Journal of Control Science and Engineering 1 (2013) 1-12

Hardware Type 2 Fuzzy Logic Position Controller Based on Karnik-Mendel Algorithms

Pedro Ponce-Cruz, Arturo Molina and Arturo Tellez-Velazquez

Tecnologico de Monterrey, Distrito Federal 07240, Mexico

Abstract: This paper presents an analysis of the KM (Karnik-Mendel) algorithms performance under real time implementation using 3 types: the non-iterative, the iterative and the enhanced, and their feasibility for real-time interval type 2 fuzzy logic control system applications. The results are also compared against NT (Nie-Tan) method that is one of the fastest and simplest defuzzification methods. Because the DC (direct current) servo-motor is one of the most used motors in different industrial applications and the model of the motor is nonlinear, this motor was selected for validating the implementation in real time hardware. This DC motor is a perfect option for studying the real time performance of KM algorithms in order to show up its limits and possibilities for real-time control system applications. These methodologies are implemented in National Instruments LabVIEW FPGA (field programmable gate array) module hardware which is one of the most used platforms in the industry. The results show that the E-KM (enhanced KM) algorithm and the NT method present good results for implementing real-time control applications in real time hardware. Although fuzzy logic type 2 is a good option for working with nonlinear and noise from the sensors, the defuzzification method has to react in a short period of time in order to allow good control response. Hence, a complete study of defuzzification is needed for improving the real time implementations of fuzzy type 2.

Key words: Fuzzy logic type 2, KM algorithms, NT method, defuzzification, type-reduction, DC servo-motor control.

1 .Introduction

Different research works present three KM (Karnik-Mendel) algorithms for the IT2FLS (interval type 2 fuzzy logic systems*)*: the non-iterative [1] and the iterative [2] types that provide the same numerical results; the main difference is the improved total iteration count of the iterative type compared with the non-iterative one. Actually, both types are outperformed by the enhanced KM algorithm [3] which needs several initial conditions and lets the system converge faster than the other ones. These defuzzification methods get the generalized centroid of an IT2FLS.

Different applications require opportune decisions; those decisions which require the noise immunity that the IT2FLS can provide [4]. The KM algorithms are very intensive and sometimes they are not appropriated for real-time applications. When they are dealing with a big amount of data, a late response appears generating an incorrect decision which cannot be acceptable in real-time hardware systems.

The hardware implementation of IT2FLS offers good results but the software implementation is not good enough as it was shown in Ref. [5]. A hardware implementation is considered in this work for proving the KM algorithms in real-time applications.

For this purpose, a DC (direct current) servo-motor is analyzed. The IT2FLS is implemented in hardware by FPGA (field programmable gate array) based on LabVIEW and each method was implemented and tested independently. Two hardware considerations were taken: the first one is the swiftness and the second one is the final resource utilization. Every algorithm was compared using these two hardware conditions; also this paper shows a complete chart that

Corresponding author: Pedro Ponce-Cruz, Dr., research fields: robotics, control systems, instrumentation and artificial intelligence. E-mail: pedro.ponce@itesm.mx.

presents the complexity of every algorithm according to the discrete discourse universe points.

This paper defines a new procedure of establishing the hardware implementation based on LabVIEW FPGA for KM algorithms. According to the results presented, various KM algorithms are not appropriated for real-time control applications. Finally, expert must take into account the ratio between speed and area.

2. Interval Type 2 Fuzzy Logic Systems

The IT2FLS topology is the same as the T1FLS (type 1 fuzzy logic systems). It provides the fuzzification, the inference and the defuzzification stages.

The IT2FLS fuzzification maps the crisp values x_j into several membership values according to its membership degree. This means that a crisp value could belong to more than one IT2FS (interval type 2 fuzzy sets); those two membership degrees form a FOU (footprint of uncertainty). Each membership obtained after defuzzification process is related using fuzzy logic operations as conjunction (AND) and disjunction (OR). The fuzzy conjunction allows expert relating all the implied premises (input sets) and the fuzzy disjunction allows expert aggregating all these implied values in order to obtain a specific consequent (output set). With all these relations, a rule set is built. The rule set represents the IT2FLS conventional configuration.

Fig. 1 shows a complete picture of the IT2FLS which can be used in hardware implementations. Each block can be an independent hardware entity. The inferred set is calculated and stored in a memory location.

3. The KM Algorithms

In previous works [1-3, 6], the KM algorithms includes several modifications in order to decrement the impact of the large number of iterations required to generate a single centroid. This method is computationally exhaustive, different research works proposed some initial conditions and modifications based on the seminal algorithm. These modifications have the purpose of decreasing the search space for both the left and the right centroids.

Assume that the output discourse universe Y is a set of all the possible crisp outputs that can be obtained from a defuzzification method, i.e. $y \in Y$. Now, let B be a consequent set defined along Y. The key of finding each approximated centroid is to find a switch point where the ES (embedded set) will change from one outer membership function to another; specifically, the LMF (lower membership function) or $\underline{\mu}_{\tilde{B}}(y)$ and

