한국퍼지 및 지능 시스템학회 논문지 1998, Vol. 8, No. 2. (1998)70~81

승산기 및 제산기 없는 저비용 고정밀 COA 비퍼지화기
A Cost-Effective and Accurate COA Defuzzifier Without Multipliers and Dividers

김대진*, 이한별*, 강대성**
Daijin Kim*, Han-Pyul Lee*, and Dae-Seong Kang**
*동아대학교 컴퓨터공학과, **동아대학교 전자공학과

요 약

본 논문은 저비용이면서 정확한 비퍼지화 연산을 수행하는 새로운 COA(Center of Area) 비퍼지화기를 제안한다. 제안한 COA 비퍼지화기의 정확성은 비퍼지화 연산시 소속값 뿐 아니라 소속 함수의 폭을 고려함으로서 얻어진다. 제안한 COA 비퍼지화기의 저비용성은 비퍼지화 연산시 요구되는 나눗셈 연산을 좌·우측 모멘트의 균형점을 찾는 것으로 대신함으로서 얻어진다. 제안한 COA 비퍼지화기는 승산기가 부가적으로 필요하고 모멘트의 균형점을 찾는데 많은 시간이 걸리는 단점이 있다. 첫번째 단점은 승산기를 확률론적 AND 연산으로 대치함으로서 극복되고, 두 번째 단점은 모멘트의 균형점을 빠르게 탐색하는 coarse-to-fine 탐색 알고리즘에 의해 해결된다. VHDL 시뮬레이션을 통해 제안한 COA 비퍼지화기를 트럭 후진 주차 문제에 적용한 결과 기존의 COA 비퍼지화기보다 향상된 평균 주행 거리 특성을 보임을 확인하였다.

ABSTRACT

This paper proposes an accurate and cost-effective COA defuzzifier of fuzzy logic controller (FLC). The accuracy of the proposed COA defuzzifier is obtained by involving both membership values and spans of membership functions in calculating a crisp value. The cost-effectiveness of the proposed COA defuzzifier is obtained by replacing the division in the COA defuzzifier by finding an equilibrium point of both the left and right moments. The proposed COA defuzzifier has two disadvantages that it ncreases the hardware complexity due to the additional multipliers and it takes a lot of computation time to find the moment equilibrium point. The first disadvantage is overcome by replacing the multipliers with the stochastic AND operations. The second disadvantage is alleviated by using a coarse-to-fine searching algorithm that accelerates the finding of moment equilibrium point. Application of the proposed COA defuzzifier to the truck backer-upper control problem is performed in the VHDL simulation and the control accuracy of the proposed COA defuzzifier is compared with that of the conventional COA defuzzifier in terms of average tracing distance.

1. 서 론

퍼지 논리 제어기는 가전 및 산업 분야의 공정 제어에 폭넓게 응용되고 있다. 특히 시스템의 특성이 복잡하여 기존의 정량적인 방법으로는 해석할 수 없거나, 얻어지는 정보가 정성적이며, 부정확하고 불확실한 경우에 있어서 기존의 제어기보다 우수한 제어결과를 나타낸다[1]. 그림 1은 퍼지 제어기의 일반적 구성도를 나타낸 것으로 크게 4가지 구성 요소 - 퍼지화부, 추론 엔진부, 퍼지 규칙 베이스부, 그리고 비퍼지화부로 나뉜다. 각 부분의 동작 설명은 다음과 같다.

퍼지화부에서는 입력 변수의 값을 측정하고, 입력 변수의 영역을 전체 집합범위에 맞게끔 크기 변환한 뒤 입력값을 적절한 언어적인 값으로 변환시키고, 추론부에서는 퍼지 관계와 퍼지 논리의 추

※ 이 논문은 1997년도 동아대학교 공모과제 연구비에 의해 연구되었음.

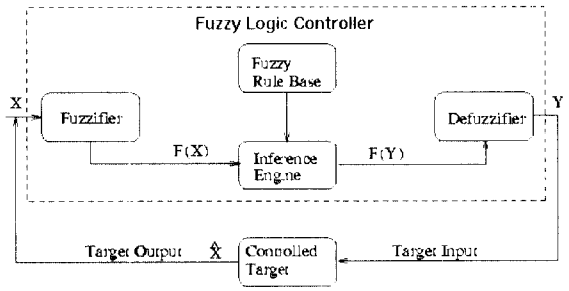


그림 1. 퍼지 제어기 일반적 구성도

론 규칙을 사용하여 퍼지 제어 출력을 결정하며, 퍼지 규칙 베이스부에서는 퍼지 논리 제어에서의 퍼지 자료를 조작하고 언어적 제어 규칙을 정의하는데 필요한 사항들을 정의한 데이타 베이스와 제어 전문가가 수행하는 일련의 제어 과정을 언어적 제어 규칙들로 나타낸 제어 규칙부로 구성되고, 그리고 비퍼지화부에서는 퍼지 출력값을 실제 제어 입력에 맞게끔 변환시켜 실제 제어 입력으로 사용할 수 있는 확정된 출력값으로 변환시켜 준다.

흔히 사용되는 비퍼지화 방법에는 최대값 방법(Max), 최대 평균법(Mean Of Maximum:MOM), 그리고 무게 중심법(Center Of Area:COA)이 있다[2]. COA 비퍼지화기를 하드웨어로 구성하는 경우 흔히 하드웨어의 복잡도를 줄이기 위해 출력 변수의 소속 함수 중심에서의 단일 출력값(Singleton)을 사용하여 비퍼지화를 행하는데, 이는 비퍼지화 출력값에 많은 오차를 유발하여 정확한 제어를 어렵게 만들 수 있다. 이를 극복하기 위해 출력 변수가 갖는 소속 함수 중심에서의 단일 출력값뿐 아니라 소속 함수의 면적도 동시에 고려하여 비퍼지화값을 연산할 것을 제안하였는데[3], 면적을 적분기에 의해 직접 계산하면 연산 시간이 많이 걸리므로 적분기를 사용할 때에 비해 제어 결과가 크게 뒤떨어지지 않으면서 비퍼지화 연산시 하드웨어 복잡도를 크게 증가시키지 않는 방안으로 소속 함수의 면적을 소속 함수가 갖는 폭으로 근사화시키는 방안을 사용하였다.

기존의 COA 비퍼지화기가 갖는 또다른 단점은 비퍼지화값 연산시 구현 비용이 높고 연산 시간이 많이 걸리는 나눗셈을 수행해야 한다. Ruiz 등[4]은 나눗셈을 출력 변수상의 좌·우측 모멘트의 균형점을 찾는 것으로 대신할 수 있음을 보였다. 그러나 이들이 제안한 모멘트 균형점 탐색은 매 탐색마다 출력 변수상에 매우 작은 크기의 동일 간격만큼 좌측 또는 우측으로 탐색점을 이동시킴으로서 탐색 시간이 많이 걸리는 단점이 있다. 본 논문은 이러한 단점을 coarse-to-fine 탐색 방법으로 해결하였다. 이 방법은 coarse 탐색시 출력 변수상 탐색점의 이동을 퍼지항 단위로 수행하며, 인접하는 두 퍼지항에 도달한 다음 Ruiz 등이 사용한 Fine 탐색을 수행함으로서 탐색 시간을 크게 줄일 수 있다.

