Anytime learning and adaptation of structured fuzzy behaviors

Andrea Bonarini
Politecnico di Milano Artificial Intelligence and Robotics Project
Dipartimento di Elettronica e Informazione - Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano - Italy
E-mail: bonarini@elet.polimi.it
URL: http://www.elet.polimi.it/people/bonarini/
Phone: +39 2 2399 3525
Fax: +39 2 2399 3411

Abstract

We present an approach to support effective learning and adaptation of behaviors for autonomous agents with *reinforcement learning* algorithms. These methods can identify control systems that optimize a reinforcement program, which is, usually, a straightforward representation of the designer's goals. Reinforcement learning algorithms are usually too slow to be applied in real time on embodied agents, although they provide a suitable way to represent the desired behavior. We have tackled three aspects of this problem: the *speed of the algorithm*, the *learning procedure*, and the *control system architecture*. The *learning algorithm* we have developed includes features to speed up learning, such as *niche-based* learning, and a representation of the control modules in terms of *fuzzy rules* that reduces the search space, and improves robustness to noisy data. Our *learning procedure* exploits methodologies such as *learning from easy missions* and *transfer of policy* from simpler environments to the more complex. The *architecture* of our control system is layered and modular, so that each module has a low complexity and it can be learned in a short time. The composition of the actions proposed by the modules is either learned or pre-defined. Finally, we adopt an *anytime learning* approach to improve the quality of the control system on-line and to adapt it to dynamic environments.

The experiments we present in this paper concern learning to reach another moving agent in a real, dynamic environment that includes nontrivial situations such as when the moving target is faster than the agent, and when the target is hidden by obstacles.

Keywords: Reinforcement Learning, Autonomous Agents, Fuzzy Control, Hierarchical Behaviors, Anytime Learning, Anytime Algorithms.

Running title: Learning fuzzy behaviors

Accepted for pubblication on the Special Issue of the Adaptive Behavior Journal about "Complete agent learning in complex environments", Maja Mataric (Ed.), n.5, 1997.

1. INTRODUCTION

Machine Learning (*ML*) techniques are often adopted to learn the control system of an autonomous agent that should perform a pre-defined task. ML is used to save design efforts, when the environment is not easy to model, and the task is only known in terms of a high level description of the goals. In particular, *Reinforcement Learning* (*RL*) (Kaelbling, Littman, Moore, 1996) seems, in principle, suitable for addressing this problem. The task and goal descriptions can be easily used to define a *reinforcement program* that rewards those control actions that bring the agent toward the achievement of the goal. The RL algorithms typically optimizes the expected future reward, resulting in a control system that implements the desired behavior.

However, in many situations, learning with a real, embodied agent may be hard, since the learning algorithm is slow, and data acquisition makes it even slower. It is typical to have a *control cycle* (i.e., a sense-act sequence) lasting up to 1 second, and learning algorithms requiring millions of control cycles to converge. This means that a learning activity may last days, whereas most of our agents can survive less than 1 hour without recharging their batteries.

We present an approach to speed-up reinforcement learning that considers three aspects: the *learning algorithm*, the *learning session* and the *control system structure*. We will demonstrate a RL algorithm effective in learning complete behaviors for embodied autonomous agents. We operate in an *anytime learning* framework (Grefenstette, 1992) that makes it possible to run the learning algorithm in a simulated environment in parallel with the real agent to improve the quality of the control system, and to adapt it to dynamic environments.

1.1 SPEEDING UP REINFORCEMENT LEARNING

There are many possibilities to speed up RL algorithms. One of them consists of selecting a limited set of variables and values, such that the search space is small. Often, this has been done by partitioning the variable ranges into few intervals (e.g., (Mahadevan and Connell, 1992) (Dorigo and Colombetti, 1994)). This may produce discontinuous output, and an excessive approximation that may lead to *perceptual aliasing* (Whitehead and Ballard, 1991). An alternative approach consists of partitioning the variable values into *fuzzy sets* (Zadeh, 1966). The resulting model still exploits the data precision available from sensors and actuators, although it is based on few linguistic values (Saffiotti, Konolige, and Ruspini, 1995), (Bonarini, 1996a). We discuss the use of fuzzy sets to represent the variables of the control system in Section 2. We have successfully adopted fuzzy models in *ELF* (*Evolutionary Learning of Fuzzy rules*) (Bonarini, 1993), (Bonarini, 1994a), (Bonarini, 1996a), the learning system that we have also used in the experimental activity presented in this paper. In Section 3, we present *ELF* and discuss its main features, including its *niche-based* partitioning of the population, which reduces competition among different solutions and contributes to reaching a solution in a short time. A complete description and discussion of *ELF* has been presented elsewhere (Bonarini, 1996a), and it is outside the scope of this paper.

Another approach to increasing the speed of a reinforcement learning algorithm is based on *decomposing* the task into simpler ones. This approach is very common both in Robotics (e.g., (Brooks, 1986), (Arkin, 1992), (Saffiotti et al., 1995)(Mataric, 1995)), and in Reinforcement Learning (e.g., (Mahadevan and Connell, 1992), (Singh, 1992), (Dorigo and Colombetti, 1994), (Mataric, in press)). Different decomposition methods have been proposed. In Section 5 we discuss some of them, and we propose a decomposition technique based on the analysis of the variables used by the modules of the control architecture. We present an example of the application of such techniques to the task described in Section 4, where the agent attempts to learn to reach a moving target, even when this runs faster, and it is occasionally hidden by obstacles.

The last approach to speed up learning that we consider in this paper consists of designing accurately the *learning sessions*. For instance, we may run the learning system on increasingly difficult tasks, performing shaping of the agent (Singh, 1992), (Dorigo, Colombetti, 1994). When the task can be so decomposed, "*learning from easy missions*" (Asada et al., 1994) may bring the system to converge quickly to the desired behavior. In some cases it may also be possible to learn behaviors in simplified environments, so that the corresponding control system can successfully act in more complex situations (*transfer of policy*). In Section 6, we discuss how we have adopted these techniques to obtain fast convergence of our learning algorithm applied to the above mentioned task. In Section 7 we report experimental results supporting all the design choices discussed in the paper.

1.2 ANYTIME LEARNING

Despite the techniques we mentioned to speed up learning, it may still be hard to learn on real, embodied autonomous agents. Perhaps for this reason, most of the papers about learning agents concern simulated environments. However, it is well known that the models used in simulation may hide important aspects that could make the agent fail in a real environment, when running the behavior learned in simulation (Brooks, Mataric, 1993).

Learning in simulation is much faster than in the real world: a simulated control cycle may last few milliseconds, thus feeding data to the learning algorithm at a rate up to 1,000 times faster than the real agent running in real time. Moreover, many learning techniques may run only in simulated environments, since they rely on specific session settings, such as random re-positioning of the learning agent every few steps to enhance the exploration of the search space, or evaluation of different behaviors in the same situations, as, for instance, is done with *Learning Classifier Systems* (*LCS*) adopting the "Pitts" schema (Booker, 1988)(Booker, et al., 1989).

The main problem with learning in simulated environments concerns the model of the interacting agent and environment adopted in simulation. If it is a "good" representation of the real world, and it has a computationally tractable mathematical structure, then *Optimal Control* (Kirk, 1970) and *Adaptive Control* (Sastry and Bodston, 1989) techniques are guaranteed to produce the best controllers. In practice, these techniques can only be applied to few classes of models, which are inadequate to represent all the relevant aspects of the environment, the goal, and the agent. In general, the behavior learned on an inaccurate, simplified model may not be appropriate in the real environment. Moreover, a model identified at a given moment may become invalid at a later time, since the main characteristics of the environment and the agent may vary.

To address all these problems we have adopted an *anytime learning* approach. The term "*anytime learning*" was introduced by Grefenstette (Grefenstette, 1992) to denote a class of learning algorithms having the characteristics of *anytime algorithms* (Dean and Boddy, 1988): the algorithm can be suspended and resumed with negligible overhead, the algorithm can be terminated at any time and will return some answer, and the returned answers improve over time. The anytime learning algorithm proposed by Grefenstette has two modules running in parallel: a *learning system* that learns behaviors running on a simulated environment, and an *execution system* that controls the embodied agent in the real environment. The execution system uses the best controller produced so far by the learning system, which continues to try to improve it, running in parallel. At the same time a *monitor* checks whether the model adopted for the simulation still matches the actual environment, and, eventually, updates the simulation model. Thus, when the environment changes, the simulation model is modified accordingly, and the learning system works on a simulated environment always reflecting the real one.

In Grefenstette's original work (Grefenstette, 1992) (Grefenstette, in press), only few parameters are monitored and updated. Moreover, the learning system is a *Learning Classifier System* (Booker, 1988), (Booker et al., 1989), where a *Genetic Algorithm* (Goldberg, 1989) learns the best controller by evolving a population of controllers, thus following the "Pitts" schema. This requires the evaluation of a lot of genes, each one representing a whole controller, and each controller should be tested enough to estimate its value. Since this may require a large amount of computational time (Grefenstette, in press), it may not be appropriate for many applications (Bonarini, 1996a).

The proposal we present in this paper belongs to the anytime learning framework, but with a more efficient learning algorithm and a different control model, based on fuzzy sets instead of intervals. Moreover, we exploit the anytime learning schema, by learning from simulated easy missions, and with a comparative reinforcement program, applicable only in a simulated setting. Finally, our *monitor* checks not only the limit parameters (such as maximum speed and maximum steering angle (Grefenstette, in press)), but also more interesting parameters, as discussed in Section 5.