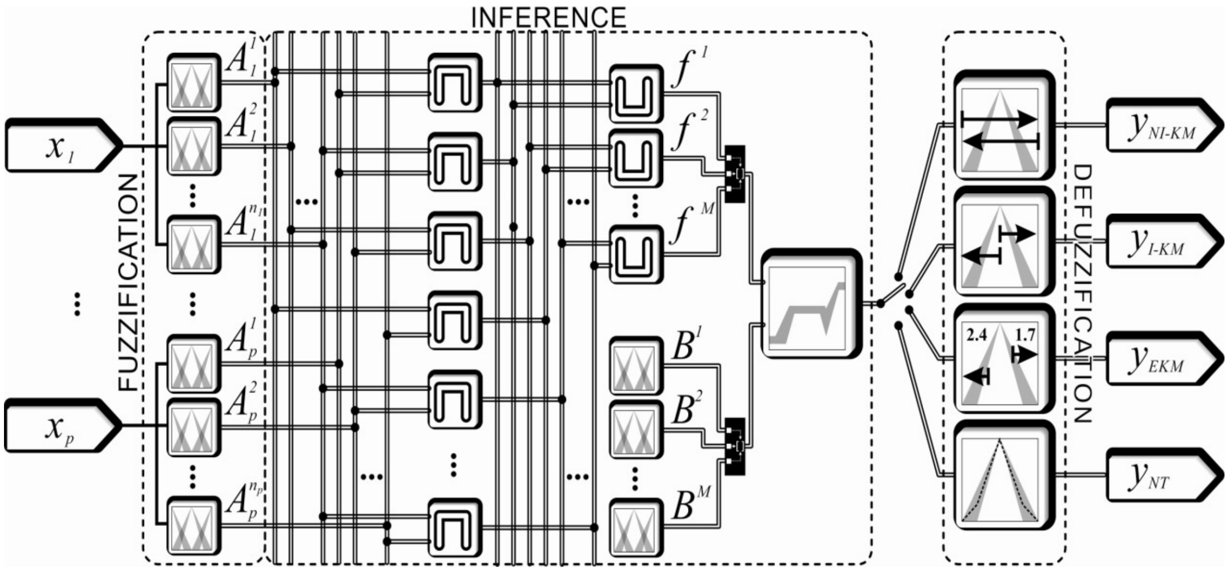


Fig. 1 IT2FLS with four defuzzification methods: the non-iterative, iterative and enhanced KM algorithms, and the NT (Nie-Tan) method.

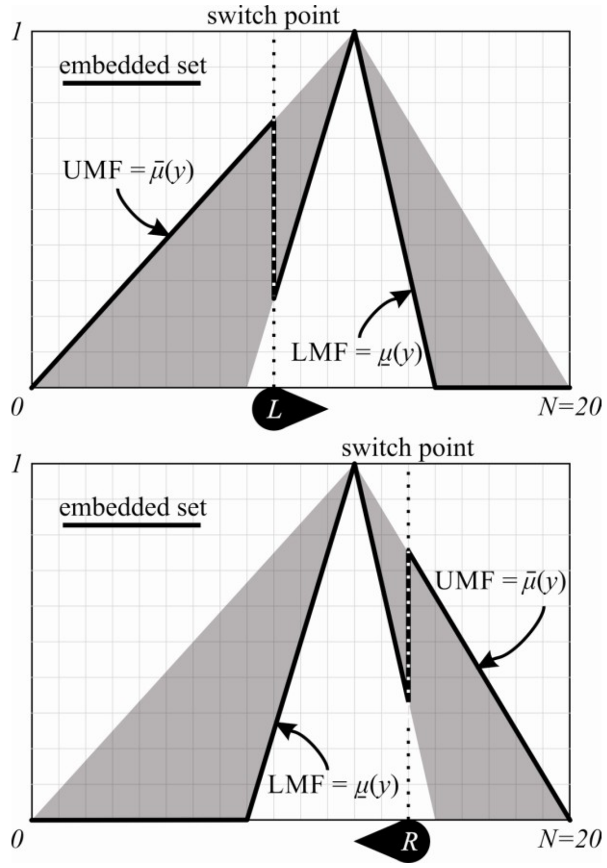


Fig. 2 The general centroid search in the KM algorithms. There is not centroid search for this algorithm; all the centroids are computed. This method is known as the NI-KM (non-iterative Karnik-Mendel) algorithm.

the UMF (upper membership function) or $\bar{\mu}_{\tilde{B}}(y)$, which comprises the FOU. Fig. 2 provides a graphical description of the KM algorithm, in general.

3.1 Non-Iterative Version

The NI-KM algorithm [1] creates all the embedded sets θ_L and θ_R as possible from 1 to N. For finding the left and right centroids, the expert must calculate the following formulas:

$$\theta_L = \begin{cases} \bar{\mu}_{\tilde{B}}(y_i) & y_i \le L \\ \underline{\mu}_{\tilde{B}}(y_i) & \text{otherwise} \end{cases}$$

$$c_{\theta L} = \frac{\sum_{i=1}^{N} y_i \cdot \mu_{\theta L}(y_i)}{\sum_{i=1}^{N} \mu_{\theta L}(y_i)} \tag{1}$$

where, $c_{\theta L} \in \Theta_l$

$$c_l \in \min(\Theta_l)$$

$$\theta_R = \begin{cases} \underline{\mu}_{\tilde{B}}(y_i) & y_i \le R \\ \bar{\mu}_{\tilde{B}}(y_i) & \text{otherwise} \end{cases}$$

$$c_{\theta R} = \frac{\sum_{i=1}^{N} y_i \cdot \mu_{\theta R}(y_i)}{\sum_{i=1}^{N} \mu_{\theta R}(y_i)} \tag{2}$$

where, $c_{\theta R} \in \Theta_r$

$$c_r \in \max(\Theta_r).$$

This non-iterative version creates a vector of left centroids Θ_l and a vector of right centroids Θ_r; from the left centroid vector, the far left centroid (the minimum) is selected while from the right centroid (maximum) vector, the far right centroid is selected. Fig. 3 shows the non-iterative version.

Eqs. (1) and (2) can be rewritten as two summations in the numerator and the denominator as follow:

$$c_{\theta L} = \frac{\sum_{i=1}^{L} y_i \cdot \bar{\mu}_{\tilde{B}}(y_i) + \sum_{i=L+1}^{N} y_i \cdot \underline{\mu}_{\tilde{B}}(y_i)}{\sum_{i=1}^{L} \bar{\mu}_{\tilde{B}}(y_i) + \sum_{i=L+1}^{N} \underline{\mu}_{\tilde{B}}(y_i)} \tag{3}$$

where, $c_{\theta L} \in \Theta_l$

$$c_l \in \min(\Theta_l)$$

$$c_{\theta R} = \frac{\sum_{i=1}^{R} y_i \cdot \underline{\mu}_{\tilde{B}}(y_i) + \sum_{i=R+1}^{N} y_i \cdot \bar{\mu}_{\tilde{B}}(y_i)}{\sum_{i=1}^{R} \underline{\mu}_{\tilde{B}}(y_i) + \sum_{i=R+1}^{N} \bar{\mu}_{\tilde{B}}(y_i)} \tag{4}$$

where, $c_{\theta R} \in \Theta_r$

$$c_r \in \max(\Theta_r).$$

3.2 Iterative Version

This iterative version [2, 7, 8] searches for the left and right centroids starting from a convenient initial embedded set θ_i. After the initial centroid, the following centroid search of θ_i for the left and right centroids helps the algorithm to converge to the final centroid faster than the non-iterative version. Fig. 4 shows the iterative KM algorithm.

