Title	An adaptive fuzzy approach to obstacle avoidance
Author(s)	Yung, NHC; Ye, C
Citation	IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings, San Diego, California, USA, 11-14 October 1998, v. 4, p. 3418-3423
Issued Date	1998
URL	http://hdl.handle.net/10722/46151
Rights	©1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

An Adaptive Fuzzy Approach to Obstacle Avoidance

N. H. C. Yung and C. Ye
Department of Electrical & Electronic Engineering
The University of Hong Kong
Chow Yei Ching Building, Pokfulam Road, Hong Kong SAR

ABSTRACT

Reinforcement learning based on a new training method previously reported guarantees convergence and an almost complete set of rules. However, there are two shortcomings remained: first, the membership functions of the input sensor readings are determined manually and take the same form; and second, there are still a small number of blank rules needed to be manually inserted. To address these two issues, this paper proposes an adaptive fuzzy approach using a supervised learning method based on back propagation to determine the parameters for the membership functions for each sensor reading. By having different input fuzzy sets, each sensor reading contributes differently in avoiding obstacles. Our simulations show that the proposed system converges rapidly to a complete set of rules, and if there are no conflicting input-output data pairs in the training sets, the proposed system performs collision-free obstacle avoidance.

1. INTRODUCTION

The problem of mobile vehicle (MV) navigation particularly through a dynamically changing environment is a challenging one. The case of navigation through a static obstacle course is more manageable than the dynamic case as obstacles and the environment remain time-independent throughout. Many solutions have been proposed in the past with reasonable success, although real environments and situations are hardly that simplistic. The case of navigation through a dynamic obstacle is a lot more demanding in many ways. As the environment may contain moving obstacles or its topological properties may be changing, it has to be considered unknown and time-varying, as in most practical environments. The crux of the problem is how obstacle avoidance is performed.

Apart from those assuming a known environment, many methods have been proposed to tackle the obstacle avoidance problem utilizing on-line information acquired from environment sensors. Of the various schemes proposed, the idea of the reactive system stands out above the rest [1]. Its main focus is to build a mapping from the perceived situations to the correct actions and to refine the mapping until a goal is reached. This approach seems logical in describing the navigation task and much effort has been devoted to using neural network or fuzzy logic to construct the situation-action relationship [2]. Between the two methods for constructing such relationship, fuzzy logic seems promising as it deals with the various navigation situations without requiring the construction of an analytical model of the environment. When compared with the neural network approach, it has an added advantage that each rule of the rule base has an associated physical meaning and deals with a specific situation. This makes it possible to tune the rules by using expert knowledge. However, the way in which the rule base is constructed becomes extremely important. This is because real world environments are inherently complex, and it is rather difficult to formulate a consistent set of rules for tasks such as obstacle avoidance, even if expert knowledge is readily available. For this reason, research effort has been directed to how the rule base can be constructed correctly and efficiently [3]. Overall, many rule bases constructed suffer two limitations: consistency of the rules is hard to maintain and tuning the rules is time consuming.

To tackle these, supervised learning methods using neural networks have been proposed [4]. Unfortunately, these methods require a large set of representative patterns to characterize the environment when training the network, which contain no contradictory input/output pairs. This problem is partially resolved by using reinforcement learning. The reinforcement signal ensures the input/output pairs are consistent, and allows the rules to be tuned to some extent [5]. However, the theoretical limitations of reinforcement learning inevitably result in a slow convergence and incomplete rule base. The issue of learning speed and convergence was dealt with by [6], where a new training method for the reinforcement learning was proposed. This method guarantees convergence and an almost complete set of rules. However, there are two shortcomings remained: first, the membership functions of the input sensor readings are determined manually and take the same form; and second, there are still a small number of blank rules (3 out of 243) needed to be manually inserted. The reasons are first, the fuzzy sets determined manually for the input variables may not be optimal, and second, the sensors should contribute differently when avoiding obstacles. The fuzzy terms of *very near*, *near*, and *far* should ideally be determined through learning rather than manually.

To address these two issues, this paper proposes an adaptive fuzzy approach using a supervised learning method based on back propagation to determine the parameters for the membership functions for each sensor reading. By having different input fuzzy sets, each sensor reading contributes differently in avoiding obstacles. Given a fuzzy logic system $f_k(d)$, our approach determines the parameters, x_{ri} and b_j, such that for a given input-output pair $(\boldsymbol{d}, y_d)$, where y_d is the desired output of the fuzzy system, the actual output of the fuzzy

0-7803-4778-1/98 $10.00 © 1998 IEEE

system y converges to the desired output. The difference between y_a and y_d is used to train the fuzzy logic system, $f_k(d)$. Our simulations show that the proposed system converges rapidly to a complete set of rules, and if there are no conflicting input-output data pairs in the training sets, the proposed system performs collision-free obstacle avoidance.

