Fuzzy System Implementation through its Approximation with Simplified Radial Basis Networks

Leonid REZNIK
School of Communications & Informatics
Victoria University
P.O. Box 14428, Melbourne City MC VIC 8001 AUSTRALIA
Email: Leon.Reznik@vu.edu.au

Abstract - The paper investigates the method of a fuzzy system design through its approximation with neural networks. It concentrates on further simplification by replacement of a Gaussian radial basis function with its linear and piecewise linear approximation. Different approximating possibilities are tested on four controllers chosen as benchmarks.

I. INTRODUCTION

Various specifications and conditions of employment of intelligent systems diversify design requirements and should be supported by the spectrum of implementation solutions providing their efficient and effective usage. There are two commonly followed ways of a fuzzy system (FS) implementation: realisation on a general purpose processor or on a specialised one. This paper considers both of them as microprocessor implementation in embedded systems. It describes research results on developing a FS implementation on a cheap general microprocessor through its emulation as a neural network (see also [1] for a general problem description). The previous paper [1] investigated a possibility of further simplification of a FS implementation by making the right choice of the NN and its parameters. It considered the choice of the NN type, while [2] concentrated on the vector distance calculation. This paper primarily researches possibility of approximating gaussian function, which is applied in radial basis neural networks (RBNN).

Implementation of the fuzzy control algorithm on the general purpose micro-controller may impose tough restrictions on speed and memory required due to the available resources. Typically an application may not consume the entire code space available, however the speed of the processor cannot be changed without a cost increase. In many cases the replacement of a the control algorithm developed and tested earlier with a simpler one may either significantly improve the real controller characteristics or even expand the range of possible micro-controller applications in soft computing systems without sacrificing the quality. Improvement of the FS realisation efficiency is achieved by training a neural network to emulate FS input-output surface, then implementing the less complex neural network structure on an 8-bit microprocessor, which is cheap and still very popular with the industry. Producing a FS output can require many different equations to be evaluated, subsequently the code can easily become complicated. Unless additional code is written, the designer is usually restricted to only certain types of fuzzy operations. Non-linear transformations can compound the complexity, particularly when dealing with 8-bit micro-controllers that do not directly support floating-point calculations. Commercial products are available to design and compile the necessary pre-fabricated code. The software typically imposes limits on which functions are supported and the number and type of rules allowed. The software does not typically provide optimisation options for speed and memory, making the designer redesign the system with less detail due to resource limitations. Integrated circuits (IC) designed for calculating FSs provide a convenient option for projects requiring fast operation. However, low budget or space conscious projects cannot always afford the luxury of a dedicated IC.

Neural Networks (NN) have been proven to become a suitable structure to replace the function of a static FS to a desired degree of accuracy, which is predominantly controlled by the number of neurons used in the network. NN operate independently on the FS type. Therefore, the software is generic in its use, only the weights need to be changed to implement an alternative surface. The cellular structure of a NN and the simplicity of calculating each neuron's output facilitate an efficient code development and processing. The repetitive loop computations are relatively simplistic to understand, and uncomplicated to debug. The same NN engine can be used to support multiple control surfaces, which may be generated from different network sizes. During development, trading the number of neurons used for accuracy provides design implementation flexibility with respect to both processing speed and program memory.

II. FUZZY SYSTEM PERFORMANCE IMPROVEMENT WITH NEURAL NETWORK APPROXIMATION

Two network structures have been identified as most suited to function approximation, multi-layer perceptron networks (MLPN) and radial basis networks (RBN). A preliminary case study investigated these network types and several prospective variants with full floating point arithmetic. Based on their respective entropy, a single layer perceptron network performed quite well owing to its global approximation characteristic. In fact, several perceptron configurations reduced the median error to less than 0.1%, however the surface detail was less accurate with at least a 7% maximum error. The RBN networks were capable of providing a lower maximum error, due to their localised response characteristic. As the requirement for approximation accuracy increased, the MLPN type networks were unable to maintain the high efficiency, the result of

over-generalisation, whereas the RBNs were capable of providing surface detail.

Each of these NN was investigated to identify those components of the algorithms less suitable to the micro-controller architecture. It was identified that floating point arithmetic required more memory and more processing to compute operations than integer based arithmetic. In addition, multiplications and divisions are time consuming to process, particularly when they involve operands of sizes greater than 8-bits. Combined with the practical inability to process logarithmic or exponential functions, the choice of network types is limited.

