Learning Weighted Linguistic Rules to Control an Autonomous Robot

M. Mucientes,[1,*] R. Alcalá,[2,†] J. Alcalá-Fdez,[2,‡] J. Casillas[2,§]
[1]*Department of Electronics and Computer Science, University of Santiago de Compostela, 15782 Santiago de Compostela, Spain*
[2]*Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain*

A methodology for learning behaviors in mobile robotics has been developed. It consists of a technique to automatically generate input–output data plus a genetic fuzzy system that obtains cooperative weighted rules. The advantages of our methodology over other approaches are that the designer has to choose the values of only a few parameters, the obtained controllers are general (the quality of the controller does not depend on the environment), and the learning process takes place in simulation, but the controllers work also on the real robot with good performance. The methodology has been used to learn the wall-following behavior, and the obtained controller has been tested using a Nomad 200 robot in both simulated and real environments. © 2009 Wiley Periodicals, Inc.

1. INTRODUCTION

Control in mobile robotics requires different levels of action: planning (high level) and reacting (low level). Usually, the reactive layer is composed of behaviors that act directed by the planning level. These behaviors are implemented in different ways, being one of the most usual a fuzzy controller. The characteristic that makes specially useful a fuzzy controller for the implementation of a behavior is the ability that fuzzy controllers have to cope with noisy inputs. This noise appears when the sensors of the robot detect the surrounding environment and is particularly high when using ultrasound sensors (specular reflection, low angular resolution, etc.).

The design of an effective fuzzy controller is a tedious task, and for this reason some learning techniques such as evolutionary algorithms[1–7] and neural networks[8,9] have been applied. Evolutionary algorithms have some characteristics that make them specially useful for the design of fuzzy controllers: they are flexible to design different components of a controller, constraints can be easily included, and they

*Author to whom all correspondence should be addressed; e-mail: manuel.mucientes@usc.es.
†e-mail: alcala@decsai.ugr.es.
‡e-mail: jalcala@decsai.ugr.es.
§e-mail: casillas@decsai.ugr.es.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 24, 226–251 (2009)
 Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20334

let the designer decide the most adequate interpretability-accuracy trade-off for a specific controller.

However, some shortcomings appear when applying evolutionary learning to the design of these behaviors for mobile robotics: long learning tasks,[3,7,10–11] need of an initial partial description of the knowledge base,[2] and environment-dependent designs.[2,3,7,12,13] As a result, the learned behavior is not reliable and its implementation on the real robot is not adequate. In fact, in most cases only simulated experiments are provided, or a deep tuning is made to adapt the learned fuzzy controller to a real robot.

To deal with these drawbacks, we propose a simple but efficient and effective learning methodology (composed of a training data generation and a learning method) that obtains general fuzzy controllers that can be applied to any kind of environment. Besides, we are specially interested in not only generating fuzzy controllers with good behavior in simulated experiments but also to use them directly in the real robot, without any postprocessing or tuning task. Therefore, our methodology is able to quickly learn fuzzy controllers that can be directly applied to real environments.

The advantages of the methodology over other approaches are the following:

- The designer has to choose the values of only a few parameters: the universe of discourse, precision and granularity of each variable, and the scoring function. This characteristic facilitates the reusability of the methodology for a wide range of very different behaviors: wall-following, obstacle avoidance, moving object following, door crossing, and so on.
- The obtained controllers are general. This feature is very important because it ensures that the quality and reliability of the controller will not be affected by the environment in which the robot is placed. To get this characteristic, the learning process cannot be done in an environment or set of environments, because this does not guarantee that all the possible states of the robot have been tested. The solution is to learn from a set of examples that cover the universe of discourse of all the variables with a certain precision (values are discretized).
- Finally, learning on the real robot is usually not a good solution, as the learning process takes a lot of time (the robot needs to recharge batteries interrupting the learning process) and, also, during the learning stage the robot has to try control actions that can take it to hazardous situations (the robot could, e.g., crash with a wall). For these reasons, it seems much more appropriate to learn the behaviors in simulation, and then export the controllers to the real robot. Usually this requires a new tuning stage, but with our methodology this stage can be skipped and the learned controllers can be directly executed on the real robot without any change, showing a good performance.

The proposed learning methodology for behavior design in mobile robotics is twofold. First, a heuristic process is performed to automatically generate a data set that represents the behavior to be designed. Then, supervised learning is applied on this data set to extract an accurate fuzzy controller with good interpretability.

In this sense, we have used a learning method previously proposed in Ref. 14 that we believe fits perfectly with that purpose. Nevertheless, we have adapted some parts of the learning algorithm (mainly the use of several output variables and a different fitness function to regulate the number of rules) when applied to the proposed mobile robot problem. As shown in this paper, the considered learning methodology has been applied to the wall-following behavior with successful results

in both simulated and real environments. Comparison with other learning algorithms is also included.

The paper is structured as follows. Section 2 introduces the technique to generate data sets for the wall-following behavior, including the sensorial information preprocessing in the real robot. Section 3 presents the considered learning algorithm. Section 4 shows and discusses the obtained results in simulated and real environments. Finally, Section 5 outlines some conclusions.

2. LEARNING THE WALL-FOLLOWING BEHAVIOR

The proposed methodology is general and can be applied to different behaviors. To describe it in detail and evaluate its performance, we have selected the wall following behavior. This behavior is well known in mobile robotics and frequently used for the exploration of unknown indoor environments and also for the navigation of a robot between two points in a map. The requirements of a good wall-following controller are, first, to maintain a suitable distance from the wall that is being followed. In second place, the robot should also move as fast as possible, and finally the controller should 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 (d_{wall}), which is the desired distance between the robot and the selected wall, and the maximum velocity attainable by the robot ($v_{\max}$). 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 distances from the robot to the right (RD) and left walls (DQ), the orientation of the robot with respect to the wall, and its linear velocity (LV).

$$RD = \frac{\text{Right-hand distance}}{d_{\text{wall}}} \tag{1}$$

$$DQ = \frac{\text{Left-hand distance}}{\text{Right-hand distance}} \tag{2}$$

RD represents the relative right distance, thus a value of 1 indicates that the robot is at the reference distance to the wall it is following, a value lower than 1 means that it is closer, whereas a value more than 1 reflects a distance higher than the reference. In that way, any modification in the value of the reference distance to the wall (d_{wall}) will not affect the knowledge base that has been learned if we do not choose values of d_{wall} much lower than the one used for learning. DQ shows the relative position of the robot inside a corridor. A high value for DQ means that the robot is closer to the right-hand wall, whereas a low value indicates that the closer wall is the left-hand one.

The orientation of the robot (OR) measures the angle between the wall and the advance direction of the robot. Positive orientations indicate that the robot is

approaching to the wall, whereas negative values mean that the robot is moving away. Finally, the linear velocity of the robot (LV) is

$$LV = \frac{v_r}{v_{\max}} \tag{3}$$

where v_r is the real velocity of the robot. As it happened with d_{wall}, $v_{\max}$ can be modified in the learned behavior and the controller will have the same performance if the selected value for $v_{\max}$ is lower than the value used for the learning stage. The output variables of the controller are the linear acceleration (LA) and the angular velocity (AV)

$$LA = \frac{\text{Linear acceleration}}{a_{\max}} \tag{4}$$

$$AV = \frac{\text{Angular velocity}}{\omega_{\max}} \tag{5}$$

where $a_{\max}$ and $\omega_{\max}$ are the maximum linear acceleration and the maximum angular velocity of the robot, respectively.

2.1. Sensorial Information Preprocessing

The robot we have used in the experiments is a Nomad 200 (Figure 1a). The most important characteristics of this robot are a diameter of 53 cm, a maximum linear velocity of 61 cm/s, a maximum angular velocity of 45°/s, and an ultrasound range of between 15 and 647 cm. The robot is endowed with a ring of ultrasonic sensors. This kind of sensors is cheap and covers a range of distances from a few centimeters up to several meters. The disadvantage of these sensors is that they are very noisy, mainly due to their low angular resolution and also the specular reflection. The first problem appears when the robot is quite far from an obstacle (Figure 1b). When an object is hit by the sonar beam, it is not possible to distinguish in which area of the beam is the object placed. Thus, the sensor model places it in the middle of the beam. For high distances, the sensor beam is quite wide and the errors in the position of the objects can be high (1 m of error for objects placed at 5 m). For example, gap B in Figure 1b is not detected by the sonars, and obstacle A is detected in a wrong position.

On the other hand, specular reflection (Figure 1c) occurs when the angle between the sonar beam and the normal vector of the obstacle is high. The threshold angle will depend on the surface material of the obstacle (glass, wood, etc.). When a specular reflection occurs, a distance higher than the real one is measured in that direction.

