Online tuning of a fuzzy controller in mobile robotics

Manuel Mucientes
Dept. of Elect. & Computer Science
Univ. of Santiago de Compostela
15782 Santiago de Compostela, Spain
manuel@dec.usc.es

Alberto Bugarín
Dept. of Elect. & Computer Science
Univ. of Santiago de Compostela
15782 Santiago de Compostela, Spain
alberto@dec.usc.es

Senén Barro
Dept. of Elect. & Computer Science
Univ. of Santiago de Compostela
15782 Santiago de Compostela, Spain
senen@dec.usc.es

***Abstract*— The design of behaviors is essential for the construction of complex control architectures in mobile robotics. The controller that implements a behavior can be learned in two stages: offline learning stage, and online tuning stage. In this paper we propose a methodology for online tuning, based on an evolutive strategy that is adequate to be implemented on the real robot. In order to be able to analyze the behavior of the robot along several control actions, we have considered the behavior as a multi-step problem, which means that the payoff is obtained after some control cycles (episode) using the Q-learning technique. The methodology has been applied for tuning the wall-following behavior using the Nomad 200 simulation software.**

I. Introduction

In mobile robotics, the control of the robot is usually implemented by hybrid architectures in which at the higher level the deliberative and planning tasks are solved, while at the lower level the reactive control is done. This makes the robot able to cope with complex tasks and, at the same time, interact and move in real environments with reactive capabilities. This reactive layer is generally composed of different behaviors that are selected and coordinated by the higher level. One of the main difficulties of mobile robotics is the high amount of uncertainty present in the measurements provided by the ultrasound sensors. Fuzzy logic has shown to be an appropriate tool for the implementation of behaviors [1], as the designed controllers can deal with the unreliable and noisy data.

On the other hand, the implementation of a controller requires expert knowledge, and a tedious tuning process in order to adjust the control actions that are going to be implemented on the real robot. This is not always possible, and is also highly time consuming. For these reasons, different learning techniques have been applied for obtaining controllers: evolutionary algorithms, neural networks, etc. In particular, evolutionary algorithms have been successfully applied for this task [2], [3], [4], [5]. One of the great advantages of evolutionary algorithms is that they let the designer to choose the most adequate tradeoff between accuracy and interpretability.

Learning of knowledge bases using evolutionary algorithms has three main approaches: Michigan, Pittsburgh and Iterative Rule Learning (IRL) [6]. In the Michigan approach, each chromosome represents an individual rule. Rules evolve through 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 [7]. The objective is to obtain a good rule base, which means to obtain good individual rules, but also rules that cooperate with each other to get adequate outputs.

On the other hand, in the Pittsburgh approach each chromosome represents a full knowledge base. This approach has a higher computational cost, since several knowledge bases have to be evaluated. Finally, in the IRL approach, each chromosome represents an individual rule, but contrary to the Michigan approach, a single rule is learned after each sequence of iterations, and not the whole rule base.

Learning a behavior on the real robot from the beginning is always very time consuming, difficult, and means that the robot can reach hazardous situations, because some non-adequate control actions can be selected during the learning process. For these reasons it seems adequate to divide the learning into two different stages: a first one in which the behavior is learned offline using a set of examples [2], [8]. At this stage, the time spent for learning is not as important, and also a wide range of control actions can be tested without any risk for the robot. The second stage consists on tuning the knowledge base obtained in the previous step on the real robot. In this way, the learning process on the real robot is faster and safer.

A number of different approaches have been described in the literature for learning behaviors in mobile robotics. They have different shortcomings: some of them spend a lot of time to learn the behaviors [2], [9], [10], [11]. Others need the definition of a lot of parameters, or a partial description of a knowledge base [12], which makes the design of new behaviors more difficult: parameters must be tuned and expert knowledge has to be acquired. Besides, sometimes the learned behavior is not general [11], [12], [13], thus the performance is adequate in some environments, but poor in others: the learned behavior is not reliable, and its implementation on the real robot will not be adequate. Finally, the interpretability of some of the learned controllers is low [2], [10], [13], [14] and, as a consequence, it will be difficult to detect and solve errors during the operation of the controller.

In this paper, a methodology for the tuning of a fuzzy controller on a mobile robot is presented. This approach

is valid for the tuning of different kinds of behaviors, as the designer has only to redefine the function that scores the states the robot has reached. The methodology is quite fast, only a few parameters must be defined, the obtained behavior is general (as will be shown with the results), and the interpretability of the knowledge base is high.