2. FUZZY MODELS AND REINFORCEMENT LEARNING

In this Section, we introduce and motivate the adoption of fuzzy models for the control system to be learned by reinforcement learning algorithms (Bonarini, 1996c). Fuzzy sets are a powerful tool for representing sets with fuzzy boundaries. Any element of the universe of discourse can belong to a fuzzy set with a given *degree of membership* μ that ranges from 0 to 1. Given a range of continuous values (such as real numbers) it is possible to define a membership function that gives the *degree of membership* μ to a fuzzy set of any value belonging to the universe of discourse. Each fuzzy set can be denoted by a *label* that, usually, also denotes a *class* of values significant for the application. For

instance, when designing an embodied autonomous agent, it may be relevant to distinguish between *close* and *distant* objects, or among *large*, *medium*, and *low* speed.

Fuzzy sets have the interesting property of admitting partial overlapping of their membership functions. This gives the possibility of a gradual change of classification of contiguous values as belonging to contiguous classes, as shown by the fuzzy representation on the left of figure 1. Here, we have three fuzzy sets, each corresponding to a class relevant for the interpretation of the variable *Dist*, the distance from an object in front of the agent.

*** INSERT FIGURE 1 ABOUT HERE ***

With a fuzzy classification, the value 100 is considered as completely *Normal*, $\mu_{Normal}(100) = 1$; the nearby value 110 is still considered as *Normal* with a degree $\mu_{Normal}(110) = 0.5$, but it is also considered as *Distant* to some extent $\mu_{Distant}(110) = 0.5$. With an interval-based classification as the one shown on the right, which is commonly adopted for autonomous agents, the value 109 is considered as *Normal*, and the close value 111 is considered as *Distant*. This is in contrast to the common sense interpretation of these classes, and may result in abrupt changes of the action taken by a control system based on this classification. It is possible that people and animals adopt a classification approach similar to the fuzzy one (Zadeh, 1966), since it is easier to reason with few concepts instead of with single members of a potentially infinite set of real values; in addition, the quality of the available sensors encourages the development of concepts with fuzzy borders and partially overlapping. The same considerations can also be drawn for embodied artificial agents.

As shown on the right of figure 1, a fuzzy set may also describe an interval, and in particular an interval consisting of only one point: a crisp value. Therefore fuzzy sets can be adopted to integrate fuzzy and crisp concepts. Notice also that the *definition* of a fuzzy set is not fuzzy, but crisp; in particular, it can be used to map real values to pairs $<label, \mu>$.

A variable ranging on a set of fuzzy sets is usually called a *fuzzy variable*. For instance, we can have the fuzzy variable *Dist* ranging on the set {*Close, Normal, Distant*}, where the three labels denote the fuzzy sets shown in figure 1. When we use fuzzy variables in a control system, usually we have a set of fuzzy rules. A *fuzzy rule* is an *If-then* rule mapping fuzzy variables to other fuzzy variables. In fuzzy control, the variables in the antecedent classify the inputs from sensors, and the variables in the consequent classify the control actions. However, as we will see in Section 5, fuzzy rules may also have different input and output, and can be used to implement higher levels of a structured control system.

The *fuzzy inference cycle* starts by matching the inputs, usually real numbers: for each, the degree of matching with the fuzzy values of the corresponding variable is computed. Then, fuzzy operators (Dubois and Prade, 1980) are used to combine the matching degrees of the different variables to obtain a matching degree for the state described by all the input variables. At this point, all the matching rules are triggered: each one proposes an action with a weight that depends on the degree of matching of the rule. The output comes from the *aggregation* (Dubois and Prade, 1980) of all the proposed, weighted outputs. Finally, the global output is defuzzyfied, thus becoming a real-value. In other terms, a fuzzy inference process implements a mapping from real numbers to real numbers, and operates on a representation of a mapping -- the fuzzy rules -- between classifications of these numbers, the fuzzy set values for the variables. The resulting *Fuzzy Logic Controller* (*FLC*) has many, well-known, interesting properties, such as robustness, smoothness of action, and wide range of applicability (Pedrycz, 1992).

As an example, let us consider a part of a simple obstacle avoidance behavior for an autonomous agent, implemented as[1]:

```
...
(IF (> FRONT-DIST minimum-safe-dist)
     (IF (< LEFT-SIDE RIGHT-SIDE) (TURN RIGHT a-lot)
...
```

Here, the designer has used a symbolic label (*minimum-safe-dist*) to denote a concept relevant for this application: the minimum distance from an unknown obstacle that is safe for the autonomous agent. In figure 2, we show two possible interpretations of the label *minimum-safe-dist*. The one on the right is based on intervals, that on the left on fuzzy sets.

[1] This is an adaptation of one of the behaviors proposed in (Mataric, 1989), written in the Behavior language (Brooks, 1990). At this point, we have preferred to consider this way of representing a control system, to show that a fuzzy model can also be integrated in approaches not based on fuzzy rules. In the rest of the paper, we will consider fuzzy rules.

*** INSERT FIGURE 2 ABOUT HERE ***

The above part of program can now be written as:

```
     ...
(IF (FRONT-DIST IS normal)
     (IF (< LEFT-SIDE RIGHT-SIDE) (TURN RIGHT a-lot)    ...
```

The corresponding tracks of the autonomous agents controlled by this behavior are shown in figure 3. In figure 3.a the autonomous agent (the white box) is controlled by an interval-based interpretation of the term `minimum-safe-dist`, and takes crisp decisions on the border of the area (in gray) delimited by the distance `minimum-safe-dist` from the obstacle (in black). In figure 3.b the autonomous agent is controlled by the fuzzy set interpretation of the term; notice how the action is smoother than that reported in figure 3.a.

*** INSERT FIGURE 3 ABOUT HERE ***

In figures 3.c and 3.d we see what may happen when the distance sensor does not work properly, and it measures an incorrectly larger distance from the obstacle; in 3.c the autonomous agent is controlled by the interval-based controller, and it is less robust with respect to this type of errors than the one shown in figure 3.d, controlled by the fuzzy system.

Another important benefit that comes from modeling the application domain and the control system by fuzzy sets is that we have at the same time a relatively small model, and the potential to exploit all the precision available from real-valued data. An interval-based model maps a whole range of real values onto a unique interval, denoted by a label; in other terms, the control system considers all of the values belonging to the interval in the same way. A fuzzy set model maps a set of real values onto a potentially infinite number of pairs $<label, \mu>$ that bring information about both the class of the real value, and the degree of this classification. This is much more informative than a simple interval mapping. Thus, a fuzzy model provides at the same time the possibility to reason effectively on small models based on the names of the classes (the labels), and to exploit all the precision available from real-valued data.

We call a state described by a vector of values of fuzzy variables a *fuzzy state*. An agent may partially visit a fuzzy state, in the sense that the real-valued description of the agent's state may be matched by a description based on fuzzy sets with a degree between 0 and 1. For instance, referring to the first example presented in this Section, the agent's state described by a value of the real-valued variable *Dist* equal to 110, can be interpreted as the fuzzy state "*Normal*" with the degree 0.5, and as a fuzzy state "*Distant*" with the degree 0.5. A fuzzy state may be also described by values of more than one variable (for instance, *distance* and *direction* from a moving target). The concept of fuzzy state is interesting for the implementation of controllers for autonomous agents, where it is important to also evaluate partial achievements of states. Moreover, since more than one fuzzy state may be visited at the same time, we have a smooth transition between a state and its neighbors, and, consequently, smooth changes in the actions. This is also a desired property for many behaviors for autonomous agents.

3. *ELF*, THE LEARNING SYSTEM

ELF is a *Fuzzy Classifier System* (Valenzuela-Rendón, 1991) that evolves a population of fuzzy rules to maximize the expected future reward given by a *reinforcement program*; this evaluates the behavior of the agent controlled by the learned rules and produces reinforcement distributed to the rules that have contributed to the evaluated behavior. *ELF* may either build a rule base from scratch or start with an initial rule base, or even with constraints on the rule shape (Bonarini, 1996a).

Here, we first introduce the data structures implemented in *ELF*; then, we summarize the algorithm, with special attention to its *reinforcement distribution* aspect.

3.1 DATA STRUCTURE

In *ELF*, we use a *population of fuzzy rules*, and we represent them with strings of numbers. Each position of the string represents a fuzzy variable and each value a fuzzy set for the corresponding

variable. Each position can also be occupied by a "don't care" symbol ("#"), stating that the corresponding variable can take any value. Thus, the rule:

```
IF  (PREY-DISTANCE IS close) AND (PREY-DIRECTION IS left) AND
    (OBSTACLE-DISTANCE IS far) AND (OBSTACLE-DIRECTION IS any)
THEN (SPEED IS fast) AND (DIRECTION IS left)
```

is represented by the string *003# 30*.

ELF learns the best consequent (in this example SPEED and DIRECTION) values for each antecedent, i.e., the rules for each fuzzy state that maximize the expected, future reinforcement. We associate with each rule a real number between 0 and 1: its *strength* (*s*). This is an estimate of how useful the rule is in achieving the behavior that optimizes the reinforcement program. The aim of the learning system is to find the rules with the highest strength that estimates at best their real usefulness. The rule strength is only used for learning purposes, as specified below, and it does not affect the contribution of a rule to the control action.

The population of fuzzy rules is partitioned into *sub-populations* whose members share the same values for the fuzzy variables in the antecedent; in other terms, rules belonging to a sub-population are candidates for firing when the input values are classified as belonging to the same fuzzy state. Therefore, in each sub-population we have rules with the same antecedent, and different consequents. They compete to propose the best consequent for the fuzzy state described by the antecedent. This implements a fuzzy version of the idea of a *niche* adopted in *niche-GA* (Booker, 1989), (Wilson, 1994) to reduce undesirable competition.

