Applying classifier systems to learn the reactions in mobile robots

ARACELI SANCHIS†, PEDRO ISASI†, JOSÉ M. MOLINA† and J. SEGOVIA‡

The navigation problem involves how to reach a goal avoiding obstacles in dynamic environments. This problem can be faced considering reactions and sequences of actions. Classifier systems (CSs) have proven their ability of continuous learning, however, they have some problems in reactive systems. A modified CS, namely a reactive classifier system (RCS), is proposed to overcome those problems. Two special mechanisms are included in the RCS: the non-existence of internal cycles inside the CS (no internal cycles) and the fusion of environmental message with the messages posted to the message list in the previous instant (generation list through fusion). These mechanisms allow the learning of both reactions and sequences of actions. This learning process involves two main tasks: first, discriminate between 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 generalized solution. The results show the ability of the system for continuous learning and adaptation to new situations.

1. Introduction

A wide range of robotic systems applied in industry are autonomous mobile robots. Sometimes the working environment is stationary, that is automatic floor-cleaning, automatic assembly, transporting parts in a factory, etc. Other problems involve interactions with dynamic environments, where robots have to be able to deal with unexpected events. The successful operation in such environments depends on the ability of adaptation to the changes.

A fundamental requirement for autonomous mobile robots is navigation. This task moves the robot from place to place with safety and no damage. Approaches based on the classical paradigms (abstraction, planning, heuristic search, etc.) were 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 (1991) and has been successfully implemented on several robots at the Massachusetts Institute of Technology and other institutes. The base of the subsumption architecture is 'behaviour'. Each behaviour reacts in a situation and the global control is a composition of behaviours. Different systems, from finite-state machines to fuzzy controllers, have been used for the implementation of these behaviours. The rules of these behaviours could be designed by a human expert, designed *ad hoc* for the problem, or learned using different artificial intelligence techniques (Schultz and Grefenstette 1990, Schultz 1991).

Machine learning has been applied to shape the behaviour of autonomous agents in this kind of environment. Some of these techniques become inapplicable to the learning reactive behaviour problem 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 (GAs) to evolve fuzzy controllers (Lee and Takagi 1993, Matellán *et al.* 1998), evolution strategies to evolve connection weights in a Braitenberg approach (Isasi *et*

Received 31 July 1998. Revised 28 April 1999. Accepted 26 January 2000.

† Grupo de Vida Artificial, Departamento de Informática, Universidad Carlos III de Madrid, Avenida de la Universidad, 30 Leganés, Madrid, Spain. Fax +34 91 624 91 29;
email: masm@inf.uc3m.es.

‡ Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, Madrid, Spain.

al. 1997, Molina *et al*. 1997) or neural networks to learn behaviours (Mondada and Franzi 1993).

The above-mentioned learning systems evaluate the complete behaviour without discriminating between different internal parts, that is, if the behaviour is composed of a set of rules, the evaluation does not discriminate between rules. However, for discovering new rules in isolation, some kind of measure of the accuracy of each rule is needed.

Classifier systems (CSs) (Brooker *et al*. 1989) are well suited to learn multiple (different) concepts incrementally under pay-off. These systems have been widely implemented and tested for a large number of theoretical problems, (Brooker 1982, Dorigo 1995, Holland 1975, 1995, Sanchis *et al*. 1996a, b), but there are not many cases in which they are included in real systems (Colombetti and Dorigo 1993, Dorigo 1995, Wilson 1987).

To survive in a dynamic environment, a system has to possess associations between environmental signals and actions that will 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 (Westerdale 1987).

(1) *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 becomes stronger when CSs are applied to problems in which a quick response is needed.

(2) *Rules chain*. CSs are able not only to learn rules but also to make a chain of previously learned rules. 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, owing to the high degree of interrelationships between rules.

The principal problem of CSs when they are applied to reactive problems, as Wilson (1985) and Grefenstette (1988) detected, is that during several CS internal cycles, the system becomes blind to environmental changes and, furthermore, in dynamic systems these changes happen repeatedly. The solution proposed by these workers does not allow the chaining of rules; thus, each time that an environmental input arrives an output is produced by a rule in isolation. The solution outlined by Wilson and by Grefenstette is too restrictive, which produces poor results. Therefore, the use of a CS was abandoned, in this type of problem, until the work of Dorigo and Sirtori (1991) and Dorigo (1995). In these papers, several designs are introduced in order to speed up the response of the CS. The new CS proposed by Dorigo (1995) 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 into several levels, building a hierarchical architecture, where a set of CSs learns to co-operate. Thus, the 'reactivity' is based on 'parallelism': different levels of CS are executed in different machines and, also, different CSs take charge of different tasks. Using these ideas, the response time becomes smaller. However, the system continues to be blind to environmental changes during internal cycles.

Another interesting approach by Weiβ (1994) employs what he calls hierarchical 'chunking' to the application of CSs 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 sequences A and B. However, when an aggregated rule is executed, without considering new environmental information, the system becomes blind in the same way as in Dorigo's work.

The capacity of the system to facilitate a quick response not only should be approached from techniques that attempt to increase the speed of the process but also 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 the 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. This CS is called a reactive classifier system (RCS) and modifies the general process in order to allow reactions without losing the possibility of a chain of rules. The new process integrates the environmental input with the internal state of the previous input. Then, from the input, the RCS gives directly an action and, at the same time, modifies the internal state. When the next input arrives, the message is fused with the previous internal state to allow a new reaction or an action that is in a chain with the previous action.

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 its description. The RCS robot controller starts without information about the right associations between sensor inputs and motor velocities. From this situation the robot is able to learn through experience to reach the highest adaptability grade to the sensors' 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 through pay-off; past experience is implicitly represented by the evolved rules. In order to fit the environmental pay-off, 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.

2. Definition of a classifier system

CSs are a specialized form of production system that have been designed to be amenable to the use of GAs (Goldberg 1989). These systems were developed by Holland and Reitman (1978), and later refined and modelled by Holland (1986a). CSs are machine learning systems that learn syntactically simple string rules (called *classifiers*) to guide their performance in an arbitrary environment (Goldberg 1989).

2.1. *Architecture*

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

(1) *Performance* (also called the 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 pay-off. Structurally, the performance level consists 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.

These 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 GA as a discovery operator that automatically generates new rules.

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

2.2. *Sequence of operations*

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; a typical value of n is usually four (Goldberg 1989). Finally, a message is chosen to give the output through the correspondent interface. The sequence of operations is summarized in table 1.

In the credit assignment level, a reinforcement algorithm (called the bucket brigade (BB) (Holland 1986b)) is used to solve the credit assignment problem: how to

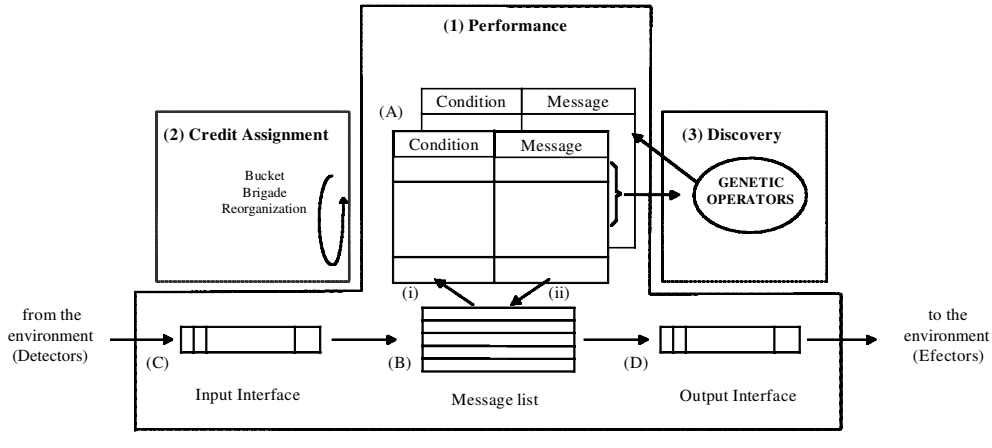


Figure 1. Representation of a CS. (i) All messages are tested with all classifiers. (ii) Winning classifiers post their messages to the message list.

Table 1. Representation of a sequence of operations in a CS.

Step	Operation
1.	A codified message of length k arrives from the environment through the input interface
2.	Clear the message list
3.	The message is set in the message list
4.	All the classifiers that match some message of the message list are fired. Several rules could be activated in parallel by a message
5.	When a condition is satisfied, the message is posted to the message list
6.	Steps 4 and 5 are repeated for n internal cycles
7.	Finally, a message is chosen to give the output through the correspondent interface

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 the *strength*, which 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 classifier responsible for their activation, and their messages are posted to the message list.

A GA is used in level (3) 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 rule's strength.

3. Definition of a reactive classifier system

In order to develop a CS able to react, the necessities of a reactive controller must be analysed. A reactive system obtains a new output for each new environmental information sending by the sensors. In this way, a decision cycle in a generic robot could be defined as is shown in table 2.

This process fixed the time range of reacting to environmental changes. The sequence of operations in a traditional CS only consider a new message, as shown in step 1 of table 1; from this point all the decisions are taken internally without new environmental information. The necessity of reacting leads the search for a new mechanism in CSs that allows us to include new environmental codified message in each internal cycle of the performance level.

Table 2. Sequence of operations of a decision cycle in a reactive system.