2. ADAPTIVE FUZZY LOGIC SYSTEM

2.1 General Overview

As an extension to the research described in [6], let us consider an Adaptive Fuzzy Logic System (AFLS) for obstacle avoidance as depicted in Fig. 1. The AFLS consists of two almost identical five-input, single-output fuzzy systems, with desired output (v_d and $\Delta\theta_d$) and feedback. The input variables are the sensor readings, d_i for i=1,.5, while the control variables are v_a and $\Delta\theta_a$ for obstacle avoidance.

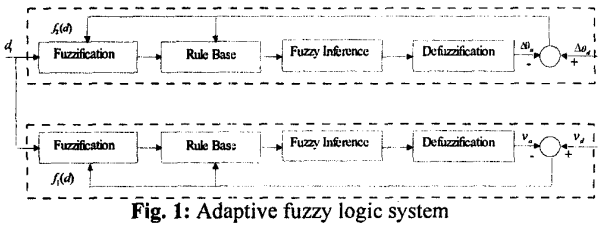
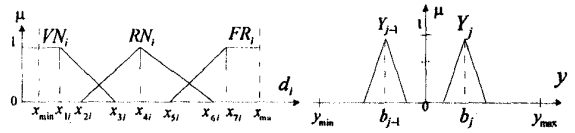


Fig. 1: Adaptive fuzzy logic system

At each learning step, two input-output pairs, (d, v_d) and (d, $\Delta\theta_d$) are provided, where $d=(d_1,\ldots, d_5)$ is the sensor input vector, v_d and $\Delta\theta_d$ are the desired velocity and the desired steering angle for avoidance. For each input vector d, the AFLS determines the output v_a and $\Delta\theta_a$ by $f_1(d)$ and $f_2(d)$ respectively. The difference between v_a and v_d is then used to train the fuzzy logic system adaptively, i.e. to tune its rule base and parameters for the membership functions of d_i, until the system error is below some pre-defined value. As the two fuzzy logic systems take a similar form, our task can be reduced into the design of a fuzzy logic system $y=f(d)$, of which y stands for either v_a or $\Delta\theta_a$ and $f(d)$ is either $f_1(d)$ or $f_2(d)$. Taking the usual approach, the design of $f(d)$ involves six steps: (1) define the membership functions for the sensor input and control output variables, (2) fuzzify the input variables, (3) construct the rule base, (4) carry out optimization through supervised learning, (5) perform fuzzy inference, and (6) defuzzify the output variables to obtain v_a and $\Delta\theta_a$. Details of these steps are given in the following sections.

2.2 Definition of the Membership Functions

The membership functions of the input and output variables are illustrated in Fig. 2. In Fig. 2(a), d_i is the crisp value of each input variable, which is bounded by the minimum value: $d_{min}=R_v+l_{min}$ and the maximum value: $d_{max}=R_v+l_{max}$, where R_v is the radius of the vehicle, l_{min} is the minimum detectable distance by the sensors and l_{max} is the maximum detectable distance. The crisp value of d_i is fuzzified and expressed by the fuzzy sets: VN_i, NR_i and FR_i where they stand for *very near*, *near* and *far*, respectively. The subscript i denotes the fuzzy set for the ith sensor reading. The parameters, x_{ri} for r=1,...,7, are bounded by x_{min} and x_{max}, for $x_{min}= d_{min}$ and $d_{min}<x_{max}\le d_{max}$, and denote the details of the membership functions.



(a) Input (b) Output

Fig. 2: Membership functions of d_i and y

With reference to Fig. 2(b), the output variable y denotes one of the two actions of the vehicle: v_a or $\Delta\theta_a$. Its upper bound and lower bound are y_{max} and y_{min}, respectively. With three linguistic values and five distance sensor inputs, the fuzzy rule base has 243 rules. As a result, it requires 243 fuzzy sets, Y_j for j=1,...,243, to represent the action space y. These fuzzy sets Y_j take the shape of the triangular membership functions of which their centers b_j, are also determined by the learning algorithm. As the parameters, x_{ri} are to be tuned based on the error, e, between v_d and y, the different metric used by x_{ri} and e must be normalized first. In other words, the input d_i and the output y must be normalized to $\bar{d}_i$ and $\bar{y}$ respectively by the following:

$$\bar{d}_i = \frac{d_i - d_{min}}{d_{max} - d_{min}}, \tag{1}$$

$$\bar{y} = \frac{y - y_{min}}{y_{max} - y_{min}}. \tag{2}$$

Similarly, the parameters x_{ri} and b_j are normalized to $\bar{x}_{ri}$ and $\bar{b}_j$, respectively .

