DCT 기반 MPEG-2/H.264 변환을 위한 1/2 화소 보정

(Half-Pixel Correction for MPEG-2/H.264 Transcoding)

권 순 영 † 이 주 경 † 정 기 동 ††
(Soon-young Kwon) (Joo-kyong Lee) (Ki-dong Chung)

요 약 최신 동영상 압축 표준인 H.264는 압축 효율을 높이기 위해 기존의 표준과는 다른 1/2 화소 생성 방법을 사용한다. 그러므로 기존의 동영상 압축표준으로 압축된 비트열을 DCT 상에서 H.264로 트랜스코딩(transcoding)하기 위해서는 추가적인 보정 작업이 필요하다. 본 논문에서는 MPEG-2로 압축된 비트열을 DCT 상에서 H.264로 트랜스코딩 할 때 두 표준 간 1/2 화소 값의 차이를 보정하는 기법을 제안한다. 제안된 1/2 화소 보정 기법에서는 DCT 상태의 참조 프레임을 이용하여 두 표준 간의 차이 값을 구하여 입력으로 들어온 블록의 값에 더하여 보정한다. 픽셀 기반에서 보정하는 기법과 성능을 비교한 결과 제안하는 기법이 화질 면에서 우수하였다.

키워드 : MPEG-2/H.264 트랜스코더, 1/2 화소 보정, 역 움직임 추정

Abstract To improve video quality and coding efficiency, H.264/AVC adopts different half pixel calculating method compared with the previous standards. So, the transcoder requires additional works to transcode the pre-coded video contents with the previous standards to H.264/AVC in DCT domain. In this paper, we propose the first half-pixel correction method for MPEG-2 to H.264 transcoding in DCT domain. In the proposed method, MPEG-2 block is added to the correction block obtained by difference calculation of half-pixel values between two standards using DCT reference frame. Experimental results show that the proposed achieves better quality than pixel based cascaded transcoding method.

Key words : MPEG-2 / H.264 Transcoder, half pixel correction, Inverse Motion Compensation

1. 서 론

최신의 동영상 압축 기법인 H.264/AVC는 ISO(International Organization for Standardization)와 ITU(International Telecommunication Union)에서 2003년 표준으로 승인되었다. 이 표준은 MPEG-2 압축 방식의 화질을 유지하면서 압축률을 50%로 낮추기 위해 가변 블록 움직임 보상, 복수 참조 영상, 1/4 화소 움직임 벡터와 같은 다양한 기법을 추가하였다[1]. 동영상 스트리밍 서비스를 지원하는 응용프로그램에서 성능을 평가한 결과, H.264/AVC Main Profile의 평균 비트율은 MPEG-2에 비하여 63%, MPEG-4 Visual Advanced Simple Profile에 비하여 37% 정도 성능이 더 우수한 것으로 나타났다[2]. 우수한 압축 성능으로 H.264는 다양한 사용자 환경에 사용될 것으로 보이며 기존의 압축 표준을 대체할 것으로 예상된다. 특히, HDTV(High Definition Television), DVD(Digital Versatile Disc)에 적합한 MPEG-2 데이타의 대용량으로 인해 H.264로의 급격한 전환이 이루어질 것으로 전망된다. 따라서 MPEG-2로 압축된 데이타를 H.264로 변환할 가능성이 매우 높으며 효율적인 트랜스코더의 필요성이 대두되고 있다[3].

MPEG-2에서 H.264로의 트랜스코딩에 대한 연구에서는 기존 표준들과의 차이 때문에 픽셀 상에서 트랜스코딩을 수행하였다. 특히 H.264는 MPEG-2와는 다른 1/2 화소 연산 식을 사용한다. 본 논문에서는 앞에서 지적한 1/2 화소 차이점을 보정하여 DCT상에서 트랜스코딩이 가능하도록 한다.