그러나, 제안한 모멘트 균형점의 coarse-to-fine 탐색 알고리즘은 승산기를 추가적으로 요구한다. 이 문제는 확률론적 연산 이론의 도움으로 해결하고자 한다. 즉, 확률론적 연산에 따르면 한 수치값이 확률적으로 비트열내의 "1"의 개수와 비례하도록 표현되며, 이 경우 약간의 연산 오차가 발생되지만 두 수의 곱이 두 비트열간의 AND 연산에 의해 가능하다. 따라서 승산기를 하나의 AND 게이트에 의해 구현함으로서 구현 비용을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 COA 비퍼지화 방법이 갖는 문제점을 고찰해 보고 이를 개선하는 새로운 비퍼지화기를 제안한다. 3장에서는 제안한 비퍼지화기의 하드웨어 복잡도를 줄이는 두 가지 방안을 제시한다. 4장에서는 제안한 비퍼지화기의 하드웨어 아키텍츄어의 구성과 동작을 설명한다. 5장에서는 제안한 비퍼지화기를 포함하는 FLC의 VHDL 시뮬레이션을 수행하여 제어 성능의 개선 정도를 보이고, 이를 재구성 가능한 FPGA 시스템상에 실제 구현한 경우 퍼지 연산의 수행 속도 향상 정도를 보인다. 마지막으로, 6장은 결론과 앞으로의 연구 방향을 언급한다.

2. 새로 제안한 COA 비퍼지화기

COA 비퍼지화기는 비퍼지화값을 다음 식에 의해 얻는다.

$$y_c = \frac{\sum_{i=1}^{n} y_i \cdot \mu_Y(y_i)}{\sum_{i=1}^{n} \mu_Y(y_i)} \qquad (1)$$

여기서 n은 이산 퍼지항의 개수, y_i는 i번째 퍼지항의 단일 지지값(singleton), $\mu_Y(y_i)$는 단일 지

지값 y_i에서의 소속 함수값을 나타낸다. 이 비퍼지화값은 (1) 출력 변수의 소속 함수가 대칭이고, (2) 소속 함수 모두가 같은 폭을 가지며, (3) 소속 함수 중심이 동일 간격으로 놓여 있을 때만이 타당하다. 그러나 실제 응용에서는 이러한 제약들은 정확한 제어를 위해서는 적당하지 않다. 그림 2는 이러한 제약들이 만족하지 않는 경우, 기존의 COA 비퍼지화기가 오용된 사례를 보인 것이다. 아래 그림 2-(a)는 두 소속 함수가 같은 폭을 갖지만 그림 2-(b)는 두 소속 함수가 다른 폭을 갖는다. 만약 두 경우가 같은 단일 지지값을 갖는다면, 기존의 COA 비퍼지화기는 단일 지지값의 소속값만을 고려하기 때문에 (a)와 (b)는 서로 같은 비퍼지화값을 나타낸다 $y_c^a = y_c^b$.

그러나 이 결과는 면적이 더 넓은 곳 쪽으로 무게 중심이 이동해야 하는 사실과는 거리가 멀다.

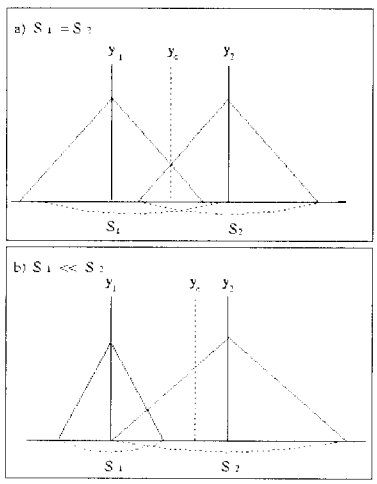


그림 2. 기존 COA 비퍼지화기의 오류의 예시

이를 극복하고자 본 논문에서는 기존의 COA 비퍼지화기 연시 소속 함수의 단일 지지값을 절단된 소속 함수 아래의 면적으로 대신한 아래 식에 의해 계산할 것을 제안한다.

$$y_c = \frac{\sum_{i=1}^{n} A_i(y_i)\cdot y_i}{\sum_{i=1}^{n} A_i(y_i)} \qquad (2)$$

여기서 $A_i(y)$는 i번째 추론된 소속함수의 절단된 소속 함수값의 아래 영역, n은 퍼지항의 개수, y_i는 i번째 퍼지항의 단일 지지값을 나타낸다.

그러나, 절단된 소속 함수값의 아래 영역의 연산은 적분기를 요구하므로 이를 하드웨어로 구현하면 구현 비용이 증가하고 연산 시간이 오래 걸리는 단점이 있다. 따라서 이 문제는 정확한 연산을 위해서는 면적 적분을 위한 적분기가 요구되는 점과 이를 하드웨어적으로 구현하면 너무 하드웨어가 복잡해지는 성능 대비 비용이라는 공학적 trade-off 문제임을 알 수 있다. 이러한 타협 문제에 대한 한 해결책으로 적분기를 사용할 때에 비해 제어 결과가 크게 뒤떨어지지 않으면서 비퍼지화 연산시 하드웨어 복잡도를 크게 증가시키지 않는 방안으로 아래 식과 같이 소속 함수의 면적을 소속 함수가 갖는 폭으로 근사화시키는 방안을 고려하였다[3].

$$y_c^a = \frac{\sum_{i=0}^{n} A_Y(y_i)\cdot y_i}{\sum_{i=0}^{n} A_Y(y_i)} = \frac{\sum_{i=0}^{n} \mu_Y(y_i)\cdot S_i \cdot y_i}{\sum_{i=0}^{n} \mu_Y(y_i)\cdot S_i} \qquad (3)$$

위 식을 기존의 COA 비퍼지화기와 비교해보면, 근사적 비퍼지화값 y_c^a 를 계산하기 위해서는 부가적인 승산기를 요구한다. 그러나, 뒷장의 시뮬레이션 결과에서 볼 수 있듯이, 이러한 추가적인 하드웨어의 요구는 제어 성능의 향상에 의해서 보상되어진다.

3. 하드웨어 복잡도 감소 방안

3.1 승산 연산을 확률론적 AND 연산으로 대치

식 (3)에서 알 수 있듯이 제안한 COA 비퍼지화기는 비퍼지화값을 계산하기 위하여 소속값과 소속함수 폭의 값을 동시에 곱셈 처리해야 하므로 추가적인 승산기를 요구한다. 본 논문에서 추가적 승산기 요구 문제를 확률론적 (stochastic) 연산 이론[5]에 기초한 간단한 AND 연산으로 대치함으로써 추가적인 승산기 요구에 따른 하드웨어 복잡도 증가 문제를 해결하고자 한다.

