INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 10, NO. 4, DECEMBER 2005, 258-268

# High-speed Integer Operations in the Fuzzy Consequent Part and the Defuzzification Stage for Intelligent Systems

Sang Gu LEE and John D. CARPINELLI

*Abstract* - **In high-speed fuzzy control systems applied to intelligent systems such as robot control, one of the most important problems is the improvement of the execution speed of the fuzzy inference. In particular, it is more important to have high-speed operations in the consequent part and the defuzzification stage. To improve the speedup of fuzzy controllers for intelligent systems, this paper presents an integer line mapping algorithm to convert [0, 1] real values of the fuzzy membership functions in the consequent part to a 400 × 30 grid of integer values. In addition, this paper presents a method of eliminating the unnecessary operations of the zero items in the defuzzification stage. With this representation, a center of gravity method can be implemented with only integer additions and one integer division. The proposed system is analyzed in the air conditioner control system for execution speed and COG, and applied to the truck backer-upper control system. The proposed system shows a significant increase in speed as compared with conventional methods with minimal error; simulations indicate a speedup of an order of magnitude. This system can be applied to real-time high-speed intelligent systems such as robot arm control.**

***Index Terms***— Fuzzy hardware, Intelligent system, Integer operation, Defuzzification, COG, Fuzzy control system

## 1. INTRODUCTION

Since fuzzy set theory was proposed by Zadeh [18], and fuzzy control logic was first implemented by Mamdani [11], fuzzy systems have attracted a great deal of worldwide attention. Fuzzy logic models are successfully being used in many engineering applications involving the fuzzy control, data mining, expert system, pattern recognition, pattern clustering, prediction, and medical diagnosis [16].

Fuzzy set properties are useful in performing operations involving membership functions. A membership function maps every element of the universe of discourse X to the interval [0, 1], and this mapping can be represented as

$$\mu(x): X \rightarrow [0, 1].$$

In fuzzy sets, all these properties can be expressed using the membership functions of the sets involved and the definitions of union, intersection, and complement. Many floating-point operations of the membership functions are required to calculate output values.

In general arithmetic operations, the difference in the execution speed between the integer operation and the floating-point operation is quite large. Although this speed is different for each CPU, integer multiplication and division are about ten times faster than floating point multiplication and division. In the case of the complex and large volumes of computations, the difference of speed can dominate overall system performance.

Fuzzy control systems are rule-based systems in which a set of fuzzy rules represents a control decision to adjust the effects of certain outputs generated by the system. The aim of fuzzy control systems is to substitute for a skilled human expert with a fuzzy rule-based system. The controller inputs crisp data directly from a number of sensors; through the process of fuzzification these are changed to linguistic or fuzzy membership functions. They then go through a set of fuzzy "If – Then" rules in an inference engine much like an expert system and result in some fuzzy outputs. The fuzzy outputs are then changed back into crisp (non-fuzzy) values through a process called defuzzification. Defuzzification is a mapping from a space of fuzzy control actions defined over an output universe of discourse into a space of crisp control actions. Defuzzification produces a non-fuzzy control action that best represents the inferred fuzzy output variable. In this defuzzification stage, a large number of operations including multiplication, addition, and division are performed. Thus, defuzzification is usually one of the most time-consuming procedures in fuzzy processing.

There are many defuzzification methods, including center-of-area (COA), center of gravity (COG), height defuzzification (HD), center-of-largest-area (COLA), and mean-of maxima (MOM) [10]. In most cases, the COG method is used and has been shown to yield superior results. This paper also uses the COG method.

In fuzzy systems, it takes little time to compute the α (degree of fulfillment) values in the condition part for all the fuzzy rules. However, in the consequent part, it takes much time and requires a large number of floating-point operations to compute the center of gravity in the defuzzification stage. If the universe of discourse is continuous, the COG method generates an output as shown in Eq. (1).

$$COG = \frac{\int x \cdot f(x) dx}{\int f(x) dx} \tag{1}$$

Here, $x$ and $f(x)$ are the output fuzzy variable and its membership function due to the consequent fuzzy rules, respectively. However, integration requires a relatively large amount of time to calculate, so Eq. (2) is often used in the case of a discrete universe.

$$COG = \frac{\sum x \cdot f(x)}{\sum f(x)} \tag{2}$$

In Eq. (2), the $x$-axis is subdivided into many quantized points, each having a floating-point value, and then

Manuscript received August 18, 2005; revised November 2, 2005.
Sang Gu Lee is with the Department of Computer Engineering, Hannam University, 133 Ojeong-dong Daedeok-gu, Daejon 306-791, Republic of Korea (email:sglee@hannam.ac.kr). John D. Carpinelli is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA (email: carpinelli@njit.edu)

summations are performed. For this, in general, it takes much time to compute many floating-point operations.

Fuzzy processing using conventional methods can be summarized as follows. Fuzzy computing can be carried out with either floating-point arithmetic operations or LUT (lookup table) method. In fuzzy systems using microcontrollers, such as the 68HC12 or I8051 chips, the LUT method is generally used [5]. This method can reduce the size of memory space; however, it requires the operation of real value interpolation processing [19]. The FLASP system [14] uses the LUT method and introduces pipelined processing technique for speed enhancement. Aranguren [2] used the LUT method, and pipeline processing is applied in the defuzzification. In the KAFA system [7], floating-point operations are used and a simple parallel processing technique is used in the defuzzification stage. The FZP-0401A system [17, 9] also uses floating-point operations. Like these systems, most conventional fuzzy controllers and fuzzy software tools [5] use LUT or floating-point operations for fuzzy inference. Until now, however, there have been no fuzzy controllers using only integer operations, mostly integer add operations, in the consequent part and defuzzification stage.

To overcome these problems, this paper uses Bresenham's scan line algorithm [3] to map the real values to a 400 × 30 integer grid. Also we propose a method of eliminating the unnecessary operations of the continuous zero items in the defuzzification stage. We apply the proposed system to an air conditioner control system and a truck backer-upper control system for performance evaluation. The proposed system is much faster than the conventional methods using floating-point operations and introduces only a minimal error. This system can be applied to the real-time high-speed intelligent systems such as robot arm control.