RBNs require a radial basis neuron, which includes the use of several multiplications, a square root operation and a Gaussian basis transfer function. The first section of the neuron process finds the vector distance, known as Euclidean Distance (ED), between the input and weight vectors, and the second section transposes the result to represent the degree of equality with the two vectors. The arithmetic and variable sizes required to compute this function are not practical, therefore an alternative was created.

The ED was replaced with an absolute distance function, known as the Manhattan distance (MD). Given the MD is of functional equivalence to the ED function, the Neural Network will accommodate the alternative during training. In general, the difference between the MD and ED functions when used in a Neural Network is negligible, except when the generalisation between neurons is insufficient. This results in very few neighbouring neurons overlapping, reducing the interpolative ability. The use of small networks is, however, limited and their application is only practical when approximating surfaces that are void of detail and quite smooth. Case studies proved that the performance was in some cases better with the simplified function, predominantly due to the higher gradient of the function that allowed finer detail to be represented.

III. WAYS OF GAUSSIAN BASIS FUNCTION APPROXIMATION

Similarly, the Gaussian function used to translate the distance between the inputs and weights to the degree of equality was replaced for a practical substitute. Three, linear piece-wise alternatives which were formulated to approximate the shape of the original Gaussian function were investigated. An investigation of different approximation strategies has been conducted on the base of four FLCs comparison. The first one is used to control motors that mechanically feed banknotes in automatic teller machines [3]. The second is applied for an anti-lock braking system in a vehicle [4]. The third one is used by a robotic arm for force feedback movement [5], while the last one is applied in automatic cruise control [6]. Each of these controllers has been selected to represent the diversity in which they typically operate.

The use of a Gaussian basis function, can only be implemented with either a lookup table, polynomial approximation or by the use of an exponential series calculation. Implementation of the function can be described within limits, which will effect the possible use of different basis function types. Fig. 1 shows the range of the spreading factor from the data collated for the controllers chosen for study in this research. This data represents networks that use between 9 to 49 neurons. The range extends from 0.25 to 6, however as more neurons are used it can be expected that the spreading factor may increase. Most importantly, the lower limit defines the largest spread of the function.

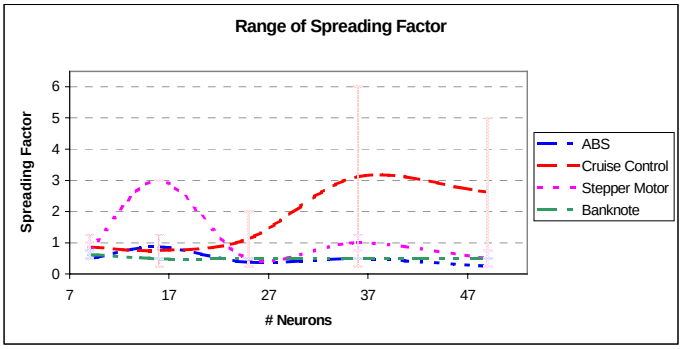

Fig. 1 – Range of the spreading factor for each surface shown with error bars

A lookup table is quite possibly a practical solution, however it is likely it will consume more memory than the neuron weights themselves. The actual range of the values expected to be applied to the basis function is quite dependent on the number of neurons. However, as the function decreases towards zero at the positive and negative extents, the input would need to be close to zero to get an output other than zero.

Realistically speaking, if the spreading factor is limited then it may been beneficial to use a lookup table. As the input is only positive, only one side of the function must be stored and all points beyond the range of the table are assumed as zero. An example of a suitable 64 point radial basis lookup table is defined (1).

$$F_s = \begin{cases} round\left(255 * e^{-(nb)^2}\right) & for\ \ x = \{0,1,2 \ldots 63\} \\ 0 & for\ \ x \geq 64 \end{cases} \qquad (1)$$

If the required level of memory is available, then this method will provide a fast and convenient alternative to using an equation. However, the radial basis function may not always have a range that converges to zero within a practical finite limit. Theoretically there is no limit to applicable range, however practically, the number of neurons and the spreading factor restrict it. Given that the RB networks performance has been optimal around small spreading factors, the range has been on the large side. Subsequently, this is likely to restrict the use of using a lookup table. Ideally with no restrictions, the lookup table would need to reserve approximately 3K of data. Even if a few least significant bits were truncated, the use of the memory is quite

excessive and considered inefficient for the function it provides.