Since the rules are fuzzy, the same real-valued input may be matched by different antecedents, with different matching degrees. Each rule in the matching set proposes a fuzzy output, and the real-valued output is computed by combining them with one of the standard combination operators (Dubois and Prade, 1980). Therefore, the sub-populations *cooperate* to produce the control action, while the members of each sub-population *compete* with each other to cover the same niche with the highest strength. This is a way to exploit both the most interesting feature of *fuzzy logic controllers*, that is *cooperation* among rules, and the feature needed by *reinforcement learning algorithms*, that is *competition* among the members of a population.

The *cardinality* of each sub-population is dynamically adapted according to the current performance of the autonomous agent in the corresponding fuzzy state (Bonarini, 1996a). At the beginning, all the sub-populations grow to explore a large search space. As the performance of the agent improves in a fuzzy state, the cardinality of the sub-population that matches this state is decreased, and the worst rule is eventually deleted. In the version of ELF adopted for the experiments we present in this paper, the optimal number of rules for a sub-population (*o_c*) is computed by the heuristic formula:

$$o\_c = \max\left(1, \left\lceil \frac{Max\_reinf - (Max\_reinf * 0.1) - Max\_vote\_sp}{(Max\_reinf * 0.1)} \right\rceil\right)$$

where *Max_reinf* is the maximum value for the reinforcement (in our case 1,000), and *Max_vote_sp* is the maximum vote so-far obtained by rules of the sub-population. From this formula, we can see that for a sub-population whose best rule has obtained a reinforcement higher than 900, the optimal cardinality is 1, whereas, if the best rule has obtained only 500, the optimal cardinality is 4. In more recent versions of ELF, the cardinality of the sub-population is computed also taking into account other aspects, such as how much the sub-population has been tested, thus estimating the reliability of *Max_vote_sp* . The goal is always to obtain the minimum number of enough tested rules with a satisfactory performance.

3.2 THE ALGORITHM

In figure 4 we give the high-level pseudo-code of *ELF*.

*** INSERT FIGURE 4 ABOUT HERE ***

During the initialization phase (line {1}), the first state is detected, and, if no rules are present in the population, a *cover detector* operator (Wilson, 1985) generates a given number of new rules, according

to the optimal cardinality of the sub-population defined above. The new rules are different from each other, they have the antecedent that best matches the detected state (possibly including also some "don't cares"), and randomly generated consequents. These constitute the first sub-population.

We call a sequence of *sense-act* (or *control*) *cycles* that end with a state evaluation an *episode*. Each episode either lasts for a given number of control cycles, or ends when a condition is matched (for instance, a given state is reached). *ELF* loops on episodes {2-9} until the end of the trial, usually determined by a maximum number of episodes.

Each episode is composed of control cycles {3-6}. At the beginning of each cycle, *ELF* identifies the set of sub-populations matching the current state. One rule from each matching sub-population is randomly selected during each episode {4}. Let us consider an antecedent a_i corresponding to a sub-population sp_i , and matching the current, real-valued state S_j. Since each antecedent consists of fuzzy sets, more than one antecedent can match the same state. Let us focus on the *i-th* antecedent. If it has not yet matched a state during the current episode, *ELF* selects a rule from sp_i, with a likelihood proportional to its strength: this rule will represent its sub-population sp_i during the current episode. In the other case, if a_i has already matched a previous state during the current episode, then *ELF* triggers the rule selected at that point. This gives ELF the possibility to evaluate the contribution that one rule per matching sub-population gives to the agent's performance. Since rules are randomly selected, all of them will have their chance to contribute to the control actions, and, so, to be evaluated.

At each control cycle, once selected the triggering rules, *ELF* computes the corresponding control action {5} by composing the actions proposed by the matching rules, by means of one of the standard composition operators (Dubois, Prade, 1980).

At the end of each episode, *ELF evaluates* the current state {7}, by running the reinforcement program. As discussed in (Bonarini, 1996b), the performance evaluation at the end of an episode, instead than at each control cycle, produces some interesting effects. If the episode is terminated when the agent reaches a particular (fuzzy) state, then the performance evaluation is done when something relevant happens, and likely provides novel information. In any case, the episode plays the role of a *filter* that averages the effects of the single actions, and has a stabilizing effect. Once *ELF* has obtained the reinforcement, it distributes it to the rules that have triggered during an episode {8}. We discuss this critical activity in detail in the next sub-section.

Next, the population is *updated* {9}. When the number of rules in a sub-population is smaller than the one estimated as the optimal with respect to the current performance of the sub-population, the cover detector is activated to produce new rules. This stimulates *exploration* when the agent needs to improve its performance in a niche. If the matching sub-population is larger than its dynamically optimal dimension, then the worst rules are deleted, thus reducing exploration and stimulating *exploitation*.

Another updating activity is done when the whole population performs better than a given threshold, and it is not updated for a given number of episodes. In this case, *ELF* saves the population and mutates a certain number of the worst rules before starting the new episode. This operation is similar to what is done in Genetic Algorithms and LCSs to escape from sub-optimal solutions.

At the end of its execution {10}, *ELF* selects the best, saved rulebase, i.e., the one that had the higher performance value when it was saved. If some of its sub-populations still have more than one rule, the rulebase should be considered as equivalent to many, potential FLCs, one for each possible rule combination, obtained by taking one rule from each sub-population. *ELF* generates the final FLC, collecting only the best rules, i.e., the rules that have the highest strength in their sub-populations.

3.3 REINFORCEMENT DISTRIBUTION IN *ELF*

In this Section we discuss in detail the reinforcement distribution aspects, since these are relevant to understand how *ELF* can evolve sets of fuzzy rules also when the reinforcement is *sparse* and *delayed*.

The strength of a rule (s_r, in the formula below) is updated by the function:

$$s_r(t) = s_r(t-1) + \left(reinf(t) - s_r(t-1)\right) * \frac{cc_r(t)}{pc_r(t)}$$

In other terms, the rule strength is incremented by a quantity proportional to the difference between the present reinforcement (*reinf*) and the past strength, multiplied by a *learning rate.* This is the ratio between the contribution of the rule to the actions performed in the current episode (cc_r), and the parameter pc_r. The current contribution (cc_r) is a number stating how much the actions proposed by the rule have contributed to the actions done in the episode. More formally:

$$cc_{\hat{r}} = \frac{\sum_{s \in S(e)} \mu_s(\hat{r})}{\sum_{\substack{s \in S(e) \\ r \in R(e)}} \mu_s(r)}$$

where $\hat{r}$ is the rule under examination, *s* is a state belonging to the set of states *S(e)* visited during the episode *e*, $\mu_s(r)$ is the degree of firing of rule *r* in the state *s*, and *R(e)* is the set of all the rules that fire during the episode *e*. For instance, suppose we have an episode lasting two control cycles, where a rule r_1 matches both the fuzzy states visited by the agent during the episode, respectively with degrees 0.5 and 0.7, and another rule r_2 matches the same states respectively with degrees 0.5 and 0.3. Then, cc_{r_1}= (0.5+0.7) / (0.5+0.7+0.5+0.3). The cc_r factor takes into account the nature of the fuzzy inferential algorithm. Because of this, a rule contributes to the global action proportionally to its degree of firing, which, in turn, is proportional to the degree of matching with the current state.

The other component of the learning rate (pc_r) is updated at each rule activation by adding the current contribution (cc_r) to the old value of pc_r, up to a given maximum, named *EnoughTested*; a typical value for *EnoughTested* is in [10..20]. Thus, the learning rate evolves during the life-time of the rule: it is a weighted average until the rule has been sufficiently tested; then, the weight of the new contributions (pc_r), becomes constant. In other terms, the reinforcement taken by the rule at the end of an episode is more important at the beginning of its activity, thus supporting a fast assessment of the rule strength.

ELF may also reinforce rules triggered during past episodes. The agent designer may state that there is some *correlation* (represented as a value in [0..1]) between an episode and the previous ones. He or she may consider that, for a given behavior, a state depends not only on what happened during the current episode, but also on the history. The rules triggered in a past episode e_{t-j} receive a reinforcement at time *t*, given by:

$$s_r(t) = s_r(t-1) + \left(reinf(t) - s_r(t-1)\right) * \frac{cc_r(t-j)}{pc_r(t-j)} * decay$$

where *reinf* is the reinforcement obtained at the end of the current episode, $\frac{cc_r(t-j)}{pc_r(t-j)}$ is computed with respect to episode e_{t-j} (occurred *j* time steps before the current episode e_t), and *decay* is a value given by:

$$decay = correlation^n$$

where *n* is the number of episodes from episode *e* to the current one. This mechanism tends to evolve rules that bring the agent through a chain of states to a goal state, and it should be used when the reinforcement is *sparse* or *delayed*. It is similar to the *eligibility trace* mechanism (Sutton, 1984).

As it emerges from the above description, this reinforcement distribution algorithm belongs to the *Temporal Difference* class of algorithms (Sutton, 1988), and it is quite different from the *Bucket Brigade* algorithm typical of *LCS* (Holland, 1985), (Booker, 1988). In particular, in *ELF*:

- rules do not have to bid to have the right of participating in the action; they are selected with a likelihood proportional to their strength which is left unmodified until the end of the episode;
- rules do not pay any fraction of their strength to rules triggered before; the reinforcement is distributed to all the rules, according to the formulae given above.