· 본 연구보고서는 정보통신부 정보통신연구진흥원에서 지원하고 있는 정보통신기초연구지원사업의 연구결과입니다.
† 학생회원 : 부산대학교 컴퓨터공학과
ksy2020@melon.cs.pusan.ac.kr
jklee@melon.cs.pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
kdchung@melon.cs.pusan.ac.kr
논문접수 : 2005년 3월 23일
심사완료 : 2005년 8월 24일

제안하는 기법은 DCT 상태의 참조 블록을 이용해서 두 표준간의 1/2 화소 차이 값을 구하고 이를 입력 데이타에 더해준다. 이 과정에서 사용하는 행렬 값은 미리 정해진 상수 값으로 메모리에 저장이 가능하므로 계산량을 줄일 수 있다. 또한 기존에 발표된 DCT 기반 트랜스코딩 기법(예를 들어 프레임률 조정, 비트율 감소, 해상도 조정 등)을 적용 할 수 있는 장점이 있다.

실험에서는 다양한 비디오에 대해서 픽셀 기반 트랜스코더와 화질, 계산량 비교를 수행한다. 실험 결과 픽셀 기반 트랜스코더 보다 더 좋은 화질을 생성하는 것을 확인하였다. 이는 트랜스코딩 과정에서 발생하는 반올림의 오류 값을 줄였기 때문이다. 하지만 역 움직임 추정과정에서 행렬 연산 횟수가 많아서 계산량이 다소 높은 것을 확인하였다. 움직임이 적을 경우 제안하는 기법이 더 낮은 계산량을 나타내지만 움직임이 많을수록 제안하는 기법의 계산량이 더 높아지는 것을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서 MPEG-2와 H.264에서 1/2화소 값을 생성하는 과정을 살펴보고, 제안하는 1/2화소 보정 기법에 대해서 3장에서 알아본다. 제안된 기법의 비교 분석을 4장에서 기술하고, 마지막으로 5장에서 결론을 맺는다.

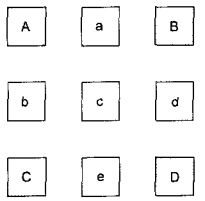
2. 동영상 압축 표준에서의 1/2 화소

다양한 동영상 압축 표준에서는 압축효율을 높이기 위해 1/2 화소의 움직임 벡터를 사용한다. 더 나아가 MPEG-4, H.264의 경우 1/4 움직임 벡터를 사용한다. 1/2, 1/4 화소는 프레임 간 참조의 압축 효율을 높이기 위해 참조 프레임에 있는 실제 화소 값을 이용하여 생성된 가상의 값이다. 1/2 화소의 경우 MPEG-2, H.263, MPEG-4에서는 생성식이 동일한 반면 H.264는 다른 생성 식을 사용한다. 그러므로 MPEG-2에서 H.264로의 트랜스코딩을 할 때, 움직임 벡터를 그대로 사용하려면 1/2 화소 보정을 수행해야 한다.

1/2 화소 정확도를 좀 더 자세히 표현하기 위해 본 논문에서는 1/2 화소를 3가지 경우로 나눈다. 첫 번째는 모션벡터에서 수직성분이 1/2화소 움직임 벡터를 가진 경우이다. 두 번째는 모션벡터에서 수평성분이 1/2화소 움직임 벡터를 가진 경우이다. 세 번째는 모션벡터에서 수직, 수평 성분 모두가 1/2 화소 움직임 벡터를 가진 경우이다. 각각의 예를 들면 MV(1, 1.5), MV (1.5, 1), MV(1.5, 1.5)이다. 이를 본 논문에는 수직 1/2화소, 수평 1/2화소, 수직수평 1/2화소라 정의 한다.

