Soft Comput (2006) 10: 881–889
DOI 10.1007/s00500-005-0014-x

FOCUS

M. Mucientes · D. L. Moreno · A. Bugarín · S. Barro

Evolutionary learning of a fuzzy controller for wall-following behavior in mobile robotics

Published online: 28 September 2005

Abstract The design of fuzzy controllers for the implementation of behaviors in mobile robotics is a complex and highly time-consuming task. The use of machine learning techniques such as evolutionary algorithms or artificial neural networks for the learning of these controllers allows to automate the design process. In this paper, the automated design of a fuzzy controller using genetic algorithms for the implementation of the wall-following behavior in a mobile robot is described. The algorithm is based on the iterative rule learning approach, and is characterized by three main points. First, learning has no restrictions neither in the number of membership functions, nor in their values. In the second place, the training set is composed of a set of examples uniformly distributed along the universe of discourse of the variables. This warrantees that the quality of the learned behavior does not depend on the environment, and also that the robot will be capable to face different situations. Finally, the trade off between the number of rules and the quality/accuracy of the controller can be adjusted selecting the value of a parameter. Once the knowledge base has been learned, a process for its reduction and tuning is applied, increasing the cooperation between rules and reducing its number.

Keywords Evolutionary algorithms · Fuzzy control · Mobile robotics · Wall-following behavior

1 Introduction

Fuzzy control has shown to be a very useful tool in the field of autonomous mobile robotics, characterized by a high uncertainty in the knowledge about the environment where a robot evolves.

The design of a fuzzy controller is generally made using expert knowledge about the task to be controlled. Expert knowledge is applied in order to decide the number of linguistic labels for each variable, to tune the membership functions, to select the most adequate linguistic values for the consequents, and to define the rules in the fuzzy knowledge base. This process is tedious and highly time-consuming. For this reason, automated learning techniques, such as evolutionary algorithms, have been employed for helping in some, or in all, of the tasks involved in the design process [4, 12, 19]. In some of the approaches evolutionary algorithms are used just for tuning the membership functions. In others, the complete rule base is learned, starting from a hand designed data base (number and definition of the linguistic values and universe of discourse of the variables). But only in a few of them both the data base and the rule base are learned.

In this paper, we describe the learning of a fuzzy controller for the wall-following behavior in a mobile robot. The learning methodology is characterized by three main points. First, learning has no restrictions neither in the number of membership functions, nor in their values. In the second place, the training set is composed of a set of examples uniformly distributed along the universe of discourse of the variables. This warrantees that the quality of the learned behavior does not depend on the environment, and also that the robot will be capable to face different situations. Finally, the trade off between the number of rules and the quality/accuracy of the controller can be adjusted selecting the value of a parameter. The methodology is based on the iterative rule learning (IRL) approach [6]. Once the knowledge base has been obtained, a new stage is developed in order to reduce and tune it.

The paper is organized as follows: in Sect. 2 some general comments about the evolutionary learning of knowledge bases are made. Section 3 describes the IRL methodology employed, whilst in Sect. 4 the process for the reduction and tuning of the knowledge base is explained. Section 5 describes the application of the proposed algorithm to the wall-following behavior, and Sect. 6 presents the obtained results. Finally, conclusions and future work are pointed out in Sect. 7.

M. Mucientes (✉)· D. L. Moreno · A. Bugarín · S. Barro
Dept. of Electronics and Computer Science,
University of Santiago de Compostela,
15782 Santiago de Compostela, Spain
E-mail: manuel@dec.usc.es; dave@dec.usc.es; alberto@dec.usc.es;
senen@dec.usc.es

2 Evolutionary learning of knowledge bases

Learning of knowledge bases using evolutionary algorithms has three main approaches: Michigan, Pittsburgh and IRL [7]. In the Michigan approach [13], each chromosome represents an individual rule, and the entire population is the rule base. Rules evolve along time due to their interaction with the environment. The major problem of this approach is that of resolving the conflict between the performance of individual rules and that of the rule base. The objective is to obtain a good rule base which means not only to obtain good individual rules, but also rules that cooperate between each other to get adequate outputs. This could be sometimes conflicting, for example when an individual rule that receives a high payoff is not adequately cooperating with other rules. This problem is addressed in [3].

This conflict was overcome by the Pittsburgh approach [5], where each chromosome represents a full knowledge base. Length of the chromosomes can be variable, which permits dealing with knowledge bases with a variable number of rules. This approach has a higher computational cost, because several knowledge bases have to be evaluated, while for the Michigan approach a single rule base is evaluated.

In the third approach (IRL [6]), each chromosome represents an individual rule, but contrary to the Michigan approach, a single rule is learned by the evolutionary algorithm and not the whole rule base. After each sequence of iterations, the best rule is selected and added to the final rule base. The selected rule must be penalized in order to induce niche formation in the search space. Niching is necessary for solving multimodal problems, as occurs with knowledge bases learning. In this case, each of the rules of the knowledge base is a solution (highly multimodal problem), and all the solutions must be taken into account to get the complete knowledge base. A common way to penalize the rules that have been obtained is to delete those training examples that have been covered by the set of rules that integrate the final rule base. The final step of the IRL approach is to check whether the obtained set of rules is a solution to the problem. In the case it is not, the process is repeated. A weak point of this approach is that the cooperation between rules is not taken into account when a rule is evaluated.

