자동차 환경에서 Oak DSP 코어 기반 음성 인식 시스템 실시간 구현

A Real-Time Implementation of Speech Recognition System Using Oak DSP core in the Car Noise Environment

우경호 · 양태영 · 이충용 · 윤대희 · 차일환*
(K-H. Woo · T-Y. Yang · C. Lee · D-H. Youn · I-H. Cha)

ABSTRACT

This paper presents a real-time implementation of a speaker independent speech recognition system based on a discrete hidden markov model(DHMM). This system is developed for a car navigation system to design on-chip VLSI system of speech recognition which is used by fixed point Oak DSP core of DSP GROUP LTD. We analyze recognition procedure with C language to implement fixed point real-time algorithms. Based on the analyses, we improve the algorithms which are possible to operate in real-time, and can verify the recognition result at the same time as speech ends, by processing all recognition routines within a frame. A car noise is the colored noise concentrated heavily on the low frequency segment under 400 Hz. For the noise robust processing, the high pass filtering and the liftering on the distance measure of feature vectors are applied to the recognition system. Recognition experiments on the twelve isolated command words were performed. The recognition rates of the baseline recognizer were 98.68 % in a stopping situation and 80.7 % in a running situation. Using the noise processing methods, the recognition rates were enhanced to 89.04% in a running situation.

Keyword : speech recognition, real-time implementation, fixed point Oak DSP

1. 서 론

오늘날 정보 통신 분야는 가장 빠르게 성장하고 있는 응용 분야이다. 음성 신호 처리 기술은 기계와 인간이 자연스럽게 많은 양의 정보를 시간이나 장소 등의 제약 없이 효율적으로 전달할 수 있는 방법으로서 꾸준히 연구되고 있다. 특히 음성 인식은 시스템들이 기계적인 작동 없이 동작하도록 하는 기술로써 정보 통신 영역에서 중요한 역할을 한다.

* 연세대학교 전기 · 컴퓨터공학과 음향 음성 및 신호처리 연구실

최근 많은 국가에서 무선 통신 기술들을 이용하여 운전자, 자동차, 지명, 여러 가지 정보 네트웍 시스템과 연결시킬 뿐 아니라 자동차 운행 중에 필요한 명령어 등 다양한 정보를 제공하는 지능형 교통 시스템(Intelligent Transport System)을 개발하고 있다. 특히 운행을 위한 교통 상황, 도로 구조, 상황에 따른 항법 가이드 등 다양한 정보를 운전자가 쉽게 조작하기 위해서는 인간-기계 인터페이스가 반드시 필요하다[1].

따라서, 이 논문에서는 차량 항법 및 카스테레오와 같은 자동차 편의 장치에 대한 명령어 및 조작을 위한 음성 인식 시스템을 실시간 구현하였다.

또한 복잡한 음성인식 시스템을 소형화하여 단일 칩으로 개발하기 위해 16비트 고정 소수점 Oak DSP 코어를 이용하였다. 실시간으로 동작하는 효과적인 시스템을 구현하기 위해 각 연산들의 관계와 소요시간 등을 분석한 후, 계산량 감소를 위해 알고리듬을 개선하였다.

논문의 구성은 2장에서는 인식 시스템의 구성 및 특징을 설명하고, 3장에서는 인식 알고리듬의 실시간으로 구현하기 위한 알고리듬의 최적화에 대해 설명하였다. 4장에서는 인식 실험의 과정과 결과를 제시하며, 마지막으로 5장에서는 결론으로 끝을 맺었다.

2. 인식 시스템의 구성 및 특징

이 논문에서 구현한 시스템은 DHMM 기반으로 특징 벡터로는 멜 켑스트럼, 델타 멜 켑스트럼, 델타 에너지와 델타 델타 에너지 등 3개를 사용하였다[2][3]. 음성이 약 1.5초 동안 지속된다고 가정하여 끝점검출을 하였으며, 단어 모델 앞뒤에 소음 모델을 첨가함으로써 끝점 검출의 오류를 줄일 수 있었다.

2.1 인식 시스템의 특징

시스템은 Oak DSP를 사용하는 ODkit와 CODEC AD1848/6kJ(16bit, linear PCM)를 사용하는 AD/DA 변환기로 이루어진 시스템을 PC와 연결하여 구동하도록 구현하였다. Oak DSP 코어는 40 MIPS(Million Instruction Per Second)의 계산능력과 16-bit 데이터와 프로그램 버스를 지니며, 저전력으로 설계되어 있어 통신과 전자 응용분야에 좋은 성능을 지닌다[4][5][6][7].

이 코어의 메모리 구조는 각각 64 Kword의 프로그램과 데이터 메모리로 구성되어 있고, 구현한 시스템은 프로그램 메모리 1.6 Kword, 데이터 메모리 45 Kword를 사용하였다.

시스템의 실시간 프로그램의 기본구조는 더블버퍼링으로써, CODEC의 A/D 변환할 때 발생하는 인터럽트를 이용하여 구현하였다[8]. 구현된 시스템의 알고리듬 블럭도를 그림 2에 나타내었다. 구현된 인식 시스템의 소프트웨어 구성은 표 1과 같다.