Step	Action
1.	Read the sensors
2.	Codify the sensors' information to obtain inputs for the system
3.	Apply the rules over the inputs to obtain a new output
4.	Decodify the output in numerical values
5.	Write the numerical values over the actuators
6.	Go to step 1

The application of a 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 provide a chain of 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 table 1) makes the learning process of a reactive controller difficult. On the other hand, internal cycles are necessary to develop more complex action sequences. The designed RCS, proposed in this work, modifies the performance level to include the possibility of both actions and reactions. In § 3.1, this new architecture is described. The special mechanisms, included in this architecture, modify the sequence of operations of a traditional CS (see § 2.2). This new sequence is presented in § 3.2.

3.1. *Architecture*

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

(1) environmental information;

(2) information related to rules fired in a previous instant (internal conditions);

(3) Information about the decisions.

As can be seen in figure 2, rules in a RCS are composed of two parts: condition and message. They are divided into three blocks. Each condition block is a string over the alphabet {0, 1, #} (as in a traditional CS). The first message block is a string over the alphabet {#} because this block overlaps environmental information in each cycle. Then, the first block of the message, namely the environmental

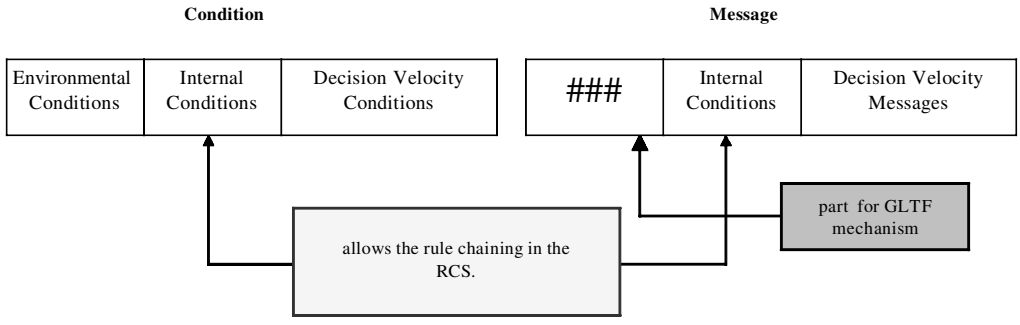


Figure 2. Composition of conditions and messages.

block, is empty and is used to fuse the environmental message with messages of previous activated rules. The rest of message blocks (two and three) are strings over the alphabet $\{0, 1\}$.

The complete sequence of operations will be explained in more detail in § 3.2. This fusion mechanism allows the controller to learn complex actions, composed of a sequence of actions. Besides fused messages, another message with only the first block of message, namely the environmental part, is posted to the message list. This mechanism allows learning reactions, breaking the chain of rules.

In figure 3 the performance level of the RCS 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*

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 of 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 from the next environmental input. The rule chain is over different inputs, using internal conditions and message fusion, allowing it to learn the reactions and actions sequence. The sequence of operations is summarized in table 3 and figure 4.

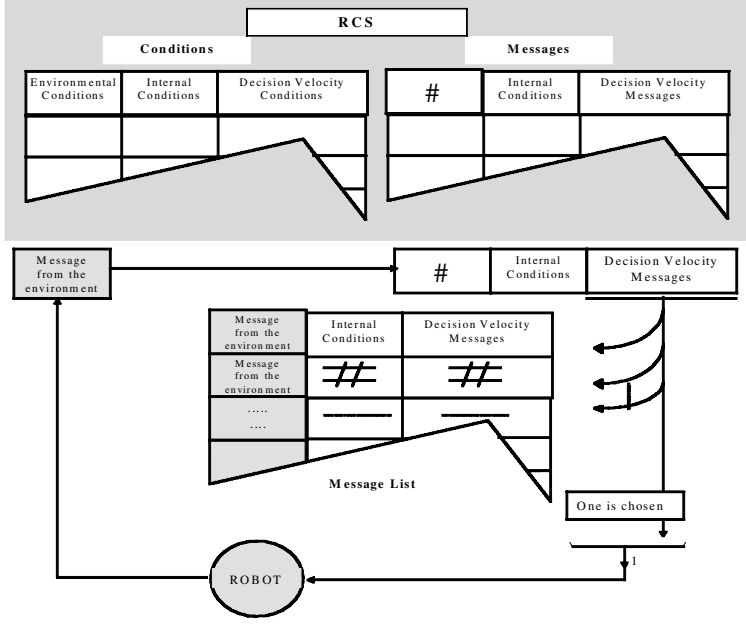


Figure 3. Developed CS and information interchange between the robot and the CS.

Table 3. Representation of a sequence of operations in a reactive classifier system.

Step	Operation
1.	A codified message arrives from the environment through the input interface
2.	The environmental message is fused with messages of previously activated rules
3.	The environmental message without fusion is also posted to the message list
4.	All the classifiers that match some message of the message list are fired
5.	All the messages of fired classifiers are posted to the message list
6.	A message is chosen among the rules that satisfied the conditions

These sequences of operations are related to the performance level. The credit assignment level will be the level that decides what activated rules win in the competition, in the same way as in the traditional CS. This sequence presents two main differences from the traditional CS.

(1) *Generation of message list through fusion (GLTF).* Steps 2 and 3 in the traditional CS, 'clear the message list' and 'the codified environmental message is posted to the message list' are translated into two new operations in an RCS: step 2, 'fusion of the new message with previous messages', and step 3, 'post a new message'.

(2) *No internal cycles (NICs).* Step 6 in the traditional CS, 'the repetition of steps 4 and 5 for n internal cycles', is not necessary because chaining of the rules is performed in each cycle of the performance level.

The loss of the internal cycles (NIC) breaks the rules sequence so characteristic of the traditional CS. To permit chaining of the rules the codification of the rules in the RCS has been modified. Additional information related to the rules fired in the previous instant has been included. This new information is called internal tags (ITs).

In this way, chaining of the actions is obtained, taking into account two special mechanisms in conditions and messages: the environmental message is fused with the previously posted messages (GLTF) and internal conditions are added to evolve a chain strategy. This strategy allows a chain of rules to be activated by the environmental message with previous activated rules. In addition to the environmental message fusion, the RCS

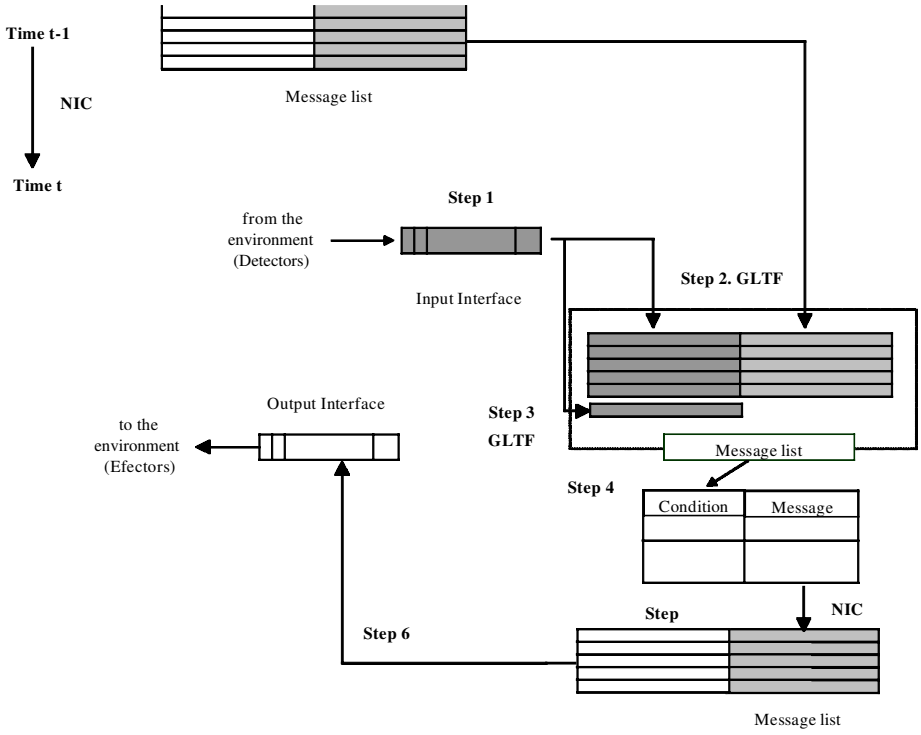


Figure 4. Sequence of operations in a decision cycle in an RCS graphically including GLTF and NICs mechanisms. See text for explanation of the GLTF and NIC mechanisms.

requires the inclusion of internal conditions that provide evolution of a chain strategy. Fusion is the method that allows a chain of rules and the internal conditions support knowledge about the relationship between rules. The evolution process over the internal conditions provided by the GA leads to learning 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 3). The matching process considers environmental conditions only and the system is able to break the chain of rules 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 follows:

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

4. Experimental environment

4.1. *Robot description*

The codification of information in the CS (the design of environmental and output messages) is based on the particular problem where the CS will be applied. In this work, the RCS is used as a controller of an autonomous robot named Khepera (Mondada and Franzi 1993). The mini-robot Khepera is a commercial robot developed at LAMI (Ecole Polytechnique Fédérale de Lausanne, Switzerland). The robot characteristics as follows: a circular shape of 5.5 cm diameter, 3 cm height and 70 g mass. The sensory inputs come in from eight infrared proximity sensors. These sensors are composed of two devices: an infrared emitter and a receiver. The emitter and the receiver are independent, therefore 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 measurement is a function not only of the distance to an object in front of the emitter but also of the environmental light and the object nature (colour and texture). So the value of distance is modified by the measurement of the ambient light and the object nature, the light used is constant and all the obstacles used have the same colour 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 5). The amount of light received in the sensor depends on the distance of the light source. The response curve of each real sensor is described by a sigmoidal function (Mondada and Franzi 1993). When the robot is placed near a light source (figure 5), each sensor gives a value of light intensity based on the sigmoidal function. In figure 5, an example of the different values in each sensor is represented. In this case, sensor 6 returns the minimum value from all sensors, the value is used to obtain the distance and the sensor number (6 in this case) to obtain the angle to the light source.

