An approach to the design of reinforcement functions in real world, agent-based applications

Andrea Bonarini, Claudio Bonacina, Matteo Matteucci

Abstract— **The success of any reinforcement learning (RL) application is in large part due to the design of an appropriate reinforcement function. A methodological framework to support the design of reinforcement functions has not been defined yet, and this critical and often underestimated activity is left to the ability of the RL application designer. We propose an approach to support reinforcement function design in RL applications concerning learning behaviors for autonomous agents. We define some dimensions along which we can describe reinforcement functions; we consider the distribution of reinforcement values, their coherence and their matching with the designer's perspective. We give hints to define measures that objectively describe the reinforcement function; we discuss the trade-offs that should be considered to improve learning and we introduce the dimensions along which this improvement can be expected. The approach we are presenting is general enough to be adopted in a large number of RL projects. We show how to apply it in the design of Learning Classifier Systems (LCS) applications. We consider a simple, but quite complete case study in evolutionary robotics, and we discuss reinforcement function design issues in this sample context.**

I. Introduction

The interaction of an autonomous agent with the environment is its major source of knowledge. One of the most interesting computational approaches to increase knowledge while interacting with the environment is *reinforcement learning* (RL) [1], [2]. In this paper, we present an approach to design and evaluate reinforcement functions used by RL algorithms to produce control systems for real-world and simulated, robotic, autonomous agents.

In RL, an agent operates in an environment by performing actions and receives from the environment data that describe, maybe only partially, the state of the agent-environment system. A subset of these data is considered as input to the control system (*control input* or $\mathbf{I^C}$) of the agent which, on the basis of this perception, produces the next action to be sent to the actuators. Another subset of the data (*reinforcement input* or $\mathbf{I^R}$) perceived from the environment, maybe partially overlapping with the first, can be used by a RL system to evaluate the performance of the agent. Using these data, the RL algorithm may modify the control system to improve the agent's performance. Reinforcement is computed from the reinforcement input by means of a *reinforcement function* (more generally a *reinforcement program*). The aim of the learning system is to improve the controller so as to maximize some function of the reinforcement, e.g., its instantaneous value, or the expected sum of its values, or its average. Usually, to reduce the search space, the real-valued (or fine-grained) input variables are discretized in a small number of intervals. Our approach supports the definition and the analysis of reinforcement functions in this situation. In figure 1, we show a schematic view of the components of the described system.

Politecnico di Milano AI and Robotics Project. Dipartimento di Elettronica e Informazione – Politecnico di Milano. Piazza Leonardo da Vinci, 32 – 20133 Milano – Italy – E-mail: {bonarini, matteucc}@elet.polimi.it, c2-bonacina@uwe.ac.uk – http://www.elet.polimi.it/people/bonarini

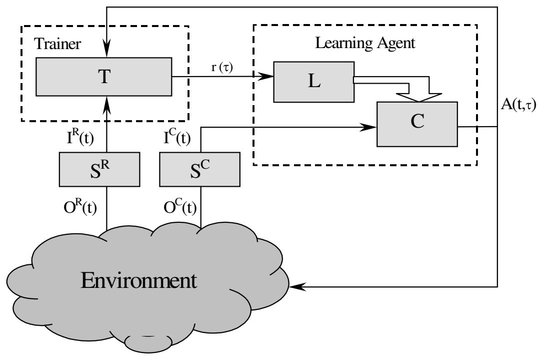

Fig. 1. A schema of the agent-environment system

The primary role of the reinforcement function is to provide the RL algorithm with enough and correct information about the task to be learned and the performance of the agent. The designer should define the reinforcement function by considering knowledge about the task that he wishes the agent to learn. Translating the designer knowledge from natural language to a formal model, which is usually represented by a mathematical formalism, can be a complex task. Our approach supports it. We suggest linking all the design aspects to an objective model of the agent-environment system.

In most of the applications reported in the RL literature, it seems that the reinforcement function design is approached by trial and error: a good designer sooner or later comes to the definition of a reinforcement function that makes the learning system obtain a controller for the agent that matches the specifications. We propose an approach to design reinforcement functions in an engineering perspective considering the configuration space of the agent (*CbD – c-space based Design*).

Although for some algorithms (Q-learning [3], TD(λ) [4]) there is a proof of convergence (at least under some assumptions), the performance of RL algorithms strongly depends on aspects (such as the definition of the reinforcement function), that are usually underestimated. Here, we present an analysis of the implications of some reinforcement function features on the learning process.

In the next section, we introduce a formalism to describe the agent-environment system from a learning point of view, and a formal description of the goals to be learned. In section III we consider design problems and reinforcement function features discussing their influence on the learning activity. We present our solution to some open problems in section IV. Then, we will show the application of our approach to a particular class of RL algorithms: learning classifier systems [5] [6]. We will also discuss the relationship between perception and reinforcement function design, and we report the results of our experiments in section V. Throughout the paper we will give examples concerning a classical application (a mobile robot navigating in an unknown environment), which we have designed to highlight most of the aspects that may characterize reinforcement learning experiments.

II. The agent-environment system

In this section, we define all the elements of the agent-environment system (see figure 1). The modules composing this system are dynamic systems, so they depend on time t; some of them change their output at discrete time points τ, when particular events occur. We consider the sequence of time points τ as a second time line overlapping with the primary time line t. We use the notation $\overline{\tau} \prec \overline{t}$ to express that a particular instant $\overline{\tau}$ in time line τ is strictly before $\overline{t}$ in t; $\overline{\tau} \preceq \overline{t}$ means that $\overline{\tau}$ is before or equal to $\overline{t}$. By $\overline{t}^{\preceq}$ we represent the set of time points previous or contemporaneous to $\overline{t}$; $\overline{t}^{\preceq}$ can be read as "until instant $\overline{t}$", and $\overline{t}^{\prec}$ as "before instant $\overline{t}$".

We consider the control program as a function depending on a set of parameters which may be updated by the learning system. The control system C uses the control input $\mathbf{I^C}(t^{\preceq})$ to generate the action $A(t,\tau)$ sent to the actuators to be executed. Here, t represents the control time and τ the time when the control parameters $\mathbf{P}(\tau)$ are updated by the learning system. For instance, vector $\mathbf{P}$ may represent the parameters of a PID controller [7], the weights of a neural network [8], or the Q-table in a Q-learning algorithm [3].

During learning, function C depends on vector $\mathbf{P}(\tau)$ so that $A(t,\tau)$ is defined as:

$$A(t,\tau) = C(\mathbf{I^C}(t^{\preceq}); \mathbf{P}(\tau)) \qquad \tau \preceq t, \tag{1}$$

where the action depends on the $\mathbf{I^C}$ values collected until the present time, and the last control parameters produced by the RL algorithm. An agent using this kind of control program is classified as *dynamic* [9], since at time t it uses internal states as a memory of $\mathbf{I^C}(t^{\prec})$.

To define a purely *reactive* agent we have to consider a different formulation for $A(t,\tau)$:

$$A(t,\tau) = C(\mathbf{I^C}(t); \mathbf{P}(\tau)) \qquad \tau \preceq t, \tag{2}$$

where the control action is decided considering only the present $\mathbf{I^C}(t)$.

The learning algorithm L is a function that updates the $\mathbf{P}$ vector when particular events occur; this function may consider some of the previous values of $\mathbf{P}$ and the reinforcement value $r(\tau)$ generated by the trainer

$$\mathbf{P}(\tau) = L(\mathbf{P}(\tau^{\prec}); r(\tau)) \tag{3}$$

The reinforcement function T generates $r(\tau)$ at time τ. This function, in its most general form, ranges on reinforcement input $\mathbf{I^R}(t^{\preceq})$, with $t \preceq \tau$, and the actions $A(t^{\prec}, \tau - 1)$ performed by the agent before time t:

$$r(\tau) = T(\mathbf{I^R}(t^{\preceq}); A(t^{\prec}, \tau - 1)) \qquad t \preceq \tau. \tag{4}$$

In formulae 1, 2, 4 we use vectors $\mathbf{I^C}$ and $\mathbf{I^R}$ to define respectively the input to the control system and the trainer. These vectors represent the perception of the environment for the agent and the trainer; they are generated by the associated sensing and interpretation systems, respectively $\mathbf{S^C}$ and $\mathbf{S^R}$. We consider each of these systems as a set of functions $S_i^k(t)$ that at time t produce the input I_i^k from data $\mathbf{O^k}(t^{\preceq})$, observed from the environment until that time, and each concerning a specific feature.

$$I_i^C(t) = S_i^C(\mathbf{O^C}(t^{\preceq})) \qquad |\mathbf{I_i^C}| = m, \tag{5}$$

$$I_i^R(t) = S_i^R(\mathbf{O^R}(t^{\preceq})) \qquad |\mathbf{I_i^R}| = n, \tag{6}$$

where m and n are respectively the cardinality of $\mathbf{I_i^C}$ and $\mathbf{I_i^R}$.

Notice that, in general, S_i^k provide an interpretation model of data (e.g., their discretization), and $\mathbf{O^k}(t)$ come from a stochastic process implemented by physical sensors interfaced to the environment. This means that the input to our system may be different from the same data observed by another system (for instance, the designer) by another sensorial apparatus. This difference may produce problems that we discuss in section III-D.

After the learning process is terminated, the behavior of the agent is defined by a control program $\overline{C}$, whose vector $\overline{\mathbf{P}}$ is fixed:

$$A(t) = \overline{C}(\mathbf{I^C}(t^{\preceq}); \overline{\mathbf{P}}) \qquad (\overline{\mathbf{P}} = \mathbf{P}(\overline{\tau})) \wedge (\overline{\tau} \prec t). \tag{7}$$

We denote a generic sequence of actions implemented by $\overline{C}$ as $\overline{\mathbf{C^T}}$.