2.3 Fuzzification of Normalized Input Variables

The membership functions of the fuzzy sets $\overline{VN}_i$, $\overline{NR}_i$ and $\overline{FR}_i$ can be described by:

$$\mu_{\overline{VN}_i}(\bar{d}_i) = \begin{cases} (\bar{x}_{3i} - \bar{d}_i)/(\bar{x}_{3i} - \bar{x}_{1i}) & \bar{x}_{1i} \le \bar{d}_i \le \bar{x}_{3i} \\ 1 & 0 \le \bar{d}_i < \bar{x}_{1i} \\ 0 & otherwise \end{cases}, \tag{3a}$$

$$\mu_{\overline{NR}_i}(\bar{d}_i) = \begin{cases} (\bar{d}_i - \bar{x}_{2i})/(\bar{x}_{4i} - \bar{x}_{2i}) & \bar{x}_{2i} \le \bar{d}_i \le \bar{x}_{4i} \\ (\bar{x}_{6i} - \bar{d}_i)/(\bar{x}_{6i} - \bar{x}_{4i}) & \bar{x}_{4i} < \bar{d}_i \le \bar{x}_{6i} \\ 0 & otherwise \end{cases}, \tag{3b}$$

$$\mu_{\overline{FR}_i}(\bar{d}_i) = \begin{cases} (\bar{d}_i - \bar{x}_{5i})/(\bar{x}_{7i} - \bar{x}_{5i}) & \bar{x}_{5i} \le \bar{d}_i \le \bar{x}_{7i} \\ 1 & \bar{d}_i > \bar{x}_{7i} \\ 0 & otherwise \end{cases}. \tag{3c}$$

2.4 Rule Base Construction and Fuzzy Reasoning

According to the definition of the fuzzy sets for the input variables, the fuzzy rules are denoted by:

R_j: IF $\bar{d}_1$ is $\overline{D}_{j1}$ AND ... AND $\bar{d}_5$ is $\overline{D}_{j5}$, THEN $\bar{y}$ is $\bar{Y}_j$,

where R_j denotes the jth rule; $\overline{D}_{ji}$ denote the fuzzy sets for $\bar{d}_i$ in the universe of discourse $U_{\bar{d}_i} \subset R$ in the jth rule, which take the linguistic value of $\overline{VN}_i$, $\overline{NR}_i$ or $\overline{FR}_i$, and $\bar{Y}_j$ is the fuzzy set for $\bar{y}$ in the universe of discourse $U_{\bar{y}} \subset R$ in the jth rule. As the jth

rule R_j can be represented as a fuzzy relation, $R_j : \bar{D}_{j1} \times \bar{D}_{j2} \times \bar{D}_{j3} \times \bar{D}_{j4} \times \bar{D}_{j5} \rightarrow \bar{Y}_j$, the rule base can be represented as the union of all the rules:

$$R = \bigcup_{j=1}^{243} [\bar{D}_{j1} \times \bar{D}_{j2} \times \bar{D}_{j3} \times \bar{D}_{j4} \times \bar{D}_{j5} \rightarrow \bar{Y}_j]. \quad (4)$$
$$= \bigcup_{j=1}^{243} R\bar{Y}_j = R\bar{Y}$$

Therefore, the jth rule, $R\bar{Y}_j$ is a fuzzy relation in the product space, $U_{\bar{d}} \times U_{\bar{y}}$, where $U_{\bar{d}} = U_{\bar{d}_1} \times U_{\bar{d}_2} \times U_{\bar{d}_3} \times U_{\bar{d}_4} \times U_{\bar{d}_5}$. Thus, the rules can be implemented as fuzzy relations with their corresponding membership functions. The membership value of the jth rule, $R\bar{Y}_j$ can be denoted by $\mu_{R\bar{Y}_j}(\bar{d}, \bar{y})$. Consider the input $\bar{d}' = (\bar{d}_1', ..., \bar{d}_5')$, which can be treated as fuzzy singleton $\bar{D}' = (\bar{D}_1', ..., \bar{D}_5')$ in the universe of discourse $U_{\bar{d}}$, the fuzzy control action, $\bar{Y}'$ can be inferred by Larsen's product rule of inference as given below:

$$\bar{Y}' = \bar{D}' \bullet \bigcup_{j=1}^{243} R\bar{Y}_j', \quad (5)$$

where $\bullet$ denotes a product operation. If Larsen's product operation is applied to the fuzzy relations, the membership value of $\bar{Y}_j$ is calculated by

$$\mu_{\bar{Y}'}(\bar{y}) = \bigcup_{j=1}^{243} \mu_j(\bar{d}') \cdot \mu_{\bar{Y}_j}(\bar{y}), \quad (6)$$

where $\mu_j(\bar{d}')$ denotes the fire strength of the jth rule for the inputs $\bar{d}' = (\bar{d}_1', ..., \bar{d}_5')$, which is given by

$$\mu_j(\bar{d}') = \mu_{\bar{D}_{j1}}(\bar{d}_1')\mu_{\bar{D}_{j2}}(\bar{d}_2')\mu_{\bar{D}_{j3}}(\bar{d}_3')\mu_{\bar{D}_{j4}}(\bar{d}_4')\mu_{\bar{D}_{j5}}(\bar{d}_5'). \quad (7)$$

It should be noted that the use of Larsen's product rule is to ensure a continuous derivative of $\mu_j(\bar{d}')$ with respective to $\bar{x}_{ri}$.