2.2 소음 처리

그림 1은 100 km/h로 주행 중인 자동차의 햇빛 가리개에 마이크를 설치하고 1초 동안 녹음한 음성 파형의 스펙트로그램으로 자동차 소음 에너지가 400 Hz이하의 저주파 영역

에 집중된 유색 소음인 것을 알 수 있다. 따라서, 고역 통과 필터(High Pass Filter: HPF)를 사용하여 소음을 제거할 수 있다. 그러나, 400 Hz이하에는 음성 신호의 중요한 정보도 포함되어 있으므로, 차단 주파수의 선택에 주의가 필요하다.

다음으로 자동차 소음을 대처하기 위해 거리 측정시의 리프터를 사용하였다. 기본 인식 시스템의 거리 측정 방법인 유클리디안(Euclidian : EUC) 거리 측정 방법과 소음 환경에서 좋은 성능을 보이는 것으로 알려져 있는 가중 켑스트럴 거리 측정 방법인 RPS(Root Power Sum)[9], SLL(Smoothed Linear Lifer)[10], BPL(Band Pass Lifer)[11], GEL (General Exponential Lifer)[12]에 대하여 실험하였다.

표 1. 구현된 최종 인식 시스템

인식 단어	12단어
인식 방법	단독음(화자독립)
표본화	9.6 kHz, 16비트
프리엠퍼시스	$1 - 0.95z^{-1}$
윈도우	Hamming Window 20ms size, 10ms shift
특징벡터	13차 멜 켑스트럼 13차 델타 멜 켑스트럼 델타 에너지, 델타-델타 에너지
소음 제거 방법	High Pass Filtering
인식 알고리즘	이산 HMM(DHMM)
거리측정방법	SLL
코드워드 개수	128개(켑스트럼), 64개(에너지)
FFT 크기	1024 points
상태 수	10개

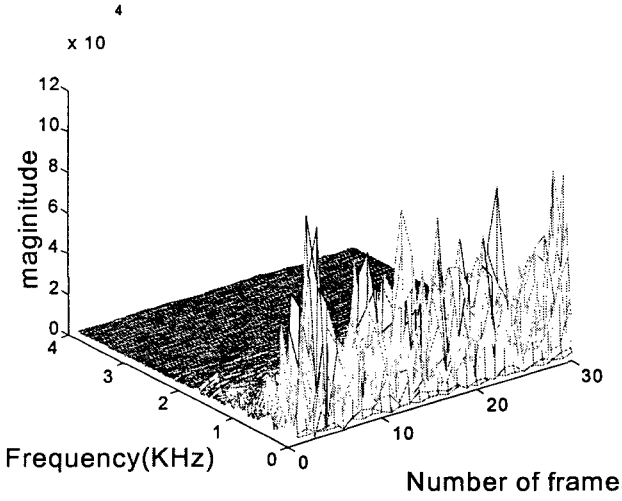


그림 1. 100 km/h의 속도로 주행 중인 자동차 소음의 스펙트로그램

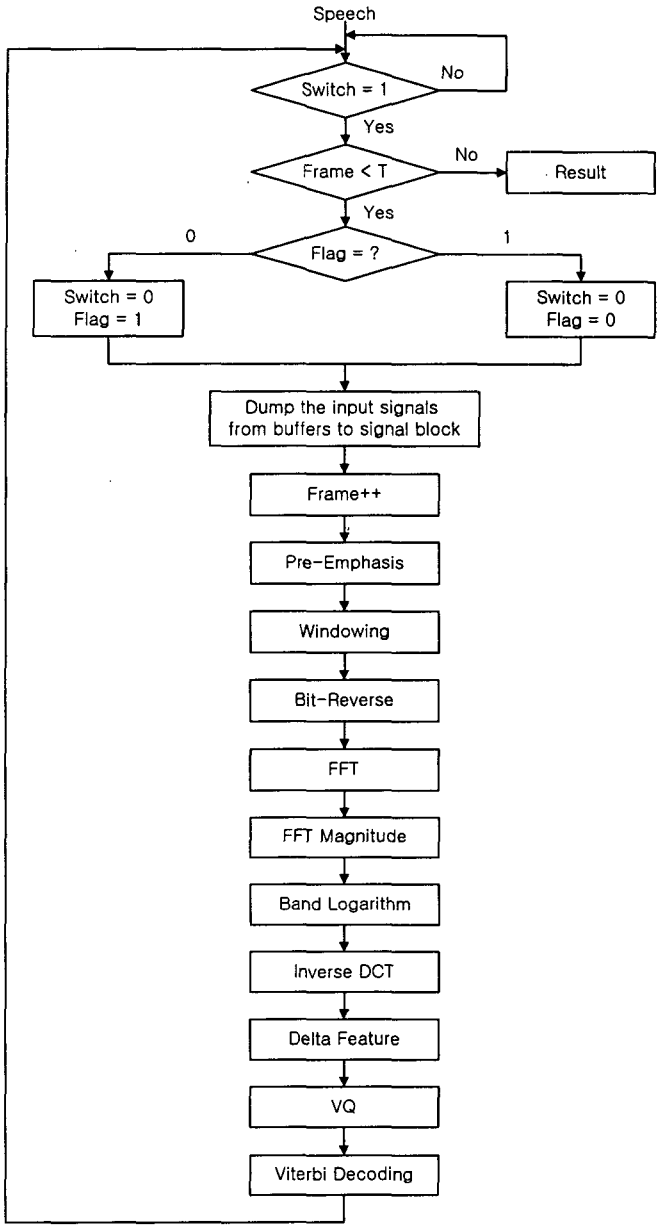


그림 2. 구현된 시스템의 알고리듬 블럭도

3. 인식 알고리듬의 실시간 구현을 위한 최적화

이 논문에서 구현한 음성 인식 시스템은 인식 과정 중에 필요한 전처리, 특징벡터 추출, 벡터 양자화, 비터비 과정을 모두 한 프레임 시간 안에 처리하도록 하였다.
특히 사용되는 프로세서는 모든 연산을 고정 소수점으로 수행하는데, 고정 소수점은 제한