For the KM algorithm, the procedure to find the left and right centroid c_l and c_r is the following:

(1) Sort all the discourse universe values y_i in ascending order, where i = 1, 2 , ⋯ , N , such $y_1 \le y_2 \le \cdots \le y_N$. Associate each y_i with its corresponding $\underline{\mu}_{\tilde{B}}(y_i)$ and $\bar{\mu}_{\tilde{B}}(y_i)$.

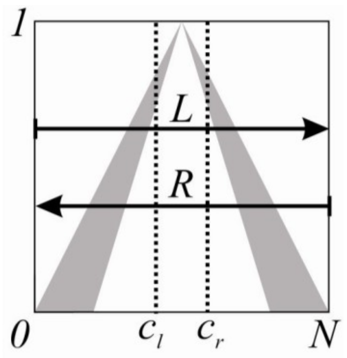


Fig. 3 The NI-KM algorithm performs all the centroid calculation.

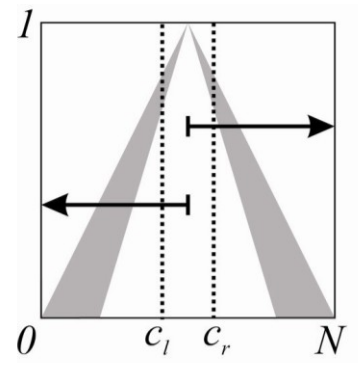


Fig. 4 The I-KM (iterative KM)algorithm. This algorithm is optimized for starting the centroid search from almost the half of discourse universe.

(2) Initialize

$$\theta_i = \frac{\underline{\mu}_{\tilde{B}}(y_i) + \overline{\mu}_{\tilde{B}}(y_i)}{2} \tag{5}$$

and compute

$$y = \frac{\sum_{i=1}^{N} y_i \theta_i}{\sum_{i=1}^{N} \theta_i} \tag{6}$$

(3) Find the switch point k, such $1 \le k \le N - 1$ and $y_k \le y_{k+1}$,

(4) Establish

For the left centroid:

$$\theta_r = \begin{cases} \overline{\mu}_{\tilde{B}}(y_i) & i \le k \\ \underline{\mu}_{\tilde{B}}(y_i) & \text{otherwise} \end{cases}$$

For the right centroid:

$$\theta_i = \begin{cases} \underline{\mu}_{\tilde{B}}(y_i) & i \le k \\ \overline{\mu}_{\tilde{B}}(y_i) & \text{otherwise} \end{cases}$$

and compute

$$y = \frac{\sum_{i=1}^{N} y_i \theta_i}{\sum_{i=1}^{N} \theta_i}$$

(5) If $y' = y$. If true, then stop and assign $c_l = y$ or $c_r = y$, correspondingly, else continue.

(6) Assign $y = y$, y go to step 3.

In both versions, the generalized centroid is calculated simply by averaging both c_l and c_r centroids:

$$y = \frac{c_l + c_r}{2} \tag{7}$$

3.3 Enhanced KM Algorithm

To improve the KM algorithm calculations, the E-KM (enhanced KM) algorithm [3] does not start only at the initial embedded set; it also starts from a convenient switch point (Fig. 5).

The algorithm to find the left centroid c_l is the following:

(1) Sort all the discourse universe values x_i in ascending order, where $i = 1, 2, \cdots, N$, such $y_1 \le y_2 \le \cdots \le y_N$. Associate each y_i with its corresponding $\underline{\mu}_{\tilde{B}}(y_i)$ and $\overline{\mu}_{\tilde{B}}(y_i)$.

(2) Establish k = round (N/2.4) and compute

$$a = \sum_{i=1}^{k} y_i \overline{\mu}_{\tilde{B}}(y_i) + \sum_{i=k+1}^{N} y_i \underline{\mu}_{\tilde{B}}(y_i)$$

$$b = \sum_{i=1}^{k} \overline{\mu}_{\tilde{B}}(y_i) + \sum_{i=k+1}^{N} \underline{\mu}_{\tilde{B}}(y_i)$$

$$y = \frac{a}{b}$$

(3) Find the switch point $k' \in [1, N\text{-}1]$, such

$$y_{k'} \le y \le y_{k'+1} \tag{8}$$

(4) Check if $k' = k$. If true, stop and assign $c_l = y$; else continue.

(5) Compute

$$a' = a \pm \sum_{i=\min(k,k')}^{\max(k,k')} y[\overline{\mu}_{\tilde{B}}(y_i) - \underline{\mu}_{\tilde{B}}(y_i)]$$

$$b' = b \pm \sum_{i=\min(k,k')+1}^{\max(k,k')} [\overline{\mu}_{\tilde{B}}(y_i) - \underline{\mu}_{\tilde{B}}(y_i)]$$

and compute again

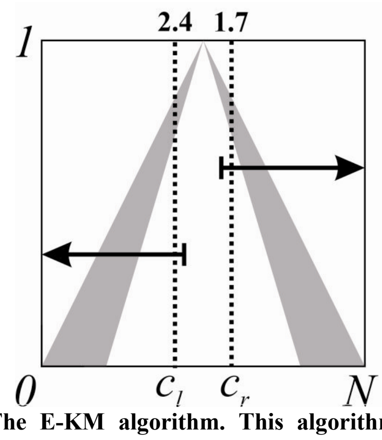


Fig. 5 The E-KM algorithm. This algorithm is doubly optimized for starting the centroid search eight percent around N/2 of discourse universe. That is why some literature uses the values 1.7 and 2.4 to start the centroid search.

$$y' = \frac{a'}{b'}$$

(6) Assign $y = y'$, $a = a'$, $b = b'$, $k = k'$ and go to step 3.