2.5 Defuzzification of Output Variables

In order to determine the crisp output action, y, from the fuzzy control action, $\bar{Y}'$, first, defuzzification is required, which follows by de-normalization using Eqt. (2). For the reason of limiting the computing cost, the method of height defuzzification is used.

3. SUPERVISED LEARNING

3.1 Learning Algorithm in Delta Rule Form

The output action in a de-normalized form is expressed as:

$$y = f(d') = h / g, \quad (8)$$

where $h = \sum_{j=1}^{j=243} \mu_j(d')b_j$ and $g = \sum_{j=1}^{243} \mu_j(d')$. As y is determined by $\mu_j(d')$ and b_j, Eqt. (8) can be solved by finding a set of x_{ri} and b_j, such that for a given input-output pair (d', y_d), the actual output of the fuzzy logic system y is as close as possible to the desired output, y_d. This optimization is carried out on the mean-square error as defined below:

$$J = \frac{1}{2}(y - y_d)^2, \quad (9)$$

where J is the cost function of the fuzzy logic system. For the given value of y_d, J is a function of x_{ri} and b_j. Therefore, the optimization becomes a search of x_{ri} and b_j on the error surface that give a minimum J in the global sense. To do that, the method of Steepest Descent (SD) is adopted. According to this method, the parameters, x_{ri} and b_j, assume a time-varying form, and move along the error surface with the aim of converging them progressively toward the optimum solution. In principle, such movement is in the direction of the steepest descent on the error surface.

Let $x_{ri}(k)$ and $b_j(k)$ denote the values of the parameters, x_{ri} and b_j at iteration k by the SD method, respectively. In the same way, the gradients of the error surface with respect to x_{ri} and b_j at iteration k are denoted by $\partial J(k)/\partial x_{ri}(k)$ and $\partial J(k)/\partial b_j(k)$ respectively. According to the SD method, the updated values of x_{ri} and b_j at the next iteration $k+1$ is calculated as

$$x_{ri}(k+1) = x_{ri}(k) - \eta \frac{\partial J(k)}{\partial x_{ri}(k)}, \quad (10)$$

$$b_j(k+1) = b_j(k) - \eta \frac{\partial J(k)}{b_j(k)}, \quad (11)$$

where η, $0 < \eta \leq 1$, defines the learning rate and Eqt. (10) & (11) are called the Delta Rule.

As y (hence J) depends on x_{ri} through h and g, while y (hence J) depends on b_j only through h. Therefore, by using the chain rule, the gradients of the error surface with respect to x_{ri} and b_j in general are given below:

$$\frac{\partial J}{\partial x_{ri}} = (y - y_d)\frac{1}{g^2}\left(g\frac{\partial h}{\partial x_{ri}} - h\frac{\partial g}{\partial x_{ri}}\right) = \frac{y - y_d}{g}\left(\sum_{j=1}^{243} b_j \frac{\partial \mu_j}{\partial x_{ri}} - y\sum_{j=1}^{243}\frac{\partial \mu_j}{\partial x_{ri}}\right), \quad (12)$$

$$\frac{\partial J}{\partial b_j} = (y - y_d)\frac{\partial y}{\partial h}\frac{\partial h}{\partial b_j} = \frac{y - y_d}{g}\mu_j. \quad (13)$$

here $\mu_j(d')$ is denoted by μ_j for simplicity. Substituting Eqt. (12) into (10), and Eqt. (13) into (11), we obtain the learning algorithm for x_{ri} and b_j as follows

$$x_{ri}(k+1) = x_{ri}(k) - \eta \frac{y(k) - y_d}{g}\left(\sum_{j=1}^{243} b_j(k)\frac{\partial \mu_j(k)}{\partial x_{ri}(k)} - y(k)\sum_{j=1}^{243}\frac{\partial \mu_j(k)}{\partial x_{ri}(k)}\right), \quad (14)$$

$$b_j(k+1) = b_j(k) - \eta \frac{y(k) - y_d}{g}\mu_j(k), \quad (15)$$

where $y(k)$ and $\mu_j(k)$ denote the system's actual output and the value of μ_j at iteration k, respectively. Similarly, $\partial \mu_j(k)/\partial x_{ri}(k)$ is the value of $\partial \mu_j / \partial x_{ri}$ at iteration k, which is

$$\frac{\partial \mu_j}{\partial x_{ri}} = \begin{cases} \dfrac{\mu_j(d')}{\mu_{D_{ji}}(d_i')} \cdot \dfrac{\partial \mu_{D_{ji}}(d_i')}{\partial x_{ri}} & \mu_{D_{ji}}(d_i') \neq 0, \\ 0 & \mu_{D_{ji}}(d_i') = 0 \end{cases} \quad (16)$$

where $\partial \mu_{D_{ji}}(d_i')/\partial x_{ri}$ can be derived directly from Eqt. (3) as follows:

