Microprocessor Implementation of Fuzzy Systems and Neural Networks

Jeremy Binfet
Micron Technology
jbinfet@micron.com

Bogdan M. Wilamowski
University of Idaho
wilam@ieee.org

Abstract

Systems were implemented on8-bit Motorola 68HC711E9 microcontroller. The on-board features of the HC711 are 512 bytes of RAM and EEPROM and 12K bytes of UV erasable EPROM. The processor was used with an 8 MHz crystal, allowing an internal clock frequency of 2 MHz. ICC11 for Windows V5 was the compiler used to program the HC711E9. In the case of fuzzy systems three different membership functions were used: trapezoidal, triangular, and Gaussian and two different defuzzification processes: Zadeh and Tagagi-Sugeno. In the case of neural networks all architectures were developed and optimized with a help of SNNS. Both, layered and fully connected structures were investigated. In the case of neural controllers implemented on a microprocessor the code is simpler, much shorter; the processing time is comparable with fuzzy controllers. Control surfaces obtained from neural controllers also do not exhibit the roughness of fuzzy controllers

1 Introduction

Significant amount of research has been devoted in the development of fuzzy controllers [1][2][3][4]. In hardware, fuzzy systems dominate current trends in both microprocessor applications [5] and in custom designed VLSI chips [6]. Fuzzy controllers are especially useful for nonlinear systems. Since membership functions and fuzzy rules are chosen arbitrarily and therefore, fuzzy controllers are often good, but not optimal. Control surfaces obtained from fuzzy controllers are rough, which can cause unstable control. On other hand neural networks usually require a computation of tangent hyperbolic activation functions. This tasks it often too complex for simple microprocessors. Even though neural networks are primarily implemented in software, their good approximation properties make them an attractive alternative in hardware [7][8]. Microprocessor realization can be easily achieved by using special activation functions such as the Elliot that are easy to compute, which allows fast execution time. Devices such as analog to digital and digital to analog converters must be also be used, but since they are asynchronous devices, there is only a slight additional penalty.

In presented approach tangent hyperbolic functions were replaced by Elliot function and neural controllers were implemented on simple HC11 Motorola microprocessors. With proposed approach neural network implementations resulted with shorter code, faster operation, and much more accurate results. The purpose of this document is to compare several controllers for the same desired control surface implemented in the popular HC11 micro-controller using various fuzzy and neural network architectures.

2 Fuzzy controllers

Fuzzy controllers are used to provide solutions to control problems that cannot be described by complex mathematical models. They are relatively easy to design and produce reasonable results. It is the simplicity that makes them more attractive than neural controllers. With unlimited resources such as memory or chip space, fuzzy controllers can handle problems having multiple inputs and outputs. However, in most cases, resources are limited, which causes two problems. The first is that for numerous inputs, each one has to have a small number of membership functions. This is because the fuzzy table grows exponentially with each input added. The second problem is that small membership functions yield very poor results. Although fuzzy controllers can theoretically have multiple inputs and outputs, when they are implemented in hardware, a decision has to be made on which parameter is more important. That is, low error with few inputs or high error with many inputs.

The basic fuzzy principle is similar to that of Boolean logic except that there are more than two states because fuzzy variables can have any value from 0 to 1. So for fuzzy logic, the min, max and not operations are used. Instead of the AND function, min or $\wedge$ is used while the max or $\vee$ is used for the OR operations. For NOT operations, the value is subtracted from one to produce the inverse. This means that the NOT of 0.4 becomes 0.6. Examples of each function are below:

$$A \wedge B \wedge C \Rightarrow \min\{A,B,C\} - \text{smallest value of } A, B \text{ or } C$$

$$A \vee B \vee C \Rightarrow \max\{A,B,C\} - \text{largest value of } A, B \text{ or } C$$

$$\overline{A} \quad \Rightarrow 1 - A \quad - \text{one minus A}$$

These rules are also know as Zadeh AND, OR and NOT operators [22].