Among the three main approaches for learning knowledge bases, Michigan has been selected. The other two approaches are not adequate because they require a high amount of time to obtain a valid knowledge base. Tuning is implemented with a (1, 1)-ES (Evolution Strategy) that is applied to the consequents of the rules. It has been considered as a multi-step problem, in order to analyze the states reached by the robot along a number of iterations of the controller (episode). That way, the influence of the noisy measurements from the sensors, and the changes in the state of the robot due to its movement (detection of new features in the environment, movement of people, ...) can be reduced. The methodology has been applied for the tuning of the wall-following behavior using the Nomad 200 simulation software.

The paper is structured as follows: in the next section the tuning methodology is presented. Section 3 describes the application of the methodology to the wall-following behavior. Then, the obtained results are presented and, finally, conclusions and future work are commented.

II. Online tuning of behaviors

The rules that are going to be tuned have been obtained during a previous offline learning process, as described in [2], [8]. They are conventional fuzzy rules in Disjunctive Normal Form (DNF) like:

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

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 input variables, and NC the number of output variables. $\widetilde{A_j^i}$ is a subset of all the possible linguistic values of variable j, which are connected by a t-conorm (the maximum has been used):

$$\widetilde{A_j^i} = A_{j,1} \vee \ldots \vee A_{j,NL_j^i} \tag{2}$$

and $A_{j,l}$ and B_k^i are linguistic values, and NL_j^i is the number of linguistic labels in $\widetilde{A_j^i}$.

Prior to the beginning of the tuning process, the initialization of the population is done. The initial population will consist of κ copies of the rules of the previously obtained knowledge base ($NR = \kappa \cdot \#RB_0$), where $\#RB_0$ is the number of rules of the previous rule base. Then, $\kappa - 1$ copies of each rule are mutated in its consequent part, until not repeated rules remain. The mutation operator simply increases or decreases, with equal probability, the integer value that represents a linguistic label of a variable.

For each rule of the population the chromosome is constructed in the following way:

$$C^i = c_1^i,\, ...,\, c_{NC}^i \tag{3}$$

where c_k^i is an integer number that represents the linguistic label of the output variable k for rule i.

The tuning algorithm is as follows:

1) Construct the match set
2) Construct the subsets of rules
3) Select the subset of rules to be fired
4) Calculate payoff of previous action
5) If episode is over
   a) Calculate weights of the individuals
   b) Adjust fitness of the individuals
   c) Apply (1, 1)-ES

In each control cycle, the match set will be composed of all the rules that fulfill:

$$\mu_i = \widetilde{A_1^i}\,(x_1) \wedge \ldots \wedge \widetilde{A_{NA}^i}\,(x_{NA}) > 0 \tag{4}$$

Taking into account all the rules of the match set, the active niches (input subspaces) for this iteration are determined. The subsets of rules are constructed using all the combinations of the rules of the match set, considering the following constraints:

- All the active niches must be covered by each subset.
- Two rules covering the same active niche are not allowed in a subset.

The score of each subset m is calculated as:

$$ss_m = \sum_p \mu_p \cdot f_p \tag{5}$$

where f_p is the fitness of individual p. The subset of rules that will be fired can be selected in different ways: choosing the best subset, with a non-deterministic procedure (roulette wheel selection, etc.), ... In this paper we have applied the selection of the best subset.

An episode is defined as a number of control cycles, le, along which the payoff is calculated. At the end of the episode the reward is distributed among the rules. As will be explained when commenting the obtained results, using an episode with $le > 1$ increases the robustness of the learned behavior, because it minimizes the importance of the noisy measurements and the changes in the state of the robot due to the movement.

Payoff P is estimated using the Q-learning technique, which is a classic model-free algorithm for reinforcement learning from delayed rewards [15]. Q-learning is applied along the iterations of the episode in the following way:

$$P(t) = P(t-1) + \gamma^t \cdot SF(s\_ss_{t-1}(x(t-1))) \tag{6}$$

where t is the iteration of the episode, $\gamma \in [0, 1]$ is the discount factor, and $SF(s\_ss_{t-1}(x(t-1)))$ is the score assigned to the state reached by the robot starting from state $x(t-1)$ and applying the control actions proposed by the subset of rules

s_ss_{t-1} (the one selected in the previous iteration). Scoring function, SF, must be defined by the designer and is behavior dependent.