Considering the agent-environment system, this is completely determined from the physical point of view by the set of its free physical variables $\mathcal{V}$. A *configuration* c is a tuple of values for all the variables in $\mathcal{V}$, and $\mathcal{C}$ is the set of all the possible configurations of the system. The space $\mathcal{C}$ is known as the *configuration space*, or *c-space* [10] and it is used in robotics for several issues like, for example, the definition of the kinematic model of a robot. It is described by independent variables and physical constraints on their values; thus, it is an objective description of the agent-environment system.

We call the set of *goal configurations* $\mathcal{G}$. It is a subset of $\mathcal{C}$:

$$\mathcal{G} \subseteq \mathcal{C}. \tag{8}$$

The agent should learn a *goal behavior GOAL* that may belong to different typologies (*achieve, maintain, avoid*) or a combination of these (e.g., *achieve and maintain*):

$$GOAL = \begin{cases} avoid \quad \mathcal{G} \\ achieve \quad \mathcal{G} \\ maintain \quad \mathcal{G} \end{cases} \tag{9}$$

These typologies are defined as

$$GOAL\ achieve : \forall\, c_0 \in \mathcal{C}, \exists\, \tilde{g} \in \mathcal{G} :\ c_0\, \overline{\mathbf{C^T}}\, \tilde{g}, \tag{10}$$

i.e., for any starting configuration, applying the control relation an undefined number of times, the agent-environment system reaches one of the goal configurations;

$$GOAL\ maintain : \forall\, g_0 \in \mathcal{G}, \exists\, \tilde{g} \in \mathcal{G} :\ g_0\, \overline{\mathbf{C}}\, \tilde{g}, \tag{11}$$

i.e., starting from any of the goal configurations, applying once the control relation the agent-environment system goes to a goal configuration;

$$GOAL\ avoid : \forall\, c_0 \in \mathcal{C}, \nexists\, \tilde{g} \in \mathcal{G} :\ c_0\, \overline{\mathbf{C^T}}\, \tilde{g}, \tag{12}$$

i.e., starting from any configuration and applying the control relation an undefined number of times, the agent does not reach any goal configuration. The *avoid* goal type could in principle be derived from the *achieve* by complementation of the argument, but it is more natural to define it in this way.

Notice that in many applications not all the configurations belong to $\mathcal{G}$, so we may want first achieve (or avoid) a goal configuration, and then maintain the system in the desired situation.

The definitions given above are ideal and in many real cases they should be relaxed, to enable the RL algorithm to produce the desired results. We propose two types of relaxation for both *achieve* and *maintain* goal types. The first *achieve* relaxation allows the goal configuration to be achieved from a subset $\mathcal{C}'$ of the configuration space, instead of from the whole $\mathcal{C}$.

$$\begin{aligned} GOAL\ achieve : \quad & \exists\, \mathcal{C}' \subseteq \mathcal{C}, \forall\, c_0 \in \mathcal{C}', \\ & \exists\, \tilde{g} \in \mathcal{G} :\ c_0\, \overline{\mathbf{C^T}}\, \tilde{g}. \end{aligned} \tag{13}$$

The second type of *achieve* relaxation allows to achieve a configuration in a bounded neighborhood of a goal configuration.

$$\begin{aligned} GOAL\ achieve : \quad & \forall\, c_0 \in \mathcal{C}, \exists\, \tilde{g} \in \mathcal{G}, \exists\, \tilde{c} \in \mathcal{C} : \\ & c_0\, \overline{\mathbf{C^T}}\, \tilde{c}, \wedge \parallel \tilde{g} - \tilde{c} \parallel < N. \end{aligned} \tag{14}$$

Also for the *maintain* goal type we can relax the generality of the control program by:

$$\begin{aligned} GOAL\ maintain : \quad & \exists\, \mathcal{G}' \subseteq \mathcal{G}, \forall\, g_0 \in \mathcal{G}', \\ & \exists\, \tilde{g} \in \mathcal{G} :\ g_0\, \overline{\mathbf{C}}\, \tilde{g}, \end{aligned} \tag{15}$$

and its accuracy by giving the possibility of leaving the goal configuration for a finite number of control steps N:

$$GOAL\ maintain : \forall\, g_0 \in \mathcal{G}, \exists\, \tilde{g} \in \mathcal{G} :\ g_0\, \overline{\mathbf{C^N}}\, \tilde{g}. \tag{16}$$

These definitions concern only simple attitudes towards goal configurations, and have to be extended to take into account goal behaviors involving temporal relationships with goal configurations, such as: achieving a set of configurations in a given sequence (e.g., "first push the green button, then the red one"), achieving a set of configurations if a particular event occurred in the past (e.g., "reward coffee distribution only if it was requested" [11]), alternating achieving and avoiding ("go to the light, but, when there is a sound, escape from it" [12]). In this paper we focus on simple atemporal attitudes; a complete treatment of goal definition is left to future work.

III. Design problems

In this section, we present some of the reinforcement function design issues. We first discuss problems related to the different perception of the state of the agent–environment system that the designer, the trainer, and the agent itself may have. Then, we discuss the possibility to define reinforcement functions that are able to represent all the features of the desired behavior. This is followed by a discussion about temporal reinforcement distribution. All these issues raise the need for a careful design of data acquisition systems and of the reinforcement function. In this paper we focus only on this last aspect, presenting our approach in section IV and its application in section V. Data acquisition and interpretation issues will be treated in details in a forthcoming paper.

Let us now start with the presentation of open problems.

A. The designer

Let us consider that the designer has clearly in mind what the agent should learn, so that natural language specifications are available. We call $\mathbf{O^O}$ the data that can be objectively observed by a person looking at the performing agent, and that are considered in the natural language specifications. Although in principle it would be possible to have a human trainer providing reinforcement, this is not usual nor practical. In most applications the learning system perceives the reinforcement input from the environment, via the sensorial interface $\mathbf{S^R}$. Then, it computes the reinforcement without any human intervention via the reinforcement function. The main problem with the definition of the reinforcement function is that it has to derived from data available during the learning trials ($\mathbf{O^R}$), which are different from $\mathbf{O^O}$. This difference is the first relevant aspect of the designer problem; also when data are conceptually the same (e.g., a distance from a given object, or a color) the different nature of the sensors provide different perceptions.

The first activity that the designer should face is to define a relationship between the original natural language specification, which considers $\mathbf{O^O}$, and the reinforcement

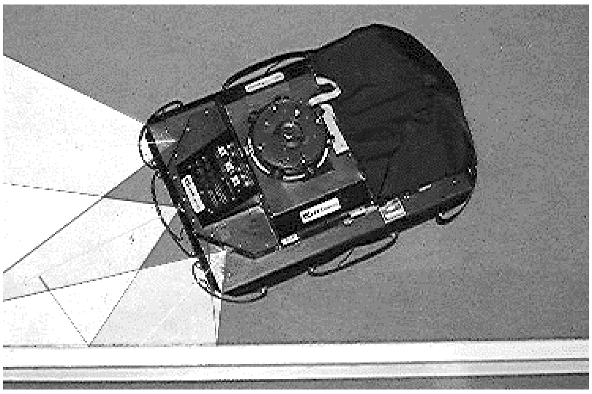

Fig. 2. A robotic agent close to a wall, but not perceiving it. The light gray cones represent the sonar wave.

function, which should consider the only input available to the trainer $\mathbf{O^R}$.

Let us introduce a simple example where a mobile robot should learn "to navigate in the middle of a corridor". The RL designer may decide to give a positive reinforcement to the agent when it is moving in the middle of the corridor. To implement this kind of reinforcement function we may put sensors on the robot that can perceive the distance from both the walls, thus making $\mathbf{O^R}$ matching $\mathbf{O^O}$, or, more realistically, making $\mathbf{I^R}$ matching $\mathbf{I^O}$, where $\mathbf{I^O}$ is the conceptual input to the observer, coming from its sensorial apparatus.

If we cannot modify physical aspects of the robot, we have to relate the task description with the available data. Let us consider that the agent can measure distances from objects in a 120 degrees range on the front of the robot, using a set of sonar sensors that produce $\mathbf{O^R}$. In this case, we may decide to give positive reinforcement when the agent does not perceive anything from its frontal sensors, which is a situation related with the original specifications when the corridor width is smaller than twice the sensor range. However, this may still introduce ambiguity. In a situation like the one shown in figure 2, the sensors cannot detect anything in front of the agent although it is very close to a wall, since the relative orientation between the sensor and the wall does not make any reflected ultrasonic wave that bounce back to the sensor. This problem of ambiguity arises from a mismatch between the desired feature (a distance in a given direction) and the available sensors. This type of problems can be easily solved when working in simulated environments but they are more problematic in real environments.

B. The Trainer

The next aspect to take into consideration concerns the trainer (see figure 1). The trainer, using $\mathbf{I^R}$, computes the reinforcement signal r via a reinforcement function T. The learning algorithm uses this signal to evolve a mapping from the control input $\mathbf{I^C}$ to the action A. Let us consider that the information provided by $\mathbf{I^C}$ is sufficient to implement the task. If it is not, the control input should be redefined, or the task definition modified to match the available information.

The different perception of the environment by the trainer and the control system can be classified according to the differences between $\mathbf{I^C}$ and $\mathbf{I^R}$. We have two classes of trainers: if $\mathbf{I^R} = \mathbf{I^C}$ the trainer is *internal*, otherwise it is *external* [9].