Each preliminary definition of $k_l = N/2.4$ and $k_r =$ N/1.7 suggests that the initial centroids can be found ±8% around the middle of the set support, i.e., $k_l = (N/2) - 0.08N$ and $k_r = (N/2) + 0.08N$. These values were obtained experimentally [3]. This initial search reduces the search of the final centroid.

3.4 Nie-Tan Method

Although the NT method is not a KM algorithm derivation, this is used in the iterative-version KM algorithms to find the initial switching point. This method is discussed in this work because of its simplicity. The simplicity of the algorithm is based on the average between the LMF and UMF as it is shown in Fig. 6.

The NT method [9, 10, 11] is the simplest and fastest defuzzification method for IT2FLS.

It searches the middle MF θ_i between UMF and LMF. Its centroid is the approximated generalized centroid:

$$\theta_i = \frac{\overline{\mu}_{\tilde{B}}(y_i) + \underline{\mu}_{\tilde{B}}(y_i)}{2} \tag{9}$$

$$y = \frac{\sum_{i=1}^{N} y_i \cdot \mu_\theta(y_i)}{\sum_{i=1}^{N} \mu_\theta(y_i)} \tag{10}$$

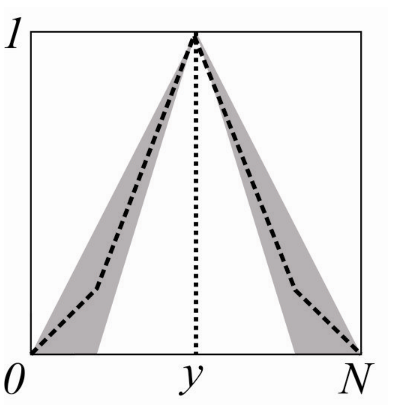


Fig.6 The NT method.

This method is one of the fastest methods because it requires only a division by 2 (1-position right shift register) and only a single centroid calculation.

Following section explains a hardware overview in FPGA in which the KM algorithms and the NT method were implemented.

4. DC Servo-Motor

This paper presents a hardware study between several defuzzification methods for IT2FLS. For doing this, the DC Servo-Motor application is proposed. The following section provides some theories about the DC servo-motor plant which is controlled by the IT2FLS.

The DC servo-motor model is well defined by conventional differential equations according to the Fig. 7. This model contains the complete description of the servomotor. Normally, in the control of DC servo-motor appears noise from the sensors, so a controller that could get rid of the noise effect is able to improve the whole performance of the system. The following figure shows the basic topology of the DC motor model.

The equations that describe the DC motor performance are presented below:

$$T = K_e \cdot i \tag{11}$$

$$e_a = K_f \cdot w_m \tag{12}$$

$$T = J \cdot \frac{\mathrm{d}}{\mathrm{d}t} w_m + b \cdot w_m \tag{13}$$

$$L \cdot \frac{\mathrm{d}}{\mathrm{d}t} i + R \cdot i = V - e_a \tag{14}$$

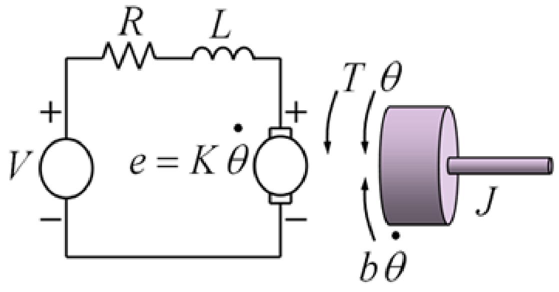


Fig. 7 DC servo-motor model.

where,

T: motor torque;

K_e: torque constant;

K_f: voltage constant;

i: armature current;

e_a: generated voltage;

w_m: angular velocity;

J: inertia;

b: damping ratio;

L: armature inductance;

R: armature resistance;

V: input voltage.

4.1 Laplace Transform Model

Rearranging Eqs. (11)-(14) :

$$K \cdot i(t) - b \cdot w_m(t) = J \cdot \frac{d}{dt} w_m(t) \qquad (15)$$

$$V(t) - K \cdot w_m(t) - R \cdot i(t) = L \cdot \frac{d}{dt} i(t) \qquad (16)$$

Our principal goal is to control the DC servo-motor speed by changing the input voltage V, so the Eqs. (15) and (16) can be transformed in terms of voltage V and angular speed w_m.

$$i(t) = \frac{J}{K} \cdot \frac{d}{dt} w_m(t) + \frac{b}{K} \cdot w_m(t) \qquad (17)$$

Replacing Eq. (17) in Eq. (16) we get,

$$\left(\frac{L \cdot J}{K}\right) \cdot \frac{d^2}{dt^2} w_m(t) + \left(\frac{L \cdot b + R \cdot J}{K}\right) \cdot \frac{d}{dt} w_m(t) + \left(\frac{K^2 + R \cdot b}{K}\right) \cdot w_m(t) = V(t) \qquad (18)$$

Before applying Laplace transform is necessary to represent the last equation in terms of deviation variables,

$$\left(\frac{L \cdot J}{K}\right) \cdot \frac{d^2}{dt^2} W_m(t) + \left(\frac{L \cdot b + R \cdot J}{K}\right) \cdot \frac{d}{dt} W_m(t) + \left(\frac{K^2 + R \cdot b}{K}\right) \cdot W_m(t) = \forall(t) \qquad (19)$$

where,

$$W_m(t) = w_m(t) - w_m(0)$$

$$\forall(t) = V(t) - V(0)$$

And $w_m(0)$ and $V(0)$ are the initial conditions.

