An Enhanced Classifier System for Autonomous Robot Navigation in Dynamic Environments

José M. Molina, Araceli Sanchis, Antonio Berlanga, Pedro Isasi

Grupo de Vida Artificial. Departamento de Informática.
Universidad Carlos III de Madrid, Spain.
Avda. Butarque 15, Leganés. Madrid.
Fax: +34 1 624 94 30, e-mail: molina@ia.uc3m.es

Abstract: In many cases, a real robot application requires the navigation in dynamic environments. The navigation problem involves two main tasks: to avoid obstacles and to reach a goal. Generally, this problem could be faced considering reactions and sequences of actions. For solving the navigation problem a complete controller, including actions and reactions, is needed. Machine learning techniques has been applied to learn these controllers. Classifier Systems (CS) have proven their ability of continuos learning in these domains. However, CS have some problems in reactive systems. In this paper, a modified CS is proposed to overcome these problems. Two special mechanisms are included in the developed CS to allow the learning of both reactions and sequences of actions. The learning process has been divided in two main tasks: first, the discrimination between a predefined set of rules and second, the discovery of new rules to obtain a successful operation in dynamic environments. Different experiments have been carried out using a mini-robot Khepera to find a generalised solution. The results show the ability of the system to continuous learning and adaptation to new situations.

Key Words : Learning, Reactive Systems, Autonomous Robots, Rule Discovery, Bucket Brigade Algorithm, Genetic Algorithms, Classifier Systems.

1. Introduction

Multiple robotic systems applied in industry are autonomous mobile robots working in stationary environments, i.e. automatic floor-cleaning, automatic assembly, transporting parts in a factory, etc. Other applications of robotic systems involve interactions with dynamic environments, where the autonomous robot deals with unexpected events. The successful operation in such environments depends on the ability of adaptation to the changes. The adaptation ability goes through the introduction of current situation in each decision process.

A fundamental requirement of autonomous mobile robots is navigation. This task gets the robot from place to place with safety and no damage. Approaches based on the classical paradigms (abstraction, planning, heuristic search, etc.) are not completely suitable for unpredictable and dynamic environments. Other approaches consider reaction as the new paradigm to built intelligent systems. One classical instance of this kind of architecture is the subsumption architecture which was proposed by Brooks [3] and has been successfully implemented on several robots of MIT and other institutes.

The base of the subsumption architecture is "behavior". Each behavior occurs in a situation and the global control is a composition of behaviors. The implementation of these behaviors uses different systems, from finite state machines to fuzzy controllers. The rules of these behaviors could be designed by a human expert, designed "ad-hoc" for the problem, or learned using different artificial intelligence techniques [13].

Machine learning has been applied to shape the behavior of autonomous agents in this kind of environment. Some of these techniques become inapplicable to the problem of learning reactive behavior because they require more information than the problem constraints allow. Thus, it would seem reasonable to use an automatic system that gradually builds up a control system of an autonomous agent by exploiting the changing interactions between the environment and the agent itself. Some approaches use Genetic Algorithms to evolve fuzzy controllers [12], Evolution Strategies to evolve connection weights in a Braitenberg approach [26] or Neural Networks to learn behaviors [16].

The learning system proposed in this work evaluate the complete behavior without discriminating among different internal parts, i.e. if the behavior is composed by a set of rules, the evaluation does not discriminate among rules. On the other hand, the system ought to learn new rules, that replace old ones, in order to improve the set of rules as a whole. However, for discovering new rules in isolation, some kind of measure of the accuracy of each rule is needed.

Classifier Systems (CS) [7] are well suited to learn multiple different concepts incrementally under payoff. These systems have been widely implemented and tested for a large number of theoretical problems, [8, 1, 6, 17, 20, 21], but there are not many cases in which they are included in real systems [4, 5, 17].

To survive in a dynamic environment, a system has to possess associations between environmental signals and actions that will lead to satisfy its needs. In a CS, these associations are represented by condition/action rules. Conditions match both environment and internal state, and actions modify the internal state or execute an external action. In general, the learning process in CS shows two main problems:

- Decision Time. In order to produce elaborate solutions, where the rules are interrelated, the decision ought to be taken in several internal cycles. This problem become stronger when CS are applied to problems in which a quick response is needed.
- Rules Chain. CS are able not only to learn rules but to chain previously learned ones. Rules belonging to a chain make no sense in isolation. Then, the loss of a rule in the chain could imply the loss of all the knowledge, due to the high degree of interrelationships between rules.

The principal problem of CS when are applied to reactive problems, as Wilson [21] and Grefeenstette [24] detected, is that during several SC internal cycles, the system gets blind to environmental changes and, furthermore, in dynamic systems these changes happen repeatedly. The solution proposed by these authors does not

allow the chaining of rules, then, each time an environmental input arrives an output is produced by a rule in isolation. The solution outlined by Wilson and Grefeenstette is too restrictive, what produces poor results. Therefore, the use of CS was abandoned, in this type of problems, until Dorigo's works [22] [23]. In these works, several designs are introduced in order to speed up the response of CS. The new CS proposed by Dorigo in [23] is based on: parallelism, a distributed architecture and a special training process. The perspective adopted by Dorigo to solve the reactive problem is the division of the problem in several levels, building a hierarchical architecture, where a set of CS learns to co-operate. Then, the "reactivity" is based on "parallelism": different levels of CS are executed in different machines and, besides, different CS take charge of different tasks. Using these ideas, the response time becomes smaller. However, the system continues being blind to environmental changes during internal cycles.

Another interesting approach [25] employs what the author calls hierarchical "chunking" to the application of CS in reactive systems. The basic idea of this work is as follows: two rules are related when both are executed consecutively. A new aggregated rule (C) is created by two related rules (A and B) in this way: the condition of C is the condition of A and the consequence of C is the response of both in sequence A and B. However, when an aggregated rule is executed, without considering new environmental information, the system gets blind in the same way as in Dorigo's work.

The capacity of the system to facilitate a quick response should not be approached only from techniques that attempt to increase the speed of the process. It can be approached from a different perspective: the introduction of data from the environment at the same time that the CS takes intermediate decisions. In this sense, a modification of the philosophy of CS is proposed, allowing reactions without losing the possibility of rules sequencing. The new CS integrates the environmental input with the internal state of previous input, in order to take a new decision.

In the proposed learning process, the only previous information is about the number of inputs (robot sensors), the range of the sensors, the number of outputs (number of robot motors) and motor description (velocity values). The robot controller starts without information about the proper associations between sensor inputs and motor velocities. And from this situation the robot is able to learn through experience to reach its highest adaptability to the sensor information.

The robot has to use its experience to discover an effective set of rules. The system should not use all its storage capacity for raw experience, so it must be able to extract relevant information from each situation when it occurs. In this way, the system learns incrementally; past experience is implicitly represented by the evolved rules.

2. Classifier Systems

A generic production system is composed of a set of rules defined linguistically, i.e. IF A THEN B. The application of Genetic Algorithms over rules of these systems requires an intermediate representation, "codified rules", for genetic operators to act.

Classifiers Systems are a specialized form of production system that have been designed to be amenable to the use of Genetic Algorithms [6]. These systems were developed by Holland and Reitman [8], and later refined and modeled by Holland [9]. CS are machine learning systems that learn syntactically simple string rules (called *classifiers*) to guide their performance in an arbitrary environment [6].

2.1 Classifier System Architecture

A schematic representation of a Classifier System is showed in figure 1. In these systems, it can be distinguished three activity levels:

(1) Performance, also called rule and message system: it interacts with the environment, gathering information through the input interface and producing the output through the output interface; it also receives the payoff. Structurally, the performance level consist of: (A) a finite population of fixed length condition/action rules, (B) a message list, (C) an input interface consisting of a set of environmental feature detectors and (D) an output interface for acting in the environment, that are also shown in figure 1.
(2) Credit Assignment: it causes rules to be established (fitting a rate of rules) on the basis of their observed utility to the systems goal.
(3) Discovery: it employs a genetic algorithm as a discovery operator that automatically generates new rules.