B.1 Internal trainer

$\mathbf{I^C}$ is often used as $\mathbf{I^R}$ due to two main motivations:

- *availability*: if the only interface with the environment is provided by $\mathbf{O^C}$, then we can exploit all the available information in the reinforcement function by considering $\mathbf{O^R} = \mathbf{O^C}$ and $\mathbf{I^R} = \mathbf{I^C}$
- *speed up*: considering $\mathbf{I^C} = \mathbf{I^R}$ (and $\mathbf{S^R} = \mathbf{S^C}$) may speed up and improve the learning process since the agent has not to find out the relationship between the input used to reinforce it ($\mathbf{I^R}$) and that used to control it ($\mathbf{I^C}$). Learning this relationship is needed as appears evident by substituting the expression of $r(\tau)$ from equation (4) in equation (3) and the so obtained value of $P(\tau)$ in equation (2) (or in (1)), obtaining that

$$A = C(\mathbf{I^C}; \mathbf{I^R}). \tag{17}$$

This means that A depends on the values of $\mathbf{I^R}$ during the learning trials, and that this leaves a trace on the learned controller.

Usually, learning the relationship between $\mathbf{I^C}$ and $\mathbf{I^R}$ is not considered an issue. Given that it may require computational resources and affect the quality of the learning process, we think it is worth being aware of it. We will see in section IV how to take it into consideration, and in section V what may happen if we do not consider this issue. We better clarify that we do not consider learning the relationship between $\mathbf{I^C}$ and $\mathbf{I^R}$ as a different learning process rather as an implicit mapping process due to the differences between $\mathbf{I^C}$ and $\mathbf{I^R}$.

The reinforcement function design considering $\mathbf{I^R} = \mathbf{I^C}$ may present some problems:

- *abstraction mismatch*: $\mathbf{I^C}$ may be at a too low abstraction level with respect to the designer's original natural language specifications
- *lack of information*: $\mathbf{I^C}$ may not provide enough information to the designer, so that it becomes hard to define the task solely with this information [12] [9].

B.2 External trainer

The main motivation to select $\mathbf{I^R} \neq \mathbf{I^C}$ concerns the possible low representation power of $\mathbf{I^C}$; we may want $\mathbf{I^R}$ different from $\mathbf{I^C}$ to represent some features not directly derivable from this [9]. This makes easier to represent the designer's expectations, although the agent should learn also the relationship between $\mathbf{I^R}$ and $\mathbf{I^C}$, as mentioned above.

This approach has some advantages, but introduces some issues, too:

• *relationship between* $\mathbf{I^R}$ *and* $\mathbf{I^C}$: the agent has to learn the relationship between $\mathbf{I^R}$ and $\mathbf{I^C}$, and this may not exist, or it may be too difficult to be learned
• *aliasing*: the presence of aliasing may make harder the learning process (see section III-D).

C. Immediate and delayed reinforcement

Immediate reinforcement can be provided after the execution of each action if it is possible to define a reinforcement function to estimate how much each single action in a given situation contributes to reach the goal. This means that $\tau \equiv t$ in equation (4) and that the goal configuration set $\mathcal{G}$ is extended to the whole configuration set $\mathcal{C}$. In this case the reinforcement function is a metric to estimate how each action contributes to the accomplishment of the task. Such kind of reinforcement functions provide a lot of information and, when available, they seem more appealing than those providing reinforcement only in specific situations (*delayed reinforcement*), where $\tau \neq t$ and $\mathcal{G} \subset \mathcal{C}$. However, immediate reinforcement may also introduce some problems. These reinforcement functions often include a lot of a priori knowledge [13] which in principle could bias the learning process in undesired and non optimal directions. In [13] and [14] the author recognizes this problem but underlines how it is important to provide continuous reinforcement (*progress estimator*) to speed up the learning process.

A typical problem with delayed reinforcement is its distribution over the configuration set $\mathcal{C}$. If it is too rare, then even the most suitable algorithms such as TD(λ) [15] [16] perform poorly, since for a large part of the trial the agent does not know whether its actions bring it towards the goal. The frequency of the reinforcement depends on at least three factors: the distribution of the reinforced configurations with respect to $\mathcal{C}$, the probability of being reinforced during the learning phase and the exploration policy. We consider them in our approach (section IV).

Let us discuss these issues by introducing a problem that we have faced in the past. In this problem the agent has to learn how to reach a moving target [17]. The control input variable is the difference between the agent's orientation with respect to the target, before and after having executed an action. The reinforcement is proportional to the reduction of the distance to the target in the direction where it was detected when the action was selected. This reinforcement function is effective only when the agent is faster than the target, since it produces a behavior such that the agent tends to align itself with the target's trajectory, in order to maximize the reinforcement. Unfortunately, if the target is faster, it can easily escape, since this agent is not really induced to reach it, but only to follow it. The designer has included a bias coming from the expectation that going in the direction of the target would enable the agent to reach it. In our experiment [17], the agent was able to learn to reach the target, even if the target was faster, when we adopted a less informative but also less ambiguous, reinforcement function, which provided delayed reinforcement only when the agent reaches the target. With slightly faster targets the agent was able to predict the target trajectory and take a more effective one, whereas with even faster targets the agent is able to decide to stop and wait for the target to come to it. However, in order to apply such a reinforcement function we had to design our trials according to the *LEM* (*Learning from Easy Missions* [18]) methodology, where the agent has to face increasingly complex tasks to learn incrementally the final behavior. By doing short experiments, it is possible to provide enough reinforcement.

D. Coherence and aliasing

Another problem in designing reinforcement functions concerns the coherence of the reinforcement function throughout the configuration space $\mathcal{C}$ as perceived by the agent. This is related to the aliasing problem, which arises when different configurations are perceived as the same. This may be due either to intrinsic partial observability of the environment or to misinterpretation of observations.

We have *perceptual aliasing* [19] when different configurations correspond to the same $\mathbf{I^C}$ values. In particular, the agent may not be able to observe enough data to distinguish among different configurations; in this case, $\mathbf{O^C}$ is affected by perceptual aliasing, and the only solution is to redesign the sensorial apparatus or to introduce some kind of memory as an internal state. Perceptual aliasing also arises when different observations are interpreted as the same input, affecting $\mathbf{I^C}$. This is a relevant problem if the agent should take different actions from the undistinguishable configurations.

The problem of *cluster aliasing* [20] arises when a grid model frames the real-valued data and the reinforcement function provides different reinforcement for configurations belonging to the same cell or *cluster*. Grid-based interpretation is needed by learning algorithms working on discrete models such as Q-learning and TD(λ), and it is also common in learning classifier systems [6] [21]. We have cluster aliasing when configurations that receive different reinforcement correspond to the same $\mathbf{I^C}$ value, that is when

$$\exists c_1, c_2 \in C : (r_{c_1} \neq r_{c_2}) \wedge (I^C_{c_1} = I^C_{c_2}). \tag{18}$$

In this case, the control system cannot distinguish the reinforced configurations from others. For instance, we may decide that our robot receives a reinforcement whenever it is far enough from the corridor walls. This produces cluster aliasing, if classifying the distance by $\mathbf{I^C}$ we put in the same cluster the reinforced configurations and some others. In particular, when the agent gets closer to the wall, but it is still in the same $\mathbf{I^C}$ cluster corresponding to the middle position, it does not receive the same reinforcement as when it is closer to the centre. We show the effect of this situation in section V-B.

One may argue that an appropriate value function may reduce the effects of cluster aliasing. This is possible, but it is better to reduce cluster aliasing by designing an appropriate reinforcement function (being aware of the clusters' shape and distribution) without modifying the reinforcement distribution algorithm and the value function. Another approach is to change the way of how the clustering was made; we will focus our attention on it in a forthcoming paper.

IV. The *CbD* approach

Most of the problems mentioned in the previous section come from mismatch among the inputs to the agent, the trainer and the designer. We consider the relationships among these inputs by referring them to the c-space, which is objective by definition. We will show how this approach may help to clarify important aspects and to improve performance.

The c-space is often the set of variables closer to $\mathbf{O^O}$, since the designer usually considers the physical aspects of the agent-environment system. In general, the variables considered in $\mathbf{O^O}$ are a subset of $\mathcal{V}$. Therefore, it is relatively easy for the designer to define the goal configurations on the c-space, or on spaces that can be easily reduced to it.

If we consider any physical object, its configuration specifies the position of any point belonging to the object, relative to a fixed frame of reference. For a rigid object, it is enough to give its position and orientation with respect to the world frame. The cardinality of the c-space is the number of independent variables, m, required to represent it (e.g., 3 for physical objects operating in 2-D environments, and 6 for 3-D). Of course, the c-space can only be trivially plotted if $m \leq 3$, but it is anyway possible to make considerations on subspaces of this cardinality also when the c-space is larger. In mobile robotics, for example, it is often possible to apply a linear decomposition of a complex c-space into several plottable subspaces (in the target tracking example, it is possible to think about two different c-spaces for the target and the agent). Moreover the principles behind this approach are not dependent on the dimension of the c-space. The possibility to plot hyper-surfaces in the c-space makes only easier a task that can be done anyway by matrix manipulation techniques. These techniques are commonly used in robotics.

We suggest defining the reinforcement function by assigning values to hyper-surfaces defined in the c-space. The basic shape of the reinforcement function we propose is thus:

$$r(\tau) = f(\mathbf{I^R}(\tau)) = f(i(\mathbf{V}(\tau))) \qquad \mathbf{V} \in \mathcal{V}, \qquad (19)$$

where i is a function of the c-space variables. Let us now discuss the impact of this choice on the solution of the problems mentioned in section III.

A. Input mismatch

The definition of reinforcement functions related to the c-space is in principle easy for the designer and provides an objective basis to analyze $\mathbf{I^C}$, $\mathbf{I^R}$, $\mathbf{O^C}$, $\mathbf{O^R}$ and $\mathbf{O^O}$, and eventually design $\mathbf{S^C}$ and $\mathbf{S^R}$, or select new sensors. Notice that most of the information needed to relate the input to the c-space is available since it has been defined when the robot was designed. However, it is also quite easy to define it by simple considerations on sensor models, or by data collection sessions.