3 Learning of fuzzy-rule based controllers

Our proposal consists on a learning method based on the IRL approach in which both the data and rule bases are simultaneously learned. The only predefined parameters are the universe of discourse and the granularity of each variable. Both the number and shape of the membership functions, and the rules' structure (a variable could not be considered in a rule) will be learned. The algorithm has the following steps:

1. Obtain a rule for the system.
   (a) Initialize population.
   (b) Evaluate population.
   (c) Eliminate bad rules and fill up population.
   (d) Scale the fitness values.
   (e) While the maximum number of iterations is not exceeded.
      i. Select the individuals of the population.
      ii. Crossover and mutate the individuals.
      iii. Evaluate population.
      iv. Eliminate bad rules and fill up population.
      v. Scale the fitness values.
2. Add the best rule to the final rule set.
3. Penalize the selected rule.
4. If the knowledge base does not solve the problem, return to Step 1.

The rules that are going to be learned are conventional fuzzy rules like:

$$R^i: \text{If } X_1^i \text{ is } A_1^i \text{ AND } \cdots \text{ AND } X_{NA}^i \text{ is } A_{NA}^i \quad \text{Then } Y_1^i \text{ is } B_1^i \text{ AND } \cdots \text{ AND } Y_{NC}^i \text{ is } B_{NC}^i \tag{1}$$

where R^i, $i = 1, \ldots, NR$, is the i-th rule, X_j^i, $j = 1, \ldots, NA$, and Y_k^i, $k = 1, \ldots, NC$, are linguistic variables of the antecedent and consequent parts, respectively. NR is the number of rules, NA the number of antecedents in a rule, NC the number of consequents, and A_j^i and B_k^i are linguistic values (labels) of these variables.

A set of examples has been chosen for learning the knowledge base. These examples cover the universe of discourse of all the variables in the antecedent part of the rule. The universes of discourse have been discretised, in order to minimize the search space, with a step or granularity g_n, $n = 1, ..., NV$, where $NV = NA + NC$ is the number of variables. The function, SF, that scores the action of a rule over an example (not the fitness function) is application dependent.

An example, e^l, is covered by rule R^i if it complies with the following two conditions:

$$A_1^i\left(e_1^l\right) \wedge \cdots \wedge A_{NA}^i\left(e_{NA}^l\right) > 0 \tag{2}$$

where $A_j^i\left(e_j^l\right)$ represents the membership degree of the value of variable j in the example e^l to A_j^i. A new parameter, δ, is defined with the aim of selecting the relation between the number of rules and the quality and accuracy of the controller. In that way, a second condition is imposed:

$$\frac{SF\left(R^i\left(e^l\right)\right)}{\max\left(SF\left(e^l\right)\right)} > \delta \tag{3}$$

where $SF\left(R^i\left(e^l\right)\right)$ is the score assigned to the state reached by the system after applying rule R^i over example e^l, and $\max\left(SF\left(e^l\right)\right)$ is the maximum score that an action can obtain for example e^l. These values are obtained before the beginning of the algorithm, trying and scoring all the possible actions for each example. The value of parameter δ can be adjusted in a range between 0 and 1. A low value of δ

produces a lower number of rules in the final knowledge base, but the quality and the accuracy of the controller decreases. On the contrary, a high value of δ increases the quality of the controller, but also the number of rules.

The use of δ can be clarified by means of the following example. Let us suppose a robot must reach a point by turning 30°. Although this may be labeled as the best control action, also a rule proposing a turning of 20° should be considered as a good rule, even though the goal point is not fully reached. Parameter δ indicates the minimum quality a rule must have in order to be a valid rule for being added to the final knowledge base.

The shape of the membership functions that are going to be learned is shown in Fig. 1. Parameters b_n^i and c_n^i are learned, but points a_n^i and d_n^i are calculated as:

$$a_n^i = \max\left\{b_n^i - g_n,\ \text{lower}_n\right\} \tag{4}$$

$$d_n^i = \min\left\{c_n^i + g_n,\ \text{upper}_n\right\} \tag{5}$$

where lower_n and upper_n are the extreme points of the universe of discourse of variable n. In that way, it is not possible to exceed the range of the universe of discourse of the variables during the learning process.

Chromosomes are real coded, and a rule as the one shown in (1) is encoded into a chromosome C^i as:

$$C^i = \left(a_1^i,\ b_1^i,\ c_1^i,\ d_1^i,\ \ldots,\ a_{NV}^i,\ b_{NV}^i,\ c_{NV}^i,\ d_{NV}^i\right) \tag{6}$$

3.1 Steps of the genetic algorithm

The first step of the genetic algorithm consists on initializing the population. Rules in the initial population are created in the following way. An example currently not covered by any rule in the final knowledge base is randomly selected. The created rule is going to cover that single example, named e^l, at this initial generation. The membership functions for this rule are constructed as: $b_n^i = c_n^i = e_n^l$, whilst a_n^i and d_n^i are calculated using (4) and (5), respectively.