된 길이의 정수로 표현하여 연산하므로 양자화 오차가 존재한다. 이러한 영향을 최적화하기 위해 고정 소수점 모의실험에서 알고리듬을 개선해야 한다. 따라서 이 논문에서는 먼저 C언어를 사용하여 부동 소수점 모의 실험을 수행하여 알고리듬을 검증한 후, 역시 C언어를 사용하여 고정 소수점 모의 실험으로 고정 소수점 음성 인식 알고리듬을 검증하고, 이를 바탕으로 실시간으로 구현 가능한 알고리듬으로 개선하였다.

DSP 구현에 있어서 가장 중요한 어셈블러 언어를 이용한 최적화 부분에서 Oak DSP 코어의 구조에 맞추어 직접 어셈블러 코딩을 함으로써, 코어가 가진 장점을 최대한 활용하였다.

3.1 특징 벡터 추출 과정

그림 2의 인식 과정 중 특징 벡터 추출 과정에서 수행할 고속 푸리에 변환(FFT)이 가장 많은 연산량이 필요함을 알 수 있다. 따라서 이에 대한 계산량 감소와 복잡한 비선형 함수인 로그 및 지수 연산에 대한 알고리듬 개선이 필요하다.

3.1.1 실수 고속 푸리에 알고리듬

입력이 음성 신호처럼 순수한 실수이므로, 그들의 대칭 정리로 인해 2N포인트 실수 DFT를 N포인트 복소수 FFT로 계산할 수 있다[13][14]. 이로 인해 계산량을 반으로 감소시킬 수 있었다. FFT의 고정 소수점 오차는 구현 알고리듬에 따라서도 달라진다. 이 논문에서는 고정 소수점 오차의 평균 소음 전력이 작고[15], 오버플로우가 일어날 확률이 적으며, DSP 코어의 MAC 연산구조와 비슷한 DIT(Decimation In Time) Radix-2 알고리듬을 선택하였다. 또한 고정 소수점 오차를 줄이기 위해 트위들 계수를 바꾸는 방법을 사용하였다[16]. FFT에 사용하는 트위들 계수 W_N^k 는 $-1.0 \le W_N^k < 1.0$의 값을 가지는데, 고정 소수점 연산에 사용되는 데이터 형식은 -1.0은 표현할 수 있지만, 1.0의 값을 정확히 표현할 수 없다. 이 현상에 의한 고정 소수점 오차를 줄이기 위해 트위들 계수의 부호를 바꾸는 방법을 사용하였다. 그러면 그림 3처럼 버터플라이 구조의 계산을 약간 수정하면 전체 알고리듬을 바꾸지 않아도 된다.

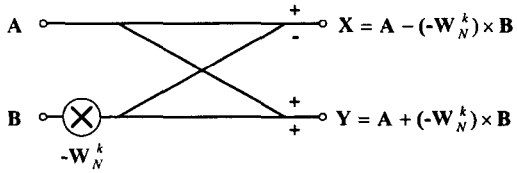


그림 3. 수정된 버터플라이 구조

이 논문에서 사용된 알고리즘에서는 각 단계의 버터플라이 그룹마다 동일한 트위들 계수를 사용하므로 트위들 계수의 주소 발생이 쉬운 장점이 있다. 트위들 계수를 메모리에 bit-reverse 순서로 저장을 하면 메모리 인덱스를 단지 선형 증가시킴으로써 트위들 계수의 주소 발생을 할 수 있다. 또한 첫 번째와 두 번째 단계의 트위들 계수값이 모두 ±1,

$\pm j$ 이므로 처음 두 개의 단계에서 곱셈을 덧셈과 뺄셈으로 모두 바꿀 수 있다. 곱셈의 횟수가 줄어들므로 그만큼 고정 소수점 오차가 줄어드는 장점이 있다.

그런데, Oak DSP 코어는 bit-reverse를 제공하지 않으므로, bit-reversed 입력 데이터들의 주소를 테이블로 저장하여 매핑시키는 방법을 사용하여 bit-reverse를 하였다.

3.1.2 로그 및 지수 연산

고정 소수점의 로그와 지수연산을 계산하는데 있어서 주로 쓰이는 알고리듬은 테일러 급수를 이용하는 방법과 수열의 수렴조건을 이용하는 방법, 그리고 테이블을 이용하는 방법이 있다[17]. 본 논문에서는 미리 계산 결과를 테이블로 만들어 놓고 입력 값에 따라 테이블에서 결과 값을 찾는 방법을 이용했다. 이는 아무리 복잡한 연산이라도 짧은 시간에 수행할 수 있지만 입력값의 범위만큼 메모리가 요구되는 단점이 있다.

3.1.2.1 로그 연산

먼저, 로그 함수의 입력과 출력의 데이터 형태를 결정해야 하는데, 입력 x 는 16 비트의 signed 형태($0 \le x \le 2^{15}$)를 갖고, 출력은 정수 11 보다 작은 값이므로, 5.11 signed 형태를 선택하였다.