The analysis of input makes it possible to define maps from configurations to input values, and to highlight whether the available input produces aliasing or cluster aliasing. The design of the input interpretation ($\mathbf{S^C}$ and $\mathbf{S^R}$) may reduce undesirable effects, as they become evident by the analysis. For instance, the designer can verify whether the available sensors may provide enough information for the reinforcement function, and, eventually, design special-purpose sensor systems to implement an external trainer [9]. Since it is possible to highlight the reinforced configurations, how they are perceived, and which other configurations are perceived as the same, and so indirectly rewarded, it is possible to redefine $\mathbf{S^C}$ or $\mathbf{S^R}$ to reduce cluster aliasing. By knowing the reinforced configurations it is also possible to control coherence.

B. Immediate and delayed reinforcement

The percentage and location of reinforced configurations may be designed with the support of the c-space and possible problems may be faced. We define two properties of the reinforcement function designed on the c-space:

- *completeness*: the capability of the reinforcement function to provide appropriate reinforcement in *all* the goal configurations
- *minimality*: the capability of the reinforcement function to reward *only* the goal configurations

Completeness is a necessary condition to minimality. An incomplete reinforcement function may prevent the learning system to achieve the desired behavior. A non-minimal reinforcement function includes some aspects different from the ones strictly needed to describe the goal, and it may lose coherence with respect to the goal definition. A minimal reinforcement function may provide too scarce a reinforcement. We can study on the c-space an appropriate trade-off between reinforcement rate [22] and minimal goal description. In order to increase the probability of obtaining reinforcement it is better to lose minimality than completeness. This can be obtained by enlarging $\mathcal{G}$ with reinforced sub-goals related to the main goal. As a particular case it is possible to consider providing reinforcement to all the configurations (immediate reinforcement), although this is not always possible [12], and may introduce undesired bias [13] when not directly related to the main goal, as discussed in section II. This may also introduce aliasing and incoherence, which can be studied on the c-space.

It is possible to analyze, or even design, the exploration of the c-space, highlighting eventual biases, or problems in the exploration strategy. It is also possible to check the distribution of the reinforced configurations on the c-space: if they are evenly distributed, it is possible to adopt standard exploration strategies, otherwise it is possible to tune the reinforcement function parameters to optimize the temporal reinforcement distribution [22], or even select special purpose exploration strategies [23], and check on the c-space their effectiveness.

Finally, it is possible to analyze the evolution of the system in the c-space after the learning phase, in order to evaluate its performance.

C. Extensions

To face specific designer's needs, we may consider an extended class of reinforcement functions, depending on past values of the c-space variables:

$$r(\tau) = f(\mathbf{I^R}(\tau^{\preceq})) = f(i(\mathbf{V}(\tau^{\preceq}))) \qquad \mathbf{V} \in \mathcal{V}. \qquad (20)$$

This allows, for instance, definition of the reinforcement functions based on the difference between present and past values of variables in the c-space, as in the target tracking example presented in section III-C. However this *temporal extension* also increases the complexity of analysis and design which are still based on the objective definitions provided by the c-space.

An independent extension can be done on the concept of the c-space itself. We introduce the *augmented c-space* (*ac-space*) as the c-space augmented with variables not related to physical aspects, such as messages coming from other agents. In some applications, we may also need to represent this *information data*, since they are part of the description of the state of the system, and may take part in the definition of the task, or of the input.

V. Case Study

In this section we show the application of our approach on a case study. We discuss the development of a reinforcement function to make a mobile robot learn to run in the middle of a corridor. By this simple example we try to face the problem of designing a correct reinforcement function , while taking into account the features described in section IV.

This section is composed of three parts: in the first we use an a priori analysis to define the reinforcement function considering the c-space, in the second part we verify our a priori analysis and in the last part we validate our approach with the evaluation of the performance of the learned controller.

A. A priori analysis

To deeply understand the issues discussed in section III-B, we define a reinforcement function that considers $\mathbf{I^R}$ different from $\mathbf{I^C}$. We expect the agent to learn a correct mapping from $\mathbf{I^C}$ to actions to reach the $GOAL$ defined as *achieve* and *maintain* the centre of the corridor. The reinforcement function directly reinforces the goal configurations for such a task.

The reader will see that in our approach we start from a minimal and complete reinforcement function designed on the c-space for the specific task. We observe that this function cannot supply sufficient reinforcement and we relax the boundary condition as explained in section III-C: our purpose is to find out a trade-off between probability of obtaining a reinforcement and cluster aliasing, trying to preserve completeness and coherence.

The environment is a corridor 4 meters wide and indefinitely long. For this particular task, we can represent the c-space of the agent-environment system by a plot, since the problem can be faced in 2 dimensions because of the infinite length of the corridor. The free variables considered are the distance of the robot from the left wall and its orientation in the corridor. We have selected these settings to make easier the presentation. We may also notice that the three variables typical of bidimensional problems, become two on this analysis since the position with respect to the corridor axis is irrelevant for this application. This is a simplification that can be adopted to reduce the dimension of the considered c-space.

The analysis of c-space has to consider the presence of obstacles or constraints due to robot's kinematics. In our environment there are no obstacles, but robot shape, kinematics and the decision of maintaining a positive constant speed give some constraints on the minimum steering radius, so that in some configurations the robot cannot avoid a future collision with a wall.

Figure 3 represents the c-space for our environment. It is described by the distance d of the robot barycentre from the left wall, and its orientation θ with respect to a line orthogonal to the walls. In black are the configurations the agent cannot reach because of its dimension (constraint). In dark gray are the configurations from which the agent cannot escape, given its limited steering abilities (r_{max} is its steering radius) and its fixed speed forward. In figure 3 it is also shown, in transparent gray, the set of configurations where the robot perceives the distance to the wall with a sonar sensor orthogonally directed toward its left side. The actual robot has six such sensors, covering the frontal 210°; the values from each sensor are classified in three intervals. In figure **??** we plot the 84 areas of the c-space corresponding to the sensorial clusters for our example. Different grey levels correspond to sets of configurations perceived as different.

As described above, the task is to achieve the middle of the corridor and maintain this position while moving forward. Black dots marked as $T1$ and $T2$ in figure 3 represent the goal configurations; the $GOAL$ is to *achieve* and *maintain* these configurations, that is, we want the agent to reach the centre of the corridor and run along it, either in the North or South directions; indeed, due to perceptual aliasing it will not be able to perceive the difference between them.

A delayed reinforcement defined on these two points is minimal, complete and preserves coherence with the goal defined by the designer. In spite of this, anyone can notice the small size of the reinforced area of the c-space. As discussed in section III-C, it is possible to find out a trade-off among minimality, coherence, aliasing, and probability to obtain a reinforcement by considering a reinforced area larger than the goal area.

This trade-off can be obtained by relaxing the bounds in the definition of the goal configuration. We consider a reinforcement function that gives a reward when the robot crosses the interval of ± 30 cm around the centre of the corridor. We also analyze how relaxing the bounds on the orientation of the robot when crossing this area can affect aliasing and coherence. This requires considering different, rectangular reinforced areas on the c-space and compare

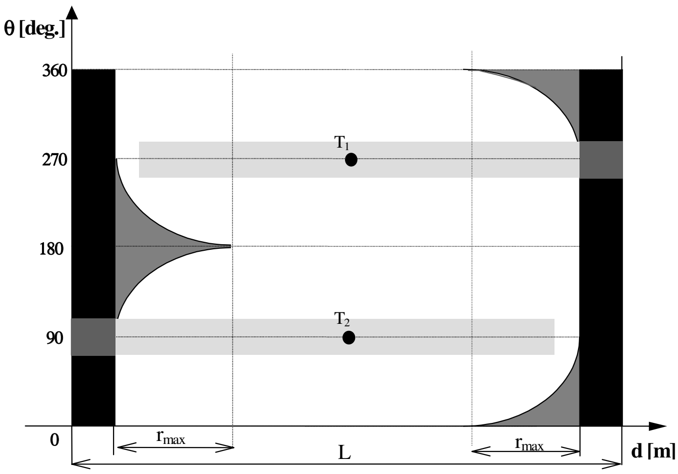

Fig. 3. The configuration space for a mobile robot operating in a corridor

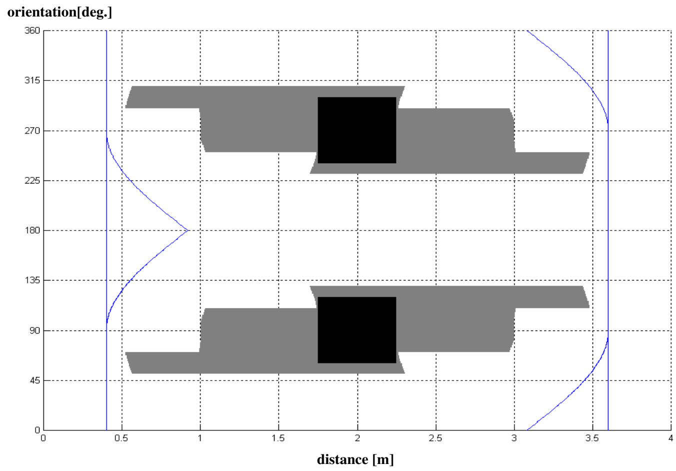

Fig. 4. The Real area covered by a reinforcement function designed on the c-space (light gray) and the theoretical (black).

them with the corresponding cluster areas. In figure 4 we present an example of the impact of the cluster aliasing. The reinforcement is given when the robot is within ±30 cm from the centre of the corridor, with an orientation in the range of ±30° with respect to the walls: the figure shows in black the reinforced area, and in light gray the clusters containing both reinforced and partially reinforced configurations.

Following our approach we want to increase the reinforced area without introducing too much cluster aliasing. In order to understand this feature, we analyze how the enlargement of the reinforced area in the c-space affects the ratio between the area corresponding to reinforced configurations and the area corresponding to clusters containing reinforced configurations. In figure 5, we plot the mentioned ratio and we notice a growing trend reaching its maximum at about ±38°, and decreasing after that. Looking at this plot we expect to find the best trade-off among the above mentioned reinforcement function features (probability of obtaining a reinforcement and cluster aliasing) in the proximity of its maximum. After reaching the maximum, the trend in figure 5 decreases; this means that the reward that we give to the agent is increasing cluster aliasing, to some extent. In fact, reinforcement is also given to configurations that we do not want or expect to reward. Notice that this can also be computed in multi-dimensional spaces.

