Evolutionary learning of a fuzzy controller for mobile robotics

M. Mucientes, D. L. Moreno, A. Bugarín, and S. Barro

Dept. of Electronics and Computer Science. University of Santiago de Compostela. 15782 Santiago de Compostela, Spain.
{manuel, dave, alberto, senen}@dec.usc.es

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 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.

A knowledge base is formed of a data base (number and definition of the linguistic values, and universe of discourse of each of the variables) and the rule base. Some authors have utilized evolutionary algorithms for learning or tuning fuzzy controllers in robotics [1, 2, 3]. 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. 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 behaviour in a mobile robot. No restrictions are placed neither in the number of linguistic labels, nor in the values of the membership functions. The methodology is based on the Iterative Rule Learning (IRL) approach [4]. The paper is organized as follows: in section 2 some general comments about the evolutionary learning of knowledge bases are made, whilst in section 3 the genetic learning methodology employed is explained. Section 4 describes the application of the proposed algorithm to the wall-following behaviour, and section 5 presents the results we obtained. Finally, conclusions and future work are pointed out in section 6.

2 Evolutionary learning of knowledge bases

Learning of knowledge bases using evolutionary algorithms has three main approaches: Michigan, Pittsburgh and IRL [5]. In the Michigan approach [6], 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 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 [7].

This conflict is overcome by the Pittsburgh approach [8], 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 [4]), 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 penalised 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.

When learning knowledge bases, many approaches use a predefined number of linguistic labels for each variable, or the shape of the membership functions is constrained, facilitating the learning, but also limiting the solutions space and the strength of the method. The learning of both the data and rule bases can be done simultaneously, or in different stages. Some approaches firstly learn the data base and finally generate the rule base using the learned data base. Other approaches obtain the rule base using a predefined data base, and then tuning the shape of the membership functions to improve the learned knowledge base.

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 of linguistic labels, the 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) Initialise 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:

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

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 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. 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. 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:

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

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. 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 labelled 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 figure 1. Parameters b_n^i and c_n^i are learned, but points a_n^i and d_n^i are calculated as:

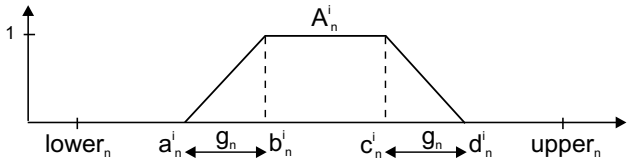


Fig. 1. Shape of the membership function for the linguistic value A_n^i.

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

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

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. For example, for a variable n having the following values: $lower_n = 0$, $upper_n = 60$, $g_n = 10$, $b_n^i = 40$ and $c_n^i = 60$, then $a_n^i = \max\{40 - 10,\, 0\} = 30$ and $d_n^i = \min\{60 + 10,\, 60\} = 60$.

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

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

The first step of the genetic algorithm consists in initialising 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 fulfil (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):

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

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 any 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 taken 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 sequence 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.

4 Learning the wall-following behaviour

The methodology presented in the previous section is applied here for the design of a fuzzy controller for the wall-following behaviour in mobile robotics.

The wall-following behaviour 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 layout of the environment is permitting, 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.

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

$$DQ = \frac{left - hand\, distance}{RD} \tag{8}$$

As it can be seen (figure 2), 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 a Nomad 200 robot. The distances and the orientation are obtained in two ways: if any of the walls (left or right) can be modelled 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.

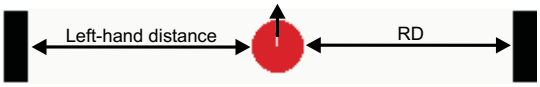


Fig. 2. Description of some of the distances used for the calculation of the input variables.