The rest of this paper is organized as follows. In Section 2, we propose the algorithm for the integer operations in the consequent part and the defuzzification stage. We present the integer mapping algorithm for the consequent part, the data structure for the consequent part and the defuzzification stage, and the proposed COG operations using integer addition operations. This section also analyzes the execution speed and COG in an air conditioner control system and simulates the truck backer-upper control problem and validates the control results of the proposed system, and the proposed method is compared with the conventional methods. Section 3 analyzes the performance of the algorithm using both high-level software implementation and VHDL synthesis. This section also examines methods to improve performance via parallel processing. Section 4 presents concluding remarks and directions for further research.

## 2. VERY HIGH SPEED FUZZY SYSTEM USING INTEGER OPERATIONS

Most fuzzy membership functions have generally triangular or trapezoidal shapes. The graph of a fuzzy membership function is a set of lines that connect the vertices in the function. In fuzzy inferencing, if fuzzy inputs are given as singleton values, it takes little times to find α, the degree of fulfillment, in each fuzzy rule. However, in the consequent part, it takes much more time than in the condition part due to the many real-valued operations in [0, 1] corresponding to the universe of discourse. Also, in the defuzzification stage, in order to find the center of gravity (COG), many floating-point multiplications, additions, and one division are required.

In this paper, we propose a new method to use only integer operations in the consequent part and the defuzzification stage. Here, we use an integer pixel grid in the consequent part composed of 400 × 30 pixels. After computing the α value, it is multiplied by 30 and converted to an integer by rounding. Let *Round*(α × 30) be β. The integer β value is transferred to the consequent as a modified degree of fulfillment. Figure 1 shows the proposed fuzzy system. Part (a) of this figure shows the intersection of two conditions. The value α is generated in (b) and this value is used to calculate β in part (c).

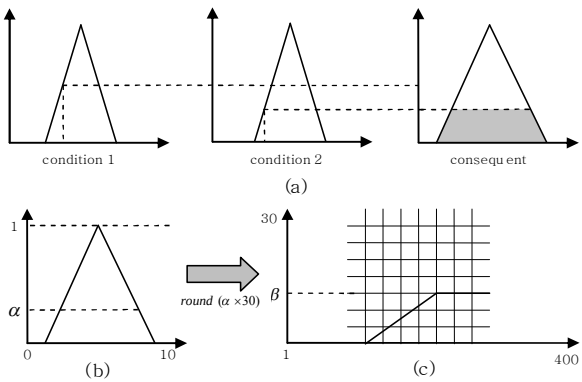



Fig. 1. The entire model of the proposed fuzzy system.

### 2.1 Integer Mapping Algorithm

Integer mapping of the fuzzy membership function is the first step in the proposed algorithm. Fig. 2 shows the shape of a membership function in the consequent part. Assume that the α value is 0.4 and the *x*-interval from 0.0 to 10.0 is quantized into 400 discrete points. In Fig. 2, line (a) has a *y*-interval from 0.0 to 0.4 along the line from point (3.0, 0.0) to point (5.0, 1.0). The equation of line (a) is

$$y = 0.5x - 1.5 \qquad (3)$$

Therefore, in the conventional method, we have to compute the *y*-axis values by incrementing *x* by 0.025 until the *y* value reaches 0.4.

$x$ =3.0 $y$ =0.0
$x$ =3.025 $y$ =0.0125
$x$ =3.05 $y$ =0.025
$x$ =3.075 $y$ =0.0375

$$\begin{array}{cc} \vdots & \vdots \\ x=3.8 & y=0.4 \end{array}$$

This conventional method requires many floating-point additions and many floating-point multiplications. In this paper, however, we use only integer additions without real additions and real multiplications.

We propose a new method to calculate quantized $y$ values in the given line using only integer addition operations. Removing floating point operations produces significant improvement in speed as compared to conventional methods. In this proposed method, we have a $400 \times 30$ integer pixel array. In this grid, a line can be represented by connecting the integer pixels from the starting point pixel to the ending point pixel.

In general, the equation of a line will be represented by $y = ax + b$, where $a$ and $b$ are the slope of the line and the intercept, the value of $y$ when $x = 0$, respectively. In the integer grid pixel, let $(x_0,y_0)$ be the starting point, and $(x_k,y_k)$ be the ending point. Points from $(x_0,y_0)$ to $(x_k,y_k)$ are represented as $(x_0,y_0)$, $(x_1,y_1)$, $(x_2,y_2)$, … $(x_{k-1},y_{k-1})$, and $(x_k,y_k)$, where all of the $x$ and $y$ values are integers. If the absolute value of the slope of the line is less than or equal to 1, i.e. $|a| \le 1$, one can find the next $y$ value by incrementing $x$. This $y$ value is real, so a rounding operation appears to be necessary. To avoid using the *Round* functions and floating point multiplications, we can use a mid-point technique. In this, assume that the slope of a line $a$ satisfies $0 \le a \le 1$, and $(I, J)$ is the current point on the integer pixel grid. If $y$ is the value of a point on the line segment for the value $I$, we know that

$$J \le y + 0.5 \le J + 1.$$

The $x$ coordinate of the next pixel is $I + 1$. Therefore, the integer pixel is either $(I + 1, J)$ or $(I + 1, J + 1)$ [1]. Using this mid-point technique, it is very efficient to compute the next $y$ pixel value at given $x$ pixel value using only integer additions. This procedure is as follows [4].

Let $F(x,y) = ax + by + c = 0$.

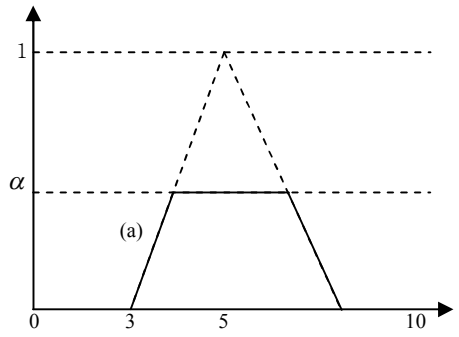



Fig. 2. The shape of a membership function in the consequent part.

If $dy = y_1 - y_0$, and $dx = x_1 - x_0$, then $y = \frac{dy}{dx}x + B$, where $B$ is the y intercept, $(x_0, y_0)$ is the start point, and $(x_1, y_1)$ is the end point.

$\therefore F(x,y) = dy \cdot x - dx\, y + Bdx = 0$, where $a = dy$, $b = -dx$, and $c = Bdx$.