There are strong relationships between Q-learning (Watkins and Dayan, 1985) and the *ELF* reinforcement distribution algorithm. ELF is a fuzzy extension of Q-learning, conceptually similar to Fuzzy Q-Learning (Glorennec, 1994). A complete discussion about this topic is presented in (Bonarini, 1996b).

4. THE TASK

The task we present in this paper is the classical *prey-predator* task: a predator should catch a prey that runs in a closed environment. In our case, the predator has sensors for direction and distance from both the prey and the closest wall. The prey moves either randomly or with a pre-defined strategy, and it can only sense contacts with walls and obstacles, but not the predator. Obstacles may prevent the predator from detecting the prey; thus, there are situations where the predator loses contact with the prey, and should look for it.

The predator should learn to catch the prey. This task is quite trivial when there are no obstacles and when the predator is faster than the prey. In the presence of obstacles, the predator should develop a strategy to avoid them, and look for the prey when it is hidden. If the prey is faster than the predator, this should develop strategies more effective than simple chasing. Therefore, relative speed is a critical parameter for strategy selection, and, consequently, for improving the performance. However, no sensor is available to detect the relative speed directly. It is even more critical if we consider the real agents we have used: battery consumption is almost irrelevant to the prey's speed, whereas it may reduce the predator's speed by more than 40% of its maximum value. Therefore, the predator should adapt its strategy also when the prey is constantly moving.

The *predator* is *CAT* (*Completely Autonomous Truck*) (Bonarini, 1996a), depicted in figure 5.

*** INSERT FIGURE 5 ABOUT HERE ***

It is based on a toy truck, about 60 cm long, and 60 cm wide. CAT can move forward and back at a maximum speed of 0.3 m/s, and its two steering, front wheels can turn by a maximum of 30 degrees. CAT has a 60 cm high, rotating turret, moved by a stepper motor. The turret holds 4 Polaroid, ultrasonic sensors that are set to detect the distance from objects between 20 cm and 3.50 meters. In this application, they are used to detect the distance from the closest wall in front of CAT. The sensors have an average error of 3% with respect to a reflecting surface perpendicular to the sonar axis; in practice, the accuracy is lower due to the interpretation of sonar data. A sonar echo comes back following the reflection law. The sonar beam can be roughly approximated by a cone about 30˚ wide. Any surface perpendicular to any ray belonging to this cone may potentially reflect the signal. To reduce errors, any signal received is considered as coming from a surface perpendicular to the center of the cone. The worst situation is the one depicted in figure 6, where the first reflected ray received by the sonar is on the border of the cone, on the left.

*** INSERT FIGURE 6 ABOUT HERE ***

This brings another maximum contribution to the error of about 3%. Therefore, the maximum error is about 6%. The fuzzy controllers we consider in this paper are, in principle, robust enough with respect to errors of this magnitude, as summarized in Section 2. Moreover, for the task we consider here, the distance measured by the sonar is relevant for detecting close objects, when the absolute error is negligible, since it is reduced to few millimeters. In the application we present here, we have verified that this type of errors does not affect the performance significantly.

The rotating turret also holds an infrared receiver that can detect the distance from beacons that emit infrared signals at a given frequency. The beacons can be positioned from 30 cm up to 6 meters from the sensor. Combining the information that comes from the infrared receiver with information about the absolute rotation of the turret, it is also possible to detect the direction of the beacons with a precision of 3.5 degrees. A system based on a Fuzzy ARTMAP neural network (Carpenter et al., 1992) gives an interpretation of the infrared intensity as a relative distance (Bonarini, in press). The use of a fuzzy neural network to interpret the *signal intensity* as a *distance* between CAT and the prey is a solution to the problem of the identification of the non-linear relationship between these two variables. Fuzzy ARTMAP is a supervised network based on the Adaptive Resonance Theory (Grossberg, 1976) that can be trained on real-valued inputs and their respective classification, defined by the designer. In our implementation, we use as input the infrared intensity, and as output its fuzzy classification as a distance. The network is trained off-line, and provides a stable and reliable classification; since its accuracy is about 90%, this interpretation has not been subject to further adaptation on-line. During operation, the network receives as input the intensity of the infrared signal,

and produces as output a fuzzy interpretation (<*label*, μ>) that can be fed directly to a fuzzy controller. We have mentioned this solution here just to show how any type of signal interpretation can be integrated with our approach.

CAT also has boolean bumpers implemented by switches behind the eight plates delimiting its body.

Finally, the last of the CAT's sensors we have used in this application are the two encoders, mounted on independent wheels, coaxial with the rear, traction wheels. The encoders give the possibility to estimate some parameters of the model of CAT used to learn the control system.

The *prey* is *SPOT* (*Special Purpose cOoperative agenT*), another small (20 cm of diameter) robot shown in figure 7.

*** INSERT FIGURE 7 ABOUT HERE ***

It has two independent wheels, and it may travel at a maximum speed of 0.2 m/s. It has a 20 cm high turret that holds infrared emitters implementing a 360° wide beacon. Moreover it has bumpers providing real-valued information about the direction of the contact, so that SPOT can easily follow any wall.

SPOT has no sensors to detect CAT, so, in the application we discuss in this paper, it plays the role of a blind prey. It moves according to a pre-defined strategy, selected among three strategies:

- wander randomly;
- follow a fixed, circular trajectory;
- follow the wall.

We will show that the final performance of CAT is qualitatively the same for any of the three SPOT's moving strategies.

5. BEHAVIOR DECOMPOSITION AND CONTROL ARCHITECTURE

To learn the *complex behavior* for the task described in the previous Section, we adopt a control architecture based on different modules, each implementing a *simple behavio*r. Each behavioral module is a set of fuzzy rules. We have decomposed the complex behavior in simpler modules, according to the behavior decomposition strategy that we present below.

5.1 BEHAVIOR DECOMPOSITION

The decomposition of a complex behavior is common practice to reduce the complexity of design (possibly supported by learning) of control systems for autonomous agents. Here, we mention only some of the significant approaches.

In the *subsumption architecture* (Brooks, 1986), behaviors are layered and a network of *inhibition* and *suppression* links among behaviors determines the action to be taken in a given situation. On the opposite side, Arkin's *motor schemata* (Arkin, 1992) produce action vectors that are combined to generate a local *potential field*, whose gradient is followed by the agent. Arbitration is explicitly excluded. On a similar line are the *fuzzy behaviors* proposed by Saffiotti et al. (Saffiotti et al., 1995) who use fuzzy operators to produce an analogous field. Mataric (Mataric, 1995) proposes two types of combination: *complementary* behaviors combine their outputs with pre-defined weights, whereas only one of the mutually exclusive outputs of *contradictory* behaviors is selected using sensorial and internal data.

In the reinforcement learning community, the subsumption architecture has been adopted to learn behaviors organized in a pre-defined framework (Mahadevan and Connell, 1992). In this approach, the final behavior is *functionally* decomposed. ALECSYS (Dorigo and Colombetti, 1994) learns both basic behaviors and coordinators organized in a hierarchical architecture, adopting a *shaping* approach analogous to that adopted to train animals (Hilgard and Bower, 1975). The *functional* decomposition is here structured in different types of hierarchies. The shaping procedure is also adopted by Singh (Singh, 1992) to learn task sequences. In this case the final behavior is *temporally* decomposed into a chain of simpler behaviors.

In almost all the mentioned approaches, it seems that the decomposition strategy derives from an *ad hoc* analysis of the specific task, aimed at the identification of *functional* roles, possibly at different hierarchical levels.

Here, we propose a general approach to behavior decomposition that appears to improve the functional decomposition. Our approach considers general features of the problem that could be found

in many applications. In particular we analyze: the relationships among the considered variables, the classification of these variables according to their temporal characteristics, the complexity of the behavior with respect to the input data. The advantage of following a general approach consists of the exploitation of already identified solutions to common problems. Now, let us discuss how the mentioned features may influence the partitioning strategy.

If it is possible to identify a relationship between a subset of the input variables and a subset of the goals, it is desirable to consider a behavioral decomposition that limits the interaction between variables, and so, the complexity of the behaviors. We call *Strong Functional Behavior Decomposition (SFBD)* a behavior decomposition where behaviors observe different variables and have different goals. In our example, we may consider behaviors concerning the prey (*Chase_Prey*), others concerning walls and obstacles (*Avoid_Contacts*), others that affect actions without considering sensorial inputs (*Wait*). Among the interesting features of the modules coming from this type of decomposition, we mention the possibility to learn the different modules in completely different environments. This may reduce the complexity of learning and, also, the resource needs to learn the different modules. For instance, to learn to avoid contacts, we don't need to have a prey. Notice that this approach is on the same line of the one taken in classical design of control systems, where it is preferred to identify a certain number of *SISO* (*Single Input - Single Output*) modules, hypothesizing a scarce interactivity among their variables, rather than a single, more complex, *MIMO* (*Multiple Input - Multiple Output*) system.

Another type of behavioral decomposition concerns memory of past values. A behavior may either consider only the current input, or also its difference with respect to a value taken at a previous time. We call the behaviors that have in input the difference between a previous value and the current one, Δ behaviors. The corresponding decomposition is a *Δ Behavioral Decomposition* (*ΔBD).* For instance, to implement the *Chase-Prey* behavior, we have two interacting control systems, one of which has as inputs and outputs the variations of the variables considered by the other:

- a basic, *stimulus-response* (*s-r*) module that observes relative direction and distance of the prey and controls speed and steering of the predator;
- a Δ control module that observes variations of relative direction and speed of the prey (i.e., the difference between the current and a past value), and gives variations (increments) to the speed and steering proposed by the *s-r* behavior.