IV. POLYNOMIAL PIECEWISE APPROXIMATION

Calculating the Gaussian function from an exponential series is simply not practical due to the number calculation required, particularly when this calculation must be repeated for each neuron. However a polynomial representation can be used to reduce the computation requirement while providing an accuracy alternative. At least a 5th order polynomial is required to correctly characterize the curve of the bell shaped function. Such a polynomial was generated by finding the coefficients that fit the original curve, in a least-squares sense (2). This equation is quite sensitive to coefficient errors in the high order components.

(2)

$$a = 2.557 * 10^{-9} n^5 - 2.059 * 10^{-6} n^4 + 0.5931 * 10^{-3} n^3 - 0.06595 n^2 + 0.5876 n + 253.2$$

Evaluating this equation requires ten multiplications, a few additions and subtractions. Unfortunately, unless floating point calculations are used this equation becomes slightly more complicated to program. An alternative approach is to use a piece-wise polynomial function, as described in (3).

(3)

$$a = \begin{cases} -0.02573 n^2 - 0.214 n + 255.5 & \text{for } 0 \le n < 60 \\ -37.89 * 10^{-6} n^3 + 0.02513 n^2 - 5.53 n + 404 & \text{for } 60 \le n < 255 \\ 0 & \text{for } n \ge 255 \end{cases}$$

This was generated in the same manner as the previous example, however the first polynomial section requires a 2nd order equation and the next a 3rd order equation. The above linear equation is less sensitive to coefficient errors, however coding this while avoiding floating point calculations is again difficult. It does however reduce the computational requirements by at least two multiplications. Practically, this method is not flexible in that the spreading factor cannot be simply adjusted, the coefficients must be recalculated. Scaling the input value will allow the same equation to be used, however this will require at least one multiplication and unnecessarily add to the complexity of the software.

The purpose of the Gaussian basis function is to define the level of activation for how close the input vector matches the weight vector. It a function cannot be removed because it provides both output scaling and characterizes the extent of local neighbourhood. As long as the basis function satisfies the basic requirements, the network will perform as expected. The difference between using a Gaussian and another alternative is how the interpolation between overlapping neurons interact.

Put simply, a linear triangular function can replace the non-linear Gaussian function. Expectedly, the interpolation between neurons will differ, particularly with a surface that is highly non-linear or using few neurons. However, the simplistic nature of the triangular function, or possibly piece-wise linear functions, can reduce the resources required considerably. Not only is this simplification important to satisfy software implementation, but this approach has also been used for VLSI hardware optimisation. replaces the standard Gaussian basis function with an alternative piece-wise linear one.

It is important that the distinction between perceptron networks and RB networks is made regarding the use of non-linear functions. Perceptron networks need a non-linear transfer function for the backpropagation training algorithm to work. It has also previously been established that such networks using only linear transfer functions cannot approximate non-linear surfaces. So far the vector distance of the RB network has replaced with a linear alternative and now it's proposed that the basis function be also linearised. This is possible, because unlike the perceptron network, the RB basis function if network is locally trained. Simply speaking it can be as an effective lookup table, which can represent non-linear functions.

A total of three linear and piece-wise linear basis will be tested and compared with the standard Gaussian function. The specific shape of each function will be defined to closely match the shape of the Gaussian function in order to provide a direct evaluation. Using slightly different shapes may improve the performance, however the purpose of the comparison is to identify if the proposed linear type functions significantly effect the networks approximation capability.

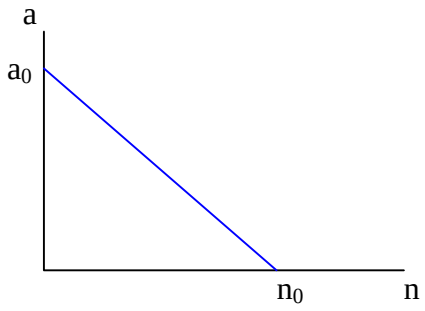

Fig. 2 Triangular basis function.

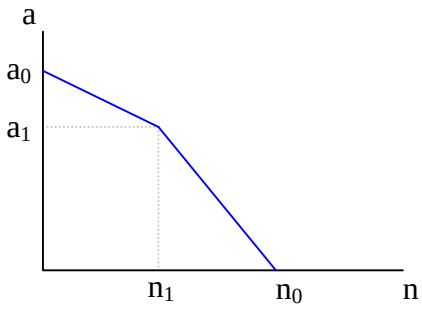

Fig. 3 – Two-piece linear basis function.

