Evolutionary learning of Quantified Fuzzy Rules for hierarchical grouping of laser sensor data in intelligent control

M. Mucientes, I. Rodríguez-Fdez, A. Bugarín

Intelligent Systems Group
Department of Electronics and Computer Science, University of Santiago de Compostela
Email: manuel.mucientes@usc.es, ismael.rodriguez1@rai.usc.es, alberto.bugarin.diz@usc.es

Abstract— *In complex systems it often occurs that relevant information about the system state and behavior is provided by groups of low-level variables rather than single variables. This grouping into high-level variables introduces a hierachy in the knowledge that can only be captured by means of rules involving propositions with a representation capability that is more complex than usual ones. In this paper we describe a genetic programming based approach for automated learning of Quantified Fuzzy Rules that are capable to deal with such representation capability. An application of this approach for hierarchical grouping of the distance measures provided by the laser sensors of a mobile robot (for the wall-following behaviour) is presented. Experimentation results show the control action is acceptable although no prior knowledge on the variables definition and structure was introduced in the controller.*

Keywords— Evolutionary algorithms, Genetic Programming, Fuzzy Quantification, Intelligent control.

1 Introduction

Modeling of systems using rules overcomes the interpretability issue and is a more adequate approach for scenarios when the model has to cope with noisy data or uncertainty. In this case fuzzy rules are a good choice.

Evolutionary algorithms have some characteristics that make them specially adequate for learning fuzzy rules, such as the flexibility in the representation of solutions through chromosomes, that may range from codifying a complete knowledge base, a single rule, some parameters of the fuzzy sets, ... The well-known combination of fuzzy logic with evolutionary algorithms (genetic fuzzy systems [1]) allows the designer to determine the most appropriate trade-off between accuracy and interpretability for a given system.

However, there are systems that can only be correctly modeled with rule bases that contain rules with different structures, and many times these structures are unknown. In order to learn such knowledge bases, the algorithm must have the ability to represent a set of structures. This ability is provided by genetic programming, where each chromosome of the population is represented as a tree of variable length. A complete flexibility in the structure of the rules is, generally, not desired and some restrictions have to be imposed, as not all the structures are valid. A compact representation of the valid structures of rules can be defined through a context-free grammar.

In systems with a huge number of (low level) input variables it is frequent that the values of the individual variables are not meaningful. In these cases relevant information about the system is only obtained by analyzing sets of low level variables, grouped into high level variables. This provides a hierarchical structure of knowledge that is meaningful for the experts, where the meaning of high level variables cannot be extracted taking neither an average value of the low level variables of the set, nor the maximum or minimum values. Usually this meaning can be directly captured using Quantified Fuzzy Propositions (QFPs) like *”most of the variables in the set take a high value”*.

In this paper we propose a genetic based approach to learn Quantified Fuzzy Rules (QFRs), defined as fuzzy rules of variable structure involving QFPs. The algorithm is based on the genetic programming approach and has been designed to solve regression problems in which the number of input variables is really high, and the model of the system needs the grouping of these variables in several sets (high level variables). In order to illustrate the utility of the algorithm, an application of use for hierarchical grouping of the distance measures provided by the laser sensors in a mobile robot is presented.

The paper is structured as follows: Sec. 2 introduces the motivation of the use of Quantified Fuzzy Rules (QFRs) with variable structure. Sec. 3 presents the QFRs model, while Sec. 4 describes the genetic programming algorithm that has been used to learn the QFRs. Finally, Sec. 5 points out some results, conclusions and future improvements of the algorithm.

2 Motivation for QFRs

Modeling the behavior of a mobile robot that performs a task, autonomously or remotely controlled by a human operator, is a challenging problem if no prior knowledge is provided. In previous papers [2] two successful approaches to learn fuzzy rules for the control of a robot in two different tasks were described. In both cases, the learned rules were conventional fuzzy rules and all the input variables were defined by a human expert. For the wall following task these variables were *right distance*, *left distance*, *orientation*, and *velocity*. For the moving object tracking task the variables were *distance to the object*, *deviation*, *difference in velocity with the object* and *difference in angle*.