To choose the next pixel, the first midpoint is $(x_p + 1, y_p + ½)$. We need only to compute the value of $F(x_p + 1, y_p + ½)$, and to test its sign. If the sign is positive, we select the point $(x + 1, y + 1)$ as a next pixel. If the sign is negative, we select the point $(x + 1, y)$. If $d$ is the decision variable for this choice, then

$$d_{start} = F(x_p + 1, y_p + ½) = a(x_p + 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) + a + b/2 = a + b/2 = 2dy - dx$$

If $d<0$, then the next pixel is $(x_p + 1, y_p)$, so we select 'E'(East). If $d \ge 0$, then the next pixel is $(x_p + 1, y_p + 1)$, so we select 'NE' (North East).

If 'E' is selected, the next mid-point is incremented by 1 in the $x$ direction. To find the next pixel, we have to compute the amount of variation, $\Delta E$, and update the decision variable $d$.

$$d_{new} = F(x_p + 2, y_p + ½) = a(x_p + 2) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) + 2a + b/2 = 2a + b/2 = 4dy - dx$$
$$d_{old} = F(x_p + 1, y_p + ½) = a(x_p + 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) + a + b/2 = a + b/2 = 2dy - dx$$
$$\Delta E = d_{new} - d_{old} = 2dy$$

If 'NE' is selected, the mid-point is incremented by 1 in the $x$ direction and also incremented by 1 in the $y$ direction. To find the next pixel, we have to compute the amount of variation, $\Delta NE$, and update the decision variable $d$.

$$d_{new} = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$$
$$= (ax_p + by_p + c) + 2a + 3b/2 = 2a + 3b/2 = 4dy - 3dx$$
$$d_{old} = F(x_p + 1, y_p + ½) = a(x_p + 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) + a + b/2 = a + b/2 = 2dy - dx$$
$$\Delta NE = d_{new} - d_{old} = 2(dy - dx)$$

Following the above procedures, we test the sign of the decision variable $d$, and update $\Delta E$ or $\Delta NE$, depending on the choice of the next pixel, incrementing $x$ by 1 in every step. ■

To illustrate how this algorithm functions, consider the construction of a line from point (5,8) to point (9,11). We can find easily the next pixels by using only integer additions. For this example,

$dx = 4$, $dy = 3$,
$d_{start} = 2$, $\Delta E = 6$, $\Delta NE = -2$

Having these values, it is straightforward to find the values of the $y$-axis as incrementing the value of the $x$-axis by 1. Figure 3 shows the procedure of selecting the next pixels.

1. from start point (5, 8), $d = 2$, $d \ge 0$, select 'NE'
2. from point (6, 9), $d = 2 - 2 = 0$, $d \ge 0$, select 'NE'
3. from point (7, 10), $d = 0 - 2 = -2$, $d < 0$, select 'E'
4. from point (8, 10), $d = -2+6 = 4$, $d \ge 0$, select 'NE'
5. (9, 11) ending point

All of the pixels can be found using integer operations in this procedure. Figure 4 shows the algorithm for integer line mapping with start point $(x_1,y_1)$ and end point $(x_2,y_2)$.

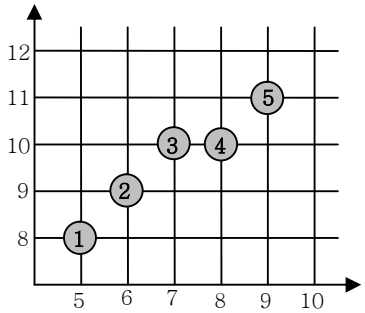


Fig. 3. Example of pixels selection.

Here, β is the modified degree of fulfillment and has an integer value in the range $0 \le \beta \le 30$. The value defuzz(xa) is the value of the membership function in the consequent part when the value of the integer pixel in the *x*-axis is xa [8].

```
procedure left_line                 xa←xa+1
begin                               begin
  dx←x2-x1                           if (d<0)
  dy←y2-y1                             d←d+2dy
  d←2dy-dx                          else
  xa←x1                               ya←ya+1
  ya←y1                               d←d+2(dy-dx)
  a←β                               end
  defuzz(xa)←ya                     defuzz(xa)←ya
  while ya < a+1 do               end
    begin                       end.
```

Fig. 4. Integer mapping algorithm (left_line).

Upon finding the next *y*-coordinate, it is possible to use only integer additions using this algorithm. We simulated and compared this algorithm with the conventional method; the proposed algorithm is much faster than the conventional method. However, results of the proposed method are very little different from the exact real values because of using integer pixels grid. In this paper, as we use the integer pixels of (400, 30), error terms are very small and can be neglected. Increasing the number of integer pixels will decrease the error terms.

In the consequent part, we can represent any arbitrary line having slope not greater than 1. In this paper, as we use the (400, 30) integer pixel grid, all of the membership functions will be mapped onto this integer grid. Next, we describe the mapping procedures of the line (b) and (c), the remaining portions of the membership function, as shown in Fig. 5.

Because the required part in the defuzzification stage is in trapezoidal form, the part from integer β+1 to 30 in the *y*-axis point does not need mapping. Therefore, we must have mappings until the value of *y* is β. In order to map the membership function in Fig. 5, we first process line segment (a), then (c), and finally (b), following the order (a)→(c)→(b). Following this order has the advantage of

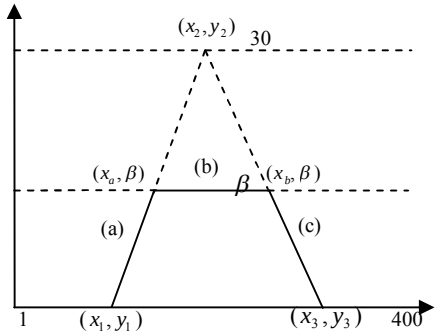


Fig. 5. Example of the representation of the consequent part using β.

calculating the endpoints of segment (b) before it is processed.

Now we describe the integer mapping in line (c). It is similar to the process used to map line (a), except *x* is decremented for each iteration.

Let $F(x,y) = ax + by + c = 0$

$$dy = y_1 - y_0,\ dx = x_1 - x_0,\ y = \frac{dy}{dx}x + B$$

$$\therefore F(x,y) = dy \cdot x - dx \cdot y + Bdx = 0$$

$$a = dy,\ b = -dx,\ c = Bdx$$

To select the next pixel, the first midpoint is $(x_p+1, y_p+½)$. We need only to compute the value of $F(x_p+1, y_p+½)$ and to test its sign. If the sign is positive, we select the point $(x - 1, y)$ as a next pixel. If the sign is negative, we select the point $(x - 1, y + 1)$. If *d* is the decision variable for this choice, then

$$d_{start} = F(x_p - 1, y_p + ½) = a(x_p - 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) - a + b/2 = -a + b/2 \rightarrow -2dy - dx$$

If $d < 0$, then the next pixel is $(x_p - 1, y_p + 1)$, so we select 'NW'. If $d \ge 0$, then the next pixel is $(x_p - 1, y_p)$, so we select 'W'.