16 비트 signed 형태를 갖는 로그 함수 입력 x 는 -1 에서 1 사이의 값을 갖는 *mantissa*와 0에서 15 사이의 값을 갖는 *exponent*를 이용하여 $x = mantissa \times 2^{exponent}$와 같이 나타낼 수 있다. 이 때 *mantissa* 는 1.15 singed 형태의 값으로서 최상위 비트는 sign 을 나타낸다. 이 식의 양변에 로그 연산을 취하게 되면,

$$10 \log_{10} x = 10 \log_{10} mantissa + exponent \times 10 \log_{10} 2 \qquad (1)$$

테이블을 이용하여 로그 함수를 구할 경우에는 식 (1)의 $10 \log_{10} mantissa$와 $exponent \times 10 \log_{10} 2$ 값을 테이블에 저장한 뒤, *mantissa*와 *exponent*에 따라 각각의 위치에 저장되어 있는 테이블 값을 더함으로써 연산을 수행하게 된다. 이 경우 *exponent*는 0과 15 사이의 정수 값을 갖기 때문에 $exponent \times 10 \log_{10} 2$는 16개의 값만을 갖게 된다. 그러나 *mantissa*는 16비트의 해상도(resolution)를 갖고 있기 때문에 $10 \log_{10} mantissa$를 저장하기 위해서는 2^{16}의 방대한 메모리가 필요하다. 테이블에 사용되는 메모리를 줄이기 위해 다음과 같은 선형 보간법(Linear Interpolation)을 이용한 근사화된 방법을 사용한다.

$$\begin{aligned} 10 \log_{10} x &= 10 \log_{10} mantissa + exponent \times 10 \log_{10} 2 \\ &\approx mantissa_{table} + difference[mantissa_{table}] \times (fractions\ of\ mantissa) \end{aligned} \qquad (2)$$

$mantissa_{table}$은 9개의 번지를 갖는 16비트 테이블이다. $mantissa_{table}$은 *mantissa*의 상위 5비트를 뽑아서, 그 중 최상위 sign 비트를 제외한 나머지 4 비트를 $mantissa_{table}$의 번지로 사용한다. 실제 *mantissa*의 개수보다 작은 수의 표를 사용하는 데서 발생하는 오차는 로그

함수를 구간 선형 함수로 근사함으로써 줄일 수 있다. $difference[mantissa_{table}]$은 *mantissa*에 의해서 선택된 $mantissa_{table}$값과 바로 이웃하는 큰 값과의 차이를 나타낸다. 그리고, 상위 5비트를 제외한 11비트를 이용하여 *fractions of mantissa*를 표현하는데, 이는 16비트의 *mantissa*를 4비트로 근사화 함으로써 발생하는 오차를 의미한다. 이런 과정을 계산한 결과 데이터는 모두 5.11 signed 형태를 갖는다.

3.1.2.2 지수 연산

앞 절에서 설명한 로그 함수처럼 지수 함수의 입력과 출력의 데이터 형태를 결정해야 하는데, 입력 x 는 32 비트의 signed 형태($0 \le x \le 2^{31}$)를 갖고, 출력은 signed 형태인 16비트 정수값을 선택하였다.

32 비트 signed 형태를 갖는 지수 함수 입력 x 는 0 에서 1 사이의 값을 갖는 *mantissa*와 0에서 31 사이의 값을 갖는 *exponent*를 이용하여 $x = mantissa \times 2^{exponent}$ 와 같이 나타낼 수 있다. 이 때 *mantissa*는 1.15 singed 형태의 값으로서 최상위 비트는 sign을 나타낸다. 이 식의 양변에 지수 연산을 취하게 되면,

$$|x|^{\frac{1}{2}} = |mantissa|^{\frac{1}{2}} \times 2^{\frac{exponent}{2}} \qquad (3)$$

테이블을 이용하여 지수 함수를 구할 경우에는 위에서 설명한 로그 함수에서처럼 *mantissa*가 32 비트 해상도를 가지므로 이를 저장하기 위해서는 2^{32}의 방대한 메모리가 필요하므로, 테이블에 사용되는 메모리를 줄이기 위해 다음과 같은 선형 보간법(linear interpolation)을 이용한 근사화된 방법을 사용한다.

$$\begin{aligned} |x|^{\frac{1}{2}} &= |mantissa|^{\frac{1}{2}} \times 2^{\frac{exponent}{2}} \\ &\approx mantissa_{table} + difference[mantissa_{table}] \times (fractions\ of\ mantissa) \end{aligned} \qquad (4)$$

식 (4)의 $mantissa_{table}$은 12개의 번지를 갖는 16비트 테이블로써, 로그 함수처럼 최상위 sign 비트를 제외한 나머지 상위 4 비트를 $mantissa_{table}$의 번지로 사용한다. 그리고, 상위 5비트를 제외한 15비트를 이용하여 *fractions of mantissa*를 표현하여 계산한다.

그림 4는 최적화 과정을 거친 멜 캡스트럼의 고정 소수점 오차를 나타낸다. 그림 (b)에 나타난 오차는 소수점 아래 7번째 비트의 차이에 해당한다. 따라서 이 논문에서 제안한 최적화 알고리듬에 의해 고정 소수점 오차의 영향이 많이 줄어들어 부동 소수점으로 계산한 결과와 거의 차이가 없음을 알 수 있다.