The domain of real values returned by the sensors is $(0, 1023)$; in this work a linear transformation function has been used to redefine the domain to $(0, 40)$ for proximity sensors and $(0, 500)$ for the distance to the light source. The desired angle is obtained by considering the sensor number with the minimum intensity value of the ambient sensors (table 4).

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 infrared sensors individually, they have been grouped to give a unique value, obtaining the average from two sensor-input values (figure 6(*a*)), and reducing the amount of information received by

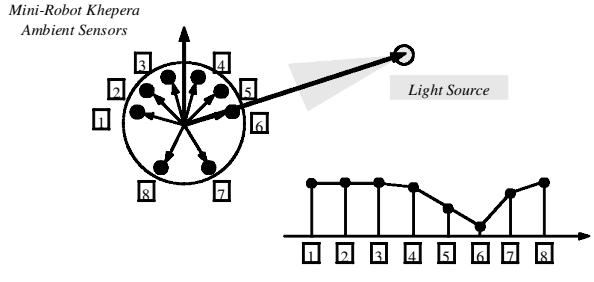


Figure 5. Incoming light distribution in the sensors.

Table 4. Desired angles for sensors.

Sensor number	1	2	3	4	5	6	7	8
Angle (degrees)	90	45	15	345	315	270	190	170

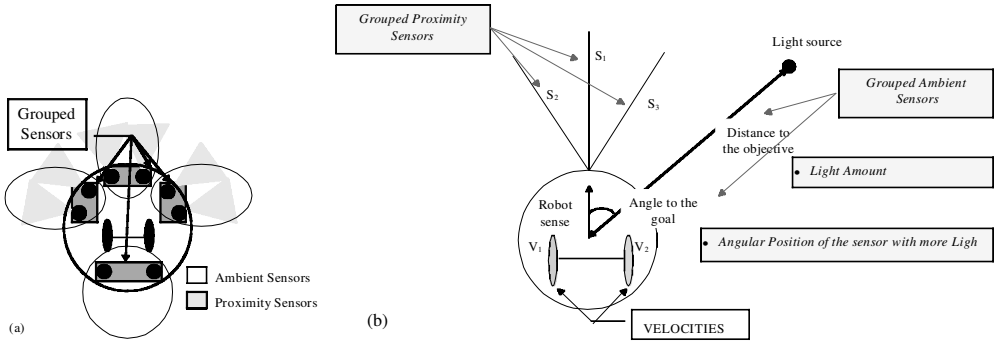


Figure 6. (*a*) Sensors considered in the real robot. (*b*) Input information to the system.

the RCS. 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) to this goal (figure 6 (*b*)).

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

4.2. *Environmental Description*

Experiments require a long time with continuous functioning of the hardware. In order to prove the different configurations of CSs, both traditional CSs and RCSs, a simulator developed in previous work (Sommaruga *et al*. 1996) has been used: SimDAI. In the simulator, the characteristics of the turtle robot model (McKerrow 1991) and the physical restrictions of the Khepera robot have been considered. SimDAI is a working prototype of a mobile robot's simulation environment for experimenting with robot navigation and control algorithms. Each mobile robot is completely independent, can navigate and interacts with other robots in a two-dimensional simulated world of obstacles, which is separately monitored. This simulator has been used in many other studies (Sanchis *et al*. 1996a, b, Isasi *et al*. 1997, Molina *et al*. 1997, Matellán *et al*. 1998).

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 7).

Different simulated worlds which resembles the real world have been defined in order to tune the pay-off from the environment before being implemented in the real world. An example of these environments can be seen in figures 7 and 8. The system developed is the same in both cases (simulated and real) except for the differences in the treatment of the sensors, by the transformation function.

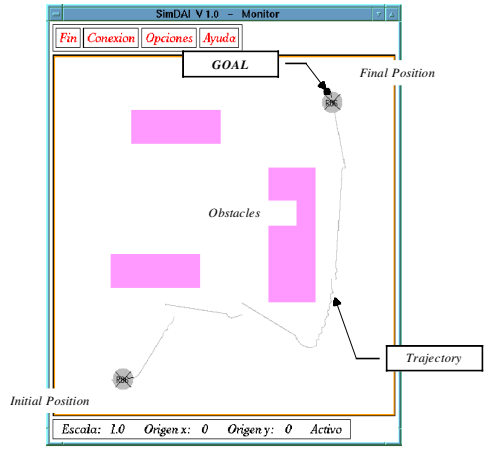


Figure 7. SimDAI simulator (example of one simulated environment).

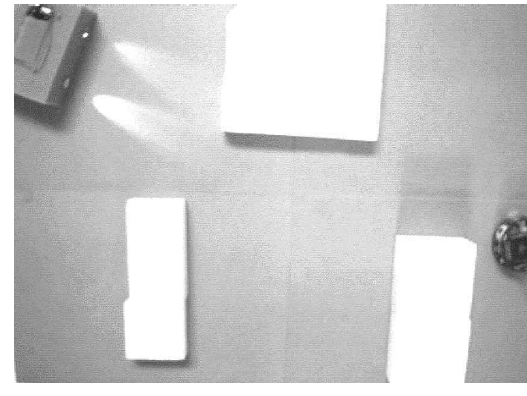


Figure 8. Example of a real experimental environment.

4.3. *Environmental and output messages*

The conditions and messages of the RCS described in the previous section are divided into three parts. The environmental part of conditions and the decision velocity part of messages concerns the robot state.

The environmental part should be matched to 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 9 (*a*). The first part of the message is composed by the proximity sensors to describe the near environment surrounding the robot. The second part corresponds to the goal description using the angle and distance information. The last part of the 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. The velocity values are decodified and applied to each wheel in the robot (figure 9 (*b*)).

4.4. *Codification*

As has been explained previously, the distance domain of a real robot has been 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 into four crisp sets with the same width. The maximum distance value 'seen' by one sensor is 40 units and is divided into ranges as shown in figure 10.

The angle sets are of different sizes to allow fine fitting of the trajectory, avoiding large oscillations when the robot follows the correct direction. The sets near 0 and 2π are smaller than the '$< \pi$' and the '$> \pi$' sets. This definition allows better navigation properties, adjusting the robot sense right up to the objective, avoiding oscillations. The input domain partitions are presented in figure 11.

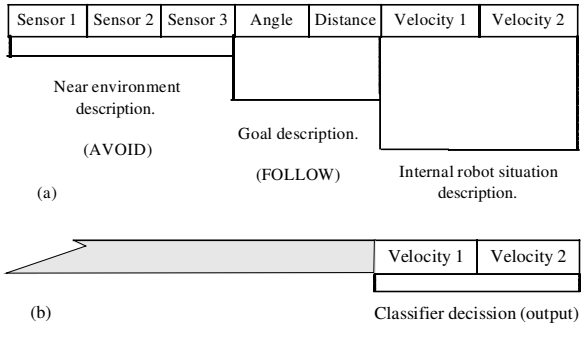


Figure 9. (*a*) Composition of the environmental message. (*b*) Decision velocities in the output message.

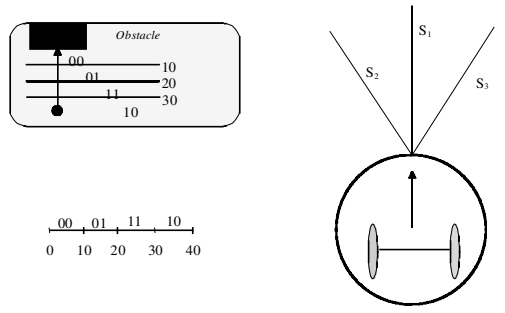


Figure 10. Codification and partition of the proximity information.

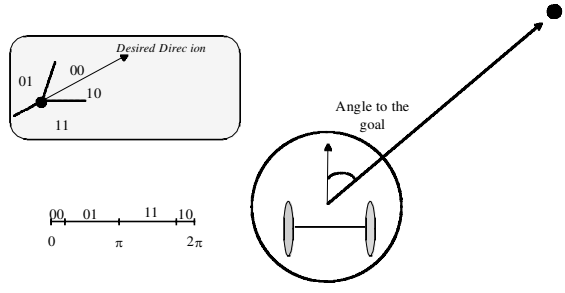


Figure 11. Codification and partition of the angle information.

To keep the independence of the 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 12. These sets are defined considering the distance over 200 as very far from the objective and partitioning the distance between 0 and 200 into three sets. The second set is 50% of 200 (from 200 to 100) and it is defined as far. The third set is defined from 100 to 25, to represent that the robot is near the objective and the last set represents that the robot has reached the objective.

Velocity values flow as the input to the classifier system and as the decision from the CS to the robot.