The question that comes up is: is it possible to model the behavior of the robot without using any prior knowledge? This means that learning relies only on the input-output data pairs, i.e. on the sensors information and on the control orders that have been given for those situations. Fig. 1 shows how a robot equipped with a laser range finder senses the environment.

Laser range finders emit at the same time beams in different directions. When a beam hits an obstacle, it is reflected and registered by the scanner’s receiver. With this information, the distance measured in the direction of each beam can be calculated. Fig. 1 shows in the shadowed area the space covered by the laser when the robot is placed in an typical office environment. The laser range finder provides the distances to the closest obstacle in each direction with a given angular resolution (number of degrees between two consecutive beams).

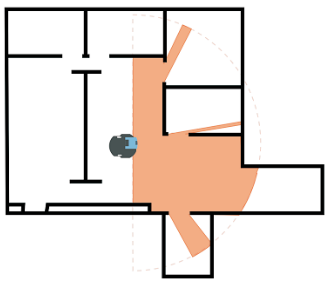


Figure 1: Sensors information for a mobile robot equipped with a laser range finder.

A robot equipped with a single laser range finder provides 361 distance measurements in each acquisition for an angular resolution of 0.5 degrees. When equipped with two laser range finders (to get information of the whole surrounding environment) it provides 722 distance input variables. As the output variables the controller must provide the linear and angular velocities. This control action cannot be decided by simply analyzing the individual distance values of each beam, since noisy measurements, gaps between objects (very frequent in cluttered environments) may occur. Usually the relevant information of the environment is modeled (or extracted) with propositions involving a more complex structure in the variables. For instance, the input variables can be arranged taking into account the importance of the information they provide to model the system.

In general, two categories can be established:

- High-level input variables: variables that provide, by themselves, information relevant and meaningful to the expert for modelling the system (e.g. the linear velocity of the robot).
- Low-level input variables: variables for which their individual values do not contribute to model the system. Their importance stems from the analysis of sets of these variables (e.g. the distance measured by a single beam in a laser range finder).

However, the values of a group of low level variables can provide valuable and meaningful information. For example, the "‘frontal sector"’ of a laser range finder is a high-level variable made up of a set of distances of single beams (low-level variables). Within this context Quantified Fuzzy Propositions (QFPs) such as *"some of the distances of the frontal sector are low"* are useful for representing relevant knowledge for the experts and therefore for performing intelligent control. QFPs require the definition of several elements:

- *Some*: how many distances of the frontal sector must be low?
- *frontal sector*: which beams belong to the frontal sector?
- *low*: what is the meaning of low?

This example clearly sets out the need to use propositions different from the conventional ones. In this paper QFPs (as "X is A in Q of S") are used for representing knowledge about high-level variables that are defined as the grouping of low-level variables. Conventional ("X is A") are used for other high-level variables, non related to low-level ones.

3 Quantified Fuzzy Rules (QFRs) model

An example of QFR is shown in Fig. 2, and involves both QFPs 1 and conventional ones 2 :

$$\text{IF } d(h) \textit{ is HIGH in most of } F^1_{sector} \text{ and} \qquad (1)$$
$$\ldots$$
$$\textit{velocity is } F_{vel} \qquad (2)$$
$$\text{THEN } \textit{linear velocity is } F_{lv} \text{ and } \textit{angular velocity is } F_{av}$$

Figure 2: A typical QFR to model the behavior of a mobile robot.

The general expression for QTPs in our case is:

$$d(h) \textit{ is } F^i_d \textit{ in } Q^i \textit{ of } F^i_{sector} \qquad (3)$$

where, for each i=1,...,N_d (N_d being the number of analyzed sectors of distances:

- $d(h)$ is the signal. In this particular case, it represents the distance measured by beam h.
- F^i_d is a linguistic value for variable $d(h)$.
- Q^i is a (spatial, defined in the laser beam domain) fuzzy quantifier.
- F^i_{sector} is a fuzzy set in the laser beam domain (e.g., the "frontal sector").

Evaluation of the Degree of Fulfillment (DOF) for 3 is carried out using Zadeh's quantification model for proportional quantifiers (such as "most of", "part of", ...) [3], that allows to consider non-persistence, partial persistence and total persistence situations for event $d(h)$ is F^i_d in the range of laser beams (spatial interval F^i_{sector}). This is a relevant characteristic of this model, since it allows to consider partial, single or total fulfillment of an event within the laser beams set.