The evaluation of each individual of the population (each rule) is done with a two level fitness function. The first level (FF) consists on counting the number of examples that are covered by this rule, i.e., that fulfill (2) and (3), and that are not yet covered by a rule of the final knowledge base. If an example is covered by a rule of the final knowledge base, it will not contribute to the fitness value of any rule in the population. A second level for the fitness function is added in order to distinguish between rules with the same antecedent part but different consequents. It is the average value of the scoring function (SF), for all the examples that verify (2):

$$\text{ASF}^i = \frac{\sum_{l=1}^{NE} \text{SF}\left(R^i\left(e^l\right)\right)}{NEC^i} \tag{7}$$

where NE is the number of examples, and NEC^i is the number of examples that verify (2) for rule i. The second level of the fitness function is only used (in conjunction with the first level) when the final generation has been reached and the best rule of the population has to be added to the final knowledge base.

If for example a rule verifies (2) but not (3), then this rule is deleted from the population. After the deletion of all the bad rules, the population must be filled up, until its size reaches NR. The rules that will be added are the best rules in the population. Finally the fitness values of the individuals of the population must be linearly scaled in order to prevent premature convergence of the population.

The selection procedure that has been employed is the stochastic remainder without replacement. An individual i will be selected $int\left(\frac{FF^i}{AFF}\right)$ times, where FF^i is the value of the fitness function (first level) for individual i, and AFF is the average value of all FF^i. Taking into account $frac\left(\frac{FF^i}{AFF}\right)$ the population is randomly filled up. After selection, the individuals are crossed (one-point crossover), mutated, and finally added to the new population. Elitism has been applied to avoid the loss of the best individuals due to crossover and mutation.

Crossover is done taking into account that the combination of points a_n^i and b_n^i cannot be truncated, and the same occurs to points c_n^i and d_n^i. This means that the slope of the sides of a membership function cannot be modified. After crossover, each chromosome is reordered for repairing bad definitions of the membership functions (e.g., $b_n^i > c_n^i$).

The mutation operator has three equally probable options to operate on a gene. It will only modify genes of type b_n^i or c_n^i: increasing or decreasing the value of the gene in an amount of g_n, or leaving the gene unchanged. This will provoke the extension or contraction of the membership function in a quantity equal to the granularity of each variable, implementing a local search in that way. After mutation, each chromosome will be reordered, and values of a_n^i and d_n^i will be calculated using (4) and (5), respectively.

Once the maximum number of iterations has been reached, the best rule of the population is added to the final knowledge base, and all the examples covered by this rule are marked. In that way these examples will not contribute to the fitness value of the individuals in the next sequences of iterations. If all the examples are covered by the rules of the final knowledge base, then this is a solution to the problem and the algorithm ends.

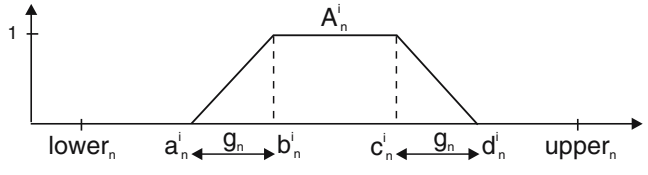

Fig. 1 Shape of the membership function, A_n^i

4 Reduction and tuning of the learned knowledge base

In this stage, starting from the knowledge base previously obtained, a genetic algorithm for the reduction and tuning of that knowledge base is applied. The aim is to decrease the number of rules (this will enhance the interpretability), trying to maintain the accuracy and quality of the controller. We have opted for applying tuning and reduction simultaneously. Thus, the chromosomes have two parts: $C = C_T C_R$. The first part of the chromosome, C_T, codifies the membership functions of all the rules of this knowledge base, in order to implement the tuning:

$$C_T = C_T^1, \ldots, C_T^{NR} \tag{8}$$

where NR is the number of rules in this knowledge base, and C_T^i, $i = 1, \ldots, NR$, codifies the membership functions of each rule i:

$$C_T^i = \left(a_1^i,\ b_1^i,\ c_1^i,\ d_1^i,\ \ldots,\ a_{NV}^i,\ b_{NV}^i,\ c_{NV}^i,\ d_{NV}^i\right) \tag{9}$$

where NV is the number of variables. The interval of performance of each of the parameters that define a membership function [10] is given by the following equations (Fig. 2):

$$a_n^i \in \left[al_n^i,\ ar_n^i\right] = \left[a_n^i - \frac{d_n^i - a_n^i}{2},\ a_n^i + \frac{b_n^i - a_n^i}{2}\right] \tag{10}$$

$$b_n^i \in \left[bl_n^i,\ br_n^i\right] = \left[b_n^i - \frac{b_n^i - a_n^i}{2},\ b_n^i + \frac{c_n^i - b_n^i}{2}\right] \tag{11}$$

$$c_n^i \in \left[cl_n^i,\ cr_n^i\right] = \left[c_n^i - \frac{c_n^i - b_n^i}{2},\ c_n^i + \frac{d_n^i - c_n^i}{2}\right] \tag{12}$$

$$d_n^i \in \left[dl_n^i,\ dr_n^i\right] = \left[d_n^i - \frac{d_n^i - c_n^i}{2},\ d_n^i + \frac{d_n^i - a_n^i}{2}\right] \tag{13}$$