확률론적 연산은 하나의 수치값에 비례하는 "1"의 개수를 갖는 상대적으로 긴 확률론적인 랜덤

펄스열을 사용한다. 한 수치값을 확률론적인 펄스열로 부호화하는 코딩 회로는 그림 3-(a)처럼 하나의 의사 난수발생기(PRNG; Pseudo-random number gener- ator)와 하나의 디지털 비교기를 사용한다. 본 논문에서는 의사 난수 발생기로서 통상적으로 사용되는 선형 궤환 쉬프트 레지스터(Linear feedback shift register) 대신에 셀룰러 오토마타(CA) 이론에 기반한 PRNG[6]를 사용하였는데, 이는 후자를 택하면 제일 처음과 마지막 셀 사이의 긴 궤환 루프가 필요하지 않기 때문이다. 최대 순환 길이가 2^n-1을 나타내는 CA 기반 PRNG는 아래 식과 같은 규칙 90과 규칙 150에 따라 그들의 상태를 바꾸는 n개의 셀로 구성된다.

$$Rule90: s_i(t+1) = s_{i-1}(t) \oplus s_{i+1}(t)$$
$$Rule150: s_i(t+1) = s_{i-1}(t) \oplus s_i(t) \oplus s_{i+1}(t) \quad (4)$$

여기서, $s_i(t)$와 $s_i(t+1)$ 각각 i번째 셀의 현재 및 다음 상태를 나타내고, ⊕는 배타적 OR 연산을 의미한다. 규칙 90은 이웃하는 셀들의 상태값에 의해 자신의 다음 상태를 결정하지만, 규칙 150은 다음 상태로 갱신하는데 자신의 현재 상태 및 이웃하는 셀들의 현재 상태를 동시에 고려한다. 그림 3-(b)는 4개의 셀을 갖는 CA 기반 PRNG의 한가지 가능한 형태를 보인 것이다.

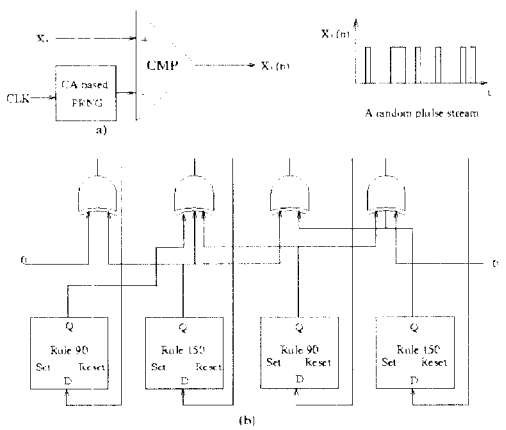


그림 3. 확률론적 연산을 위한 CA 기반 PRNG를 갖는 부호화 회로

비교기 출력에서 발생되는 각 펄스는 firing 확률 f_i를 갖는 Bernoulli 사건으로 모델링된다[7]. Firing 확률 f_i는 $\hat{f}_i = \frac{n}{N}$ 에 의해서 근사화되는데 여기서 n은 N개의 클럭 주기동안에 발생되는 "1"의 개수를 나타낸다. Firing 확률을 $\hat{f}_i$ 에 근사화시키는 것은 평균이 f_i이고 분산 $\sigma^2_{\hat{f}_i} = \frac{f_i(1-f_i)}{N}$ 인 이항 분포를 갖는다. 따라서, 펄스열 $x(n)$은 부호화 잡음 $\sigma^2_{\hat{f}_i}$ 에 의한 오차를 피할 수 없다.

확률론적 연산의 한 장점은 두 개의 확률론적 펄스열의 곱셈이 한 AND 게이트에 의해 수행 가능하다는 점이다. 세 실수 x, y 및 z ($0 <= x, y <= N$, $x \cdot y = z$)가 주어지면 x, y에 대응하는 두 확률론적 펄스열 f_x 및 f_y의 AND 연산은 아래 식에 의해 $\frac{z}{N}$ 을 나타낸다.

$$f_x \wedge f_y = \frac{n_x}{N} \cdot \frac{n_y}{N} = \frac{1}{N} \cdot \frac{n_x \cdot n_y}{N} = \frac{1}{N} \frac{n_z}{N} = \frac{f_x}{N} \quad (5)$$

이식에서, 곱셈 결과가 $\frac{1}{N}$ 에 의해 크기가 감소됨을 알 수 있는데 이러한 scaling 감소는 오히려 FLC 내의 수치들의 범위를 동일하게 유지시켜 주는 역할을 한다.

3.2 제산 연산을 모멘트 균형점 탐색으로 대치

COA 비퍼지화기는 다른 ALU 연산보다 훨씬 복잡한 나눗셈 연산을 요구한다. 본 논문에서는 나눗셈 연산을 모멘트 균형점 탐색으로 대신하고, coarse-to-fine 탐색 알고리즘에 의해 탐색 속도를 훨씬 빠르게 개선한다.

먼저, 출력 변수의 양끝에서부터 마주 보는 방향으로 출발하여 y = yc까지의 좌측 및 우측 모멘트 M_l과 M_r는 각각 아래 식과 같이 정의된다.

$$M_l = \sum_{i=0}^{j} \mu_Y(y_i) \cdot S_i \cdot y_i$$
$$M_r = \sum_{i=0}^{M-1} \mu_Y(y_i) \cdot S_i \cdot y_i \quad (6)$$

여기서 $\mu_Y(y_i)$, S_i, y_i, 와 M은 각각 i번째 소속 함수의 절단된 소속값, i번째 소속 함수의 폭, i번째 소속 함수의 단일 지지값과 출력 변수 y가 갖는 퍼지항의 개수를 나타낸다. 이때, 찾고자 하는 비퍼지화값 $y=y_c$에서 좌측 우측 모멘트가 아래 식과 같이 같은 값을 갖는다.

$$M_l(c) = M_r(c)$$
$$\sum_{i=0}^{c} \mu_Y(y_i) \cdot S_i \cdot y_i = \sum_{i=0}^{M-1} \mu_Y(y_i) \cdot S_i \cdot y_i \quad (7)$$

위의 좌측 모멘트M_l은 아래 식과 같은 반복식에 의해 쉽게 계산된다[3].

$$M_l(n) = M_l(n-1) + A_l(n-1) \cdot (y_n - y_{n-1})$$
$$A_l(n) = A_l(n-1) + \mu_n \cdot s_n \quad (8)$$

여기서 μ_n , s_n 과 y_n은 각각 n번째 반복에서 사용되는 소속함수의 절단된 소속값, 소속 함수의 폭, 그리고 소속함수의 단일 지지값을 나타내며, $A_l(n)$ 과 $M_l(n)$은 각각 n번째 반복에서 얻어지는 좌측 면적과 우측 모멘트를 나타낸다. 비슷하게 우측 면적 $A_r(n)$과 우측 모멘트 $M_r(n)$의 반복식은 아래와 같이 표현된다.

$$M_r(n) = M_r(n-1) + A_l(n-1) \cdot (y_{M-n} - y_{M-n-1})$$
$$A_r(n) = A_r(n-1) + \mu_{M-n-1} \cdot s_{M-n-1} \quad (9)$$

여기서 M은 출력 변수가 갖는 퍼지항의 개수이다. Ruiz 등은 M_l과 M_r을 계산하는데 출력 변수가 가지는 전체 정의 역을 2^p개로 균등 분할하여 탐색점을 한번에 한 간격씩 옮기면서 탐색을 수행하는데 모멘트 균형점까지 최소 2^{p-1}번에서 최대 2^p번을 옮겨야한다. 이러한 과도한 탐색점 이동 문제를 해결하기 위해 본 논문에서는 coarse-to-fine 탐색 방법을 제안한다. (여기서 대문자로 나타낸 양들은 Coarse 탐색과 관련된 것이고, 소문자로 나타낸 양들은 Fine 탐색과 관련된 것이다.)

Coarse-to-fine 탐색 방법은 다음과 같이 수행된다. 탐색을 수행하기에 앞서 A_l 과 A_r 그리고 M_l과 M_r을 0으로, I_l=0, I_r =M+1 (단, M은 퍼지항의 개수)으로 초기화시킨다. Coarse 탐색은 앞의 식 (11)과 (12)를 이용하여 M_l과 M_r을 구한 뒤 이들을 서로 비교하여 작은 값을 갖는 쪽이 한 퍼지항 단위로 이동하는데 (즉, M_l이 작으면 I_r=I_r+1 이고 M_r이 작으면 I_r=I_r-1 이 과정을 $|I_r - I_l| \le 1$이 될 때까지 반복한다. M_l=M_r인 경우에는 양쪽 모두 이동(즉, $I_l = I_l + 1$ 과 $I_l = I_l - 1$) 한다.