$$\frac{\partial \mu_{TN_i}(d_i')}{\partial x_{1i}} = \begin{cases} (x_{3i} - d_i')/(x_{3i} - x_{1i})^2 & x_{1i} < d_i' \leq x_{3i} \\ 1/2(x_{3i} - x_{1i}) & d_i' = x_{1i} \\ 0 & \text{otherwise} \end{cases}, \quad (17a)$$

$$\frac{\partial \mu_{VN_i}(d_i')}{\partial x_{3i}} = \begin{cases} (d_i' - x_{1i})/(x_{3i} - x_{1i})^2 & x_{1i} \le d_i' < x_{3i} \\ 1/2(x_{3i} - x_{1i}) & d_i' = x_{3i} \\ 0 & \text{otherwise} \end{cases}, \quad (17b)$$

$$\frac{\partial \mu_{VN_i}(d_i')}{\partial x_{ri}} = 0 \qquad r \ne 1 \text{ and } r \ne 3, \quad (17c)$$

$$\frac{\partial \mu_{NR_i}(d_i')}{\partial x_{4i}} = \begin{cases} (x_{2i} - d_i')/(x_{4i} - x_{1i})^2 & x_{2i} \le d_i' < x_{4i} \\ (x_{6i} - d_i')/(x_{6i} - x_{4i})^2 & x_{4i} < d_i' \le x_{6i} \\ (2x_{4i} - x_{2i} - x_{6i})/2(x_{6i} - x_{4i})(x_{4i} - x_{2i}) & d_i' = x_{4i} \\ 0 & \text{otherwise} \end{cases}, \quad (18a)$$

$$\frac{\partial \mu_{NR_i}(d_i')}{\partial x_{2i}} = \begin{cases} (d_i' - x_{4i})/(x_{4i} - x_{2i})^2 & x_{2i} < d_i' \le x_{4i} \\ 1/2(x_{2i} - x_{4i}) & d_i' = x_{2i} \\ 0 & \text{otherwise} \end{cases}, \quad (18b)$$

$$\frac{\partial \mu_{NR_i}(d_i')}{\partial x_{6i}} = \begin{cases} (d_i' - x_{4i})/(x_{6i} - x_{4i})^2 & x_{4i} \le d_i' < x_{6i} \\ 1/2(x_{6i} - x_{4i}) & d_i' = x_{6i} \\ 0 & \text{otherwise} \end{cases}, \quad (18c)$$

$$\frac{\partial \mu_{NR_i}(d_i')}{\partial x_{ri}} = 0 \qquad r \ne 4 \text{ and } r \ne 2 \text{ and } r \ne 6, \quad (18d)$$

$$\frac{\partial \mu_{FR_i}(d_i')}{\partial x_{7i}} = \begin{cases} (x_{7i} - d_i)/(x_{7i} - x_{5i})^2 & x_{5i} \le d_i' < x_{7i} \\ 1/2(x_{5i} - x_{7i}) & d_i' = x_{7i} \\ 0 & \text{otherwise} \end{cases}, \quad (19a)$$

$$\frac{\partial \mu_{FR_i}(d_i')}{\partial x_{5i}} = \begin{cases} (d_i' - x_{7i})/(x_{7i} - x_{5i})^2 & x_{5i} < d_i' \le x_{7i} \\ 1/2(x_{5i} - x_{7i}) & d_i' = x_{5i} \\ 0 & \text{otherwise} \end{cases}, \quad (19b)$$

$$\frac{\partial \mu_{FR_i}(d_i')}{\partial x_{ri}} = 0 \qquad r \ne 7 \text{ and } r \ne 5. \quad (19c)$$

Therefore, $x_{ri}(k+1)$ and $b_j(k+1)$ can be determined by Eqt. (14) & (15). A network representation of the above process is depicted in Fig. 3.

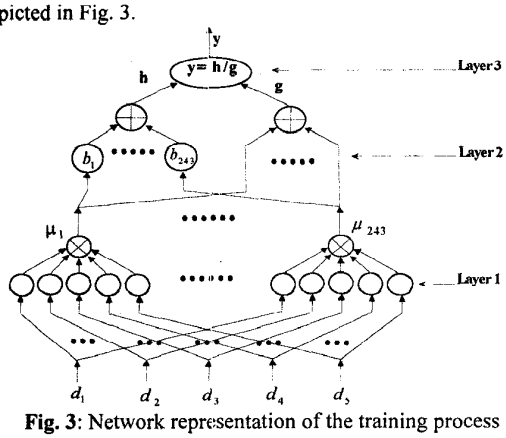


Fig. 3: Network representation of the training process

Forward Procedure

Starting with a sets of normalized initial values, $\bar{x}_{ri}(0)$ and $\bar{b}_j(0)$, at each iteration k, the sensor input d' vector is normalized and encoded into $\bar{\mu}_j(k)$, for j=1,...,243. Then $\bar{\mu}_j(k)$ are fed forward to the network where $\bar{h}$, $\bar{g}$ and the AFLS's actual output $\bar{y}(k)$ are calculated. This is called the forward procedure.