Fuzzy controllers use several conversion processes before the final output is produced. First, the analog inputs are converted into fuzzy variables. Usually 3 to 9 variables are generated for each input. Next, the fuzzy rules are applied, which produces the output fuzzy variables. Finally in the last step, the fuzzy variables are converted back into analog values. There are two types of fuzzy controllers commonly used today, Zadeh [9] and

Tagagi-Sugeno [10]. Block diagrams of each are below in Figures 1 and 2:

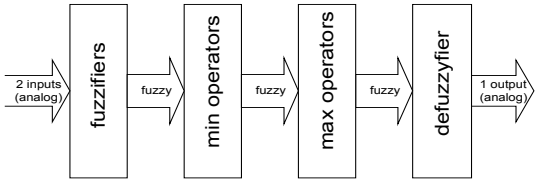

Fig 1. Block diagram for Zadeh fuzzy controller.

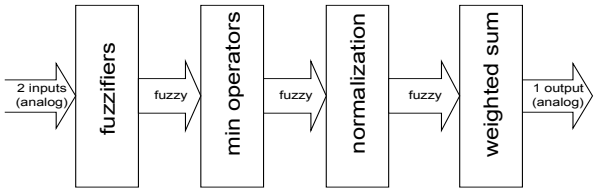

Fig 2. Block diagram for Tagagi-Sugeno.

The first step in both methods is fuzzifying the inputs. There are three rules that need to be applied when designing the membership function.

1. Each point of an input should belong to at least one of the fuzzy membership functions.
2. The sum of two overlapping functions should never exceed one.
3. For higher accuracy more membership functions should be used. However, this can cause problems in the operation of the controller and also makes the fuzzy table larger.

The next step is creating the fuzzy table. A fuzzy table is, in a sense, a grid mapping of the control surface. For Zadeh controllers, the table depends on the output membership function. For the Tagagi-Sugeno method, actual output values are used. Finally, the result can be defuzzified using equation 1.

$$\text{Output} = \frac{\sum_{k=1}^{n} z_k z_{ck}}{\sum_{k=1}^{n} z_k} \qquad (1)$$

where:
n = Number of membership functions
z_k = Fuzzy output variables
z_{ck} = Analog values from table

3. Neural Networks

The basic element of a network is the neuron, which is simple structure consisting of inputs and outputs. Its operation consists of summing the inputs into a net value and then processing the net value through its activation function, producing a final output. A neuron can supply more than one output, but each output will be the same. The activation function is what allows the network to function and is also responsible for the smooth control characteristics. There are many different types of activation functions that can be used, but some of the more common are Sigmoidal, Tangent Hyperbolic, Linear or Identity and the Elliot.

Networks are comprised of neurons connected in a specific fashion. There is a weight or gain associated with each connection and there is a constant bias or gain coupled with each neuron. The two different types of networks are cascade and MLP (multi layer perceptron). Cascade networks have one neuron in each layer arranged in an array structure. MLP networks are similar with the exception that there can be any number of neurons in each layer. There are also two types of network connections, fully connected and feed forward. In a fully connected network, input connections exist from all neurons in previous layers and also the inputs of the network. Feed forward designs only have connections between layers. Examples of each type of network can be seen in Figures 3 and 4.

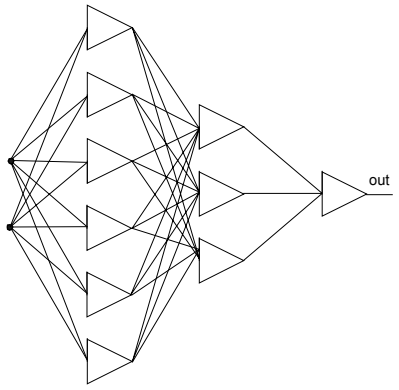

Fig. 3. MLP network with feed forward connection.

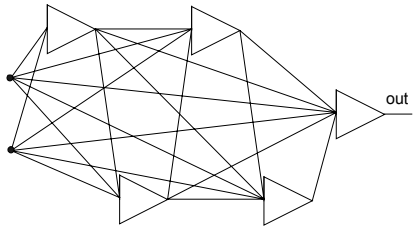

Fig. 4. Cascade network fully connected.

