논문 2006-43CI-2-6

지능 시스템을 위한 퍼지 후건부 및 비퍼지화 단계의 고속 정수연산

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

이 상 구*, 채 상 원**

(Sang Gu Lee and Sang Won Chae)

요 약

지능 시스템에 사용되는 퍼지 데이터를 고속으로 처리하기 위한 퍼지 제어시스템의 중요한 문제점들 중의 하나는 퍼지 추론 및 비퍼지화 단계에서의 수행속도의 개선이다. 특히 후건부의 계산 및 비퍼지화 단계에서의 고속 연산이 더욱 더 중요하다. 따라서 본 논문에서는 지능 시스템을 위한 퍼지 제어기의 속도향상을 위해 후건부 및 비퍼지화 단계에서 [0,1]의 실수 연산을 하지 않고, 퍼지 소속함수의 값을 정수형 격자 (400×30)에 매핑시켜 고속의 정수 덧셈 연산만으로 수행할 수 있는 알고리듬 및 비퍼지화 단계에서 곱셈이 필요 없는 새로운 알고리듬을 제안하고, truck backer-upper 제어시스템에 적용하여 기존의 방법보다 매우 빠른 실시간 고속 퍼지 시스템을 보여준다. 본 논문에서 제안한 시스템은 로봇의 팔 움직임 제어와 같은 실시간 고속 지능 시스템에 잘 활용될 수 있다.

Abstract

In a fuzzy control system to process fuzzy data in high-speed for intelligent systems, one of the important problems is the improvement of the execution speed in the fuzzy inference and defuzzification stages. Especially, it is more important to have high-speed operations in the consequent part and defuzzification stage. Therefore, in this paper, to improve the speedup of the fuzzy controllers for intelligent systems, we propose an integer line mapping algorithm using only integer addition to convert [0,1] real values in the fuzzy membership functions in the consequent part to integer grid pixels (400×30). This paper also shows a novel defuzzification algorithm without multiplications. Also we apply the proposed system to the truck backer-upper control system. As a result, this system shows a real-time very high speed fuzzy control as compared as the conventional methods. This system will be applied to the real-time high-speed intelligent systems such as robot arm control.

Keywords : Intelligent system, Integer operation, Defuzzification, Truck backer-upper control system

I. 서 론

퍼지 이론은 과거에는 퍼지 제어를 중심으로 시스템 제어, 공학분야에 국한되어 활용되어 왔으나 현재에는 데이터 마이닝, 전문가시스템, 패턴인식등 새로운 응용분야에서도 퍼지 이론의 효용성을 보이고 있으며 퍼지 이론을 기반으로 설계되는 퍼지 시스템은 인간의 언어적 개념을 정량적 수치로 표시할 수 있다는 장점 때문에 지능 시스템 및 소프트 컴퓨팅 등의 공학적 기술로서 널리 응용되고 있다[1].

퍼지 논리는 기존의 부울 논리를 확장한 것으로 특정 집합 A에 대하여 구성원소의 소속 정도를 0 또는 1의 이진정보가 아닌 0과 1사이의 실수로 나타낸다. 이러한 퍼지값은 퍼지 추론과 비퍼지화 과정을 거쳐 시스템에서 요구하는 제어값이 된다. 퍼지 추론은 외부에서 입력되는 조건부의 퍼지 정보에 대해 각각의 소속함수를 통해 소속정도를 구하고 이 소속정도들에 퍼지 제어규칙을 적용하여 적합도를 구한다. 개개의 제어규칙에서

* 정회원, ** 학생회원, 한남대학교 컴퓨터공학과
(Department of Computer Eng., Hannam University)
접수일자: 2005년11월29일, 수정완료일: 2006년3월3일

얻어진 추론결과들에 대한 비퍼지화를 통해 제어값을 구하게 된다. 입력되는 퍼지 정보의 소속 정도는 정수값이 아닌 0과 1사이의 실수로 표현되기 때문에 이러한 추론 과정을 거치는 동안 퍼지 제어기는 많은 양의 실수연산을 필요로 한다[2]. 소속함수나 퍼지 제어규칙의 수가 많아질수록 연산량은 더욱 증가하고 그만큼 많은 실수 연산을 하게 된다.

일반적으로 퍼지 시스템에서는 전건부에서 각각의 퍼지 규칙에 대한 α값(degree of fulfillment)을 계산하는데에는 그리 많은 계산을 필요로 하지 않는다. 그러나, 후건부의 계산 및 무게중심(center of gravity)을 계산하는 비퍼지화(defuzzification) 단계에 필요한 많은 양의 실수 연산을 필요로 하게 된다. 실제로 무게중심을 구하는 공식은 다음의 식 (1)과 같다.

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

그러나 식(1)은 적분을 하여야 하므로 많은 시간이 소요되고, 보통의 경우는

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

식 (2)와 같이 후건부를 많은 수의 일정한 간격으로 나누어 양자화시켜 덧셈을 계산한다. 이러한 많은 수의 양자화 레벨을 실수로 계산하기 위해서는 후건부에서 x축(discourse of universe)을 일정한 간격으로 나누어 많은 수의 실수연산을 하여야 한다[3].

이러한 연산을 위한 종래의 방법들을 간단히 요약하면 다음과 같다. 68HC12와 같은 마이크로컨트롤러 시스템에서는 LUT(Lookup table)의 방법을 사용한다. LUT 방법은 메모리의 크기를 줄일 수는 있지만, 퍼지 데이터 항목들을 계산할 때 실수 연산의 보간 과정이 필요하게 된다. FLASP[4] 시스템에도 LUT 방법을 사용하였고, 시스템의 속도향상을 위해서 파이프라인 방법을 도입하였다. KAFA[5] 시스템에서는 실수연산 방법을 적용하였고, 비퍼지화 단계에서 병렬처리의 방법을 사용하였다. FZP-0401A[6] 시스템에서도 실수연산 방법을 적용하였다. Aranguren[7]은 LUT의 방법을 사용하였고, 비퍼지화 단계에서 파이프라인 방법을 적용하였다. 이와 같이 종래의 대부분의 퍼지제어기들은 LUT 또는 실수의 연산을 적용하여 퍼지 추론 및 비퍼지화 단계를 행한다.