Because of this kind of errors of the ultrasound sensors, the estimation of the position of the obstacles of the environment must take into account not only the present measurements but also measurements from different times. In that way, it is possible to build a local map of the environment that provides more reliability for the calculation of the distances and the orientations of the robot to the obstacles.

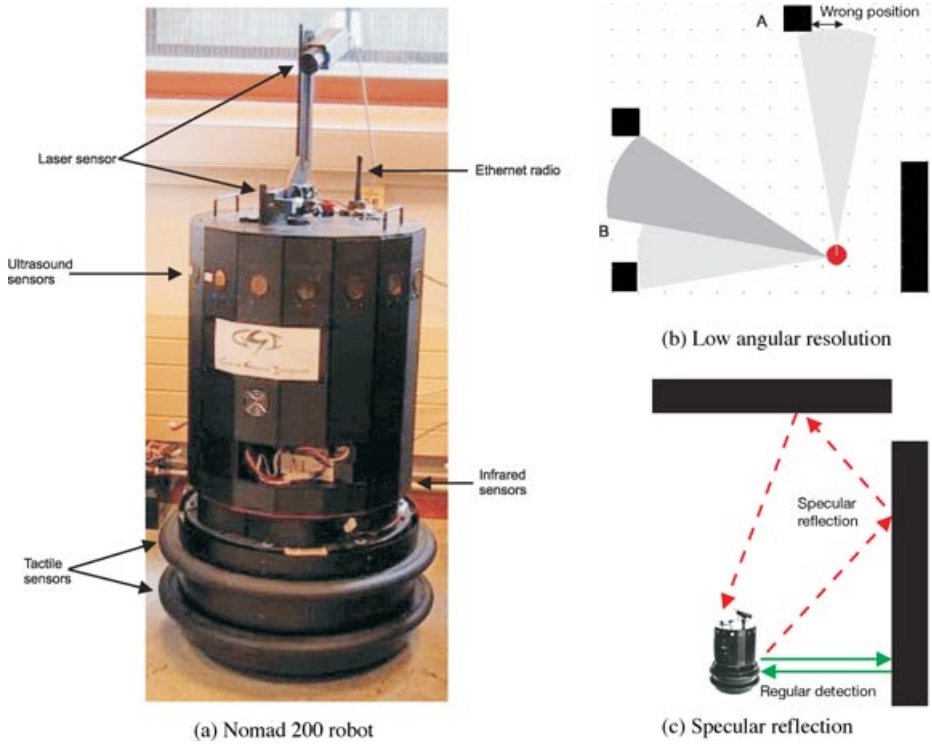


Figure 1. Nomad 200 robot and some sensorial problems.

The local map is a geometric map,[15] thus it is composed of lines. The key idea is to extract line segments from sonar range scans and to integrate the new lines with the line segments stored in the map. The line segments are obtained from a scan using the split-and-merge algorithm, which recursively subdivides the scan into sets of neighboring points that can be approximated by lines.

To fit a set of points to a line, we have used the approach proposed in Ref. 16. With this method, a line is defined by

$$\tan(2\phi) = \frac{-2\sum_i(\overline{x} - x_i)(\overline{y} - y_i)}{\sum_i((\overline{y} - y_i)^2 - (\overline{x} - x_i)^2)} \tag{6}$$

$$r = \overline{x}\cos\phi + \overline{y}\sin\phi \tag{7}$$

where $x_i(y_i)$ is the coordinate $x(y)$ of point i, $\overline{x}(\overline{y})$ in the average value of that coordinate over the point set, ϕ is the angle of the normal to the line, and r in the normal distance from the line to the origin of coordinates.

The integration of the new lines with the previous existing lines of the map is done in the following way:[15] first, the algorithm looks for the line in the map that is closer (and under a threshold) to the new line. The distance is measured adding

the distances of the endpoints of the new line to the map line. Then both lines will be integrated if they overlap. Integration means that the points of the new line are combined with the points of the map line, and the equation of the resulting line is calculated again using Equations 6 and 7.

2.2. Generation of the Training Set

If we want to learn a behavior offline for a mobile robot, but the behavior must work in any environment and on the real robot, then the examples training set must be carefully chosen. Also the number of examples cannot be very high as the learning algorithm must run in a reasonable amount of time. We can remove redundant examples from the training set, but not those that are essential for learning. As Jakobi[17] points out, we have to *distinguish those features of the world that are behaviorally relevant from those that are not*.

With the proposed methodology, the ability of the designer to distinguish those examples that are relevant to the problem only modifies the size of the training set, that is, the time that the learning algorithm lasts, but the performance of the obtained behavior is not affected. To learn the wall-following behavior, a set of 5,070 examples has been chosen. The examples have been automatically generated covering the whole universe of discourse of all the input variables. The process has the following steps:

- First, the designer must define for each variable the universe of discourse. Using the real universe of discourse, many values will not contribute to the learning process because they represent a similar situation and, thus, the same control action is implemented. For example, the *RD* variable can take any positive value (real universe of discourse), but the control action that the robot must implement for a value of $RD = 9.0$ will be the same as for $RD = 3.0$ (in both cases the robot is far away from the wall). For this reason, to learn a behavior, a reduced universe of discourse containing only the meaningful values will be defined for each variable. For the input variable OR (orientation), the real universe of discourse is $(-1800, 1800]$ (tenths of degree), but the one used for the generation of the training examples is $[-450, 450]$.
- For the same reason, the universes of discourse have been discretized, to minimize the search space, with a step or precision p_n, where n is the variable. For example, variable OR has a precision of 75, as changes in the orientation of, for example, 10 tenths of degree are not meaningful.
- The values that each one of the variables can take in the examples set are obtained using the universes of discourse, and the precisions previously defined. These values are calculated starting at the minimum value of a variable and increasing the value with p_n in each step until the maximum value of that variable is reached. For example, variable OR will take values in the following set: $-450, -375, -300, \ldots, 450$. Combining all the possible values for all the input variables, the examples are generated.
- The examples training set has been generated, but it is incomplete, as the values of the output variables for each example have not been calculated. This is done for each example trying all the possible combinations of discretized output values and selecting the best one, that is that which generates a lower value of the scoring function (Equation 8).

The process for the generation of the examples training set is quite fast, as it takes around 1 s (with an Intel(R) Pentium(R) 4 CPU 3.20 GHz processor). Table I

Table I. Characteristics of the variables (5,070 examples).

Variable	Universe of discourse	Precision (p_n)	Granularity	Number of values
RD	[0, 3.]	0.25	4	13
DQ	[0, 2.]	0.5	2	5
OR	[−450, 450]	75	5	13
LV	[0, 1]	0.2	2	6
LA	[−1, 1]	0.125	9	17
AV	[−1, 1]	0.05	9	40

shows the details of the universes of discourse, precisions, granularities (number of linguistic labels), and number of possible values for each variable. These values have not been optimized and, probably, there are still many redundant examples. If, instead of those values, higher precisions or wider universes of discourse are selected, then the number of training examples will be higher (there will be redundant examples), but the learned knowledge base will have a very similar performance.

2.3. Evaluation of the Knowledge Bases

The evaluation of a control action for the robot is done measuring how close is the state reached by the robot after the control action to the best state that the robot can reach for the same beginning state.

To obtain this evaluation for all the examples of the training set over a knowledge base, it is necessary to define function SF (scoring function) that scores the action of the rule base over an example. SF is a sum of values α_n, where each α_n measures the deviation in the value of variable n from its ideal value. For the wall-following behavior SF is

$$SF(RB(e^l)) = \alpha_{RD} + \alpha_{LV} + \alpha_{OR} \tag{8}$$

where e^l is an example, and α_{RD}, α_{LV}, and α_{OR} are, respectively,

$$\alpha_{RD} = 100 \cdot \frac{|RD - d_{\text{ref}}|}{p_{RD}} \tag{9}$$

$$\alpha_{LV} = 10 \cdot \frac{|v_{\max} - LV|}{p_{LV}} \tag{10}$$

$$\alpha_{OR} = \frac{|OR|}{p_{OR}} \tag{11}$$

α_{RD} represents the difference between RD and the d_{ref}. d_{ref} takes generally the value of d_{wall} (the distance at which the robot should follow the wall). But when the passageway is very narrow (values under two times d_{wall}), d_{ref} will take half of the passageway width. As the reader has realized, there is not α_{DQ}. The reason is that there is not an ideal value for DQ, but its value will depend on the environment

characteristics. Nevertheless, it is necessary to include *DQ* as an input variable of the fuzzy controller, mainly for those situations in which the passageway is very narrow.