There are advantages to using each type of connection. Feed forward connected networks are simple in nature, which make them quite easy to understand and debug. This stems from the fact that the network can be divided into layers, allowing one layer can be analyzed at a time. On the down side, feed forward coupled networks lack the computational power that fully connected networks have. This requires larger networks to produce the same result. Thus, making fully connected networks a better choice.

It is advantageous to use a cascade network for several reasons. The first is that a cascade network can produce better results than a MLP network using fewer neurons. This leads to the next advantage, which is that the overall size of the networks will be smaller with less weights and connections to deal with. Therefore, cascade networks are easier to implement in hardware, because

they require fewer transistors and connections, which makes the circuit easier to create and debug.

Training is the most essential part of constructing a neural network. It is what determines all the weights and biases of the network so that it can properly function. To train a network, a list file has to be generated that has the inputs matched with their corresponding output(s), (which is then used with a training algorithm. The most common type of training is error back propagation. A block diagram of this process is shown in Figure 5.

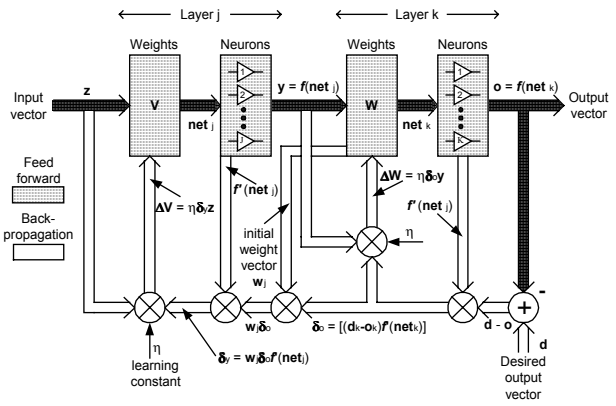

Fig. 5. Block diagram of Error Back Propagation training algorithm.

This type of algorithm uses single pattern vector, which is one set of inputs and output(s), in each cycle. First, the inputs are propagated through the network using initial weights and biases. At the same time, the output values of all the neurons in each hidden layer are stored in a vector, **y**, to be used later in the training algorithm. Next, the training error and the local error are computed. Equation (2) describes the training error where $\boldsymbol{d}$ is the desired output, $\boldsymbol{o}$ is the actual output and **E** is the error. Initially the error is zero.

$$\boldsymbol{E} \leftarrow \boldsymbol{E} + \frac{1}{2}\|\boldsymbol{d} - \boldsymbol{o}\| \tag{2}$$

Equation 3 gives the local error $\boldsymbol{e}$.

$$\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{o} \tag{3}$$

Now, the error signal vectors, $\boldsymbol{\delta}_o$ and $\boldsymbol{\delta}_y$, for the output layer and the hidden layers respectively, must be calculated using the activation function derivative as shown in Equations (4) and (5).

$$\boldsymbol{\delta}_o = [(\boldsymbol{d} - \boldsymbol{o})\boldsymbol{f}'(\boldsymbol{net}_k)] \tag{4}$$

$$\boldsymbol{\delta}_y = \boldsymbol{w}_j \boldsymbol{\delta}_o \boldsymbol{f}'(\boldsymbol{net}_j) \tag{5}$$

Note that the weight vector ($\boldsymbol{w}_j$) used in Equation (5) is the vector that was used in the feed forward phase. The next step is updating the weights for the output layer. First, the change in weights is calculated using Equation (6) where η is the learning constant and then the weights are updated using Equation (7).

$$\Delta \boldsymbol{W} = \eta \boldsymbol{\delta}_o \boldsymbol{f}'(\boldsymbol{net}_k) \tag{6}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W} \tag{7}$$

The hidden layer weights are updated in the same fashion except the error signal vector of the hidden layer ($\boldsymbol{\delta}_y$) is used as in Equations (8) and (9).