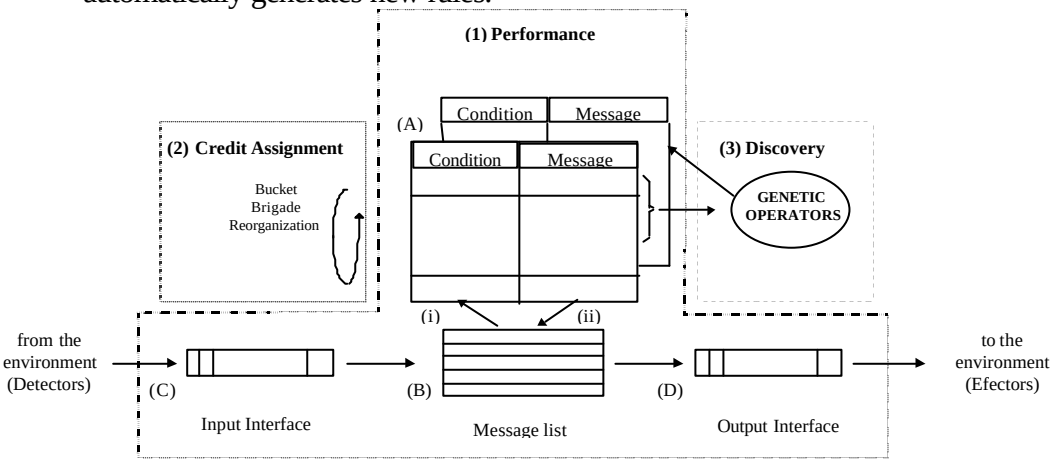

Figure 1: Representation of a Classifier System. (i) All messages are tested with all classifiers. (ii) Winning classifiers post their messages to the message list.

In a CS, rules are composed of two parts: condition and message and they are codified as strings: each condition is a string of fixed length k over the alphabet $\{0,1,\#\}$ (don't care symbols, "#", match both 0 as 1) and each message another string of fixed length k over the alphabet $\{0,1\}$.

2.2 Sequence of Operations in a Classifier System

In the performance level, when a codified message arrives from the environment (through the input interface), the message is set in the message list. The message list is compared with all the classifiers and those that match with some message are fired. The fired rules post their messages into the message list. Several rules could be activated in parallel by a message. Before rules post messages, the message list ought to be cleaned. Activation of rules is repeated for *n* cycles. Finally, a message is chosen to give the output through the correspondent interface. The sequence of operations is summarized in Figure 2.

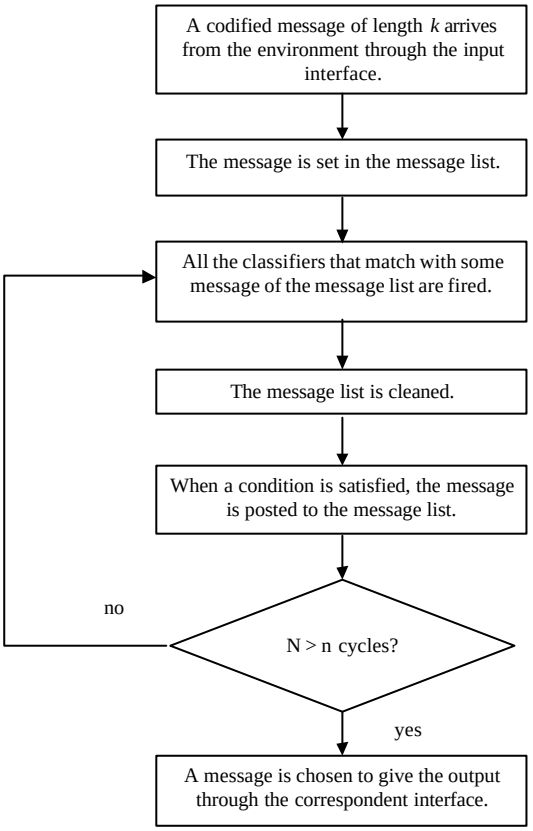

Figure 2: Representation of a Sequence of Operations in a Classifier System.

In the Credit Assignment level, a reinforcement algorithm (called the Bucket Brigade, BB, [10]) is used to solve the credit assignment problem: how to reinforce individual rules in a multistep chain when the external reward is given only at the chain conclusion. This algorithm also allows selection among incompatible or contradictory solutions. BB assigns to each rule a value, called *strength*, that indicates the rule usefulness to the systems goal. When a classifier is matched, it is qualified to participate in an activation auction. To participate in the auction, a classifier makes a bid, proportional to its strength and its specificity (this value is concerned with the number of don't care symbols in the rule). Winning classifiers pay a portion of their strength (their

bid) to the one responsible of their activation, and their messages are posted to the message list.

A genetic algorithm is used in level three, to generate new, and possibly better, rules into the system. From a CS, a set of rules with higher strength values is selected, genetic operators are applied and the new rules obtained are set into the new CS. After this, the BB will reorganize the rules strength.

3. Autonomous Robot Classifier System

The application of CS to solve the navigation problem needs both actions and reactions. Therefore, a CS able to react (considering only the sensorial input information) and to chain actions (considering information of the sensorial input and the previous state of the CS) ought to be developed. The existence of internal cycles in CS (see Figure 2) makes difficult the learning process of a reactive controller. On the other hand, internal cycles are necessary to develop more complex actions sequences. The designed CS, proposed in this work, modifies the performance level to include the possibility of both actions and reactions. In section 3.1, this new architecture is described. The special mechanisms, included in this architecture, modify the sequence of operations of traditional CS (see section 2.2). This new sequence is presented in section 3.2.

3.1 Autonomous Robot Classifier System Architecture

Following the architecture presented in section 2.1, the performance level has been modified to learn reaction and actions. The performance level is composed by condition and messages in the same way than a general CS except for two main differences: (1) condition/message length, k, is longer than the environmental message length, m, $(k>m)$, and (2) both conditions and messages are divided in three blocks. Each block contains different kind of information (Figure 3):

1. Environmental information.
2. Information related with rules fired in a previous instant (internal conditions).
3. Information about the decisions.

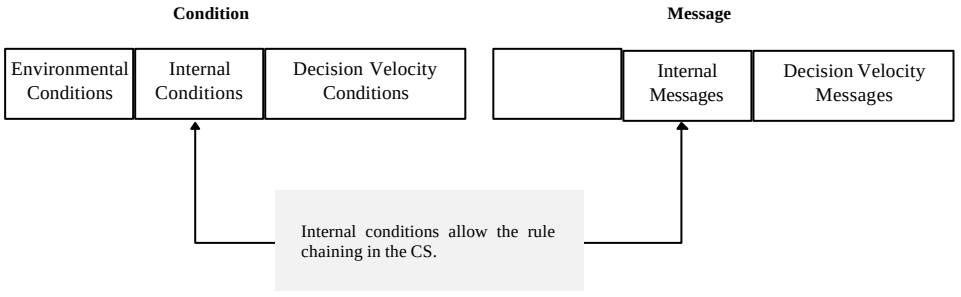

Figure 3: Composition of Conditions and Messages.

As it could be seen in Figure 3, the first block of the message, environmental block, is empty. This empty block is used to fuse the environmental message with messages of previous activated rules. The complete sequence of operations will be explained in more detail in section 3.2. This fusion mechanism allows the controller to learn complex actions, composed by a sequence of actions. Besides fused messages, another message with only the first block of message, environmental part, is posted to message list. This mechanism allows learning reactions, breaking the chain of rules.

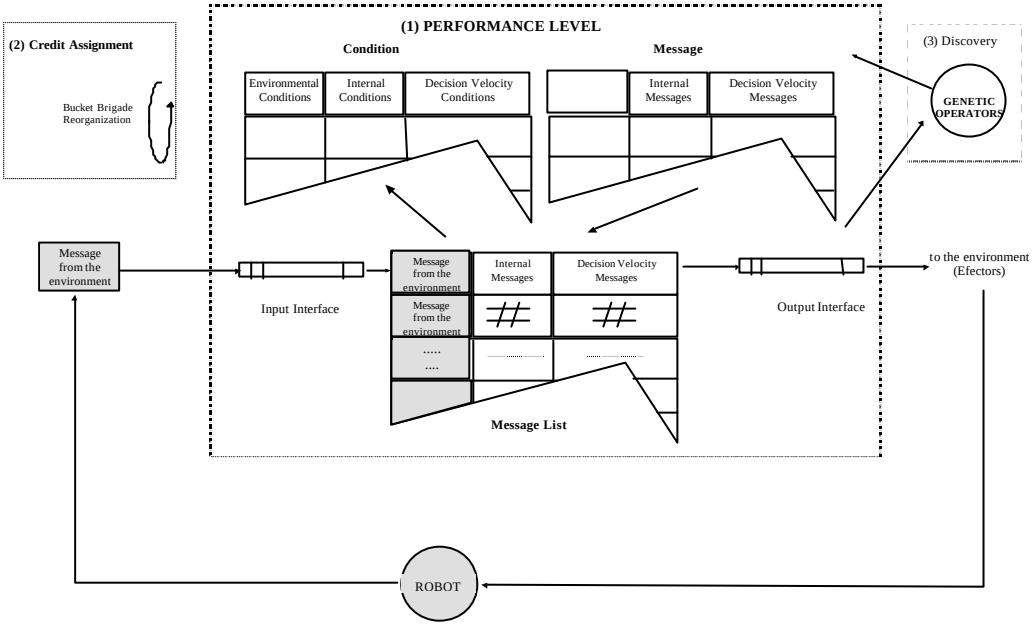

Figure 4: Developed CS and information interchange between the robot and the CS.