따라서 본 논문에서는 후건부에서 많은 양의 실수 연산으로 인한 퍼지 연산의 속도 저하 문제를 근본적으로 해결하기 위해 후건부의 퍼지 소속함수 그래프를 정수형 격자에 매핑하여 정수 덧셈연산만으로 다음의 정수의 격자의 좌표를 계산할 수 있는 새로운 알고리듬을 제안하여, 후건부에서 고속으로 정수연산만을 수행하고, 이를 바탕으로 비퍼지화 단계에서도 정수의 덧셈연산을 사용하여 무게중심을 구하는 알고리듬을 제안하려고 한다. 무게중심을 구하는 알고리듬은 일반적으로 많은 수의 곱셈과 덧셈, 한번의 나눗셈이 필요하지만, 본 논문에서 제안하는 방법은 덧셈연산들과 한 번의 나눗셈만을 사용한다. 따라서 본 논문에서 제안하는 방법은 기존의 실수형 연산에 비해 훨씬 좋은 성능을 얻을 수 있다. 본 논문에서의 전건부 및 후건부의 소속함수들은 GUI(Graphic User Interface)방법에 의해 삼각형 모양의 소속함수를 사용하며, 삼각형의 꼭지점(vertex) 3개를 GUI시스템에 입력하여 소속함수의 값들을 내부에서 계산하는 방법을 사용한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 후건부 및 비퍼지화 단계에서의 정수형 연산 알고리듬을 제안하고, 후건부에서의 정수 매핑 알고리듬, 비퍼지화 단계에서의 자료구조 및 정수덧셈 형태의 COG 연산에 대해 설명하고, Ⅲ장에서는 후건부에서의 정수형 연산과 실수형 연산의 오차분석 및 수행시간 분석을 하고, Ⅳ장에서는 본 논문의 성능평가를 위하여 truck backer-upper 제어 시스템에 적용하고, Ⅴ장에서는 결론 및 향후 연구 과제를 제시한다.

Ⅱ. 후건부 및 비퍼지화 단계에서의 정수형 연산 알고리듬

퍼지 소속함수의 그래프는 대부분 삼각형 또는 사다리꼴의 형태를 취하고 있다. 이 그래프들은 삼각형 또는 사다리꼴의 정점(vertex)을 연결하는 직선들의 집합이다. 퍼지 추론시에 전건부가 삼각형 또는 사다리꼴의 함수 형태에서 퍼지 입력이 단일값으로 들어올 때, 각 규칙마다 α값을 구하는 것은 그리 많은 양의 계산을 필요로 하지 않는다. 그러나 후건부의 연산에서는 각각의 퍼지 규칙에 대해 모든 x축 값에 대한 y축의 실수값을 연산해야 하므로 많은 양의 실수의 연산을 필요로 한다. 또한 비퍼지화 단계에서 무게중심을 구하는 단계

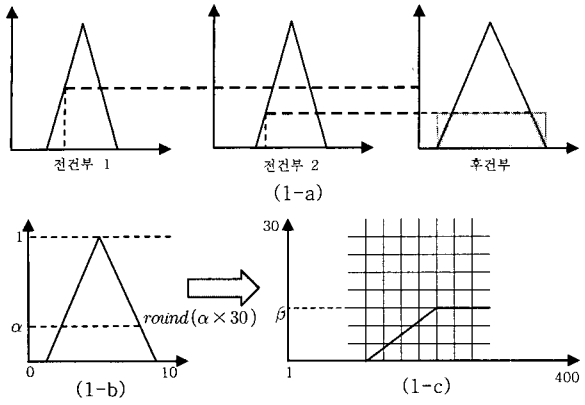


그림 1. 제안하는 퍼지 시스템
Fig. 1. The proposed fuzzy system.

에서는 많은 양의 실수의 곱셈 및 나눗셈의 연산이 필요하게 된다[8].

본 논문에서 새로이 제안하는 알고리듬은 후건부의 연산 및 비퍼지화 단계에서 모든 x축에 대한 y값을 구할 때 기존의 방법인 실수 연산을 전혀 사용하지 않고 후건부를 정수형 격자에 매핑시켜 정수의 덧셈 연산만으로 그래프의 값을 얻어낼 수 있는 방법으로 기존의 방법보다 속도면에서 매우 빠르다.

본 본문에서는 정수형 매핑 알고리듬을 사용하여 [0,1]의 실수형의 소속 함수를 가장 가까운 정수형 격자에 매핑시켜 정수값 만을 갖도록 한다. 후건부의 모양을 계산할 때 후건부의 삼각형 또는 사다리꼴의 모양의 각 정점을 잇는 직선 위에 위치한 정수형 격자점들을 연결하여 표현할 수 있다. 이 과정을 그림 1에 나타낸다.

각각의 규칙에 대해 전건부에서의 적합도(α값, degree of fulfillment)가 계산되면, 후건부에서는 삼각형 또는 사다리꼴의 각 점을 계산하기 위해 가로 400, 세로 30개를 갖는 정수 격자 좌표에 대응시킨다. 전건부에서 α값(실수)이 구해지면 $round(\alpha \times 30)$을 계산한다. 이 값을 β(정수)라 하면, β값이 후건부의 적합도로서 입력된다. 이 때, 그림 1-c에서 후건부의 그래프에 대응되는 격자점들을 계산할 때 다음과 같은 정수 덧셈만으로 계산함으로서 매우 빠르게 계산할 수 있다.

1. 후건부에서의 정수 매핑 알고리듬

후건부에서 정수형 덧셈만을 사용하여 정수 격자에 매핑하는 과정을 설명하기 위해 그림 2의 예를 살펴본다.