Coarse 탐색의 종료 조건을 만족하는 순간의 I_l과 I_r을 각각 I_l^c , I_r^c 라고 하면, 왼쪽 및 오른쪽 모멘트의 값은 각각 $M_l(I_l^c)$ 과 $M_r(I_r^c)$가 된다. 이들 값을 Fine 탐색을 위한 초기값으로 사용하기 위해 $i_l = I_l^c$ $i_r = I_r^c$ $a_l = A_l(I_l^c)$, $a_r = A_r(I_r^c)$로 둔다. Fine 탐색 과정에서 왼쪽 및 오른쪽 모멘트의 반복식은 아래와 같다.

$$m_l(n+1) = m_l(n) + a_l \qquad n = i_l, i_l + 1, ...$$
$$m_r(n+1) = m_r(n) + a_r \qquad n = i_r, i_r - 1, ... \quad (10)$$

실제 구현에서는 위 식의 덧셈을 AND 게이트의 한 입력을 항상 "1"로 둔 채 확률론적 곱셈에 의해서 수행하였다. 위 식을 이용하여 계산된m_l과 m_r을 서로 비교하여 작은 값의 모멘트를 갖는 쪽을 한 단위만큼 (여기서 한 단위는 $\frac{\text{출력 변수 범위}}{2}$의 크기를 갖는다.) 이동하는데, (즉, m_l이 작으면 i_l = i_l + 1 이고 m_r이 작으면 i_r = i_r - 1) 이 과정을 $|i_R - i_l| \le 1$이 될 때까지 반복한다. m_l = m_r인 경우에는 양쪽 모두 이동 (즉, i_l = i_l + 1 과 i_r = i_r - 1) 한다. Fine 탐색이 끝난 뒤, 비퍼지화값 y_c는 다음 식에 의해 결정된다.

$$y_c = \begin{cases} i_l & if \quad m_l \ge m_r \\ i_r & if \quad m_l \langle m_r \end{cases} \quad (11)$$

그림 4는 제안한 coarse-to-fine 탐색 알고리즘에서 모멘트 균형점을 찾는 과정을 나타낸 것이다. 앞에서 설명한 것처럼 coarse 탐색에서는 탐색점이 그림 4의 아래 부분에 나타낸 인덱스의 순서에 따라 퍼지항 단위로 움직인다. 일단 좌측 및 우측 모멘트가 서로 인접하는 두 항까지 도달한 다음에는 미리 정의한 간격 ($\frac{|\theta_{max} - \theta_{min}|}{2^p}$) 으로 움직이는 fine 탐색이 시작된다. Coarse 탐색동안 모멘트 탐색점의 이동은 $\frac{M}{2} \sim M$ (여기서 M은 퍼지항의 개수)번이고 fine 탐색동안 모멘트 탐색점의 이동은, 인접한 소속 함수의 중심간 거리가 평균적으로 $\frac{2^p}{M}$일 때, ($\frac{2^p}{2 \cdot M} \sim \frac{2^p}{M}$) 번이다. 따라서 제안한 coarse-to-fine 탐색 알고리즘의 전체 모멘트 탐색점의 이동은 최소 $\frac{M}{2} + \frac{2^p}{2 \cdot M}$ 번, 최대 $M + \frac{2^p}{M}$ 번이다. 따라서 $2^p \gg M$이라면 Ruiz 알고리즘에 비해 최대 $O(M)$의 탐색 속도 향상을 얻을 수 있다.

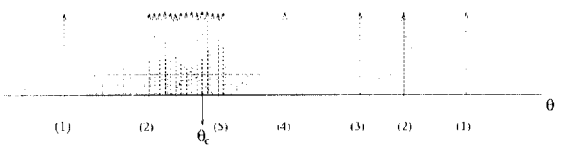


그림 4. 모멘트 균형점 탐색 과정 예시

4. 제안한 COA 비퍼지화기의 아키텍츄어

아래 그림 5는 한 FLC의 하드웨어 구조를 나타낸 것으로, MIN 모듈, 추론 모듈, MAX 모듈, 비퍼지화 모듈, 및 제어 모듈로 구성되어 있다. 본 논문의 주요 목적이 새로운 COA 비퍼지화기에만 국한된 것이므로 여기서는 제안한 비퍼지화기의 시뮬레이션 모델만 설명하지만, 실제 시뮬레이션에서는 FLC 전체의 시뮬레이션 모델을 구현하여야만 한다.

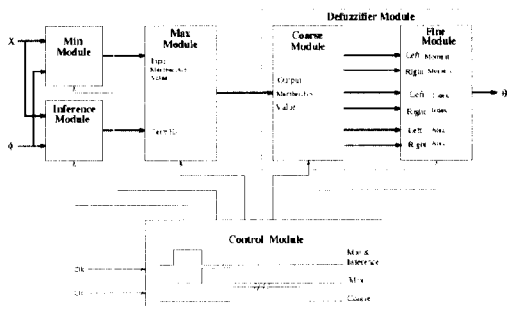
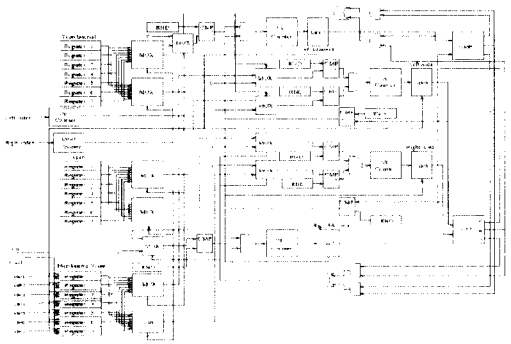