In figure 4 the complete Classifier System proposed and the information flow are shown. In order to deal with this new architecture, it is necessary to define a new sequence of operations.

3.2 Sequence of Operations in an Autonomous Robot Classifier System

When a codified message of length *m* arrives from the environment through the input interface, the message is fused with messages of previously activated rules. A message composed with the environmental message and don't care symbols is posted to the message list. All the classifiers that match with some message of the message list are fired. A message is chosen from these fired rules. The list is kept to the next decision cycle. These operations do not contain the repetition of the matching process of the general CS because the chain of rules needs the information of the next environmental

input. The rule chain is over different inputs, using internal conditions and message fusion, allowing to learn reactions and actions sequence. The sequence of operations is summarized in Figure 5.

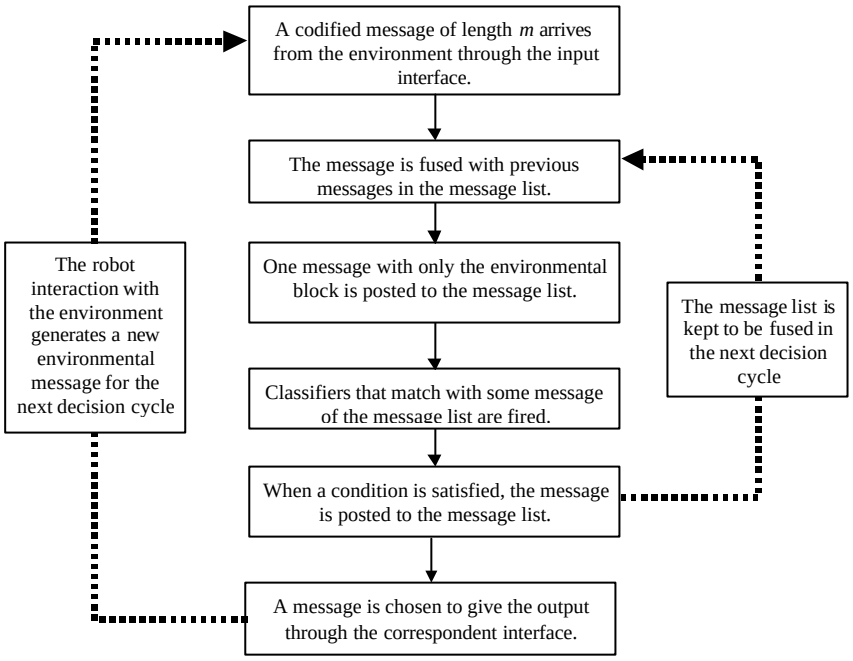

Figure 5: Representation of a Sequence of Operations in an Autonomous Robot Classifier System.

In this way, actions chaining is obtained taking into account two special mechanisms in conditions and messages: the environmental message is fused with the previously posted messages and internal conditions are added to evolve a chaining strategy. This strategy allows to chain rules activated by the environmental message with previous activated rules. In addition to the environmental message fusion, the CS requires the inclusion of internal conditions that provide the evolution of a chaining strategy. The fusion method gives the way to chain rules and the internal conditions support the knowledge about the relationship between rules. The evolution process over the internal conditions provided by the genetic algorithm leads to learn sequences of rules through time.

Although all the messages in the message list are composed by fusion, there is always one message with only the environment block filled with the environmental message (don't care symbols, "#", filling the other two blocks, see Figure 4). The matching process considers environmental conditions only and the system is able to break the rule chain and to react to the environment. In this way, reactions are obtained when a message, with the environmental information only, is posted to the message list.

These mechanisms allow the generation of more complex rules needed for the final solution of the problem. An example of Condition/Action rules that could evolve is as follows:

```
IF    External_Signal IS <type x>        AND
      Last_Rule_Fired IS <type y>        AND
      Decision_Velocity_Part IS <Vi, Vj>
THEN Send_Message <001...>
```

The reaction mechanism, on the other hand, allows the evolution of traditional reaction rules as:

```
IF    External_Signal IS <type x>
THEN Send_Message <001...>
```

4. Environmental and output messages design

The codification of information in CS (the design of environmental and output messages) is based on the special problem where CS will be applied. In this work, the CS is used as a controller of an autonomous robot named Khepera [15]. The mini-robot Khepera is a commercial robot developed at LAMI (EPFL, Laussanne, Switzerland). The robot characteristics are: 5.5 cm of diameter in circular shape, 3 cm of height and 70 gr of weight. The sensory inputs come in from eight infra-red proximity sensors. These sensors are composed of two devices: an IR emitter and a receiver. The emitter and the receiver are independent, then it is possible to use the receiver to measure the reflected light (with the emitter active) or to measure the environmental light (without emission). The reflected light measurement can give some information about the obstacles. In fact, this measure is not only a function of the distance to an object in front of the emitter but also the environmental light and the object nature (color and texture). So the value of distance is modified by the measure of the ambient light and the object nature, the light used is constant and all the obstacles used have the same color and texture. The robot has two wheels controlled by two independent DC motors with incremental encoder that allow any type of movement. Each wheel velocity could be read by a speedometer.

Using the ambient sensors it is possible to measure the distance and the angle to a light source. The distribution of the amount of light coming into the eight sensors is used to evaluate the distance and the angle to the source (Figure 6). The amount of light received in the sensor depends on the distance of the light source. Each sensor is described by a sigmoidal function [15]. When the robot is placed near a light source, Figure 6, each sensor gives a value of light intensity based on the sigmoidal function. In Figure 6, an example of different values in each sensor is represented. In this case, the sensor 6 returns the minimum value from all sensors, the value is used to obtain the distance and the sensor number (ID 6) to obtain the angle to the light source.

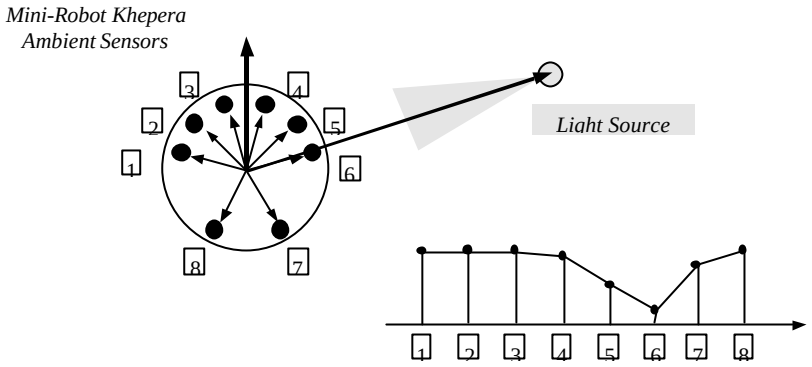

Figure 6: Light Incoming Distribution in the sensors

The transformation function of the proximity sensors is a linear function between (0,0) and (1023,40). The first point corresponds to the minimum distance, 0, both in the real robot as in the codified domain. The second point corresponds to the maximum value, 1023, in the real robot and in the codified domain, 40. The function that transforms the ambient sensors into distance an angle searches the minimum intensity value of the proximity sensors and the ID of that sensor. This intensity value is transformed by means of a linear function to get the distance. The desired angle is obtained considering the ID of the sensor with the minimum intensity valued found previously:

Sensor ID	1	2	3	4	5	6	7	8
Angle	90	45	15	345	315	270	190	170

4.1 Codification

The sensors (proximity, ambient and speedometer) supply three kinds of incoming information: proximity to the obstacles, ambient light and velocity. Instead of using the eight infra-red sensors individually, they have been grouped giving a unique value from two sensor input values (Figure 7a), reducing the amount of information received by the CS. Representing the goal by a light source, the ambient information lets the robot know the angle (the angle position in the robot of the ambient sensor receiving more light) and the distance (the amount of light in the sensor) (Figure 7b).