α값이 0.4이고, 실수 연산시에 x구간 [0.0,10.0]을 400

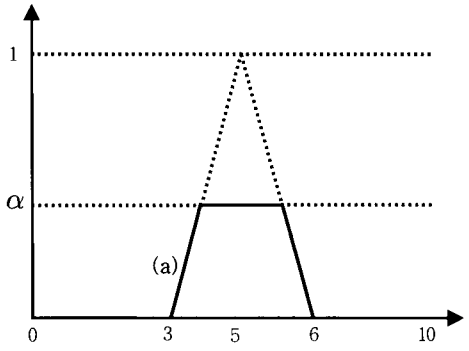


그림 2. 후건부에서의 소속함수의 예
Fig. 2. Example of a membership function in the consequent part.

개로 양자화하여 계산한다고 가정하자. 직선(a) 부분은 (3.0, 0.0)과 (5.0, 1.0)을 지나는 직선에서 y구간이 [0.0, 0.4]인 부분이다. 따라서 직선의 식은 다음과 같다.

$$y = 0.5x - 1.5 \tag{3}$$

따라서 식 (3)의 y의 구간 [0.0, 0.4]에서 x를 0.025 단위로 증가시켜 계산하면 된다.

$$\begin{array}{ll} x = 3.0 & y = 0.0 \\ x = 3.025 & y = 0.0125 \\ x = 3.05 & y = 0.025 \\ x = 3.075 & y = 0.0375 \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ x = 3.8 & y = 0.4 \end{array}$$

이와 같은 종래의 연산방법은 매우 많은 실수의 덧셈, 곱셈 시간이 필요하다.

그러나, 본 논문에서 제안하는 방법은 실수의 곱셈이 아닌 정수형 덧셈만을 사용하여 소속함수 그래프의 값들을 얻어내므로 기존의 방법에 비해 빠른 연산을 할 수 있다.

즉, 실수형의 소속함수를 정수형 격자에 매핑시켜 정수형의 값을 갖도록 한다. 격자좌표에서 직선은 시작점의 좌표와 끝점의 좌표를 잇는 직선 위에 위치한 격자점들을 이어서 표현된다. 일반적으로 직선의 방정식은 기울기와 y축에 대한 접점을 이용하여 $y = ax + b$로 표현한다. a는 직선의 기울기이고 b는 y절편이다. 시작점을 (x_0, y_0), 끝점을 (x_k, y_k)라고 한다면 두 점 사이의 점들을 시작점부터 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$……,

(x_{k-1}, y_{k-1})이다. 여기서 x, y의 값은 정수이다. 이 직선의 기울기가 양수이며 1보다 작을 경우에는 x의 값을 x_0에서 1씩 증가하면서 이에 대응되는 y값을 구하는데 이 때 y값은 실수를 갖게 된다. 이렇게 구한 y값을 단순히 반올림하여 사용한다면 실수연산이 필요하므로 매우 비효율적이다. 이러한 문제점을 해결하기 위해 본 논문에서는 다음과 같은 알고리듬을 사용한다.

기울기가 양수이며 1보다 작을 경우 $x_{k+1}(=x_k+1)$에서의 y좌표 값은 $y=a(x_{k+1})+b$을 이용하여 계산을 하면 y값은 정수가 아닌 실수 값을 가질 수도 있다. 그러나 격자의 좌표는 정수이므로 y의 값에 따라 적당한 정수의 값을 선택하는데 정수 좌표 선택 방법론으로 임의의 좌표 x_k에 해당하는 y좌표, 즉 y_k값을 이용한다. x가 x_k에서 $x_k+\frac{1}{2}$의 y값이 y_k에 가까운지 y_k+1에 가까운지를 판단하여 만약 y_k쪽에 가깝다면 x_k+1에서의 y좌표는 y_k로, 그렇지 않고 y_k+1에 가깝다면 x_k+1에서의 y좌표는 y_k+1을 선택한다[9]. 이러한 중앙점(midpoint)들의 개념을 사용하여 덧셈연산만으로 다음 x격자 좌표의 y값(정수)들을 효율적으로 연산하는 구체적인 알고리듬은 다음과 같다.

일반적인 직선의 방정식의 식에서 다음과 같이 표현할 수 있다.

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

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

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

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

좌표 결정을 위해 중앙점을 사용하며 첫 번째 중앙점은 $(x_p+1, y_p+\frac{1}{2})$이 된다.

$$d_{start}=F(x_p+1,y_p+\frac{1}{2})=a(x_p+1)+b(y_p+\frac{1}{2})+c$$

$$=(ax_p+by_p+c)+a+\frac{b}{2}=a+\frac{b}{2}\rightarrow 2dy-dx$$

(d : Decision Variable)

$d<0$ 일때 'E' (x_p+1, y_p), $d\geq 0$ 일때 'NE' (x_p+1, y_p+1)을 선택

'E'를 선택 하였을 경우 x값이 증가할 때 y값은 증가되지 않고 그대로 유지되고, 'NE'를 선택 하였을 경우 x값이 증가할 때 y값도 증가된다.

-'E' 가 선택 되었을 때 ΔE의 값을 구하여 d값에 더해준다.

$$d_{new}=F(x_p+2,y_p+\frac{1}{2})=a(x_p+2)+b(y_p+\frac{1}{2})+c$$

$$=ax_p+by_p+c+2a+\frac{b}{2}=2a+\frac{b}{2}\rightarrow 4dy-dx$$

$$d_{old}=F(x_p+1,y_p+\frac{1}{2})=a(x_p+1)+b(y_p+\frac{1}{2})+c$$

$$=ax_p+by_p+c+a+\frac{b}{2}=a+\frac{b}{2}\rightarrow 2dy-dx$$

$$\Delta E=d_{new}-d_{old}=2dy$$

-'NE' 가 선택 되었을 때 ΔNE의 값을 구하여 d값에 더해준다.