If 'NW' is selected, the next mid-point is decremented by 1 in the *x* direction and is incremented by 1 in the *y* direction. To find the next pixel, we compute the amount of variation, $\Delta NW$, and update the decision variable *d*.

$$d_{new} = F(x_p - 2, y_p + 3/2) = a(x_p - 2) + b(y_p + 3/2) + c$$
$$= (ax_p + by_p + c) - 2a + 3b/2 = -2a + 3b/2 \rightarrow -4dy - 3dx$$
$$d_{old} = F(x_p - 1, y_p + ½) = a(x_p - 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) - a + b/2 = -a + b/2 \rightarrow -2dy - dx$$
$$\Delta NW = d_{new} - d_{old} = -2dy - 2dx = -2(dy + dx)$$

If 'W' is selected, the next mid-point is decremented by 1 in the x direction and the *y* value is not changed. To find the next pixel, we compute the amount of variation, $\Delta W$, and update the decision variable *d*.

$$d_{new} = F(x_p - 2, y_p + ½) = a(x_p - 2) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) - 2a + b/2 = -2a + b/2 \rightarrow -4dy - dx$$
$$d_{old} = F(x_p - 1, y_p + ½) = a(x_p - 1) + b(y_p + ½) + c$$
$$= (ax_p + by_p + c) - a + b/2 = -a + b/2 \rightarrow -2dy - dx$$
$$\Delta W = d_{new} - d_{old} = -2dy$$

As in the previous procedure, we test the sign of the decision variable $d$, and update $\Delta W$ or $\Delta NW$, depending on the choice of the next pixel, decrementing $x$ by 1 in every step. ■

Now consider the construction of a line from point (9, 8) to point (5, 11). In this example we can easily determine the next pixels by using only integer additions.

$dx = -4$, $dy = 3$
$d_{start} = -2$, $\Delta W = -6$, $\Delta NW = 2$

With these values, we can find the values of the $y$-axis by decrementing the value of the $x$-axis by 1. Fig. 6 shows the procedure for selecting the next pixels.
1. from start point (9, 8), $d = -2$, $d < 0$, select 'NW'
2. from point (8, 9), $d = -2 + 2 = 0$, $d \geq 0$, select 'W'
3. from point (7, 9), $d = 0 - 6 = -6$, $d < 0$, select 'NW'
4. from point (6, 10), $d = -6 + 2 = -4$, $d < 0$, select 'NW'
5. (5, 11) ending point

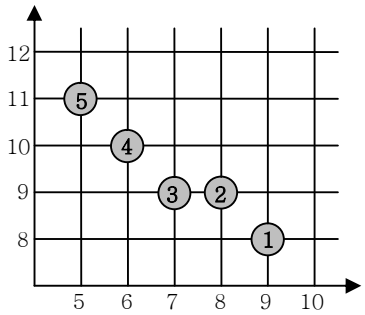



Fig. 6. Selected points in the integer pixels.

All of the pixels can be found using integer operations as shown above. Figure 7 shows the algorithm for integer line mapping with start point $(x_3,y_3)$ and end point $(x_2,y_2)$ [8]. After mapping lines (a) and (c) to the integer pixels, the final task is the mapping of line (b) to the grid. To map line (b), we must set $y = \beta$ for all pixels from point $(x_a,\beta)$ to $(x_b,\beta)$. These two points are the ending points previously computed in lines (a) and (c). Therefore, all y values are set to β in the interval of the x-axis from $x_a$ to $x_b$. An integer mapping algorithm to connect a line from starting point $(x_a,\beta)$ to ending point $(x_b,\beta)$ is shown in Fig. 8.

This integer mapping method offers a significant advantage over the methods that use floating-point operations, as it avoids the floating-point operations. However, this does require a tradeoff. In this algorithm, a quantization error between the real $y$ value and its equivalent integer pixel $y$ value exists because this method selects the nearer integer pixel among two neighboring pixels. This quantization error is inversely proportional to the number of $y$-axis integer pixels. The root mean square value of the quantization error decreases as the number of $y$-axis integer pixels increases [12]. In spite of this drawback, the increase in speed justifies the use of this procedure for application that can tolerate the small error it introduces. In practice, the size of the $y$-axis pixels can be adjusted according to the application domain.

```
procedure right_line
begin
  dx←x3-x2
  dy←y3-y2
  d←-2dy-dx
  xb←x3
  yb←y3
  a←β
  defuzz(xb)←yb
  while yb < a+1 do
    begin
      xb←xb-1
      begin
        if (d<0)
          d←d-2(dy+dx)
          yb←yb+1
        else
          d←d-2dy
      end
      defuzz(xb)←yb
    end
end.
```

Fig. 7. Integer mapping algorithm (right_line).

```
procedure middle_line
begin
  a←β
  x←xa
  while x < xb do
    begin
      x←x+1
      defuzz(x)←a
    end
end.
```

Fig. 8. Integer mapping algorithm (middle line).

### 2.2 Data Structure in the Consequent Part

The required data structure in the consequent part is an integer array that can store the $y$-axis integer values corresponding to the 400 $x$-axis integer pixels in the consequent part. In this array, defuzz($x$), the $y$-axis integer values from 0 to 30 are stored. On computing the consequent part, this array must contain the maximum values in all fuzzy rules. For this, we need another array *maxval*(x). Figure 9 shows the algorithm for this function [8]. Here, $n$ is the number of the fuzzy rules, and $m$ is the number of the pixels in the x-axis.

After computing the consequent part, in general, many values of *maxval(x)* have values of zero continuously from either end of the array. It is not necessary to compute any operations in that interval less than *lower* or greater than *upper* in the defuzzification process. To overcome this problem, we check $x_1$ and $x_3$ for each fuzzy rule, setting *lower* to the minimum of all $x_1$ and *upper* to the maximum of all $x_3$. The lower part of Fig. 9 shows the non-zero item detection algorithm.