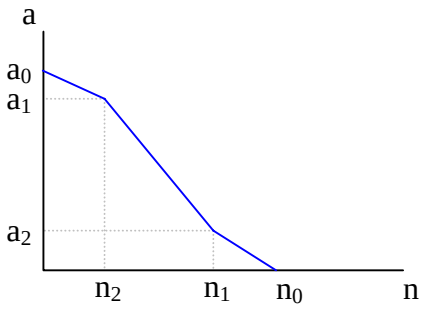

Fig. 4 – Three-piece basis function

The first basis function, termed triangular, is a simple line limited to +ve values only (4), Fig.2. Implementation of this function is relatively straight forward with at most, one multiply, divide and subtraction involved. If the parameters were carefully selected, the divide may be eliminated or restricted to powers of two to facilitate quick division by two rotate instructions.

$$(4)\quad a^{Triangular}(n)=\begin{cases} a_0-\dfrac{a_0 n}{n_0} & 0\le n<n_0 \\ 0 & n\ge 0 \end{cases}$$

where for an approximation of the Gaussian basis function, the parameters are:

$$a_0=1$$

$$n_0=\frac{\sqrt{-\log_e 0.05}}{SF}$$

Extending this to a closer original approximation of the Gaussian function, a two-piece linear basis function is proposed (5). This type has the ability to produce a broader neighborhood close to the center, rather than a sharp point produced by a triangular type. The maximum number of multiplies and divides is not far different from the triangular basis function, as the parameters are constant and can be reduced to single coefficients. It is more likely that these parameter cannot be simplified as easy as the previous case, but in addition, there is the extra overhead of range testing (6).

$$(5)\quad a^{two-piece}(n)=\begin{cases} a_0-\dfrac{n(a_0-a_1)}{n_1} & 0\le n<n_1 \\ \dfrac{a_1(n_0-n)}{n_0-n_1} & n1\le n<n0 \\ 0 & n\ge 0 \end{cases}$$

where for an approximation of the Gaussian basis function, the parameters are:

$$a_0=1;\ n_0=\frac{\sqrt{-\log_e 0.05}}{SF};\ a_1=a_0-\frac{a_0}{8};\ n_1=\frac{n_0}{4}$$