$$d_{new}=F(x_p+2,y_p+\frac{3}{2})=a(x_p+2)+b(y_p+\frac{3}{2})+c$$

$$=ax_p+by_p+c+2a+\frac{3b}{2}=2a+\frac{3b}{2}\rightarrow 4dy-3dx$$

$$d_{old}=F(x_p+1,y_p+\frac{1}{2})=a(x_p+1)+b(y_p+\frac{1}{2})+c$$

$$=ax_p+by_p+c+a+\frac{b}{2}=a+\frac{b}{2}\rightarrow 2dy-dx$$

$$\Delta NE=d_{new}-d_{old}=2(dy-dx)$$

이와 같이 $\Delta E, \Delta NE$ 값을 구하여 x값이 증가할 때마다 덧셈만으로 d값을 갱신하여 새로운 점을 찾아낸다.■

따라서 이러한 알고리듬을 사용하면 연속된 격자점들의 y좌표를 구할 때 덧셈연산만으로 가능하다. 물론 제안된 방법에서는 격자좌표를 정수만 사용하므로 실제의 실수값과 약간의 오차가 있다. 그러나 본 논문에서는 x축 400개, y축 30개의 정수 격자를 사용하기 때문에 오차는 약 0.5% 정도로 무시할 수 있을 만큼 작다.

예1) 시작점이 (5, 8)이고 끝점이 (9, 11)일 경우, 두 점을 잇는 직선을 앞의 방법으로 구해본다.

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

이 값을 가지고 x축을 하나씩 증가시키며 y축 값을 구하는 방법은 다음과 같으며 그림 3에서와 같이 좌표점이 선택된다.

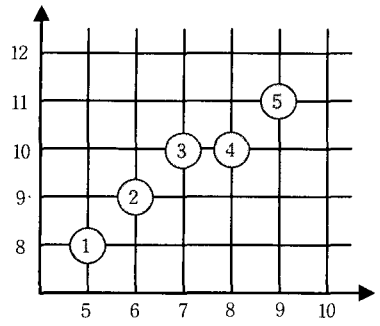


그림 3. 격자 좌표 선정의 예
Fig. 3. Example of pixels selection.

1. (5, 8)에서 d=2, d≥0 'NE' 선택
2. (6, 9)에서 d=2-2=0, d≥0 'NE' 선택
3. (7,10)에서 d=0-2=-2, d<0 'E' 선택
4. (8,10)에서 d=-2+6=4, d≥0 'NE' 선택
5. (9,11) 끝점

위와 같이 정수연산만을 사용하여 모든 점들의 좌표를 구할 수 있다. 시작점 (x_1, y_1)과 끝점 (x_2, y_2)을 잇는 직선의 정수형 매핑 알고리듬은 그림 4와 같다.

후건부에서의 복잡한 다각형 형태를 그래프로 나타내기 위해서는 기울기의 절대값이 1보다 작은 임의의 직선들을 나타낼 수 있어야 한다. 본 논문에서 제안하는 후건부에서의 정수 격자의 형태는 x축 400개, y축 30개를 사용함으로, 실제 퍼지 추론에 관련되는 모든 후건부의 소속함수의 기울기의 절대값이 1보다 작은 임의의 직선들을 표현할 수 있다. 그림 5에서의 예처럼 삼각형 형태의 그래프에서 α값에 따라 표현되는 후건부의 사다리꼴 모양의 그래프를 표현하기 위해 직선(b), (c)의 매핑 방법에 대해서도 알아본다.

실제로 비퍼지화에서 필요한 부분은 사다리꼴 형태

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

그림 4. 정수형 매핑 알고리듬
Fig. 4. integer mapping algorithm.

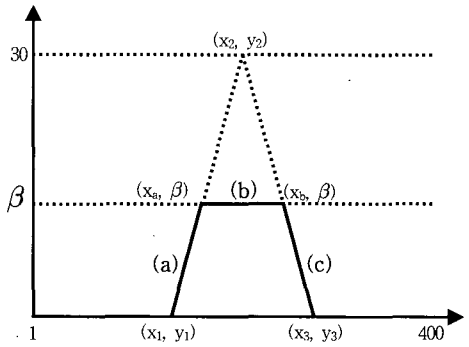


그림 5. β값에 따른 후건부 표현의 예
Fig. 5. Example of representation of the consequent part using β.

의 그래프이기 때문에 y축에서 정수 $\beta+1$부터 30까지의 부분은 매핑될 필요가 없다. 따라서 정수형 격자점을 표현하는데 있어 β값의 위치까지만 나타내야 한다. 그림 5와 같은 형태의 그래프를 표현할 때에는 순서가 (a)→(c)→(b) 의 순서로 이루어진다. 이것은 (a)→(b)→(c)의 순서로 매핑 된다면 (b)부분을 x축의 어느 지점까지 매핑해야 한다는 것을 판단하기 위해서는 두 점 (x_2, y_2)와 (x_3, y_3)의 직선의 방정식을 이용해서 β값에 대응되는 x값을 계산해야 하는 번거로움이 생긴다. 하지만 (a)→(c)→(b) 순서로 계산을 하게 된다면 (x_3, y_3)에서부터 (x_b, β)까지 정수 격자점에 매핑되기 때문에 직선(c)를 그리게 되면 (b)부분의 직선을 그리는 것은 간단해진다. 여기서 (c)부분을 그리는 방법은 시작점 (x_2, y_2)와 끝점 (x_3, y_3)를 그리는 것이 아니라 (x_3, y_3)에서부터 (x_2, y_2)까지 매핑된다. 이 때 사용되는 계산 방법은 다음과 같다.