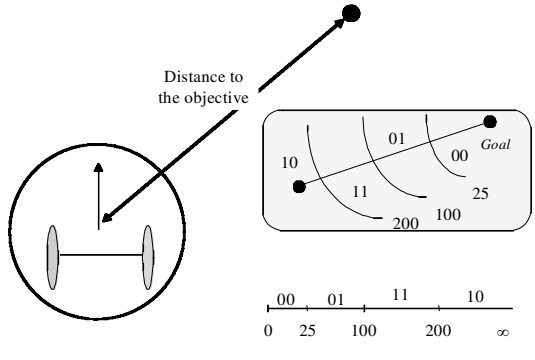


Figure 12. Codification and partition of the distance information.

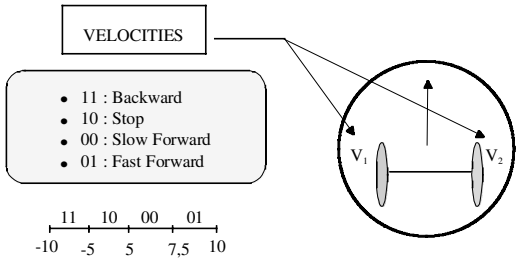


Figure 13. Codification and partition of the velocity information.

The values are defined by the maximum and minimum velocities (10, −10). This range is divided into four equal sets as is shown in figure 13.

The 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 10–13.

5. Experimental results

Learning reactions in a real robot by means of a CS involve two main tasks: first, to discriminate the better rules from a set of rules and second to discover new rules to face new situations or to improve its performance. In order to test the performance of the RCS, different types of experiments have been carried out.

(1) The first is related to adjusting the environmental pay-off.

(2) The second is related to the validation of BB algorithm in the RCS. In this way, a discrimination experiment between rules, without the addition of new rules, has been carried out. So an RCS with a constant set of *ad hoc* rules and a complex environment where most types of situation could be found has been used.

(3) The third measures the improvements produced by this new approach (RCS) compared with the traditional perspective (CS). The experiment has been carried out with and without background knowledge and it has been executed in the simulator because of the long time of functioning needed. In order to compare the two systems, the traditional CS and RCS have been started with random rules.

5.1. *Payment function adjustment*

To estimate the function, the objective factors that can give an idea of whether an action has been correct or incorrect has been considered. In a navigation problem to a goal through an environment with obstacles, those actions that permit the robot not to collide will be considered as positive. This is the case, for example, when increasing the distance to some obstacle or when approaching the goal. The alignment of the robot in the goal direction is another positive action. Negative actions are those that remove it from the goal, for example the increase in the distance travelled by the robot, or that cause it to approach obstacles, for example that may have produced some collision. So, the considered factors to calculate the payment are the increase in the distance to an obstacle, the approximation to the objective and the alignment or drift towards the objective.

If the three previous situations occur, payment would be positive but, in most cases, fulfilment of these points implies non-fulfilment of others; therefore these factors will have to be weighted. The function chosen to calculate the payment is given by which will constitute the final payment through

$$P_s = P_1 \times \mathrm{coef}_1 + P_2 \times \mathrm{coef}_2 + P_3 \times \mathrm{coef}_3, \quad (1)$$

$$P_T = P_s K_s, \quad (2)$$

where P_1 corresponds to the approximation to the objective. Its value is determined by the difference between the distance in the previous execution cycle and the current distance. Coef_1 is a constant applied to P_1. P_2 corresponds to the alignment towards the objective. Its value determines the difference between the angle in the previous cycle and the current angle, in radians; it is positive if turns towards the objective and negative otherwise. Coef_2 is a constant applied to P_2. P_3 corresponds to the distance to objects. It is calculated by evaluating the values for the left sensor S2 and the right sensor S3. If the value for S2 is less than for S3, it is paid by turning to the right and, if it is the opposite, it is paid by turning to the left. If the turn is wrong, it is penalized in the same quantity. If the values for S2 and S3 are equal, neither is paid nor is penalized. Coef_3 is a constant applied to P_3. P_s collects the result of previous payments. K_s is a constant applied to P_s. P_T is the final payment. Different constants are employed to obtain the appropriate influence of each of the factors without distorting the strengths.

A collision with an object is not included in the previous function. In this case, a punishment greater than any other case is applied, with a fixed value of P_s since, as there is no movement, there is no evaluation, nor turns, nor approximations.

Fitting and correct selection of this function will determine the success of this kind of system; so the steps that have enabled the result of the selected function to be proved valid will be described. A set of *ad hoc* elected classifiers is used and no calls to the GA are accomplished; so the discovery phase of the RCS is annulled in order to study the relationship between the action and reassignment of credit levels. Besides no calls

to the GA, the value of the percentage of payment between rules, the bids in the credits reassignment algorithm, has to be reduced to ensure that payment among rules does not hide the payment function effect. In this way, the growth or decrement of classifiers strength will be, fundamentally, due to external pay, which it is required to adjust.

It is necessary to include in an RCS all classifiers, containing all possible conditions and all possible actions (messages) for each condition. Thus, for each possible condition 16, different messages are generated (the speed of each wheel is codified with 2 bits, and then 2^4). Once all the possibilities have been taken into account, when executing the RCS, the strengths of better classifiers would have to increase and reduce the values of the worse classifiers. This happens until each of the classifiers obtains a strength that reflects, in a real way, their usefulness in the system.

An important problem, related to the number of classifiers necessary to reproduce all the possible situations, appears. This number is calculated considering the number of bits involved in each possibility. It will be n, the total number of necessary classifiers: $n = \text{comb\_S1S2S3} \times \text{comb\_AngDist} \times \text{comb\_V1V2}_{\text{dec}}$, with $\text{comb\_S1S2S3} = 2^6, \text{comb\_AngDist} = 2^4$ and $\text{comb\_V1V2}_{\text{dec}} = 2^4$. This produces $n = 2^{14} = 16\,384$ different classifiers. This is an excessively high number of rules that would be impossible to handle, even in the most potent machines of today, which makes necessary to utilize to some other system.

The proposed solution is to divide the classifiers of the RCS into two groups: one for following rules and the other for avoiding rules. Thus, classifiers of the following group will have in their conditions combinations corresponding to the following part and the rest with the symbols #, and classifies of the avoiding group will have the same but with the avoiding conditions. Thus the number of rules of the RCS would be $n = (\text{comb\_S1S2S3} + \text{comb\_AngDist}) \times \text{comb\_V1V2}_{\text{dec}}$. This corresponds to 1280 different classifiers and is also an excessively high number of classifiers; therefore it has been decided to eliminate those classifiers whose conditions will be redundant. For example, if one of the sensors perceives that there is an obstacle very near, 00 in the corresponding position, values for the other sensors, distance and angle lose importance since the results are necessary to turn in opposite sense and can be removed from the object. Thus, three conditions appear (table 5).

In this way, 544 classifiers are eliminated. Sensors S2 and S3 detect the presence of obstacles by the left and right respectively of the robot; if some obstacle apears in those positions, it is necessary to avoid the obstacle, independently of the rest of the values. Thus, the conditions in table 6 remain. This reduces the number of classifiers by 192.

With respect to the following part in the conditions, it is only possible to reduce the part that considers the minimal distance to the objective, without considering the angle, since it is considered that the objective has already been reached. In a first approximation this seems to say that, when the robot is at a very small distance to the objective, without considering the direction, the navigation problem will have been solved. Evidently, with the existing rules that previously will have caused the robot alignment towards the objective, when the robot is very near the objective, the direction is therefore irrelevant. Thus the conditions in table 7 remain. This reduces the number of classifiers by 48. Finally, classifier number will be $n = 1280 - 544 - 192 - 48 = 496$.

It is not possible to reduce the number of classifiers more; however, 496 is still an excessively high number for an RCS applied to a reactive problem. So, four RCS, each containing 124 classifiers, are going to be used and a competition between them is going to be held. It is necessary to ensure that, for each possible classifier condition, all the possible movements of the wheels are represented. Thus, for each condition the actions given in table 8 are fixed.

Finally, in trying to obtain a generalized solution, 15 executions are accomplished over each RCS, with three different initial situations of the robots. First the robot in position 1 is considered, second that in position 2 and third that in position 3; the process is repeated five times, to obtain 15 executions. Initial values of the three robots are given in table 9.

The competition between all possibilities is held in order to adjust the function. Some initial RCSs have been defined that contain rules groups and, through each execution, new RCSs have been generated containing better rules than the previous rules. The compe-

Table 5. Conditions of rules considering 00 information in the sensors.

S1	S2	S3	Angle	Distance	V1	V2	CI	$V1_{dec}$	$V2_{dec}$
00	##	##	##	##	##	##	##	##	##
##	00	##	##	##	##	##	##	##	##
##	##	00	##	##	##	##	##	##	##

Table 6. Conditions of rules considering 01 or 10 information in the sensors.

S1	S2	S3	Angle	Distance	V1	V2	CI	$V1_{dec}$	$V2_{dec}$
##	01	##	##	##	##	##	##	##	##
##	##	01	##	##	##	##	##	##	##
##	01	01	##	##	##	##	##	##	##
01	10	10	##	##	##	##	##	##	##
01	10	11	##	##	##	##	##	##	##
01	11	10	##	##	##	##	##	##	##
01	11	11	##	##	##	##	##	##	##
10	10	10	##	##	##	##	##	##	##
10	10	11	##	##	##	##	##	##	##
10	11	10	##	##	##	##	##	##	##
10	11	11	##	##	##	##	##	##	##
11	10	10	##	##	##	##	##	##	##
11	10	11	##	##	##	##	##	##	##
11	11	10	##	##	##	##	##	##	##
11	11	11	##	##	##	##	##	##	##

Table 7. Conditions of rules considering information about the angle and the distance in the sensors.