그림 5. FLC 하드웨어의 아키텍츄어

기본적으로 비퍼지화기 모듈은 두 개의 기능 블록 (coarse 탐색 블록(B_c)과 fine 탐색 블록(B_f)) 으로 구성된다. 다시 coarse 탐색 블록(B_c)는 각각 좌·우측 모멘트를 계산하는 두개의 부블록 B_c^l과 B_c^r 로 구성된다. 부블록 B_c^l 는 다시 두 개의 기능 유닛 $B_c^{l,m}$ 과 $B_c^{l,a}$ 로 구성되고, 부블록 B_c^r 는 다시 두 개의 기능 유닛 $B_c^{r,m}$ 과 $B_c^{r,a}$ 로 구성된다. 두 개의 부블록 B_c^l 과 B_r^c 는 같은 구조를 갖고 동작 과정이 비슷하므로 여기서는 좌측 부블록 B_c^l 에 대해서만 설명한다. 첫 번째 기능 유닛 $B_c^{l,m}$ 는 좌측 모멘트를 연산하는 역할을 하는데 이는 $A_i(i)$와 $y_i(i) - y_i(i\text{-}1)$와의 확률론적 AND 연산 결과를 카운터에 누적 가산을 함으로서 이루어진다. 두 번째 기능 유닛 $B_c^{l,a}$는 좌측 면적을 연산하는 역할을 하는데 이는 $\mu_i(y_i)$ 와 s_i 와의 확률론적 AND 연산 결과를 카운터에 누적 가산을 함으로서 이루어진다. 여기서, AND 게이트의 두 입력은 CA 기반 PRNG와 디지털 비교기에 의해 발생된 256비트 비트열들이다. 두 기능성 유닛의 동작 속도 T_i는 256 · CLK이며 여기서 CLK는 확률론적 곱셈 연산시 사용되는 한 펄스당 시간을 의미한다. T_i의 마지막 클럭 주기에서, 두 카운터 M_i과 A_i에 누산된 값들은 $Latch_c^m$ 과 $Latch_c^a$ 에 각각 옮겨진다. 두 기능 유닛사이에는 더 작은 모멘트 값을 갖는 부블록의 인덱스 포인터를 모멘트 균형점으로 이동하도록 하는 제어 회로가 존재한다. 두 부블록의 모멘트 값이 동일하면 양 인덱스 포인터가 모멘트 균형점으로 이동한다. 원하는 대로 동작하기 위해 인덱스 포인터가 이동하지 않는 부블록은 비활성화(deactivation) 시키는 것이 요구된다. 이를 위해 인덱스 포인터가 이동하지 않는 부블록은 입력으로 $M_Y(y_i)$와 s_i 대신 각각 "0"를 선택하도록 하였다. 이러한 제어 전략은 비활성화를 위해 추가적인 제어용 하드웨어가 필요하지 않으므로 매우 효과적이다.

제안한 coarse 탐색을 위해 각 소속 함수의 폭 s_i와 두 소속 함수간의 중심값의 간격 y_i - y_{i-1}, 그리고 소속 함수값 $M_Y(y_i)$ 를 저장하기 위해 각각 퍼지항 개수 M만큼의 크기를 갖는 세 개의 레지스터 파일이 필요하다. 각 레지스터 파일 다음에는 두 개의 멀티플렉서가 뒤따르는데 하나는 좌측 모멘트 부블럭을 위한 것이고 다른 하나는 우측 모멘트 부블럭을 위한 것이다. 두 개의 인덱스 포인터 I_l과 I_r은 레지스터 파일중 하나의 레지스터 값을 선택하기 위해 사용하는데 I_l은 up 카운터로 I_r



은 down 카운터로 작용한다. 그림 6은 출력 변수 그림 6. Coarse 탐색을 수행하는 회로 블록도의 퍼지항이 7인 경우 제안한 COA 비퍼지화기의 모멘트 균형점 탐색의 coarse 탐색을 수행하는 회로의 블록도를 나타낸 것이다.

Fine 탐색 블록 B_f 역시 두 개의 부블록 B_f^l 과 B_f^r 로 구성된다. 두 부블록 B_f^l과 B_f^r는 각각 좌·우측 모멘트를 계산하는 두 개의 가산기 ADD_l과 ADD_r로 구성된다. 두 개의 부블록이 같은 구조를 갖고 같은 방식으로 동작하므로 본 논문에서는 좌측 부블럭 B_f^l 에 대해서만 설명한다. Fine 탐색을 위한 두 개의 인덱스 포인터 i_l과 i_r는 초기값으로 각각 $y_c(I_c^l)$ 과 $y_c(I_c^r)$를 갖는다. 가산기 ADD_l의 두 입력은 a_l과 m_l로 각각 coarse 탐색 종료후 면적과 모멘트 값을 나타낸다. Fine 탐색의 동작 주파수는 클럭 주파수 CLK와 같다. 매 탐색마다, 두 가산기 출력이 서로 비교되어 작은 모멘트 값을 갖는 부블럭이 모멘트 균형점으로 나아간다. 두 모멘트 값이 동일하면, 양 부블럭의 두 인덱스 포인터가 모멘트 균형점쪽으로 나아간다. Fine 탐색을 원하는 대로 동작시키기 위해서도 coarse 탐색과 같은 제어 전략을 사용한다. 그림 7은 제안한 COA 비퍼지화기의 모멘트 균형점 탐색의 fine 탐색을 수행하는 회로의 블록도를 나타낸 것이다.

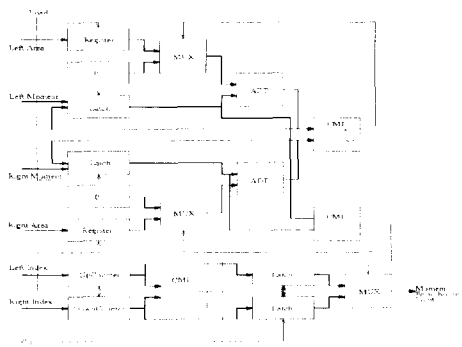


그림 7. Fine 탐색을 수행하는 회로 블록도

전체 FLC는 각 모듈이 언제 시작하고 언제 끝나는지를 제어하는 여러 제어 신호들에 따라서 동작한다. 기본 클럭 주기(CLK)는 확률론적 곱셈을 하는데 쓰이는 한 펄스당 시간이다. 따라서, 한 변수값이 256개의 이진 펄스열에 표현되므로 확률론적 곱셈을 마치는데는 256 · CLK이 소요되는데 본 논문에서는 256 · CLK을 CLK256으로 정의한다. 그러면, 제안한 FLC에서 하나의 제어 연산을 수행하는데 걸리는 연산 시간은 다음과 같다. MIN 연산과 추론 연산은 하나의 CLK256이 걸리며 이들은 동시에 수행 가능하다. MAX 연산도 한 CLK256이 걸리며, 비퍼지화 모듈내의 coarse 탐색은 N_c · CLK256이 걸리는 데, 여기서 N_c는 coarse 탐색에서 발생된 모멘트 탐색점의 이동 횟수이다. 끝으로 비퍼지화 모듈내의 fine 탐색은 한 CLK256이 걸린다. 따라서 하나의 퍼지 연산을 수행하는데 걸리는 연산 시간은 최소 $(3+\frac{M}{2})$ · CLK25에서 최대(3+ M) · CLK256이 걸린다.

FLC는 제어 대상물로부터 보내어오는 STARTUP 신호에 의해 제어 동작을 시작한다. 제어 동작을 시작하기 앞서, RESET 신호가 모멘트 함수값을 메모리로부터 레지스터 파일에 로드하고 카운터, 가산기, 레지스터 파일들을 초기화한다. 다음 START 신호에 의해 퍼지 연산이 시작된다. 각 모듈은 CLK256 클럭의 마지막 클럭이 시작하면서 해당 연산의 끝을 알리는 END 신호를 발생한다. CLK256 클럭의 마지막 클럭동안 다음 상태를 위한 모든 초기화 동작이 이루어지고 다음 단계는 새로운 CLK256의 첫 번째 클럭과 함께 시작된다. 이러한 ripple 형태의 동작은 비퍼지화 모듈의 fine 탐색의 끝날 때까지 계속되며, fine 탐색이 끝나면 새로운 START 신호를 발생한다. 이러한 퍼지 제어 동작은 제어 대상물로부터 오는 STOP 신호에 의해 종료된다.