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

$$dy = y_3 - y_2,\ dx = x_3 - x_2,\ y = \frac{dy}{dx}x + B$$

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

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

좌표 결정을 위해 중앙점을 사용하며 첫 번째 중앙점은 $(x_p - 1, y_p + \frac{1}{2})$이 된다.

$$d_{start} = F(x_p - 1, y_p + \frac{1}{2}) = a(x_p - 1) + b(y_p + \frac{1}{2}) + c$$

$$= (ax_p + by_p + c) - a + \frac{b}{2}$$

$$= -a + \frac{b}{2} \rightarrow -2dy - dx$$

(d : Decision Variable)

$d<0$ 일때 'NW' (x_p-1, y_p+1), $d \geq 0$ 일때 'W' (x_p-1, y_p)을 선택

'NW'를 선택 하였을 경우 x값이 감소할 때 y값은 증가하고, 'W'를 선택 하였을 경우 x값이 감소할 때 y 값은 증가되지 않고 그대로 유지된다.

-'NW' 가 선택 되었을 때 ΔNW의 값을 구하여 d 값에 더해준다.

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

-'W' 가 선택 되었을 때 ΔW의 값을 구하여 d값에 더해준다.

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

이와 같이 $\Delta NW, \Delta W$ 값을 구하여 x값이 감소할 때 마다 d값을 갱신하여 새로운 점을 찾아낸다.■

예2) 시작점이 (9, 8)이고 끝점이 (5, 11)일 경우, 두 점을 잇는 직선을 앞의 방법으로 구해본다.

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

이 값을 가지고 x축을 하나씩 감소시키며 y축 값을 구하는 방법은 다음과 같으며 그림 6에서와 같이 좌표점이 선택된다.

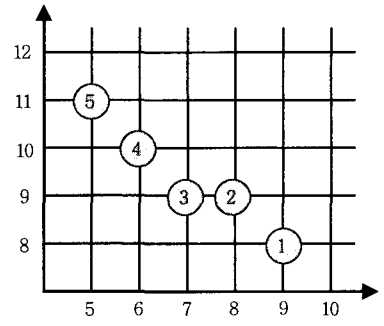


그림 6. 격자 좌표에서 선택된 좌표점
Fig. 6. Selected points in the integer pixels.

1. (9, 8)에서 d=-2, d<0 'NW' 선택
2. (8, 9)에서 d=-2+2=0, d≥0 'W' 선택
3. (7, 9)에서 d=0-6=-6, d<0 'NW' 선택
4. (6,10)에서 d=-6+2=-4, d<0 'NW' 선택
5. (5,11) 끝점

직선 (a)와 (c)가 매핑이 이루어지면 남은 직선 (b)만 계산하면 된다. 직선 (b) 부분은 앞서 매핑한 두 직선 (a)와 (c)의 최종 격자값 (x_a, β)와 (x_b, β)사이의 모든 x값에 β값을 채워주면 된다. x_a에서부터 x축을 하나씩

```
procedure right_line
dx ← x2-x3; dy ← y2-y3; 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.
```

그림 7. 정수형 매핑 알고리듬
Fig. 7. integer mapping algorithm.

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

그림 8. 정수형 매핑 알고리듬
Fig. 8. integer mapping algorithm.

증가시키면서 대응되는 y값을 모두 β로 유지시켜주면 필요로 하는 (b)부분의 매핑이 된다. 시작점 (x_3, y_3)과 끝점 (x_2, y_2)를 잇는 직선의 정수형 매핑 알고리듬은 다음의 그림 7과 같이 쓸 수 있다.

또한 직선(b)는 (a)와 (c)의 결과를 가지고 그림 8과 같이 쓸 수 있다.

2. 비퍼지화에서의 자료구조

비퍼지화 부분에서 필요한 자료구조는 후건부에서 정수형 격자 400개에 대응되는 y축의 정수값을 저장할 수 있는 정수 배열이 필요하다. 이 정수 배열에는 앞에서 설명한 각 퍼지 규칙의 후건부에서 정수형 픽셀의 좌표 (1~30)가 들어간다. 각각의 규칙의 비퍼지화 부분을 계산할 때 후건부에서의 픽셀의 최대값이 들어가게 된다. 즉 이를 위한 알고리듬은 그림 9와 같다.

```
begin
  integer array defuzz[1:m]
  integer array max[1:m] ← 0

  for i := 1 step 1 until n do
  begin
    for j := 1 step 1 until m do
    begin
      정수형 pixel 연산
        return value ← defuzz(1:m)
      if max(j) < defuzz(j)
        then max(j) ← defuzz(j)
    end
  end
end.
```

그림 9. 최대값 갱신 알고리듬
Fig. 9. Max data update algorithm.

```
begin
  lower ← 400
  upper ← 1
  begin
    procedure left_line
    procedure right_line
    procedure middle_line
  end
  if (x1<lower)
    then lower ← x1
  if(x3>upper)
    then upper ← x3
end.
```

그림 10. Non-zero item 탐지 알고리듬
Fig. 10. Non-zero item detection algorithm.

여기서 n은 퍼지 규칙의 개수이고, m은 후건부의 x축의 정수형 격자의 수이다.

일반적으로 후건부의 x축에 대응되는 y축의 값이 0인 경우가 대부분이기 때문에 COG 연산에서 불필요한 연산으로 많은 시간이 소비된다[10]. 이것을 보완하기 위해 non-zero item의 시작 위치와 끝 위치를 확인하여 두 위치 사이의 구간에 대해서만 COG를 연산하게 되면 불필요한 연산에 따른 오버헤드를 줄일 수 있다. 이를 위한 알고리듬은 그림 10처럼 간단하게 표현할 수 있다[11].

3. 비퍼지화에서의 COG 연산

기존에 사용되는 COG 연산 방법은 많은 양의 실수의 곱셈과 나눗셈으로 이루어진다. 실수연산은 정수형 매핑 알고리듬을 이용하여 알맞은 정수형으로 변환하여 해결하였지만 많은 양의 곱셈과 나눗셈 연산에 따른 연산시간 지연을 해결하기 위해 본 논문에서는 정수의 덧셈과 한번의 나눗셈만으로 COG 연산을 할 수 있는 방법을 제안한다.