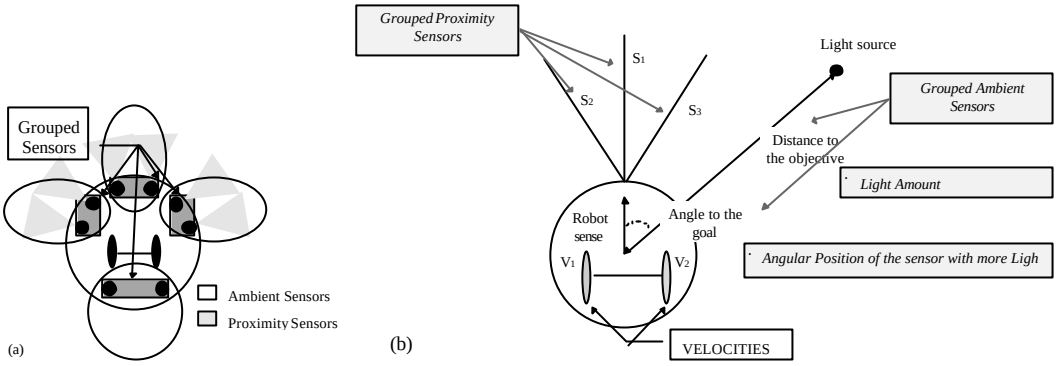

Figure 7: (a) Sensors considered in the real robot. (b) Input information to the system.

The input to the CS consists of three proximity sensors, angle and goal distance (given by ambient sensors) and velocity values obtained by the speedometer.

The distance information of proximity sensors is obtained by the response curve of the sensors, that is a sigmoidal function defined over the intensity values domain. The distance domain is transformed, translating it into a simpler domain to codify the values. This transformation allows both the CS and the robot to be independent. So the CS could be developed for any robot by changing the transformation function. The input domain has been partitioned in four crisp sets. The maximum distance value "seen" by one sensor is 40 units and is divided in ranges as is shown in Figure 8.

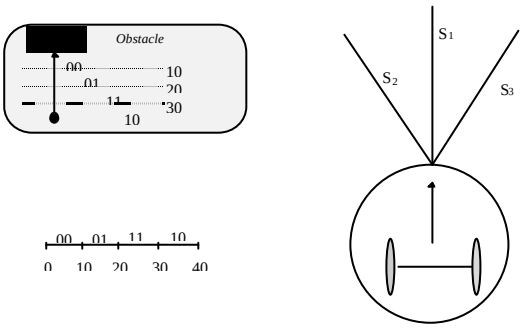

Figure 8: Codification and partition of the proximity information.

The angle sets are of different size to consider a fine fitting of the trajectory, avoiding big oscillations when the robot follows the right direction (the sets near 0 and 2π are smaller than the "$<\pi$" and the "$>\pi$" ones). The input domain partitions are presented in Figure 9.

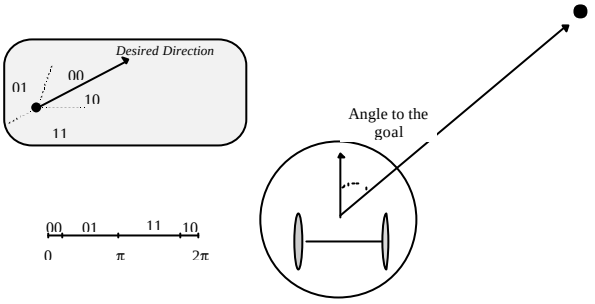

Figure 9: Codification and partition of the angle information.

To keep the independence between robot and CS, the distance values are translated from the real sensor values to a domain defined from 0 to ∞. The input domain has been partitioned in four crisp sets as is shown in Figure 10.

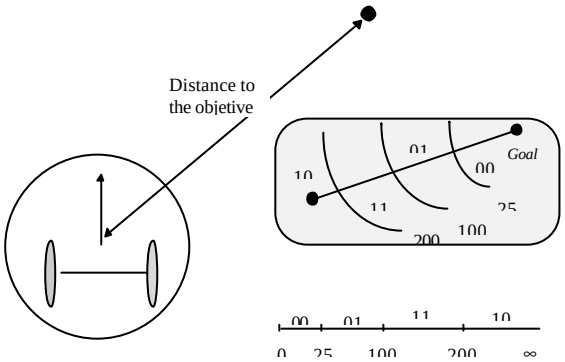

Figure 10: Codification and partition of the distance information.

Velocity values flow as input to the classifier system and as decision from the CS to the robot. The values are defined by the maximum and minimum velocities (10, -10). This range is divided in four equal sets as is shown in Figure 11.

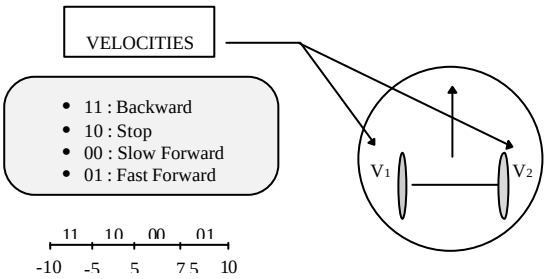

Figure 11: Codification and partition of the velocity information.

All these sets should be codified to build the message from the environment. Two binary digits are needed to represent each set. The codified inputs to the robot are also displayed in figures 8, 9, 10, 11.

4.2 Messages Composition

As it was previously mentioned (section 3) conditions and messages of the CS are divided in three blocks. The environmental block should be matched with the environmental message arriving from the robot and it is defined by codified sensor values. The environmental message includes all the codified sensors, composed as in Figure 12a. The first information of the environmental message is about the proximity sensors to describe the near environment surrounding the robot. The second information corresponds to the goal description using the angle and distance measures. The last information of the environmental message deals with the actual velocity to consider the difference between the real and the last decision velocity.

The decision velocity is codified in the output message. Velocity values are decodified an applied to each wheel in the robot, Figure 12b.

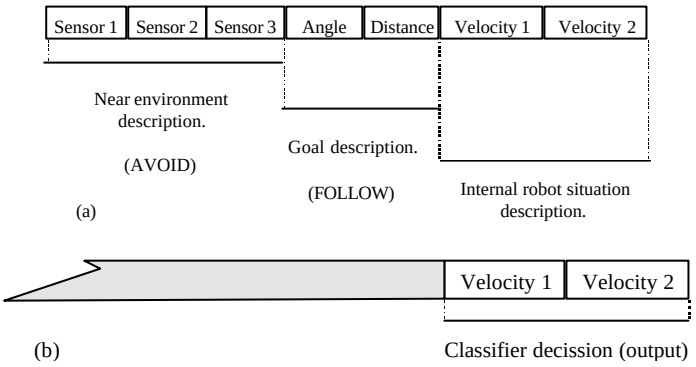

Figure 12. (a) Composition of the environmental message,
(b) Decision velocities in the output message.

5. Experimental Results

Learning reactions in a real robot involves two main tasks: first, to discriminate better rules from a set of rules, and second to discover new rules to face new situations or to improve its performance. In a CS the learning process is very sensitive to the payoff. To fit the environmental payoff, a CS with a constant set of rules and a complex environment where most kind of situations could be found have been used. To evaluate several systems in different environments, fitting the payoff, it is useful to work with a simulator. The result obtained in the simulator has to be directly transferable to the real robot, so the simulation requires specification of environment, needs, sensory and motor equipment and learning method. The resulting system has been tested in the real robot using different environments.

5.1 Experimental Environment

Evolution takes a long time (days) of continuous functioning of the hardware. In order to prove the different configurations of the CS, a simulator developed in a previous work [19] has been used. In the simulator, the characteristics of the turtle robot model [15] and the physical restrictions of the Khepera robot have been considered.

The simulation world consists of a rectangular map of user defined dimensions where particular objects are located. In this world it is possible to define a final position for the robot. In this case the robot is represented with three proximity sensors and two special sensors to measure the distance and the angle to the goal (Figure 13).