Once the maximum number of iterations (le) of the episode is reached, the payoff must be distributed among the rules. To do this, the weight of each rule is calculated:

$$w_i = \frac{\sum_{t=1}^{le} \mu_i^t \cdot sm_i^t \cdot (\gamma)^t}{\sum_{t=1}^{le} (\gamma)^t} \tag{7}$$

where μ_i^t is the degree of fulfillment of rule i at iteration t, and sm_i^t measures the similarity between the output of the controller and the output proposed by rule i in the following way:

$$sm_i^t = \prod_{k=0}^{NC} \frac{\max\left(0,\, \sigma_k - \left|y_k^t - \widetilde{y_{ik}^t}\right|\right)}{\sigma_k} \tag{8}$$

where y_k^t is the value for output variable k selected by the controller at iteration t, $\widetilde{y_{ik}^t}$ is the center of gravity of $\widetilde{B_k^i}$ (B_k^i after inference), and σ_k is the maximum difference that is permitted.

Thus the weight of a rule is proportional to the degree of fulfillment of the rule, but also to the closeness of the output proposed by the rule to the controller output. Similarity (sm_i^t) is useful in that situations in which a rule that is good for a state of the robot is fired with other rules that are bad for the same state, and also the selected action is not adequate (and vice versa). If similarity is not taken into account, the good rule would be penalized.

The point is that a good control action does not mean that the rules that contribute to that output are also good. As an example, suppose that four rules have been fired with equal degree of fulfillment: one of them with -1 as value for the output variable, two with 0, and the other one with $+1$. The control output will be 0 and, if this is a good output for that state, the payoff will be high. Without considering similarity all the rules will get the same payoff, while when including the similarity to calculate the weight, the first and last rules will not receive payoff. So the closer the output of the rule to the controller output the higher the similarity (and the weight) of the rule, and the proportion of the payoff the rule will receive. Of course, there can occur that sometimes a good rule does not receive payoff although it has been fired, but this only means that its fitness will not be adjusted at this iteration (the rule is neither penalized, nor rewarded). The same applies for bad rules.

The fitness for each individual is adjusted using the standard Widrow-Hoff delta rule [16], but taking into account the weights of the rules to distribute the adjustment [17]:

$$f_i \leftarrow f_i + \beta \cdot w_i \cdot (P - f_i) \tag{9}$$

where P is the payoff at the end of the episode, and $\beta \in [0, 1]$ is the learning rate parameter. In that way the difference between the payoff of the episode and the fitness of the rule contributes to change the fitness of the individual, but proportionally to the weight of the rule and the learning rate. As in [16], the Widrow-Hoff procedure is applied only after the individual has been adjusted at least $1/\beta$ times. Prior to that, the MAM technique is applied. This technique lets the fitness values approach faster to their true values, and makes the system less sensitive to the initial values of some parameters [16]. In this case, we have used a weighted average of the payoff values:

$$f_i = \sum_{t=1}^{1/\beta} v_i \cdot P(t) \tag{10}$$

where v_i is defined as:

$$v_i = \frac{\sum_{t=1}^{1/\beta} \mu_i^t \cdot (\gamma)^t}{\sum_{t=1}^{1/\beta} (\gamma)^t} \tag{11}$$

After the adjustment of the fitness values, an evolution strategy (1, 1)-ES is applied to those individuals i, for which $w_i > 0$ at the end of the episode, and that have been fired at least $1/\beta$ times since the beginning of the algorithm. (μ, λ)-strategies [18] use μ parents to create λ descendants, and the μ best individuals, out of the descendants only, are selected for the next population ($\mu \leq \lambda$). The (1, 1)-ES is applied to the consequents of the rules (individuals) with probability θ_m.

The procedure is as follows: once an individual is selected for applying the ES, the consequents of this rule are copied on the worst individual of the niche that rule belongs to. Then, the (1, 1)-ES is performed on this modified worst individual. With this strategy we maintain the best individual (elitism), and we modify the worst individual of the niche. The mutation operator is the same used in the initialization of the population.

III. Tuning the wall-following behavior

In order to evaluate the proposed methodology, we have selected the wall-following behavior, which 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.