The reduction process is based on the selection of some of the rules from the rule set. In that way, the second part of the chromosome, C_R, codifies if a rule belongs to the rule base (the allele is set to 1) or not (allele set to 0). The genetic algorithm tries to minimize a fitness function based on the mean square error (MSE), and the number of rules [8], in order to penalize those knowledge bases with a high number of rules:

$$F_{RT} = \omega_1 \cdot MSE + \omega_2 \cdot NR \tag{14}$$

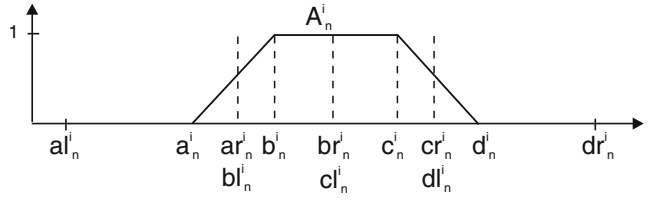

Fig. 2 Intervals of performance of each of the parameters that define the membership function of rule i and variable n

where ω_1 has been set to 1,

$$\omega_2 = \beta \cdot \frac{MSE_{irl}}{NR_{irl}} \tag{15}$$

with $\beta = 0.1$, and MSE_{irl} and NR_{irl}, being respectively the mean square error and the number of rules for the knowledge base obtained after the IRL algorithm. Finally, MSE is given by:

$$MSE = \frac{1}{2 \cdot NE} \sum_{l=1}^{NE} \left\{ \left\{ 1 - \frac{SF\left(R\left(e^l\right)\right)}{\max\left(SF\left(e^l\right)\right)} \right\} \cdot \omega_3 \right\}^2 \tag{16}$$

where $SF\left(R\left(e^l\right)\right)$ is the score assigned to the state reached by the system after applying the rules of the knowledge base over example e^l, and ω_3 is a scaling factor that has been set to 1000.

The population has been initialized setting all the genes in the C_R part to 1. For the C_T part, the first individual has been copied from the one obtained after the IRL algorithm, and the remaining individuals have been generated randomly, taking into account the corresponding intervals of performance. The genetic algorithm uses the Baker's stochastic universal sampling procedure [2] for the selection, together with elitism. The crossover operator will depend on the chromosome part where it is applied. Thus, for the C_T part, the max–min-arithmetical crossover [9] is considered. Then, if $C_v^t = (c_1, \ldots, c_k, \ldots, c_H)$ and $C_w^t = \left(c'_1, \ldots, c'_k, \ldots, c'_H\right)$ are going to be crossed, four sons will be generated:

1. $C_1^{t+1} = aC_v^t + (1-a)\, C_w^t$
2. $C_2^{t+1} = aC_w^t + (1-a)\, C_v^t$
3. C_3^{t+1} with $c_{3,k}^{t+1} = min\left\{c_k, c'_k\right\}$
4. C_4^{t+1} with $c_{4,k}^{t+1} = max\left\{c_k, c'_k\right\}$

Parameter a is selected by the designer, and in this application takes a value of 0.35. For the C_R part of the chromosome, the standard two-point crossover is used, obtaining two sons. This two sons are joined to the four sons generated from the C_T part, giving a total of eight sons. The two best combinations of the offspring will replace their parents.

The mutation operator also depends on the selected part of the chromosome. For the C_R part, the mutation operator simply changes the allele from 1 to 0 and vice versa. For the C_T part, a value in the interval of performance (Eqs. 10–13) is randomly selected.

5 Learning the wall-following behavior

The methodology presented in the previous sections is applied here for the design of a fuzzy controller for the wall-following behavior in a Nomad 200 mobile robot.

The wall-following behavior is usually implemented when the robot is exploring an unknown area, or when it is moving between two points in a map. A good wall-following controller is characterized by three features: to maintain a suitable distance from the wall that is being followed, to move at a high

velocity whenever possible, and finally to avoid sharp movements, making smooth and progressive turns and changes in velocity. The controller can be configured modifying the values of two parameters: the reference distance, which is the desired distance between the robot and the selected wall, and the maximum velocity attainable by the robot. In what follows we assume that the robot is going to follow a contour that is on its right side. Of course, the robot could also follow the left-hand wall, but this can be easily dealt with by simply interchanging the sensorial inputs.

In the bibliography, the wall-following behavior has been implemented with different techniques, the most used being fuzzy logic [1,15,20] and neural networks [11]. The advantage of using fuzzy logic lies in the interpretability of the fuzzy rules, as opposed to the weights of the neural networks. Also, some authors have merged both techniques, giving way to neuro-fuzzy controllers [18].

The input variables of the control system are the right-hand distance (RD), the distances quotient (DQ), which is calculated as:

$$DQ = \frac{\text{left-hand distance}}{RD} \tag{17}$$

As it can be seen (Fig. 3), DQ shows the relative position of the robot inside a corridor, which provides with information that is more relevant to the problem than simply using the left-hand distance. A high value for DQ means that the robot is closer to the right-hand wall, whilst a low value indicates that the closer wall is the left-hand one. The other input variables are the linear velocity of the robot (LV), and the orientation of the robot with respect to the wall it is following. A positive value of the orientation indicates that the robot is approaching to the wall, whilst a negative value means the robot is moving away from the wall. The output variables are the linear acceleration and the angular velocity.