To analyze coherence, we sample this trend every 20° (±10°, ±30°, ±50°, ±70° error accepted in the robot orientation) and we plot for each cluster the *percentage of the area covered by the reinforcement function*; this index provides a measure of how much a particular cluster is perceived as containing goal configurations. The higher the index value, the larger the area covered by the reinforcement function with respect to the cluster area. Values between 1 and 0 correspond to aliased areas. The aliasing is maximum for 0.5. Also this can be easily computed in multi-dimensional spaces.

In figure 6(a) (±10°), we notice how little of the c-

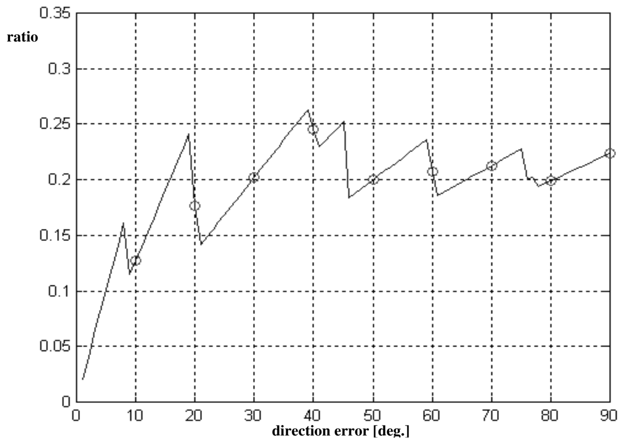

Fig. 5. The ratio between the area covered by the reinforcement function and the area covered by the states partially overlappingwith it.

space is reinforced; thus the corresponding reinforcement function gives a low probability to obtain reinforcement. Figure 6(b)(±30°) corresponds to a reinforcement function providing enough reward for the desired configuration. Cluster aliasing is still quite low; we expect to obtain satisfactory results using this reinforcement function. In figure 6(c) (±50°) we have a correct reinforcement in the neighborhood of the goal, but a certain level of aliasing as well. As presented before, this negatively influences the learning process. We guess that this may require the agent longer learning experiments to reach the $GOAL$, since the problem is not related to any incoherence in the reinforcement function. We find some incoherence in the last reinforcement function (figure 6(d) ±70°), where we reinforce new, alternative configurations (corresponding to the thinner peaks).

Totally relaxing the bound in the robot orientation (±90°, not shown in figure 6) when crossing the middle of the corridor, we obtain a reinforcement function that gives the agent a high probability of receiving a reinforce-

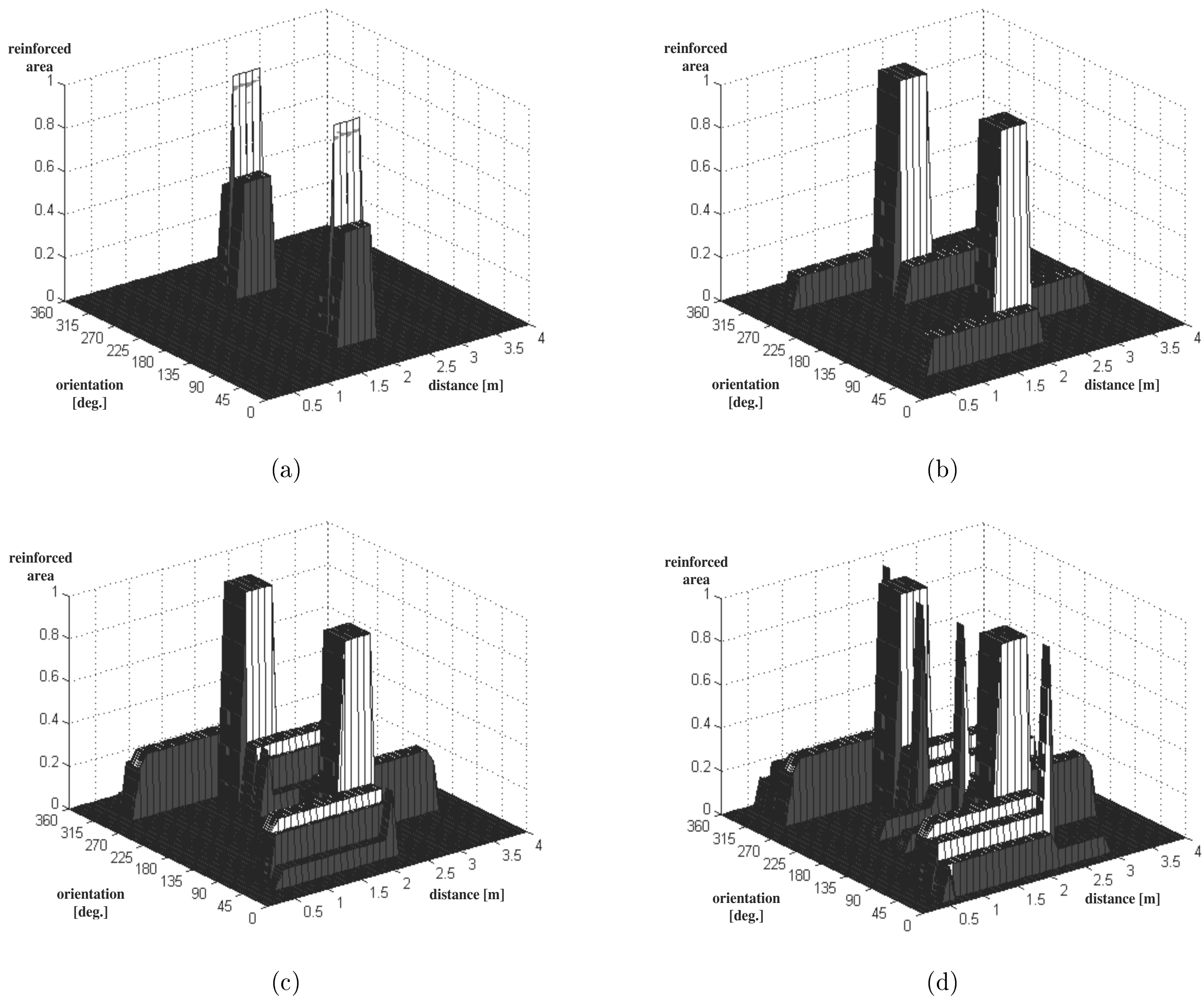

Fig. 6. The distribution of the reward on the c-space with ±10° (a), ±30° (b), ±50° (c), ±70° (d) accepted error in the direction, during the learning experiments

ment, but with a lot of cluster aliasing, incoherence and incompleteness. This is because we are describing the performance considering only the position of the agent in the corridor, without taking its orientation into account.

B. Experimental Results

In this section we verify the expectations of the a priori analysis, by comparing the results obtained by different reinforcement functions. This comparison is possible in a simulated environment, while in real applications the a priori analysis should directly guide selection of a single, correct, experimental setting. Given the large number of experiments and the long duration of each one of them (we totally perform about 8 million simulation steps) it would have been impossible to use real robots (2700 hours of experiments) to obtain the same set of results.

B.1 The learning system

The approach we are presenting in this paper is independent from the particular reinforcement learning algorithm. In the experiments we are presenting, we use a Learning Classifier System (LCS) [5][6] that we have developed to test properties of models and reinforcement distribution algorithms [24] [25]. We can consider LCSs as RL algorithms working on a model of the agent's behavior in the environment.

A classifier has a condition/message structure; the condition encodes the state of the system needed to activate the classifier. In the message part, we consider only actions to be submitted to the output interface.

We consider a simplified type of LCS [26], where each of the symbols in conditions and messages denotes an interval of real input values for the corresponding variable. $\mathbf{S^C}$ and $\mathbf{S^R}$ classify the real input values into classes corresponding to real-valued intervals; the same can be done with the output interface for real-valued output.

A rule base containing the classifiers is used as the knowledge base in a LCS, and it is updated by the RL algorithm during the interaction with the environment.

At each control step, one classifier is selected among the ones whose antecedent match the current perception (active classifiers). The consequent values of the selected classifier are sent to the output interface. The action result is evaluated by the reinforcement function, and the reinforcement distributed to the rules that have contributed to reach this state. Finally, evolutionary operators are applied to improve the rule base, by considering the strength of each rule, and/or its accuracy.

In the case study we are presenting, we show results obtained by a LCS using Q-learning as reinforcement distribution algorithm. Here, the exploration strategy is random choice with probability $p = 0.2$. We have excluded generalization, and, with it, the need for crossover: the classifiers are generated only by cover detection [6] and mutation is limited to the consequent part.

Adopting of this simplified model makes it possible to focus attention on the role of the reinforcement function.

B.2 Experimental setting

The learning activity is done on a simulated model of the mobile robot CAT [27] featuring six sonar sensors, covering the frontal 210°. The distance obtained by each sonar is affected by noise, uniformly distributed in the interval ±5% of the range. Notice that the task we have selected is apparently simple from the behavior point of view, but it is quite complex from the learning point of view, given the dimension of the search space. The learning process consists of 20 trials, each including 11 consecutive experiments. Each experiment lasts 500 control cycles, and the robot starts from a different position selected in a set of pre-defined positions, for a total of 110,000 control steps. Sets of trials to be compared with each other, at the end of the learning process, have the same randomization seed. At the end of the learning trials the set of classifiers can be divided into subsets with respect to their activation conditions. We select the best classifier from each of these subsets.