Applying the Laplace transform we get,

$$\left(\frac{L \cdot J}{K}\right) \cdot s^2 W_m(s) + \left(\frac{L \cdot b + R \cdot J}{K}\right) \cdot s W_m(s) + \left(\frac{K^2 + R \cdot b}{K}\right) \cdot W_m(s) = \forall(s) \qquad (20)$$

Rearranging Eq.(20) we obtain the transfer function that represents the servomotor plant in terms of Laplace transform as follows:

$$\frac{W_m(t)}{\forall(s)} = \frac{1}{\left(\frac{L \cdot J}{K}\right) \cdot s^2 + \left(\frac{L \cdot b + R \cdot J}{K}\right) \cdot s + \left(\frac{K^2 + R \cdot b}{K}\right)} \qquad (21)$$

4.2 State-Space Ttransfer Function

From Eq. (18), it can be defined the state-space model introducing the next variables,

$$x_1 = w_m(t) \qquad (22)$$

$$x_2 = \frac{d}{dt} w_m(t) \qquad (23)$$

The model is defined as,

$$\dot{x}_1 = \frac{d}{dt} w_m(t) = x_2 \qquad (24)$$

$$\dot{x}_2 = \frac{d^2}{dt^2} w_m(t) = \left(\frac{K}{L \cdot J}\right) \cdot V(t) - \left(\frac{L \cdot b + R \cdot J}{L \cdot J}\right) \cdot x_2 - \left(\frac{K^2 + R \cdot b}{L \cdot J}\right) \cdot x_1 \qquad (25)$$

$$y = w_m(t) = x_1 \qquad (26)$$

$$u = V(t) \qquad (27)$$

Finally, the state-space equations are,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\left(\frac{K^2 + R \cdot b}{L \cdot J}\right) & -\left(\frac{L \cdot b + R \cdot J}{L \cdot J}\right) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \left(\frac{K}{L \cdot J}\right) \end{bmatrix} [u] \tag{28}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{29}$$

And the DC motor model constants are,

$$A = \begin{bmatrix} 0 & 1 \\ -\left(\frac{K^2 + R \cdot b}{L \cdot J}\right) & -\left(\frac{L \cdot b + R \cdot J}{L \cdot J}\right) \end{bmatrix} \tag{30}$$

$$B = \begin{bmatrix} 0 \\ \left(\frac{K}{L \cdot J}\right) \end{bmatrix} \tag{31}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{32}$$

$$D = 0 \tag{33}$$

4.3 Servo-Motor Control System

The servo-motor control system for the defuzzification method performance comparison is shown in Fig. 8, where the IT2FLS performs the position control using the position error θ and the change of the position θ; also, the servomotor plant can be represented by Eq. (21) or Eqs. (30) and (33). Fig. 9presents some details about the IT2FLS and its rules; nine rules are performed relating the antecedents to infer the consequences.

5. The Hardware Complexity

A real-time application can be managed by a computational device but the device selection will depend on the latency of that digital system that will manage it. A real-time application requires fast digital systems (with low latency) because its real-time characteristic will be defined by its WCET (worst case execution time).

The centroid calculation had been considered as a problem in T1FLS, for real-time applications because of its large latency. Although several applications were solved without problems with the centroid calculation in T1FLS, the use of this in the KM algorithms are very common and intensive and therefore the IT2FLS are hugely more complex than the T1FLS. The search of each centroid for each embedded set is globally several times more complex compared with the T1FLS.

This is one of the reasons why the IT2FLS are not used as the T1FLS today, and its robustness capability is not approached. The IT2FLS are often implemented in hardware such as RT-MCU (real-time microcontroller units), or DSP (digital signal processors) [12] and FPGA [13], because the rest of the hardware (computers with sequential program execution) becomes impractical and not feasible for real-time applications. These digital systems are often more expensive, especially the FPGA, and commonly this economic reason limits the application implementation in most cases.

Some of the high-end FPGA hardware elements that can be found in real-time applications are:

- high-speed dedicated multiplications;
- high-speed dedicated memory;
- DSP blocks;
- real-parallelism;
- partial reconfiguration.

Full-customizable multiplications are available in the FPGAs as a solution for improving the timing performance. Xilinx is the main FPGA brand that is used widely in several applications along the world. Some Xilinx FPGA families provide faster multiplication blocks and additional high performance resources (like DSP blocks and RT processors in the same land) in their higher end devices. Some high-speed RAM (random access memory) blocks are available for storing data, for instance.

These resources help the expert to design high speed and low cost T1FLS and IT2FLS in hardware among the software approaches, because a sequential program is executed in a fixed computing architecture [5]. If the specific application deserves the use of IT2FLS, then the expert must take into account how to select the appropriated defuzzification method that will be used for its real-time application.

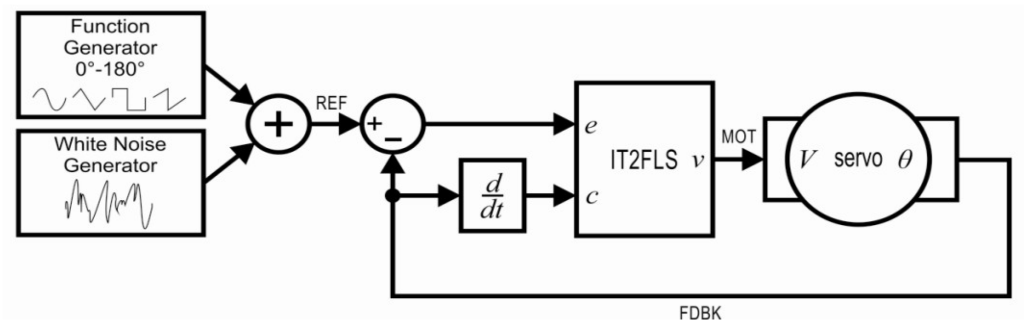
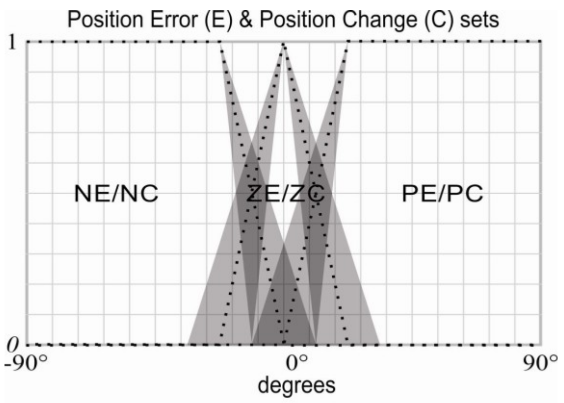


Fig. 8 The servo-motor control system.