Precisions (p_n) are used in these equations 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 Equations 9, 10, and 11, respectively) have been heuristically determined and indicate how important the deviation in the value of a variable is with respect to the deviation of other variables. These weight values are the only ones that have been tried. 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 with velocity and, finally, the least important contribution to function SF is for the orientation of the robot.

Thus, low values of SF indicate a good control action. A score of 0 means that the robot has reached the best state, i.e., that in which it is at the reference distance to the wall, parallel to it and with the maximum speed. The index that measures the global quality of the encoded rule set is

$$f(RB) = \frac{1}{2 \cdot NE} \sum_{l=1}^{NE} \left(g(e^l)\right)^2 \tag{12}$$

where NE is the number of examples, and $g(e^l)$ is defined as

$$g(e^l) = \begin{cases} \left(1 - h(e^l)\right) \cdot \omega + 1 & \text{if } h(e^l) \leq 1 \\ \exp\left(1 - h(e^l)\right) & \text{if } h(e^l) > 1 \end{cases} \tag{13}$$

being ω a scaling factor that has been set to 1000 (no other values have been tried), and $h(e^l)$:

$$h(e^l) = \frac{scoreEx\left(e^l\right) + 1}{SF\left(RB(e^l)\right) + 1} \tag{14}$$

where $scoreEx(e^l)$ is the score obtained example by e^l after applying the control action codified in the example, and $SF(RB(e^l))$ is the score for that example after the control action of the knowledge base is executed. In this way, it is possible to compare the output proposed by the example with the output selected during the learning process by the current knowledge base.

Equation 14 indicates how good is the control action selected by the knowledge base for example e^l. The higher the value of $h(e^l)$, the better the action. It is interesting to note that the output selected by the knowledge base can obtain a better (lower) score (Equation 8) than the control action codified in the example. The reason is that the generation of the training set uses discrete values for the variables (also for the

output variables), but the knowledge base selects outputs in a continuous universe of discourse. For example, the linear acceleration (LA) codified in an example can be 0.75, but the best value for LA can be in the set $(0.75 - p_{LA},\ 0.75 + p_{LA})$. Thus a value of LA in this set could obtain a better score for this example.

2.4. Reusability of the Methodology

The proposed methodology has two main characteristics that make it specially useful to learn behaviors in mobile robotics:

- The designer only has to define a few parameters: universe of discourse, precision, and granularity of each variable, and the scoring function.
- The methodology can be applied to learn different behaviors just changing some of these parameters.

The steps that a designer should follow to implement a new behavior are

1. Define the input and output variables.
2. Select the universe of discourse and precision of each variable to automatically generate the examples training set. In the worst case, the designer will select as universe of discourse the real universe of discourse of the variable, and as precision a very low value. Accordingly, there will be redundant examples and the learning process will be slower due to the higher number of examples, but the performance of the obtained controller will be very similar.
3. Choose the granularity of each variable. Values over the optimal ones will increase the number of rules of the learned knowledge base, but the performance will be similar.
4. Define the equations to evaluate the different knowledge bases during the learning process. Equations 12–14 are general. This means that they can be applied without any modification to the learning of different behaviors. Only the scoring function (Equation 8) must be modified in a very simple way. As SF measures the deviation of a state from the ideal state, an α_n must be generated for each variable n. The equation for each α_n will have the following general form:

$$\alpha_n = weight_n \cdot \frac{|value_{variable} - value_{ideal}|}{p_n} \tag{15}$$

where $weight_n$ must take into account the importance of that variable.

In the next section, the algorithm that has been used to learn the knowledge bases is described.

3. LEARNING OF COOPERATIVE WEIGHTED LINGUISTIC RULES

A good technique to improve the cooperation of the rules is to use weighted fuzzy rules,[18,19] in which the modification of the linguistic model structure by an importance factor (weight) is considered for each rule. By means of this technique, the way in which these rules interact with their neighbor ones can be indicated.

In Ref. 14, the weighted COR (WCOR) methodology was presented to include the weight learning within the original COR methodology (proposed in Ref. 20 and extended in Ref. 21). In this way, considering a *miso* (multiple input single output) fuzzy system, for each possible antecedent combination in the problem input space, this method automatically learns the best consequent label and its associated weight.

To learn a fuzzy controller for the problem of mobile robot navigation, we have adapted the learning algorithm proposed in Ref. 14, considering in this case a *mimo* (multiple input multiple output) fuzzy system. The fitness function has also been adapted from the one presented in Ref. 14 to penalize rule bases with too many rules. This is an important issue in fuzzy control for mobile robot navigation, as the actions of the robot are easily understandable in a model with less rules.

The WCOR methodology is guided by example covering criteria to obtain antecedents (fuzzy input subspaces) and candidate consequents.[14] Depending on the combination of this technique with different *ad hoc data-driven methods*, different learning approaches can arise. In this work, we will consider the Wang and Mendel's method[22] (WM) for this purpose—approach guided by examples. The WCOR methodology following this approach consists of two main stages:

1. *Search space construction*: It obtains a set of candidate consequents for each rule.
2. *Selection of the most cooperative fuzzy rule set and learning of weights*: It performs a combinatorial search among these sets looking for the combination of consequents with the best global accuracy and learning rule weights to improve even more the interaction among the different rules.

In this section, we present the adaptation of the WCOR methodology to obtain a cooperative set of weighted linguistic rules in the mobile robotic navigation problem. In the following, we describe the use of the weighted linguistic rules, the mentioned WCOR methodology, and the evolutionary algorithm applied to the WCOR methodology.

3.1. The Use of Weighted Linguistic Rules

Using rule weights[18,19] has been usually considered to improve the way in which rules interact, improving the accuracy of the learned model. In this way, rule weights suppose an effective extension of the conventional fuzzy reasoning system that allows the tuning of the system to be developed at the rule level.[18,19]

When weights are applied to complete rules, the corresponding weight is used to modulate the firing strength of a rule in the process of computing the defuzzified value. From human beings, it is very close to consider this weight as an important degree associated with the rule, determining how this rule interacts with its neighbor ones. We will follow this approach, because the interpretability of the system is appropriately maintained. In addition, we will only consider weight values in [0, 1] because it preserves the model readability. In this way, the use of rule weights represents an ideal framework for extended linguistic fuzzy modeling when we search for a trade-off between accuracy and interpretability. To do so, we will follow the weighted rule structure and the inference system proposed in Ref. 19

extended for multiple output variables:

$$\begin{array}{l}\text{IF } X_1 \text{ is } \mathcal{A}_1 \text{ and } \ldots \text{ and } X_n \text{ is } \mathcal{A}_n \\ \text{THEN } Y_1 \text{ is } \mathcal{B}_1 \text{ and } \ldots \text{ and } Y_m \text{ is } \mathcal{B}_m \ with\ [w],\end{array} \tag{16}$$

where X_i (Y_j) are the linguistic input (output) variables, $A_i(B_j)$ are the linguistic labels used in the input (output) variables, w is the real-valued rule weight, and *with* is the operator modeling the weighting of a rule.

With this structure, the fuzzy reasoning must be extended. The classical approach is to infer with the FITA (first infer, then aggregate) scheme and compute the defuzzified output of the jth variable as the following *weighted sum*:

$$y(j) = \frac{\sum_h m_h \cdot w_h \cdot P_h(j)}{\sum_h m_h \cdot w_h}, \tag{17}$$

with m_h being the matching degree of the hth rule, w_h being the weight associated with the hth rule, and $P_h(j)$ being the characteristic value of the output fuzzy set corresponding to that rule in the jth variable. In this contribution, the center of gravity will be considered as a characteristic value and the *minimum t-norm* will play the role of the implication and conjunctive operators.

3.2. The Weighted COR Methodology

Let us assume the previous existence of the following input–output data set and an initial fuzzy partition:

- *Input–output data set-* $E = \{e_1, \ldots, e_l, \ldots, e_N\}$, with $e_l = (x_1^l, \ldots, x_n^l, y_1^l, \ldots, y_m^l)$, $l \in \{1, \ldots, N\}$, N being the data set size, and $n(m)$ being the number of input (output) variables—*representing the behavior of the problem being solved.*
- *Fuzzy partition of the variable spaces*. In our case, uniformly distributed fuzzy sets are created. Let $\mathcal{A}_i$ be the set of linguistic terms of the ith input variable, with $i \in \{1, \ldots, n\}$, and $\mathcal{B}_j$ be the set of linguistic terms of the jth output variable, with $j \in \{1, \ldots, m\}$, with $|\mathcal{A}_i|$ ($|\mathcal{B}_j|$) being the number of labels of the ith (jth) input (output) variable.