2.1 기존 표준 및 H.264에서 1/2 화소 값 생성

MPEG-2, H.263, MPEG-4 등의 표준에서는 비교적 간단하게 이웃하는 2개 또는 4개 화소의 평균값을 1/2 화소로 정한다[4,5]. 그림 1에서 MPEG-2 방식의 1/2 화소 계산의 예를 보여주고 있다. 여기에서 대문자는 화소를 소문자는 1/2 또는 1/4화소를 나타낸다.



$$a=\frac{(A+B+1)}{2}$$

$$b=\frac{(A+C+1)}{2} \quad (1)$$

$$c=\frac{(A+B+C+D+2)}{4}$$

그림 1 MPEG-2의 1/2 화소 계산

H.264에서는 기존의 표준과는 달리 주위 6개의 화소들을 이용하여 1/2 화소를 계산한다[1,6].

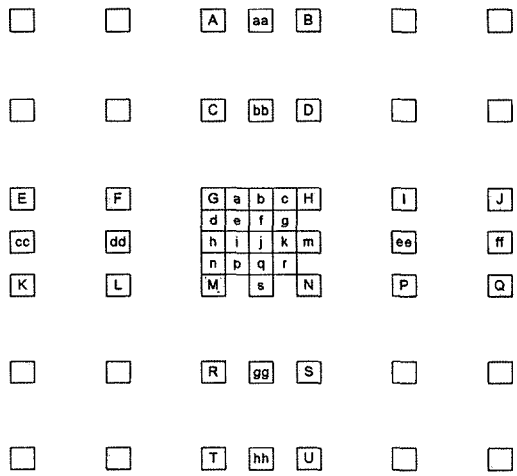


그림 2 H.264 1/2 화소 계산

그림 2는 H.264에서 1/2 화소의 예를 보여주고 있다. 수평 1/2 화소인 b와 수직 1/2 화소 h를 구하기 위해서 아래와 같이 6-taps 필터를 적용한다.

$$b_1=(E-5F+20G+20H-5I+J) \quad (2)$$

$$h_1=(A-5C+20G+20M-5R+T) \quad (3)$$

계산된 값을 0-255 사이의 값이 되도록 아래의 수식을 이용하여 조정한다.

$$b=(b_1+16)\gg 5 \quad (4)$$

$$h=(h_1+16)\gg 5 \quad (5)$$

수직수평 1/2화소인 j의 경우는 아래와 같은 6-taps 필터를 거치게 되며 cc,dd,h_1,m_1,ee,ff들은 b_1과 비슷한 방법으로 얻어진다.

$$j_1=cc-5dd+20h_1+20m_1-5ee+ff \quad (6)$$

$$j=(j_1+512)\gg 10 \quad (7)$$

두 표준간의 1/2화소 계산 방식의 차이 때문에 트랜

스코더에서의 보정이 필요하게 된다.

3. DCT 상에서의 1/2 화소 보정기법

그림 3은 MPEG-2에서 H.264로 변환하기 위한 본 논문에서 제안하는 DCT 기반 트랜스코더 구조를 블록 다이어그램으로 나타낸 것이다. 그림에서 Q_1은 MPEG-2의 양자화, Q_2는 H.264의 양자화 단계를 나타내고 DCT_Conv(DCT Conversion)은 두 표준간의 DCT 변환(Conversion)을 수행한다. DCT_Conv 단계를 거치면 하나의 8×8 실수형 DCT 블록이 4개의 4×4 정수형 DCT 블록으로 변환이 된다. 여기서 만들어진 4×4 정수형 DCT 블록은 H.264로 만들어진 4×4 정수형 DCT 블록과 동일하므로 Q_2 과정과 VLC 과정은 원래 있던 방법을 그대로 사용하여도 무방하다. 좀 더 자세한 내용은 [7,8]을 참조하라. 본 논문의 초점이 DCT 상에서의 1/2 화소의 보정이므로 1/2 화소를 보정하는 CHPD (Compensation for Half-Pixel Difference)블록에 대하여 자세히 살펴보기로 한다. 수식에서 소문자는 픽셀상의 블록을 나타내며 대문자는 DCT 상의 블록임을 나타낸다.