정수형 매핑 알고리듬에 따라 y값은 하나씩 증가 하거나 감소하거나 유지되기 때문에 x축의 값이 하나씩 증가함에 따라 y축에 대응되는 값들은 세 가지 경우로 나타난다. 후건부 연산을 위한 400개의 배열에서도 이웃한 배열공간끼리도 마찬가지로 -1, 0, 1 씩 차이가 나게 된다. 그림 11은 정수형 덧셈만으로 COG 연산을 위한 배열의 예이다.

기존의 COG 연산 방법은 다음과 같은 식으로 표현된다.

$$COG = \frac{\sum_{x=1}^{400} x \cdot f(x)}{\sum_{x=1}^{400} f(x)}$$

$$= \frac{1\times 0+\cdots+175\times 1+176\times 2+\cdots+185\times 2+186\times 1+\cdots+400\times 0}{0+\cdots\cdots+1+2+\cdots+2+1+\cdots\cdots+0}$$

$$= 180.5$$

하지만 이런 기존의 방법은 n개의 item에 대해 n번의 곱셈과 $2(n-1)$번의 덧셈, 한번의 나눗셈을 사용하기

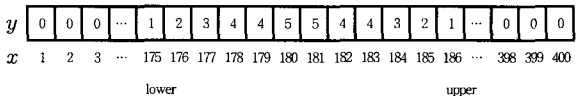


그림 11. COG 연산을 위한 배열의 예
Fig. 11. Example of integer array for COG operation.

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

그림 12. COG 연산 알고리듬
Fig. 12. COG operation algorithm.

때문에 많은 시간이 걸리게 된다. 본 논문에서 제안하는 방법은 그림 12와 같다.

제안된 방법으로 그림 11의 예를 계산해 보면

defuzz(175) = 1	temp = 1	sum = 1
defuzz(176) = 2	temp = 3	sum = 4
defuzz(177) = 3	temp = 6	sum = 10
⋮	⋮	⋮
defuzz(186) = 1	temp = 38	sum = 247

$$COG = \frac{sum}{temp} + (lower - 1)$$

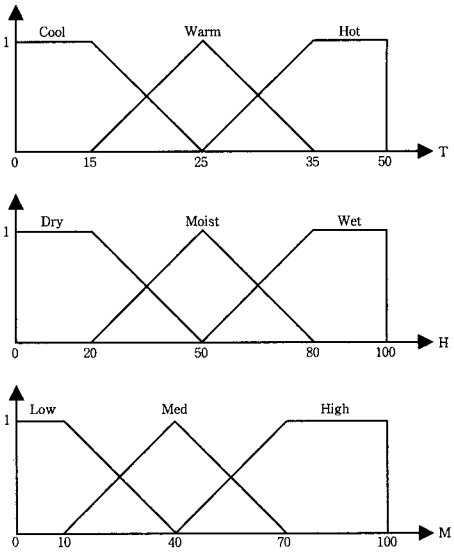
$$= 6.5 + (175 - 1) = 180.5$$

temp는 기존의 무게중심을 구하는 식에서 분모가 되며 n개의 non-zero item 에 대해 $2n+1$번의 덧셈과 한번의 뺄셈, 한번의 나눗셈으로 계산이 가능하다. 이것은 n번의 곱셈을 하는 것 보다 빠른 계산 시간을 보여준다.

Ⅲ. 연 산 속 도 분 석

본 논문에서는 실수의 덧셈, 곱셈 연산을 대신할 수 있는 정수의 덧셈 연산만으로 퍼지추론 및 비퍼지화 과정을 연산할 수 있는 새로운 기법을 제안하여 기존의 퍼지 논리 시스템에 비해 고속의 연산 결과를 얻을 수 있다. 그러나 추론 결과의 정확성 측면에서는 [0, 1]의 실수 연산을 사용한 경우와 비교하여 다소 오차가 발생하지만, 무시할 수 있을 만큼 작다. 따라서 본 장에서는 간단한 퍼지 시스템인 에어컨의 모터를 제어하는 시스템에서 기존의 fast 알고리듬[3]과 비교하여 퍼지연산의 한 프레임만의 수행시간의 분석 및 비교를 수행한다. 대상이 되는 퍼지 제어 시스템은 [3]에 있는 퍼지 시스템으로 하여 그림 13과 같은 9개의 퍼지 규칙과 각 소속함수를 사용한다.

이 경우에 온도가 17℃이고 습도가 32%인 경우 정확한 적분에 의한 식(1)의 경우에 COG 방법에 의한 무게



Temperature

Humidity	Cool	Warm	Hot
Dry	Low	Med	High
Moist	Low	Med	High
Wet	Med	High	High

그림 13. 에어컨 제어 시스템의 소속함수와 퍼지규칙
Fig. 13. Membership function and fuzzy rules of air conditioner control.

중심은 24.73이다[3]. 이 경우 본 논문에서 제안한 방법과 기존의 실수 연산의 방법을 x축의 구간을 400개와 800개로 나누어 연산 수행속도와 무게중심 값을 비교하여 표 1에 요약하였다.

표 1에서 알 수 있듯이 (400×30) 격자의 경우에 무게중심 값은 기존의 방법인 경우 24.77, 제안된 방법인 경우에 24.83으로 나타났다. (800×30) 격자의 경우에는 무게중심 값은 기존의 방법인 경우 24.75, 제안된 방법의 경우 24.80으로 나타났다. 따라서 오차는 (400×30) 격자일 때 0.1, (800×30) 격자 일 때 0.07로 줄었다. 한편 수행속도는 7.3배 정도 빠르게 결과값을 얻을 수 있다. 본

표 1. 산수행시간과 무게중심 비교
Table 1. Comparison of execution times and COGs.

구 분	(400×30) 격자		(800×30) 격자	
	연 산 수행시간	무게중심	연 산 수행시간	무게중심
이론치(적분)	-	24.73	-	24.73
기존의 방법[3]	2.70 μs	24.77	4.67 μs	24.75
제안된 방법	0.37 μs	24.83	0.64 μs	24.80

(조건 : 온도 = 17℃, 습도 = 32% 일 때)

논문에서 제안된 방법에서 오차를 더욱 줄이려면 (400×30) 격자 대신에 (800×30) 격자와 같이 x축에서의 격자수를 2배로 늘리면 연산수행시간은 조금 더 걸리지만 오차는 더욱 더 줄어든다.