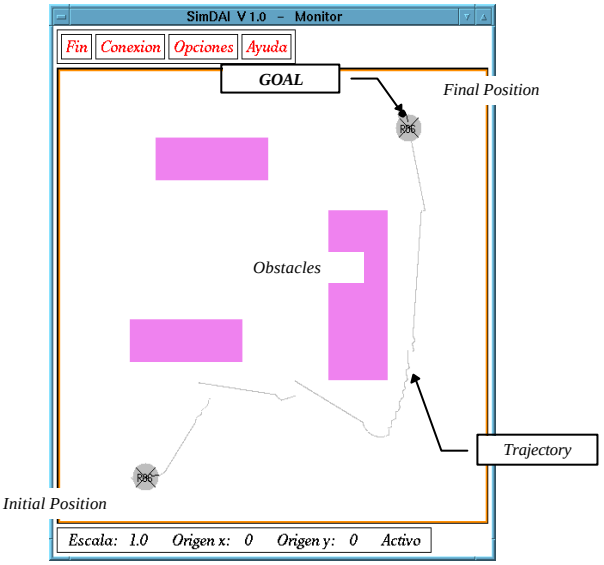

Figure 13: SimDAI Simulator (Example of one simulated environment).

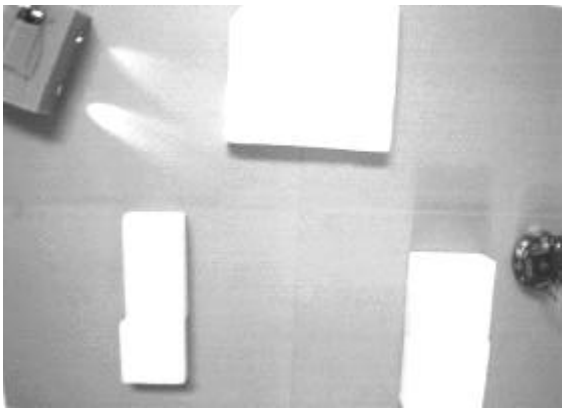

Figure 14: Example of a real experimental environment.

Different simulated worlds which resembles the real ones have been defined in order to tune the payoff from the environment before being implemented in the real

world. An example of these environments can be seen in figure 13 and figure 14. The pay off function is based on the analysis of the new situation produced by the CS output. This situation is measured by means of distance to surrounding obstacles and the relative position to the goal. It is necessary to adjust parameters that weight each part of the fitness function to define the relative importance of each one. The relations among parameters on the simulator will be kept constant in the real environment. The transformation function is used to keep constant the values of parameters. Then, the system developed is the same in both cases (simulated and real) except the differences in the treatment of the sensors, by the transformation function.

5.2 Learning without Genetic Algorithms

Different payoff parameters have been tested, the proposed reward from the environment is a function of the distance and the angle to the goal and the proximity of obstacles. With this reward the system learns to avoid the obstacles when the proximity is dangerous and to follow a direct path to the goal when no obstacles are near.

The system learns from an initial situation using: the reward, the bucket brigade algorithm (to distribute the rule strength when a rule is activated by another rule), and a function that assures that strength of not-fired rules decreases to differentiate them from the fired rules, commonly used in CS [6] [10]. This decrease of rule strength is called a tax.

Experimental results show a learning behavior where the strengths of the best rules for the problem increase while the strength of the other rules decrease vs. cycles in the execution. The set of rules for the CS is collected in table 1.

(a) Condition	Message
##0000########10####	###############11111
##00##########10####	1
####00########10####	###############11101
000101########10####	1
001101########10####	###############11111
001001########10####	0
000111########10####	###############11111
001111########10####	1
001011########10####	###############11110
000110########10####	0
001110########10####	###############11110
001010########10####	0
	###############11001

(c) Condition	Message
########00####11####	1
######0101####11####	###############11111
######1101####11####	1
######0001####11####	###############11100
######1001####11####	0
######0111####11####	###############11001
######1111####11####	1
######0011####11####	###############11001
######1011####11####	0
######0110####11####	###############11111
######1110####11####	1
######0010####11####	
######1010####11####	

(b) Condition	Message
010101########1#####	###############001111
011101########1#####	###############001000
011001########1#####	###############001000
010111########1#####	###############000010
011111########1#####	###############000000
011011########1#####	###############000001
010110########1#####	###############000010
011110########1#####	###############000100
011010########1#####	###############000000
110101########1#####	###############000000
111101########1#####	###############001000
111001########1#####	###############001000
110111########1#####	###############000010
111111########1#####	###############000000
111011########1#####	###############000001
110110########1#####	###############000010
111110########1#####	###############000100
111010########1#####	###############000101
100101########1#####	###############000000
101101########1#####	###############001000
101001########1#####	###############000001
100111########1#####	###############000010
101111########1#####	###############000000
101011########1#####	###############000001
100110########1#####	###############000100
101110########1#####	###############000100
101010########1#####	###############000101

Message
###############11101
0
###############11001
1
###############11110
0
###############11001
0
###############11100
0
###############11001
1
###############11110
0
###############11001
0
###############11100
0
###############11001
0
###############11100
0
###############11010
0
###############11000
1

Table 1: Rules of the CS: (a) avoiding when obstacles are near, (b) avoiding when obstacles are far, (c) following.

These rules could be clustered in three groups (table 1). First group (a) is related with situations in which there are collision danger. With this rules, the robot turns to the right direction in presence of obstacles. Second group (b) corresponds to situations in which there are no obstacles near, in this case, the robot will modify its trajectory in order to avoid obstacles when there are no collision danger. This set of rules allows the robot wandering around the experimental environment without taking into account the goal. The last group (c) consist of rules that independently of obstacles position, change the trajectory of the robot facing the goal.

An example of rule in the three cases is shown in figure 15.

	CONDITION										MESSAGE			
	S1	S2	S3	Ang	Dist	Vel 1	Vel 2	Internal	Vel 1	Vel 2		Internal	Vel 1	Vel 2
(1)	00	11	01	##	##	##	##	10	##	##	....##	10	11	00
(2)	01	11	11	##	##	##	##	1#	##	##	....##	00	00	00
(3)	##	##	##	10	10	##	##	11	##	##	....##	11	00	01

Figure 15: Examples of rules.

In more detail, the meaning of rule (1) is explained: **If** there is an obstacle in front of the robot at a distance between 0 and 10 (S1=00, very near) and other obstacle on the left at a distance between 20 and 30 (S2=11, far) and another on the right at a distance between 10 and 20 (S3=01, near) and the last message sent is type 10 (internal=10) **Then** send a message type 10 that turn abruptly to the left (vel1 =-5, vel2=5).

The essential rules for solving the problem belong to groups (a) and (c). Group (b) rules are superfluous because they allow the robot to avoid obstacles when they are far (something not very useful) and they are not able to follow the goal. The most efficient strategy is following the goal except when there is collision danger. This is accomplished by rules of groups (a) and (c), in an efficient way when the appropriated rules from these groups are selected.

In order to fit the environmental payoff, several simulations have been carried out. As a result the reward is built considering four positive payment contributions when: there are no collisions and/or distance to the goal is decreased and/or angle to the goal is decreased and/or the distance to an obstacle is increased. The implemented tax is a function of the time and the actual strength, so the strength is reduced during the execution when the rule is not fired.

Once these parameters are set, the CS with 52 rules has been applied to the real robot, with different strength initial values. The CS shows the capability of discriminating among the three rules groups. It has been experimentally tested that rules belonging to groups (a) and (c) have an average strength value above rules of group (b). In figure 16 the rule strength evolution (over 900 cycles of running) of a rule belonging to every group is shown.

The CS has also showed the ability to discriminate rules inside each set. Some rules of the set are better (useful) than others because they wait until objects are near or turn less abruptly. The CS is able to select (giving higher strength values) the more

convenient rules of each set and to chain rules of different sets. Rules that have the ability to solve a great part of the problem by themselves in some special environmental configuration (e.g. when there are no obstacles to the goal) have their strength increased. This strength growing takes place in a short number of cycles as can be seen in figure 16a. The meaning of evolution of strength values is: while the special conditions for these rules to be useful aren't reached yet, their strength is decreased as an effect of the taxes mechanism. On the other hand, when they are fired (or could be fired), and due to the fact that they can solve a great part of the problem, they will be fired once and so on, to grow in strength quickly. If the rules are fired as a part of a chain in execution, their strength values are kept constant, more or less, as a result of composing the grow for firing and the loose for taxes. When the robot faces a complex situation, any rule try to solve the problem in isolation. In this case, strength values of good rules increases or decreases depending on whether it takes part or not in the chain that is on execution and that execution ends with a positive or negative payoff (figure 16c). Finally, all rules have a tendency to decrease their strength as an effect of the taxes. This is more evident in no necessary rules (figure 16b).