### 2.3 New COG Operation without Multiplications

There are many different methods used for defuzzification. Among them, the center of gravity (COG) method is the most common. In conventional fuzzy systems, the COG method requires many additions, many multiplications and one division. In this paper, we also propose an algorithm to reduce the computation for the multiplications by using

```
begin
  lower←400, upper←1
  integer array defuzz[1:m]
  integer array maxval[1:m]←0

  for i := 1 step 1 until n do
   begin
        procedure left_line
        procedure right_line
        procedure middle_line
   for j:=x1 step 1 until x3 do
         if maxval( j ) < defuzz( j )
              then maxval( j ) ← defuzz ( j )
   if (x1 < lower) then lower←x1
   if (x3 > upper) then upper←x3
    end
end
```

Fig. 9. Algorithm of updating *maxval* and non-zero item detection

only integer additions to calculate the nominator part of the COG operation. One integer division operation is still needed to compute the final COG. Figure 10 shows an example of the array for the COG operation. Here, "lower" and "upper" mean the smallest and largest values of the x-pixel excluding the consecutive zeroes from either end of the 0 to 400 range.

The conventional COG operation method is calculated as

$$COG = \frac{\sum_{x=1}^{400} x \cdot f(x)}{\sum_{x=1}^{400} f(x)} = \frac{1\times 0 + \cdots + 175 \times 1 + 176 \times 2 + \cdots + 185 \times 2 + 186 \times 1 + \cdots + 400 \times 0}{0 + \cdots\cdots + 1 + 2 + \cdots + 2 + 1 + \cdots\cdots + 0} = 180.5$$

Here, if we set (upper-lower+1) to *n*, *n* multiplications, 2(*n*-1) additions, and one division operation are required. In this paper, however, we propose a novel method of computing the nominator of the COG. For example, we consider the following case; for all values not shown, *f(x)* = 0.

a b c d e f -- value of *f(x)*
4 5 6 7 8 9 -- value of the *x* pixel

In this case, the values of *lower* and *upper* are 4 and 9, respectively. To calculate the nominator of the COG function, we must compute a×4 + b×5 + c×6 + d×7 + e×8 + f×9. This requires six multiplications and five additions. However, we can arrange this as the following form, and add all terms without multiplications.

a + a + a + a
b + b + b + b + b
c + c + c + c + c + c
d + d + d + d + d + d + d
e + e + e + e + e + e + e + e
f + f + f + f + f + f + f + f + f + f

Since the denominator term is (a + b + c + d + e + f), we need only add the following.

a
b + b
c + c + c
d + d + d + d
e + e + e + e + e
f + f + f + f + f + f
Denominator × (*lower* – 1)

Figure 11 shows the algorithm of COG operation for the proposed method [8].

```
for i := upper step = –1 until lower
  begin
    temp ← maxval( i ) + temp
    sum ← sum + temp
  end
COG ← sum / temp
COG ← COG + (lower – 1)
```

Fig. 11. The proposed algorithm for COG operations

Table 1 shows the computation procedures of the iteration loop for Fig. 11.

For the data in Fig. 10 and Table 1, the algorithm calculates,

$$COG \leftarrow \frac{sum}{temp} = \frac{247}{38} = 6.5$$

$$COG \leftarrow COG + (lower - 1) = 6.5 + (175 - 1) = 180.5$$

The final "temp" value in Fig. 11 becomes the denominator.

| 0 | 0 | 0 | … | 0 | 1 | 2 | 3 | 4 | 4 | 5 | 5 | 4 | 4 | 3 | 2 | 1 | 0 | … | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | … | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | … | 398 | 399 | 400 |
| | | | | | ↑ | | | | | ↑ | | | | | | ↑ | | | | | |
| | | | | | lower | | | | | COG | | | | | | upper | | | | | |
| | | | | | | | | | | 180.5 | | | | | | | | | | | |

Fig. 10. An example of an integer array for COG operations.

Table 1. Computation procedures of the iteration loop

| *i* | *maxval(i)* | *temp* | *sum* |
|---|---|---|---|
| 186(upper) | 1 | 1 | 1 |
| 185 | 2 | 3 | 4 |
| 184 | 3 | 6 | 10 |
| 183 | 4 | 10 | 20 |
| 182 | 4 | 14 | 34 |
| 181 | 5 | 19 | 53 |
| 180 | 5 | 24 | 77 |
| 179 | 4 | 28 | 105 |
| 178 | 4 | 32 | 137 |
| 177 | 3 | 35 | 172 |
| 176 | 2 | 37 | 209 |
| 175(lower) | 1 | 38 | 247 |

In this method, for $n$ non-zero items, only $2 \times n$ additions and 1 division are required. Table 2 compares the conventional method without considering the non-zero items, the conventional method considering the non-zero items, and the proposed method. Note that the proposed method does not require any multiplications. Here, $1 \leq lower \leq upper \leq 400$.

### 2.4 Execution Speed Analysis and Simulation

In this paper, we proposed a novel method of computing with only integer operations in the consequent part and defuzzification algorithm without multiplications. This system gets higher speed performance compared to the conventional methods. However, very small errors are introduced. Here, we compare and analyze the execution times in one fuzzy inference cycle and the precision degree of COG between the proposed method and the faster algorithm [13] among the conventional methods. The target object is an air conditioner control system. Figure 12 shows the membership functions and fuzzy rules of the system. In this case, nine fuzzy rules are used.

For a temperature of 17°C and a relative humidity of 32%, the COG is calculated using Eq. (1) as 24.73 [13]. We have tested our method with grid sizes of 400×30 and 800×30 pixels. Results are summarized in the Table 3.

Table 2. Comparison of arithmetic operations in Center of Gravity calculations

| | Conventional method without considering non-zero items | Conventional method considering non-zero items | Proposed method |
|---|---|---|---|
| additions | $2 \times (400 - 1)$ | $2 \times (upper - lower + 1)$ | $2 \times (upper - lower + 1)$ |
| multiplications | 400 | $upper - lower + 1$ | 0 |
| divisions | 1 | 1 | 1 |

As we can see in Table 3, COGs calculated using the method presented in [13] and the proposed method are 24.77 and 24.83 for the 400×30 pixel array, respectively. For the 800×30 array, the COGs in [13] and the proposed method are 24.75 and 24.80, respectively.