S1	S2	S3	Angle	Distance	V1	V2	CI	$V1_{dec}$	$V2_{dec}$
##	##	##	##	00	##	##	##	##	##
##	##	##	00	01	##	##	##	##	##
##	##	##	01	01	##	##	##	##	##
##	##	##	10	01	##	##	##	##	##
##	##	##	11	01	##	##	##	##	##
##	##	##	00	10	##	##	##	##	##
##	##	##	01	10	##	##	##	##	##
##	##	##	10	10	##	##	##	##	##
##	##	##	11	10	##	##	##	##	##
##	##	##	00	11	##	##	##	##	##
##	##	##	01	11	##	##	##	##	##
##	##	##	10	11	##	##	##	##	##
##	##	##	11	11	##	##	##	##	##

Table 8. Fixed actions for RCS1, RCS2, RCS3 and RCS4.

RCS1		RCS 2		RCS 3		RCS 4	
V1	V2	V1	V2	V1	V2	V1	V2
0.5	0	0.5	−1	0.5	0.5	0.5	1
1	0.5	1	1	1	−1	1	0
0	−1	0	0	0	1	0	0.5
−1	1	−1	0.5	−1	0	−1	−1

Table 9. Initial values of the three robots.

Robot	*X*	*Y*	Sense
Robot 1	50	400	0
Robot 2	300	450	180
Robot 3	50	150	0

tition process is defined in the following way: once RCS1, RCS2, RCS3 and RCS4 have undergone 15 executions (five of each for three different initial situations of the robot), better actions for each condition of each RCS are chosen (those with higher values of strength). Thus, starting from four RCSs, two will be obtained, named RCS5 and RCS6. Repeating the process with these two new RCSs, finally, RCS7 is obtained,

which contains better classifiers of the total. Repeating the RCS7 execution and choosing the better two classifiers for each possible condition, RCS8 has been obtained, which contains 62 classifiers. In figure 14 a scheme for the selection process of RCSs for the adjustment of the payment function is shown.

The empirically obtained payment function is described in equations (1) and (2). Different contributions are as follows: contribution 1 referred to the approximation to the objective, contribution 2 is related to the angle between the robot and the objective, and contribution 3 is related to the distance to the obstacles. Through experiments, it has been determined that contributions 1 and 3 depend on the situations of the robot. Therefore the values of the Coef_1 and Coef_3 in equation (1) depend on each situation and will be divided into two constants: one fixed and other dependent of the situation. With this consideration, the selected function for the payment calculation is given by the following equation (the final payment will be calculated using equation (2)):

$$P_s = P_1 K_1 C_1 + P_2 K_2 + P_3 K_3 C_3, \qquad (3)$$

where P_1, P_2, P_3 and P_s are the previously described parameters. K_1 is a fixed value applied to P_1. C_1 is a variable value applied to P_1, a function of the nearness to the objective. Its values are 4 if the robot is very near to the objective, 2 in the intermediate case and 1 if it is far from the objective. K_2 is a fixed value applied to P_2. K_3 is a fixed value applied to P_3. C_3 is a fixed value applied to P_3, whose value depends on the distance to objects. Its value is 8 if it is very near to the object to avoid, 4 in the following distances section and 1 in the rest.

The values obtained for the function constants are given in table 10. Despite the fact that these parameter values have been calculated empirically, they reflect the corresponding importance of each of the contributions. Thus, a higher and, therefore, more important value has to deal with the withdrawal of the obstacles; this value is tenfold greater than the corresponding factor for the contribution from being aligned with the objective. Although the approximation factor to the objective could seem to be smaller, the angle magnitudes are less than those of the travelled distance. Furthermore, this difference in magnitude is taken into consideration through the constant C_1, according to where the robot is found. Thus, when the robot is near the objective, this constant causes the coefficient for the approximation, Coef_1, to be equal to the alignment coefficient, Coef_2 while, when it is far away from the objective, both coefficients are specified by K_1 and K_2, maintaining a magnitude relationship.

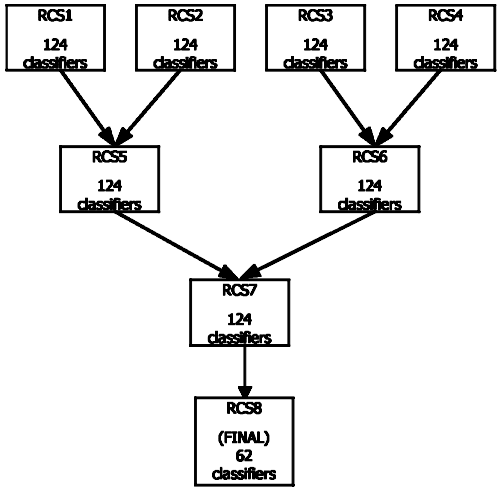


Figure 14. RCS selection process scheme.

Table 10. Values obtained for the function constants.

K_1	K_2	K_3	K_s	P_s
0.3	1	3	0.02	−10 (if collision happens)

5.2. *Validation of the bucket bridgade algorithm for the 'generation of message list through fusion' mechanism in a reactive classifier system*

The system learns from an initial situation using the payment function, namely the BB algorithm (to distribute the rule strength when a rule is activated by another rule). A function that ensures that the strength of the not-fired rules decreases to differentiate them from the fired rules, commonly used in CSs (Dorigo 1995, Holland 1986a), is also used in this work. This decrease in rule strength is called a tax.

Experimental results show a learning behaviour where the strengths of the best rules for the problem increase while the strengths of the other rules decrease versus cycles in the execution. The set of rules for the CS is collected in table 11.

These rules could be clustered into three groups (table 11). The first group *a* is related to situations in which there is collision danger. With these rules, the robot turns in the correct direction in the presence of obstacles. The 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 is no collision danger. This set of rules allows the robot to wander around the experimental environment without taking into account the goal. The last group c consist of rules that, independently of obstacle position, change the trajectory of the robot facing the goal.

Table 11. Rules of the CS: group a, avoiding when obstacles are near; group b, avoiding when obstacles are far; group c, following.

Condition	Message
Group a	
##0000########10####	##############111111
##00##########10####	##############111011
####00########10####	##############111110
000101########10####	##############111111
001101########10####	##############111100
001001########10####	##############111100
000111########10####	##############110011
001111########10####	##############111111
001011########10####	##############111000
000110########10####	##############110011
001110########10####	##############110010
001010########10####	##############111111
Group b	
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
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#####	##############000001
100110########1#####	##############000100
101110########1#####	##############000100
101010########1#####	##############000101
Group c	
########00####11####	##############111010
######0101####11####	##############110011
######1101####11####	##############111100
######0001####11####	##############110010
######1001####11####	##############111000
######0111####11####	##############110011
######1111####11####	##############111100
######0011####11####	##############110010
######1011####11####	##############111000
######0110####11####	##############110010
######1110####11####	##############111000
######0010####11####	##############110100
######1010####11####	##############110001

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

In more detail, the meaning of rule (1) in figure 15 is explained. *If* there is an obstacle in front of the robot at a distance between 0 and 10 (S1 = 00, very near), another obstacle on the left at a distance between 20 and 30 (S2 = 11, far), 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 to 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 (which is not very useful) and they are not able to follow the goal. The most efficient strategy is to follow the goal except when there is collision danger. This is accomplished by the rules of groups a and c, in an efficient way when the appropriate rules from these groups are selected.

The CS with 52 rules has been applied to the real robot, with different initial strengths (table 12). The CS shows the capability of discriminating between the three groups of rules. It has been experimentally tested that rules belonging to groups a and c have an average strength above the 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 between rules inside each set. Some rules of the set are better (more 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 provide a chain of rules for 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 increase in strength takes place in a short number of cycles as can be seen in figure 16(*a*). The meaning of evolution of strength is that, while the special conditions for these rules to be useful are not yet reached, their strength is decreased as an effect of the tax mechanism. On the other hand, when they are fired (or could be fired), and because they can solve a great part of the problem, they will be fired once and so on, to increase in strength quickly. If the rules are fired as a part of a chain in execution, their strengths are kept constant, more or less, as a result of composing the growth for firing and the loss for taxes. When the robot faces a complex situation, any rule tries to solve the problem in isolation. In this case, the strength of good rules increases or decreases depending on whether it takes part or not in the chain that is being executed and that execution ends with a positive or negative payoff (figure 16(*c*)). Finally, all rules have a tendency to

	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.

Table 12. Average strength of groups a, b and c with different initial strengths.

Initial Strength			Final Average Strength		
Group a	Group b	Group c	Group a	Group b	Group c
300	300	300	365	202	340
300	300	600	370	210	502
300	600	300	325	305	310
300	600	600	310	315	520
600	300	300	640	201	340
600	300	600	635	204	550
600	600	300	615	295	340
600	600	600	650	276	545

decrease their strength as an effect of the taxes. This is more evident when there are no necessary rules (figure 16(*b*)).