Considering this data set and once the initial fuzzy partition is determined, the WCOR algorithm will consist of the following steps:

1. *Search space construction*:

   1.1. *Define the fuzzy input subspaces containing positive examples*: To do so, we should define the positive example set ($E^+(S_s)$) for each fuzzy input subspace $S_s = (A_1^s, \ldots, A_i^s, \ldots, A_n^s)$, with $A_i^s \in \mathcal{A}_i$ being a label, $s \in \{1, \ldots, N_S\}$, and $N_S = \prod_{i=1}^n |\mathcal{A}_i|$ being the number of fuzzy input subspaces. In this paper, we use the following:

$$\begin{array}{l}E^+(S_s) = \{\ e_l \in E \mid \forall i \in \{1, \ldots, n\}, \\ \qquad\qquad \forall A_i' \in \mathcal{A}_i, \mu_{A_i^s}(x_i^l) \geq \mu_{A_i'}(x_i^l)\ \ \}\end{array} \tag{18}$$

   with $\mu_{A_i^s}(\cdot)$ being the membership function associated with the label A_i^s.

Among all the N_S possible fuzzy input subspaces, consider only those containing at least one positive example. To do so, the set of subspaces with positive examples is defined as $S^+ = \{S_h \mid E^+(S_h) \neq \emptyset\}$.

1.2. *Generate the set of candidate rules in each subspace with positive examples*: First, the candidate consequent set associated with each subspace containing at least an example, $S_h \in S^+$, is defined. In this paper, we use the following:

$$\begin{aligned} C(S_h) = \{\ & (B_1^{k_h}, \ldots, B_m^{k_h}) \in \mathcal{B}_1 \times \ldots \times \mathcal{B}_m \mid \\ & \exists e_l \in E^+(S_h)\ where\ \forall j \in \{1, \ldots, m\}, \\ & \forall B'_j \in \mathcal{B}_j,\ \mu_{B_j^{k_h}}(y_j^l) \geq \mu_{B'_j}(y_j^l)\ \}. \end{aligned} \tag{19}$$

Then, the candidate rule set for each subspace is defined as $CR(S_h) = \{R_{k_h} = [\text{IF } X_1 \text{ is } A_1^h \text{ and ... and } X_n \text{ is } A_n^h \text{ THEN } Y_1 \text{ is } B_1^{k_h} \text{ and ... and } Y_m \text{ is } B_m^{k_h}]$ such that $(B_1^{k_h}, \ldots, B_m^{k_h}) \in C(S_h)\}$.

To allow COR to reduce the initial number of fuzzy rules, the special element $R_\emptyset$ (which means "do not care") is added to each candidate rule set, i.e., $CR(S_h) = CR(S_h) \cup R_\emptyset$. If it is selected, no rules are used in the corresponding fuzzy input subspace.

2. *Selection of the most cooperative fuzzy rule set and learning of weights*.

- *Problem representation*: For each final rule R_h, we have S_h, $C(S_h)$, and $w_h \in [0, 1]$. Since S_h is kept fixed, the problem will consist of determining the consequents and the weight associated with each rule. Two vectors, c_1 and c_2, of size $|S^+|$ (number of rules finally obtained), are defined to represent this information, where,

$$c_1[h] = k_h \mid R_{k_h} \in CR(S_h) \tag{20}$$

$$c_2[h] = w_h,\ \forall h \in \{1, \ldots, |S^+|\} \tag{21}$$

In this way, the c_1 part is an integer-valued vector in which each cell represents the index of the consequents used to build the corresponding rule. The c_2 part is a real-valued vector in which each cell represents the weight associated with this rule. Finally, a problem solution is represented as follows:

$$c = c_1\ c_2 \tag{22}$$

- *Search algorithm*: This stage is performed by running a combinatorial search algorithm to look for the combination in the c vector that represents the RB with the best accuracy. Since the tackled search space is usually large, approximate search techniques should be used. To do that, we consider the use of the simple genetic algorithm (GA) presented in the next subsection.

An index $f(RB)$ measuring the global quality of the encoded rule set is considered to evaluate the quality of each solution. To obtain solutions with a high interpretability, the original function is modified to penalize excessive number of rules:

$$f'(RB) = f(RB) + \beta \cdot f(RB_0) \cdot \frac{\#RB}{|S^+|} \tag{23}$$

with $\beta \in [0, 1]$ being a parameter defined by the designer to regulate the importance of the number of rules, $\#RB$ being the number of rules used in the evaluated solution (i.e., $|S^+| - |\{R_h \in RB\ such\ that\ R_h = R_\emptyset\}|$), and RB_0 being the initial rule base considered by the search algorithm.

Since it is based on the original COR methodology,[20,21] the WCOR methodology has some inherited advantages that make it very useful to learn fuzzy controllers in mobile robot navigation. We can mainly highlight two characteristics:

1. *Search space reduction*: The WCOR methodology reduces the search space when compared with other rule base learning methods[23] and allows it to be quicker and to make a better solution exploration in the space of the consequents, complementing this search with the use of the weights that improves the interaction among the different rules. It is due to two main reasons:

   - The fact of assigning each example to only one subspace will involve to reduce the number of candidate consequents, because the positive example sets are reduced.
   - The use of a restrictive condition to construct $C(S_h)$ (see Equation 19) that generates a low number of candidate rules.

   This is an important issue for the learning of fuzzy controllers, where a high number of examples are used. In the wall-following behavior presented in this paper, 5,070 examples have been used, and the employed methodology spends around 3 h (with an Intel Centrino 1.73 GHz processor) to obtain the controller.
2. *Interpretability issues*: This methodology also has some interesting advantages from the interpretability of the obtained fuzzy knowledge point of view. In this case, the membership functions keep invariable although we have to consider the rule weights. As said, these weights can be considered as importance factors because they present values in the interval [0.0, 1.0]. Furthermore, the COR methodology achieves a rule reduction process at the same time as the learning one with the aim of improving the accuracy (the cooperation among rules and thus the system performance can be improved by removing rules) and interpretability (a model with less rules is more interpretable) of the learned model. In most of the cases, the learning of the rule weights helps to remove a major number of rules. These are important issues in fuzzy control for mobile robot navigation, as the actions of the robot are easily understandable.

3.3. Genetic Algorithm Applied to the WCOR Methodology

As said, a GA is used to perform an approximate search among the candidate rules with the main aim of selecting the set of consequents with the best cooperation and simultaneously learning the weights associated with the obtained rules. It is based on a standard binary-coded genetic algorithm whose characteristics are presented in the following. The general scheme for this basic GA is showed in Figure 2.

- *Selection and Reproduction*: The selection probability calculation follows the Baker's linear ranking.[24] Chromosomes are sorted in order of raw fitness, and then the selection probability of each chromosome, $p_s(c^p)$, is computed according to its rank, $rank(c^p)$—with $rank(c^{best}) = 1$— by using the following nonincreasing assignment function:

$$p_s(c^p) = \frac{1}{N_C} \cdot \left(\eta_{\max} - (\eta_{\max} - \eta_{\min}) \cdot \frac{rank(c^p) - 1}{N_C - 1} \right), \tag{24}$$

  where N_C is the number of chromosomes and $\eta_{\min} \in [0, 1]$ specifies the expected number of copies for the worst chromosome (the best one has $\eta_{\max} = 2 - \eta_{\min}$ expected copies).

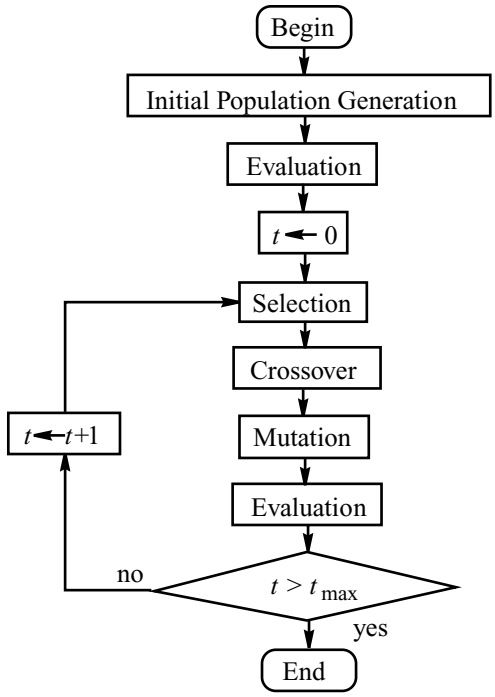


Figure 2. Flowchart of the implemented GA.

Most of the works in the literature have fixed $\eta_{\min}$ to 0.75, becoming a standard value since this value works well practically in all the cases.