Automatic learning of QFRs for this application (Fig. 2) demands an algorithm with the ability to represent rules with different structures, as the number N_d of analyzed sectors of distances, and therefore the number of QFPs per rule (Fig. 2) can change among rules. Moreover, the number of parameters involved in the definition of a proposition is higher in QFP than in conventional propositions (3 vs. 1).

4 Genetic programming algorithm

The proposed evolutionary algorithm is a genetic programming algorithm based on the genetic cooperative-competitive learning (GCCL) approach. In genetic programming, an individual is a tree of variable length. Each individual in the population can have a different structure, and the introduction of restrictions in that structure of the chromosome can be solved using, for example, a context-free grammar. As the number of beams sectors is unknown and can change among rules, the

ISBN: 978-989-95079-6-8

flexibility in the structure of the rules can be managed by the genetic programming algorithm.

In the GCCL approach [4, 5] rules evolve together but competing among them to obtain the highest fitness. In this approach it is fundamental to include a mechanism to maintain the diversity of the population (niche induction). The mechanism must warrant that there is competition among individuals of the same niche, but also has to avoid the deletion of those weak individuals that occupy a niche not covered by other individuals of the population.

We have chosen token competition [6] as the mechanism for maintaining the diversity. According to [7], this mechanism is adequate for genetic programming, as in this kind of evolutionary algorithms the structure of the individuals can be completely different and, thus, the evaluation of the similarities is hard. The advantage of token competition over other approaches, like crowding or fitness sharing, is that it is not necessary to estimate the similarities between pairs of individuals.

The learning process is based on a set of training examples. Each example e^l is represented by a tuple:

$$e^l = (d(1), \ldots, d(N_{LB}), \textit{velocity}, \textit{vlin}, \textit{vang}) \quad (4)$$

where $d(h)$ is the distance measured by beam h, *velocity* is the velocity of the robot, and *vlin* and *vang* are the output variables linear and angular velocity. In token competition, each example of the training set has a token and, of all the individuals that cover this example, the token will be seized by the individual with the highest raw fitness. In this way, the individual with the highest strength in the niche will exploit it, while individuals that are weaker will reduce its strength as they cannot compete with the best individual in the niche.

4.1 *Description of the context-free grammar*

As we pointed out before, in genetic programming each individual is a tree of variable size. Thus, the structure of the individuals can be very different among them. In order to generate valid individuals of the population, and to produce right structures for the individuals after crossover and mutation, some restrictions have to be applied. With a context-free grammar all the valid structures of a tree (chromosome) in the population can be defined in a compact form. A context-free grammar is a quadruple (V, Σ, P, S), where V is a finite set of variables, Σ is a finite set of terminal symbols, P is a finite set of rules or productions, and S is an element of V called the start variable.

The grammar is described in Fig. 3. The first item enumerates the variables, then the terminal symbols, in third place the start variable is defined, and finally the rules for each variable are enumerated. When a variable has more than one rule, rules are separated by symbol $|$. Fig. 4 represents a typical chromosome generated with this context-free grammar. Terminal symbols (leafs of the tree) are represented by circles, and variables are shown as flatted circles. Terminal symbols F_{vel}, F_d, F_{lv}, F_{av}, Q correspond with the linguistic labels and the quantifier defined in Fig. 2. Moreover, F_{sector} has been codified with *initialB* and *finalB*, which represent the extreme values of a trapezium.

A description of the evolutionary algorithm is shown in Fig. 5. It is based on the GCCL approach with a population of variable size. The following sections describe each of the stages of the algorithm.

- V = { rule, antecedent, consequent, distances }
- Σ = { F_{lv}, F_{av}, F_{vel}, F_d, *initialB*, *finalB*, Q }
- S = rule
- Productions:
  - rule $\longrightarrow$ antecedent consequent
  - antecedent $\longrightarrow$ distances F_{vel} | distances
  - consequent $\longrightarrow$ F_{lv} F_{av}
  - distances $\longrightarrow$ F_d *initialB* *finalB* Q distances | F_d *initialB* *finalB* Q

Figure 3: Context-free grammar.