To check the robustness of the developed CS, some experiments have been made on the real robot, considering different initial strength. A similar set of rules has been discriminated. The CS selects the better rules; these rules belong to the clusters a and c where they represent the rules that avoid dangerous situations and follow the correct path in the 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 has been done as follows: if the rule belongs to clusters a and c, it is set to 200 and to 600 otherwise (cluster b). In these conditions,

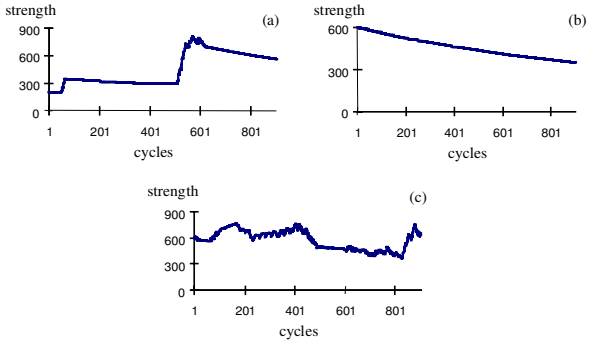


Figure 16. Evolution of the rules strength of (*a*) avoiding with obstacles near, (*b*) no obstacles and (*c*) following rule.

evolution of good rules is much more difficult because they have, initially, a third part of the strength.

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

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. Evaluation of the system performance is based on a quantitative measure. This measure does not take part in the evolution process but it reflects the system's global performance evolution. To measure the system evolution, the following features have been considered:

(1) The time needed to reach the goal (seconds in the real robot and cycles in the simulator);

(2) The trajectory length (measured by means of the velocity values of the motor wheels);

(3) The number of collisions (measured using the minimum value of the proximity sensors).

The environment makes the RCSs adapt to the set of rules that cooperate to achieve a common goal in order to perform incrementally the 'reach-and-avoid' behaviour with less time, fewer collisions and a straighter trajectory to the goal.

There are several approaches for initializing rules in CSs. The most common method is random initializa-

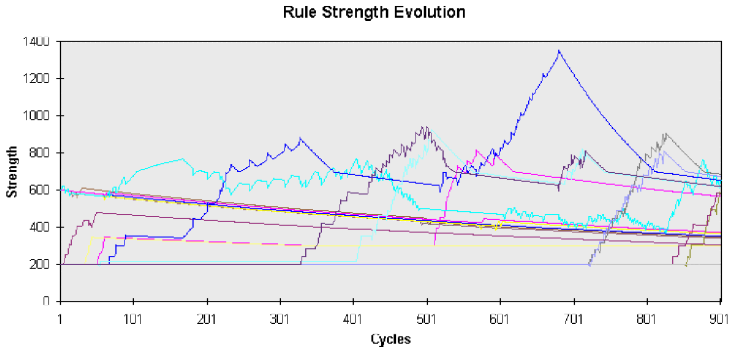


Figure 17. Evolution of the strength of the whole set of rules over 1000 cycles.

tion. This represents the maximum challenge to the learning algorithm but does not 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 (Schultz 1991).

The first experiment, where an RCS has been seeded with previous domain knowledge, has been carried out to prove the effect of the RCS as a method of selecting and incorporating the necessary knowledge for solving the problem. The initial population of the RCS is composed of the ten better rules obtained in previous experiments (the rules with higher average strength; see § 5.2) and, for each rule, four random new rules have been added; so 50 rules are obtained. The GA is utilized at the end of an execution. The robot navigation in an execution starts from an initial random point and it ends when the goal is reached, or the time exceeds some limit, or the number of collisions exceeds a maximum threshold. A problem of the Michigan approach (Holland 1986a), which is the same as in a CS, is the sensitivity of the rules. This is because the strength of some rules depends on the strength of other rules. To overcome this problem, a high degree of overlapping has been used in such a way that the difference between generations is insignificant. The selected parameters of the GA are a cross-over probability of 1 a mutation probability of 0.02 and overlapping of 0.85. The effects of removing some of the *ad hoc* rules (§ 4.2) are reflected in the ability of the robot to reach the goal; thus is achieved in an inefficient way. Through evolution some new good rules have been added, making the robot follow a better trajectory without collisions.

On the other hand, it has been intended to prove also the capacity of the system as a method for solving problems without domain knowledge. This second alternative is the most general for problems in which the final objective, namely the rules that govern the process or both, is unknown. For this general case, the experiment is intended to accomplish a comparison with the traditional perspective in order to evaluate performances.

In these experiments, the initial population of the RCS is randomly generated. In this case, the ability and improvement of the RCS to learn reactions compared with the traditional approach can be probed. The parameters of the CS (traditional and new) are the same:

(*a*) the GA which is utilized after 100 cycles of decisions;

(*b*) a cross-over probability of 1;

(*c*) a mutation probability of 0.01;

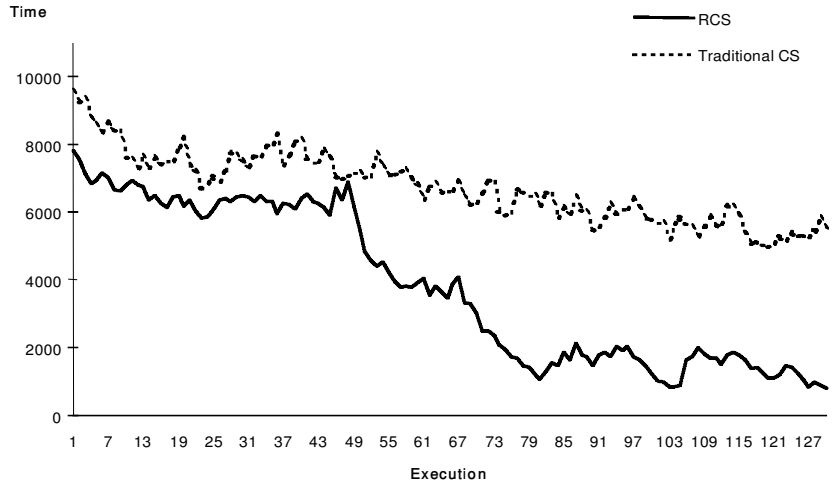
(*d*) overlapping of 0.3.

The value of overlapping is lower than in previous experiments to allow faster generation of new rules because now the initial population is randomly generated. Four internal cycles in the performing level are considered in the traditional CS.

The simulator executes the robot controller as in the real world; so, while traditional and reactive CSs take a decision, the robot is continuously working. The velocity of the robot in this period is the previously decided velocity. This velocity is changed when the CS takes a decision for the incoming environmental message. This consideration is the main feature in a traditional CS because it executes four internal cycles before taking a decision.

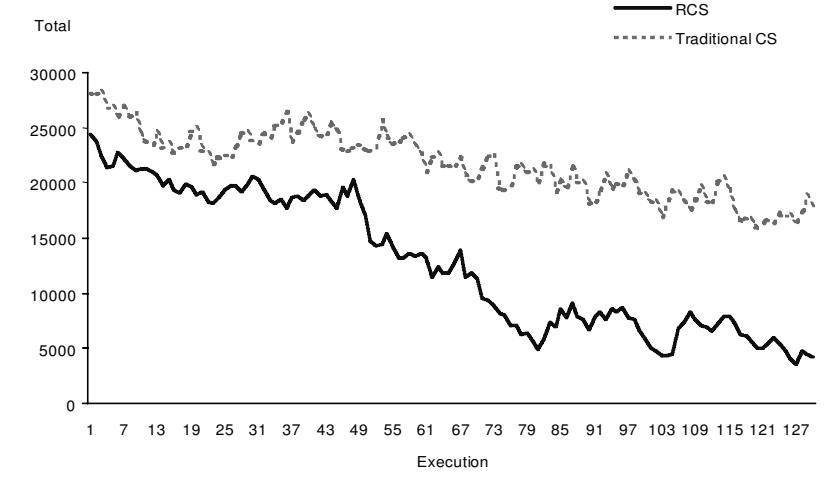
Figure 18 shows the evolution of the evaluation parameters for the two types of classifier. In figure 18(*c*) a function that linearly combines the two parameters is shown, the function is $1.5 \times \text{time} + \text{distance}$.

Figure 18 shows the results of 50 experiments. In these experiments, the seed required to create the populations is changed in each generation, therefore, each experiment has a different set of initial rules. On the x axis, executions are represented. An execution is the navigation of the robot from the initial situation until the goal is reached. On the y axis the average value over 50 experiments of the measured variable is represented



(*a*)

(*b*)



(*c*)

Figure 18. (*a*) Time to reach the goal by the RCS and the traditional CS. (*b*) Distance to reach the goal by the RCS and the traditional CS. (*c*) Global evaluation of the two systems.

both for a CS and an RCS. The variance values of these experiments are limited to 10% of the average value at each point of the curves.

Figure 18(*a*) shows the time in finding the goal. It can be seen how the rule learning process causes the robot to find the goal more rapidly in both cases. However, while the traditional CS causes a decrease of about 30%, the RCS could reach a decrease of 70%. This is because the RCS is able to learn rules that will be fired just in time, because of the lack of delay between a rule execution and its reward from the environment. In figure 18(*b*) the trajectory of the robot is represented. This value is related to the previous value; so the two graphics are expected to have the same shape. The learning of valid chain rules makes the RCS move more rapidly and straighter to the goal than the CS does.

The improvements in the RCS over the traditional CS can be seen in figure 18(*c*), where the effects of the two measurements are combined. The rules achieved in the RCS improve the performance of the robot by 60% compared with the rules obtained with the traditional CS.