3.2 델타 특징 벡터 추출

스펙트럼의 순간적인 변화를 나타내는 델타 특징 벡터를 본 논문에서는 실시간 구현을 위해 식 (5)처럼 비교적 계산량이 적은 프레임간의 차분에 의해서 구현하였다.

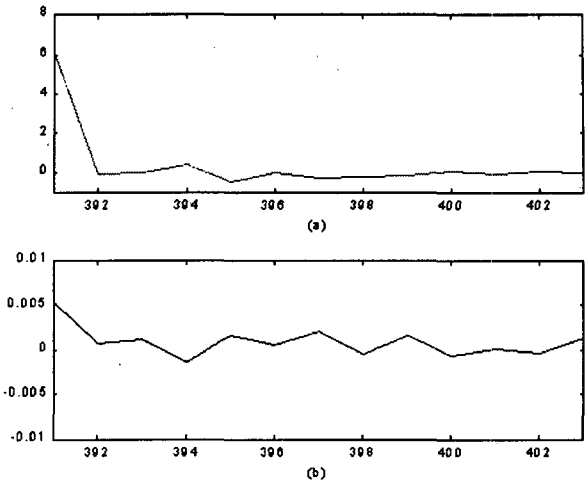


그림 4. 특징 벡터 추출 과정의 고정 소수점 오차 최적화 결과
(a) 부동 소수점과 고정 소수점 연산 결과
(b) 부동 소수점과 고정 소수점 연산 결과와의 오차

특징 벡터인 켑스트럼은 6.10 signed 형태이므로 델타 특징 벡터인 델타 켑스트럼과 델타, 델타 델타 에너지도 역시 6.10 signed 형태를 갖는다.

그런데 처음 몇 개의 특징 벡터들의 델타 값은 차분할 대상이 없으므로 0으로 대체한다. 이런 영향으로 델타 켑스트럼은 4프레임, 델타, 델타 델타 에너지는 6프레임의 지연을 갖는다.

구현한 델타 특징 벡터 추출의 블록도를 그림 5에 나타내었다.

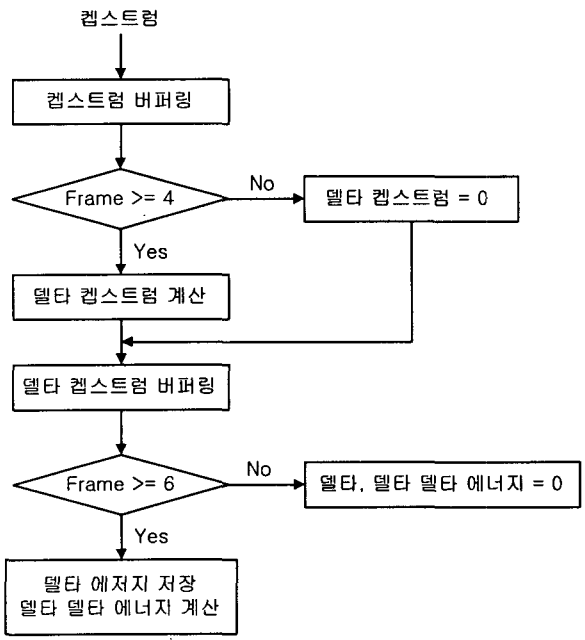


그림 5. 델타 특징 벡터 추출의 블록도

$$\widehat{\Delta c_k}(t) = c_k(t+\delta) - c_k(t-\delta) \qquad (5)$$

3.3 벡터 양자화

구현한 인식 시스템은 DHMM을 이용하므로 벡터 양자화를 해야 하는데 이는 코드북의 생성과 양자화된 관찰열의 생성과정으로 나뉜다. 이때 이용하는 코드북의 크기는 켑스트럼은 128개, 에너지는 64개이며 LBG 알고리듬[18]을 이용하여 생성하였다.

구현을 위해 생성된 코드북은 켑스트럼과 델타 켑스트럼은 1.15 signed 형태, 델타, 델타 델타 에너지는 4.12 signed 형태로 16비트 데이터를 테이블로 저장하였다.

관찰열들을 양자화하기 위해서 3장에서 설명한 소음의 영향에 강인한 SLL 거리 측정 방법을 사용하였다. 이때 가중 함수의 계수는 4.12 signed 형태이다. 그런데 거리 측정시 거리의 상대적인 값을 비교하므로, 계산량이 많은 식 (6) 을 사용하지 않고 식 (7)처럼 절대값을 사용하여 거리를 측정하였다.

$$d(c, c') = \sum_{k=1}^{p} w_k^2 (c_k - c_k')^2 = \sum_{k=1}^{p} (w_k (c_k - c_k'))^2 \qquad (6)$$

$$d(c, c') = \sum_{k=1}^{p} | w_k (c_k - c_k') | \qquad (7)$$

3.4 비터비(Viterbi) 알고리듬

이 논문에서는 모든 인식 과정을 한 프레임 안에 처리해야 하므로, 프레임별 비터비 알고리듬을 구현하기 위해 두 개의 버퍼를 서로 스위칭을 하면서 누적된 확률값을 구한다. 이렇게 구한 누적된 확률 중에서 관찰열 O가 가질 수 있는 최적의 상태열에서 얻어지는 최적의 확률을 구하는 방법을 사용하였다.

