한국산업정보학회논문지 제5권 제2호 (2000.6)

# 백터양자화기의 신속코더백터 찾기
# (Fast Codevector Search on Vector Quantization)

우 홍 체*
(Hong-Chae Woo)

**요 약** 백터 양자화기는 음성 부호화, 오디오 부호화, 그리고 비디오 부호화와 같은 많은 고품질 고전송률 데이터 압축응용에서 널리 사용되고 있다. 백터 양자화기의 코더북의 크기가 매우 클 때, 코더북 전체를 찾는 방식은 많은 응용의 경우에서 계산량 때문에 상당한 문제점이 된다. 계산량을 낮추기 위하여 삼각형의 변 길이에 대한 부등식과 같은 코더북의 특성을 활용하는 많은 알고리즘들이 제안되고 연구되어 왔다. 본 논문에서는 최적의 코더백터를 찾기 위하여 다단구조에 기반한 신속 코더백터 찾기 알고리즘을 제안하고자 한다. 간단한 2 단계 구조의 이 알고리즘을 사용하여도 상당한 계산 복잡성을 압축대상의 품질을 손상시키지 않고 줄일 수 있다.

**Abstract** Vector quantization(VQ) is widely used in many high-quality and high-rate data compression applications such as speech coding, audio coding, image coding and video coding. When the size of a VQ codebook is large, the computational complexity for the full codeword search method is a significant problem for many applications. A number of complexity reduction algorithms have been proposed and investigated using such properties of the codebook as the triangle inequality. This paper proposes a new fast VQ search algorithm that is based on a multi-stage structure for searching for the best codeword. Even using only two stages, a significant complexity reduction can be obtained without any loss of quality.

## 1. 서 론

정보화 사회의 발전에 따라서 더욱 고품질 및 고전송률의 음성, 오디오, 영상, 그리고 비디오 응용제품들이 개발되고 있다. 이러한 제품들에서 고압축율을 가지는 백터양자화(vector quantization) 방식은 더욱 많이 사용되고 있다. 이 백터양자화는 매우 강력한 데이터 압축방식으로 정지영상 압축 방식의 국제표준인 JPEG(joint picture experts group) 및 동영상 압축 방식의 국제표준인 MPEG(moving picture experts group) 등에서 사용되고 있다. 또한 저가격의 고성능 컴퓨터와 다양한 디지털신호처리 소자의 발전은 더욱 백터부호화기의 사용을 용이하게 하고 있다. 하지만 대용량의 코더북(codebook)에서는 큰 저장공간 및 높은 계산량이 아직도 백터부호화기의 다양한 응용에 걸림돌이 되고 있다.

* 대구대학교 정보통신공학부

이제 백터부호화기의 기본 원리에 대하여 간략하게 살펴본다. 백터부호화기의 압축방식은 원정보에 손실은 가지는 압축방식으로 백터부호화기에 들어오는 여러 개의 입력을 손실을 최소화할 수 있는 대표 값으로 근사 시키는 방식이다. 이때 대표 값들은 Linde-Buzo-Gray(LBG) 알고리즘과 같은 코더북 설계과정을 통하여 얻어진다. LBG 알고리즘에서는 코더북 크기의 수십 배에 달하는 훈련 백터로부터 최근사 이웃(nearest neighbor) 조건과 중심값(centeroid) 조건을 사용하는 반복적인 방법으로 최적한 코더북을 얻는다. 얻어진 코더북은 입력백터를 부호화하기 위하여 저장된 후 사용된다 [1].

주어진 코더북에서 입력에 가장 근사한 코더백터(codevector)를 찾는 과정은 기본적으로 코더북에 있는 모든 코더백터를 입력과 비교하여 가장 오차가 적은 코더백터를 얻는 과정이다. 입력과 코더백터를 비교할 때 오차계산의 기준으로 흔히 최소 유클리디안(Euclidean) 거리 기준을 사용한다. 따라서 백터양자화는 $k$ 차원 입력 백터 집합 $X$를 같은 차원의 코더북 $C$로 맵핑(mapping)하는 과정이라 할 수 있다. 백터양자화기의 부호화기는 다음과

같이 나타낼 수 있다.