Backward Procedure

Once $\bar{y}(k)$ is calculated, the backward procedure begins. Firstly, to train $\bar{x}_{ri}$, the AFLS's normalized error term, $\bar{y}(k) - \bar{y}_d$ is back-propagated to the processing units of Layer 1 whose output is $\bar{\mu}_j(k)$. Secondly, $\bar{x}_{ri}$ is updated, in which $\bar{b}_j(k)$ and $\bar{y}(k)$ can be obtained from Layer 2 and Layer 3 respectively, while $\partial\bar{u}_j/\partial\bar{x}_{ri}$ can be obtained. Thirdly, to train $\bar{b}_j$, the error, $\bar{y}(k) - \bar{y}_d$ is back-propagated to $\bar{b}_j$ in Layer 2 where it is updated. Usually, a number of these forward-backward cycles may be conducted for an input-output pair before moving onto the next pair. Due to this, $\bar{x}_{ri}$ and $\bar{b}_j$ could become out of bound at some point. To prevent the possible numeric overflow of $\bar{x}_{ri}$ and $\bar{b}_j$, boundary conditions are applied to the training process. These conditions are given below in the usual form:

$$\begin{cases} x_{\min} \le x_{1i} \le x_{1i}(0) \\ x_{7i}(0) \le x_{7i} \le x_{\max} \\ (3x_{1i} + x_{7i})/4 \le x_{4i} \le (3x_{7i} + x_{1i})/4, \\ (x_{1i} + x_{4i})/2 \le x_{3i} \le x_{4i} \\ x_{1i} \le x_{2i} \le (x_{1i} + x_{4i})/2 \\ (x_{4i} + x_{7i})/2 \le x_{6i} \le x_{7i} \\ x_{4i} \le x_{5i} \le (x_{4i} + x_{7i})/2 \\ y_{\min} \le b_j \le y_{\max} \end{cases} \quad (20)$$

In principle, during iterations, these boundary conditions must be met. If any of these conditions are not met, the out-of-bound value is forced to its boundary value.

3.2 Learning Algorithm in General Delta Rule Form

The back-propagation learning algorithm given by Eqt. (14) & (15) provides an approximation to the trajectory in weight space computed by the method of steepest descent. In general, if the learning rate η is small, the change in the weights per iteration is small and the change to the trajectory in weight space is smooth. Such approximation to the trajectory is desired, but is penalized by the long learning process because of the small learning rate. On the other hand, if η is large, the learning speed is improved, but the large change in the weights may likely cause the network to become unstable. As discussed by [7], a simple method of increasing the learning speed and yet avoiding the danger of instability is to modify Eqt. (14) & (15) by including a *momentum* term. The modified equations are given below:

$$x_{ri}(k+1) = x_{ri}(k) + \alpha\Delta x_{ri}(k-1) - \eta\frac{y(k) - y_d}{g}\left(\sum_{j=1}^{243} b_j(k)\frac{\partial\mu_j(k)}{\partial x_{ri}(k)} - y(k)\sum_{j=1}^{243}\frac{\partial\mu_j(k)}{\partial x_{ri}(k)}\right), \quad (21)$$

$$b_j(k+1) = b_j(k) + \alpha\Delta b_j(k-1) - \eta\frac{y(k) - y_d}{g}\mu_j(k), \quad (22)$$

where $\Delta x_{ri}(k-1) = x_{ri}(k) - x_{ri}(k-1)$ and α is positive number called the *momentum* constant. These two equations are called the General Delta Rule (GDR), in which the Delta Rule as in Eqt. (14) & (15) is a special case. Eqt. (21) can be

rewritten as a time series with index t which goes from an initial time 0 to the current time k, as given below:

$$\Delta x_{ri}(t) = x_{ri}(t+1) - x_{ri}(t) = \alpha \Delta x_{ri}(t-1) - \eta \frac{\partial J(t)}{\partial x_{ri}(t)}. \quad (23)$$

Eqt. (28) may be viewed as a 1st-order difference equation on the weight correction $\Delta x_{ri}(k)$. Hence, solving this equation for $\Delta x_{ri}(k)$, we have

$$\Delta x_{ri}(k) = -\eta \sum_{t=0}^{k} \alpha^{k-t} \frac{\partial J(t)}{\partial x_{ri}(t)}. \quad (24)$$

Similarly, Eqt. (22) can be rewritten as

$$\Delta b_j(k) = -\eta \sum_{t=0}^{k} \alpha^{k-t} \frac{\partial J(t)}{\partial b_j(t)}. \quad (25)$$