5. 실험 결과 및 분석

제안한 비퍼지화기를 트럭 후진 주차 문제에 적용하여 제어 성능을 기존의 비퍼지화기와 비교한다. 비록 본 논문에서는 비퍼지화 모듈이외의 다른 모듈을 언급하지는 않았지만 본 실험을 위해 다른 모든 모듈들의 시뮬레이션 모델과 모델 트럭의 운동 방정식을 VHDL[8]을 사용하여 기술한 다음 이를 SYNOPSYS사의 VHDL 시뮬레이터[8]상에서 제어 과정을 시뮬레이션 하였다. 아래에 있는 운동 방정식을 시뮬레이션하기 위해서는 SYNOPSYS사가 제공하는 기본적인 IEEE 라이브러리 이외에도 Math-real package가 요구된다. VHDL 시뮬레이션 환경 관점에서 보면 제안한

비퍼지화기를 포함하여 설계된 FLC는 테스트중인 하나의 컴파일된 VHDL component 유닛 (Unit under test; UUT)으로 생각할 수 있으며, 모델 트럭은 testbed인 설계된 FLC에 stimulus 입력을 제공하는 하나의 프로세스로 볼 수 있다[9].

트럭 주차 문제의 목표는 가능한 빨리, 그리고 정확하게 트럭을 주차시키는 것이며, 이 문제는 기존 제어 기술로는 풀기 힘든 전형적인 비선형 제어 문제이다. 그림 8은 트럭 주차 제어 문제에서 사용된 트럭과 주차대의 위치를 보여준다. 트럭의 위치는 (x, y, ϕ)에 의해 결정된다. 단, 여기에서 ϕ는 트럭 진행 방향과 x축간의 각도이며, 트럭의 후진 주행 제어는 ϕ와 핸들의 축 간의 각도인 θ에 의해 결정된다. 트럭이 움직이는 운동 방정식은 아래와 같이 나타내어진다[10].

$$
\begin{aligned}
x(t+1) &= x(t) + \cos[\phi(t) + \theta(t)] + \sin[\theta(t) \cdot \sin[\phi(t)] \\
y(t+1) &= y(t) + \sin[\phi(t) + \theta(t)] - \sin[\theta(t) \cdot \sin[\phi(t)] \\
\phi(t+1) &= \phi(t) - \sin^{-1}[2\sin(\tfrac{\theta(t)}{b})]
\end{aligned}
\quad (12)
$$

여기서 b는 트럭의 길이이며, 본 논문에서는 b = 4로 하였다.

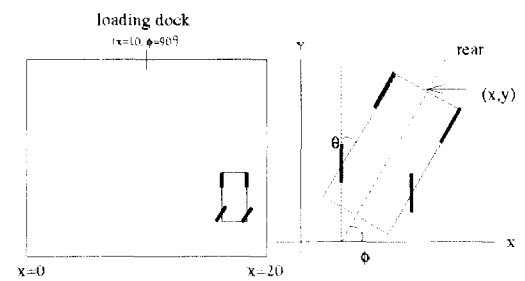


그림 8. 모형 트럭과 주차대 위치

만약 트럭과 주차대까지의 거리가 충분하다면 트럭이 $x=10$, $\phi=90°$ 가까이 오면 트럭을 곧장 후진하기만 하면 되기 때문에 변수 y를 퍼지 입력 변수(x, y, ϕ)에서 뺄 수 있다. 그러므로 트럭 주차 제어 문제는 주어진 공간내 $10 \le x \le 20$, $-90° \le \phi \le 270°$}임의의 초기 위치(x_0, ϕ_0)에서 가능하면 신속·정확하게 주차대 ($x=10$, $\phi=90°$) 쪽으로 후진하도록 바퀴 각도 $\theta(-40 \le \phi \le 40)$를 제어하는 것이 요구된다. 그림 9는 각각 Wang과 Mendel[11]이 사용한 퍼지 제어기의 입·출력 변수의 소속함수와 퍼지 제어를 위한 퍼지 규칙 베이스를 나타낸 것이다.

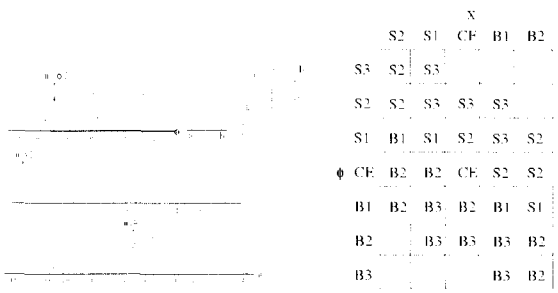


그림 9. 트럭 후진 주차제어에 사용된 소속 함수와 퍼지 규칙 베이스

그림 10과 11은 각각 제안한 COA와 기존의 COA비퍼지화기를 포함하는 FLC가 사용될 때, 모델 트럭이 임의의 한 상태 (x, y, ϕ)=(13.5, 0.0, 225.0°)에서 출발하여 경우의 VHDL 시뮬레이션 결과를 보인 것이다. 여기서 모든 변수값은 256 레벨로 양자화되어 있다. 예를 들면, 입력 변수 ϕ가 갖는 범위 [-90°, 270°]는 양자화 레벨 [00H, FFH]에 대응한다. 따라서, 시작 상태 (13.5, 0.0, 225.00°)는 한 16진수 (AFH, $7FH$, EFH)에 대응한다. 제안한 비퍼지화기를 포함하는 FLC가 목적지에 9 스텝만에 도달한 반면, 기존의 COA비퍼지화기를 포함하는 FLC는 목적지까지 15 스텝이 걸렸다.

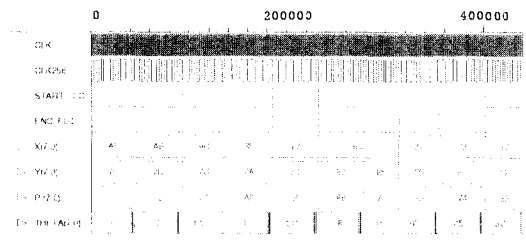


그림 10. 제안한 COA 비퍼지화기를 갖는 FLC의VHDL 시뮬레이션 결과

MAX-MIN 추론과 비퍼지화 연산을 시간적으로 나타낸 것이다. 이 경우, 하나의 퍼지 연산을 수행하는데 CLK256의 주기로 8 사이클이 걸림을 알