부호화기: $X \rightarrow C$

여기서 입력백터, $x_i$의 집합은 $X=\{x_i\}$, $i=1,\cdots,L$로 표현되며, 코더백터, $c_i$의 집합은 $C=\{c_i\}$, $i=1,\cdots,N$로 나타낸다. $L$은 입력백터의 개수이며, $N$은 코더백터의 개수이다. 부호화를 위해 $b$ 비트(bit)를 사용하면, $N$은 $2^b$ 값을 가진다. 즉 부호화기는 임의의 입력백터 $x_i$가 들어오면 $x_i$와 모든 $c_i$와 비교하여 유클리디안 거리로 볼 때 가장 가까운 하나의 코더백터, $c_j$를 선택한다. $c_j$의 인덱스(index), $j$는 복호기로 보내어진다.

백터양자화기에서 복호화 과정은 매우 간단한 테이블 보기(table look-up) 과정이다. 받아들여진 인덱스, $j$에 해당되는 코더백터를 코더북에서 보면 된다. 그래서 부호화기의 계산량이 최적 코더백터 찾기 과정 때문에 복호화기보다는 부호화기에 많이 필요하게 된다. 백터부호화기를 실제 제품에서 사용하고자 할 때는 필요한 전체 계산량을 줄임으로써 저가격의 제품을 개발할 수 있게 된다.

## 2. 신속 코더백터 찾기 알고리즘

코더북은 $N$개의 코더백터를 가지며, 코더백터의 차원은 $k$이며, 최적 코더백터를 찾기 위한 기준으로는 최소평균자승오차(minimum mean square error, MMSE) 기준을 본 연구에서는 사용한다. 일반적 전체 찾기(full search) 알고리즘은 $N\times k$번의 곱셈(multiplication), $N\times(2k-1)$번의 덧셈(addition) 및 뺄셈(subtraction), 그리고 $(N-1)$번의 비교(comparison)만큼의 계산량을 필요로 한다. 코더북의 크기, $N$ 혹은 코더백터의 차원, $k$가 증가할수록 백터양자화기의 전체 계산량은 매우 커지게 되어 실제 시스템에서 전체 찾기 알고리즘의 구현이 어려워진다.

예를 들면, 10차원 입력백터를 16 비트로 백터양자화한다면, $N$은 $2^{16}$이므로, 코드북의 크기는 65536이 되고, 이때 부호화기에서 필요한 곱셈회수는 $N\times k=65536\times 10$으로 약 65만이나 된다. 이러한 문제를 해결하기 위해 전체 찾기 알고리즘의 계산량을 낮출 수 있는 다양한 코드백터 찾기 알고리즘들이 개발되어 왔다. 이러한 알고리즘들 대부분에서 사용하는 기본적인 아이디어는 예비 과정을 통하여 자승오차를 계산할 대상 코드백터의 개수를 줄이는 것이다. 이제 이러한 알고리즘들에 대하여 살펴보기로 한다.

PDE(partial distortion elimination) 알고리즘은 $j$ 번째까지의 코더백터의 요소와 입력백터사이의 자승오차합이 현재 최소 자승오차합보다 크게되면 현재 코더백터는 최적 코더백터가 될 수 없으므로 즉시 현재 코더백터에 대한 오차계산을 중단하여 계산량을 줄이는 방식이다 [2]. PDE 알고리즘은 다음과 같이 작동한다.