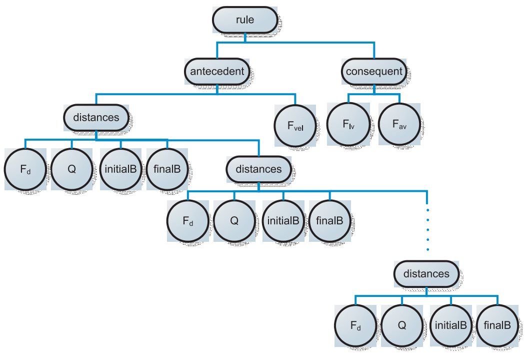


Figure 4: A chromosome representing a QFR to model the behavior of a robot.

1. Initialization
2. for iteration = 1 to *maxIterations*
   (a) Selection
   (b) Crossover and mutation
   (c) Evaluation
   (d) Resize population

Figure 5: Evolutionary algorithm.

4.2 *Initialization*

The first step of the algorithm is the initialization of the population: a chromosome (Fig. 4) is generated for each of the first $pop_{maxSize}$ examples in the training set. All the fuzzy sets of conventional fuzzy propositions (F_{vel}) and those of the consequent part (F_{lv} and F_{av}) are initialized with a triangular membership function centered in the corresponding example value, and with the extremes of the triangle at a distance equal to $prec_j$ from the center of the triangle. $prec_j$ represents the lowest meaningful change in the value of variable j.

The initialization of F_d, *initialB*, *finalB* and Q is quite more complex. Firstly, the number of QFPs that should be included in each rule is unknown and may be different for each example. Therefore, starting with the first beam, the average value

ISBN: 978-989-95079-6-8

of the distance is calculated incorporating consecutive beams until the variance of the distances is over a threshold. The process is repeated until all the beams have been asigned to a sector. Those sectors with a low number of beams are discarded. For each of the other sectors, *initialB* and *finalB* are directly defined using the information of the sector. F_d is generated in the same way as the fuzzy sets of the conventional propositions, but the triangular membership function is centered in the average value of the distances of the beams of the sector. Finally, Q is calculated as the percentage of beams of the sector that fulfill F_d.

4.3 Selection

The first stage of the iterative part of the algorithm is the selection of the individuals of the population. We have used a binary tournament selection. In a k-tournament selection, k individuals are randomly picked from the population with replacement, and the best of them is selected. In this case, $k = 2$ (binary tournament selection).

4.4 Evaluation

In order to estimate the fitness of an individual of the population, we first need to know the probability that an example e^i matches the output C_j:

$$P\left(C_j|e^i\right) = \exp\left(-\frac{error_j^i}{ME}\right) \tag{5}$$

where $error_j^i$ is the error due to the selection of an output different from the one codified in the example, and *ME* is a parameter that defines the meaningful error for the application. $error_j^i$ can be defined as:

$$error_j^i = \left(y_i^i - y_j^i\right)^2 \tag{6}$$

where y_i^i is the state of the system reached when the output of example e^i is applied to the state defined by e^i, and y_j^i is the state of the system reached when output C_j is applied to the state defined by e^i. As we are working with regression problems, an example can have several outputs that are different from the one codified in the example, but that make small errors, i.e., that are very similar to the desired output.

The accuracy of an individual of the population can be described as:

$$confidence = \frac{GC_{ex}}{covered_{ex}} \tag{7}$$

where GC_{ex} is the number of covered examples with $P\left(C_j|e^i\right)$ over a threshold P_{min}. The ability of generalization of a rule is calculated as:

$$support = \frac{GC_{ex}}{GC_{ex} + GU_{ex}} \tag{8}$$

where GU_{ex} is the number of uncovered examples with $P\left(C_j|e^i\right) > P_{min}$. Finally, we can define $fitness_{raw}$ as the combination of both values:

$$fitness_{raw} = \alpha_f \cdot confidence + (1 - \alpha_f) \cdot support \tag{9}$$

which represents the strength of an individual without taking into account the others. $\alpha_f \in [0,\ 1]$ is a parameter that codifies the trade-off between accuracy and generalization.

As we are using the GCCL approach, $fitness_{raw}$ will be used to decide which individual seizes each example. Thus, the fitness of an individual is defined as:

$$fitness = fitness_{raw} \cdot \frac{seized_{ex}}{covered_{ex}} \tag{10}$$

where $seized_{ex}$ is the number of examples seized by the individual, while $covered_{ex}$ is the number of examples that have been covered by it.