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

$$DQ = \frac{left - hand\, distance}{RD} \tag{12}$$

As it can be seen (figure 1), 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.

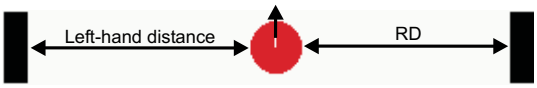

Fig. 1. Description of some of the distances used to calculate input variables.

The values for the distances and the orientation are obtained from the distances measured by the ultrasound sensors of the robot. We use the *distributed perception* [19]: distance is measured as the minimum distance of a set of sensors, and the orientation will be a weighted sum of the orientation of each sensor in the set, giving more weight to those sensors that detect closer obstacles.

Function SF, that scores the state reached by the robot, is defined as:

$$SF\,(x) = \sqrt{\frac{1}{\alpha_1 + \alpha_2 + \alpha_3 + 1}} \cdot \omega \tag{13}$$

where x is the state of the robot, ω is a scaling factor, and α_1, α_2, and α_3 are respectively:

$$\alpha_1 = 100 \cdot \frac{|RD - reference\,distance|}{p_{RD}} \tag{14}$$

$$\alpha_2 = 10 \cdot \frac{|maximum\,velocity - LV|}{p_{LV}} \tag{15}$$

$$\alpha_3 = \frac{|orientation|}{p_{orientation}} \tag{16}$$

p_{RD}, p_{LV}, and $p_{orientation}$ are the precisions of the respective input variables. 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 possible the comparison of the deviations of different variables and, as a consequence, the assignment of the weights for each one of the variables. These weights (100, 10 and 1 for (14), (15), and (16) 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. 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.

When learning a behavior on the real robot or with the simulation software, a collision avoidance module must oversee that none of the control actions that are implemented is going to provoke a collision. This module, taking into account the current velocity, advance direction, distances to the obstacles, and the linear acceleration and angular velocity that are going to be implemented, determines if this will make the robot reach a hazardous situation. In that case, the module tries to stop the robot, the episode ends, and the payoff is set to 0, so the rules that have been fired along the episode are penalized decreasing their fitness.

IV. Results

The proposed system has been tested using the Nomad 200 simulation software. The initial rule base has been obtained using a set of 48 examples and a hand-designed data base with the labels uniformly distributed along the universes of discourse of the variables. The number of labels is $\{3, 2, 3, 2, 9, 9\}$ for RD, DQ, $orientation$, LV, $linear\,acceleration$ and $angular\,velocity$, respectively. This generates 36 rules, one for each niche or input subspace. An example belongs to the niche that better covers its antecedent part. In the same way, the linguistic labels of the output variables for each rule are selected choosing those that better cover an example of the niche to which the rule belongs. The values of the parameters used for the online tuning methodology are: $\kappa = 5$, $\#RB_0 = 36$, $le = 5$, $\gamma = 0.8$, $\beta = 0.1$, $\theta_m = 0.05$, $\omega = 100$, and $\sigma_1 = \sigma_2 = 0.5$.

Tuning of the rule base was made in environment A (figure 2), and lasted five laps (approximately 11 minutes). Six environments (including environment A) have been chosen for comparing the tuned rule base with the initial rule base. These environments include very different situations that the robot usually faces during navigation: straight walls of different lengths, followed and/or preceded of a number of concave and convex corners, gaps, ... thus covering a wide range of contours to follow and truly defining very complex test environments.

Figure 2 shows the robot path along environment A, where tuning took place. The robot trajectory is represented by circular marks. A higher concentration of marks indicates lower velocity. 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. It is important to choose a training environment which covers as many different situations as possible in order to tune the rules of all the niches. However, there will be rules in some niches that will not be fired enough times ($1/\beta$ times), and in these cases the original rule will be selected for the final rule base.