- 현재 최소자승오차합, $d_{\min}$을 $\infty$로 설정한다. 실제로는 매우 큰 값으로 설정한다.
- 입력백터 $x_i$의 첫 번째 요소, $x_{i1}$와 첫 번째 코드백터, $c_1$의 첫 번째 요소 $c_{11}$ 사이의 자승오차, $d_{temp}$를 구한다. 즉, $d_{temp}=d_{temp}+(x_{i1}-c_{11})^2$를 계산한 후, $d_{temp}>d_{\min}$을 판별한다.
- 부등식이 성립하면, 현재 코드백터, $c_1$에 대한 자승오차 계산을 중단하고 입력백터와 두 번째 코더백터, $c_2$와 자승오차 계산을 새롭게 시작한다. 여기서 $d_{temp}$는 새로운 코드백터 만날 때마다 0이 된다.
- 부등식이 성립되지 않으면 두 번째 요소에 대한 자승오차를 현재 자승오차합 $d_{temp}$에 더하게 된다. 즉 $d_{temp}=d_{temp}+(x_{i2}-c_{12})^2$를 계산하여 다시 부등식 $d_{temp}>d_{\min}$를 조사하여 부등식이 성립하면 다음 코더백터에 대한 자승오차 계산으로 넘어가고, 그렇지 않으면 세 번째 요소에 대한 자승오차 계산를 계속한다.
- $k$ 번째 요소까지의 자승오차합 계산 후에도 부등식이 성립되지 않으면 현재 최소자승 오차합, $d_{\min}$을 $d_{temp}$로 교체한다.
- 최적 코더백터를 찾기위한 위의 반복과정은 $N$번째 코더백터에 도달할 때까지 한다.

오차값의 일부분만을 계산하는 PDE 알고리즘은 초기 단계에서 현재 최소 자승오차합이 최종 최소 자승오차합에 근사할 때 매우 효과적이다. 기본 PDE 알고리즘에 초기 최소 자승오차합에 대하여 예측 과정을 포함하여 보다 효과적으로 개선된 PDE 방식이 예측(predictive) PDE 알고리즘이다 [3].

계산량 감소 알고리즘에서 사용하는 다른 하나의 기본 원리는 삼각형에 대한 부등식이다. 삼각형에서 "두 변의 합은 빗변보다 길다"는 삼각부등식(triangle inequality)를 사용하면 최적 코더백터를 찾는 과정에서 자승오차 계산을 해야하는 코더백터 수를 많이 줄일 수 있게 된다. 코더백터 $c_i$가 $d(c_i, c_j) > 2\cdot d_{\min}$를 만족하면 $c_i$는 최적 코더백터 후보가 되지 못한다. 여기서 $d(c_i, c_j)$는 두 코더백터 사이의 자승오차이며, $c_j$는 최적 코더백터를 찾는

과정에서 입력백터에 대한 현재까지의 최적 코더백터이며, 그리고 $d_{\min}$은 현재까지 찾아진 최소 자승오차합이다. 이 알고리즘에서는 모든 코더백터 쌍들 사이의 자승오차 $d(c_i, c_j)$, $1 \le i \le N$ and $1 \le j \le (N-1)$를 구하여 저장하고 있다. 따라서 $N \times (N-1)$ 개의 저장공간을 가지는 메모리가 더 필요하게 된다 [4].

PSPD(partial search partial distortion) 알고리즘은 각 코더백터의 평균값을 저장하여 계산량 감소에 활용한다. 코더백터의 평균값이 아래 식에서 주어지는 최대 평균값, $m_{\max}$과 최소 평균값 $m_{\min}$사이에 있을 때만 이들 코더백터를 최적 코더백터의 후보로 생각하여 자승오차를 계산한다. $m_{\max}$와 $m_{\min}$에 대한 식은 다음과 같다.

$$m_{\max} = m_{x_i} + \frac{d_{\min}}{\sqrt{k}}$$

$$m_{\min} = m_{x_i} - \frac{d_{\min}}{\sqrt{k}}$$

여기서 $m_{x_i}$는 현재 입력백터, $x_i$에 대한 평균이다. 정해진 평균의 범위에 들어오는 코더백터들에 대하여서만 PDE 알고리즘을 적용하는 방식이 PSPD 알고리즘이다 [5].

