Evolutionary design of a fuzzy controller for behaviors in mobile robots navigation

M. Mucientes[1] **D. L. Moreno**[1] **C. V. Regueiro**[2] **A. Bugarín**[1] **S. Barro**[1]
[1]Dept. of Electronics and Computer Science. University of Santiago de Compostela.
15782 Santiago de Compostela, Spain. {manuel, dave, alberto, senen}@dec.usc.es
[2]Dept. of Electronics and Systems. University of A Coruña.
15071 A Coruña, Spain. cvazquez@des.fi.udc.es

Abstract

The paper presents the evolutionary learning of a fuzzy controller for the wall-following behavior in mobile robotics. Learning comprises two stages: learning of the data base and a general rule base (Pittsburgh approach), and reduction of the generated rule base. The system has been tested implementing the learned behavior for a Nomad 200 mobile robot in different simulated environments.

Keywords: Evolutionary algorithms, fuzzy control, mobile robotics, wall-following behavior.

1 Introduction

The field of mobile robotics is characterized by the high amount of uncertainty present in real and unconstrained environments. Also, information provided by robot sensors is noisy and unreliable. Fuzzy logic has shown to be a useful tool when dealing with this uncertainty, and has been widely used for the design of behaviors in robotics [1]. The design of fuzzy controllers requires a deep knowledge on the task to be controlled and forces to spend a lot of time tuning the controller [2]. For these reasons, in the last years the use of learning methods for the design of fuzzy controllers has been generalized.

In this paper we present a methodology based on evolutionary algorithms for the automated design of fuzzy controllers. As opposed to other alternatives, for example the learning of Artificial Neural Networks (ANN) [3], the combination of evolutionary techniques and fuzzy control presents the advantage of the interpretability of the knowledge base (data and rule base).

There are three main approaches for the learning of knowledge bases with evolutionary algorithms: Pittsburgh, Michigan, and Iterative Rule Learning (IRL) [4]. The Pittsburgh approach codifies a complete knowledge base in each individual. Thus, it is very powerful, but also requires a lot of processing for the evaluation of such a number of controllers. On the contrary, in the Michigan approach, each individual codifies a rule. The main drawback of this method is the cooperation and competition between rules. Finally, IRL codifies a rule in each individual, but in each set of iterations only the best rule is added to the final knowledge base. This process continues until the final knowledge base is considered as a good solution to the problem.

The proposed system described in this paper is an improvement of the one presented in [5]. The objective is to reduce the number of rules, but maintaining the quality and accuracy of the controller. The current system comprises two stages: the first one is based on the Pittsburgh approach, and determines a good data base for the controller and a general rule base. In the next step, an algorithm that merges the existing membership functions reduces the previous rule base, and also selects the most adequate consequents for the final rule base.

Designer is only requested to select a few parameters: the precision of variables, also a parameter, δ, which relates the number of rules in the controller with its quality and accuracy, and parameter γ, that together with the precision gives the maximum number of linguistic labels for each variable. With the aim of proving the validity of the method, the wall-following behavior for a mobile robot has been implemented. The system has been tested in different simulated environments, showing a good performance as can be seen through the analysis of the average values of some parameters.

The paper is structured as follows: in section 2 the methodology we followed is presented. Section 3 explains the application of the methodology to the wall-following behavior. Section 4 shows the obtained results and, finally, conclusions and future work are discussed.

2 Description of the methodology

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

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

where R^i, *i=1, ..., NR*, is the i-th rule, X_j^i, *j=1, ..., NA*, and Y_k^i, *k=1, ..., NC*, are linguistic variables of the antecedent and consequent parts, respectively. NR is the number of rules, NA the number of input variables, NC the number of output variables, and A_j^i and B_k^i are linguistic values (labels) of these variables.