The classical *generational*[25] scheme has been considered in this algorithm. In this way, linear ranking is performed along with *stochastic universal sampling*.[24] This procedure guarantees that the number of copies of any chromosome is bounded by the floor and by the ceiling of its expected number of copies. Together with the Baker's stochastic universal sampling procedure, an elitist mechanism (that ensures to maintain the best individual of the previous generation) has been considered.

- *Initial Gene Pool*: The initial pool is obtained by generating a possible combination at random for the c_1 part of each individual in the population. And for the c_2 part, it is obtained with an individual having all the genes with value 1, and the remaining individuals generated at random in [0, 1].
- *Fitness Function*: The fitness function will be the mentioned objective function, defined in Equation 23.
- *Crossover*: Owing to the different nature of both parts considered for each chromosome, different operators working in each part, c_1 and c_2, are required. Taking into account this aspect, the following operators are considered: In the c_1 part, the standard two-point crossover[26] is used, whereas in the c_2 part, the max-min-arithmetical crossover[27,28] is considered (see Figure 3). Two-point crossover is a classical operator for binary and integer coding, which properly handles the search space for the c_1 part, becoming one of the most used operators for this coding type. Max-min-arithmetical crossover is an adequate operator for real-coded genetic algorithms and allows different levels of exploitation and exploration to be introduced simultaneously in the search process.[29]

Two-point crossover involves interchanging the fragments of the parents contained between two points selected at random (resulting two descendants). On the other hand,

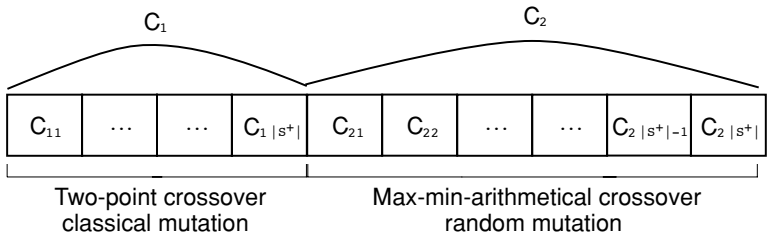


Figure 3. Genetic representation and operators' application scope.

by using the max-min-arithmetical crossover, if $c_2^v = (c[1], \ldots, c[k], \ldots, c[n])$ and $c_2^w = (c'[1], \ldots, c'[k], \ldots, c'[n])$ are crossed, the next four offsprings are obtained:

$$c_2^1 = a \cdot c_2^w + (1 - a) \cdot c_2^v \tag{25}$$

$$c_2^2 = a \cdot c_2^v + (1 - a) \cdot c_2^w \tag{26}$$

$$c_2^3 \text{ with } c^3[k] = \min\{c[k], c'[k]\} \tag{27}$$

$$c_2^4 \text{ with } c^4[k] = \max\{c[k], c'[k]\} \tag{28}$$

with $a \in [0, 1]$ being a parameter chosen by the GA designer. Typically, a is fixed to 0.3, which gives one offspring nearer to the first parent and the other nearer to the second parent.[27,29]

In this case, eight offspring are generated by combining the two from the c_1 part (two-point crossover) with the four from the c_2 part (max-min-arithmetical crossover). The two best offspring obtained replace the two corresponding parents in the population.

- *Mutation*: The operator considered in the c_1 part randomly selects a gene ($h \in \{1, \ldots, |S^+|\}$) and changes at random the current consequent B_{k_h} by other consequent $B_{k_h'}$ such that $R_{k_h'} \in CR(S_h)$. On the other hand, the selected gene in the c_2 part takes a value at random within the interval [0, 1].

Figure 3 shows the application scope of the different operators considered.

4. RESULTS

The quality of the learned controllers has been tested on a Nomad 200 robot, both with the Nomad 200 simulation software and on the real robot. We present a deep analysis of the different parameters that are specific of our methodology. Nevertheless, a study on the parameters related to the GA has not been performed because there are different studies on these parameters and we consider general values that have demonstrated a good performance in different problems. The following values have been considered for the parameters that are specific for the GA[a]:

[a]With these values, we have tried to select standard common parameters that work well in most cases, instead of searching for very specific values to apply the GA to our specific problem. Moreover, we have set a large number of generations to allow the algorithm to achieve an appropriate convergence. No significant changes were achieved by increasing that number of generations or by reasonably changing these parameters.

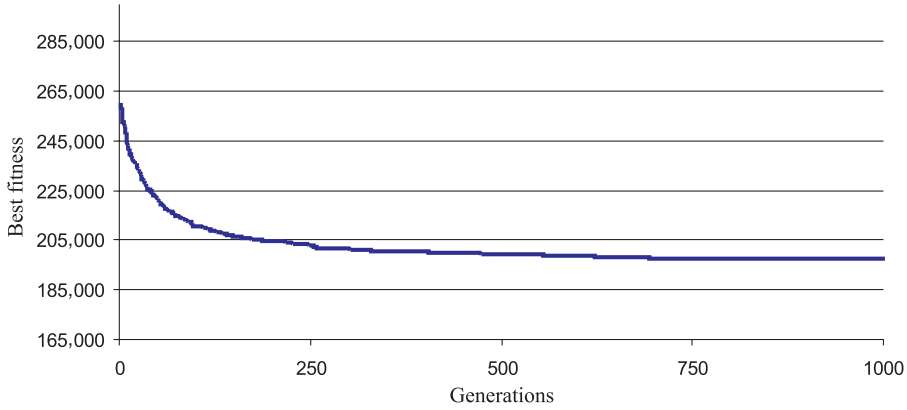


Figure 4. Convergence of the algorithm in one of the experiments.

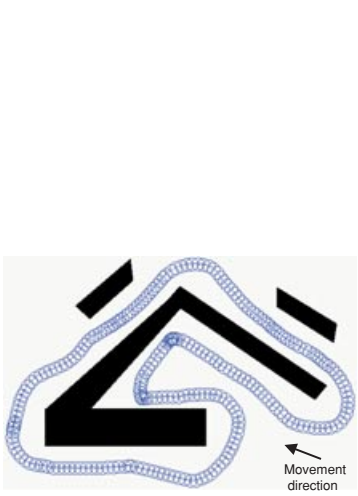
61 individuals, 1000 generations, 0.1 and 0.6 as mutation and crossover probability per chromosome, and 0.3 for the factor a in the max-min-arithmetical crossover. Figure 4 shows the convergence of the algorithm in one of the experiments: As the number of generations increases (x axis), the fitness of the best individual (y axis) decreases. Over a threshold, new generations do not improve the results of the algorithm.

4.1. Simulated Robot

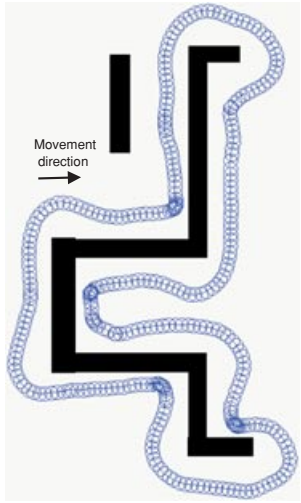
Six environments have been chosen to test the learned controllers. These environments include very different situations that the robot usually faces during navigation: straight walls of different lengths, followed and/or preceded by a number of concave and convex corners, gaps, and so on thus covering a wide range of contours to follow and truly defining very complex test environments. It is important to remark that none of these environments have been used during the learning process. Thus, the training set has been a set of 5,070 examples that uniformly cover the universe of discourse of the input variables.

Figure 5 shows the robot path along four of the test environments. 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.

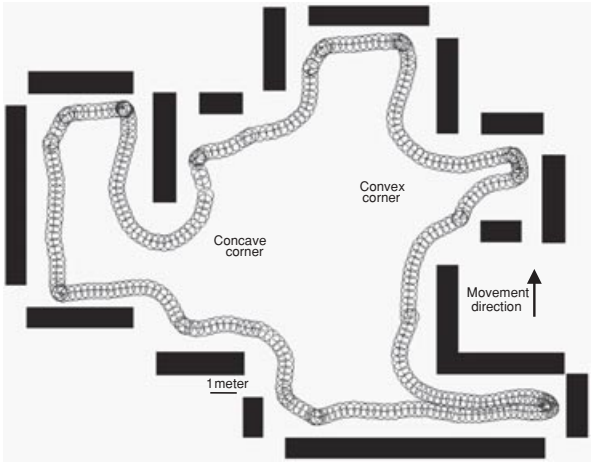
Several controllers have been learned for different values of β (Equation 23), a parameter to regulate the importance of the number of rules. Finally, the controller with $\beta = 0.2$ has been selected. The controller has 41 rules. Ten tests have been done for each one of the analyzed environments. The average values measured for some parameters that reflect the controllers performance are shown in Table II. 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



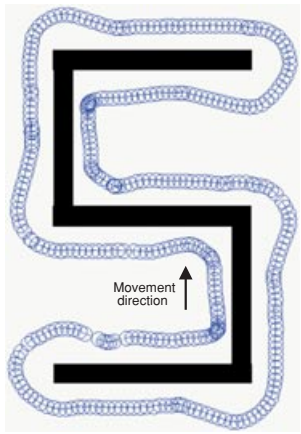
(a) Environment C



(b) Environment F



(c) Environment E



(d) Environment D

Figure 5. Path of the robot along some of the different simulated environments.

linear velocity between two consecutive cycles, reflecting the smoothness of the behavior.