B.3 Learning trials

In our experiment, we use the reinforcement function considering a ±38° error in the orientation, since this is the maximum in the trend of figure 5. Figure 7(a) represents the average reinforcement during the learning phase. Learning activity reaches a good result around the 14^{th} trial: by observation, we noticed that the robot moves around the centre of the corridor with small oscillations.

In figure 8 we show the results of the comparison of different reinforcement functions in the same experimental setting. In the first plot (a), we use the reinforcement function with the maximum value when the robot crosses the central area with an orientation within ±10° with respect to the corridor axis. As expected, the learning phase reflects a certain lack of information if compared to the graph of figure 7(a), and it is not yet stable at the end of our trial. Looking at figure 8(b), we notice how, using ±50°, the learning phase is slower, since we introduce cluster aliasing in the reinforcement function. The introduction of some incoherence with ±70°(see figure 8(c)) increases the difficulties of the learning process. In figure 8(d) we use the incomplete reinforcement function (±90°) presented in section V: it is evident that the agent does not learn the task in the expected time.

All the experiments reach a good result with a long training time due to the convergence of Q-learning in finding the optimal policy, but comparing figure 7(a) with figure 8 you may notice how the reinforcement function affects the learning process.

C. Performance evaluation

We evaluate the performance of the learned controller by a *function* defined on the c-space and an *evaluation procedure*. Since the *GOAL* is to *achieve* and *maintain* configurations in the middle of the corridor, our performance evaluation function covers the whole c-space. It returns a real value in $[-1\ \ 1]$, inversely proportional to the distance of the present configuration from the goal configuration $\mathcal{G}$. We evaluate the performance in 27 trials, each lasting 700 control steps, starting from the same 11 positions of the learning session and from 16 intermediate ones. This is carried out in order to test the generality of the learned controller.

To compare the quality of the controllers obtained by the learning algorithm using different kinds of reinforcement functions we present in figure 9 a different set of experiments testing the performance of the learned controllers at half of the learning phase for reinforcement functions allowing an error in orientation respectively of ±10°(a), ±38°(b), ±50°(c) and ±70°(d). We consider the controllers obtained at half of the learning time since this highlights some interesting aspects that we would like to discuss.

For the first reinforcement function (see figure 9(a)) the test phase shows an agent that wanders in the corridor, and, when starting from the 18^{th} position, crashes on the wall, this is due to a poor learned behavior. The designed reinforcement function (±38°) obtains a good result also in half the learning time (figure 9(b)); the second part of learning time is spent to reduce the oscillations (as shown in figure 7(b)). Notice that the agent stays close to the centre of the corridor in all the trials. In figure 9(c), we present the results of the experiment with a reinforcement accepting a ±50° deviation from the corridor direction. The agent's difficulty to learn the correct behavior from the information given by the reinforcement function is evident. The difficulties in this case are due to the presence of aliasing in the reinforcement function definition. The results of the last experiment (figure 9(d)(±70°)) reflect the presence of incoherence in the reinforcement function affecting the first part of the learning phase. Observing the performance we noticed that the agent does not learn a correct behavior for all the starting positions, since it keeps turning around in almost all the trials except for those where it starts from the central position with the correct orientation.

In summary, we have obtained best results with the reinforcement function designed according to our a priori analysis, and the worst with incoherent and aliased reinforcement functions.

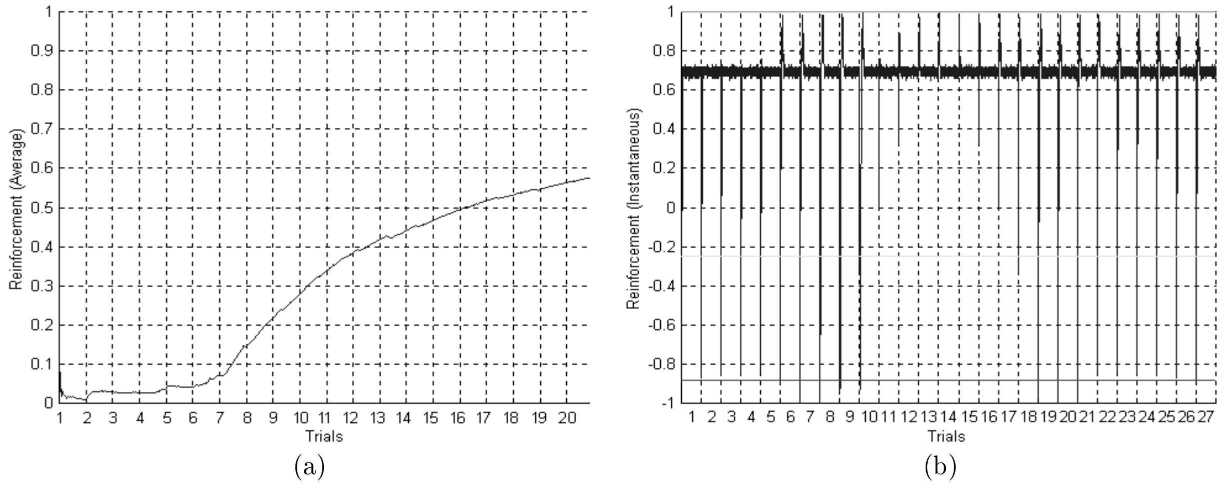

Fig. 7. The average reinforcement during the learning experiment (a) and during the evaluation test (b) for the reinforcement function accepting only a $\pm 38°$ error on direction.

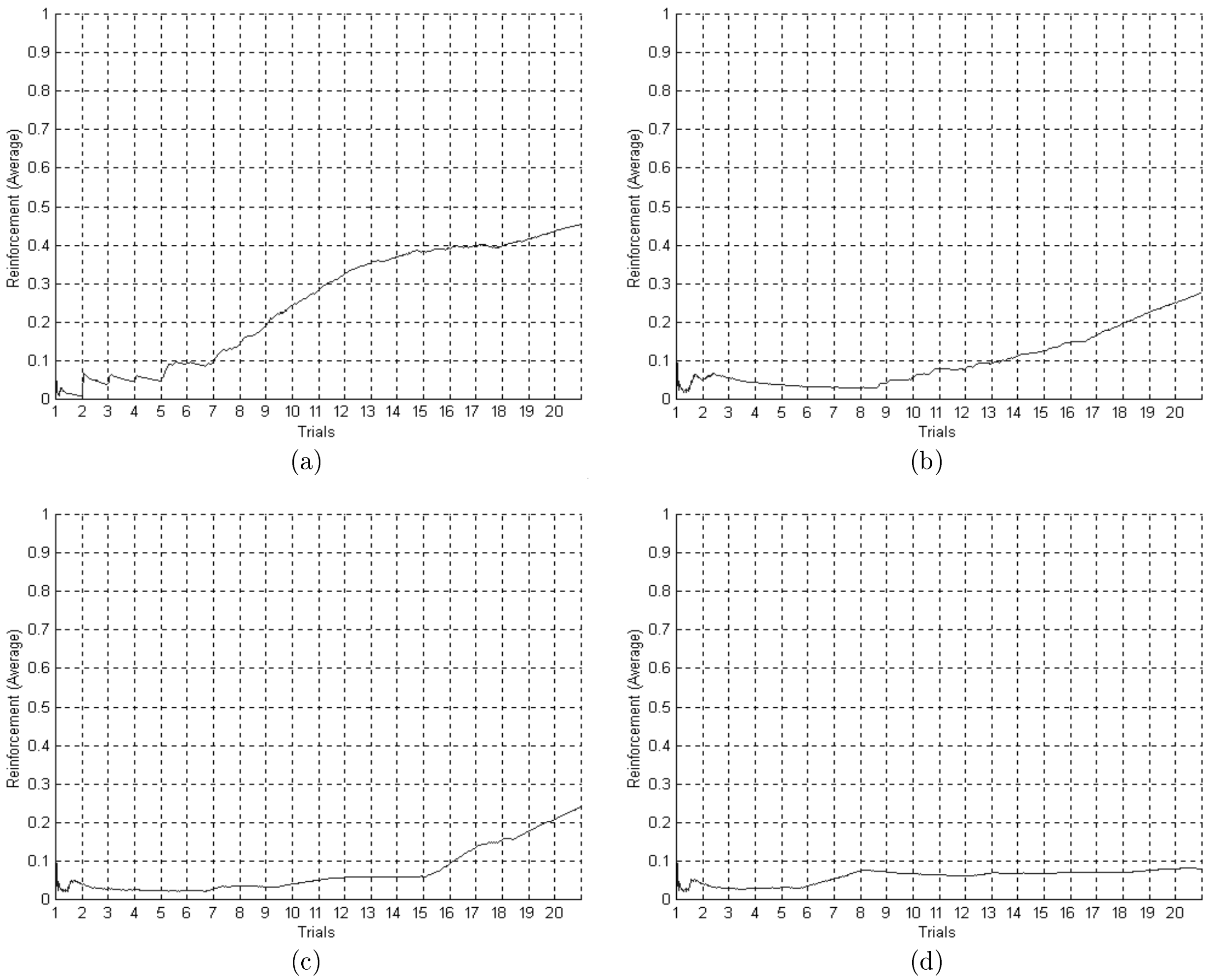

Fig. 8. The average reinforcement during the learning experiment with different errors in the robot orientation ($\pm 10°$(a), $\pm 50°$(b), $\pm 70°$(c) and $\pm 90°$(d))