A number of examples have 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 precision p_n, $n = 1, ..., NV$, where $NV = NA + NC$ is the number of variables. Prior to the application of the learning method, the best action for each one of the examples is determined and saved in the variables in the consequent part of that example. These values are obtained before the beginning of the algorithm, trying and scoring all the possible actions for each example. Function SF, that scores the action of a rule over an example is application dependent[1], and a higher value describes a better action.

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

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

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. Parameter, $\delta \in [0,\ 1]$ 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. 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. This is because rules must be more specific, so they cover a lower number of examples and therefore the number of rules increases.

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 the final knowledge base.

[1] SF is not the fitness function.

The shape of the membership functions that are going to be learned is shown in figure 1. Points a_n^i and d_n^i are calculated using the values of b_n^i and c_n^i as:

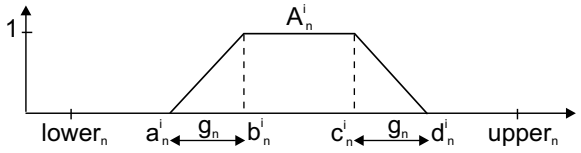


Figure 1: Shape of the membership function for the linguistic value A_n^i.

$$a_n^i = \max\left\{b_n^i - p_n,\, lower_n\right\} \tag{4}$$

$$d_n^i = \min\left\{c_n^i + p_n,\, upper_n\right\} \tag{5}$$

where $lower_n$ and $upper_n$ are the extreme points of the universe of discourse of variable n.

2.1 Obtaining the data base

In the first stage (Pittsburgh approach), the rules are generated using all the possible combinations of labels for each one of the input variables, so $NR = \prod_{n=1}^{NA} L_n$, where L_n (the number of labels for variable n) is a percentage, γ (predefined and equal for all the variables), of the number of values that the variable can take (remember that the values of the variables are discretised). Two types of real coded chromosomes are evolved. C_1^r (for individual r of the population) codifies the data base of all the variables as:

$$C_1^r = \left(c_{1,1}^r,\, ...,\, c_{1,L_1}^r,\, ...,\, c_{NV,1}^r,\, ...,\, c_{NV,L_{NV}}^r\right) \tag{6}$$

where $c_{n,t}^r$ represents (figure 2) the third parameter of a membership function of variable n ($t = 1, ..., L_n$).

On the other hand, C_2^r is codified as:

$$C_2^r = \left(s_{1,1}^r,\, ...,\, s_{1,NR}^r,\, ...,\, s_{NC,1}^r,\, ...,\, s_{NC,NR}^r\right) \tag{7}$$

where $s_{k,i}^r$ codifies the label that has been selected for the variable of the consequent part k of rule i.

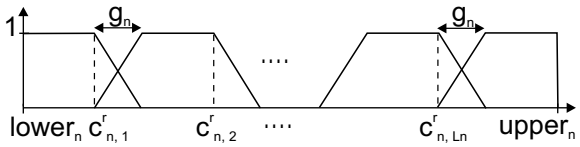


Figure 2: Universe of discourse of variable n for individual r.

The algorithm for the Pittsburgh approach has the following steps:

1. Initialize population.
2. Evaluate population.
3. Scale the fitness values.
4. While the maximum number of iterations is not exceeded.
   (a) Select the individuals.
   (b) Crossover and mutate.
   (c) Evaluate population.
   (d) Scale the fitness values.

The first step of the genetic algorithm randomly initializes the population of chromosomes C_1. The evaluation of each of the individuals of the population requires the preprocessing of chromosomes C_1 and C_2. If the individual has been crossed in a given iteration, then C_1 must be reordered and checked in order to avoid duplicate values. Afterwards, C_2 is calculated using the average values of the best actions for all of the examples covered by each rule. On the other hand, if the individual has not been crossed in a given iteration, then C_1 and C_2 remain unchanged. After preprocessing, all the individuals in the population are evaluated using the following function:

$$EF^r = \sum_{l=1}^{NE}\sum_{i=1}^{NR} SF\left(R^i\left(e^l\right)\right) \tag{8}$$

where NE is the number of examples. 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 r will be selected $int\left(\frac{EF^r}{AEF}\right)$ times, where AEF is the average value of all EF^r. Taking into account $frac\left(\frac{EF^r}{AEF}\right)$ the population is randomly filled up. Elitism has been implemented to avoid the loss of the best individuals due to crossover and mutation. For these individuals, an evolutionary strategy, (1+1)-ES, has been applied to C_2, but only over those genes that represent a rule which is not covering adequately its examples. The mutation operator modifies the genes, selecting a value between $s^r_{k,i} - 2$ and $s^r_{k,i} + 2$ with probabilities 10%, 30%, 20%, 30% and 10% respectively.