코더백터를 미리 여러 집단으로 분류하여 계산량 감소를 얻는 방식으로 분류 예비선택(classified pre-selection) 방식이 있다. 이 알고리즘에서는 각 코더백터 집단들의 중심점 값을 저장하고 있다. 전체 코더백터에서 최적 코더백터를 찾지 않고, 먼저 입력백터와 가까운 3개 혹은 4개의 집단의 중심점들과 자승오차를 계산하여 가장 가까운 집단을 찾고 그 집단에 속하는 코드백터들에 대하여서만 전체 코드백터 찾기를 하는 방식이다 [6]. 이 방식에서는 전체 코드백터 찾기 방식에 비교할 때 최적 코더백터를 항상 찾지 못할 수도 있기 때문에 이 방식을 적용하는 과정에서 추가적인 오차를 백터양자화기에 가져올 수 있다.

최적 코더백터를 찾는 과정에는 계산량을 줄일 수 있는 접근 방식으로 제한되지 않은(unconstrained) 코더북에 대한 제한된 찾기 과정을 가지는 알고리즘이 있다. 하지만 역사적으로 tree-structured VQ, residual VQ, multi-stage VQ 등과 같은 제한된 코더북을 사용하여 계산량을 줄이는 방식이 크게 성공을 하였다. 하지만 백터양자화 코드북 자체에 제한을 하면 전체 양자화 성능은 나빠지기 마련이다. 이제 제한되지 않은 코더북에 대한 제한된 찾기 과정을 통하여 계산량을 줄일 수 있는 새로운 알고리즘을 제안하고자 한다.

## 3. 다단계 신속 코더백터 찾기 알고리즘

### 3.1. 코더백터집단 설계

최적 코더북의 설계는 근본적 매우 풀기 어려운 NP-hard 문제여서 일반적으로 최적에 근사한 코더북을 얻는 방법을 사용한다. 가장 잘 알려진 근사 방식은 LBG 알고리즘이다. 주어진 훈련 백터에 대하여 LBG 알고리즘은 다음과 같이 작동한다.

- 코더북의 $N$개 코더백터를 초기화한다.
- 유클리디안 거리 기준으로 훈련백터들이 가장 가까운 코더백터를 중심으로 하는 집단에 모이도록 한다.
- 새 코더백터를 얻기 위해 각 훈련백터 집단별로 다시 중심값을 계산한다.
- 코더백터에 큰 변화가 없을 때까지 위의 2와 3단계를 반복한다.

LBG 알고리즘에는 여러 가지 형태의 변형 알고리즘들이 있는데, 좋은 초기 코더북 선택, 설계 과정의 계산량 감소, 그리고 보다 최적에 가까운 코더북 설계 등에 대하여 개선을 하고 있다. 하지만 LBG 알고리즘은 지역 최적(local optimal) 코더북을 주지만 전역적 최적(global optimal) 최적 코더북은 주지 못하고 또한 접근속도도 느리지만, 매우 간단하여 널리 사용되는 알고리즘 중의 하나이다.

백터양자화기에서 계산량을 줄이기 위한 방식으로 여기서 제안하는 알고리즘은 전체 찾기과정을 여러 단계로 나누는 것이다. 한 단계에서 다음 단계로 갈수록 점점 범위를 좁혀가면서 최적 코더백터를 찾는 것이다. 제안된 알고리즘을 우리는 MSFS(multi-stage fast search) 알고리즘이라 한다. MSFS 알고리즘은 추가적인 오차 없이 보다 효과적인 방법으로 최적 코더백터를 찾을 수 있게 한다. MSFS 알고리즘은 다단계 찾기 과정을 가질 수 있지만 본 연구에서는 2 단계로 제한하여 연구를 하였다.