All the information used to calculate distances and orientations comes from the ultrasound sensors of the robot. The distances and the orientation are obtained in two ways: if any of the walls (left or right) can be modeled with a straight line using a least square mean of the raw sensor data, then the corresponding distance and orientation are measured from that line. Otherwise, distance is measured as the minimum distance of a set of sensors, and the orientation will be the orientation of that sensor with respect to the advance direction.

The function SF used in (3) is defined for this application as:

$$SF\left(R^i\left(e^l\right)\right) = \frac{1}{\alpha_1 + \alpha_2 + \alpha_3 + 1} \tag{18}$$

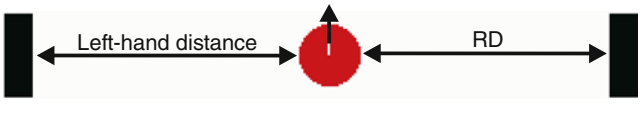

Fig. 3 Description of some of the distances used for the calculation of the input variables

where α_1, α_2, and α_3 are respectively:

$$\alpha_1 = 100\frac{|\text{RD-reference distance}|}{g_{RD}} \tag{19}$$

$$\alpha_2 = 10\frac{|\text{maximum velocity} - LV|}{g_{LV}} \tag{20}$$

$$\alpha_3 = \frac{|\text{orientation}|}{g_{orientation}} \tag{21}$$

and g_{RD}, g_{LV}, and $g_{orientation}$ are the granularities of the respective input variables. The granularities are used in these equations in order to evaluate the deviations of the values of the variables from the desired ones in a relative manner (the deviation of the value of variable n from the desired one is measured in units of g_n). This makes the comparison of the deviations of different variables possible and, as a consequence, the assignment of the weights for each one of the variables. These weights (100, 10 and 1 for (19), (20), and (21), respectively) have been heuristically determined, and indicate how much important the deviation in the value of a variable is with respect to the deviation of other variables.

SF takes values in [0, 1]. The highest weight has been assigned to the distance, as small variations of RD with respect to the reference distance should be highly penalized. An intermediate weight is associated to velocity and, finally, the least important contribution to function SF is for the orientation of the robot.

The defuzzification method that has been used for the learned fuzzy controller is the height defuzzifier [14]:

$$O_k = \frac{\sum_{i=1}^{NR} \overline{Y}^i_k \, B^i_k(\overline{Y}^i_k)}{\sum_{i=1}^{NR} B^i_k(\overline{Y}^i_k)} \tag{22}$$

where $\overline{Y}^i_k$ is the center of gravity of the membership function B^i_k, and O_k is the defuzzified value for variable k.

6 Results

The system has been implemented using the parameters described in Table 1.

For the IRL stage, these values have been selected in order to focus search in a few promising areas (low crossover probability), but exploiting those areas doing a local search due to the high mutation probability, in a similar way evolution strategies, for example (1+1)-ES, work. We noticed that a high crossover probability distracted the search due to the

Table 1 Values of some of the parameters for the two stages of the system

Parameter	IRL	R+T
Generations	50	460
Population size	300	61
Crossover probability	0.2	0.6
Mutation probability	0.4 (per gene)	0.2 (per chromosome)

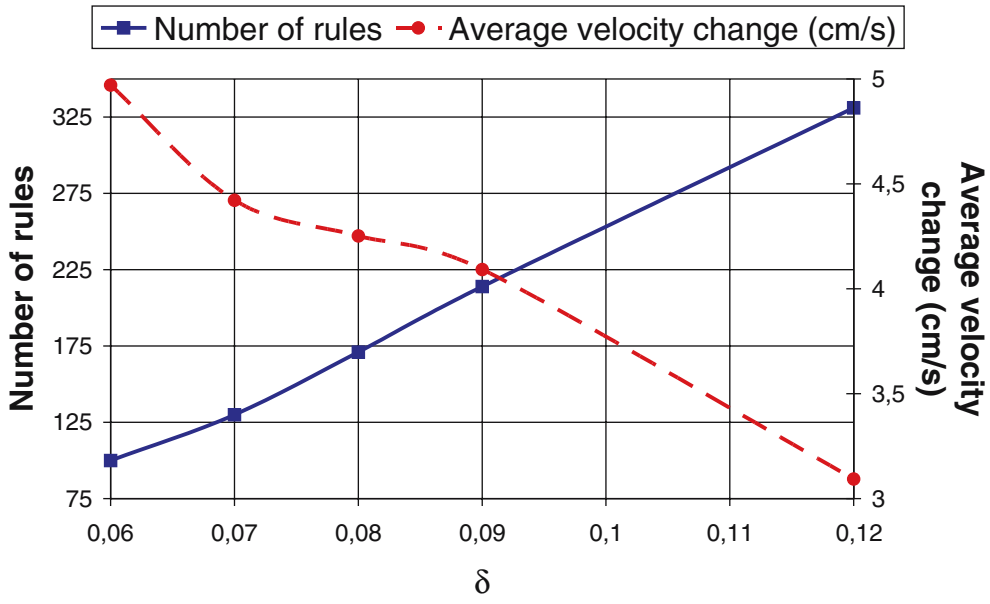

Fig. 4 Variation of the number of rules and the average velocity change of the final knowledge base, after iterative rule learning (IRL), with δ