CHPD 블록에서 두 표준 간 1/2 화소 값의 차이를 보정하는 과정을 수행하며 세 부분으로 나눌 수 있다. 첫째, DCT 상태의 이전 프레임을 이용해서 참조 블록을 얻어오는 과정 즉 역 움직임 추정이다. 둘째, 참조 블록에서 두 표준간의 1/2 화소 차이 값을 구한다. 마지막으로 구해진 보정 블록을 입력 데이타에 더해줌으로써 1/2 화소 보정이 이루어진다.

참조 블록을 얻어 오기 위해 Frame Store 버퍼에 저장되어 있는 DCT 상태의 이전 프레임을 이용해서 역 움직임 추정을 수행한다[6,9]. MPEG-2는 8x8 블록 단위로 DCT가 이루어지므로 기존의 연구에서는 MPEG-2의 1/2 화소를 위한 역 움직임 추정은 8×8 블록을 기본단위로 사용하였다[9]. 하지만 본 논문에서 제안하는 방법을 적용하기 위해서 8×8 블록이 아닌 최대 왼쪽 2 픽셀, 오른쪽 3픽셀, 위쪽 2픽셀, 그리고 아래쪽 3픽셀이 더 필요하다. 즉 13×13 블록($\hat{B}$)이 필요하다. 왜냐하면 H.264에서 1/2 화소를 구하기 위해선 자신을 포함한 주위 6개의 픽셀 값이 더 필요하기 때문이다. 수직,수평 1/2 화소를 구하기 위하여 8×8 블록의 경계선에 있는 픽셀들은 추가적으로 2 또는 3개의 픽셀 정보가 더 필요하다. 13×13의 경우는 최대 9개의 8×8 블록과 겹칠 수 있다는 것을 그림 4에서 알 수 있다. 여기서 구한 $\hat{B}$ 블록을 이용해서 보정 블록을 구한다.

그림 4와 같은 경우에 $\hat{b}$을 구하기 위해서는 참조 프레임의 8×8 블록 9개의 정보를 이용을 하여야 하며 블록의 정보를 이용하여 $\hat{b}$을 구하는 과정이 그림 5에 나

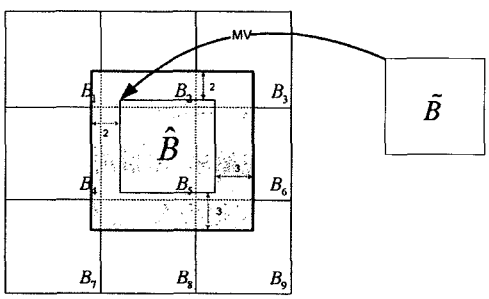


그림 4 역 움직임 추정(입력 블록이 참조 프레임의 9개 블록과 겹친다).