The learned controller obtains an average right distance very close to the reference distance in most of the environments, whereas the average speed is very high. At environment *C*, owing to the existence of a couple of narrow passageways, this average distance is, logically, reduced, together with the average velocity.

The knowledge base of the model obtained by WCOR is presented in Figure 6. In the rule base, each row of the table represents a fuzzy subspace and contains

Table II. Average values of some parameters for WCOR controller (41 rules)

Environment	*RD* (cm)	Velocity (cm/s)	Δ Velocity (cm/s)	Time (s)
A	51	52	7.94	101
B	52	54	8.14	62
C	45	43	5.90	90
D	52	53	8.88	111
E	47	49	9.82	111
F	51	54	7.03	96

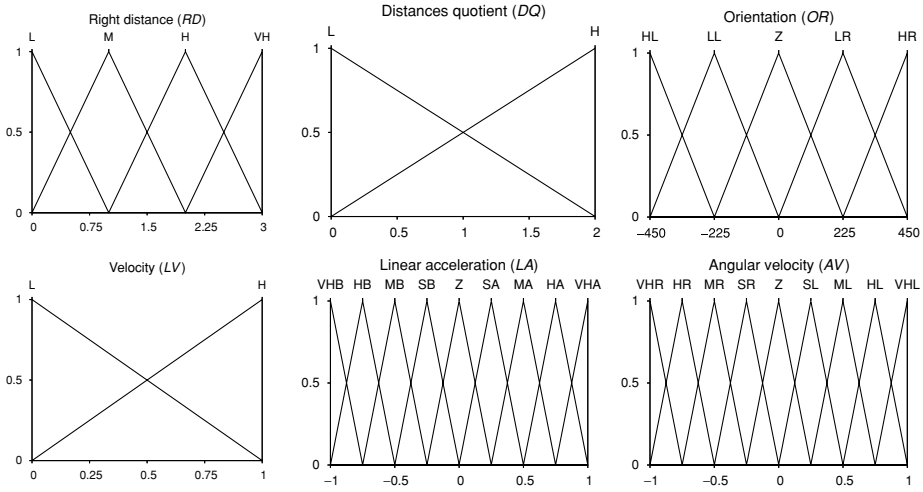
its associated output consequents, that is the corresponding labels together with the respective rounded rule weight. Moreover, the *absolute importance weight* for each fuzzy rule has been graphically showed by means of the gray color scale, from black (1.0) to white (0.0). In this way, we can easily see the importance of a rule with *respect to their neighbor ones*, which could help the system experts to identify important rules. Rules that have a higher weight than their neighbors or rules that are the ones of their regions can be considered as important or predominant rules. Rules that have a more or less similar weight than their neighbors can be considered as cooperative rules in their corresponding zones. And rules that have a lower weight than their neighbor ones can be considered as complementary rules that help important rules to better model the control surface.

To show the quality of the controller, we are going to describe in detail the path of the robot in environment E (Figure 5c). This environment is quite complex, with 10 concave corners and 6 convex corners in a circuit of a length of 57 m. The measurements of the ultrasound sensors are quite noisy due to the gaps present in the wall and also because of the convex corners. The controller must also significantly reduce velocity at corners. All these situations reduce the average distance and velocity. As it 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) to turn.

4.1.1. Analysis of Parameter β

Parameter β regulates the importance of the number of rules. The greater the value of β, the lower the number of rules. The objective of this section is to show that if we do not select a very high value of β, the performance of the controller is very similar for different values of this parameter. Table III shows for several values of β the number of rules of the knowledge bases, and the average value of some parameters for environment E.

As the reader can see, values of β under 0.4 obtain a similar performance. Low values of β have a higher number of rules and, as a consequence the smoothness of the behavior is greater (lower value of parameter ΔVelocity). For $\beta = 0.4$ and 0.5, the number of rules is really low, so the behavior is not smooth and the average velocity decreases. In particular, for $\beta = 0.5$ the average right distance takes a value more than the 20% lower than the expected reference distance.



(a) Data base

Rule	*RD*	*DQ*	*OR*	*V*	*LA*	*AV*	*Weight*	*Rule*	*RD*	*DQ*	*OR*	*V*	*LA*	*AV*	*Weight*
R_1	L	L	LL	L	VHB	VHR	0.4610	R_{22}	H	L	Z	L	HA	VHR	0.5758
R_2	L	L	LL	H	VHB	VHR	0.4896	R_{23}	H	L	LR	H	SA	MR	0.2513
R_3	L	L	Z	L	Z	MR	0.6664	R_{24}	H	L	HR	L	HA	VHL	0.5471
R_4	L	L	Z	H	HB	SR	0.5435	R_{25}	H	L	HR	H	SA	HL	0.5595
R_5	L	H	LL	L	MA	HR	0.7276	R_{26}	H	H	HL	L	VHB	VHR	0.9999
R_6	L	H	Z	L	MA	HL	0.4845	R_{27}	H	H	HL	H	VHB	VHR	0.9563
R_7	L	H	Z	H	HB	ML	0.5023	R_{28}	H	H	LL	L	HA	VHR	0.9506
R_8	L	H	LR	H	VHB	VHL	0.7363	R_{29}	H	H	Z	L	HA	VHR	0.4529
R_9	L	H	HR	L	VHB	VHL	0.9441	R_{30}	H	H	Z	H	SA	VHR	0.2210
R_{10}	M	L	Z	H	SA	HR	0.3402	R_{31}	H	H	LR	L	HA	MR	0.3612
R_{11}	M	L	LR	H	Z	VHL	0.4244	R_{32}	H	H	LR	H	SA	MR	0.2122
R_{12}	M	L	HR	L	SA	HL	0.5472	R_{33}	H	H	HR	L	HA	HL	0.7878
R_{13}	M	L	HR	H	MB	VHL	0.4369	R_{34}	H	H	HR	H	SA	VHL	0.3859
R_{14}	M	H	HL	L	Z	VHR	0.1770	R_{35}	VH	L	LR	L	HA	VHR	0.5530
R_{15}	M	H	HL	H	VHB	VHR	0.4526	R_{36}	VH	L	HR	L	HA	HR	0.4223
R_{16}	M	H	LL	H	SA	VHR	0.2548	R_{37}	VH	L	HR	H	SA	HR	0.3854
R_{17}	M	H	Z	L	HA	Z	0.2084	R_{38}	VH	H	LL	L	HA	VHR	0.0936
R_{18}	M	H	LR	L	HA	VHL	0.6242	R_{39}	VH	H	LR	L	HA	VHR	0.7325
R_{19}	M	H	LR	H	SA	VHL	0.3779	R_{40}	VH	H	LR	H	SA	VHR	0.5631
R_{20}	M	H	HR	L	Z	VHL	0.6931	R_{41}	VH	H	HR	L	HA	HR	0.5146
R_{21}	M	H	HR	H	VHB	VHL	0.7580								

(b) Rule base

Figure 6. Knowledge base generated by the WCOR method.

Table III. Average values of some parameters at environment E for different values of β.

β	Rules	*RD* (cm)	Velocity (cm/s)	ΔVel. (cm/s)	Time (s)
0.0	73	53	49	4.76	114
0.1	53	49	49	8.92	113
0.2	41	47	49	9.82	111
0.3	38	47	48	12.24	115
0.4	28	51	38	20.90	141
0.5	25	40	29	19.41	177

Table IV. Characteristics of the variables (24,624 examples).

Variable	Universe of discourse	Precision (p_i)	Granularity	Number of values
RD	[0, 3.]	0.2	4	16
DQ	[0, 2.]	0.25	2	9
OR	[−450, 450]	50	5	19
LV	[0, 1]	0.125	2	9
LA	[−1, 1]	0.125	9	17
AV	[−1, 1]	0.05	9	40

4.1.2. Influence of the Number of Examples

Our objective is to show that the quality of the learned behavior is independent of the size of the training set. Table IV shows the universe of discourse, precision, granularity, and the number of values for each variable of the new training set. We have modified all the precisions of the input variables of Table I, and instead of 5,070 examples, now the training set has 24,624. This increases the learning times, but the performance of the learned behavior is not affected.