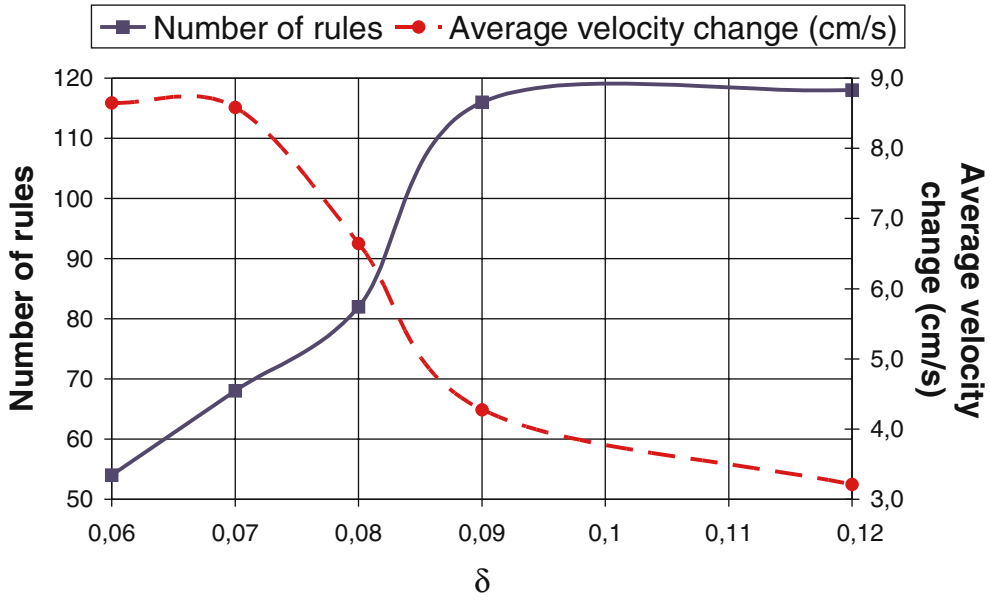

Fig. 5 Variation of the number of rules and the average velocity change of the final knowledge base, after the reduction and tuning stage (IRL-RT), with δ

combination of rules from quite different areas of the search space (this is due to the standard one-point crossover used), but a low crossover probability is still necessary to discover new promising areas. Different values for parameter δ (Eq. 3) have been tried. All the parameters of evolutionary learning have been heuristically obtained.

Figure 4 shows the variation of the number of rules and the average velocity change of the final knowledge base after the IRL algorithm with δ. As δ increases the number of rules rises. Values of $\delta < 0.06$ have been discarded since they did not produce valid controllers. The average velocity change of the robot was measured for the environment shown in Fig. 6(b). This variable evaluates the average change of the robot's velocity between two consecutive control iterations. Low values of the average velocity change indicate smooth and progressive changes in velocity, reflecting more accuracy and quality in the control actions. As can be seen in Fig. 4, as δ increases the average velocity change decreases, and consequently the accuracy and quality of the controller rises.

Once the first stage (IRL) has finished, the reduction and tuning process is implemented. Figure 5 shows the variation of the number of rules and the average velocity change of