Table 3. Comparison of execution time and COGs at 17°C and 32% relative humidity

| | (400×30) pixels | | (800×30) pixels | |
|---|---|---|---|---|
| | execution time(μs) | COG | execution time(μs) | COG |
| integration | - | 24.73 | - | 24.73 |
| [13] | 2.70 | 24.77 | 3.67 | 24.75 |
| proposed | 0.21 | 24.83 | 0.29 | 24.80 |

The errors in the proposed method are 0.1 (400×30) and 0.07 (800×30), or about 0.4% and 0.28%, respectively. However, the execution speed of the proposed method is over 12 times faster than that in [13]. It is noted that if we increase the pixel size, execution time is increased and the error is reduced even more.

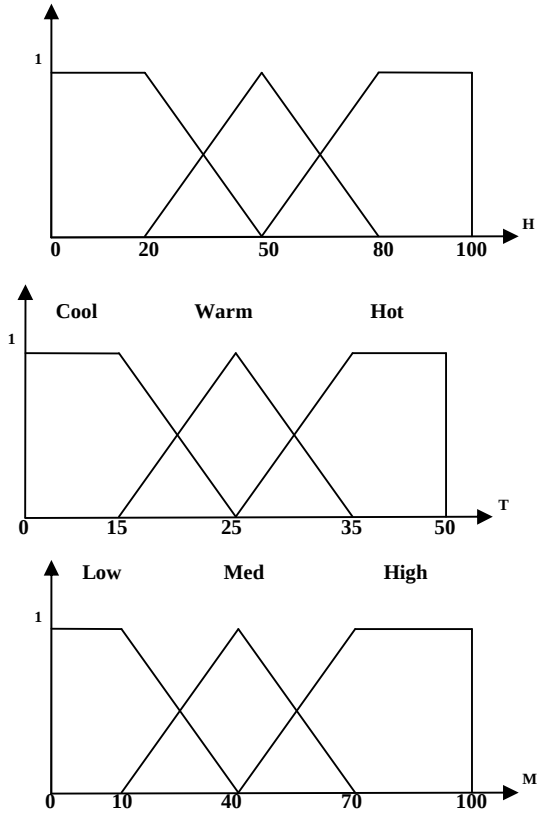
To evaluate the performance of the proposed system, it is also simulated on the truck backer-upper control problem. The goal of this problem is to back a truck to a loading dock as quickly and precisely as possible. This type of the problem is a typical non-linear control system that cannot be solved using conventional control methods. Fig. 13 shows the model of the truck backer-upper control [6,15], initial and goal positions of the truck. Here, $x$ is the position of the truck in the $x$-axis, and $\phi$ is the angle between the $x$-axis and the truck's direction. The truck position is determined by the three state variables $x$, $y$ and $\phi$. For simplicity, we consider only the $x$-axis direction in this simulation; only backing up is considered. The tracking control for the truck is the angle $\theta$, the angle between the truck's direction and the axis of the front wheel. In this problem, a fuzzy rule is of the form

IF $x$ =Small and $\phi$ =Medium THEN $\theta$ =Large

Figure 14 shows the membership functions of the condition part 1($x$) and the condition part 2($\phi$), the membership function of the consequent part ($\theta$), and fuzzy rules [6]. In this figure, VS, SM, MS, ME, ML, LA, and VL are Very Small, SMall, Medium Small, MEdium, Medium Large, LArge, and Very Large, respectively.

The procedure of the fuzzy inference is as follows.

1. Using the value of the initial $x$ and $\phi$, inference and defuzzify to find $\theta$.
2. Using the control kinetics as shown in equations (4) and (5), shown below, find the $x$ and $\phi$ values for the next stage.
3. Repeat steps (1) and (2) until the goal is reached.




**Temperature**

| Relative Humidity | Cool | Warm | Hot |
|---|---|---|---|
| Dry | Low | Med | High |
| Moist | Low | Med | High |
| Wet | Med | High | High |

Fig. 12. Membership functions and fuzzy rules

$$x(t+1) = x(t) + \cos[\phi(t) + \theta(t)] + \sin[\theta(t)]\sin[\phi(t)] \quad (4)$$

$$\phi(t+1) = \phi(t) - \sin^{-1}[2\sin(\theta(t))/b] \quad (5)$$

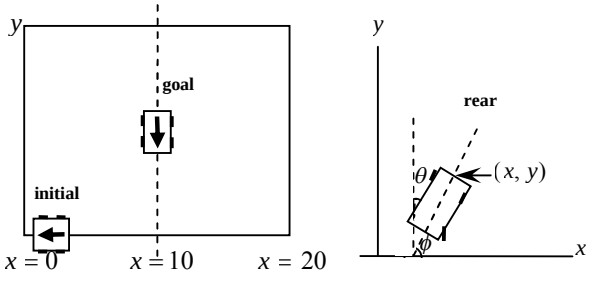
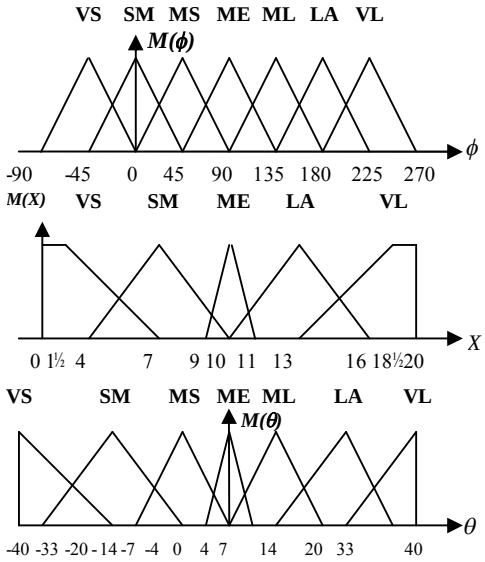



Fig. 13. Truck backer-upper control system.




| φ \ x | VS | SM | ME | LA | VL |
|---|---|---|---|---|---|
| VS | SM | VS | | | |
| SM | SM | VS | VS | VS | |
| MS | ML | MS | SM | VS | SM |
| ME | LA | LA | ME | SM | SM |
| ML | LA | VL | LA | ML | SM |
| LA | | VL | VL | VL | MS |
| VL | | | | VL | LA |

Fig. 14. Membership functions and fuzzy rules of the truck backer-upper control.