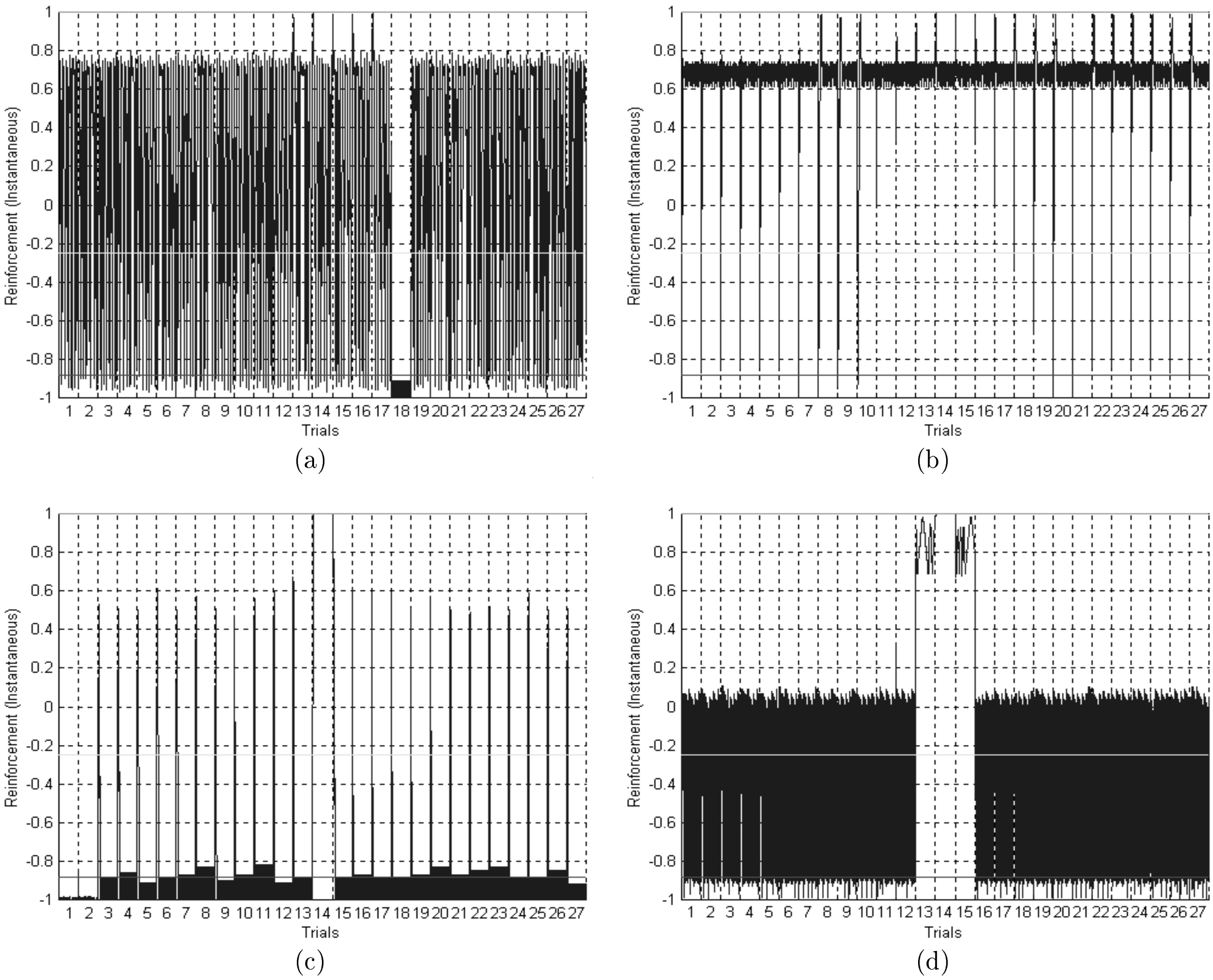

Fig. 9. The performance of the robot measured at half of the learning process with different errors in the robot orientation ($\pm 10°$(a), $\pm 38°$(b), $\pm 50°$(c) and $\pm 70°$(d))

VI. Discussion

In our case study we have considered a c-space consisting of only two variables for the sake of simplicity of presentation. The *CbD* approach can be adopted also in more complex c-spaces and with more challenging tasks. The general features of the approach can guide the definition of effective reinforcement functions even though visualization can be obtained only for sub-spaces. In some application it is possible to partition the c-space (defined on all the free physical variables) in two sub-spaces, one relevant for the task and the other not. In this case the analysis can focus on the first sub-space reducing the complexity of the reinforcement function design. In other applications such decomposition is not possible and the analysis should be done on the whole c-space. We discuss a couple of examples: in the first we partition the c-space in sub-spaces, in the second we use the whole c-space.

We first consider a target-tracking task (prey – predator): a typical task in robot learning. In this case the c-space consists of six variables which could be either the absolute positions and orientation of the two agents, or the absolute position and orientation of one of them and the relative position and orientation of the other. We focus on the second representation. The predator's task is to reach the prey while this is moving in the environment. The relevant variables for this task are those representing the relative position of the prey with respect to the predator; the absolute position and orientation of the predator and the orientation of the prey are irrelevant. The complexity of the c-space is now reduced: two variables instead of six. Now it is possible to define a reinforcement function rewarding the configurations where the predator is close to the prey analyzing the influence of cluster aliasing and the reinforcement distribution in this reduced c-space. For example, rewarding the configurations where the distance to the prey is less than a given threshold $\overline{d}$ it should be possible to choose $\overline{d}$ such that there is no sensorial cluster that is only partially covered by the reinforced configurations; thus reducing cluster aliasing problems.

In the second example, we consider a robot that should prevent intrusion in a controlled area for a surveillance task. It has also to catch the eventual intruder only if it enters a "forbidden" area. In this case, the task is similar to the previous one, but we cannot consider the absolute position of the predator as irrelevant. Reasoning on the task the designer may detect different aspects of the goal. Here, the global behavior is obtained by the intersection of two separated behaviors: catching the intruder and the control of a restricted area. It is possible to consider the desired goal as the intersection of these two sub-goals each defined on a separate sub-space, each one described by a different set of variables: on each of them it is possible to apply the *CbD* approach to define the corresponding reinforcement function. The reinforcement function we described in the prey-predator example can be used also for the intruder catching task, while for the surveillance task we have to consider the variables describing the position of the agent. By appropriately choosing the variables it is possible to define a threshold-based reinforcement function also for the surveillance task. Given such a definition, goal configurations are bounded in two hyper-cubes. The intersection of these two hyper-cubes represents the global goal configurations, and this is easily computed analytically. If it is not possible to bound reinforced configurations by simple hyper-cubes, the intersection between non-linear regions can be computed by discretization of the c-space and numerical analysis. This kind of reasoning may appear incoherent with the sensorial cluster analysis since this implies reasoning on the projection of each cluster on to the sub-spaces. However the selection of the sensors and their interpretation is often done by considering the sub-tasks as separate; also in this case the sensors for robot position estimation are different from those used to detect the intruder.

As you can see, the *CbD* approach can be adopted in complex domains once a basic analysis of the task, the selection of variables, and the design of the sensorial apparatus are defined. Given the little design effort involved we obtain the expected results from the learning process.

VII. Related work

Other researchers have studied reinforcement function design using different approaches. Let us briefly discuss their relationship with our approach.

Santos [22] [28] uses a reinforcement function definition considering $\mathbf{O^R} = \mathbf{O^C}$ and tries to face the designer problem with the estimation of high level features based on analysis of data trends. He defines a parametric reinforcement function, which is subsumed by our definition 4. The reinforcement function is defined in terms of constraints among estimators of natural language features, based on the set of the available data. Such estimators produce the input $\mathbf{I^R}$ different from $\mathbf{I^C}$, and can be considered as virtual sensors $\mathbf{S^R}$; the proposed trainer is thus an external trainer. The reinforcement is delayed.

Santos proposes the Update Parameter Algorithm (UPA) to tune off-line the constraints for estimators. This is done in order to optimize reinforcement distribution from the exploration point of view; it is aimed at guaranteeing a given amount of reinforcement assuming a random exploration policy. This may cause incoherence problems, since adapting the reinforcement function on random exploration may not guarantee a description for the desired behavior close to the designer's natural language specification.

Santos also proposes the Dynamic-UPA to modify the reinforcement function during learning. This is done to keep constant (and high enough) the percentage of received reinforcement. Following this methodology it is possible to lose minimality and completeness, since the modification of constraint bounds changes the distribution of reinforcement among the configurations, giving reinforcement to the ones close to the original goals (affecting minimality) or reducing the number of reinforced configurations (affecting completeness). This process is uncontrolled by the designer, but automatically driven by D-UPA.

Santos' approach gives a solution to the problem of the too-delayed reinforcement, since increases the number of reinforced configurations when the reinforcement is too scarce. Problems about coherence and cluster aliasing may arise, since continuous changes in reinforcement function deriving from D-UPA may affect the agent's perception of the desired behavior. In fact, the same perceived configuration may receive different reinforcement in different moments, due to the dynamic adaptation of the constraint, and $\mathbf{I^R} \neq \mathbf{I^C}$. In the application presented in the paper, this approach is anyway successful, due to the small changes induced and to the quite stable learning algorithm.

Also Matarić studied the definition of reinforcement function [13] [14] focusing on situated environment. She claims that in this case it is important to give both delayed and immediate reinforcement (*heterogeneous reinforcement*); she proposes to introduce some knowledge about the task in the immediate component (*progress estimators*) of the reinforcement function to speed up the learning process. The variables in her reinforcement functions refer to increments of c-space variables or to variables of ac-space; therefore, her proposal is subsumed by *CbD*.

Bacchus, Boutilier and Grove [11] propose a temporal logic to represent reinforcement function for temporally extended behaviors. They represent a $GOAL$ by a temporal proposition and they define it as the simplest reinforcement function. This is on the line of our approach which in principle can be extended to include temporal logic formulae as relationships between c-space variables appearing in the reinforcement function (see eq. 20).

VIII. Conclusion

We have presented an approach to the analysis and the design of reinforcement functions that is especially relevant for RL algorithms working on a finite number of states and actions such as Q-learning and TD(λ).

We have presented some problems, and discussed how to tackle them. In particular, we have introduced *CbD* as an approach to the definition of reinforcement functions and sensorial data interpretation, based on an objective model of the system. *CbD* is a contribution to the definition of a reinforcement learning engineering, as opposed to crafting; thus it is possible to produce expectations which will be confirmed by experience, instead of working by trial and error relying on the designer's experience.