본 논문에서 제안된 방법은 실수연산을 사용하는 기존의 방법과 비교하여 (400×30) 정수형 격자를 사용할 때, 퍼지추론 및 비퍼지화 부분에서 대략 7.3배 정도의 속도향상을 얻을 수 있으므로 기존의 퍼지 연산에 비해 매우 빠름을 알 수 있다.

Ⅳ. 시뮬레이션

본 논문에서 제안하는 알고리듬의 성능 평가를 위해 Truck backer-upper 제어 시스템에 적용하여 시뮬레이션 하였다. Truck backer-upper 제어 시스템은 그림 14와 같이 트럭을 주어진 공간에 주차하는 문제이다[12, 13]. 펜티엄4 2.4G, 메모리 512MB, 윈도우 XP환경에서 시뮬레이션을 하였다.

여기서 x는 트럭 위치의 x좌표를 의미하고, ϕ는 x축과 차체와의 각도이다. 이 때 트럭의 앞바퀴 축을 θ만큼 회전시켜 후진을 하면서 정해진 위치에 트럭을 주차시키는 문제이다. 이 시스템에서의 퍼지 규칙은 다음과 같은 형태로 구성된다.

IF x=Small and ϕ=Medium THEN θ=Large

전건부 1(x), 전건부 2(ϕ) 및 후건부(θ)의 소속함수 및 퍼지 규칙들은 그림 15와 같다[13].

Truck backer-upper 제어시스템에서는 초기의 x와 ϕ를 가지고 추론한 다음, 식 (4)와 (5)와 같은 동역학에서 접근하는 방식으로 함수를 이용하여 다음 단계의 x와 ϕ값을 결정하고, 결정된 값을 가지고 다시 추론을 하여 최종 목표에 도달할 때까지 반복적으로 수행한다. 여기서는 트럭의 y좌표는 고려하지 않는다.

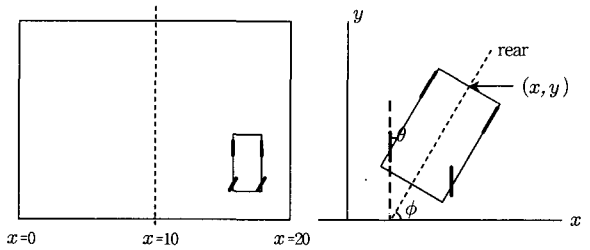
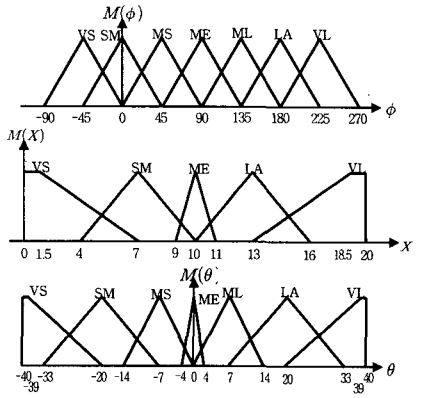


그림 14. 트럭 backer-upper 제어 시스템
Fig. 14. Truck backer-upper control system.



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

그림 15. 트럭 backer-upper 제어 시스템의 소속함수와 퍼지 규칙
Fig. 15. Membership functions and fuzzy rules of truck backer-upper control.

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

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

식 (5)에서 b는 트럭의 길이를 나타낸다. 시뮬레이션에서의 트럭의 길이는 4로 하였다.

시뮬레이션의 초기치 값은 $x = 1.0$, $\phi = 0.0^\circ$로 하여 트럭을 $x = 10.0$ 위치에 주차하도록 하였다.

표 2는 기존의 방법과 본 논문에서 제안한 방법의 결과를 비교해 놓은 표이다. 여기서 기존의 방법은 실수연산을 통한 실험결과이다.

표 2에서 보는 바와 같이 본 논문에서 제안하는 방법은 기존의 실수 방법과 비교하여 약간의 오차가 있음을 알 수 있다. 이것은 후건부 및 비퍼지화 단계에서 정수형 연산만을 사용하여 고속으로 처리되었기 때문이다. 오차는 약 0.4% 정도이다. 이러한 오차는 퍼지 제어시스템에 있어 무시할 수 있을 만큼 작다. 그러나, 기존의 방법과 비교하여 제안된 방법은 속도면에서 전체적으로 3.4 배만큼 빠른 처리속도를 얻을 수 있었다. 앞의 (표 1) 에서의 결과는 한 프레임의 퍼지추론 및 비퍼지화부

표 2. 시뮬레이션 결과
Table 2. Results of simulation.

기존의 방법				제안된 방법			
t	x	ϕ°	θ°	t	x	ϕ°	θ°
0	1.00	0.00	-19.00	0	1.00	1.00	-18.58
1	1.95	9.37	-17.95	1	1.89	9.42	-17.57
2	2.88	18.23	-16.90	2	2.81	18.28	-16.47
3	3.79	26.59	-15.85	3	3.71	26.62	-15.44
4	4.65	34.44	-14.80	4	4.57	34.48	-14.40
5	5.45	41.78	-13.75	5	5.38	41.81	-13.36
6	6.18	48.60	-12.70	6	6.09	48.65	-12.31
7	7.48	54.91	-11.65	7	7.45	54.95	-11.30
8	7.99	60.71	-10.60	8	7.97	60.74	-10.18
9	8.72	65.99	-9.55	9	8.66	66.04	-9.15
10	9.01	70.75	-8.50	10	8.97	70.80	-8.10
11	9.28	74.98	-7.45	11	9.24	75.02	-7.04
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	-4.94
14	9.72	84.57	-4.30	14	9.69	84.61	-3.92
15	9.81	86.72	-3.25	15	9.77	86.77	-2.90
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	0.16

분에만 적용된 성능향상이고 (7.3배), truck backer-upper 시스템과 같은 실제응용에서는 액츄에이터를 위한 방정식을 계산하는 시간도 포함되므로 전체적인 성능향상(3.4배)은 다소 낮게 나왔다. 따라서 본 논문에서 제안한 방법의 속도향상의 upper bound는 7.3배가 된다고 할 수 있다. 제안된 시스템은 로봇의 팔 움직임과 같은 고속의 지능 시스템에 잘 활용될 수 있다.