After selection, chromosomes C_1 of the individuals are crossed and mutated (simple mutation). A one-point crossover has been used as crossover operator, but it is applied only for the section of the chromosome corresponding to the variable, n, that has been randomly selected. If a pair of individuals are not crossed, then a (1+1)-ES is implemented over C_2 in the same way as for elitism. Finally, all the individuals are added to the new population.

2.2 Reduction of the rule base

Starting from the knowledge base obtained in the previous stage, a genetic algorithm is used in order to reduce the rule base merging adjacent membership functions. The algorithm comprises 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.
      ii. Crossover and mutate.
      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.

Each chromosome represents a rule, and has the following structure:

$$C^i = \left(l^i_1, r^i_1, ..., l^i_{NA}, r^i_{NA}, u^i_1, ..., u^i_{NC}\right) \quad (9)$$

Each gene codifies the label of a membership function from the data base. Variables from the antecedent part of the rule use two genes, l^i_j and r^i_j, so the membership function of this rule for this variable comprises all the linguistic labels from the data base between these ones connected with "or" operator. In that way, a reduction of the rule base can be done joining different labels from the previous data base and obtaining more general rules. Membership functions from the consequent part are codified only with a gene, u^i_k, which represents a linguistic label from the data base (no merging is needed in this part of the rule).

The first step of the algorithm initializes the population. This is randomly done among the rules in the previous rule base that cover an example that has not been covered yet, neither by the final rule base, nor by the present rule base. 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 covered yet by a rule of the final knowledge base. If an example is covered by a rule of the final knowledge base, it will decrease the fitness value of any rule of the population that is covering it. 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):

$$ASF^i = \frac{\sum_{l=1}^{NE} SF\left(R^i\left(e^l\right)\right)}{NEC^i} \quad (10)$$

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.

Rules verifying (2) but not (3) for any example, are 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 are firstly added are selected as in the population initialization stage. If it is the case that after that the population is still incomplete, then the best rules in the population are added. The selection procedure that has been applied is the same as in the previous stage. Elitism is also implemented, but in this step the best individuals are simply copied to the new population. Finally, the crossover operator is a one-point crossover, and the mutation operator has three equally probable options to operate on a gene: increase, decrease or leave the gene unchanged. This will provoke the extension or contraction of the membership function in a quantity equal to a linguistic label of the variable.

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 sequence of iterations. Once all the examples are covered by the rules in the resulting knowledge base the algorithm ends, and a solution to the problem is found.

3 Learning the wall-following behavior

The methodology presented in the previous section is applied in this paper for the design of a fuzzy controller for the wall-following behavior in mobile robotics. The main characteristics of the proposed approach are, first, the use of SF in order to evaluate the action of a rule over the robot. 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. Also, the trade off between the number of rules and the quality/accuracy of the controller can be adjusted selecting the value of δ.

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 the environment layout is permitting, and finally to avoid sharp movements, making smooth and progressive turns and changes in velocity. The controller can be set up 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.

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

$$DQ = \frac{left - hand\, distance}{RD} \qquad (11)$$

As it can be seen (figure 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 closest 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 objective linear velocity and orientation.

All the information used to calculate distances and orientations comes from the ultrasound sensors of a Nomad 200 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 given set of sensors, and the orientation will be the orientation of that sensor with respect to the advance direction.