6. Testing a reactive classifier system in dynamic environments

6.1. *Simulated experiments*

The proposed RCS has learned to react and to provide a chain of 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 to those in the static experiments but a circular object wanders in 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 by avoiding the mobile obstacle without losing the tendency to arrive at 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 the trajectory lengths, collisions and times. Each experiment is defined by the robot initial position (three different positions have been used: robot 1, robot 2 and robot 3) and the number of static obstacles (one, two or three). Static objects are the same as in figure 7. Each robot is defined by coordinates (x, y) and their initial direction:

robot 1: $x = 50; y = 400$; initial direction, 0°;
robot 2: $x = 300; y = 450$; initial direction, 180°;
robot 3: $x = 50, y = 150$; initial direction, 0°;

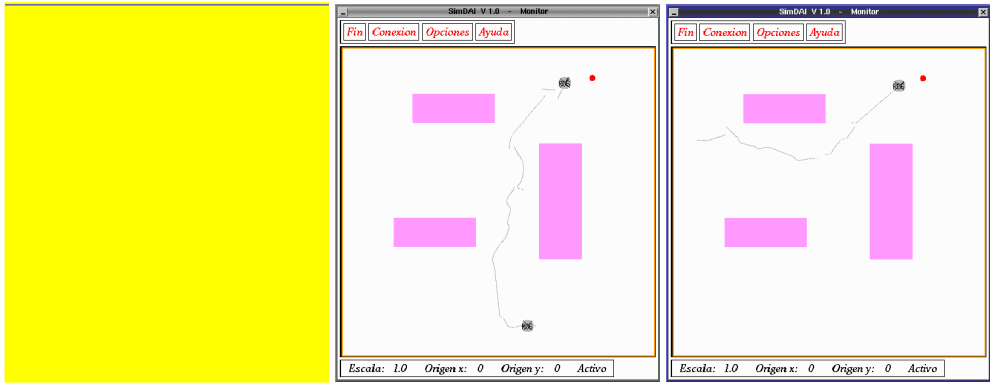
The average results of the CS with respect to time, trajectory length and collisions are shown in table 13 from 50 experiments. Three selected examples of these experiments are shown in figure 19.

In table 14, the numerical values obtained in these dynamic experiments, with a mobile obstacle, are shown. Figures 20(*a*), (*b*) and (*c*) show several trajectories starting from the same point as in figures 19(*a*), (*b*) and (*c*) respectively.

As can be seen in tables 13 and 14, the robot behaviours are similar in different environments. The navigation problem is solved from different initial positions and with different configurations of objects (both static and dynamic). Although the robot arrives at the goal in any circumstance, the results are different in static environments from those in dynamic environments; there are more collisions, and the time is larger and the distance larger in dynamic environments because of the difficulty of the mobile object. The results show that the learning rate of the CS allows the navigation problem to be solved in different environments, both static and dynamic.

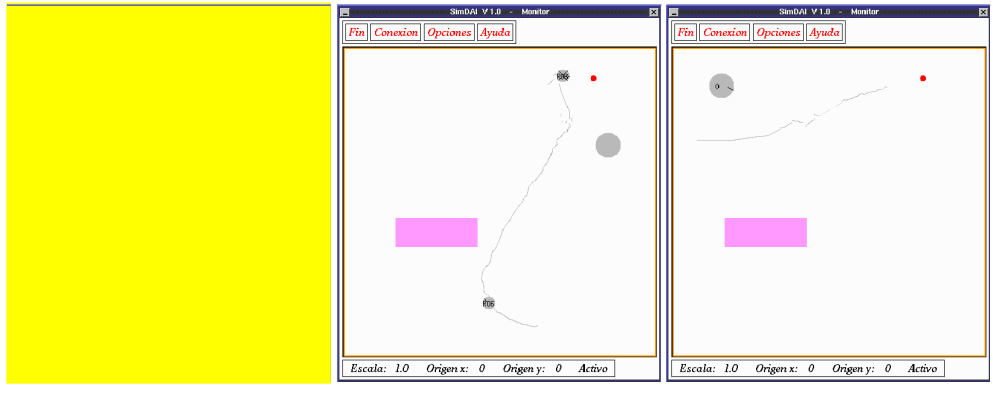
Table 13. Numerical results of static experiments.

Robot	Environment	Average time	Average distance	Average collisions
Robot 1	3 objects	91.80	678.10	0.7
Robot 1	2 objects	82.21	650.30	0.7
Robot 1	1 object	68.32	635.20	0.7
Robot 2	3 objects	86.70	583.10	0.8
Robot 2	2 objects	73.20	523.43	0.3
Robot 2	1 object	62.30	427.80	0
Robot 3	3 objects	69.80	461.50	0
Robot 3	2 objects	27.50	333.20	0
Robot 3	1 object	27.30	332.70	0



(*a*) (*b*) (*c*)

Figure 19. Three static experiments: (*a*) robot 1; (*b*) robot 2; (*c*) robot 3.



(*a*) (*b*) (*c*)

Figure 20. Three dynamic experiments: (*a*) robot 1; (*b*) robot 2; (*c*) robot 3.

Table 14. Numerical results of dynamic experiments.

Robot	Environment	Average time	Average distance	Average collisions
Robot 1	3 objects	157.50	791.51	3
Robot 1	2 objects	158.17	743.34	1.5
Robot 1	1 object	132.08	654.73	1.5
Robot 2	3 objects	154.21	657.62	1.6
Robot 2	2 objects	80.12	580.23	1
Robot 2	1 object	74.14	473.79	0
Robot 3	3 objects	77.33	394.62	0
Robot 3	2 objects	50.67	343.09	0
Robot 3	1 object	51.50	320.70	0

6.2. *Real robot experiments*

Evolved RCSs have been used to control the real robot in different environments. In figure 21, a real experiment is shown; figure 21(*a*) represents the starting point, figure 23(*b*) the intermediate state and figure 23(*c*) the final position.

The experiments accomplished resemble those in the simulator, in order to compare the results obtained by the same CS both in the simulator and in a real environment, with the robot Khepera. The environment consists of several elements:

(*a*) a wood enclosure which is white in colour and 6.5 cm in height, with a perimeter of 70 cm × 70 cm;

(*b*) a bulb at voltage 2.5 V, placed in a foam slab, and fed by a continuous current generator;

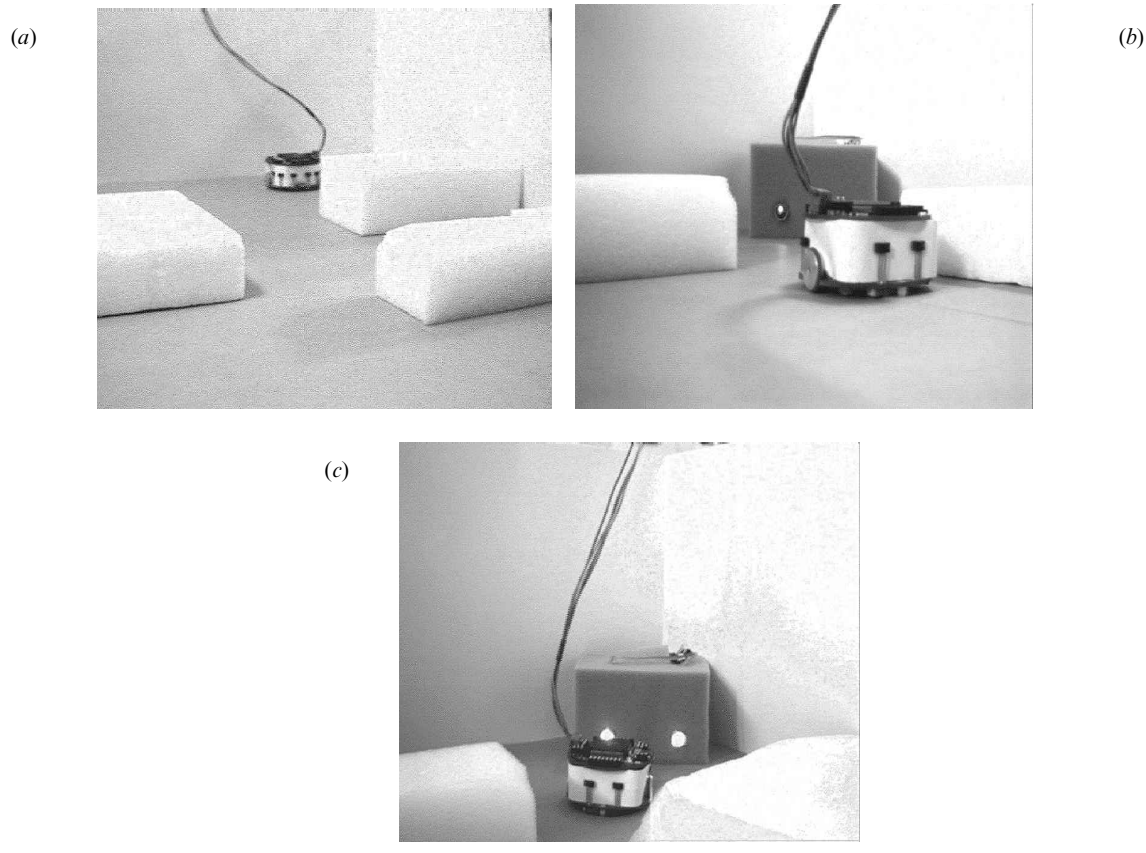


Figure 21. Systems evolution examples in one real experimental environment: (*a*) starting position; (*b*) intermediate; (*c*) goal reached.

(*c*) the surface of the enclosure which is covered with black cardboard, to assess the optimum behaviour of the robot with respect to the source of light;

(*d*) three objects which have been placed in the enclosure in a similar way to in the simulator world, and which are white in colour, and 6 cm. in height, with a perimeter of 10 cm × 10 cm.