In Eq. (5), $b$ is the length of the truck. For this simulation, we set $b$ to 4. We set the initial values as $x$ = 1.0 and $\phi$ = 0.0°, and $x$ = 10.0, that is, the goal of the $x$ coordinate. Table 4 and Fig. 15 show the comparison of the conventional method and the proposed method. Here, the conventional method represents the system that uses real valued operations in [0, 1]. In Table 4, the real values in the right-hand side are converted from the 400 × 30 integer grid for comparison.

As shown in Table 4, there is little error between the proposed system and the conventional method that uses real operations. This is caused by using the integer operations in the consequent part and the defuzzification stage. The error is about 0.4%. This error term is very small and can be neglected in many applications. The more important thing is the notable improvement of the total speed. The proposed system is over 10 times faster than the conventional method. Therefore, the proposed system can be well applied to the high-speed intelligent systems such as the control of a robotic arm.

Table 4. Simulation results

| Conventional method | | | | Proposed method | | | |
|---|---|---|---|---|---|---|---|
| | $x$ | $\phi$ | $\theta$ | | $x$ | $\phi$ | $\theta$ |
| 0 | 1.00 | 0.00 | -19.00 | 0 | 1.00 | 0.00 | -18.88 |
| 1 | 1.95 | 9.37 | -17.95 | 1 | 1.93 | 9.42 | -17.97 |
| 2 | 2.88 | 18.23 | -16.90 | 2 | 2.81 | 18.28 | -16.77 |
| 3 | 3.79 | 26.59 | -15.85 | 3 | 3.74 | 26.62 | -15.64 |
| 4 | 4.65 | 34.44 | -14.80 | 4 | 4.51 | 34.48 | -14.80 |
| 5 | 5.45 | 41.78 | -13.75 | 5 | 5.48 | 41.81 | -13.36 |
| 6 | 6.18 | 48.60 | -12.70 | 6 | 6.19 | 48.65 | -12.61 |
| 7 | 7.48 | 54.91 | -11.65 | 7 | 7.45 | 54.95 | -11.42 |
| 8 | 7.99 | 60.71 | -10.60 | 8 | 7.97 | 60.74 | -10.48 |
| 9 | 8.72 | 65.99 | -9.55 | 9 | 8.69 | 66.04 | -9.51 |
| 10 | 9.01 | 70.75 | -8.50 | 10 | 8.97 | 70.80 | -8.43 |
| 11 | 9.28 | 74.98 | -7.45 | 11 | 9.24 | 75.02 | -7.44 |
| 12 | 9.46 | 78.70 | -6.40 | 12 | 9.43 | 78.73 | -6.28 |
| 13 | 9.59 | 81.90 | -5.34 | 13 | 9.55 | 81.95 | -5.29 |
| 14 | 9.72 | 84.57 | -4.30 | 14 | 9.69 | 84.61 | -4.32 |
| 15 | 9.81 | 86.72 | -3.25 | 15 | 9.79 | 86.77 | -3.19 |
| 16 | 9.88 | 88.34 | -2.20 | 16 | 9.85 | 88.37 | -1.98 |
| 17 | 9.91 | 89.44 | 0.00 | 17 | 9.89 | 89.50 | 0.02 |

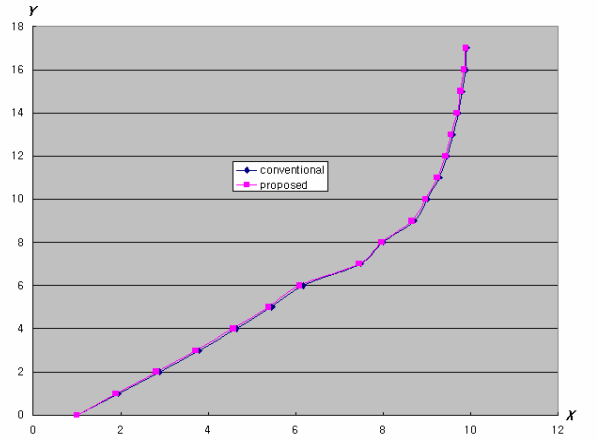



Fig. 15. Trace of truck trajectory

## 3. PERFORMANCE ANALYSIS

To analyze the performance of the proposed method, we simulated it using both high-level programs and VHDL synthesis. First, algorithms for both methods were coded in C/C++ and execution speeds were timed. By using only integer operations, the proposed algorithm executed approximately 12.75 times faster than the traditional algorithm that uses floating point operations.

We also developed a VHDL implementation of the proposed algorithm and compared its performance to that of the traditional algorithm. For this implementation, the proposed algorithm reduced the time needed to calculate system outputs by an order of magnitude [8]. Both the software and VHDL simulation results are consistent with the speedup achieved for the truck backer-upper system discussed in the previous section.

Table 5 shows the comparison results of each fuzzy hardware system. FLASP and the proposed system have flexible bit-width for each fuzzy variable, while the other systems in this table support only fixed bit lengths. The KAFA system supports all of the fuzzy operators such as fuzzy logical product, logical sum, algebraic product, algebraic sum, bounded product, bounded sum, drastic product, and drastic sum. However, most of the fuzzy systems generally use the fuzzy logical product (Min operator) and logical sum (Max operator) only. In the defuzzification procedure, the FLASP system and Aranguren's controller use a pipelining technique. In the computation of each membership function, we note that the existing algorithms use either lookup tables or floating point operations. Only the proposed algorithm uses integer operations.

Finally, we note that it is possible to improve the performance of the proposed algorithm by utilizing parallel processing techniques. For example, the functions for line segments (a) and (c) in Fig. 5 are independent and can be calculated in parallel. In addition, the points on line segment (b) in that figure are also independent of each other and do not have to be calculated sequentially. For membership functions that are symmetric, it is also possible to calculate only the points on line segment (a) and then make use of symmetry to generate the points on (c) without significant computations.

## 4. CONCLUSIONS

Until now, several methods of high-speed fuzzy hardware modules and their parallelizing methods to process large amount of fuzzy data have been proposed. However, most of the conventional methods, despite having fast computing hardware, use much time by using floating point operations in the condition part, in the consequent part, and in the defuzzification stage. Generally, in fuzzy computing, computations require the most time in the consequent part and the defuzzification stage rather than the condition part, so specially designed hardware modules and fast algorithms are required to process data rapidly in these stages.

In this paper, we have proposed the following.

(1) A method of the integer pixel mapping

We have proposed a novel method of mapping the membership functions to an integer grid in the consequent part. Here, we only use integer addition operations to compute the next position of the pixel.