$$
\begin{array}{ll}
r_1 & e{:}\,NE \sqcap c{:}\,NC \Rightarrow v{:}\,NV \\
r_2 & e{:}\,NE \sqcap c{:}\,ZC \Rightarrow v{:}\,NV \\
r_3 & e{:}\,NE \sqcap c{:}\,PC \Rightarrow v{:}\,NV \\
r_4 & e{:}\,ZE \sqcap c{:}\,NC \Rightarrow v{:}\,NV \\
r_5 & e{:}\,ZE \sqcap c{:}\,ZC \Rightarrow v{:}\,ZV \\
r_6 & e{:}\,ZE \sqcap c{:}\,PC \Rightarrow v{:}\,PV \\
r_7 & e{:}\,PE \sqcap c{:}\,NF \Rightarrow v{:}\,PV \\
r_8 & e{:}\,PE \sqcap c{:}\,ZF \Rightarrow v{:}\,PV \\
r_9 & e{:}\,PE \sqcap c{:}\,PF \Rightarrow v{:}\,PV
\end{array}
$$

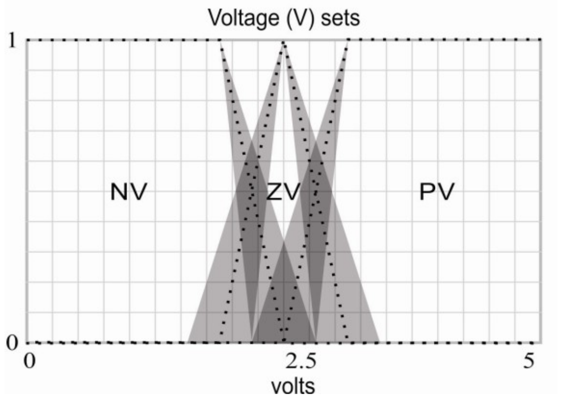


Fig. 9 The fuzzy sets and the rule set for the servo-motor application.

6. Methodology

The hardware defuzzification method comparisons are performed implementing all the KM algorithms, the NT method in FPGA hardware; National Instruments Reconfigurable I/O (compact RIO or cRIO) device is used and LabVIEW FPGA module to program every defuzzification stage enunciated in this paper and all the IT2FLS architecture. The cRIO device used is the NI-9014 with analog I/O capabilities, the C-series NI-9263 and the NI-9201 modules.

For this reason, the expert may also know some hardware details about the KM algorithms that will use.

For hardware comparison, the DC servo-motor application is proposed as a study case where the rotor position θ is tracked when a signal generator is introduced and a noise generator is used to disturb the IT2FLS performance, according to Fig. 8. Although the servo case is relatively slow, all these defuzzification methods can be analyzed and compared. Several studies have been implemented in FPGA hardware for T1FLS [14] and IT2FLS [15]. However, no comparisons have been presented, in hardware terms, where the most important defuzzification methods are applied for solving the same problem.

The DC servo-motor application, as can be seen in Figs. 8 and 10, is an electronic control training module dedicated for the cRIO device where the user can test control performance easily. The servomotor moves the rotor position from 0° to 180° according to the REF analog signal; the REF signal is used in the cRIO as an input to determine the reference that the controller will follow. The FDBCK (feedback) signal is another analog signal used in the cRIO as an input also; the difference is that this signal is used to know where the current rotor position is.

The control system described in Fig. 8 is implemented using the LabVIEW FPGA module, where several diagram blocks are used for the data acquisition (input nodes), the noise generator summed to the reference signal (signal generator), the data driving to the servo (output node), and finally the IT2FLS. The error is computed as the difference between the desired values, such the REF (reference) signal, i.e., the reference and the current position value FDBCK, as can be seen in Figs. 8 and 10.

The cRIO analog output is used to drive the desired voltage to the servo-motor and this way determine the new position of the rotor. This signal is applied in the MOT (motor) signal as can be seen in the training module of the Fig. 10.

Details about the servo-motor control training module are not known, although their general equations are defined (From Eqs.(20) to (29)). This is not important for the fuzzy controller, because the IT2FLS must determine its control law without knowing the exact system dynamics.

For comparison purposes, the noisy reference signal REF is used so that the IT2FLS can follow it, the fuzzification and inference processes are previously tuned and are the same for the four cases. Four signal shapes were tested with the four methods: the NI-KM algorithm, the I-KM algorithm, the E-KM algorithm and the NT method, as can be seen in Fig. 11.

7. Results and Discussions

7.1 Reference Tracking

The final response of the control system using all the defuzzification methods mentioned is shown in Fig. 11. Four signal shapes were used to the IT2FLS tracking.

The NI-KM algorithm is the worst case and its latency makes the decision to take late decisions. The best results were obtained with the E-KM algorithm and the NT method. This behavior is similar in each case.

7.2 The Hardware Performance

The following paragraphs are dedicatedto describe the hardware complexity between every defuzzification method described in previous sections. NI-cRIO devices are used in conjunction with NI LabVIEW FPGA module.

The following section provides a timing and area performance analysis where each KM algorithm is analyzed and compared. Also, the NT method is included.

7.2.1 Complexity and Arithmetic-related

Assume that each algorithm requires two memories with N locations and each centroid (either left or right) are computed in parallel. Each element is a byte (8-bit width). Every memory can be distributed in the FPGA by LUT (look-up tables).

The following chart provides a comparison between all the KM algorithms and the NT method.

According to Eqs.(1-9), the total iteration count can be obtained as an approximation if a counter is included in the most inner loop of the algorithm and by observation of the block diagram. In this case, $N = 256$, where N is the number of points the discourse universe is divided into. So, the non-iterative KM may

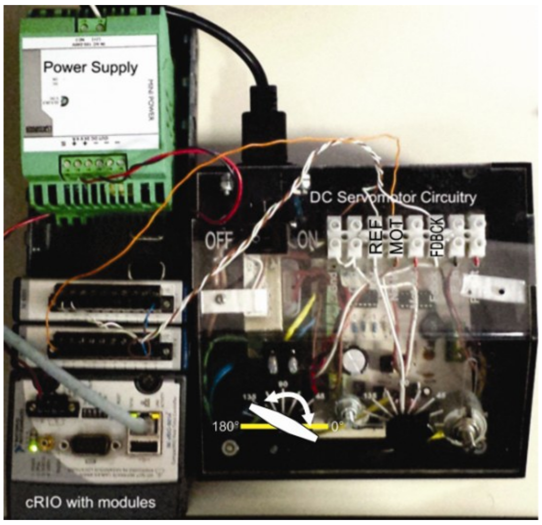


Fig. 10 The control of a servo-motor training module and the cRIO with C-series modules.