4.5 Crossover and mutation

Once pop_{size} individuals have been selected, each couple of them is crossed with probability p_c. The crossover operator is the parent-centric BLX (PCBLX) [8, 9]. This crossover operator is valid for genes representing real numbers. However, in the defined chromosomes (Fig. 4), there are also genes representing trapezoids. Given two trapezoids represented by tuples, $(\kappa_1\ \ldots\ \kappa_4)$ and $(\rho_1\ \ldots\ \rho_4)$ $(\kappa_j, \rho_j \in [a_j, b_j]$, $j = 1, \ldots, 4)$, that are going to be crossed, the following offspring are generated:

- $(\kappa'_1\ \ldots\ \kappa'_4)$, where κ'_j is randomly selected from the interval $\left[l_j^\kappa, r_j^\kappa\right]$, with $l_j^\kappa = \max\left\{a'_j, \kappa_j - I_j\right\}$, $r_j^\kappa = \min\left\{b'_j, \kappa_j + I_j\right\}$, $I_j = |\kappa_j - \rho_j| \cdot \alpha$, $\alpha \in [0, 1]$, $a'_j = a_j$ if $j \in \{1,\ 2\}$ or $a'_j = \kappa_2$ if $j \in \{3,\ 4\}$, and $b'_j = \kappa_3$ if $j \in \{1,\ 2\}$ or $b'_j = b_j$ if $j \in \{3,\ 4\}$.
- $(\rho'_1\ \ldots\ \rho'_4)$, where ρ'_j is randomly selected from the interval $\left[l_j^\rho, r_j^\rho\right]$, with $l_j^\rho = \max\left\{a'_j, y_j - I_j\right\}$ and $r_j^\rho = \min\left\{b'_j, y_j + I_j\right\}$. Finally, $a'_j = a_j$ if $j \in \{1,\ 2\}$ or $a'_j = \rho_2$ if $j \in \{3,\ 4\}$, and $b'_j = \rho_3$ if $j \in \{1,\ 2\}$ or $b'_j = b_j$ if $j \in \{3,\ 4\}$.

When two individuals are crossed, the PCBLX operator is applied to all the genes of type terminal symbol in the chromosomes. However, as the chromosomes are trees of variable structure and/or size, for a gene of the first individual, which gene should be selected in the second of the individuals? For gene F_{vel} the choice is simple: if this gene exists in both individuals crossover is performed, otherwise not. Nevertheless, for genes of type *distances* (as there are many possible choices), the gene selected in the second individual is the one that has a higher overlap with the gene of the first individual in the definition of the sector. If there is not overlap, the gene is not crossed. For genes of type *distances*, PCBLX is applied to F_d, *initialB*, *finalB*, and Q with their corresponding counterparts in the other chromosome.

After crossover, it could happen that the nodes of type *distances* are unsorted, or that two nodes overlap (they should be merged). The algorithm to repair the chromosome has the following steps:

1. Sort all the sectors taking into account *initialB* and insert them on a list.
2. While the list is not empty.
   (a) Choose the first element of the list and check if it overlaps with other sectors.
      i. If there is overlap, then merge the sectors and sort the list.

ii. If there is not overlap, delete the sector from the list and add it to the new tree.

When crossover is not performed, both individuals are mutated. Mutation can be implemented to generate a more general or a more specific rule. The higher the value of $\mathit{seized}_{ex}/\mathit{covered}_{ex}$, the higher the probability to generalize the rule by mutation. This occurs with rules that are very strong in their niche and that could be modified to seize examples in other niches. On the contrary, when the number of seized examples in very low, this means that the rule is weak in its niche, and in order to improve its performance some examples that are currently covered should be discarded.