(2) A method of eliminating the zero items in the consequent part

In the stage of the computing the value of the COG, we use only non-zero items of the membership functions in the consequent part. To do this, we have proposed a method of

Table 5. Comparison results by each hardware system [11]

| | **Microcontroller 68HC12** | **FLASP [14]** | **KAFA [7]** | **FZP-0401A [17]** | **Aranguren's Controller [2]** | ***Proposed system*** |
|---|---|---|---|---|---|---|
| Max freq. | 8 MHz | 15 MHz | 10 MHz | 25 MHz | 12 MHz | 40 MHz |
| Bitwidth of variables | Fixed (8 bits) | variable | fixed | Fixed (16 bits) | Fixed (8 bits) | variable |
| Fuzzy operator | Min/Max | Min/Max | various | Product/sum | Min/Max | Min/Max |
| Defuzzification | Serial process. | Pipelining | Parallel process. | Serial process. (COG) | Pipelining (COA) | No multiplications (COG) |
| Rule reconfiguration | Compile time | Run-time | Compile time | Compile time | Compile time | Compile time |
| Parallel structure | No | yes | no | no | yes | yes |
| HW/SW codesign | No | yes | no | no | yes | yes |
| Membership function computing | LUT (Lookup table) | LUT | Floating point operations | Floating Point operations | LUT | Only Integer Addition operations |

finding the starting point (*lower*) and the last point (*upper*) of the non-zero items.

(3) A new COG method

We have proposed a method to compute the COG without multiplications. Only integer additions and one integer division are required.

The proposed method was simulated for the truck backer-upper control system to verify its performance as compared to the conventional systems. As the results of the simulation, the proposed system is 12.75 times faster than the conventional method for the classic truck backer-upper problem and an order of magnitude faster on general fuzzy control problems. The minimal error introduced by the integer representation can be accepted by a wide variety of fuzzy control applications.

The proposed system can be applied to the ground cover map, an environmental control and GIS (Geological Information System) system that requires a fast processing time with large volumes of fuzzy data. This system can also be applied to build powerful architectures for control applications, such as robotic control, with time-critical sensor integration.

## REFERENCES


[1] E. S. Angel, *Interactive Computer Graphics, 3rd ed.*, Addison-Wesley, Boston, MA, USA, 2002.

[2] G. Aranguren, et. al., "Hardware implementation of a pipeline fuzzy controller," *Fuzzy Sets and Systems* 128 (2002) 61-79.

[3] J. E. Bresenham, "Incremental line compaction," *The Computer Journal* (1982) 116-120.

[4] J. D. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice, 2nd ed.*, Addison-Wesley, Boston, MA, USA, 1990.

[5] A. M. Ibrahim, *Fuzzy Logic for Embedded Systems Applications*, Elsevier, 2004.

[6] D. Kim and I.H. Cho, "An accurate and cost-effective COG defuzzifier without the multiplier and the divider," *Fuzzy Sets and Systems* 104 (1999) 229-244.

[7] Y. D. Kim, "High speed flexible fuzzy hardware for fuzzy information processing," *IEEE Trans. Systems, Man, and Cybernetics – Part A* 27 (1) (1997) 45-56.

[8] S. G. Lee and J. D. Carpinelli, "VHDL Implementation of Very High-speed Integer Fuzzy Controller," *Proc. 2005 IEEE Int. Conf. on Systems, Man and Cybernetics*, pp. 588-593, Honolulu, HI, USA, Oct. 2005.

[9] T-H S. Li and S-J Chang, "Autonomous Fuzzy Parking Control of a Car-Like Mobile Robot," *IEEE Trans. on Systems, Man and Cybernetics - Part A*, Vol. 33, No. 4, pp. 451-465, Jul. 2003.

[10] C-T Lin and C. S. G. Lee, *Neural Fuzzy Systems*, Prentice-Hall, 1996.

[11] E. H. Mamdani, "Application of fuzzy algorithms for control of a simple dynamic plant," *Proc. IEE* 121 (12) (1974) 1585-1588.

[12] F. Marcelloni and M. Aksit, "Fuzzy logic-based object-oriented methods to reduce quantization error and contextual bias problems in software development," *Fuzzy Sets and Systems* 145 (2004) 57-80.

[13] J. J. Saade and H. B. Diab, "Defuzzification Techniques for Fuzzy Controllers," *IEEE Trans. Systems, Man, and Cybernetics – Part B*, 30 (1) (2000) 223-229.

[14] Z. Salcic, "High-speed customizable fuzzy-logic processor: architecture and implementation," *IEEE Trans. Systems, Man, and Cybernetics – Part A* 31 (6) (2001) 731-737.

[15] L. Xin and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. System, Man, and Cybernetics* 22 (6) (1992) 1414-1427.

[16] J. Yen and R. Langari, *Fuzzy Logic: Intelligence*, Prentice Hall, Englewood Cliffs, NJ, USA, 1999.

[17] N. Yubazaki, M. Otani, et. al., "Fuzzy inference chip FZP-0401A based on interpolation algorithm," *Fuzzy Sets and Systems* 98 (1998) 299-310.

[18] L. A. Zadeh, "Fuzzy Sets," *Information and Control* 8 (1965) 338-353.

[19] Z. Zhang and J. Chang, "A fuzzy control algorithm with high controlling precision," *Fuzzy Sets and Systems*, (140), pp. 375-385, 2003.

**Sang Gu Lee** received the B. Engr. Degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1978, and the M. Engr. Degree in Computer Science from KAIST, Korea in 1981. He received the Ph.D. degree in Electrical Electronics and Computer Engineering from Waseda University, Tokyo, Japan. Since 1983, he has been a professor in the Department of Computer Engineering at the Hannam University, Korea. He was a visiting Professor in the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology in 2004-2005. His current research interests are in intelligent system, fuzzy computing, parallel processing, parallel architecture and embedded systems.

**John D. Carpinelli** received the B. Engr. degree from Stevens Institute of Technology in 1983, and the M. Engr. and Ph. D. degrees from Rensselaer Polytechnic Institute in 1984 and 1987, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology. He has served as a member of the Governing Board for the Gateway Engineering Education Coalition and is the author of the textbook *Computer Systems Organization and* Architecture. His research interests include computer architecutre, engineering education, and interconnection networks. Prof. Carpinelli is a senior member of the IEEE and a member of the ASEE.