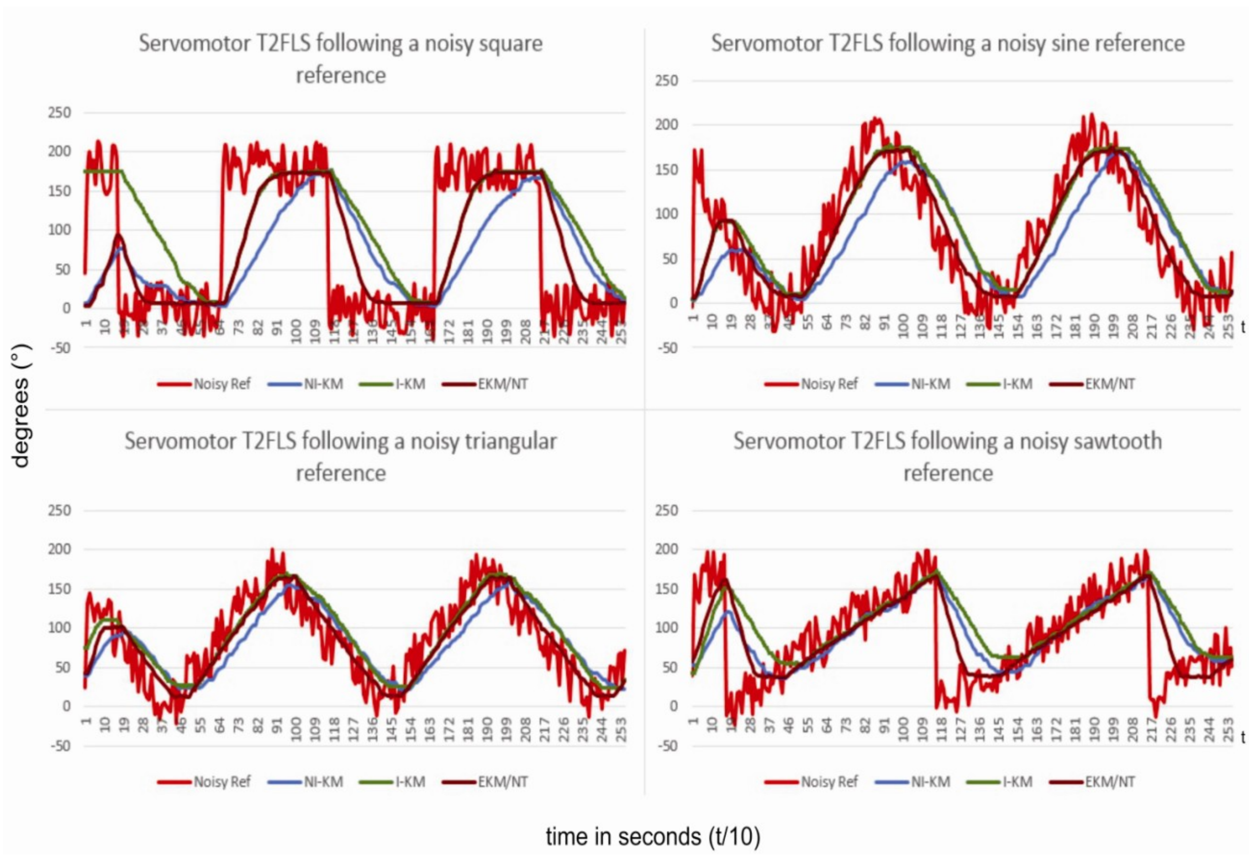


Fig. 11 All the KM algorithms and NT method tracking performance comparison in presence of noise. Four signal shapes were used to be tracked by the IT2FLS using each algorithm. The total time for each experimental test is 25.5 seconds.

last $2N^2 + 4N = 2(256)^2 + 4(256) = 132.096$ iterations for a single defuzzification without taking into account the iterations required for the T2 fuzzification and T2 inference processes, as can be seen in Table 1. For instance, the E-KM requires from 1 up to $0.1764N^2 + 2.1N = 0.1764(256)^2 + 2.1(256) = 12.098$ iterations to find the final left or right centroid. Due to each left or right centroid calculation is processes in parallel; it can be seen as a single loop. The iterative version, which can be compared with the non-iterative version and may last the same time to execute with the difference of the early termination condition, converges slower than the E-KM.

The NT method also finishes before, because its implementation requires only a single centroid calculation and only a single embedded set in the set FOU. Although the NT method seems to be the fastest, the E-KM may find the final centroid in the first iteration, which very significant compared with the NT algorithm.

Table 1 summarizes the complexity of each defuzzification method, which was obtained experimentally.

7.2.2 Resource Usage

The cRIO device provides several limited resources like multiplications, memories, amongst others which provide very high performance and the expert may use to build more complex structures like divisions or square roots. LabVIEW FPGA module is a GUI (graphical user interface) which provides several basic tools that can be used to implement high-throughput operations in a RIO device, controls that let the user modify the input data to the digital system and indicators that let the user show the output data from the digital system, related to a virtual instrument that works as the programming unit.

In functional block terms (LabVIEW FPGA module), the KM algorithms and the NT method were designed, implemented are compared according to the structures used for being implemented.

After the module compilation, LabVIEW FPGA provides the FPGA resulting resource utilization. As can be seen in the last chart (Table 2), the resource usage can be useful for comparison purposes. This way, can be seen that the E-KM is the most expensive, but this is the fastest defuzzification method, but the NT method is the cheapest method because requires only one division and one multiplication for calculating the final centroid. Because this method does not search for several embedded sets, the final centroid is a single value and that is why requires of a single centroid calculation unit. The E-KM and the I-KM require 3 centroid calculation units while the non-iterative version requires 2.