Also the *ΔBD* decomposition is in the spirit of the classical *Control Theory*. In fact, the most typical control system (the so-called *PID*) is based on the observation of the *error* (the difference between the current value and a reference), its *derivative*, and its *integral*, and all these components play well-known, qualitative roles to achieve desired features of the control. A Δ control module, such as the one mentioned above and included in the architecture of the system we are presenting here, plays a role similar to the *derivative* module, giving the possibility to predict the course of its observed variables, and to act accordingly.

The last aspect of our decomposition approach concerns complex behaviors. In this case, the agent may need to exploit different *strategies*, whose selection may depend not only on the low-level, sensor variables, but also on higher-level information such as the evaluation of the current performance. Therefore, we may have low-level behaviors that interact directly with each other, and high-level behaviors, where an *arbitrator* decides how to use lower level modules, by observing high-level variables. In our example, we have adopted such an *arbitrator* to enable the predator to stop and wait for the prey, when it realizes that running after the prey is not successful. We consider this kind of decomposition when we observe that data coming from the sensors are insufficient to decide when the current strategy is not appropriate. In general, we have observed that, as the task becomes complex, the agent needs more insight to take a decision, and it should observe higher level features describing the situation.

5.2 THE CONTROL ARCHITECTURE

The control architecture that we have identified to face the global task is shown in figure 8. We have two modules for the *s-r* and the Δ aspects of the *Chase_Prey* behavior. The *Avoid_Contacts* behavior is implemented by an independent module that combines directly its actions with *Chase_Prey*. The

Avoid_Contacts behavior observes the distance from the closer wall and the relative direction of its position, and proposes a steering action to avoid the contact.

All behaviors are implemented by sets of fuzzy rules, except for the *Wait* module, which is trivial. They are learned by *ELF* following the learning procedure described in the next Section.

*** INSERT FIGURE 8 ABOUT HERE ***

Another control system, at the *strategic level*, monitors the performance of the predator and switches among two possible *strategies*: *Wait_for_the_prey*, implemented by the only behavior *Wait*, and *Chase_the_prey_safely*, implemented by the combination of *Chase_Prey* and *Avoid_Contacts*. Notice that this *arbitrator* observes variables that have a nature completely different from the others: whereas all the other modules observe state variables more or less related to sensors, the arbitrator observes a sort of *performance evaluation variables*, namely the sign of the difference between the last and the current performance value, the number of episodes since the last change of this sign, and the current strategy.

The arbitrator is implemented by a set of fuzzy rules, and it is learned by *ELF*. In this case, only one of the three input variables is classified by standard fuzzy sets (the second of the above list), whereas the other two are fuzzyfied, binary variables. This is an example of fuzzy models used to represent in a uniform way both fuzzy and crisp variables.

Comparing this architecture with the proposals mentioned in the previous Section, we may say that we integrate a vectorial combination with a hierarchical structure. Moreover, the arbitrator implements a mechanism similar to the *inhibition* mechanism proposed in the subsumption architecture, but keeps the sensor variables separate from variables evaluating the performance. Finally, this type of arbitrator could be used also to weight dynamically the output from different behaviors, as we have done in another application (Bonarini, 1994b). Apart from the fuzzy implementation, the main difference with respect to the hierarchical approach proposed by Dorigo and Colombetti (Dorigo and Colombetti, 1994) is the possibility to coordinate behaviors by observing high-level variables independent from the lower-level behaviors. This becomes important when the arbitrator needs to compose strategies, or very complex behaviors. In this case, the information elaborated from the only input may not be sufficient to provide interesting and reliable coordination.

6. THE LEARNING SESSION

Following the above mentioned behavior decomposition, we have partitioned also the learning task in simpler subtasks. This makes it possible to learn effectively the simpler behaviors. Learning a unique, global behavior that takes into account *s-r*, Δ, and strategic aspects would be a hard task, since the learning system has to consider many variables at the same time, and this makes the complexity growing exponentially (Kaelbling et al., 1996), (Dorigo and Colombetti, 1994). Moreover, we adopt a learning schema, which is efficient, but it can only be applied in simulation. In this Section, we describe in detail the learning activity performed by *ELF* in the simulated environment. It is based on four techniques: *comparative learning*, *strategy learning*, *transfer of policy*, and *learning from easy missions*. All of them implement different aspects of a general form of *shaping* for our agent. Finally, we discuss the *adaptation* of the model, used in simulation, to the real world, discussing the model identification issues that complete our anytime learning approach.

6.1 COMPARATIVE LEARNING

To learn the *Chase-Prey* behavior we firstly apply *ELF* to learn the *s-r* control system. Then, to improve the overall performance of the agent also with relatively fast preys, *ELF* learns the Δ behavior mentioned above.

We have decided to learn the Δ behavior by considering as reinforcement the increment of performance with respect to the *s-r* behavior (*comparative learning*). In order to do that, we first learn the *s-r* behavior, then we simulate two agents facing the same initial, randomly selected situations: the first one is controlled by the *s-r* behavior, whereas the second is controlled by the composition of the *s-r* control system and the Δ. The performance of both the agents is evaluated.

In particular, we evaluate the performance of the *s-r* behavior at time *t* as:

$$p_{s-r}(t) = 1 - \frac{d_{s-r}(t)}{d_{RIF}}$$

where $p_{s\text{-}r}$ is a performance index, $d_{s\text{-}r}$ is the relative distance between prey and predator, and d_{RIF} is a normalizing constant. The reinforcement program produces a reinforcement proportional to $p_{s\text{-}r}$, thus reinforcing the approaching to the prey. Analogously, we evaluate the performance of the Δ behavior as

$$p_{\Delta}(t) = 1 - \frac{d_{\Delta}(t)}{d_{s-r}(t)}$$

where d_{Δ} is the relative distance between prey and the predator showing the Δ behavior. In other terms, we evaluate the relative performance of the Δ agent with respect to the *s-r*.

To learn the Δ control system the reinforcement program produces a reinforcement proportional to p_{Δ}.

As mentioned in Section 3, *ELF* evaluates at the end of an *episode* the performance of the population it is evolving. In this case, each episode is a sequence of control cycles terminated by a condition discussed in Section 6.3.

Comparative learning is interesting in a *shaping* activity, since it makes it possible to use the same type of performance evaluation during the development of different, increasingly complex behaviors for the agent. The agent running with the old control system acts as a reference for the new agent, which should behave better than its ancestor.

6.2 LEARNING BEHAVIORAL STRATEGIES

We have adopted comparative learning also for the next step of our shaping procedure, to learn the arbitrator. In this case, the comparison is made between an agent controlled by the Δ behavior mentioned above, and another agent controlled by the arbitrated composition between it, and the *Wait* behavior. The arbitrator monitors the performance of its agent, and proposes to activate one behavior or the other.

In particular, it receives at each control step the sign of the difference between the last and the current performance evaluation, the number of evaluation cycles since the last change of this sign, and the current strategy. In output, it gives the selection of one of the two strategies.

The performance of the *strategic* behavior is evaluated by:

$$p_{S}(t) = 1 - \frac{d_{S}(t)}{d_{\Delta}(t)}$$

where d_s is the distance at the end of the episode between the prey and the predator controlled by the strategic behavior.

For other tasks, with a different control architecture, *ELF* has learned also an arbitrator that weights the output of several behavioral modules (Bonarini, 1994b).

6.3 TRANSFER OF POLICY

Another interesting technique that we have adopted in our learning session concerns learning behaviors in simplified situations, so that the learned behavior could be successful also in the final, more complex environment.

In our task, we have learned each behavior in a simplified setting where the prey goes from a side to the other of the playground, and the predator is initially placed in a fixed position, with a random orientation (figure 9).

*** INSERT FIGURE 9 ABOUT HERE ***

In the figure, the prey is represented as a circle, the predators as triangle-marked, rounded boxes. The darker predator on the right shows an *s-r* behavior, whereas the lighter has a Δ behavior. They have been put both in the same initial position. We may notice typical, different behaviors when facing the obstacle. The *s-r* agent follows the direction between its current position and the prey; the Δ agent, once recognized the prey movement, anticipates it and succeeds in catching it also in presence of an obstacle.

This behavior is successfully applied also in the real environment where the prey moves in a closed room, and the obstacle is U-shaped, as reported in Section 7.1. Learning directly in the closed room, with a prey moving in two dimensions had introduced unnecessary, potential problems concerning the relative positioning at the beginning of the trial, the turning capabilities of the prey, the length of the trial, and many other aspects. In the simpler situation shown in figure 9, the trial terminates either when the predator reaches the prey, or when the prey comes at the end of its track. An episode terminates on these two conditions, or when the predator looses contact with the prey, hidden by the obstacle. In this case, the already learned *Avoid_Contacts* behavior is applied and the trial continues.

6.4 LEARNING FROM EASY MISSIONS

A similar learning schema is adopted to learn the *arbitrator* at the strategic level: the episode termination is the same, and the prey's speed is changed from about -30% to +70% of the predator's. This is a particular interpretation of the learning session schema known as *learning from easy missions* (Asada et al., 1994). The agent starts its learning activity on simpler tasks, in our case, chasing a prey with approximately the same speed. Once the simpler task is achieved, then more complex tasks are faced, keeping the experience so far acquired.