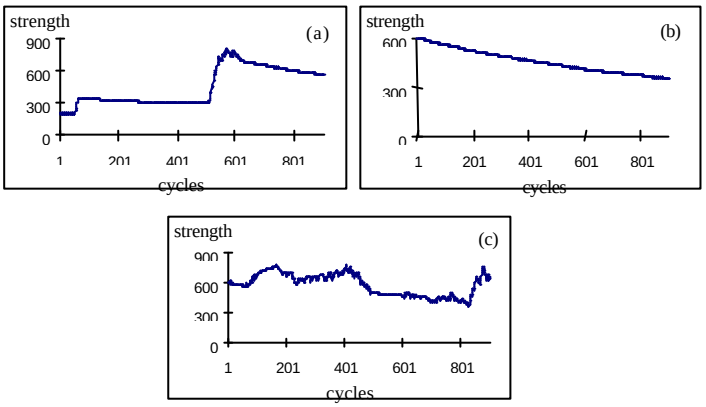

Figure 16: Evolution of the rules strength of: avoiding (a) with obstacles near (b) no obstacles and (c) following rule.

To check the robustness of the developed CS some experiments have been done on the real robot considering different initial strength values. A similar set of rules has been discriminated. The CS selects the better rules, this rules belong to the cluster (a) and (c) where are representing the rules that avoid in danger situations and follow a right path in absence of obstacles. The final effect of this discrimination is that the robot is able to reach the objective avoiding obstacles in an efficient way.

Figure 17 shows the results of the experiment where the initial assignation of strength values has been done as follows: if the rule belongs to cluster (a) and (c) it is set to 200 and to 600 otherwise (cluster b). In these conditions evolution of good rules is much more difficult because they have, initially, a third part of the strength.

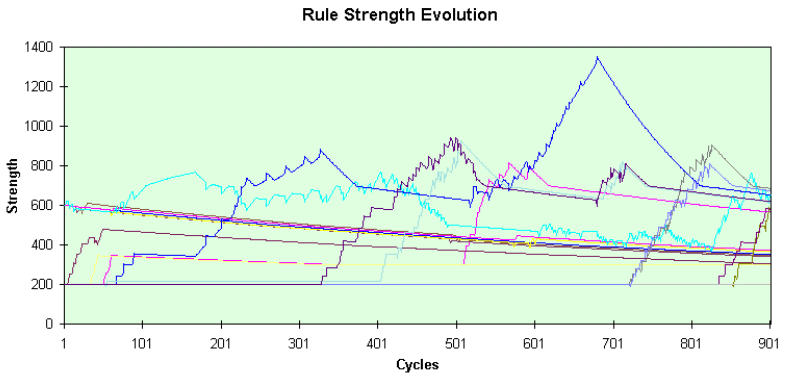

Figure 17: Evolution of the strength of the whole set of rules over 1000 cycles

It can be seen that a similar set of rules has been discriminated, all of them belonging to the first and third groups. Also can be seen similar behaviors to those shown in figure 16(a, b, c). The system has proven, thus, to be insensitive to initial strength values assignation.

5.3 Learning with Genetic Algorithms

Once it has been tested that the CS is able to discriminate good rules, a GA has been included to discover new, and probably better, rules.

The GA is called at the end of an execution. The robot navigation in an execution starts from an initial random point and it ends when:

(a) the goal is reached, or,

(b) the time exceeds some limit, fixed by the time needed to reach the goal using a fuzzy controller [13], or,

(c) the number of collisions exceeds a maximum threshold.

There are several approaches for initializing rules in a CS. The most common method is a random initialization. This presents the maximum challenge to the learning algorithm but doesn't take into account the previous acquired knowledge. As an alternative, a method consisting in seeding the initial population with previous learned knowledge can be used [18].

A combination of the two techniques could be useful to test the learning capacity of the system without losing much of the previous acquired knowledge. In this work, the last approach has been implemented.

The initial population of the CS is composed of the ten better rules obtained in previous experiments (the rules with higher average strength, section 5.2) and for each rule, five random new rules have been added.

The system follows the Michigan approach [9]. A problem with this approach in CS is the sensitivity of the rules. This is due to the fact that the strength of some rules

depends on the strength of others. To overcome this problem, a high degree of elitism has been used, in such a way that generations are similar.

The selected parameters of the GA are:

- 1 of crossover probability
- 0.02 of mutation probability
- 0.85 of elitism

Evaluation of the system performance is based in a quantitative measure. This measure doesn't take part in the evolution process but it reflects the system global performance evolution. For measuring systems evolution, following features have been considered:

- Time needed to reach the goal (seconds in the real robot and cycles in the simulator).
- Trajectory length (by means of velocity values of the motor wheels).
- Number of oscillations (measured using the difference between the wheels velocity).
- Number collisions (measured using the minimum value of the proximity sensors).

The environment makes the CS to find the set of rules that cooperate to achieve a common goal in order to incrementally perform the behavior "reach and avoid" in less time, with less collisions and following a more straight trajectory to the goal. Figure 18 shows an evolution of the trajectory from the first execution to the last one.

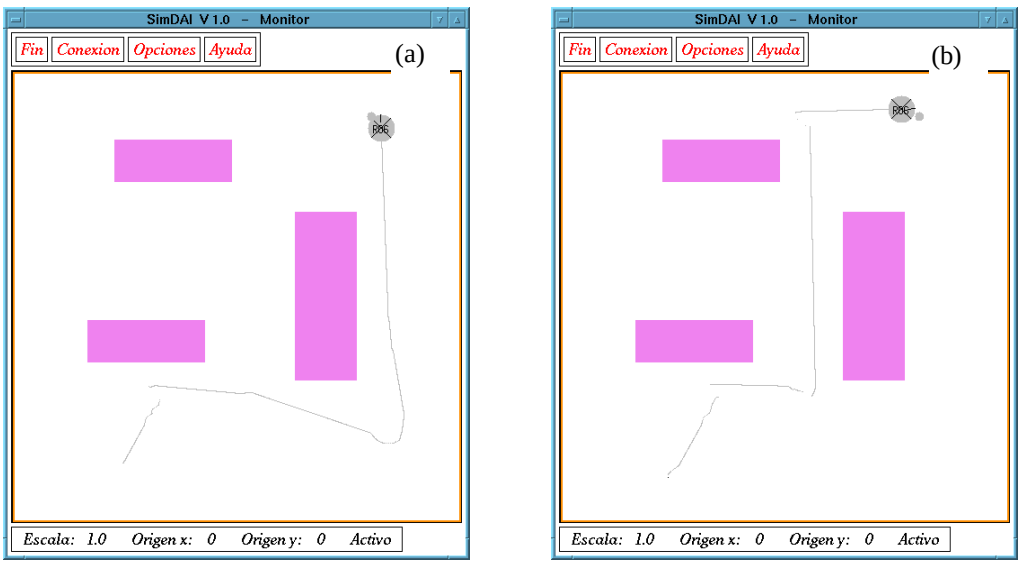

Figure 18. System Evolution Examples in one simulation. (a) Starting situation. (b) Final situation.

In figure 18a it can be seen the effects of removing some of the previous acquired knowledge. Despite of the ability of the robot to reach the goal, it is done in an

inefficient way. Through evolution some new good rules have been added making the robot to follow a better trajectory without collisions (figure 18b).

The last CS evolved has been used to control the real robot in different environments. In figure 19 a real experiment equivalent to the one of figure 18b is showed. Three frames that represents the starting point (figure 19a), intermediate state (figure 19b) and the final position (figure 19c) are collected in figure 19.