It is of great importance to consider the behavior that is being tuned as a multi-step problem. The reason is twofold: firstly, in a real environment the ultrasound sensor measurements are very noisy and unreliable. So, a good control action can be taken in a situation (robot state) that is not truly the real one. As a result, a good rule will be penalized because of a failure in the estimation of the state of the robot, or vice versa. In the same way, it sometimes occurs that the controller

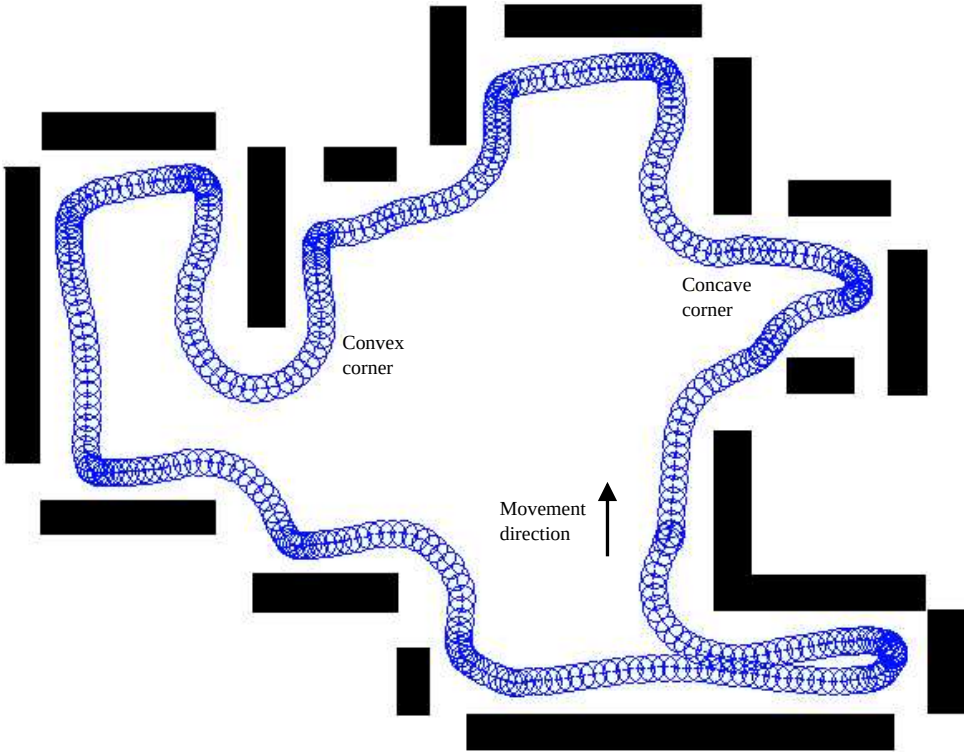

Fig. 2. Path of the robot along environment A for the tuned rule base.

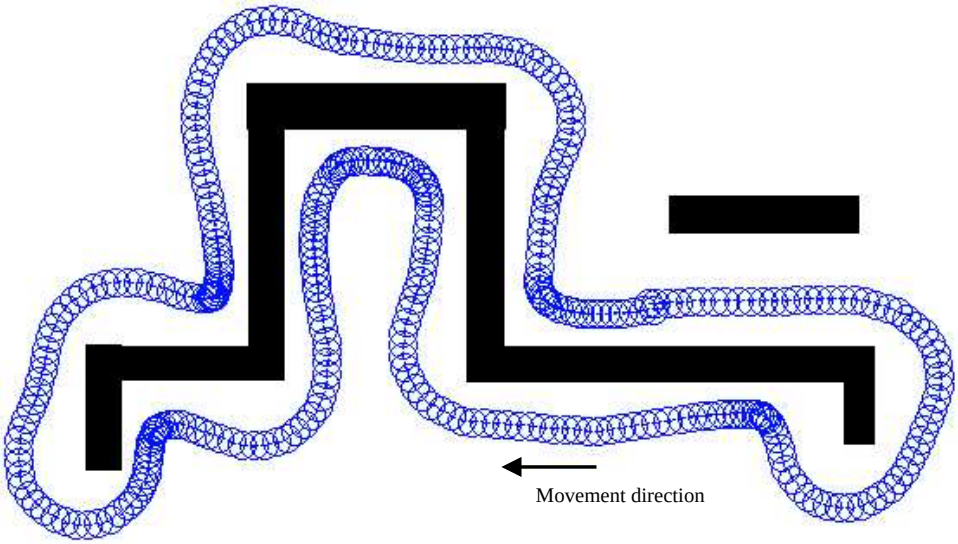

Fig. 3. Path of the robot along environment B for the tuned rule base.

selects an adequate control action for an state of the robot, but in the next cycle (when the action has to be evaluated) the configuration of the environment is completely different.