$$\Delta \boldsymbol{V} = \eta \boldsymbol{\delta}_y \boldsymbol{f}'(\boldsymbol{net}_j) \tag{8}$$

$$\boldsymbol{V} \leftarrow \boldsymbol{V} + \Delta \boldsymbol{V} \tag{9}$$

After all the weights have been updated, the process repeats for each pattern in the list file. When all of the training patterns have been used, the training error **E** is compared to a goal error GE. If **E** > GE then the whole cycle will be repeated until the desired error is reached. For some networks it is impossible for the goal error to be reached, so the network must be discarded and a different network must then be trained. It runs on both UNIX and Windows platforms. This software is ideal for training networks that use common activation functions. Once a pattern file has been loaded, networks can be easily altered until the desired result is reached.

4 Microprocessor Implementation of Fuzzy Systems

Designing a microprocessor fuzzy controller is relatively simple and straightforward. First, the input and output membership functions and the fuzzy table have to be created as for any fuzzy controller. Next, code has to be generated which will describe the processes of the controller. The code then has to be compiled and downloaded into the microprocessor. Finally, the analog to digital and digital to analog converters have to be wired to the processor.

For microprocessor implementation trapezoidal membership functions are primarily used. In order to store this function, only four bytes are required x1, x2, x3, and x4 (see Fig. 6). The triangular membership function is a special case of trapezoidal where x2=x3.

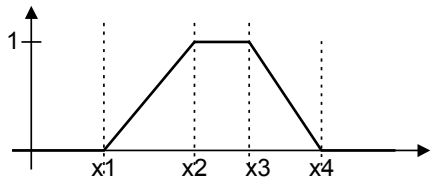

Fig. 6. Representation of the membership function in microprocessor.

For all controllers, the same rule table for each method was used for the different types of membership functions. That is, all the Zadeh controllers used the same table and all the Tagagi-Sugeno controllers used the same table. The first two examples used the Zadeh [9] approach and for the following two examples, the Tagagi-Sugeno [10] approach was implemented. All controllers were designed to emulate the control surface shown in Fig. 7. Two different membership functions were used: trapezoidal (Fig. 8 and 10), triangular (Fig. 9 and 11). Error comparisons are shown in Table 1.

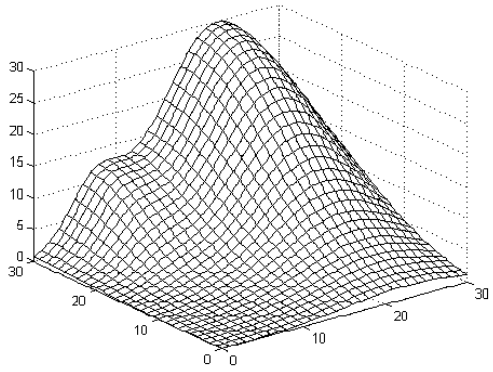

Fig. 7. Required control surface.

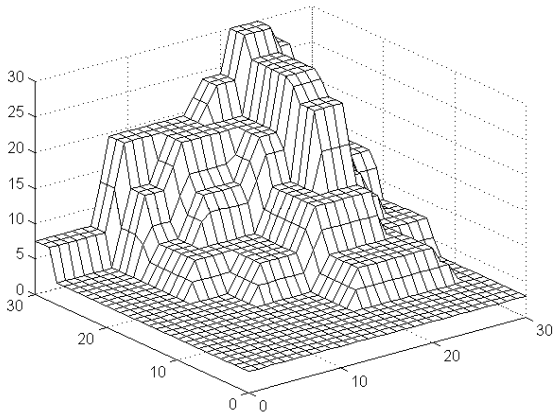

Fig. 8. Control surface obtained with trapezoidal membership functions and Zadeh approach.