V. 결 론

지금까지 대용량의 퍼지 데이터를 처리하기 위해 여러가지 방법들의 고속 퍼지 하드웨어 모듈 및 병렬화 방법에 관한 논문들이 제안되어 왔다. 그러나, 기존의 대부분의 고속 퍼지 제어기들은 고속화 방법에도 불구하고, 전건부 및 후건부의 처리, 무게중심을 구하는 단계에서 [0, 1]의 실수 연산을 통해 퍼지 추론 및 비퍼지 연산을 수행하므로 추론결과를 얻기까지 많은 시간을 실수 연산에 허비하는 문제점을 안고 있다. 일반적으로 퍼지 연산은 전건부보다 후건부 및 무게중심을 구하는 단계에서 대부분의 계산시간을 필요로 하기 때문에 후건부 및 무게중심을 고속으로 처리할 수 있는 하드웨어 모듈 및 알고리듬이 필요하다.

본 논문에서는 후건부의 연산 및 COG의 연산에서 후건부를 정수형 격자에 대응시켜 정수의 덧셈연산만으로 고속으로 처리할 수 있는 하드웨어 모듈 및 알고리듬을 제안하였다. 또한 무게중심을 구하는 단계에서 소속함수가 0인 부분은 자동으로 연산하지 않고 지나가며, 소속함수가 0이 아닌 부분만을 사용하여 정수의 덧셈연산만으로 무게중심을 구하는 방법을 제안하였다. 여기서는 정수의 덧셈연산들 및 한 번의 정수형 나눗셈만을 필요로 한다. 제안된 방법은 truck backer-upper 제어시스템에 적용하여 시뮬레이션을 통해 기존의 방법들보다 훨씬 빠르고 효율적임을 입증하였다.

본 논문에서 제안한 방법을 지능 시스템이나 고속의 추론을 요하는 퍼지 제어 시스템에 적용하면 높은 성능향상을 얻을 수 있을 것이다. 또한 인공위성으로부터 수집된 원격탐사화상과 같은 대용량의 퍼지 데이터에 대한 실시간 고속처리를 요구하는 시스템이나, 로봇의 팔 움직임을 제어하는 초고속 퍼지 시스템에도 잘 활용될 수 있다. 향후의 연구과제로서는 제안된 방법을 이용하여 퍼지 병렬 컴퓨팅을 효율적으로 지원할 수 있는 방법에 대한 연구가 이루어져야 한다.

참 고 문 헌

[1] J. Yen and R. Langari, *Fuzzy Logic : Intelligence*, Prentice Hall, 1999.

[2] E. Cox, *Fuzzy System Handbook*, AP Professional, 1994.

[3] Saade, J. J., *"Defuzzification Techniques for Fuzzy Controllers,"* IEEE Trans. System, man, and Cybernetics, vol. 30, no. 1, pp. 223-228, 2000.

[4] Z. Salcic, *"High-speed customizable fuzzy-logic processor: architecture and implementation,"* IEEE Trans. Systems, Man, and Cybernetics, Part A. no. 31, no. 6, pp. 731-737, 2001.

[5] Y. D. Kim and H. Lee-Kwang, *"High speed flexible fuzzy hardware for fuzzy information processing,"* IEEE Trans. Systems, Man, and Cybernetics, Part A. Vol 27, no. 1, pp. 45-56, 1997.

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

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

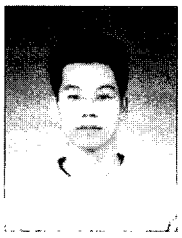
[8] J. Yen and L. Wang, *"Simplifying fuzzy rule-based models using orthogonal transformation method,"* IEEE Trans. System, man, and Cybernetics-Part B, vol. 29, no. 1, 1999.

[9] F. S. Hill, *Computer Graphics, 2nd ed*, Prentice

Hall, 2002.
[10] Kagei, S.,*"Fuzzy relational equation with defuzzification - algorithm for the largest solution,"* Fuzzy Sets and Systems, Vol. 123, pp. 119-127, 2001.
[11] Sang Gu Lee, *"High-speed Fuzzy Inference System in Integrated GUI Environment,"* International Journal of Fuzzy Logic and Intelligent Systems, vol. 4, no. 1, pp.50-55, June 2004.
[12] Daijin Kim and I.H. Cho,*"An accurate and cost-effective COG defuzzifier without the multiplier and the divider,"* Fuzzy Sets and Systems, Vol. 104, pp. 229-244, 1999.
[13] Li-Xin and Jerry M. Mendel, *"Generating fuzzy rules by learning from examples,"* IEEE Trans. System, man, and Cybernetics, vol. 22, no. 6, pp. 1414-1427, Nov. 1992.

저 자 소 개

이 상 구(정회원)
1978년 서울대학교 공과대학 전자공학과 졸업
1981년 한국과학기술원 전산학과 졸업(석사)
1995년 와세다대학 전기전자 컴퓨터공학과 졸업(박사)
1983년~현재, 한남대학교 컴퓨터공학과 교수
<주관심분야 : 퍼지이론, 컴퓨터 구조, 병렬처리 지능시스템, 임베디드 시스템>



채 상 원(학생회원)
2004년 한남대학교 컴퓨터공학과 졸업 (학사)
2006년 한남대학교 대학원 컴퓨터공학과 졸업 (석사)
<주관심분야 : 퍼지이론, 영상처리, OpenGL, 임베디드 시스템>