From these two equations, the following observations can be made: First, the variation in $\Delta x_{ri}(k)$ and ultimately $\Delta y_j(k)$, represents the sum of an exponentially weighted time series. For the time series to converge, the momentum constant must satisfy the condition that $0 \le |\alpha| < 1$. Second, when the partial derivatives, $\partial J(t)/\partial x_{ri}(t)$ and $\partial J(t)/\partial b_j(t)$, have the same algebraic signs between consecutive iterations, the exponentially weighted sums $\Delta x_{ri}(k)$ and $\Delta y_j(k)$ grow in magnitude. Thus, the momentum term tends to accelerate descent in steady downhill directions. Third, when the partial derivatives, $\partial J(t)/\partial x_{ri}(t)$ and $\partial J(t)/\partial b_j(t)$, have opposite signs, the exponentially weighted sums $\Delta x_{ri}(k)$ and $\Delta y_j(k)$ shrink in magnitude. Thus, the inclusion of this momentum term has a stabilizing effect on oscillations in signs. Fourth, the incorporation of the momentum term in the back-propagation algorithm represents a minor modification to the way in which the weights are updated, and yet it can have significant benefits to the learning behavior of the algorithm. It also has the ability of avoiding local minimum on the error surface.

4. SIMULATION AND ANALYSIS

The simulation is based on the *EXPECTATIONS* simulator reported in [8]. Learning is carried out in a restricted area populated with cylindrical static objects, where two additional views are supported. The first allows the human expert to view from the vehicle's when supervising the navigation. The second consists of controls buttons. Using this combination of views and control, the human expert may control the movement of the vehicle while inspecting its location with respect to the obstacles.

At each navigation step, the vehicle's obstacle avoidance actions (v_d and $\Delta\theta_d$) as a result of the manual control of the vehicle by a human expert, and the five distance sensor readings derived from a distance sensor simulator, are composed into two input-output data pairs: (d, v_d) and (d, $\Delta\theta_d$). These input-output pairs are utilized by the ALFA as described in this paper.

Without losing generality, the radius of 23*cm* of a practical mobile robot and its ultrasonic range were used to determine d_{min} and d_{max}, which subsequently determined x_{min} and x_{max}. Moreover, a constant velocity of *30cm/s* is assumed. This simplifies the whole simulation by having to consider the AFLS for $\Delta\theta_d$ only.

The average elapsed time consumed by graphic rendering was about 0.1s when executing on an SGI Indy R5000 with IRIX 6.2 operating system. Therefore, considering a possible on-line AFLS, a navigation step of 0.2s was used in the simulation. Given the maximum rotation velocity of the vehicle of 60 *degree/s*, the minimum and maximum steering in a navigation step are -12° and 12°, respectively. This means that the boundary values of the fuzzy sets for the steering angle (Fig. 2) are y_{min}=-0.2094 and y_{max}=0.2094. The initial value of b_j is set by Eqt. (26)

$$b_j(0) = y_{\min} + \frac{(j-1)(y_{\max} - y_{\min})}{242}, \text{ for } j=1..,243. \quad (26)$$

The initial values of the other parameters used for the simulation are tabulated in Table 1. The convergence criterion for the simulation was $J \le 1.523 \times 10^{-6}$, which is equivalent to the system's output error of 0.1^o.

$R_v = 23$	$x_{1i}(0) = 83$	$x_{2i}(0) = 83$	$x_{3i}(0) = 183$	$x_{4i}(0) = 183$
$x_{5i}(0) = 183$	$x_{6i}(0) = 283$	$x_{7i}(0) = 283$	$x_{min}(0) = 43$	$x_{max}(0) = 323$

Table 1: Initial simulation parameters (unit: *cm*)

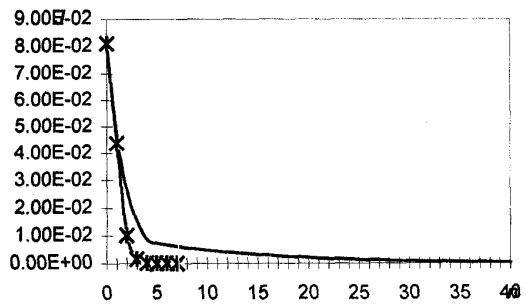
In order to compare the learning performance of the AFLS, the fuzzy logic system without optimization, i.e. only tunes the value of b_j, is considered as a Non-Adaptive Fuzzy Logic System (NFLS). Consider the Delta Rule given by Eqt. (14) & (15), simulation based on the AFLS and the NFLS were carried out at different learning rate η for a input-output pair (d^I, $\Delta\theta_d{}^I$), where d^I=(46.7, 46.7, 159.2, 159.2, 79.2) and $\Delta\theta_d{}^I$= −0.2094, the number of iteration n when the convergence criterion was achieved is tabulated in Table 2.

η	n (AFLS)	n (NFLS)
0.1	40	7313
0.2	26	3655
0.3	17	2436
0.4	12	1827
0.5	9	1461
0.6	7	1217
0.7	6	1043
0.8	5	912
0.9	4	810
0.95	3	768

Table 2: Performance comparison of AFLS and NFLS

From Table 2, a number of points are observed: (1) it is obvious that the ALFS converges exceptionally fast, particularly for large η. The NFLS was at best 140 times slower, and at worst 256 times slower. (2) For this specific input-output pair, both the AFLS and the NFLS converge faster with larger learning rate η. Many other input-output pairs have also been tested and similar results were obtained.