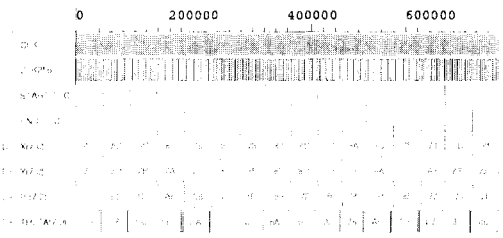


그림 11. 기존의 COA 비퍼지화기를 갖는 FLC의VHDL 시뮬레이션 결과

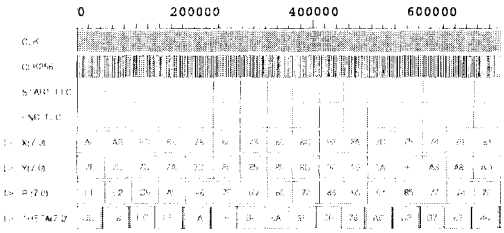


그림 12. 한 퍼지 연산의 수행 과정

수 있는데 각각 MAX-MIN 추론에 1 사이클, coarse 탐색에 6 사이클, find 탐색에 1 사이클 이 그림 12는 하나의 퍼지 연산 동안 일어나는 걸린다. 이 그림에서 좌측 인덱스 포인터(IDL)가 계속해서 6번 오른쪽으로 이동하는 것으로 보아서, 이 퍼지 연산의 경우는 출력 변수 상에 오직 하나의 절단된 소속 함수만이 존재하고 이는 가장 오른쪽(7번째) 퍼지항에 놓여 있음을 알 수 있다. 제안한 COA 비퍼지화기를 갖는 FLC와 기존의 COA 비퍼지화기를 갖는 FLC의 제어 성능을 비교하였다. 이를 위해 식 (6)과 (1)로부터 얻어진 값을 각각 제안한 COA 비퍼지화기와 기존의 COA 비퍼지화기의 비퍼지화값으로 생각하였다. 두 비퍼지화기간에 공평한 비교를 위하여 실제 시뮬레이션에서는 기존의 COA 비퍼지화기를 갖는 FLC를 테스트할 때는 제안한 COA 비퍼지화기를 갖는 FLC에서 각 소속 함수의 폭을 나타내는 span 레지스터의 값을 "0"으로 채운 채 제어를 수행하였다. 이렇게 함으로서 하드웨어의 구조를 변경시키지 않고서도 기존의 COA 비퍼지화기에서처럼 비퍼지화값 계산시 소속 함수의 폭을 고려하는 것을 효과적으로 없앨 수 있다. 제어 성능은 모델 트럭이 100개의 임의 값을 갖는 초기 위치 (x_i, ϕ_i)에서 목표점($x=10$, $\phi=90°$)까지 도달하는데 걸린 평균 주행거리에 의해서 비교된다. 실제 시뮬레이션에서는 제안한 COA 비퍼지화기가 확률론적 연산에 의한 불가피한 오차 때문에 모델 트럭이 목표점의 ±2.5%내에 도달하면 정지하도록 하였다. 표 1은 제안한 COA 비퍼지화기를 갖는 FLC와 기존의 COA 비퍼지화기를 갖는 FLC의 평균 주행거리가 각각 10.8 스텝과 14.3 스텝인 것을 보여준다. 이로부터 제안한 COA 비퍼지화기를 갖는 FLC가 기존의 COA 비퍼지화기를 갖는 FLC보다 약 24.5% 평균 주행 거리를 줄이는 성능 개선 효과를 가져옴을 알 수 있다.

표 1. 두 COA 비퍼지화기를 사용한 퍼지 제어기의 평균 주행거리 비교

비퍼지화기 형태	평균 주행 거리
기존의 COA	14.3 steps
제안한 COA	10.8 steps

제안한 FLC을 다음과 같은 과정을 거쳐 재구성 가능한 FPGA 시스템상에 구현하였다. 먼저 그림 5의 각 모듈을 VHDL 언어로 기술한 후, Synopsys사의 FPGA 컴파일러[12]에 의해 합성하였다. 합성된 각 모듈은 Xilinx사의 XactStep 6.0[13]에 의해 최적화 및 배치 배선의 back-end 처리를 하였다. 얻어진 Xilinx rawbit 파일은 VCC사의 r2h에 의해 C 언어 프로그램의 header 파일 형태의 하드웨어 object로 변환시킨 후, 이들 하드웨어 object를 포함하는 FLC 제어용 C 응용 프로그램을 작성한 후 이를 C 컴파일러에 의해 컴파일하였다. 이 실행 파일을 재구성 가능한 FPGA 시스템인 EVC1 보드[14]에 다운로드하여 원하는 제어 목적을 수행하는지 확인하였다. 이와 같이 제안한 FLC를 트럭 후진 주차 문제에 적용하여 재구성 가능한 FPGA 시스템의 일종인 EVC1 보드 상에 구현한 경우와 Synopsys사의 VHDL 시뮬레이터 상에 구현한 경우, 워크스테이션상에 C언어에 의해 구현한 경우 각각의 제어 수행시 연산 속도를 서로 비교하기 위해 목적지 도달에 관계없이 임의로 선택한 1,000개의 (x, ϕ)점에 대한 퍼지 제어기의 출력

θ가 얻어질 때까지 걸리는 시간을 평균하여 얻은 평균 퍼지 연산시간을 비교하였다. (실제 Synopsys 사의 VHDL 시뮬레이터 상에서 퍼지 연산을 수행하는데는 너무 많은 연산시간이 걸려 10개의 10스텝에 대해서만 수행하여 평균 시간을 얻었다.)

이 실험에서 세 가지 경우에 대한 수행시간 비교시 실험 조건을 될 수 있는 데로 공평하게 비교하기 위해 Synopsys VHDL 시뮬레이션의 경우 그래픽 환경에서 수행하지 않고 Shell 환경에서 Vhdlsim을 이용하였다. 표 2는 위의 세 가지 경우의 퍼지 연산에 걸리는 시간을 서로 비교한 것이다.

표 2. 세 가지 경우 평균 퍼지 연산시간

	Synopsys VHDL simulation	C language simulation	FPGA implemen tation
퍼지연산 수행 수	10	1000	1000
전체 걸린 시간	320.4	88.36	5.14
평균 퍼지 연간 시간	32.04	0.08830	0.00514

그림 13은 FPGA 구현시 걸리는 시간을 기준으로 하였을 때 다른 두 가지 시뮬레이션에 비해 얻어진 속도 향상비를 나타낸 것이다. 이 그림으로부터 제안한 FLC를 FPGA에 직접 구현하는 경우가 C 언어에 의한 시뮬레이션보다는 O(10)배, Synopsys VHDL 시뮬레이션보다는 $O(10^3)$ 배의 수행 속도 개선 효과를 얻을 수 있었다.

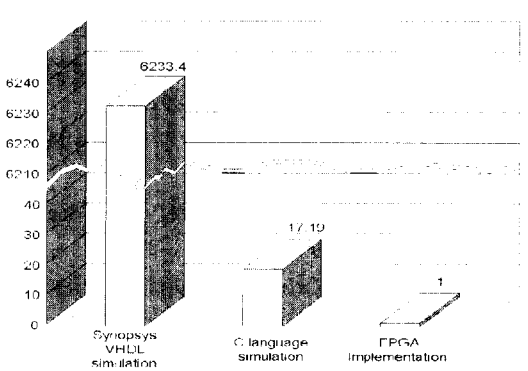


그림 13. 세 가지 경우 속도 향상비 비교

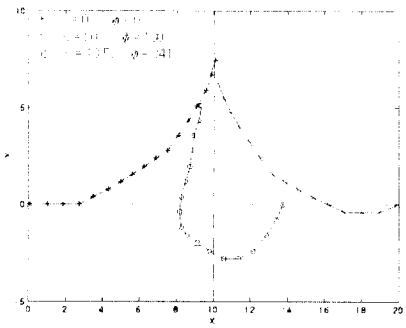


그림 14. 임의의 세점으로부터의 모형 트럭의 주행곡선

제안한 FLC를 재구성 가능한 FPGA 시스템 상에 구현한 경우 이들이 시뮬레이션에서와 똑같이 제대로 동작하는지를 알아보기 위해 임의의 세점 (왼쪽 그림 시작점 : (0.0, 0.0, 0°), 가운데 시작점 (13.5., 0.0, 241°), 오른쪽 그림 시작점 (20.0, 0.0, 180°)에서의 목적지까지 어떻게 주행하는가를 조사하였다. 그림 14로부터 세 경우 각각 16 스텝, 18 스텝, 15스텝 걸려서 목적지에 도달하는 것을 알 수 있다. 이로 미루어보아 재구성 가능한 FPGA 시스템 상에 구현한 FLC는 정확히 원하는 데로 동작하며 앞의 VHDL 시뮬레이터 상에서 보다 훨씬 빠르게 제어를 수행하고 있음을 확인할 수 있다.