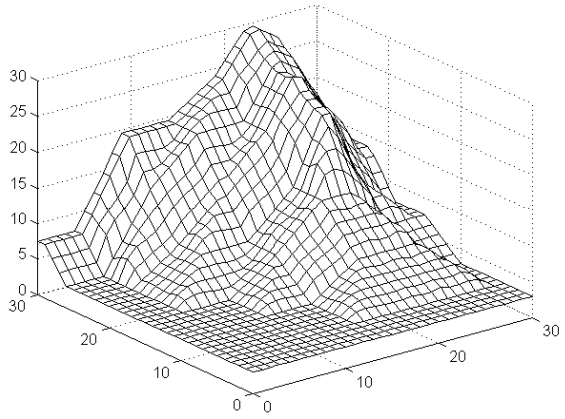

Fig. 9. Control surface obtained with triangular membership functions and Zadeh approach.

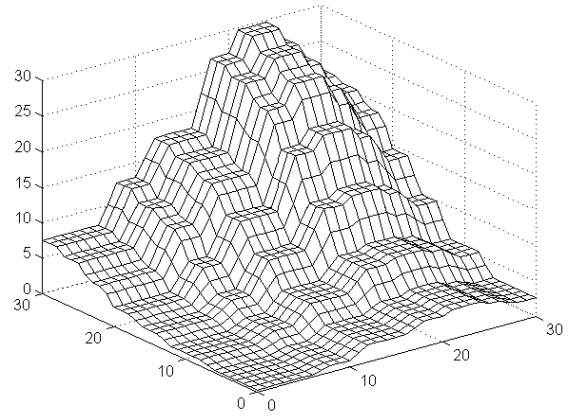

Fig. 10. Control surface obtained with trapezoidal membership functions and Tagagi-Sugeno approach.

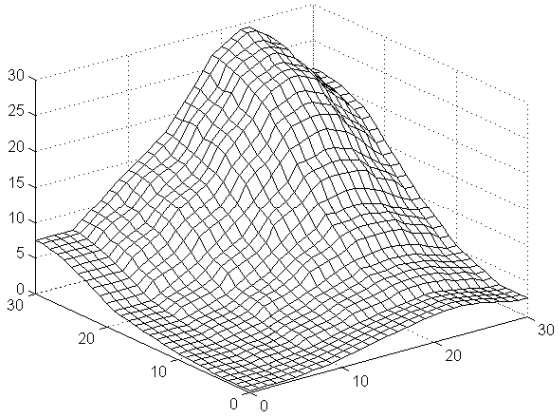

Fig. 11. Control surface obtained with triangular membership functions and Tagagi-Sugeno approach.

Table 1. Error comparison for various types of fuzzy controllers

	Type of controller 7 membership functions for each input and 7 for output	length of code in bytes	processing time (ms)	Error (SSE)
1	Zadeh fuzzy controller with trapezoidal membership function	5009	1.95	908.4
2	Zadeh fuzzy controller with triangular membership function	4140	1.95	644.4
3	Tagagi-Sugeno fuzzy controller with trapezoidal membership function	2511	28.5	296.5
4	Tagagi-Sugeno fuzzy controller with triangular membership function	1642	28.5	210.8

From the table it can be seen that the Tagagi-Sugeno approach is far superior over the Zadeh one. The Tagagi-Sugeno method produces lower errors and uses less memory. The sum of the squared errors, SSE, is calculated by squaring and then summing the differences between the desired and actual outputs. However, the Tagagi-Sugeno algorithm has a noticeably larger execution time. It is also surprising that the surface obtained from the Tagagi-Sugeno method using the triangular membership function produced the best result.

All fuzzy controllers were designed to be implemented on Motorola's 68HC711E9 micro-controller. This is a low cost, 8-bit microprocessor. The on-board features of the HC711 are 512 bytes of RAM and EEPROM and 12K bytes of UV erasable EPROM. The processor was used with an 8 MHz crystal, allowing an internal clock frequency of 2 MHz.