6.5 ANYTIME LEARNING

The last consideration about learning concerns *adaptation* to the environment. As mentioned above, we have adopted an *anytime learning* approach to adapt the behaviors learned in simulation to real environments and agents. As mentioned in Section 1.2, the simulated model is updated by a *monitor* module that runs in parallel to the learning system, and detects possible differences between the real world and the current model. We have implemented a *monitor* that uses data coming from the CAT's encoders to estimate the actual speed and steering of the predator, obtained from each control action. The *monitor* module continuously updates a set of tables representing the mapping from the control actions to their actual values. When these become different enough from the expected, the new estimate is sent to the simulation system, which updates its model and asks to the learning system to start again the learning session. The learning system saves the best control system so far obtained running on the old model, since it could be re-used if the model parameters come back to the old values (*case-based learning*). Then, it searches in its *control system base* whether it has been already activated in a situation similar to that described by the current parameters. If this is the case, the retrieved situation is restored, the best control system for that situation substitutes the current one, and *ELF* continues to try to find better control systems for this situation. If the current situation has not been faced yet, the learning system starts to work on the new model, evolving a population from where the 30% of the rules of the previous population have been randomly deleted. As the performance on the new model rises over a given threshold, the new control system substitutes the current one, on board.

7. EXPERIMENTAL RESULTS

We have studied a variety of possible speed combinations, and we have observed, at the end of all the trials, three, qualitatively different behaviors for the predator:

1. when the prey is slower than the predator, the predator just moves at each control cycle in the direction of the prey;
2. when the prey has a speed comparable with that of the predator (±5%), CAT considers also the speed of the prey in order to guess its movements and to try to optimize its trajectory (e.g., it points to an estimated future position of the prey, "cutting" corners of its trajectory);
3. when the prey is faster than the predator, CAT decides to stop and wait until SPOT is close enough (SPOT cannot sense CAT) to jump on it.

The behavior obtained for the predator makes it reaching a prey up to 40% faster than itself in more than the 90% of the trials, in a time less than that required to perform 200 control cycles, given environments such as the one shown in figure 10, where both prey and predator have random, initial position and orientation. The tracks in figure 10 are taken from the simulator, but they are analogous to the real ones.

*** INSERT FIGURE 10 ABOUT HERE ***

The situation shown on the right of the figure deserves some comments. The prey is faster, so, after a short unsuccessful activity, the predator stops and waits at point "X" (you may notice that its track is shorter than that of the prey). At this moment the arbitrator has decided to change strategy since the previous one was not effective. In this case, in a while the prey becomes visible again and goes in the direction of the predator. This is the only case where it can catch a faster prey, so CAT starts moving in the prey direction as it is close enough. Notice that this situation is similar to what happens in nature with animals with characteristics analogous to our agents; for instance, frogs wait for they preys, since these are faster.

Before giving some numerical results about our experiments, we would like to discuss some typical situations that may arise when only the *s-r* or the combination of the *s-r* and the Δ behaviors are adopted. We present paradigmatic situations that show general properties.

On the left of figure 11, we show the *s-r* behavior of a predator chasing a faster prey that runs on a square track. The best thing it can do to optimize the reinforcement, is to try to keep its distance to the prey as small as possible, thus taking a track internal to that of the prey, at a distance proportional to the difference between the two speeds.

*** INSERT FIGURE 11 ABOUT HERE ***

On the right of figure 11, we show the Δ behavior in the same situation. In this case, the observation of the relative speed of the prey gives to the predator the possibility to catch it by pointing to an estimated, future prey position, thus taking also a shorter track in the angles.

If the prey is even faster also this strategy is not enough. A qualitatively different strategy should be considered. A possibility may be to try to identify a high-level description of the behavior of the prey, such as: "it is running on a square track". This problem is not trivial if we consider that we may have many different preys, showing a large variety of behaviors, and that the sensors of the predator give it only a rough, deictic view of its environment. Moreover, once identified the behavior of the prey, the predator should be able to adapt to it, by adopting the most appropriate action (for instance, in the case of our example, it may decide to wait for the prey at a position on its regular track). We are working on this research topic, by providing the predator with some classification capabilities, but we only have preliminary results.

In the rest of this Section, we present experimental evidence that supports our approach, both for what concerns the control architecture, and anytime learning. All the results reported in Section 7.1 concerns trials on a simulated environment, aimed at demonstrating the effectiveness of the proposed architecture, coming from our decomposition approach. In Section 7.2, we report about experiments done in a real environment, to show the validity of the anytime learning approach and of the adaptation mechanism.

7.1 VALIDATION OF THE ARCHITECTURAL CHOICES

The goal of the first experiment we present is to compare a *monolithic* control architecture (consisting of only one rule base) with our decomposed architecture. We focus on the aspects concerning the learning activity that, in our approach, is performed on a simulated environment. In this experiment, the prey runs at a speed at most 5% higher than that of the predator, and the predator is controlled by a system having only the *s-r* and the Δ modules. There are no obstacles in the environment. In this case, we had four input variables (d_p and α_p and their variations), and two control variables (V and θ). These variables have respectively, 5, 4, 3, 3, 6, and 5 values (fuzzy sets) each, giving a search space of 5400 rules for the monolithic control system. Considering the decomposition presented in Section 5, we have only 600+270=870 possible rules. The experimental results confirm the theoretical expectations: in 12 trials, lasting 20,000 control cycles each, only one rule base has been saved by *ELF* learning the monolithic architecture. This means that in only one case *ELF* has been able to learn a reliable, good enough, monolithic rule base. With the architecture presented above, *ELF* has learned in the same total number of control cycles more than 7 Δ modules for each of the first 5 *s-r* modules learned. The first valid combination was available at a cost of only 910 control cycles. This result is analogous to that obtained by Dorigo and Colombetti (Dorigo and Colombetti, 1994): a structured architecture contains knowledge, provided by the designer with the decomposition activity, that should be learned when adopting a monolithic architecture.

In a second experiment, we have added the obstacle shown in figure 9, and we have compared the performance and learning time of the *s-r* and the Δ behaviors. In table 1, we summarize the results obtained in 12 trials lasting 5,000 control cycles each.

*** INSERT TABLE 1 ABOUT HERE ***

In the first pair of columns we give the average number of rules for the two final rulebases, and its standard deviation. In the second pair, the number of control cycles to obtain the first, good, stable rulebase. In the third pair of columns we report the minimum distance from the prey obtained in 12 test trials lasting 1,500 control cycles each, by the only *s-r* behavior and by its combination with the Δ. We may notice how at a relatively low learning cost (the first rulebase for the Δ behavior has been saved, in average after only 789 control cycles), we can obtain a relevant improvement of the behavior (more than one order of magnitude for the final distance to the prey).

In a third experiment we have studied the performance of the predator controlled by an architecture with or without the arbitrator. In table 2, they are respectively referred to as *A* and *B*. We have compared the performances of the two architectures in trials of 1,000 episodes each, with a prey having the average speed reported in table 2, and the predator having a speed of 0.11 m/s.

*** INSERT TABLE 2 ABOUT HERE ***

In table 2, the first performance index (*Wins*) is the number of times an agent has performed better than the other, either by catching the prey or getting closer to it at the end of the episode. The second performance index (*minD)* reported in table 2 is the minimum distance to the prey achieved at the end of the episode by each of the two agents.

As shown in table 2, the agent controlled by the most complex architecture (*A*) performs better than the other (*B*) when the prey is faster. The only difference between the two is the presence of the arbitrator, which have learned to understand when following the prey becomes useless, and when it is better to wait for it. With these results, we demonstrate that, even if the strategy of waiting for a faster prey may seem trivial, it appears to be effective.

7.2 EXPERIMENTS CONCERNING ANYTIME LEARNING

In this Section we present results of experiments performed in the anytime learning framework, with the learning system running on the on-board PC of CAT. So, *ELF* learns in simulation, while CAT acts in the real environment, chasing SPOT with the best so-far learned control system. The model used to learn in simulation is improved on line.

In our specific setting, CAT has very imprecise mechanical features. For instance, the front position of the steering system cannot be calibrated once for all in its central position, due to the imprecision of the mechanical links. Moreover, some features also vary in time. The most critical for this task is the maximum speed, which decreases almost linearly to slightly less than 60% than its initial value, with the battery consumption.

We have done experiments with SPOT either moving randomly, or following a circular trajectory, or going straight to the first wall of a rectangular room, and then following it. The respective average performance, computed as the minimum distance to the prey over 20 episodes of at most 200 control steps each (i.e., about 3 minutes each) is given in table 3. At the end of each episode (i.e., either when CAT touches SPOT, or after 3 minutes of activity) the agents are separated and put in positions randomly chosen by a computer, to avoid the experimenter's influence.

*** INSERT TABLE 3 ABOUT HERE ***

We consider an episode successful, when CAT touches SPOT before the end of the episode.

As you can see, the performance is almost independent from the blind movement strategy of SPOT, whereas the wall following behavior seems to be slightly convenient for the prey. In this case, CAT may come too close to the wall before touching SPOT, so that its *Avoid_Contacts* behavior brings it away from both the wall, and SPOT. We have reduced this effect by putting the fuzzy set corresponding to the interpretation "*too close obstacle*" at its minimum admissible value. A more conservative choice would have given SPOT a safe place around any wall. We discuss in the last Section the possibility to dynamically adapt the definition of the fuzzy sets.

The maximum CAT's speed with fresh batteries is set to 0.25 m/s, and spontaneously decreases with battery consumption to about 0.15 m/s in about 50 minutes. At that point, the batteries should be changed with fresh ones. The SPOT's speed is about 0.2 m/s at the beginning and drops suddenly to 0 after about one hour. The performance reported in table 3 concerns trials done in randomly chosen

conditions, and it should be considered as the CAT's average performance of that part of its life, where it had the need to chase either a slower and a faster prey.

