Browning. *A survey of robot learning from demonstration*, in Robotics and Autonomous Systems, 57(5):469-483, 2009.
[14] Andrea Lockerd Thomaz, Guy Hoffman and Cynthia Breazeal. *Real-time interactive reinforcement learning for robots*, in AAAI Workshop on Human Comprehensible Machine Learning, 2005.
[15] T. Kollar and N. Roy. *Using reinforcement learning to improve exploration trajectories for error minimization*, in IEEE International Conference on Robotics and Automation, 3338-3343, may 2006.