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

$$SF\left(R^i\left(e^l\right)\right) = \frac{1}{\alpha_1 + \alpha_2 + \alpha_3 + 1} \tag{9}$$

where α_1, α_2, and α_3 are respectively:

$$\alpha_1 = 100\frac{|RD - reference\,distance|}{g_{RD}} \tag{10}$$

$$\alpha_2 = 10\frac{|maximum\,velocity - LV|}{g_{LV}} \tag{11}$$

$$\alpha_3 = \frac{|orientation|}{g_{orientation}} \tag{12}$$

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 (10), (11), and (12) 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 penalised. 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 [9]:

$$O_k = \frac{\sum_{i=1}^{NR} \overline{Y}_k^i \, B_k^i(\overline{Y}_k^i)}{\sum_{i=1}^{NR} B_k^i(\overline{Y}_k^i)} \tag{13}$$

where $\overline{Y}_k^i$ is the centre of gravity of the linguistic label B_k^i, and O_k is the defuzzified value for variable k. This method does not take into account the width of the membership functions from the consequent part of the rule. For this reason, and in order to simplify the learning process, the membership functions of the variables of the consequent part are crisp. This simplification does not affect the quality of the obtained controller.

5 Results

The system described in the previous sections has been implemented with a crossover probability of 0.2 and a mutation probability (per gene) of 0.4 (remember that mutation can increase, decrease or maintain unchanged a gene with equal probability). 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 combination of rules from quite different areas of the search space, but a low crossover probability is still necessary to discover new promising areas.

The population size is 300 individuals, and the maximum number of generations is 50. Once that value is reached, the best rule of the population is added to the final knowledge base. The process is repeated until the final knowledge base covers all the examples. Different values for parameter δ (3) have been tried. All the parameters of evolutionary learning have been heuristically obtained.

Figure 3 shows the variation of the number of rules and the average velocity change of the final knowledge base 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 figure 4(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 figure 3, as δ increases the average velocity change decreases, and consequently the accuracy and quality of the controller rises.

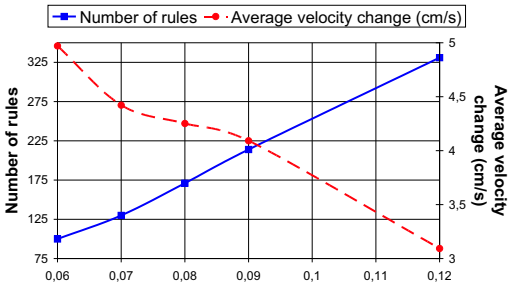


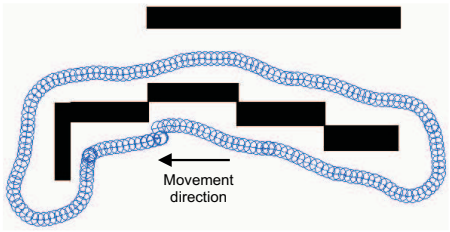
Fig. 3. Variation of the number of rules and the average velocity change of the final knowledge base with δ.

All the controllers that have been obtained were tested in three simulated environments using the Nomad 200 robot simulation software. We have to emphasize that the environments training set is different from the testing set. 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). None of the environments shown in this paper (figure 4) have been used during the learning process.

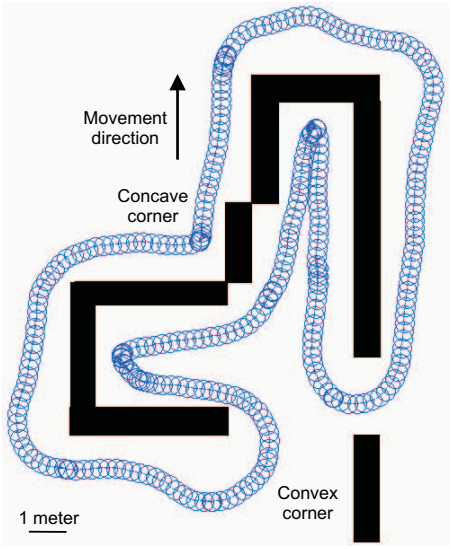
Figure 4 shows as an example the robot path in three simulated environments for $\delta = 0.06$. The robot trajectory is represented by circular marks. A higher concentration of marks indicates lower velocity. The learned controller has 100 rules, the maximum velocity the robot can reach is 61 cm/s, 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 table 1.

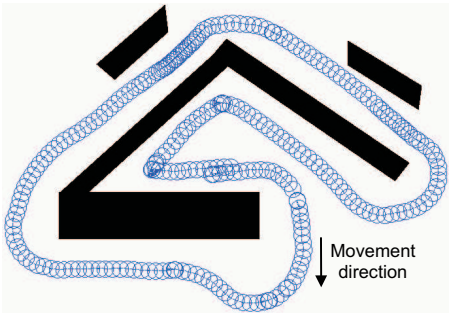
Environment	RD (cm)	Velocity (cm/s)	Velocity change (cm/s)	Time (s)
4(a)	70	57	4.23	62
4(b)	65	51	4.59	106
4(c)	63	48	5.48	87

Table 1. Average values of some parameters for the environments of figure 4.