We have presented the application of our approach to a learning classifier system that learns a behavior for a simulated mobile robot in a realistic environment. We have shown how an accurate design of the reinforcement function, supported by an appropriate set of tools, can positively affect the learning performance. We have also seen how a methodological approach to reinforcement function design may support a priori expectations about this performance, thus improving the design activity.

We have also applied our approach on a real CAT robot. The methodology does not change, since its principles are valid both in simulation and the real world. We first learned a satisfactory controller for the simulated robot and then applied it on the real one. CAT was not able to keep the center of the corridor, but had a behavior similar to the simulated one navigating slightly on the right of the middle line. Then, we have performed learning trials with the real robot that was finally able to achieve the desired result. A posteriori, we verified that the problem was due to the incomplete model used in simulation, which did not take into account mechanical asymmetry. In this case, the stochasticity of observations did not give any relevant contribution.

In general, it could be argued that $\mathbf{C}$ should take in account the stochasticity of the environment, introduced by observations, unmodeled aspects, or unpredictable events, such as those induced by other agents. In the approach we have proposed, stochasticity is not considered, since the stochastic model of the agent-environment system is usually unknown and hard to estimate [13]. The RL algorithms are designed to evolve the controller in such a stochastic environment. The reinforcement signal is anyway affected by uncertainty and stochasticity. In our approach we suggest how to link it to the objective variables of the c-space so that the design is not left only to the designer's ability, and it is possible to analyze the eventual stochasticity in perceiving the reinforced configurations.

We are also presently working on other aspects related to the design of reinforcement learning systems on real world devices. In particular, we are studying the other side of the $\mathbf{I^R}$–$\mathbf{I^C}$ relationship problem: how to cluster the sensorial data to best match the desired reinforcement function. We have studied the shape and distribution of clusters with the support of the c-space, which can be influenced by both sensor features and data interpretation; an accurate design of $\mathbf{S^C}$ and $\mathbf{S^R}$ can improve the performance of the learning system and of the obtained controller.

An open problem concerns the relationship between reinforcement function and generalization, one of the most interesting features of LCS, and RL in general. We argue that generalization is a way of facing the problem of adapting the clusters to reduce the learned knowledge base. We are studying how a reinforcement function can influence generalization, and in this we are applying techniques related to those presented here, by analyzing on the c-space the impact of generalization.

We are also working on these topics applied to *Learning Fuzzy Classifier Systems (LFCS)* [29], where a fuzzy model is learned. This introduces a better mapping between the clusters, which are fuzzy, and the real-valued data. However, some problems are also introduced in the learning process. We have discussed this approach elsewhere [24], [25], [30], and we will compare the interval-based and the fuzzy-based approaches in a forthcoming paper.

Acknowledgements

This research is partially supported by the Politecnico di Milano Research Grant "Development of autonomous agents through machine learning", and partially by the "CERTAMEN" project, co-funded by the Italian Ministry of University and Scientific and Technological Research. We are indebted with Nicolino De Marco for the first implementation of the learning system, and Fabio Fiorellato, for the development of the analysis tool. We are also grateful to Marco Dorigo and the anonymous reviewers for their suggestions to improve the presentation.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction*, MIT Press, Cambridge, Massachusetts, 1999.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[3] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[4] P. Dayan and T. J. Sejnowski, "TD(λ) converges with probability 1," *Machine Learning*, vol. 14, no. 3, pp. 295–301, 1994.

[5] L. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artificial Intelligence*, vol. 40, no. 1–3, pp. 235–282, 1989.

[6] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[7] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design and Tuning, second ed.*, ISA, 1995.

[8] N. K. Bose and P. Liang, *Neural network fundamentals with graphs, algorithms, and applications*, McGraw-Hill, New York, NY, 1996.

[9] M. Dorigo and M. Colombetti, *Robot shaping: an experiment in behavior engineering*, MIT Press / Bradford Books, Cambridge, MA, 1997.

[10] Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transaction on Computers*, vol. C-32, no. 2, pp. 26–38, feb 1983.

[11] Fahiem Bacchus, Craig Boutilier, and Adam Grove, "Rewarding behaviors," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, Menlo Park, Aug. 4–8 1996, pp. 1160–1167, AAAI Press / MIT Press.

[12] Marco Dorigo and Marco Colombetti, "The Role of the Trainer in Reinforcement Learning," in *Proceedings of MLC-COLT '94 Workshop on Robot Learning*, S. Mahadevan et al., Ed., New Brunswick, NJ, 1994, pp. 37–45.

[13] Maja J. Matarić, "Reward functions for accelerated learning," in *Proc. 11th International Conference on Machine Learning.* 1994, pp. 181–189, Morgan Kaufmann.

[14] Maja J. Matarić, "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[15] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[16] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, pp. 123–158, 1996.

[17] A. Bonarini, "Anytime learning and adaptation of hierarchical fuzzy logic behaviors," *Adaptive Behavior Journal*, vol. 5, no. 3–4, pp. 281–315, 1997.

[18] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda, "Purposive behaviour acquisition on a real robot by a vision based reinforcement learning," in *Proceedings of MLC-*

COLT '94 Workshop on Robot Learning, S. Mahadevan et al., Ed., New Brunswick, NJ, 1994.

[19] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, pp. 45–83, 1991.

[20] A. Bonarini, C. Bonacina, and . Matteucci, "Fuzzy and crisp representation of real-valued input for learning classifier systems," in *Learning Classifier System: From Foundation to Application*, P. L. Lanzi, W Stolzmann, and S. W. Wilson, Eds., pp. 107–124. Springer-Verlag, Berlin, D, 2000.

[21] P. L. Lanzi, W Stolzmann, and S. W. Wilson, Eds., *Learning Classifier System: From Foundation to Application*, Springer-Verlag, Berlin, D, 2000.

[22] J. M. Santos, *Contribution to the study and the design of reinforcement function*, Ph.D. thesis, Universidad de Buenos Aires, 1999.

[23] R. Andrew McCallum, "Efficient exploration in reinforcement learning with hidden state," in *Proceedings of the AAAI Fall Symposium on "Model-directed Autonomous Systems"*, 1997.

[24] A. Bonarini, C. Bonacina, and M. Matteucci, "Fuzzy and crisp representation of real-valued input for learning classifier systems," in *Proceedings of IWLCS99*, Cambridge, MA, 1999, pp. 228–235, AAAI Press.

[25] A. Bonarini, "Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems," in *Proceedings of GECCO99*, Cambridge, MA, 1999, pp. 52–59, AAAI Press.

[26] A. Bonarini, "Reinforcement distribution to fuzzy classifiers: a methodology to extend crisp algorithms," in *IEEE International Conference on Evolutionary Computation – WCCI-ICEC'98*, Piscataway, NJ, 1998, vol. 1, pp. 51–56, IEEE Computer Press.

[27] A. Bonarini, "Evolutionary learning of fuzzy rules: competition and cooperation," in *Fuzzy Modelling: Paradigms and Practice*, W. Pedrycz, Ed., pp. 265–284. Kluwer Academic Press, Norwell, MA, 1996.

[28] J. M. Santos and C. Touzet, "Dynamic update of the reinforcement function during learning," *Connection Science, special issue on Adaptive Robots*, vol. 11, no. 3/4, dec 1999.

[29] A. Bonarini, "Learning fuzzy classifier systems," in *Learning Classifier System: new directions and concepts*, P. L. Lanzi, W Stolzmann, and S. W. Wilson, Eds., pp. 83–106. Springer-Verlag, Berlin, D, 2000.

[30] M. Matteucci, "Fuzzy learning classifier system: Issues and architecture," Tech. Rep. 99.71, Politecnico di Milano, Department of Electronics and Information, Nov. 1999.

Andrea Bonarini was born in Milan in 1957. He received a Laurea (Master of Technology) in Electronics Engineering in 1984 and a PhD in Computer Science in 1989, both from the Politecnico di Milano, Italy. He is Associate Professor at the Department of Electronics and Computer Engineering of the Politecnico di Milano. He has been a member of the Politecnico di Milano Artificial Intelligence and Robotics (PM-AI&R Project) since 1984. He is a founding member of the AI*IA (the Italian Association for Artificial Intelligence), and among the founders of the IEEE-NN Council Italian RIG. He is coordinating the PM-AI&R Lab, where he has developed several Autonomous Mobile Robots. He is also participating to the Robocup effort (middle size league) with the Italian National Team (ART). His research interests include Behavior Engineering, Navigation, Data Interpretation and Control for Autonomous Robotic Agents, Mobile Robot Design, Reinforcement Learning and Fuzzy Systems.

Claudio Bonacina obtained his Master Degree in Computer Engineering at the Politecnico di Milano, Milan, Italy in 1999. Since October 1999 he has been working as a Ph.D. student in the Intelligent Computer Systems Centre at the University of the West of England (UWE) in Bristol, UK. His research investigates the possibility of applying Evolutionary Computation and Reinforcement Learning techniques to Multi Agent Systems. His Ph.D. is sponsored by the Intelligent Business Systems Research (IBSR) Group at BT Laboratories. He is also member of the Learning Classifier System Group (LCSG) at UWE.

Matteo Matteucci was born in Nuoro in 1974. He obtained a Laurea (Master of Technology) in Computer Engineering at the Politecnico di Milano, Milan, Italy in 1999. Since 2000 he has been working as a Ph.D. student in the Artificial Intelligence and Robotics Project at Politecnico di Milano (PM-AI&R Project), Milan, Italy. His research interests are mainly in machine learning and robotics: neural networks, genetic algorithms, fuzzy logic, and autonomous mobile robots. He is presently working on his PhD thesis about the design for adaptiveness in physical and software agents. He is also member of the AI*IA (the Italian Association for Artificial Intelligence).