여기서 이용되는 모델의 천이확률 a는 1.15 × 16 unsigned, $1-a$는 3.13 unsigned, 관찰확률 b는 3.13 unsigned 형태로 테이블을 구성하여 사용하였다.

4. 인식 실험 및 결과

4.1 데이터 베이스

이 논문의 인식 대상은 차량 항법 시스템에서 사용되는 12단어로 구성된다. 인식 시스템의 학습을 위해서 남녀 51명이 정지해 있는 차량에서 각 인식 대상 단어를 시동을 끄고 3번, 시동을 켜고 3번씩 발음한 총 3672 단어를 사용하였고, 인식 성능의 평가를 위해서 남녀 19명이 시동 켜고 정지해 있는 차량에서 각 단어를 1번 발음한 228단어와 100 km/h로 주행 중인 차량에서 2번씩 발음한 456 단어를 사용하였다. 음성 신호는 차량 조수석의 햇빛 가리개에 부착한 해드셋 마이크를 사용하여 녹음하였다. 인식 대상 단어는 표 2에 나타내었다.

표 2. 인식 대상 단어

index	command	index	command
0	텔레비전	6	지도색상
1	네비게이션	7	위성정보
2	비디오	8	주행궤적
3	오디오	9	헤드업
4	메뉴	10	노쓰업
5	환경설정	11	표준

4.2 인식 결과

4.2.1 기본 인식 시스템의 인식 결과

기본 인식 시스템의 자동차 정지 중 및 주행 중의 인식 성능은 표 3과 같다. 표 3에서 "STOP"은 정지 중에 녹음된 시험 데이터에 대한 인식률을 나타내며, "RUN"은 주행 중에 녹음된 시험 데이터에 대한 인식률을 나타낸다. 소음이 없는 정지 상태에서의 인식률은 우수하나, 자동차 주행 소음이 있는 상황에서는 인식률이 크게 저하됨을 알 수 있다.

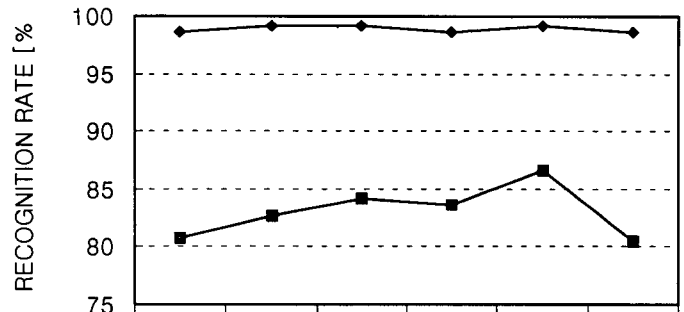
표 3. 기본 인식 시스템의 인식률(%)

	STOP	RUN
RECOGNITION RATE[%]	98.68	80.7

4.2.2 소음 처리 1 : 고역 통과 필터링

자동차 주행 소음은 그림 1처럼 400 Hz 이하의 저주파 대역에 집중 분포되어 있다. 따라서, 고역 통과 필터(HPF)를 사용하여 소음을 제거할 수 있다. 그러나 400 Hz 이하에는 음성 신호의 중요한 정보도 포함되어 있으므로, 차단 주파수의 선택에 주의가 필요하다. HPF의 차단 주파수에 따른 인식 성능을 그림 6에 나타내었다.

그림 6의 인식률을 보면 HPF의 차단 주파수가 300 Hz 이상이면 정지 중과 주행 중의 인식률이 저하됨을 볼 수 있다. 이것은 음성 신호에 손상이 가해지는 것을 의미하므로, 이보다 낮은 250 Hz를 적절한 차단 주파수로 선정하였다.

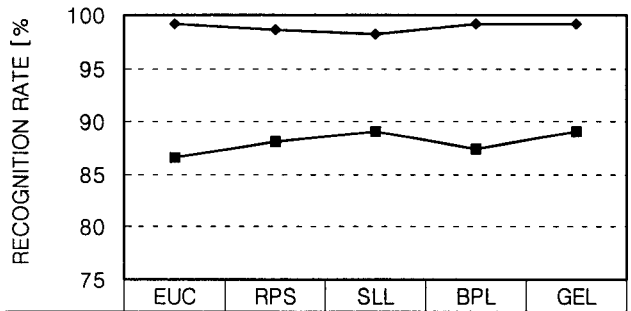


	0Hz	100Hz	150Hz	200Hz	250Hz	300Hz
STOP	98.68	99.12	99.12	98.68	99.12	98.68
RUN	80.7	82.68	84.21	83.55	86.62	80.48

그림 6. HPF의 차단 주파수에 따른 인식률(%)

4.2.3 소음 처리 2 : 리프터링

자동차 주행 소음에 대한 대비책 중 하나로 소음에 강인한 특징 벡터의 거리 측정 방법을 인식 시스템에 추가하였다. 특히 리프터링은 실시간 구현시 많은 계산량과 복잡성 없이도 인식률을 크게 높일 수 있는 점이 장점이다. 기본 인식 시스템의 거리 측정 방법인 유클리디안 거리 측정 방법과 소음 환경에서 좋은 성능을 보이는 것으로 알려져 있는 가중 켑스트럴 거리 측정 방법인 RPS, SLL, BPL, GEL에 대한 인식 성능을 비교하여 그림 7에 나타내었다.