The learning curve (J versus n) of these two systems at the learning rate $\eta=0.6$ is studied in details in Fig. 4. For simplicity, only the first 40 iterations are shown in both cases. As seen in Fig. 4, the AFLS converges after 7 iterations, while the NFLS takes 1217 iterations for convergence. We can also observe that the convergence of the NFLS slows down rapidly after 5 iterations due to the fact that the descent on the error surface becomes very slow and small. As it is, the NFLS is very time consuming compared with the AFLS.



NLFS = Solid line; ALFS = Solid line with ×

Fig. 4: Learning curve of the ALFS and NLFS (η=0.6)

For the Delta Rule (DR) algorithm and the General Delta Rule (GDR) algorithm in order to carry out a smooth search in the weight space, a small learning rate, η=0.2 and a small momentum constant, α=0.2 were used for the comparison. For the same input-output pair (d^l, $\Delta\theta_d{}^l$), where d^l=(46.7, 46.7, 159.2, 159.2, 79.2) and $\Delta\theta_d{}^l$= −0.2094, simulations were conducted to verify each learning algorithm and the results are compared in Fig. 5.

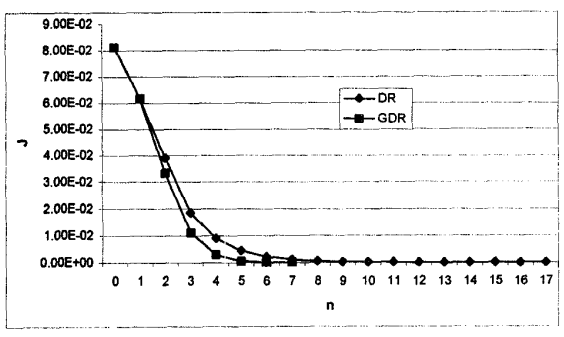


Fig. 5: Learning curves of the three AFLS (η=0.2, α=0.2)

From Fig.5, it can be observed that the DR algorithm has the slower relative convergence speed. It approaches zero at n=8. The GDR algorithm converges faster in that it approaches zero at n=5. For larger η, the convergence rate is expected to increase for both algorithms. Finally, it should also be noted that even for the DR algorithm, its convergence speed is two orders of magnitude faster than the NFLS.

5. DISCUSSION AND CONCLUSION

In this paper, an adaptive fuzzy logic system for obstacle avoidance has been presented. Instead of having the same manually determined input membership for all the inputs, this approach allows the parameters of the input membership functions and the rule base to be tuned, through the minimization of the error between the actual output and the desired output. In the error minimization, the steepest descent method was used to locate the global minimum in the error surface, from which the corresponding input and output parameters define the optimal input/output membership functions. Based on this, two learning algorithms have been considered: the Delta Rule algorithm and the General Delta Rule algorithm. The GDR is the most general form of the back-propagation approach, incorporating the learning rate and the momentum term in its equations. The DR is a special case without taking the momentum term into account. From our simulation trials, it is found that the AFLS using the DR as learning algorithm converges at some two orders of magnitude faster than a non-adaptive fuzzy system, at any learning rate. When comparing the AFLS using DR or GDR, the results indicate that the GDR converges faster than the DR algorithm. In the study presented in this paper, although the input membership functions are optimal, the major uncontrollable factor is the quality of the training data set. Comparing this with the reinforcement learning method discussed in [6], the resulting rule base offers collision-free navigation, but its learning is slow and the membership functions are not optimized.

6. REFERENCES

[1] M. J. Schoppers, "Universal plans for reactive robots in unpredictable environments", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 1039-1046, 1987.

[2] R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, 2, pp.14-23, 1986.

[3] E. Tunstel, "Mobile robot autonomy via hierarchical fuzzy behavior control", *Proc. of 6 International Symposium on robotics and Manufacturing*, 2nd World Automation Congress, Montpellier, France , 1996.

[4] Prabir K. Pal and Asim Kar, "Mobile robot navigation using a neural net", *Proc. of IEEE International Conference on Robotics and Automation*, pp.1503-1508, 1995.

[5] H. R. Beom and H. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy Logic and Reinforcement Learning", *IEEE Trans. Syst. Man Cyber.*, Vol.25, No.3, pp. 464-477, 1995.

[6] N. H. C. Yung and C. Ye, "An Intelligent Mobile Vehicle Navigator based on Fuzzy Logic and Reinforcement Learning", to appear in *IEEE Transactions on Systems, Man and Cybernetics*, 1998.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagation errors", *Nature (London)*, 323, pp. 533-536, 1986.

[8] N. H. C. Yung & C. Ye, "*EXPECTATIONS* – an autonomous mobile vehicle simulator", in Proceedings of *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2290-2295, 1997.