7.2.3 Timing and Area Resource Usage

Also, the final timing and resources of each method is presented using the LabVIEW Tick Count block. And the resource utilization is obtained from the Build Specifications in LabVIEW FPGA module.

The best timing performance is reached by the E-KM algorithm and the second best one is the NT, although the E-KM also presents the worst resource utilization. Now observe that the non-iterative KM algorithm presents the worst case, achieving about 20 FLIPS (fuzzy logic inferences per second), which is not practical for the IT2FLS applied in real-time questions. Also, the NT method presents the best resource utilization. The software VI timing performance depends on the operating system tick time, which is generally 55 milliseconds per tick, so for the NI-KM the total ticks are 1.537, then $1.537 \times 55ms = 84.535ms$, as can be seen in Table 3.

LabVIEW FPGA module is used as reference for testing hardware and performing comparisons. This module creates a whole computing architecture that generates great amount of slices for any design. Its advantage stands on the fact of the ease of hardware validation due to its graphical interface, characteristic of the virtual instruments in LabVIEW.

The final resource utilization may vary if the system is implemented directly in a Xilinx FPGA and without using the LabVIEW FPGA module.

Table 1 Number of iterations per defuzzification method.

Element/method	NI-KM	I-KM	E-KM	NT
ES calculation	N	N	N/A	N
ES centroid calculation	$N + 3$	$N + 3$	$N + 1$	$N + 3$
Total iteration count	$2N^2 + 4N$	$[N/2,\ 2N^2 + 4N]$	$[1, 0.1764N^2 + 2.1N]$	$2N + 3$

Table 2 Number of hardware elements used for each defuzzification method.

Structure/method	NI-KM	I-KM	E-KM	NT
Multiplications	1	1	6	1
Divisions	1	2	5	1
Sums/subtracts	3	4	17	5
Centroid calculation units	2	3	3	1
MUX (Comparator/multiplexers)	5	9	9	1

Table 3 Timing performance and resource utilization per defuzzification method.

Resource/methods	NI-KM	I-KM	E-KM	NT
Latency (hardware) in milliseconds	49.48	0.8875	0.1756	0.27
Latency (software) in milliseconds	84.535	1.87	1.43	1.32
Slices	1,461	2,415	2,593	915
Registers	1,454	2,087	2,828	959
LUT	2,185	3,759	3,965	1,305

8. Conclusions

The E-KM algorithm and the NT Method are the best choices for implementing real-time control applications in hardware. The NI-KM is not practical due to its poor response. Then, depending on the expert requirements, the expert must choose between speed and resources, so the E-KM algorithm provides the best speed although not the best resource usage. The NT method is fast and simple in the number of resource for real-time control applications.
In the specific case of DC motors is an excellent option to use a the E-KM in the defuzzification stage, so it could be very useful to implement it in different industrial applications like CNC (computer numerical control) machines that run in real time.

Acknowledgments

This work is sponsored by Instituto Tecnologico de Estudios Superiores de Monterrey and National Instruments.

References

[1] J.M. Mendel, Type-2 fuzzy sets and systems: An Overview, IEEE Computational Intelligence Magazine 2 (1) (2007) 20-29.

[2] N.N. Karnik, J.M. Mendel, Q. Liang,Type-2 fuzzy logic systems, IEEE Transactions on Fuzzy Systems 7 (1999) 643-658.

[3] H. Wu, J. M. Mendel, Enhanced Karnik-Mendel algorithms, IEEE Transactions on Fuzzy Systems 17 (2009) 923-934.

[4] M.Biglarbegian, W. Melek, J.Mendel, Robustness of interval type-2 fuzzy logic systems, in: IEEE NAFIPS Canada, 2010.

[5] N. Manaresi, R. Rovatti, E. Franchi, R. Guerrieri, G. Baccarani, Automatic synthesis of analog fuzzy controllers: A hardware and software approach, IEEE Transactions on Industrial Electronics 43 (1) (1996) 217-225.

[6] J. Mendel, X. Liu, Some extensions of the Karnik-Mendel algorithms for computing an interval type-2 fuzzy set centroid, in: IEEE Symposium on Advances in Type-2 Fuzzy Logic Systems, Paris, France, April 11-15, 2011.

[7] J. Mendel, F. Liu, Super-Exponential convergence of the Karnik-Mendel algorithms used for type-reduction in interval type-2 fuzzy logic systems, in: IEEE International Conference on Fuzzy Systems, Vancouver, BC, Canada, July 16-21, 2006.

[8] J. Mendel, R.I.B. John, Type-2 fuzzy sets made simple, IEEE Transactions on Fuzzy Systems 10 (2) (2002) 117-127.

[9] J.M. Mendel, X. Liu, Simplified interval type-2 fuzzy logic systems, IEEE Transactions on Fuzzy Systems 21 (6) (2013) 1056-1069.

[10] D. Wu, J. M. Mendel,On the Continuity of type-1 and interval type-2 fuzzy logic systems, IEEE Transactions on Fuzzy Systems 19 (1) (2011) 179-192.

[11] M. Nie, W.W. Tan, Towards an efficient type-reduction method for interval type-2 fuzzy logic systems, in: IEEE World Congress on Computational Intelligence, Hong Kong, China, June 1-6, 2008.

[12] R. Lauwereins, M. Engels, J.A. Peperstraete, Parallel processing enables the real-time emulation of DSP ASICs, IEEE International Workshop on Rapid System Prototyping, North Carolina, USA, June 4-7, 1990.

[13] Y. Chen, V. Dinavahi, Multi-FPGA digital hardware design for detailed large-scale real-time electromagnetic transient simulation of power systems, IET Generation, Transmission & Distribution 7 (5) (2013) 451-463.

[14] M. Cirstea, J. Khor, M. McCormick, FPGA fuzzy logic controller for variable speed generators, in: IEEE International Conference on Control Applications, Mexico, Sep. 5-7, 2001.

[15] M. Melgarejo, C.A. Pena-Reyes, Implementing interval type-2 fuzzy processors, IEEE Computational Intelligence Magazine 2 (1) (2007) 63-71.