(a)





(b) (c)

Fig. 4. Path of the robot in different simulated environments for $\delta = 0.06$.

Environment 4(b) is quite complex, with three concave corners and seven convex corners in a circuit of a length of 54 meters. 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 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 prioritised over a correct distance.

Nevertheless, some aspects could be improved. Thus, comparing this controller with [10, 11] (Fuzzy Temporal Rule-based controller, hand-designed involving 313 rules), the obtained behaviour 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 [10, 11] gets a trajectory closer to the shape of the contour being followed.

6 Conclusions and future work

A genetic algorithm based on the IRL approach for the learning of fuzzy controllers has been described. 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 behaviour. The learned control systems (for different values of δ) have been tested in a simulated environment with a high number of corners, showing a good performance both in the distance the wall was followed and in the average velocity.

Some aspects must be improved in the future, in order to get a better cooperation among rules of the final knowledge base, and enhance the interpretability of the rules. Another challenging work will be the learning of Fuzzy Temporal Rule-based controllers [12, 10, 11], which have a high degree of expressiveness and of analysing the evolution of variables, whilst taking past values into account.

Acknowledgements

Authors wish to acknowledge support from the Spanish Ministry of Education and Culture through grant TIC2000-0873-C02-01.

References

1. Braunstingl, R., Mujika, J., Uribe, J.P.: 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. Volume 5., Yokohama (Japan) (1995) 77–82

2. Leitch, D.: Genetic algorithms for the evolution of behaviours in robotics. In: Genetic Algorithms and Soft Computing. Volume 8 of Studies in fuzziness and soft computing. Physica-Verlag (1996) 306–328
3. Pratihar, D.K., Deb, K., Ghosh, A.: A genetic-fuzzy approach for mobile robot navigation among moving obstacles. International Journal of Approximate Reasoning **20** (1999) 145–172
4. Cordón, O., Herrera, F.: Hybridizing genetic algorithms with sharing scheme and evolution strategies for designing approximate fuzzy rule-based systems. Fuzzy sets and systems **118** (2001) 235–255
5. Cordón, O., Herrera, F., Hoffmann, F., Magdalena, L.: Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases. Volume 19 of Advances in Fuzzy Systems - Applications and Theory. World Scientific (2001)
6. Magdalena, L., Velasco, J.R.: Fuzzy Rule-Based Controllers that Learn by Evolving their Knowledge Base. In: Genetic Algorithms and Soft Computing. Volume 8 of Studies in fuzziness and soft computing. Physica-Verlag (1996) 172–201
7. Bonarini, A.: Evolutionary Learning of Fuzzy rules: competition and cooperation. In Pedrycz, W., ed.: Fuzzy Modelling: Paradigms and Practice. Kluwer Academic Press, Norwell (USA) (1996) 265–284
8. Carse, B., Fogarty, T.C., Munro, A.: Evolving fuzzy rule based controllers using genetic algorithms. Fuzzy sets and systems **80** (1996) 273–293
9. Mendel, J.M.: Fuzzy logic systems for engineering: A tutorial. Proceedings of the IEEE **83** (1995) 345–377
10. Mucientes, M., Iglesias, R., Regueiro, C.V., Bugarín, A., Barro, S.: A fuzzy temporal rule-based velocity controller for mobile robotics. Fuzzy Sets and Systems **134** (2003) 83–99
11. Mucientes, M., Iglesias, R., Regueiro, C.V., Bugarín, A., Barro, S.: A fuzzy temporal rule-based approach for the design of behaviors in mobile robotics. In: Intelligent Systems: Technology and Applications. Volume 2. Fuzzy Systems, Neural Networks and Expert Systems of CRC Press International Volumes on Intelligent Systems Techniques and Applications. CRC Press (2003) 373–408
12. Mucientes, M., Iglesias, R., Regueiro, C.V., Bugarín, A., Cariñena, P., Barro, S.: Fuzzy temporal rules for mobile robot guidance in dynamic environments. IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews **31** (2001) 391–398