먼저 최적 코더북에 설계되었다고 가정한다. MSFS 알고리즘의 목표는 전체 코더북을 찾지 않고 최적 코더백터를 신속하게 찾는 것이다. 이를 위해 먼저 전체 코더북의 코더백터를 새로운 훈련백터로 생각하여 전체 코더북 크기보다 훨씬 작은 새 코더북을 설계한다. 그리고 전체 코더북을 작은 새 코더북을 중심점으로 하는 집단으로 분류를 한다. 이제 MSFS 알고리즘의 최적 코더백터 찾기 과정은 단계별로 진행될 수 있는데 1단계로 작은 코더북에 대하여 입력과 가장 가까운 코더백터를 찾는다. 그러면 찾아진 작은 코더북의 코더백터를 중심하는 전체 코더북의 코더백터 집단에 대하여서만 2단계 전체 코더백터 찾기를 한다. 전체 코더백터 집단의 중심점이 되는 작은 코더백터

$\{g_1, g_2, \cdots, g_M\}$를 얻는 과정은 변형된 LBG 알고리즘을 사용한다. $M$은 $2^{b-1}$ 값을 가진다. 이 방식은 본 논문에서 제안하는 알고리즘이다.

LBG 알고리즘으로 얻어진 $k$차원의 $N$개 코더백터가 있다. 그러면 각 코더백터의 Voronoi 영역, $V_i$에 속하는 훈련백터의 집단에 대하여 이미 알고 있다. Voronoi 영역은 훈련백터들을 코더백터를 중심으로 최적하게 집단화하고 있다. 최적 MSFS 방식의 1단계 과정을 위하여 본 연구에서는 전체 코더백터를 $M$ 코더백터 집단으로 나눈다. 이들 코더백터 집단의 중심점은 다음의 변형 LBG 알고리즘으로 얻는다.

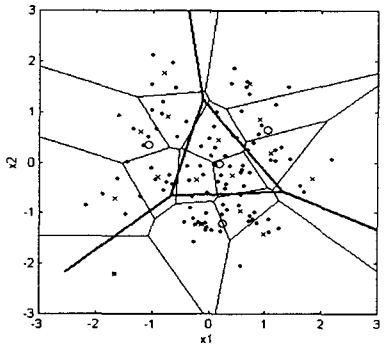
- $M$ 개의 초기 중심점, $\{g_i\}$을 선택한다.
- 유클리디안 거리 기준으로 각 중심점에 가장 가까운 코더백터들은 집단화한다.
- 현재 코더백터 집단에 속하는 코더백터의 $V_i$ 영역에 속하는 모든 훈련백터들을 사용하여 새로운 중심점을 계산한다.
- 각 코더백터 집단에서 중심점의 값의 변화가 아주 적을 때까지 이 알고리즘의 2과 3단계를 반복한다.

위의 변형 LBG 알고리즘에서 3단계는 일반 LBG 알고리즘과 차이가 나는 부분이다. 변형 LBG 알고리즘은 코더백터, $\{c_i\}$ 자체를 훈련백터로 생각하지만, 집단의 새 중심점 계산에서는 코더백터 대신에 최초 훈련백터를 사용한다. MSFS 알고리즘에서는 추가적으로 집단의 중심점을 저장하기 위한 $M$ 메모리 저장공간이 필요하다.

### 3.2 신속 코더백터 찾기 성능

간단한 예를 살펴봄으로써 MSFS 알고리즘을 자세히 설명하고 효율성을 보이고자 한다. 훈련백터는 0 평균(zero mean)과 단위 편차(unit variance)를 가지는 가우스(Gauss) 분포의 랜덤 데이터(random data) 발생으로 얻는다. 먼저 백터양자화기의 코더북이 설계되고 이 코더북의 코더백터를 사용하여 코더백터 집단의 중심점을 변형 LBG 알고리즘으로 얻는다.

그림 1은 100개의 가우스 랜덤 데이터(표시 •), 16개의 코더백터(표시 ×), 코더백터 집단의 중심점(표시 ○), 코더백터의 Voronoi 영역의 경계선(옅은 직선), 그리고 코더백터 집단의 경계선(짙은 직선)을 나타내고 있다. 여기서 MSFS 알고리즘은 16개 코더백터를 2 단계 찾기과정으로 나누어 최적 코더백터를 찾는데 1단계에서 입력백터에 가장 가까운 코더백터 집단의 중심점을 찾고 2단계로 찾아진




<그림 1> Voronoi 영역 및 코더백터 집단의 경계선(100개 가우스 랜덤 데이터--•, 16개 코더백터 --×, 4개 코더집단의 중심점--○)

중심점의 코더백터 집단에 속하는 코더백터 중에서 최적 코더백터를 찾는다.