This can be seen in figure 4: at the first position, the robot is following the wall at an adequate distance and orientation and selects a rule that increases its speed, maintaining the orientation with the wall (a good control action for that state). At the second position, the wall has finished (there is a convex corner) and the current state of the robot indicates that the distance to the wall is too high and the orientation is bad. These measures are also unreliable because of the difficulties in the detection of the wall at that position: the angle between the direction of the different ultrasound sensors (that are in the right-front sector, thus the ones used to calculate the right-hand distance) and the direction perpendicular to the wall is high. That means that probably they will not be able to properly measure that distance due to the specular reflection for high angles. Again, a good action will produce a low

payoff. These situations are more frequent in environments with many corners, gaps, ...

For all these reasons, the evaluation of the actions of the controller along an episode is more robust and tends to minimize these kind of problems, as several control actions contribute to the calculation of the payoff, and also the robot has time to recover from a bad action. We have found significant differences in the quality of the tuned behavior: if the length of the episode (le) is 1, the obtained behavior is very poor, with $le = 3$ the quality increases a lot, while for $le = 5$ the best results are obtained.

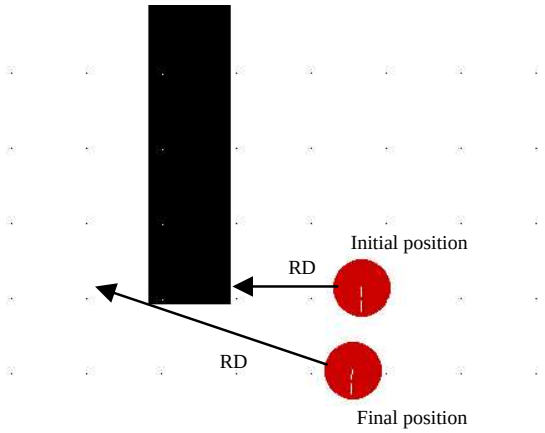

Fig. 4. Change of the state of the robot after one control cycle.

In robotics, it is difficult to compare the learned controllers with another ones proposed by other authors. The reasons are mainly due to the different characteristics of the robots (dynamic specifications, sensors of various kinds and with different distributions, ...), and also because each author designs their own environments for the test, due to the absence of a standard *test bank*. For these reasons, and in order to evaluate the obtained controller, the average values of some parameters that reflect the controller performance have been measured. The parameters are the average distance to the right wall (the wall that is being followed), the average linear velocity, the time spent by the robot along 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. Ten tests have been done for each one of the analyzed environments. The values of the parameters are shown in table I, while table II shows the values for the initial rule base.

TABLE I

AVERAGE VALUES OF SOME PARAMETERS FOR THE TUNED CONTROLLER.

Env.	RD (cm)	Velocity (cm/s)	Vel. change (cm/s)	Time (s)
A	44	45	4.41	125
B	51	51	2.83	105
C	51	51	3.94	66
D	49	54	2.15	72
E	50	55	2.18	109
F	51	52	3.08	105

TABLE II

AVERAGE VALUES OF SOME PARAMETERS FOR THE ORIGINAL CONTROLLER.

Env.	RD (cm)	Velocity (cm/s)	Vel. change (cm/s)	Time (s)
A	41	41	4.01	139
B	47	47	2.81	114
C	47	46	3.41	70
D	49	49	2.42	80
E	48	49	2.33	121
F	49	48	2.98	112

As can be seen, the tuned controller has increased the average velocity in all the environments around a 10% without degrading the smoothness of the behavior. On the other hand, the average right-hand distance has also been improved, and in some cases the reference distance has been maintained (as an average) along all the path. The worst case corresponds to environment *A* (the training environment), where the tuned system also beats the original one, but due to the complexity of the environment and the number of gaps, the average distance is not close to the reference distance.

In order to show the quality of the controller we are going to describe in detail the path of the robot in environment *A* (figure. 2). As said before, this environment is quite complex, with ten concave corners and six convex corners in a circuit of a length of 57 meters. The measurements of the ultrasound sensors are quite noisy due to the gaps present in the wall, and also because of the convex corners. These are truly difficult situations, because the robot's sensors may cease to correctly detect the wall at some given moments. The controller must also significantly reduce velocity at corners. All these situations provoke a reduction in the average distance and velocity. As can be seen, the robot follows the wall with a high precision, except at the corners, where it approaches to the wall (concave corners) or gets away from it (convex corners) in order to turn.