In order to select the examples that are going to be covered (generalization) or uncovered (specialization), the value of $P\left(C_j|e^i\right)$ (Eq. 5) for each example is considered, where C_j is the consequent of the rule. For generalization, the best C_{ex} uncovered examples are selected while, for specialization, the worst C_{ex} covered ones are chosen. Then, from the selected set of examples, S_{ex} are randomly selected to modify the rule. The selection probability of each example is directly (generalization) or inversely (specialization) proportional to $P\left(C_j|e^i\right)$. For each selected example, each proposition of the antecedent part of the rule is analyzed to decide its mutation. For generalization, if the DOF of the proposition is null, then the proposition is mutated. The opposite occurs for specialization. The mutation of a proposition for generalization can be done with the following mutation operators:

1. Delete the proposition.

2. Merge two adjacent distance sectors. The new F_d value will be obtained applying PCBLX operator to the F_d values of the old sectors. This mutation operator can only be applied for propositions of type distances.

3. Modify the fuzzy label to cover the new example.

4. Modify the quantifier to cover the new example (only for distance propositions).

On the other hand, the mutation operators for specialization are:

1. Insert a *distances* sector (only valid for distance propositions). The insertion position is randomly selected. The values for F_d and Q are generated mutating the values of the previous or next sector. Also, *initialB* and *finalB* are randomly selected using the definitions of the contiguous sectors.

2. Divide a distances sector (only valid for distance propositions). One of the new sectors will have the same F_d and Q values, while the other sector will be generated applying non-uniform mutation to the original values.

3. Modify the fuzzy label to uncover the selected example.

4. Modify the quantifier to uncover the selected example (only for distance propositions).

4.6 *Resize population*

Those individuals with null fitness are removed from the population, and the best $pop_{maxSize}$ individuals are selected for the final population. If $pop_{size} < pop_{maxSize}$, and there are still uncovered examples, then new individuals are added. These individuals are chosen from the *examples population*, randomly selecting those rules that cover examples that have not been seized yet by the individuals of the population.

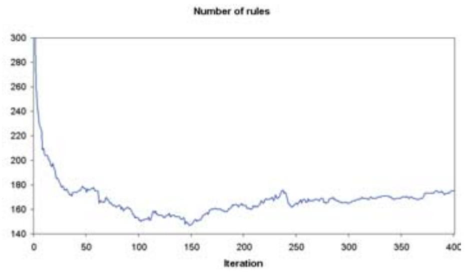
4.7 *Population evaluation*

The performance of the knowledge base obtained after each iteration has been measured as:

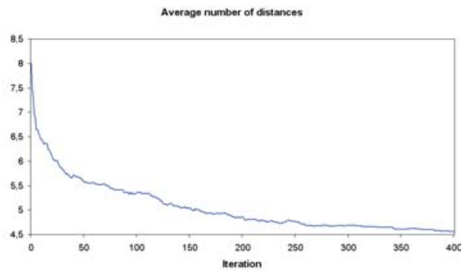
$$\mathit{fitness}_{pop} = \frac{n_{ex}}{\sum_l error_{pop^l}} \tag{11}$$

where n_{ex} is the total number of examples, and $error_{pop^l}$ is the error of the knowledge base when the input is example l.

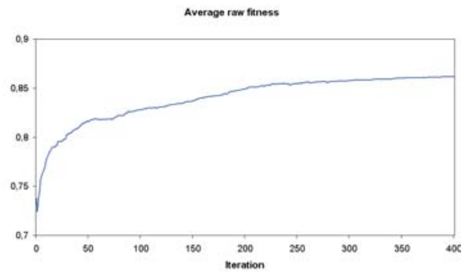
5 Results and discussion



(a) Number of rules.



(b) Average number of distances.



(c) Average $\mathit{fitness}_{raw}$.

Figure 6: Evolution of several variables for an execution of the algorithm.

The learning algorithm has been tested with a set of 795 examples recorded when the robot was following the wall in an environment. The values that have been used for the parameters of the evolutionary algorithm are: $\mathit{maxIterations} = 400$,

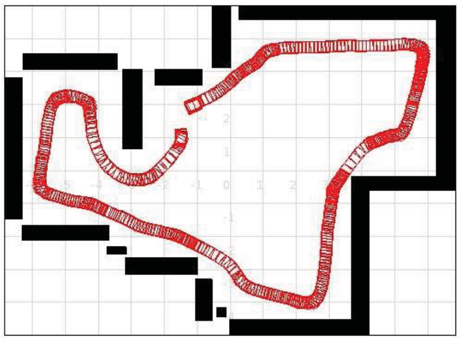


Figure 7: Path of the robot along the test environment.