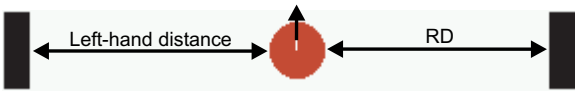


Figure 3: Description of some of the distances used for the calculation of the input variables.

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

$$SF\left(R^i\left(e^l\right)\right) = \frac{1}{\alpha_1 + \alpha_2 + \alpha_3 + 1} \quad (12)$$

where α_1, α_2, and α_3 are respectively:

$$\alpha_1 = 100\frac{|RD - reference\,distance|}{p_{RD}} \quad (13)$$

$$\alpha_2 = 10\frac{|maximum\,velocity - LV|}{p_{LV}} \quad (14)$$

$$\alpha_3 = \frac{|orientation|}{p_{orientation}} \quad (15)$$

and p_{RD}, p_{LV}, and $p_{orientation}$ are the precisions of the respective input variables. The precisions 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 p_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 (13), (14), and (15) 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.

4 Results

The system has been implemented with the values shown in table 1 for the most characteristic parameters.

Parameter	Pittsburgh	Reduction
Generations	100	50
Population	50	300
Crossover prob.	0.5	0.2
Mutation prob.	0.03	0.4
Elitism	8%	1.3%
δ	0.07	0.07
γ	0.7	

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

As can be seen, the mutation probability for the reduction stage is quite high, while the crossover probability is lower. The reason is that in this stage, the individuals are valid rules for the final knowledge base, and the aim is to do a local search (high mutation) in order to obtain more general rules. A high crossover probability should destroy good individuals, distracting the search. So, for the reduction stage, exploitation is preferred over exploration. Parameter δ must take the same value for the two stages, because it makes no sense to choose a different value for the quality of the controller in different stages of the design. Finally, γ only has to be specified in the first stage, as for the reduction stage the data base from the Pittsburgh approach is used.

It is critical in any automated learning process that the training data are diverse and chosen

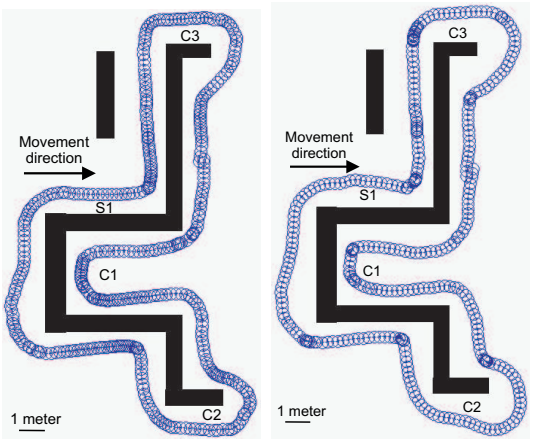
as different as possible from the test data. In this paper, the training data have been selected covering the universe of discourse of all the variables with a precision p_n, where n is a variable.

Strict comparison of the results obtained with other control systems described in the literature is not possible. In a field such as mobile robotics it is not viable to propose a standard *test bank* which would allow us to compare controllers in similar environments, and to consider the different mechanical and dynamic features of the different robots. Therefore, in order the system to be properly tested and validated we have selected a number of different simulated environments that include very different situations the robot usually faces during navigation: straight walls of different lengths, followed and/or preceded of a number of concave and convex corners, ... thus covering a wide range of contours to follow and truly defining very complex test environments, far from the complexity of the environments that are usually described in the literature.

In order to evaluate the quality of the learned behavior, a number of parameters have been measured. These are the average distance to the right wall (the wall that is being followed), the average linear velocity, the time spent by the robot going over the path, and the average velocity change. The latter parameter measures the change in the linear velocity between two consecutive cycles, reflecting the smoothness of the behavior.

Furthermore, a detailed comparison with a previous proposal described in [5] was performed attending to the parameters previously defined. Although both systems have different output variables (the objective velocity and orientation in this paper, and the linear acceleration and angular velocity in [5]), a qualitative/quantitative estimation of the results in both cases can still be done. The comparison and evaluation of the systems has been done in four different environments, and the results of the average values of ten tests for each path are presented in tables 2 (current system) and 3 (previous system).