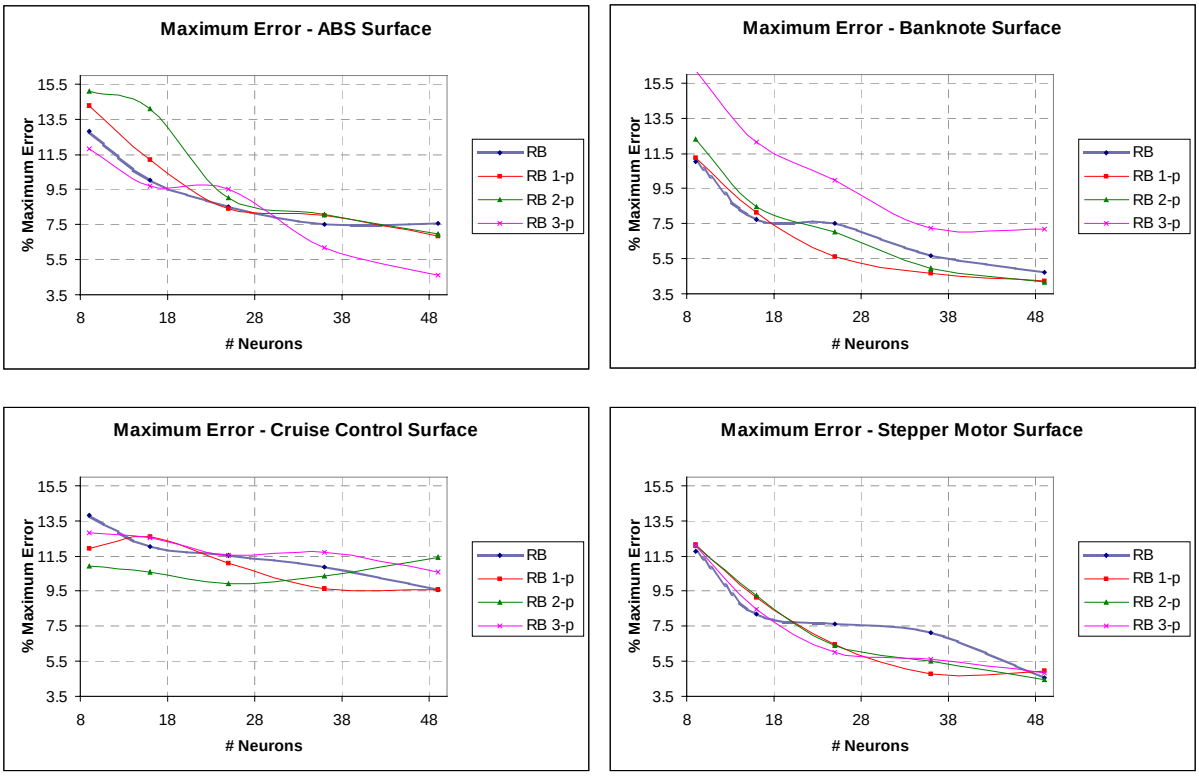

Fig. 5 – Maximum error comparison between using the Gaussian and alternative basis functions.

(6)

$$a^{three-piece}(n)=\begin{cases} a_0-\dfrac{n(a_0-a_1)}{n_2} & 0\le n<n_2 \\ \dfrac{(n-n_1)(a_1-a_2)}{n_2-n_1}+a_2 & n_2\le n<n_1 \\ \dfrac{a_2(n_0-n)}{n_0-n_1} & n_1\le n<n_0 \\ 0 & n\ge n_0 \end{cases}$$

where for an approximation of the Gaussian basis function, the parameters are:

$a_0=1$; $n_0=\dfrac{\sqrt{-\log_e 0.05}}{SF}$; $a_1=a_0-\dfrac{a_0}{8}$; $n_1=\dfrac{n_0}{\sqrt{2}}$; $a_2=\dfrac{3a_0}{16}$; $n_2=\dfrac{n_0}{4}$

V. APPROXIMATION RESULTS

The calculation of the parameters will provide an equivalent approximation to the Gaussian basis function with respect to the spreading factor. Eighty simulations have been performed to identify the difference between the proposed basis functions. These results were generated using the Manhattan distance function, as detailed in [2]. Using a spreading factor of 0.5, the maximum and median error were calculated for each of the four example fuzzy surfaces with 9 to 49 neurons in the hidden layer. Fig. 5 shows the maximum error for all fuzzy surfaces. On initial inspection, the results may appear to be somewhat inconclusive as to which basis function performs the best.

On close inspection, the 2-piece and in particular the 3-piece function are less consistent with a wide variance. Clearly in the banknote surface the 3-piece produces the worst approximation, while in the ABS surface it is the best option. The triangular 1-piece basis function is actually the most consistent and reliable, while also providing the lowest result 36% of the time. The original Gaussian basis function was only the lowest 18% of the time, which occurred only when few neurons were present.

Fig. 6 shows the median error, which is quite consistent and with little variance. Based on these results, neither basis function is clearly better than another, as the differences in error are relatively small. For the ABS surface, which is quite smooth, the 3-piece basis function has more difficulty than the others in producing a low median error, which is surprising given the maximum error was quite low. The triangular function did not perform as well on the cruise control surface, which can be attributed to the region with the high gradient, as previously identified. This can be expected because this function has the lowest gradient, however a steeper function could improve the response.