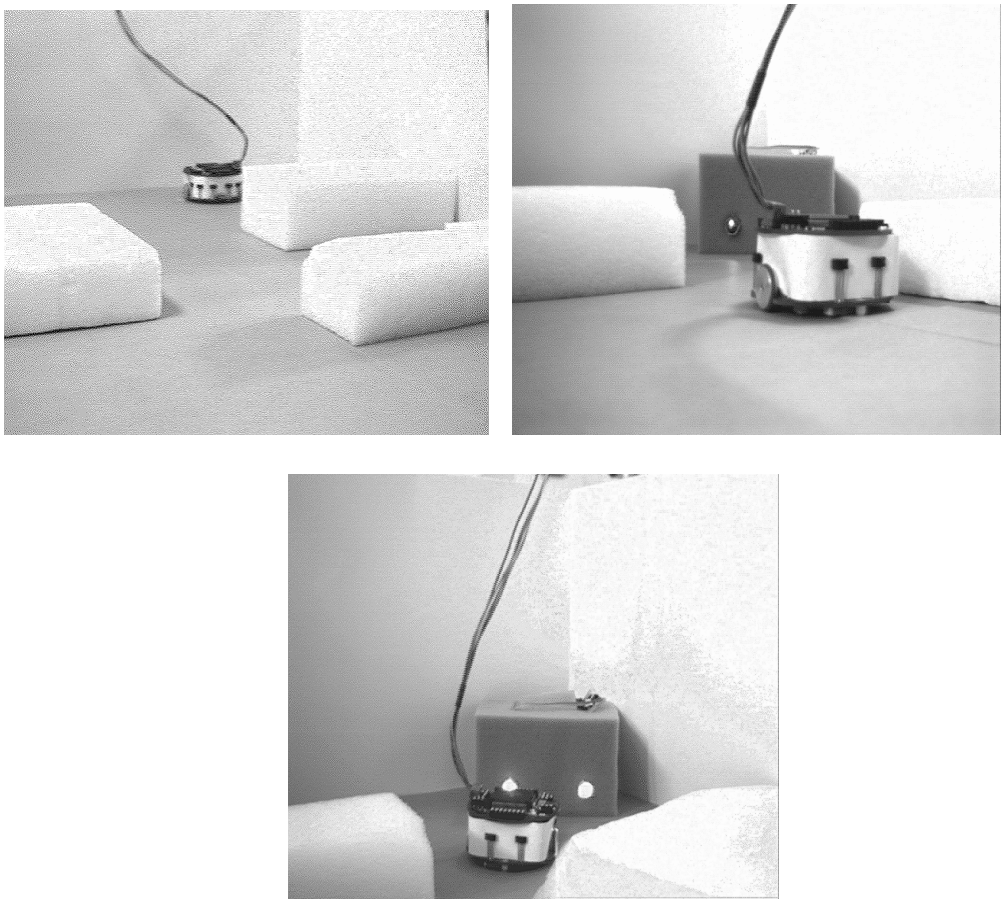

Figure 19: System Evolution Examples in one real experimental environment. (a) Starting position, (b) intermediate and (c) Goal reached.

The results of the evolution of the system, based in the quality measure previously mentioned (figure 20), shows a fast growing of the quality in the first steps. This is due to the existence of less good rules at the beginning and the evolution of new good rules is easier than when the system is filled of good rules. In the last steps the quality of the whole system reaches an stationary situation, that is, the system has evolved enough rules to react to any possible situation in a correct way.

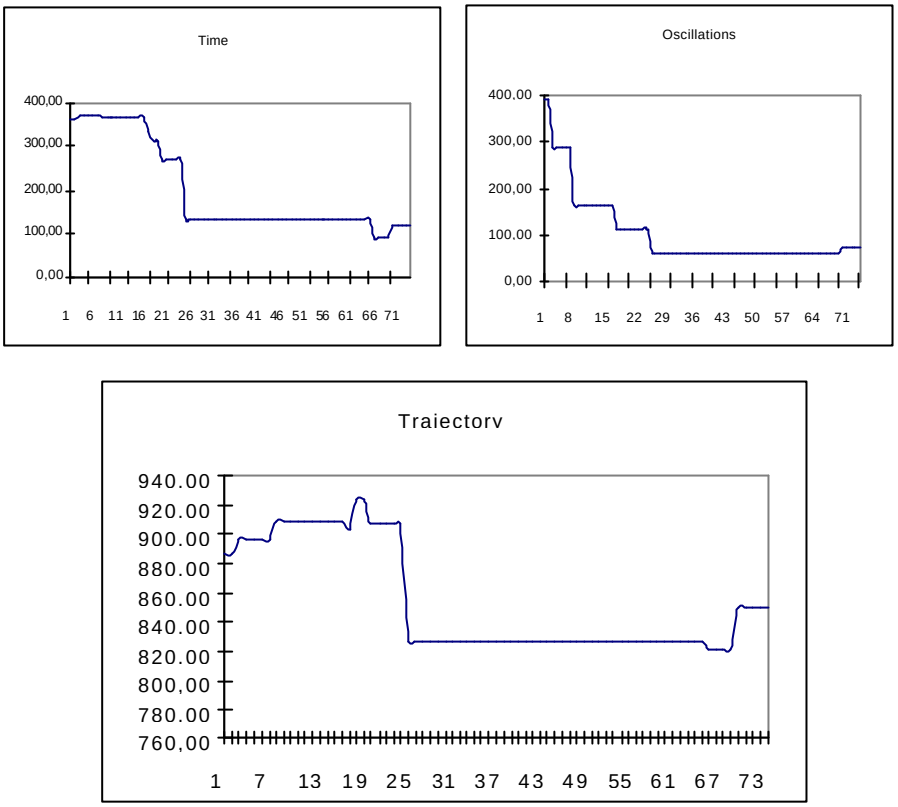

Figure 20: Evolution of the Quality Measure.

5.4 Testing CS in Dynamic Environments

The proposed Classifier System has learned to react and to chain actions to solve the navigation problem. The learned CS has also been tested in dynamic environments. A subset of static environments from previous experiments has been selected in order to compare with the results over similar dynamic environments. The dynamic experiments are defined in this way: the initial point, the situation of the goal and the static objects are equal than in the static ones but a circular object is wandered on the simulated world. The mobile obstacle starts its movement from the position (x = 100, y = 200, initial direction = 200°) with a random trajectory that crosses in many cases the robot path and avoid obstacles without a predefined goal. When the robot finds an obstacle in its way it is able to react for avoiding the mobile obstacle without losing the tendency to arrive the goal.

The static environments are defined by the initial position of the robot and the objects. Nine experiments have been defined and 50 executions have been carried out in order to obtain the average of trajectory length, collisions and time. Each experiment is defined by the robot initial position (three different positions have been used: Robot1, Robot2 and Robot3) and the number of static obstacles (one, two or three). Static objects are the same than in Figure 18. Each robot is defined by coordinates (x, y) and their initial direction:

- Robot1: x = 50, y = 400, initial direction = 0°
- Robot2: x = 300, y = 450, initial direction = 180°
- Robot3: x = 50, y = 150, initial direction = 0°

The average results of the CS in time, trajectory length and collisions are shown in table 2 from 50 experiments. Three selected examples of these experiments are shown in figures 21a, 21b and 21c.

Robot	Environment	Time Average	Distance Average	Collisions Average
Robot1	3 Objects	91.80	678.10	0.7
Robot1	2 Objects	82.21	650.30	0.7
Robot1	1 Object	68.32	635.20	0.7
Robot2	3 Objects	86.70	583.10	0.8
Robot2	2 Objects	73.20	523.43	0.3
Robot2	1 Object	62.30	427.80	0
Robot3	3 Objects	69.80	461.50	0
Robot3	2 Objects	27.50	333.20	0
Robot3	1 Object	27.30	332.70	0

Table 2: Numerical results of static experiments.

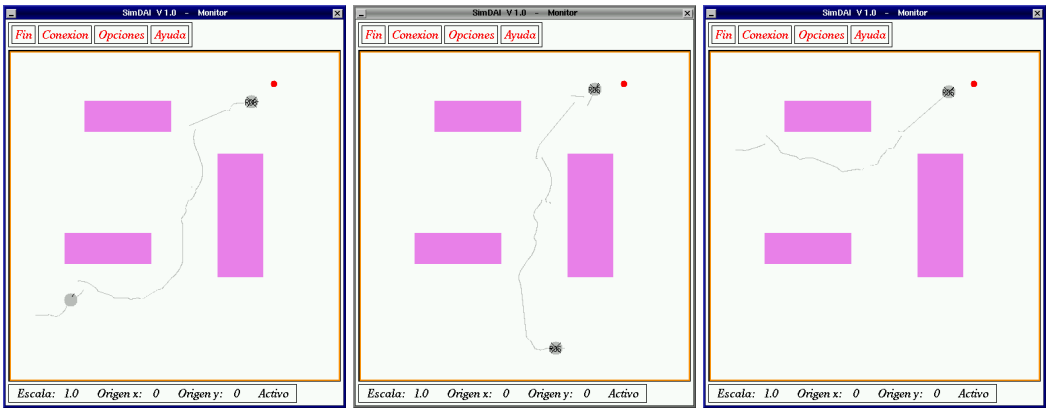

Figure 21: Three static experiments: (a) Robot 1, (b) Robot 2 and (c) Robot 3.