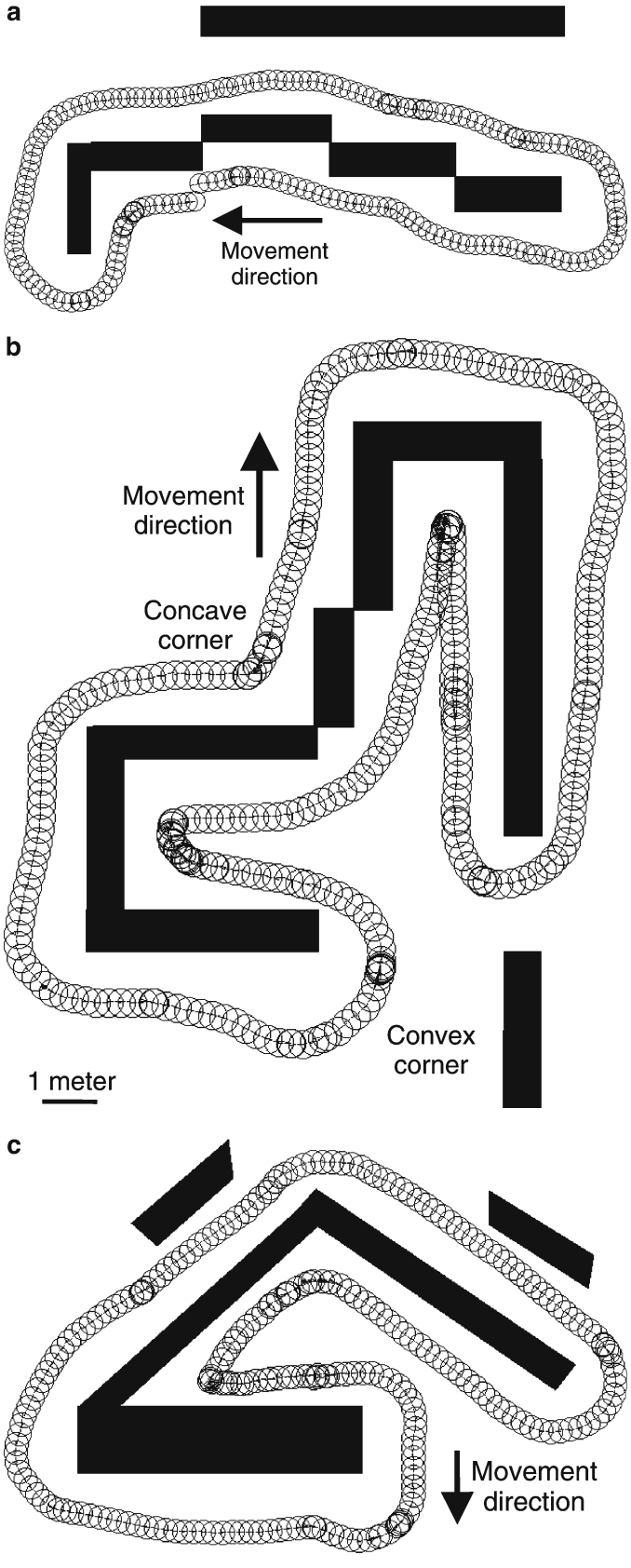

Fig. 6 Path of the robot in different simulated environments for $\delta = 0.06$ after the reduction and tuning stage

the final knowledge base after the reduction and tuning stage (IRL-RT) with δ. The main objective has been to reduce the number of rules, thus enhancing the interpretability of the knowledge base, while maintaining the quality and accuracy of the controller.

All the controllers that have been obtained were tested in three simulated environments using the Nomad 200 simulation software. The training set is composed of a list of examples, while the testing set is formed by three different environments constructed with the simulation software of the Nomad 200. These environments have not been used during the training of the controller. Learning only depends on function SF, which has to be carefully selected, and also on the examples, that must be chosen covering the input space with an adequate granularity (selection of the granularity of the variables is also of high importance). These conditions warrantee that the quality of the learned behavior does not depend on the environment, and also that the robot will be capable to face any situation.

Figure 6 shows as an example the robot path in three simulated environments for $\delta = 0.06$ after the reduction and tuning stage. The robot trajectory is represented by circular marks. A higher concentration of marks indicates lower velocity. The learned controller has 54 rules, the maximum velocity the robot can reach is 61 cm/s, and the reference distance at which the robot should follow the right wall is 51 cm. Ten tests have been done for each one of the environments. The average values measured for some parameters that reflect the controller performance are shown in Tables 2 (knowledge base after IRL – 100 rules – named IRL) and 3 (knowledge base after reduction and tuning – 54 rules – named IRL-RT).

Results show a very important reduction in the number of rules now needed to implement the behavior (from 100 to 54). The IRL-RT controller follows the wall in a more reliable way and, as a consequence, the average right distance is slightly lower. But due to this precision in the trajectory, values of average velocity and smoothness are not as good as for IRL.

Let's analyze in detail the path of the robot in environment [Fig. 6b]. This environment is quite complex, with three concave corners and seven convex corners in a circuit of a length of 54 m. Convex corners are truly difficult situations, because the robot's sensors may cease to correctly detect the wall at some given moments, even though some of them may occasionally detect it. The controller must also significantly reduce velocity at corners. In spite of these difficulties, the obtained average velocity has been quite high, and the distance at which the robot should follow the wall is near the desired reference distance. The difference between both

Table 2 Average values of some parameters for the environments of Fig. 6 for iterative rule learning (IRL)

Environment	RD (cm)	Velocity (cm/s)	Velocity change (cm/s)	Time (s)
6a	70	57	4.23	62
6b	65	51	4.59	106
6c	63	48	5.48	87

Table 3 Average values of some parameters for the environments of Fig. 6 for IRL-RT

Environment	RD (cm)	Velocity (cm/s)	Velocity change (cm/s)	Time (s)
6a	64	51	9.74	67
6b	65	45	8.65	114
6c	60	48	7.26	86

distances is caused by the high number of corners, in which the orientation of the robot is very bad (at concave corners the robot is detecting two perpendicular walls, and sometimes at convex corners it detects no wall), and a fast turning is prioritized over a correct distance.

Nevertheless, some aspects could be improved. Thus, comparing this controller with [15,16] (fuzzy temporal rule-based controller, hand-designed involving 313 rules), the obtained behavior provokes sharp changes in velocity. For this reason, one of the aspects that characterizes a good wall-following controller (smooth and progressive turns and changes in velocity) is not fulfilled. Also, the controller proposed in [15,16] gets a trajectory closer to the shape of the contour being followed. This is mainly due to two reasons: first, the higher number of rules of the temporal rule-based controller. But also the use of fuzzy temporal rules, which filter the sensorial noise and analyze the evolution of the values of the variables along temporal intervals, let the controller to smooth the behavior, anticipating future positions of the robot in the environment.

7 Conclusions and future work

An evolutionary process for the learning of fuzzy controllers has been described. The first stage is based on the IRL approach while, in the second stage, a reduction and tuning process is done. Learning has no restrictions neither in the number of linguistic values for each variable, nor in the values that define the membership functions. The process only depends on function SF, and on the selected granularities of the variables, which are application dependent and must be carefully chosen. The algorithm has been applied to the learning of the wall-following behavior.