6. 결 론

본 논문은 정확한 제어에 응용 가능하면서 동시에 저비용인 새로운 형태의 COA 비퍼지화기를 제안하였다. 기존의 COA 비퍼지화기가 비퍼지화값 연산시 소속 함수의 폭이 다름에도 불구하고 같은 비퍼지화값을 나타내는 잘못을 바로잡기 위해, 비퍼지화값을 계산하는데 있어 가중치로 절단된 소속 함수가 갖는 실효 면적을 사용하는 것을 제안하였다. 실효 면적을 계산하는 데는 적분의 사용을 배제하고자 실효 면적을 소속 함수값과 소속 함수의 폭의 곱으로 근사화하였다. 따라서 비퍼지화값 연산시 가중치로서 소속 함수값과 소속 함수의 폭을 동시에 고려함으로서 트럭 후진 제어 문제의 경우 목적지에 도달하는 데 걸리는 평균 주행 거리를 24.5% 이상 줄이는 성능 개선 효과를 얻을 수 있었다.

제안한 COA 비퍼지화기가 갖는 단점은 (1) 비

퍼지화값을 계산하는데 소속 함수의 폭을 고려함으로서 생기는 부가적 승산기에 의한 하드웨어 복잡도 증가 문제이고, (2) 비퍼지화값 연산시 나눗셈을 필요로 하는데, 제산기를 사용하지 않고서 모멘트 균형점의 탐색을 통해 얻음으로서 초래되는 비퍼지화 연산 시간의 증가 문제이다. 첫 번째 문제는 확률론적 연산에 의해 두 수의 곱이 두 비트열간의 AND 연산에 의해 계산함으로서 해결 가능하다. 따라서 부가적으로 요구하는 승산기를 하나의 AND 게이트에 의해 구현함으로서 구현 비용을 줄일 수 있다. 두 번째 문제는 효과적인 모멘트 균형점 탐색을 위한 coarse-to-fine 탐색법을 제시하였는데 coarse 탐색시 모멘트 계산점의 이동은 퍼지항 단위로 이동시킴으로써 전체적으로 탐색 시간을 크게 줄일 수 있었다. 이 결과 기존의 균일 간격 탐색법과 비교해 볼 때 최대 O(M)배의 속도 개선을 할 수 있었다.

제안한 FLC를 재구성 가능한 FPGA 시스템 상에 직접 hardwiring에 의해 구현하였다. 본 연구에서 구현한 FLC는 기존의 FLC 구조와 크게 달라, 1) 기존의 MIN-MAX 추론을 레지스터 파일 상에 read-modify-write 연산에 의해 대치했으며, 2) 비퍼지화값을 계산하는데 있어 소속 함수의 중심 출력 값과 소속함수의 폭을 동시에 고려하였고, 나눗셈기를 사용하지 않고 coarse-to-fine 탐색법에 의해 효과적으로 모멘트 균형점을 찾는 새로운 형태의 COG 비퍼지화기를 사용하였다.

재구성 가능한 FPGA 시스템 상에 구현한 FLC를 트럭 후진 주차 문제에 적용하여 그 연산 속도를 Synopsys사의 VHDL 시뮬레이터 및 워크스테이션상에 C 언어로 구현한 경우의 연산 속도와 비교하였는데, FPGA상에 구현한 경우가 VHDL 시뮬레이터보다는 $O(10^4)$배, C언어에 의한 시뮬레이션보다는 O(10)배 정도 빠름을 확인하였다. 아울러, 재구성 가능한 FPGA 시스템 상에 구현한 FLC가 제대로 동작하는지를 확인하고자 임의의 세점에 대한 모형 트럭의 주행 경로를 추적하여 보았는데 세 경우 모두 14-18 스텝 내에 목적지에 제대로 도달하는 것을 확인할 수 있었다.

참고 문헌

[1] E. H. Mandani, "Application of fuzzy algorithms for control of simple dynamic plant," IEEE Proc. Control & Science, Vol. 121, No. 12, pp. 1585-1588, Dec. 1974.

[2] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller," IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, No. 2, pp. 404-435, Feb. 1990.

[3] 김대진, 조인현, "모멘트 균형점의 효율적 탐색을 갖는 비제산기 COA 비퍼지화기", 대한 전자 공학회 논문지, 33권 10호, pp. 138-151, 1996년 10월.

[4] A. Ruiz, J. Gutirrez, and J. Fernndez, "A Fuzzy Controller with an Optimized Defuzzification Algorithm," IEEE Micro, pp. 1-10, Dec. 1995.

[5] Y. Kondo and Y. Sawada, "Functional Abilities of a Stochastic Logic Neural Network," IEEE Transactions on Nerual Networks, vol. 3, No.3, pp. 434-443, May 1992.

[6] C. Gloster and F. Brglez, "Boundary scan with cellular-based built-inself-test," IEEE International Test Conference, pp. 138-145, 1988.

[7] Richard L. Scheaffer and James T. McClave, "Probability and Statistics for Engineers," PWS-KENT Publishing Company, Boston, USA, 1990.

[8] Synopsys, "VSS Family Tutorial v3.4," Synopsys Corp., 1994.

[9] S. Mazor and P. Langstraat, A Guide to VHDL, Kluwer Academic Publishers, 1993.

[10] Li-Xin Wang And Jerry M. Mendel, "Generating Fuzzy Rules from Numerical Data with Applications," USC-SIPI Report, no.169, 1991.

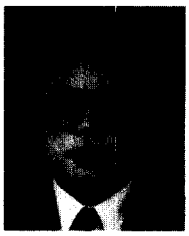
[11] Li-Xin Wang and Jerry M. Mendel, "Generating Fuzzy Rules by Learning from Examples," IEEE Transactions on System, Man, and Cybernetics, vol. 22, no. 6, pp. 1414-1427, Nov. 1992.

[12] Synopsys, "Design Analyzer Reference Manual v3.4," Synopsys Corp., 1994.

[13] Xilinx, "XACT Reference Guide," Xilinx Inc., San Jose, CA, April 1994.

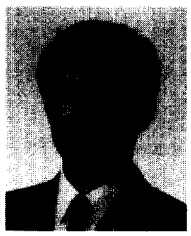
[14] VCC, "EVC1-Virtual Computer: Programming Tutorial," Virtual Computer Corporation, 1996.

[14] VCC, "EVC1-Virtual Computer: Programming Tutorial," Virtual Computer Corporation, 1996.



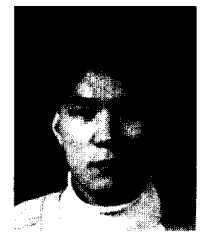
김 대 진(Dai-jin Kim) 정회원

1981:연세대학교 전자공학과(학사)
1984:KAIST 전기및 전자공학과(석사)
1991:Syracuse대학 ECE (박사)
1984~ 1987:KBS 기술연구소 HDT팀
1992~현재:동아대학교 컴퓨터공학과 부교수



강 대 성(Dae-Seong Kang)

1984:경북대학교 전자공학과(학사)
1991:Texas A&M University Electrical Eng.(석사)
1994:Texas A&M University Electrical Eng.(박사)
1984~ 989:국방과학연구소 연구원
1994~1995:한국전자통신연구소 선임연구원
1995 ~ 현재 : 동아대학교 전자공학과 조교수



이 한 별 (Han-Pyul Lee)

1997:경성대학교 컴퓨터공학과(학사)
1997~현재:동아대학교 컴퓨터공학과 석사 과정