In table 3, the numerical values obtained in these dynamical experiments, with a mobile obstacle, are shown. Figures 22a, 22b and 22c show several trajectories starting from the same point than in Figures 21a, 21b and 21c, respectively.

Robot	Environment	Time Average	Distance Average	Collisions Average
Robot1	3 Objects	157.50	791.51	3
Robot1	2 Objects	158.17	743.34	1.5
Robot1	1 Object	132.08	654.73	1.5

Robot2	3 Objects	154.21	657.62	1.6
Robot2	2 Objects	80.12	580.23	1
Robot2	1 Object	74.14	473.79	0
Robot3	3 Objects	77.33	394.62	0
Robot3	2 Objects	50.67	343.09	0
Robot3	1 Object	51.50	320.70	0

Table 3: Numerical results of dynamic experiments.

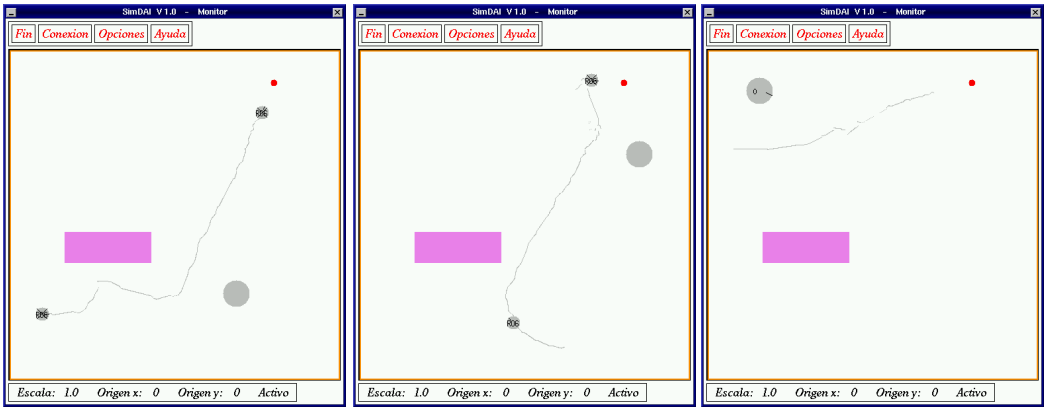

Figure 22: Three dynamic experiments: (a) Robot 1, (b) Robot 2 and (c) Robot 3.

As it could be seen in Table 2 and 3, the robot behavior is similar in different environments. The navigation problem is solved from different initial position and with different configuration of objects (both static and dynamic). Although the robot arrives the goal in any circumstance, the results are different in static than in dynamic environment, there are more collisions, the time and distance are larger in the dynamic ones due to the difficulty of the mobile object. The results show that the learning rate of CS allows to solve the navigation problem in different environments, both static and dynamic.

6. Conclusions

A reactive Classifier System ought to be embedded into the reactivity definition: "the system must decide for each input an action, and each action is determined by an input, but without losing the capacity of rule sequencing. This sequence must be produced in different instants of time". Then, CS must not be blind when decisions are taken as in [23] and [25]. A possible solution is the activation of rules in the same way that in traditional CS, but including the environmental message in the activation process.

The proposed CS has been developed to learn reactions (decision as a function of the environmental information) and actions (decision as a function of the environmental information and previous internal information). This modified Classifier

System has proven its ability to learn autonomous robot behaviors in dynamic environments.

The fusion of each environmental message with information of previous fired rules and the inclusion of internal conditions allow the generation of a sequence of actions, defined by a rule chain over different inputs. Sets of cooperative rules emerge from the evolution of the CS. Cooperation is viewed in this case as a rule chain, where a rule only has meaning if it matches with the environment and follows other specific rule in time.

The inclusion of one message without another information but the environmental input allows the evolution of reactions.

The experiments carried out without generating new rules proved the capabilities of our approach to accurately discriminate among rules in a system that is chaining rules at the same time that is receiving new inputs.

The results obtained considering the generation of new rules proved the capability of generating not only new better rules but the mechanisms for chaining new and existing rules.

Another important aspect verified in this work is the possibility of continuous learning and adaptation to new situations that allows to solve the problem even if there are mobile objects, more than one goal, and dynamical goals that could appear and disappear or move when the robot is moving.

7. References

[1] Brooker, L. "Improving behavior as an adaptation to the task environment". Doctoral Dissertation. Department of Computer and Communication Sciences, University of Michigan, Ann Arbor. (1982).

[2] Brooker L., Goldberg D. and Holland J., "Classifier Systems and Genetic Algorithms". Artificial Intelligence, 40, 1-3, 235-282, (1989).

[3] Brooks R. A., "Intelligence without Representation". Artificial Intelligence, 47, 139-159, (1991).

[4] Colombetti M. and Dorigo M., "Training Agents to Perform Sequential Behavior". Technical Report 93-023, Politechnico di Milano, Italy, (1993).

[5] Dorigo M. and Sirtori, E., "Alecsys: A Parallel Laboratory for Learning Classifier Systems", Proceedings of the Fourth International Conference on Genetic Algorithms, 296-302, (1991).

[6] Goldberg D.E., "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, New York, (1989)

[7] Holland J., "Adaptation in Natural and Artificial Systems". Ann Arbor, MI, University of Michigan Press, (1975).

[8] Holland J. and Reitman J.S. “Cognitive Systems Based on Adaptive Algorithms”, in Waterman D.A. & Hayes-Roth F. (Eds.), *Pattern-Directed Inference Systems*, New York, Academic Press, (1978).
[9] Holland J. “Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems” in Michasky R., Carbonell J. and Mitchell T.(eds.), *Machine Learning: An Artificial Intelligence Approach,* vol 2, Morgan Kaufman, los Altos, CA, (1986).
[10] Holland J., “Properties of the Bucket Brigade”, Proceedings of an International Conference on Genetic Algorithms and their Applications. Grefenstette J.J., Eds. (1986).
[11] Ishikawa S. “A Method of Autonomous Mobile Robot Navigation by using Fuzzy Control”. Advanced Robotics, vol. 9, No. 1, 29-52, (1995)
[12] Lee M.A. and Takagi H., “Integrating Design Stages of Fuzzy Systems using Genetic Algorithms”. Second International Conference on Fuzzy Systems, 612-617, (1993).
[13] Matellán V., Molina J.M., Sanz J. and Fernandez C., “Learning Fuzzy Reactive Behaviors in Autonomous Robots”. Proceedings of the Fourth European Workshop on Learning Robots, Germany, (1995).
[14] Matellán V., Molina J. and Fernández C., “Fusion of Fuzzy Behaviors for Autonomous Robots”, Proceedings of the third International Symposium on Intelligent Robotic Systems”, (1995).
[15] McKerrow P.J., “Introduction to robotics”, Addison-Wesley Publishing Company Inc. (1991)
[16] Mondada F. and Franzi P.I., “Mobile Robot Miniaturization: A Tool for Investigation in Control Algorithms”. Proceedings of the Second International Conference on Fuzzy Systems. San Francisco, USA, (1993).
[17] Schultz A., “Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation”, Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology, 213-215, (1991).
[18] Schultz A. and Grefenstette J.J., “Improving Tactical Plans with Genetic Algorithms”, Proceedings of IEEE Conference Tools for AI 90, 328-334, (1990).
[19] Sommaruga L., Merino I., Matellán V and Molina J., “A Distributed Simulator for Intelligent Autonomous Robots”, Fourth International Symposium on Intelligent Robotic Systems-SIRS96, Lisboa (Portugal), (1996).
[20] Wilson S.W., “Knowledge Growth in Artificial Animal”. Proceedings of an International Conference on Genetic Algorithms and their Applications, (1985)
[21] Wilson S.W., “Classifier Systems and the Animat Problem”, Machine Learning, 2, 199-228, (1987).
[22] Dorigo M., “Genetic and non-Genetic Operators in ALECSYS”, Evolutionary Computation, MIT Press, 1, 151-164, (1993).
[23] Dorigo M., “ALECSYS and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems”, Machine Learning, 19, 209-240, (1995).

[24] Grefenstette J.J., “Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms”, Machine Learning, vol 3, 225-245, (1988).
[25] Weiβ G., “Hierarchical Chunking in Classifier Systems”, Proc. of the 12th. International Conference on Artificial Intelligence”, 1335-1340, (1994).
[26] Isasi P., Berlanga A., Molina J. M. and Sanchis A., “Robot Controller against Environment, a Competitive Evolution”, Special Session on Evolution Computation, 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics. Germany, (1997).