하지만, 그림 1을 살펴보면 코더백터의 Voronoi 영역 경계선과 코더백터 집단의 경계선이 일치하지 않은 것을 알 수 있다. 이러한 경계선의 불일치로 인하여 최적 코더백터를 구하는 과정에서 추가적인 오차가 있을 수 있게 된다. 이 문제는 코더백터 집단의 경계선에 Voronoi 영역이 걸 쳐져 있는 코더백터를 파악하여 이러한 코더백터들도 해당 코더백터 집단에 포함하여 해결하였다.

이제 각 코더백터의 집단들은 2단계 찾기 과정에서 필요한 코더백터의 인덱스를 가지고 있다. MSFS 알고리즘의 다음과 같이 작동한다.

- 입력백터, $x_i$에 대하여 가장 가까운 중심점, $g_i$를 찾는다.
- 찾아진 $g_i$의 코더백터 집단에 속하는 코더백터 인덱스 중에서 입력백터에 가장 가까운 코드백터를 찾는다.

$g_i$는 자기에게 소속되는 코더백터 인덱스를 가지고 있지만, 이 인덱스는 정수이기 때문에 큰 메모리 저장공간을 필요로 하지 않는다. 표 1은 그림 1에 나타나 있는 각 코더백터 집단에 속하는 코더백터 개수를 보여주고 있다.

표 1은 전체 코더백터가 16개인 백터양자화기에 대한 경우인데 MSFS 알고리즘을 사용하면 최대 9번의 유클리디안 오차 계산을 해야함을 알 수 있다. 1단계에서 4번, 2단계에서 최고 5번의 계산이 필요하여 최대 9번이다. 전체 코더백터 찾기 경우에는 항상 16번의 유클리디안 계산이 필요하다. 표 1과 같은 전체 코더백터의 개수가 작은 경우에는 계산량 감소가 최소 약 44%로 그렇게 크지 않지만 코드북의 크기가 증가할수록 실제적인 계산량 감소는 상당

<표 1> 각 코더백터 집단의 코더백터 개수

| 코더백터 집단번호 | Voronoi 영역 개수 | 경계선 Voronoi 영역 개수 | 집단별 총 개수 |
|---|---|---|---|
| 1 | 4 | 1 | 5 |
| 2 | 5 | 0 | 5 |
| 3 | 1 | 2 | 3 |
| 4 | 4 | 1 | 5 |

히 크게된다. 표2는 코더북의 크기가 256일 때, 코더백터의 차원 2와 5인 경우에 대하여 MSFS 알고리즘을 적용했을 때 알고리즘의 성능을 보여주고 있다. 표 2에서는 256 코더백터를 먼저 16개의 코더백터 집단으로 변형 LBG 알고리즘을 적용하여 분류를 하였으며 또한 코더백터 집단의 경계선에 걸 처져 있는 코더백터들도 해당 집단에 포함하고 있다.

<표 2> 큰 코드북에서 MSFS 알고리즘 계산량 감소.

| 코더백터 집단 번호 | 코더백터 개수 (차원=2) | 코더백터 개수 (차원=5) |
|---|---|---|
| 1 | 21 | 23 |
| 2 | 25 | 28 |
| 3 | 20 | 34 |
| 4 | 29 | 35 |
| 5 | 21 | 30 |
| 6 | 17 | 34 |
| 7 | 25 | 31 |
| 8 | 26 | 33 |
| 9 | 20 | 33 |
| 10 | 23 | 34 |
| 11 | 20 | 37 |
| 12 | 21 | 35 |
| 13 | 23 | 34 |
| 14 | 19 | 41 |
| 15 | 20 | 30 |
| 16 | 22 | 33 |