To test the relevance of adaptation, we have performed a set of trials without any adjustment of the model, thus excluding the activity of the *monitor*. In this case, we have considered only random movements of SPOT. After about three hours of trials (and three battery packs to be recharged), the average minimum distance at the end of the episode was 78 cm, i.e., approximately 30% worst than what obtained with model adaptation. Mostly important, the failure/success ratio rose from 0.051 to 0.1. These results give an idea of the importance of adapting the model used in simulation to the dynamic environment.

8. CONCLUSION

Our goal was to demonstrate the feasibility of reinforcement learning to develop complex behavior for an embodied autonomous agent. We have faced this problem by considering different aspects to reduce learning time as much as possible: the *algorithm*, the *learning session*, the *control architecture*.

We have implemented an *algorithm* that operates on *niches*, thus focusing the learning activity only on the part of the search space that is being explored. Moreover, we have adopted a fuzzy representation for the control modules that makes it possible to treat *classified, real-valued input* and *output*, thus reducing the search space. The fuzzy classes overlap, thus producing desirable effects on the produced controller, such as robustness, and smoothness of action.

Our *learning session* is based on *shaping* and it exploits learning techniques that reduce the complexity of the environments on which the agent learns, thus contributing to reduce the time spent in this activity.

The *control architecture* is based on a partition of the whole control system, according to a *methodological framework* that identifies different control modules to implement simpler behaviors that:

- operate on different variables, such as the *Chase_Prey*, and the *Avoid_Contacts* modules presented in Section 5;
- operate on variables and their variations, such as the *s-r* and the Δ modules implementing the *Chase_Prey* behavior.

Moreover, in the framework we have proposed, the outputs of the control modules may be composed either by *static* operators, as we did for the *Chase_Prey* and the *Avoid_Contacts* behaviors, or by adopting a higher level module (an arbitrator) that *dynamically* weights the outputs with reference to high level variables. We have shown how this last mechanism can be used also to implement *strategy selection*.

In conclusion, the learning approach we have presented in this paper has been demonstrated as very effective to learn complete behaviors for our autonomous agent, at least in the considered environment. Any testing activity done in real environments has only a relative validity, since the operating conditions cannot be identified precisely, by definition, otherwise learning would not be motivated. However, our approach is also supported by the theoretical motivations presented in the paper.

Future developments include learning more complex interactive tasks for CAT and SPOT. Presently, they have been programmed to clean a room exploiting their characteristic features: SPOT goes around the wall drawing a brush, and brings dust to CAT that is large enough to remove it using a vacuum cleaner. We are now studying the role of learning and adaptation in this type of tasks. In particular, we plan to add a mechanism to implement on-line tuning of the fuzzy sets definitions used in our fuzzy rules. We are studying a system that, basing on the current performance evaluation, and on the analysis of the states that brought the agent to a dramatic drop of performance, propose alternative models not only for the environment and the agent, but also for the fuzzy sets used by the control system.

References

Arkin, R. C. (1992). Behavior-based robot navigation for extended domains. *Adaptive Behavior*, *1*(2), 201-225.

Asada, M., S. Noda, S. Tawaradsumida, & K. Osoda (1994). Purposive behavior acquisition on a real robot by vision-based reinforcement learning. *Proceedings of the MLC-COLT Workshop on Robot Learning*. New Brunswik, NJ: Rutgers University.

Bonarini, A. (1993). *ELF*: learning incomplete fuzzy rule sets for an autonomous robot. *Proceedings of the EUFIT '93*. Aachen, D: ELITE Foundation.

Bonarini, A. (1994a). Evolutionary learning of general fuzzy rules with biased evaluation functions: competition and cooperation. *Proceedings of the IEEE WCCI - Evolutionary Computation*. Piscataway, NJ: IEEE Computer Press.

Bonarini, A. (1994b). Learning to coordinate fuzzy behaviors for autonomous agents. *Proceedings of the EUFIT'94*. Aachen, D: ELITE Foundation.

Bonarini, A. (1996a). Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz (Ed.) *Fuzzy modeling: paradigms and practice*. Norwell, MA: Kluwer Academic Publishers.

Bonarini, A. (1996b). Delayed Reinforcement, Fuzzy Q-Learning and Fuzzy Logic Controllers. In Herrera, F., Verdegay, J. L. (Eds.) *Genetic Algorithms and Soft Computing* (*Studies in Fuzziness, 8*). Berlin, D: Physica-Verlag.

Bonarini, A. (1996c). Fuzzy sets and evolutionary learning: motivations and issues for integrated systems. *Proceedings of the ICML '96 Pre-Conference Workshop on Evolutionary Computing and Machine Learning*. Bari, I: University of Bari.

Bonarini, A. (in press). Symbol grounding and a neuro-fuzzy architecture for multisensor fusion. *Proceedings of the WAC-ISRAM '96*. Albuquerque, NM: TSI press.

Booker, L. B. (1988). Classifier systems that learn their internal models. *Machine Learning*, *3*, 161-192.

Booker, L. B., D. E. Goldberg, & J. H. Holland. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence. 40* (1/3), 235-282.

Brooks R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*. *2* (1), 14-23.

Brooks R. A. (1990). The Behavior language; user's guide. *AI Memo 1227*. Boston, MA: AI Lab, MIT.

Brooks, R. A., & M. J. Mataric (1993). Real robots, real learning problems. In Connell J.H. and S. Mahadevan (Eds.), *Robot learning*. Boston, MA: Kluwer Academic Publishers.

Carpenter, G. A., S. Grossberg, N. Markuzon, J. H. Reynolds, & D. B. Rosen. (1992). Fuzzy-ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks.* (3), 698-712.

Dean, T., & M. Boddy. (1988). An analysis of time-dependent planning. *Proceedings of the Seventh National Conference on AI (AAAI-88)*. San Mateo, CA: Morgan Kaufmann.

Dorigo, M., & M. Colombetti. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*. *71* (2), 321-370.

Dubois, D., H. Prade. (1980). *Fuzzy sets and systems: theory and applications*. New York, NY: Academic press.

Glorennec, P. Y. (1994). Fuzzy Q-learning and dynamic fuzzy Q-learning. *Proceedings of the Third Fuzz-IEEE*. Piscataway, NJ: IEEE Computer Press.

Goldberg, D. E. (1989). *Genetic Algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Grefenstette, J. J. (1992). An approach to anytime learning. *Proceedings of the ninth International Workshop on Machine Learning (ML92).* San Mateo, CA: Morgan Kaufmann.

Grefenstette, J. J. (in press). Genetic Learning for adaptation in autonomous robots. *Proceedings of the WAC-ISRAM '96*. Albuquerque, NM: TSI press.

Grossberg, S. (1976). Adaptive pattern classification and universal recording. I: parallel development and coding of neural feature detectors. *Biological Cybernetics*. *23*, 121-134.

Hilgard, E. L., & G. H. Bower. (1975). *Theories of learning*. Englewood Cliffs, NJ: Prentice-Hall.

Holland, J. H. (1985) Properties of the bucket brigade algorithm. *Proceedings of the first International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Kaelbling, Pack L., M. L. Littman, & A. W. Moore. (1996). Reinforcement Learning: a survey. *Journal of Artificial Intelligence Research*. *4*, 237-285.

Kirk, D. E. (1970). *Optimal control theory: an introduction*. Englewood Cliffs, NJ: Prentice-Hall.

Mahadevan, S. & J. H. Connell. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*. *55*(2), 311-365.

Mataric, M. J. (1989). A distributed model for mobile robot environment-learning and navigation. *Tecnical Report 1228*. Boston, MA: AI Lab - MIT.

Mataric, M. J. (1995). Designing and understanding adaptive group behavior. *Adaptive Behavior*. *4*(1), 51-80.

Mataric, M. J. (In press). Reinforcement learning in the mult-robot domain, *Autonomous Robots. 4*(1).

Pedrycz, W. (1992). *Fuzzy Control and Fuzzy Systems.* New York, NY: John Wiley and sons.

Saffiotti, A., K. Konolige & E. H. Ruspini. (1995) A multivalued logic approach to integrating planning and control. *Artificial Intelligence*. *76*(1/2), 481-526.

Sastry, S., & M. Bodston. (1989). *Adaptive control: stability, convergence, and robustness*. Englewood Cliffs, NJ: Prentice-Hall.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*. *8* (3/4), 323-339.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. PhD Thesis. Amehrst, MA: University of Massachussets.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*. *3*(1), 9-44.

Valenzuela-Rendón, M. (1991). The fuzzy classifier system: a classifier system for continuously varying variables. *Proceedings of the fourth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Watkins, C., & P. Dayan. (1985). Q-Learning, *Machine Learning*. *8*, 279-292.

Whitehead, S. D., & D. H. Ballard, (1991). Learning to perceive and act by trial and error. *Machine Learning*. *7*, 45-83.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Wilson, S. W. (1994). ZCS, a zeroeth level Classifier System, *Evolutionary Computation*. *2* (1), 1-18.

Zadeh, L. A. (1966). Fuzzy Sets. *Information and control*. *8*, 338-353.

Acknowledgments

This work has been partially supported by the Italian Ministry for the University and the Scientific Research, as part of the Project "Development of Autonomous Agents through Machine Learning".

I am also grateful to G. Calegari, R. Cadonati, and E. Cattozzo for the development of this application of *ELF*, and to the many other students that have contributed to the development of *ELF*, Fuzzy CAT, and SPOT.

I have also to thank Maja Mataric, Marco Colombetti, Dan Boley, Maria Gini, and the anonymous reviewers who contributed to the improvement of the first version of this paper.