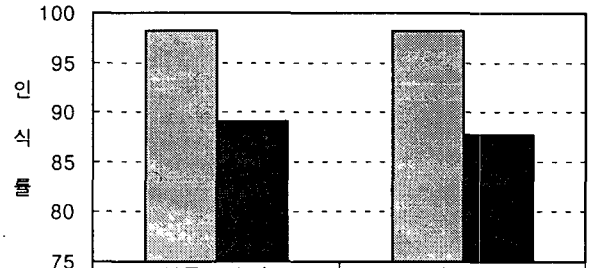
	EUC	RPS	SLL	BPL	GEL
STOP	99.12	98.68	98.25	99.12	99.12
RUN	86.62	88.16	89.04	87.5	89.04

그림 7. 거리 측정 방법에 따른 인식률(%) (HPF 사용 중)

그림 7에서 보면 250 Hz의 차단 주파수를 갖는 HPF와 함께 사용했을 경우 좋은 성능을 보인 SLL 거리 측정 방법을 선정하였다.

4.2.4 고정 소수점 모의 실험

기본 인식 시스템에 자동차 주행 소음을 제거하기 위해 사용했던 소음 처리 기법들을 적용한 인식 시스템을 Oak DSP 코어를 이용하여 실시간 구현을 위해서, 3장에서 논의했던 알고리듬 최적화를 통해 고정 소수점 모의 실험을 하였다. 실험 결과를 그림 8에 부동 소수점과 비교하여 나타내었다.



	부동 소수점	고정 소수점
STOP	98.25	98.25
RUN	89.04	87.75

그림 8. 인식 시스템의 부동 소수점과 고정 소수점 모의 실험

그림 8을 보면 고정 소수점 연산시 양자화 오차로 인해 주행 중 인식률이 1.29 % 정도 차이가 남을 볼 수 있다. 따라서 위의 결과로 인식 시스템에 사용한 소음 처리 알고리듬에 대한 검증을 확인 할 수 있었다.

4.3 구현된 음성 인식 시스템의 실시간 구현 검증

시스템에서 사용되는 클럭 주파수(f_{clk})와 샘플링 주파수(f_s)가 결정되면 한 프레임에 소요되는 클럭수를 결정할 수 있다. 클럭 주파수가 40 MHz 시스템에서 9.6 kHz로 샘플링 데이터를 처리하려면, 하나의 음성 샘플에 대응되는 클럭수는 다음과 같다.

$$\frac{1}{f_s} \times f_{clk} = \frac{1}{40 \times 10^3} \times 9.6 \times 10^3 = 4,166$$

그런데, 인식 과정을 20 ms(200 샘플)길이의 프레임 단위로 10 ms(100 샘플)씩 이동하면서 처리하므로, 한 프레임 처리에 사용할 수 있는 전체 클럭수는 416,600개이다. 따라서 구현된 음성 인식 시스템이 실시간으로 동작하기 위해서는 416,600개의 클럭 안에 한 프레임에 대한 모든 인식 과정이 이루어져야 한다.

Oak DSP 코어의 어셈블리 언어를 사용하여 구현된 전체 인식 과정의 각 작업당 소요 사이클 수를 표 4에 나타내었다.

표 4를 보면 특히 특징벡터 추출 과정 중 FFT가 125,559 클럭으로써 전체 중 49.3 % 정도로 가장 많은 계산량이 필요함을 볼 수 있다.

구현된 인식 시스템의 실제 계산량은 전체 허용 계산량의 약 70 % 정도 차지하므로 본 시스템은 실시간으로 인식 과정을 수행할 수 있다.

본 시스템에서는 총 메모리 45 Kword를 사용하였는데, 각각의 단어 모델의 초기 확률, 천이 확률, 관찰 확률에 필요한 메모리가 총 메모리의 약 83 %를 차지하고 있다. 표 5에서는 인식 시스템의 메모리 요구량을 나타내었다.

표 4. 인식 함수의 소요 클럭수

함수	소요클럭수(clock)	점유율(%)
프리엠퍼시스	1,612	0.39
윈도윙	612	0.15
특징벡터 추출	177,683	42.65
백터 양자화	87,155	20.92
비터비 디코딩	20,854	5.01
인터럽트 서비스 루틴	4,000	0.96
총 클럭수	291,916	70.07

표 5. 인식 시스템의 메모리 요구량

구분	메모리 크기(word)
인식에 필요한 변수	110
스택	256
인식에 필요한 테이블	1,606
인식에 필요한 버퍼	75
FFT를 위한 버퍼	2,048
코드북	3,200
천이확률	228
관찰확률	38,400
비터비 디코딩을 위한 버퍼	480
샘플링된 신호 저장 버퍼	512
총 메모리량	46,915

5. 결 론

이 논문에서는 자동차 환경에서 차량 항법용 화자 독립 음성 인식 시스템의 단일 칩 개발을 위해 Oak DSP 코어를 사용하여 음성 인식 시스템을 실시간 구현하였다. 개발 과정에서 인식 알고리듬의 계산량 감소와 고정 소수점 연산 과정에서의 양자화 오차를 최소화하기 위해 알고리듬을 개선하였고, 실질적으로 시스템 구현할 때 문제점에 대해 논의하였다.