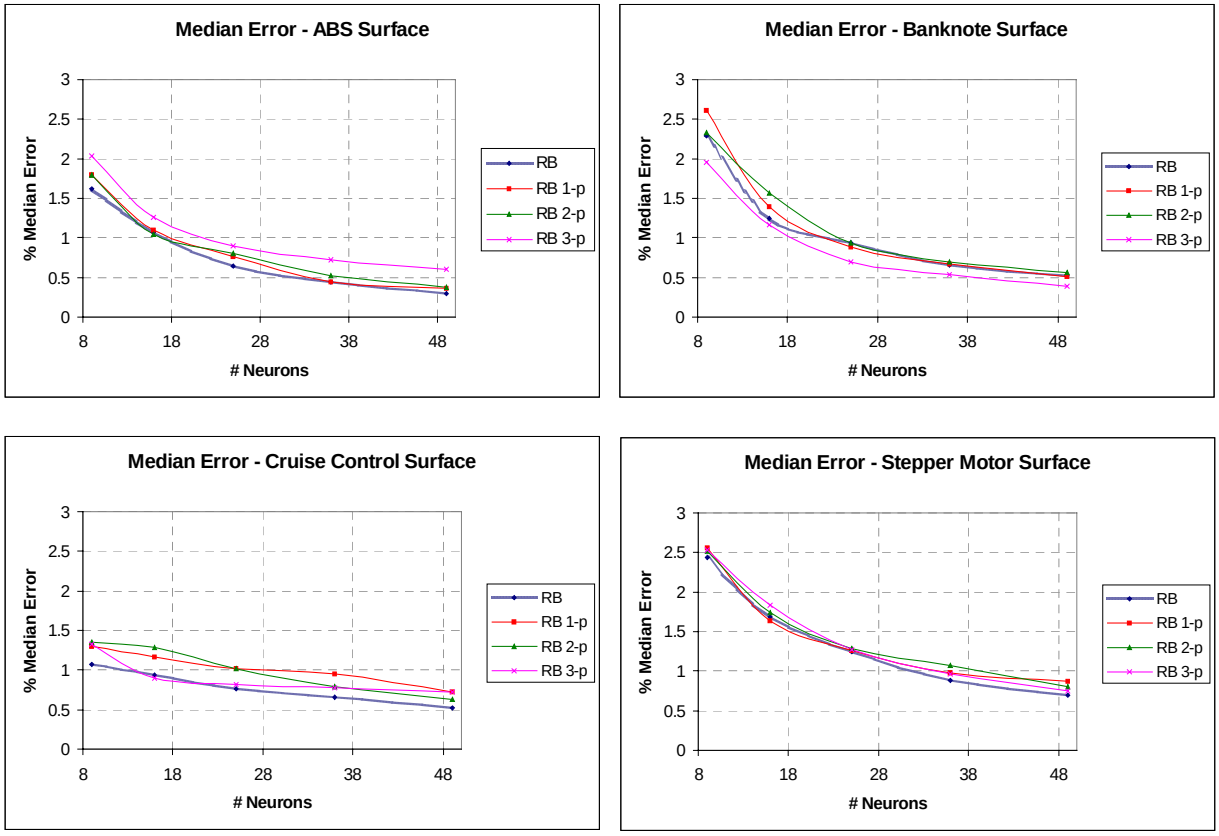

Fig. 6– Median error comparison between using the Gaussian and alternative basis functions

VI. CONCLUSION

This analysis has identified that the Gaussian basis function can be replaced by a linear or piece-wise linear function without a significant change in error. It has also shown that the alternatives are more likely to produce a better result than that of the original, unless the number of neurons is less than nine. In particular, the triangular function has shown to be the most consistent and reliable with consideration to both the maximum and median error results. Additionally, this function is the simplest and most efficient basis function tested. Therefore, the triangular basis function has been selected for the subsequent implementation of the RB networks.

Because the linear equation is quite simple and not resource demanding, a lookup table is not required. This eliminates any problems associated with input scaling to the basis function and the range the SF may extend, such as found when trying to implement the Gaussian basis function. It also does not require any polynomial approximation, which tends to complicate the design and implementation of the neural network software.

References

[1] Little A., and L. Reznik Implementation of Fuzzy Controllers with Radial Basis Neural Networks The Ninth IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2000 San Antonio, USA, 7-10 May 2000 Proceedings IEEE vol.2 p.581 – 586

[2] Little A. and L.Reznik Improving the Approximation Smoothness of Radial Basis Neural Networks, Journal of Advanced Computational Intelligence, vol. 4, No. 6, 2000, pp.1-4

[3] Sato M., Kitagawa T., Sekiguchi T., Watanabe K., Goto M. Fuzzy Logic Based Banknote Transfer Control, IEEE 1993, published in "Fuzzy Logic Technology and Applications", IEEE Update series 1994

[4] Madau D.P., Yuan F., Davis L.I.,Jr., Feldkamp L.A. Fuzzy Logic Anti-Lock Brake System for a Limited Range Coefficient of Friction Surface, IEEE 1993, published in "Fuzzy Logic Technology and Applications", IEEE Update series 1994

[5] Hollinger J.G., R.A. Bergstrom, J.S. Bay A Fuzzy Logic Force Controller for a Stepper Motor Robot, IEEE 1993, published in "Fuzzy Logic Technology and Applications", IEEE Update series 1994

[6] Muller R. and G. Nocker Intelligent Cruise Control with Fuzzy Logic IEEE 1993, published in "Fuzzy Logic Technology and Applications", IEEE Update series 1994