Results show a very important reduction in the number of rules now needed to implement the behavior (46 in this paper and 100). Figure 4 shows the robot path in environment D for both the systems. It can be globally said that the wall is followed in a more reliable way, but due to this precision in the trajectory, values of average velocity and smoothness are not as high as in the previous approach. In this environment, the average distance between the robot and the wall is now 53 cm, while the previous approach obtained an average distance of 64 cm (take into account that 50 cm is the established objective distance). On the other hand, the average velocity is lower (41 cm/s) in contrast with 48 cm/s obtained in the previous paper. Also the change in velocity is higher in this work (8.35 cm/s) than in the previous system (6.35 cm/s), which indicates that smoothness in the behavior decreases.



(a) (b)

Figure 4: Path of the robot in environment D for the current (a) and previous system (b).

The comparison of both systems for environment D (figure 4) shows that at convex corners ($C2$ and $C3$) the current system traces the curve with higher precision, while the previous system doesn't reduce velocity enough, increasing the turning radius and, as a consequence, moving away from the wall. This also happens in straight walls after the corners ($S1$) where, for the previous system, the

Environment	RD (cm)	Velocity (cm/s)	Velocity change (cm/s)	Time (s)
A	59	46	8.69	72
B	54	43	9.70	120
C	54	36	7.75	114
D	53	41	8.35	126

Table 2: Average values of some parameters in the current system (46 rules).

Environment	RD (cm)	Velocity (cm/s)	Velocity change (cm/s)	Time (s)
A	70	57	4.23	62
B	65	51	4.59	106
C	63	48	5.48	87
D	64	48	6.35	111

Table 3: Average values of some parameters in the previous system (100 rules).

robot maintains an inadequate distance to the wall. A similar situation occurs at $C1$. It can be said as a conclusion that, in spite of the reduction in the number of rules, the new system follows the wall with more accuracy and precision than the previous system. This fact provokes a reduction in the obtained average velocities and also, an increase in the average velocity change.

5 Conclusions and future work

In this paper we have presented a methodology, based on evolutionary techniques, for the design of fuzzy controllers. The data base and a general rule base are learned using the Pittsburgh approach, and then a reduction of the rule base and a tuning of the consequents of the rules merging adjacent membership functions is performed. Designer is only requested to select parameter γ that, together with the precisions of the variables, describes the maximum number of linguistic labels for each variable, and also parameter δ, which relates the number of rules of the controller with its quality and accuracy.

The wall-following behavior for a Nomad 200 mobile robot has been implemented using this methodology, and the results have been compared with the ones obtained in a previous work, showing a reduction in the number of rules, but maintaining the quality and accuracy of the controller. In future works, we will add a new stage for the tuning of the controller on the real robot.

Acknowledgements

Authors wish to acknowledge support from the Spanish Ministry of Education and Culture through grants TIC2000-0873-C02-01 and TIC2003-09400-C04-03.

References

[1] A. Saffiotti, "The uses of fuzzy logic in autonomous robot navigation," *Soft Computing*, vol. 1, no. 4, pp. 180–197, 1997.

[2] M. Mucientes, R. Iglesias, C. V. Regueiro, A. Bugarín, and S. Barro, "A fuzzy temporal rule-based velocity controller for mobile robotics," *Fuzzy Sets and Systems*, vol. 134, pp. 83–99, 2003.

[3] D. Floreano and F. Mondada, "Evolutionary neurocontrollers for autonomous mobile robots," *Neural Networks*, vol. 11, pp. 1461–1478, 1998.

[4] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases.* World Scientific, 2001, vol. 19.

[5] M. Mucientes, D. L. Moreno, A. Bugarín, and S. Barro, "Evolutionary learning of a fuzzy controller for mobile robotics," in *Proceedings of the WSC8.*