Figure Captions

Figure 1 - Fuzzy sets and intervals for the classification of values of the variable *Dist*

Figure 2 - Interpretation of the threshold value MINIMUM-SAFE-DIST

Figure 3 - Some behaviors of an autonomous agent in presence of an obstacle.

Figure 4 - The *ELF* algorithm

Figure 5: CAT

Figure 6 - Sonar approximation

Figure 7 - SPOT

Figure 8 - The control architecture. V is the speed of the predator, θ is its steering, d_p and α_p are distance and direction of the prey, d_w and α_w are distance and direction of the closest wall, Z is a block that keeps memory of the last value of its input variable.

Figure 9 - Learning to chase a prey with a speed similar to the predator's, and an obstacle.

Figure 10 - Chasing a slower prey (on the left), and a faster (on the right), in a simulated environment.

Figure 11 - CAT, chasing a faster prey: the *s*-r behavior on the left, the Δ on the right, in a simulated environment.

Figure 1

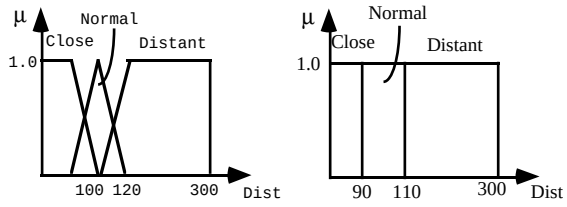


Figure 2

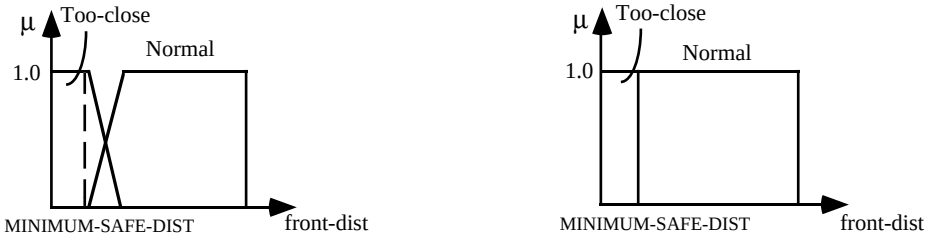


Figure 3

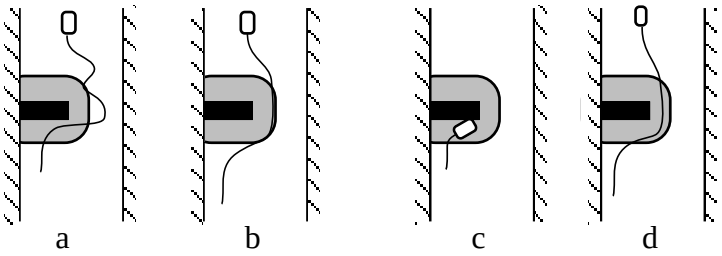


Figure 4

```
1.initialize()
2.while not end_of_trial()
3.  {while not end_of_episode()
4.            {T_rules = select_rules(State)
5.             execute_action(T_rules, State);
6.             State = detect_environment()}
7.  Reinforcement = evaluate(State);
8.  distribute(Reinforcement);
9.  population_update(State)}
10.final_rulebase:=refine(select_best_rulebase());
```

Figure 5

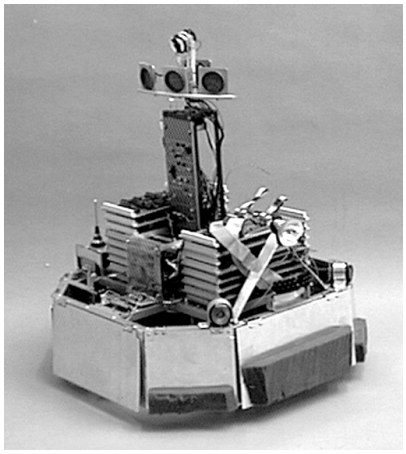


Figure 6

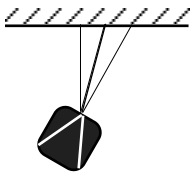


Figure 7

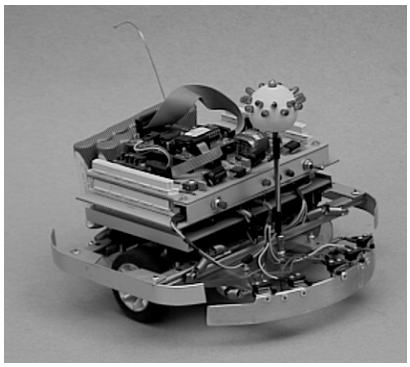


Figure 8

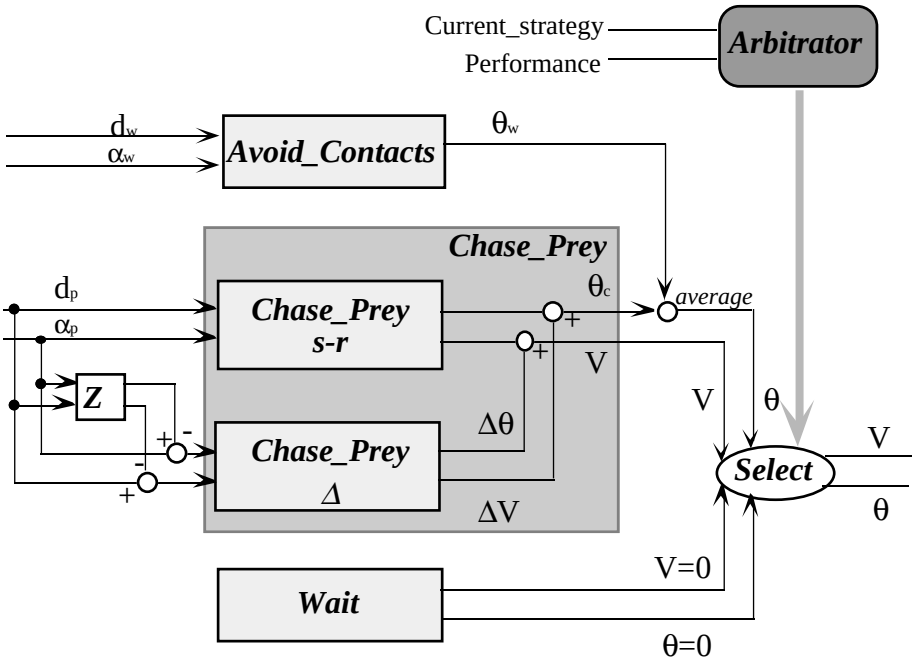


Figure 9

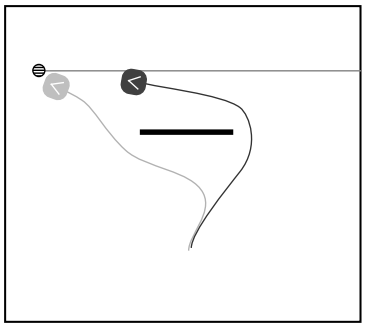


Figure 10

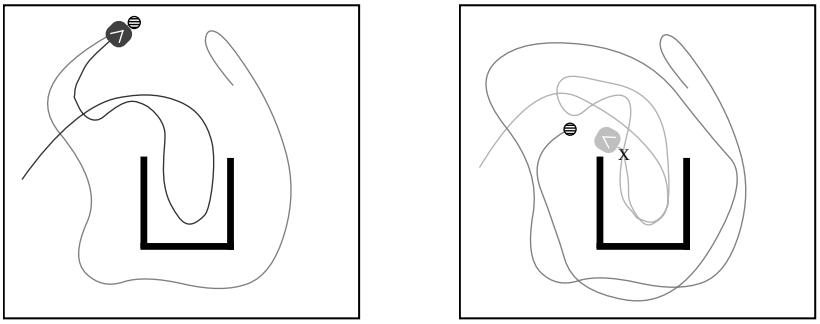
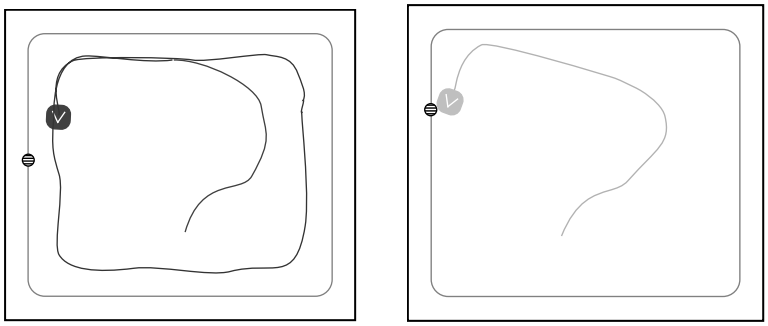


Figure 11



Tables

Table 1 - Comparison among *s-r* and *Δ* behaviors

Rule base type	# of rules		# of cycles 1st base		min distance from prey (in cm.)	
	ave	std	ave	std	ave	std
Δ	11.2	0.83	789	187.3	54.8	8.5
s-r	31.5	1.68	2711	452.8	740	159.3

Table 2 - Performance with (*A*) and without (*B*) a strategic control.

	V_{prey}=0.07				V_{prey}=0.12				V_{prey}=0.2			
	Wins		*minD*		*Wins*		*minD*		*Wins*		*minD*	
	ave	std	ave	std	ave	std	ave	std	ave	std	ave	std
A	553	37	28	11	629	61	96	39	811	36	109	44
B	447	37	45	12	371	61	288	147	189	36	627	180

Table 3 - Performance in the real world

Prey movement	failure/success	minD
random	0.051	53
circle	0.051	46
wall	0.055	67