차원 2 코더백터 양자화 경우에는 최적 코더백터를 찾기 위해서 최대 45번의 유클리디안 오차 계산이 필요함을 보여주고 있다. 16번의 계산이 1단계에서 필요하며, 2 단계에서는 최대 29번이 필요하다. 전체 코더백터 찾기의 계산량 보다는 MSFS 알고리즘은 약 82%의 계산량 감소를 보이고 있다. 차원 5 코더백터의 경우에는 MSFS 알고리즘은 최대 57번의 유클리디안 계산이 필요한데, 표 2의 코더백터 집단 번호 14번의 차원 5 경우를 보면 2 단계 찾기 과정에서 최대 41번의 계산이 필요함을 알 수 있다. 그래서 MSFS 알고리즘은 전체 찾기 알고리즘에 비하여 약 78% 계산량 감소가 있다. 평균적으로 보면 표 2 경우에는 약 80%의 계산량 감소가 있음을 알 수 있다.

MSFS 알고리즘은 최적백터 찾기 과정에 의한 추가적인 오차를 필요로 하지 않는다. 표 2의 경우에는 단지 코더백터 집단의 중심점 저장을 위한 16개의 메모리 공간과 각 코더백터 집단에 소속되는 코더백터들의 인덱스 저장을 위한 약 $2N$ 개의 정수 보관용 메모리 공간이 필요하다. 기존의 최적 코더백터 찾기 알고리즘들은 모든 코더백터 사이의 거리를 계산하여 보관해야 한다든지 혹은 모든 코더백터의 평균값을 계산한 후 저장하고 있어야 하는데, $N \times N$ 혹은 $N$개의 저장 공간을 필요로 한다. 그리고 일부 알고리즘은 추가적인 오차를 백터양자화기에 가져오기도 한다. 하지만 MSFS 알고리즘은 전체적으로 약 $2.5N$ 개 이하의 추가 메모리 공간만으로 추가적인 오차 없이 실질적인 계산량 감소를 백터양자화기에 실현할 수 있는 매우 효과적인 방식이라 할 수 있다.

## 4. 결 론

가우시안 데이터에 대한 MSFS 알고리즘을 백터양자화기에 적용하였다. MSFS 알고리즘의 1 단계에서 사용되는 코더백터 집단의 중심점들은 변형 LBG 알고리즘으로 구하였다. 256 개의 코더북에 MSFS 알고리즘을 적용한 결과를 보면, 2 단계 찾기 과정만을 적용하여도 추가적인 오차 없이 약 80%의 계산량 감소를 보였다. 이것은 실제 시스템에서 상당한 계산량 감소이며, PDE 알고리즘과 같은 다른 알고리즘을 결합하면 더욱 계산량을 줄일 수 있을 것이다.

## 참 고 문 헌

[1] A. Gersho, and R.M. Gray, Vector quantization and data compression, Kluwer, Massachusetts, 1992.

[2] C.D. Bei, and R.M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, IEEE Trans. on Comm., vol. 33, no. 10, pp. 1132-1133, 1985.

[3] J. Ngwa-Ndifor, and T. Ellis, Predictive partial search algorithm for vector quantization, IEE Electronic Letters, vol. 27, pp.1722-1723, 1991.

[4] S.H. Huang, and S.H. Chen, Fast encoding algorithm for VQ-based image coding, IEE, Eletronic Letters, vol. 26, pp. 1618-1619, 1990.

[5] G. Poggi, Fast algorithm for full-search VQ encoding, IEE Electronic Letters, vol. 13, pp. 1141-1142, 1991.

[6] C.Q. Chen, S. H. Koh, and I.Y. Soon, Fast codebook search algorithm for unconstrained vector quantization, IEE Proc.-Vis. Image Signal Process, vol. 145, no. 2, pp. 97-102, 1998.

**우 홍 체**

1980년 2월 : 경북대학교 전자공학과 (학사)
1979년 12월 ~ 1985년 12월 국방과학 연구소 연구원
1988년 12월 Texas A&M 대학교 전기 학과(석사)
1991년 12월 Texas A&M 대학교 전기과(박사)
1992년 3월 ~ 현재 대구대학교 정보통신공학부 부교수