Instead of programming the EPROM for each controller, an emulator was used. The emulator was used because it has 12K bytes of RAM instead of EPROM that can be easily downloaded to. Also, the emulator runs a version of Buffalo Bug with out loosing any of the memory space. Buffalo Bug is an onbaord assembler that also allows a serial interface with a PC using a RS232 chip. Thus, by using the emulator, code can be changed, downloaded and tested very quickly and easily. The program ICC11 for Windows was used because it is capable of converting C or assembly code into the *.S19 and this software also has a terminal window for interfacing with the emulator.

5 Microprocessor Implementation of Neural Networks

Neural network implementations usually require computation of the sigmoidal function [1][7][8][12]

$$f(net) = \frac{1}{1+\exp(-net)} \tag{11}$$

for unipolar neurons, or

$$f(net) = \tanh(net) = \frac{2}{1-\exp(-2net)} - 1 \tag{12}$$

for bipolar neurons. These functions are relatively difficult to compute, making implementation on a microprocessor difficult. If the Elliott function is used:

$$f(net) = \frac{net}{1+|net|} \tag{13}$$

instead of the sigmoidal, then the computations are relatively simple and the results are almost as good as in the case of sigmoidal function.

Neural controllers were implemented on Motorola's 68HC711E9 micro controller with the code written in C language. A block diagram of a neural controller is shown in Fig. 12.

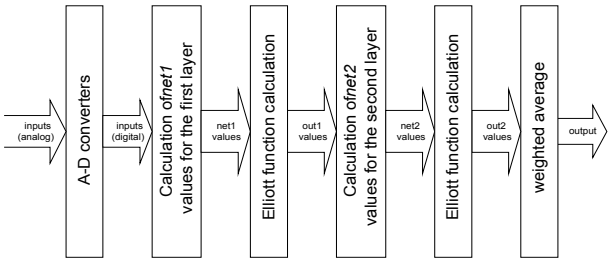

Fig. 12. Block diagram of a neural controller implemented on a microprocessor.

During the design process of a fuzzy controller, the designer must know what output should be expected for given input values. More precisely, what the output value is for a given combination of input membership functions. The exact same information can be used to train the neural network. This of course, must be done using a specially written program or by using ready software. For microprocessor implementation, the Stuttgart Neural Network Simulator (SNNS) [11] was used, since the Elliott activation function is implemented in the program. Many network configurations were tested. The goal was to keep the network as simple as possible while achieving the lowest possible error. Different types of networks that were tested include a) multiple neurons in one hidden layer (MLP), b) multiple neurons in cascade and c) multiple neurons in multiple hidden layers. RProp was the training algorithm used to train the networks. It proved to have the fastest convergence time and provided the lowest errors. Figs. 13, 14, and 15 show obtained control surfaces for several architectures and Table 2 shows error comparison.

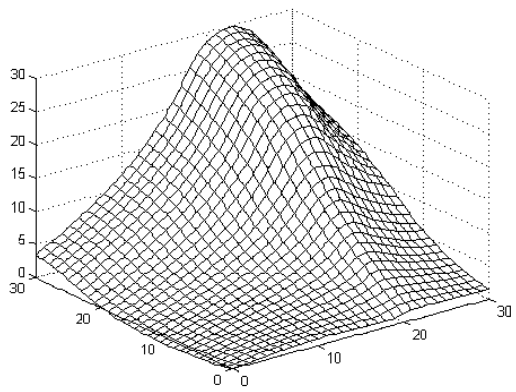

Fig. 13 Control surfaces of neural controller with two layer fully connected 2-1-1-1-1 architecture (4 neurons).

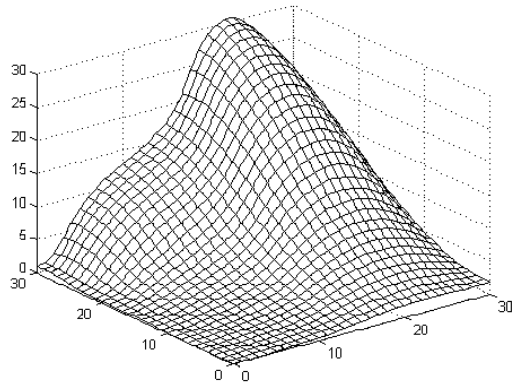

Fig. 14 Control surfaces of neural controller with two layer fully connected 2-1-1-1-1-1-1 architecture (6 neurons).