A number of control systems (for different values of δ) have been learned. The final controller has been tested in three simulated environments with a high number of corners, showing a good performance both in the distance the wall was followed and in the average velocity. Due to the learning process, the quality of the learned behavior does not depend on the environment, and also the robot will be capable to face any situation.

As future work, we will try to learn Fuzzy Temporal Rule-based controllers [15–17], which have a high degree of expressiveness and the capacity of analyzing the evolution of variables, whilst taking past values into account.

Acknowledgements The authors wish to acknowledge the support from the Spanish Ministry of Education and Culture through grant TIC2000-0873. Part of this research has been developed during a research stay of the first author at the Department of Computer Science and Artificial Intelligence of the University of Granada (Soft Computing and Intelligent Information Systems Group) that was supported by the Dirección Xeral de I+D, Xunta de Galicia. They also wish to thank Dr. Francisco Herrera and Dr. Jorge Casillas for their kind collaboration and insightful comments on this research.

References

1. Arrúe BC, Cuesta F, Braunstingl R, Ollero A (1997) Fuzzy behaviours combination to control a non-holonomic robot using virtual perception memory. In: Proceedings of the 6th IEEE international conference on fuzzy systems (Fuzz-IEEE'97), Barcelona, Spain, pp 1239–1244
2. Baker JE (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the 2nd international conference on genetic algorithms, Hillsdale, NJ USA, pp 14–21
3. Bonarini A (1996) Evolutionary learning of fuzzy rules: competition and cooperation. In: Pedrycz W (ed) Fuzzy modelling: paradigms and practice, Kluwer Academic Press, Norwell, USA, pp 265–284
4. Braunstingl R, Mujika J, Uribe JP (1995) A wall following robot with a fuzzy logic controller optimized by a genetic algorithm. In: Proceedings of the international joint conference of the fourth ieee international conference on fuzzy systems and the second international fuzzy engineering symposium, vol 5, Yokohama, Japan, pp 77–82
5. Carse B, Fogarty TC, Munro A (1996) Evolving fuzzy rule based controllers using genetic algorithms. Fuzzy Sets Syst 80:273–293
6. Cordón O, Herrera F (2001) Hybridizing genetic algorithms with sharing scheme and evolution strategies for designing approximate fuzzy rule-based systems. Fuzzy Sets Syst 118:235–255
7. Cordón O, Herrera F, Hoffmann F, Magdalena L (2001) Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases, vol 19. World Scientific
8. Cordón O, Herrera F, Villar P (2001) Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. IEEE Trans Fuzzy Syst 9(4):667–674
9. Herrera F, Lozano M, Verdegay JL (1997) Fuzzy connectives based crossover operators to model genetic algorithms population diversity. Fuzzy sets Syst 92:21–30
10. Herrera F, Lozano M, Verdegay JL (1998) A learning process for fuzzy control rules using genetic algorithms. Fuzzy sets Syst 100:143–158
11. Iglesias R, Regueiro CV, Correa J, Barro S (1998) Supervised reinforcement learning: application to a wall following behaviour in a mobile robot. In: Pasqual del Pobil A, Mira J, Ali M (eds) Tasks and methods in applied artificial intelligence (IEA-98-AIE), vol 2 of Lecture Notes in Computer Science, Benicassim (Spain), pp 300–309
12. Leitch D (1996) In: Herrera F, Verdegay JL (eds) Genetic algorithms and soft computing, vol 8 Studies in fuzziness and soft computing, chapter Genetic algorithms for the evolution of behaviours in robotics, Physica-Verlag, pp 306–328
13. Magdalena L, Velasco JR (1996) In: Herrera F, Verdegay JL (eds) Genetic algorithms and soft computing. Studies in fuzziness, vol 8 Fuzzy rule-based controllers that learn by evolving their knowledge base, Physica-Verlag, pp 172–201
14. Mendel JM (1995) Fuzzy logic systems for engineering: a tutorial. Proc IEEE 83(3):345–377
15. Mucientes M, Iglesias R, Regueiro CV, Bugarín A, Barro S (2003) A fuzzy temporal rule-based velocity controller for mobile robotics. Fuzzy Sets Syst 134:83–99
16. Mucientes M, Iglesias R, Regueiro CV, Bugarín A, Barro S (2003) Intelligent systems: technology and applications, vol 2. Fuzzy Systems, neural networks and expert systems of CRC Press International volumes on intelligent systems techniques and applications, A fuzzy temporal rule-based approach for the design of behaviors in mobile robotics, CRC Press, pp 373–408

17. Mucientes M, Iglesias R, Regueiro CV, Bugarín A, Cariñena P, Barro S (2001) Fuzzy temporal rules for mobile robot guidance in dynamic environments. IEEE Trans Syst Man Cybern-Part C: Appl Re 31(3):391–398
18. Ng KC, Trivedi MM (1998) A neuro-fuzzy controller for mobile robot navigation and multirobot convoying. IEEE Trans Syst Man Cybern-Part B: Cybern 28(6):829–840
19. Pratihar DK, Deb K, Ghosh A (1999) A genetic-fuzzy approach for mobile robot navigation among moving obstacles. Int J Approx Reason 20(2):145–172
20. Urzelai J, Uribe JP, Ezkerra M (1997) Fuzzy controller for wall-following with a non-holonomous mobile robot. In: Proceedings of the 6th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'97), Barcelona, Spain, pp 1361–1368