The average values of the usual parameters for the learned controller with $\beta = 0.2$ and the training set of 24,624 examples are shown in Table V. The obtained results are very similar to those of the original knowledge base (Table II).

4.1.3. Study of the Number of Labels

The third and last parameter analysis consists in testing a controller that has been learned with the same training set and value of β, but with a different number of labels for some of the variables. Instead of the granularities of Table I, we have used those of Table VI.

We have tested the obtained controller in all the environments (Table VII). The number of rules is higher because the initial number of rules (due to the new

Table V. Average values of some parameters for WCOR controller with $\beta = 0.2$ and 24,624 examples (37 rules).

Environment	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
A	55	53	6.00	103
B	53	52	8.28	64
C	48	47	7.50	84
D	52	53	4.94	112
E	49	50	8.17	111
F	52	51	7.19	101

Table VI. New values for the granularities.

	RD	*DQ*	*OR*	*LV*	*LA*	*AV*
Granularities	7	2	7	3	9	9

Table VII. Average values of some parameters for WCOR controller with $\beta = 0.2$, 5,070 examples, and granularities of Table VI (182 rules).

Environment	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
A	56	54	3.15	100
B	55	54	4.45	62
C	55	55	2.48	71
D	56	56	2.89	106
E	51	50	6.09	113
F	55	55	4.49	95

granularities) is also higher, but the performance is again very similar to that of the original controller (Table II). The average distance in worse in four environments, and better in one, whereas the average speed is better in all the environments. On the other hand, the smoothness of this controller is clearly better than the original controller due to the much higher number of rules.

4.1.4. Comparison with Other Learning Methodologies

Our WCOR controller (Table II) has also been compared with other three controllers obtained applying different methodologies. The first of the methodologies was presented in Ref. 30, in which the COR methodology without weights was applied for the same purpose, but in that case the search technique was an ant colony optimization algorithm, whereas in this approach a genetic algorithm has been used. Table VIII shows the values of average parameters for the COR controller,[30] which has 52 rules for $\beta = 0.2$.

As it can be seen, the controller learned using WCOR has increased the average velocity and improved the average distance in most of the environments. Also the smoothness of the behavior has been highly improved in all of the environments, in spite of the lower number of rules of the knowledge base learned with WCOR (41 vs. 52 rules).

The second of the methodologies with which WCOR has been compared is the well-known Wang and Mendel's algorithm.[22] It consists of an ad hoc algorithm based on generating a candidate fuzzy rule for each available example and then performs a selection among inconsistent set of rules. Table IX shows the results

Table VIII. Average values of some parameters for COR controller (52 rules)[30]

Environment	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
A	53	51	10.39	107
B	53	51	10.17	66
C	50	49	9.40	81
D	53	51	10.29	104
E	49	46	11.22	123
F	53	52	10.08	116

Table IX. Average values of some parameters for WM controller (80 rules).[22]

Environment	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
A	55	53	2.42	101
B	55	55	2.59	61
C	52	51	2.02	76
D	54	55	2.02	107
E	48	47	5.02	119
F	54	53	2.74	101

obtained with this methodology. WCOR (Table II) obtains a better average right distance (between 4% and 10% of improvement) in all the environments but C, whereas in E the distances are similar. The average velocities are similar in all the environments, except again in C. The reason is that in the narrow passageways the controller learned with WM does not reduce the speed and follows the left wall closer than the right one, and although the average values of the parameters are better, the behavior is worse in this environment than the one implemented with WCOR. Finally, as the number of rules of WM is the double of WCOR (80 vs. 41 rules), the smoothness of WM is clearly better. However, the interpretability degree is significantly higher in the knowledge base generated by WCOR.

The third comparison has been done with the accurate linguistic modeling (ALM) learning method.[31] This method consists of a first stage that generates a set of candidate double-consequent fuzzy rules (for each data set example, two rules with the antecedent that best matches the input variables and the two consequents that best cover the output are generated) and then applies a genetic algorithm based fuzzy rule selection process. The method obtains a lower interpretability degree due to the fact that it considers inconsistent fuzzy rules (i.e., rules with the same antecedent but different consequent) in exchange of improving the accuracy.

Table X presents the results obtained in all of the environments with the knowledge base learned with this methodology. In most of the environments, the average distance is far away from the reference distance. In some environments, the error is of more than 20%, which is unacceptable as this is the most important parameter. On the other hand, the average speed is clearly higher in two of the environments, and similar in the others, whereas the smoothness is much higher in all the environments. However, as said previously, the interpretability of the inconsistent fuzzy rule set generated by ALM is worse.

Table X. Average values of some parameters for ALM controller (73 rules).[31]

Environment	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
A	51	54	1.24	99
B	55	59	1.25	56
C	43	53	1.86	73
D	46	55	1.10	106
E	37	48	3.73	115
F	41	51	1.94	102

Table XI. Average values of some parameters for WCOR (41 rules) and COR controllers (52 rules) on the real robot (environment of Figure 7)

Controller	*RD* (cm)	Velocity (cm/s)	ΔVelocity (cm/s)	Time (s)
WCOR	62.9	32.5	1.43	154
COR	60.1	29.5	2.16	172

4.2. Real Robot

The simulation results are a good indicator of the performance of the learned controllers, but these are not completely tested until they are implemented on the real robot. Several test were done in the Department of Electronics and Computer Science of the University of Santiago de Compostela with a Nomad 200 robot. The environment was not modified for the robot, and only one box was placed in one of the passageways just to make the robot turn and reduce the length of the path. Table XI shows the average values of some parameters for WCOR (41 rules) and COR controllers (52 rules) along the same path (Figure 7).

The tested controllers are exactly the same used in the simulated environments. The only difference is that the maximum speed of the robot has been limited from 61 cm/s to 38 cm/s, because at higher speeds the robot has not the capability to stop

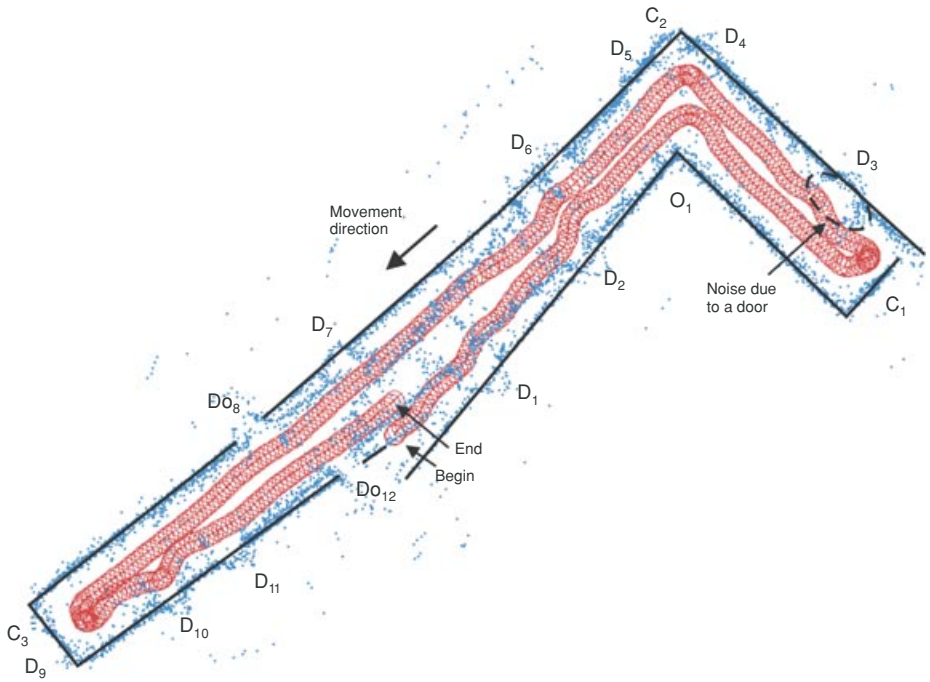


Figure 7. Path of the real robot for WCOR ($\beta = 0.2$).

in a short space (remember that it is following the wall at 51 cm). WCOR obtains an average right distance 4% worse than COR, but improves the average velocity 10%. This means that it arrives to the goal 18 s before the robot controlled with the COR knowledge base. Also the smoothness of WCOR is better, in spite of the lower number of rules. This parameter is specially important on the real robot, as frequent changes in the speed and angle of the robot visually spoil the behavior. Finally, the average right distances are slightly worse than in simulation, because the measurement of the input variables has much more noise on the real robot.