자동차 주행 소음에 대처하기 위해서 고역 통과 필터링과 기존의 리프터링을 적용한

결과, 주행 중 인식률이 80.7 %에서 89.04 %로 8.34 %의 인식 성능 향상을 얻을 수 있었다. 모의 실험 결과를 보면 고정 소수점 연산할 때 양자화 오차의 영향으로 부동 소수점 연산 실험 결과와 약 1.29 % 정도 차이가 남을 볼 수 있다.

구현된 시스템의 실시간 동작을 확인하기 위해 소요 클럭수를 측정한 결과, 인식 시스템의 실제 계산량은 전체 허용 계산량의 70.07 %를 점유함으로 실시간으로 인식 과정을 수행할 수 있음을 검증하였다.

추후 과제로는 실시간 허용 계산량 범위 안에서 주행 중 인식률을 더욱 향상시키기 위한 새로운 소음 처리 알고리듬의 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] M. Shozakai, S. Nakamura, K. Shikanp. 1998. "Robust Speech Recognition in Car Environment." in *Proc. Int. Conf. Acoust. Speech, Signal Processing,* vol. 1, pp. 269-272.

[2] 정의상, 최홍섭. 1997. "전화음성에 강인한 문장종속 화자인식에 관한 연구." *음성과학 2,* pp. 57-66.

[3] 구명완. "음성인식 기술을 이용한 열차 정보 안내 서비스." *음성과학회 제7회 학술발표 회논문집*, pp. 239-245.

[4] OakDSPCoreTM Architecture Specification.

[5] OakDBG^{TM} User's Manual.

[6] ODKit & CDI Technical Reference OakDSPCore Development Chip Specification.

[7] DSP Group Assemblers & Linker User's Manual.

[8] 정익주. 1998. "핵심어 인식을 이용한 음성 자동 편집 시스템 구현." *음성과학 3,* pp. 119-131.

[9] B. A. Hanson & H. Wakita. 1987. "Spectral Slope Distortion Measures with Linear Prediction Analysis for Word Recognition in Noise." *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-35, no. 7, pp. 968-973.

[10] F. Itakura & T. Umezaki. 1987. "Distance Measure for Speech Recognition based on the Smoothed Group Delay Spectrum," in *Proc. Int. Conf. Acoust. Speech, Signal Processing*, vol. 3, pp. 1257-1260.

[11] B. H. Juang, L. R. Rabiner, & J. G. Wilpon. 1987. "On the Use of Bandpass Lifering in Speech Recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, no. 7, pp. 947-954.

[12] J. Junqua & H. Wakita. 1989. "A Comparative Study of Cepstral Lifers and Distance Measures for All Pole Models of Speech in Noise," in *Proc. Int. Conf. Acoust. Speech, Signal Processing*, vol. 1, pp. 476-479.

[13] Henrik V. Sorensen, Douglas S. Jones, Michael T. Heideman, C. Sidney Burrus. 1987. "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on Acoust Speech and Signal Processing*, vol. ASSP-35, no. 6, pp. 849-863.

[14] "TMS320C54x DSP Applications Guide." 1996. Texas Instrument.

[15] Raimund Meydr. 1989. "Error Analysis and Comparision of FFT Implementation Structures," in *Proc. Int. Conf. Acoust. Speech, Signal Processing,* vol. 2, pp. 888-891.

[16] Andreas Elterich & Walter Stammler. 1988. "Error Analysis and Resulting Structural Improvement for Fixed Point FFTs," in *Proc. Int. Conf. Acoust. Speech, Signal Processing,* vol. 3, pp.1419-1422.

[17] 정남훈. 1996. *VLSI를 이용한 MPEG-2 오디오 부호화기 설계.* 연세대학교 석사학위 논문.

[18] Y. Linde, A. Buzo, R. M. Gray. 1980. "An Algorithm for Vector Quantizer Design," *IEEE Trans. Comm.*, vol. 28, no. 1, pp. 84-95.

접수일자: '99. 9. 5.
게재결정: '99. 10. 27.

▲ 우 경 호
서울특별시 서대문구 신촌동 134
연세대학교 공과대학 전기·컴퓨터공학과
음향 음성 및 신호처리 연구실(우: 120-749)
Tel: (02) 361-2863 (O)
e-mail: wkh@radar.yonsei.ac.kr

▲ 양 태 영
서울특별시 서대문구 신촌동 134
연세대학교 공과대학 전기·컴퓨터공학과
음향 음성 및 신호처리 연구실(우: 120-749)
Tel: (02) 361-2863 (O)
e-mail: tyyang@ghost.yonsei.ac.kr

▲ 이 충 용
서울특별시 서대문구 신촌동 134
연세대학교 공과대학 전기·컴퓨터공학과
음향 음성 및 신호처리 연구실(우: 120-749)
Tel: (02) 361-2863 (O)
e-mail: clee@radar.yonsei.ac.kr

▲ 윤 대 희
서울특별시 서대문구 신촌동 134
연세대학교 공과대학 전기·컴퓨터공학과
음향 음성 및 신호처리 연구실(우: 120-749)
Tel: (02) 361-2863 (O)
e-mail: dhyoun@radar.yonsei.ac.kr

▲ 차 일 환
서울특별시 서대문구 신촌동 134
연세대학교 공과대학 전기·컴퓨터공학과
음향 음성 및 신호처리 연구실(우: 120-749)
Tel: (02) 361-2863 (O)
e-mail: iwcha@radar.yonsei.ac.kr