$pop_{maxSize} = 300$, $p_c = 0.5$, $\alpha = 0.5$ (PCBLX crossover), and $P_{min} = 0.4$.

Fig. 6 shows the evolution of several variables for an execution of the genetic programming algorithm. These variables are the number of rules in the population after each iteration, the average (over all the individuals of the population) number of distance variables in the antecedent part of the rules, and the average $fitness_{raw}$. As can be seen from Fig. 6(a), until iteration 150 the number of rules decreases to less than 150. As the rules in the initial population where very specific (covering sometimes only one example) the algorithm finds rules with a good value of the confidence and with a higher value of the support. In the second stage, as there are rules with a high support, new rules with a higher confidence (and a lower support) seize part of the examples of the weaker rules, thus generating a higher number of rules. At the last iteration of the algorithm, the population had 175 rules. The improvement in the quality of the rules of the population is reflected by the evolution of the average raw fitness (eq. 9) of the rules (Fig. 6(c)).

Finally, the evolution in the average number of distances in the antecedent part of the rules (Fig. 6(b)) reflects the ability of the algorithm to extract valuable information about the distance sectors that are meaningful to model the system. From an initial value of more than eight distance sectors (in average), the algorithm obtains rules that, in average, use around 4.5 distance variables.

Fig. 7 shows the path of the robot along the environment that was used to test the learned controller. In this environment, the robot was following the right-hand wall. The higher the concentration of marks of the robot, the lower the linear velocity. In order to evaluate the quality of the controller we have measured four different indicators: the right distance, the linear velocity, the change in the linear velocity between two consecutive cycles —which reflects the smoothness in the control—, and the time. The average values of the indicators are calculated for each lap that the robot performs in the environment. Results presented in table 1 are the average and standard deviation values over five laps of the average values of the indicators over one lap.

Results show a distance over the expected one (0.5). However, this is in part because the distance has been measured with the information coming from only one beam (in [2, 10] the distance was measured using the input variable defined by an expert). Moreover, the velocity is low. Nevertheless, the smoothness of the control is very good.

Table 1: Results ($x \pm \sigma$) for the environment in Fig. 7

Distance (m)	0.8137 ± 0.0091
Velocity (m/s)	0.13274 ± 0.00095
Velocity change (m/s)	0.00486 ± 0.00053
Time (s)	259.26 ± 2.20

These preliminary results are promising and show an acceptable control, although no expert knowledge was introduced for the definition of the variables. Moreover, as the system has been trained with data of a unique environment, further analysis on the ability of the algorithm to generalize the extracted model of the behavior of the robot will be conducted in the future.

Acknowledgment

This work was supported by the Spanish Ministry of Science and Innovation under grant TIN2008-00040. M. Mucientes is supported by the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

References

[1] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases*, volume 19 of *Advances in Fuzzy Systems - Applications and Theory*. World Scientific, 2001.

[2] M. Mucientes and J. Casillas. Quick design of behaviors in mobile robotics by fuzzy controllers with good interpretability. *IEEE Transactions on Fuzzy Systems*, 15(4):636–651, 2007.

[3] L.A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics with Applications*, 9:149–184, 1983.

[4] A. Giornada and F. Neri. Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416, 1995.

[5] D.P. Greene and S.F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 3:229–257, 1993.

[6] K.S. Leung, Y. Leung, L. So, and K.F. Yam. Rule learning in expert systems using genetic algorithm: 1, concepts. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks*, pages 201–204, Iizuka (Japan), 1992.

[7] F.J. Berlanga, M.J. del Jesus, and F. Herrera. Learning compact fuzzy rule-based classification systems with genetic programming. In *Proceedings of the 4th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)*, pages 1027–1032, Barcelona (Spain), 2005.

[8] F. Herrera, M. Lozano, and A.M. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18:309–338, 2003.

[9] M. Lozano, F. Herrera, N. Krasnogor, and D. Molina. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary Computation*, 12:273–302, 2004.

[10] M. Mucientes, R. Alcalá, J. Alcalá-Fdez, and J. Casillas. Learning weighted linguistic rules to control an autonomous robot. *International Journal of Intelligent Systems*, 24:226–251, 2009.