In figure 22 a plan of the real environment described is shown. Three different starting positions of Khepera appear. These positions are also similar to those used in the simulator.

Twelve experiments have been accomplished, each consisting of 20 consecutive robot executions. In the experiments, the starting position and robot sense change (positions 1, 2 and 3), and also objects A, B and C, with their possible combinations (including eliminating an object). In each execution, three objective parameters has been collected: the number of collisions produced, the time elapsed until arriving at the objective, in seconds, and the distance travelled, in centimetres. Furthermore, for each experiment, the maximum and minimum values, the average values and the standard deviations have been calculated, for each of these three parameters. In figure 23, some comparative tables are shown.

As can be observed in these obtained data, an ROCS on a real robot operates almost without collisions in all situations and reached the goal in relatively short times

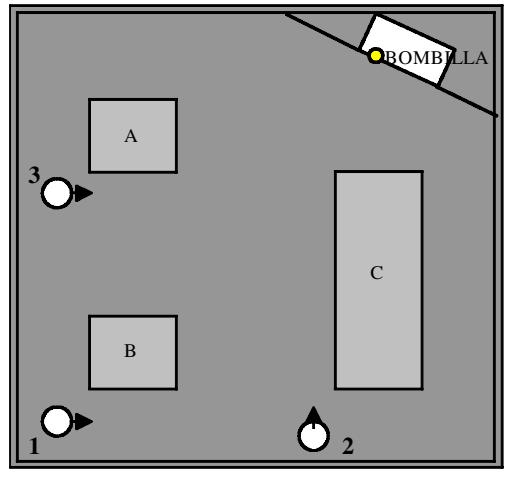


Figure 22. Scheme of real robot experiments.

	POS 1 ABC			POS 1 AB			POS 1 AC			POS 1 BC		
	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis
Maximum	6	3:01	160,3	0	1:05	85,16	4	1:44	119	4	2:36	179,9
Minimal	0	0:55	67,13	0	0:48	64,74	0	0:48	61,80	0	0:52	65,17
Deviation	1,70	0:30	24,44	0,00	0:05	5,93	0,91	0:17	18,74	0,99	0:26	29,51
Average	0,60	1:18	83,51	0,00	0:54	70,04	0,25	1:06	78,38	0,35	1:15	88,08
	POS 2 ABC			POS 2 AB			POS 2 AC			POS 2 BC		
	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis
Maximum	2	1:55	129,8	0	0:40	49,79	0	2:04	127,1	4	1:59	125,3
Minimal	0	0:39	52,39	0	0:33	46,77	0	0:42	53,31	0	0:44	53,89
Deviation	0,45	0:20	18,93	0,00	0:02	0,95	0,00	0:25	21,74	0,91	0:19	19,12
Average	0,10	0:59	65,57	0,00	0:36	47,85	0,00	1:04	69,50	0,25	1:02	71,78
	POS 3 ABC			POS 3 AB			POS 3 AC			POS 3 BC		
	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis	Col	Time	Dis
Maximum	1	1:04	60,52	0	1:09	81,24	2	1:14	72,57	0	0:38	45,52
Minimal	0	0:32	43,50	0	0:34	43,46	0	0:31	42,04	0	0:31	41,30
Deviation	0,22	0:08	4,73	0,00	0:08	8,12	0,49	0:11	7,88	0,00	0:02	1,33
Average	0,05	0:42	47,28	0,00	0:39	48,74	0,15	0:39	47,66	0,00	0:33	42,97

Figure 23. Results of an RTCS in a real robot.

(a similar duration). These results of the ROCS demonstrate that learned rules are useful for the navigation of a real robot with a stable and fixed functioning, so, the behaviour of the ROCS on the real robot demonstrates that the degree of learning of the CS is sufficient to carry out the imposed task.

7. Conclusions

The proposed RCS has been developed to learn reactions (decision is a function of the environmental information) and actions (decision is a function of the environmental information and previous internal information). This modified CS (namely an RCS) has proven its ability to learn autonomous robot behaviours in dynamic environments.

The fusion of each environmental message with information from previous fired rules (the GLTF mechanism) and the inclusion of internal conditions (the IT mechanism) allow the generation of a sequence of actions, defined by a chain of rules over different inputs. Sets of cooperative rules emerge from the evolution of the RCS. Co-operation is viewed in this case as a chain of rules, where a rule is only meaningful if it matches the environment and follows another specific rule in time.

The inclusion of a message without other information but the environmental input allows the evolution of reactions.

Experiments carried out without generating new rules proved the capabilities of our approach to discriminate accurately between rules in a system that can provide a chain of rules at the same time that it receives new inputs.

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

Another important aspect verified in this work is the possibility of continuously learning and adaptation to new situations that allow the problem to be solved even if there are mobile objects, more than one goal, and dynamic goals that could appear and disappear or move when the robot is navigating.

References

BROOKER, L., 1982, Improving behavior as an adaptation to the task environment. Doctoral Dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

BROOKER, L., GOLDBERG D., and HOLLAND, J., 1989, Classifier systems and genetic algorithms. *Artificial Intelligence*, **40**, 235–282.

BROOKS, R. A., 1991, Intelligence without representation. *Artificial Intelligence*, **47**, 139–159.

COLOMBETTI, M., and DORIGO, M., 1993, Training agents to perform sequential behaviour. Technical Report 93-023, Politechnico di Milano, Italy.

DORIGO, M., 1995, ALECSYS and the AutonoMouse: learning to control a real robot by distributed classifier systems. *Machine Learning*, **19**, 209–240.

DORIGO, M., and SIRTORI, E., 1991, Alecsys: a parallel laboratory for learning classifier systems. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 296–302.

GOLDBERG, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning* (New York: Addison-Wesley).

GREFENSTETTE, J. J., 1988, Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, **3**, 225–245.

HOLLAND, J., 1975, *Adaptation in Natural and Artificial Systems* (Ann Arbor, Michigan: University of Michigan Press); 1986a, Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. edited by R. Michasky, J. Carbonell and T. Mitchell, *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, pp. 523–622 (Los Altos, California: Morgan Kaufman); 1986b, Properties of the bucket brigade. *Proceedings of an International Conference on Genetic Algorithms and their Applications*, edited by J. J. Grefenstett; 1995, *Hidden Order: How Adaptation Builds Complexity* (Reading, Massachusetts: Addison-Wesley).

HOLLAND, J., and REITMAN, J. S., 1978, Cognitive systems based on adaptative algorithms. edited by D. A. Waterman and F. Hayes-Roth, *Pattern-Directed Inference Systems* (New York: Academic Press).

ISASI, P., BERLANGA, A., MOLINA, J. M., and SANCHIS, A., 1997, Robot controller against environment, a competitive evolution. *Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Special Session on Evolution Computation*, Germany, 1997.

LEE, M. A., and TAKAGI, H., 1993, Integrating design stages of fuzzy systems using genetic algorithms. *Proceedings of the Second International Conference on Fuzzy Systems*, pp. 612–617.

MATELLÁN, V., FERNÁNDEZ, C., and MOLINA, J. M., 1998, Genetic learning of fuzzy reactive controllers. *Robotics and Autonomous Systems*, **25**, 33–41.

MCKERROW, P. J., 1991, *Introduction to Robotics*, (Reading, Massachusetts: Addison-Wesley).

MOLINA, J. M., SANCHIS, A., BERLANGA, A., and ISASI, P., 1997, Evolving connection weight between sensors and actuators in robots. *Proceedings of the IEEE International Symposium on Industrial Electronics*, pp. 686–690, Gimaraes, Portugal, 1997 (New York: IEEE).

MONDADA, F., and FRANZI, P. I., 1993, Mobile robot miniaturization: a tool for investigation in control algorithms. *Proceedings of the Second International Conference on Fuzzy Systems*, pp. 501–513, San Francisco, California, USA, 1993.

SANCHIS, A., MOLINA, J. M., and ISASI, P., 1996a, Classifier systems for learning reactions in robotics systems. *Proceedings of the First International Workshop on Machine Learning, Forecasting and Optimization* Leganés, Spain, 1996, pp. 153–159; 1996b, Learning reactive behaviour for autonomous robots using classifier systems. *Proceedings of the International Workshop on Spatio-Temporal Models in Biological and Artificial Systems*, pp. 152–159, Sintra, Portugal, 1996.

SCHULTZ, A., 1991, Using a genetic algorithm to learn strategies for collision avoidance and local navigation. *Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology*, pp. 213–215.

SCHULTZ, A., and GREFENSTETTE, J. J., 1990, Improving tactical plans with genetic algorithms. *Proceedings of the IEEE Conference on Tools for Artificial Intelligence* (New York: IEEE), pp. 328–334.

SOMMARUGA, L., MERINO, I., MATELLÁN, V., and MOLINA, J., 1996, A distributed simulator for intelligent autonomous robots. *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, pp. 393–399, Lisboa, Portugal, 1996.

WEIβ, G., 1994, Hierarchical chunking in classifier systems. *Proceedings of the 12th. International Conference on Artificial Intelligence*, pp. 1335–1340.

WESTERDALE, T. H., 1997, Classifier systems: no wonder they don't work. *Proceedings of the Second Annual Conference on Genetic Programming*, pp. 529–537.

WILSON, S.W., 1985, Knowledge growth in artificial animal. *Proceedings of an International Conference on Genetic Algorithms and their Applications*; 1987, Classifier systems and the animat problem. *Machine Learning*, **2**, 199–228.