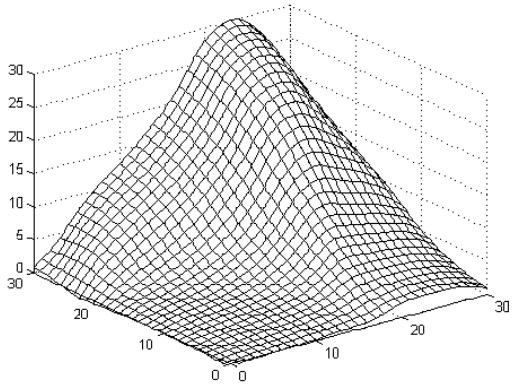

Fig. 15 Control surfaces of neural controller with two layer feed froward 2-6-1 architecture (7 neurons).

Table 2. Error comparison for various type of neural controllers

	Type of controller	length of code in bytes	processing time (ms)	Error (SSE)
1	Neural network with 4 neurons in cascade	705	1.72	0.5559
2	Neural network with 6 neurons in cascade	1119	3.3	0.0895
3	Neural network with 7 neurons in layers	840	3.8	0.2902

6 Conclusion

Neural controllers implemented on a microprocessor the code is simpler, much shorter; the processing time is comparable with fuzzy controllers. Control surfaces obtained from neural controllers also do not exhibit the roughness of fuzzy controllers that can lead to unstable or raw control. The only drawback of neural controllers is that the design process is more complicated than that of fuzzy controllers. However, this difficulty can be easily overcome with proper design tools. One severe disadvantage of a fuzzy system is its limited ability of handling problems with multiple inputs. This is not a serious limitation of neural networks. Control surfaces obtained from neural controllers also do not exhibit the roughness of fuzzy controllers that can lead to unstable or raw control.

7 References

[1] Wilamowski B. M. "Neuro-Fuzzy Systems and its applications" tutorial at 24th IEEE International Industrial Electronics Conference (IECON'98) August 31 - September 4, 1998, Aachen, Germany, vol. 1, pp. t35-t49.

[2] Kosko B., (1993) *Fuzzy Thinking, The New Science of Fuzzy Logic*. Hyperion, New York.

[3] Passino, K. M., S. Yurkovich, Fuzzy Control, Addison-Wesley, 1998.

[4] Wang, L., Adaptive Fuzzy Systems and Control, Design and Stability Analysis, PTR Prentice Hall, 1994.

[5] Wilamowski B.M. and J. Binfet, "Do Fuzzy Controllers Have Advantages over Neural Controllers in Microprocessor Implementation" Proc. of. 2-nd *International Conference on Recent Advances in Mechatronics - ICRAM'99*, Istanbul, Turkey, pp. 342-347, May 24-26, 1999.

[6] Wilamowski B. M. and R. C. Jaeger, "Neuro-Fuzzy Architecture for CMOS Implementation" accepted for *IEEE Transaction on Industrial Electronic*s.

[7] Hornik, K., "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, v.2, pp 359-366, 1989.

[8] Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks," Neural Networks, v.2, pp 183-192, 1989.

[9] L. A. Zadeh, Fuzzy sets. *Information and Control,* New York, Academic Press vol **8**, pp. 338-353, 1965.

[10] T. Takagi and M. Sugeno, Derivation of Fuzzy Control Rules from Human Operator's Control Action. *Proc. of the IFAC Symp. on Fuzzy Inf. Knowledge Representation and Decision Analysis,* pp. 55-60, July 1989.

[11] Stuttgart Neural Network Simulator SNNS http://www. informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/announce.html

[12] Wilamowski, B. M., "Neural Networks and Fuzzy Systems" chapters 124.1 to 124.8 in The Electronic Handbook. CRC Press 1996, pp. 1893-1914.