The path followed by the robot has a length of 49 m. Figure 7 shows both the trace of the robot and the measurements of the ultrasound sensors (represented by dots). When the robot moves and turns, small errors appear because the wheels can slip, the turned angle is not exactly the one measured by the motors of the robot, etc. All these kind of errors are known as odometry errors. In short distances, these errors are not significant, but for long distances they become important for mapping and localization tasks. Although they do not affect the performance of the learned behaviors, they get evident when we display the map obtained with the ultrasound sensors. Although, the wall placed on the right of the robot at the beginning is the same wall that is on the right of the robot at the end, in the map they appear displaced around 1 m.

The environment has 3 concave (C_i) and 1 convex corner (O_i) and 12 doors (D_i), 2 of them opened (Do_i). The presence of doors increases drastically the number of noisy measurements. When they are opened the gap is of 75 cm (the minimum gap that the robot needs for crossing a door is 1 m), and the detected obstacles on the other side of the door complicate the estimation of the wall. But also when the doors are closed noise appears, because the surface of the doors is very smooth, and the probability of specular reflection highly increases. These noisy measurements appear in the middle of the passageway, increasing the errors in the detection of the walls and, thus, affecting the performance of the learned controllers.

5. CONCLUSIONS

A methodology for the design of behaviors in mobile robotics has been presented. It is based on COR (cooperative rules) with weights (WCOR) and uses a genetic algorithm to perform the search. Using rule weights improves the accuracy of the knowledge base (the way rules interact), while maintaining a good interpretability.

The methodology presents several advantages over other approaches for the design of behaviors that must be implemented on the real robot. In first place, the designer only needs to define a few parameters. Also, the learned behaviors are general, so the performance and reliability of the controller in different real environments is not affected. Finally, the controllers are learned on the simulated robot, but they can be directly exported to the real robot without any change.

The system has been tested learning the wall-following behavior in mobile robotics using a Nomad 200, showing a good performance in the different environments in which it has been tested, both in simulation and on the real robot. A deep

parametric study has been done to show the independence of the performance of the learned knowledge bases with the values of some selected parameters. Also, the learned controller has been compared with other three controllers,[22,30,31] showing a better balance between interpretability and accuracy.

Acknowledgments

This work was supported in part by the Spanish Ministry of Science and Innovation under grants TIN2008-00040 and TIN2005-08386-C05-01. Manuel Mucientes is supported by the Ramóny Cajal program of the Spanish Ministry of Science and Innovation.

References

1. Dahl T, Giraud-Carrier C. Evolution-inspired incremental development of complex autonomous intelligence. In. Proc 8th Int Conf on Intelligent Autonomous Systems (IAS'04), Amsterdam (The Netherlands), 2004; pp 395–402.
2. Gu D, Hu H, Reynolds J, Tsang E. GA-based learning in behaviour based robotics. In. Proc 2003 IEEE Int Symp on Computational Intelligence in Robotics and Automation, Kobe (Japan), 2003; pp 1521–1526.
3. Hagras H, Callaghan V, Collin M. Learning and adaptation of an intelligent mobile robot navigator operating in unstructured environment based on a novel online fuzzy-genetic system. Fuzzy Sets Syst 2004; 141:107–160.
4. Izumi K, Watanabe K, Jin S-H. Obstacle avoidance of mobile robot using fuzzy behavior-based control with module learning. In. Proc 1999 IEEE/RSJ Inte Conf on Intelligent Robots and Systems 1999; pp 454–459.
5. Katagami D, Yamada S. Interactive classifier system for real robot learning. In: IEEE Int Workshop on Robot-Human Interaction (ROMAN-2000), Osaka (Japan), 2000. pp. 258–263.
6. Oh CK, Barlow GJ. Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming. In: Proc Cong on Evolutionary Computation, Portland, 2004. pp 1538–1545.
7. Yamada S. Evolutionary behavior learning for action-based environment modeling by a mobile robot. Appl Soft Comp 2005; 5(2):245–257.
8. Berlanga A, Sanchis A, Isasi P, Molina JM. A general learning co-evolution method to generalize autonomous robot navigation behavior. In. Proc 2000 Cong on Evolutionary Computation, La Jolla, CA, 2000; pp 769–776.
9. Floreano D, Mondada F. Evolutionary neurocontrollers for autonomous mobile robots. Neural Netw, 1998; 11:1461–1478.
10. Thongchai S, Behavior-based learning fuzzy rules for mobile robots. In: Proc American Control Conference, Anchorage, AK, 2002. pp 995–1000.
11. Zhou C. Robot learning with ga-based fuzzy reinforcement learning agents. Inform Sci 2002; 145:45–68.
12. Fuentes O, Rao R, Van Wie M. Hierarchical learning of reactive behaviors in an autonomous mobile robot. In. IEEE Int Conf on Systems, Man and Cybernetics 1995; pp 4691–4695.
13. Lee KJ, Zhang BT. Learning robot behaviors by evolving genetic programs. In: Proc of the 26th Int Conf on Industrial Electronics, Control and Instrumentation (IECON-2000), 2000. Vol 4, pp 2867–2872.
14. Alcalá R, Casillas J, Cordón O, Herrera F. Improving simple linguistic fuzzy models by means of the weighted COR methodology. In: 8th Ibero-American Conference on Artificial Intelligence (IBERAMIA 2002), Advances in Artificial Intelligence - IBERAMIA 2002.

Sevilla (Spain), 2002. Lecture Notes in Artificial Intelligence, vol. 2527 pp 294–302. Berlin: Springer-Verlag; 2002.
15. Sack D, Burgard W. A comparision of methods for line extraction from range data. In: Proc 5th IFAC Symp on Intelligent Autonomous Vehicles, 2004.
16. Lu F, Milios E. Robot pose estimation in unknown environments by matching 2d range scans. In: Proc IEEE Computer Vision and Patern Recognition Conference, 1994; pp 935–938.
17. Jakobi N, Half-baked, ad-hoc, and noisy: minimal simulations for evolutionary robotics. In: Proceedings of the fourth European Conference on Artificial Life. Boston, MA: MIT Press; 1993.
18. Cho JS, Park DJ. Novel fuzzy logic control based on weighting of partially inconsistent rules using neural network. J Intell Fuzzy Syst 2000; 8:99–110.
19. Pal NR, Pal K. Handling of inconsistent rules with an extended model of fuzzy reasoning. J Intell Fuzzy Syst 1999; 7:55–73.
20. Casillas J, Cordón O, Herrera F. COR: a methodology to improve ad hoc data-driven linguistic rule learning methods by inducing cooperation among rules. IEEE Trans Syst, Man, and Cybern, Part B: Cybernet 2002; 32(4):526–537.
21. Casillas J, Cordón O, Herrera F. COR methodology: a simple way to obtain linguistic fuzzy models with good interpretability and accuracy. In: Casillas J, Cordón O, Herrera F, Magdalena L. editors. Accuracy improvements in linguistic fuzzy modeling. Heidelberg, Germany, Springer; 2003.
22. Wang L-X, Mendel JM. Generating fuzzy rules by learning from examples. IEEE Trans Syst, Man, Cybern 1992; 22(6):1414–1427.
23. Thrift P. Fuzzy logic synthesis with genetic algorithms. In: Belew RK, Booker LB, editors. Proceedings of the 4th International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann; 1991. pp 509–513.
24. Baker JE, Reducing bias and inefficiency in the selection algorithm. In: Proc 2nd Int Conference on Genetic Algorithms, Hillsdale, NJ, 1987. pp 14–21.
25. Michalewicz Z. Genetic algorithms + data structures = evolution programs. Heidelberg, Germany: Springer-Verlag; 1996.
26. Eshelman LJ, Caruana A, Schaffer JD. Biases in the crossover landscape. In: Schaffer JD, editor. Proc 3rd Int Conf on Genetic Algorithms; San Mateo, CA: Morgan Kaufmann; 1989; pp 86–91.
27. Herrera F, Lozano M, Verdegay JL. Tuning fuzzy logic controllers by genetic algorithms. Int J Approx Reason 1995; 12:299–315.
28. Michalewicz Z. Genetic algorithms. Data structures. Evolutionary programs. Berlin: Springer-Verlag; 1992.
29. Herrera F, Lozano M, Sánchez AM. A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int J Intell Syst 2003; 18:309–338.
30. Mucientes M, Casillas J. Obtaining a fuzzy controller with high interpretability in mobile robots navigation. In: Proc IEEE Int Conf on Fuzzy Systems (Fuzz-IEEE 2004), Budapest (Hungary), 2004. pp. 1637–1642.
31. Cordón O, Herrera F. A proposal for improving the accuracy of linguistic modeling. IEEE Trans Fuzzy Syst 2000; 8(3):335–344.