V. CONCLUSIONS AND FUTURE WORK

A methodology for rule base tuning of a fuzzy controller has been described. This field is of special importance in mobile robotics, where different behaviors have to be designed and integrated on the real robot. The design of behaviors on the real robot is very difficult and time consuming, so a better approach consists in dividing design in two stages: learning the behavior offline, and tuning it online.

The proposed algorithm has been tested for the online tuning of the wall-following behavior using the Nomad 200 simulation software, showing a good performance, increasing the speed of the robot around a 10%, and also improving the average distance to the wall. The results are promising for the application of the methodology on the real robot, where the performance of the initial rule base will be worst, and the enhancement of the behavior due to the tuning will be more relevant.

ACKNOWLEDGMENT

This work was supported in part by the Spanish Ministry of Science and Technology under grant no. TIC2003-09400-C04-03, and the DXID of the Xunta de Galicia under grant PGIDIT04TIC206011PR.

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, D. L. Moreno, C. V. Regueiro, A. Bugarín, and S. Barro, "Design of a fuzzy controller for the wall-following behavior in mobile robotics with evolutionary algorithms," in *Proceedings of the International Conference of Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'2004)*, Perugia (Italy), 2004, pp. 175–182.

[3] D. K. Pratihar, K. Deb, and A. Ghosh, "Optimal path and gait generations simultaneously of a six-legged robot using a ga-fuzzy approach," *Robotics and Autonomous Systems*, vol. 41, pp. 1–20, 2002.

[4] D. Gu, H. Hu, and L. Spacek, "Learning fuzzy logic controller for reactive robot behaviours," in *Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, 2003, pp. 46–51.

[5] A. Bonarini and V. Trianni, "Learning fuzzy classifier systems for multi-agent coordination," *Information Sciences*, vol. 136, pp. 215–239, 2001.

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

[7] A. Bonarini, "Evolutionary Learning of Fuzzy rules: competition and cooperation," in *Fuzzy Modelling: Paradigms and Practice*, W. Pedrycz, Ed. Norwell (USA): Kluwer Academic Press, 1996, pp. 265–284.

[8] M. Mucientes and J. Casillas, "Obtaining a fuzzy controller with high interpretability in mobile robots navigation," in *Proceedings of the IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2004)*, Budapest (Hungary), 2004, pp. 1637–1642.

[9] C. Zhou, "Robot learning with ga-based fuzzy reinforcement learning agents," *Information Sciences*, vol. 145, pp. 45–68, 2002.

[10] S. Thongchai, "Behavior-based learning fuzzy rules for mobile robots," in *Proceedings of the American Control Conference*, Anchorage, AK (USA), 2002, pp. 995–1000.

[11] H. Hagras, V. Callaghan, and M. Collin, "Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy-genetic system," *Fuzzy Sets and Systems*, vol. 141, pp. 107–160, 2004.

[12] D. Gu, H. Hu, J. Reynolds, and E. Tsang, "Ga-based learning in behaviour based robotics," in *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Kobe (Japan), 2003, pp. 1521–1526.

[13] O. Fuentes, R. Rao, and M. V. Wie, "Hierarchical learning of reactive behaviors in an autonomous mobile robot," in *IEEE International Conference on Systems, Man and Cybernetics*, 1995, pp. 4691–4695.

[14] A. Berlanga, A. Sanchis, P. Isasi, and J. M. Molina, "A general learning co-evolution method to generalize autonomous robot navigation behavior," in *Proceedings of the 2000 Congress on Evolutionary Computation*, La Jolla, CA (USA), 2000, pp. 769–776.

[15] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[16] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.

[17] J. Casillas, B. Carse, and L. Bull, "Fuzzy-xcs: an accuracy-based fuzzy classifier system," in *Proceedings of the XII Congreso Español sobre Tecnología y Lógica Fuzzy (ESTYLF 2004)*, Jaén (Spain), 2004, pp. 369–376.

[18] T. Bäck and H. P. Schwefel, *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons, 1996, ch. Evolution Strategies I: Variants and their computational implementation.

[19] J. Urzelai, J. P. Uribe, and M. Ezkerra, "Fuzzy controller for wall-following with a non-holonomous mobile robot," in *Proceedings of the sixth IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'97)*, Barcelona (Spain), 1997, pp. 1361–1368.