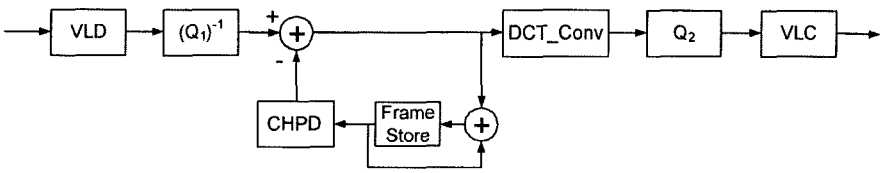


그림 3 MPEG-2에서 H.264로의 DCT기반 트랜스코더 구조

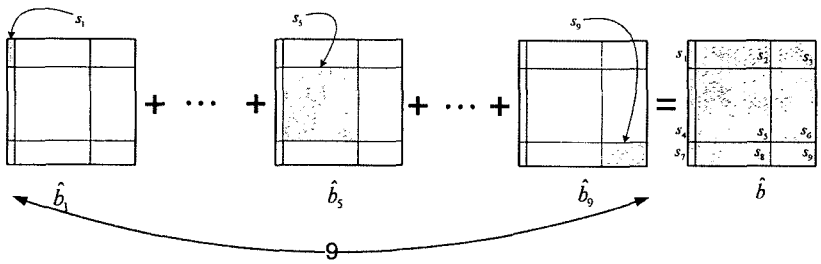


그림 5 참조 프레임에서 $\hat{b}$을 구하기 위한 과정의 예

와 있다[10,11]. 시프트 연산 행렬을 이용해서 겹치는 부분의 정보만을 $\hat{b}$의 해당 위치에 옮겨 놓을 수 있다. 여기서 s_i는 b_i번째 블록과 $\hat{b}$이 겹치는 부분을 나타내며 수식 (8)과 같이 표현된다[12].

$$\hat{b}=\sum_{i=1}^{n} e_l \cdot b_i \cdot e_r \quad n \in S, \quad S=\{2,3,4,6,9\} \tag{8}$$

$$=\underbrace{\begin{pmatrix} b_1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{\hat{b}_1}+\underbrace{\begin{pmatrix} 0 & b_2 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{\hat{b}_2}+\cdots+\underbrace{\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & b_n \end{pmatrix}}_{\hat{b}_n}$$

n은 겹쳐지는 블록의 수를 나타낸다. 1/2 화소의 방향과 움직임 벡터의 크기에 따라서 겹쳐지는 블록의 개수가 달라지므로 경우에 따라 n의 값이 정해진다. b_i는 8x8 크기의 블록이고, e_l, e_r과 $\hat{b}$은 1/2 화소의 방향에 따라 변한다. e_l과 e_r 은 시프트 연산 행렬로 아래와 같이 나타낼 수 있다. 여기서 $I_{l\times l}$은 l 크기의 단위행렬이고 l의 값은 움직임 벡터의 크기와 1/2 화소 방향에 따라서 1에서 13사이의 값을 가진다.

$$e_l=\begin{pmatrix} 0 & I_{l\times l} \\ 0 & 0 \end{pmatrix}_{m'\times n'} \quad or \quad \begin{pmatrix} 0 & 0 \\ I_{l\times l} & 0 \end{pmatrix}_{m'\times n'}$$

$$e_r=\begin{pmatrix} I_{l\times l} & 0 \\ 0 & 0 \end{pmatrix}_{m''\times n''} \quad or \quad \begin{pmatrix} 0 & 0 \\ 0 & I_{l\times l} \end{pmatrix}_{m''\times n''}$$

$$m',n',m'',n'' \in \{8,13\} \tag{9}$$

예를 들어 수직 1/2 화소인 경우에 $\hat{b}$은 13×8의 크기를 가진다. e_l은 13×8 크기의 행렬을 e_r은 8×8크기의 행렬을 가지므로 m'은 13을 n',m''과 n''은 8의 값을 가진다.

DCT는 분배법칙이 성립하므로 수식 (8)은 수식 (10)과 같이 나타낼 수 있다.

$$\hat{B}=\sum_{i=1}^{n} E_L \cdot B_i \cdot E_R \quad n \in S, \quad S=\{2,3,4,6,9\} \tag{10}$$

수식 (10)에서 구해진 $\hat{B}$와 수식 (11)을 이용하여 MPEG-2와 H.264에서의 1/2화소 차이 값을 찾는다.

$$\begin{cases} C=F_L^S \cdot \hat{B} \gg 5 \\ C=\hat{B} \cdot F_R^S \gg 5 \\ C=F_{HL} \cdot \hat{B} \cdot F_{HR} \gg 10 \; -F_{ML} \cdot \hat{B} \cdot F_{MR} \gg 2 \end{cases} \tag{11}$$

1/2 화소 차이 값은 1/2화소가 생기는 방향에 따라 수식 (11)과 같이 달라진다. 첫 번째 수식은 수직 방향, 두 번째 수식은 수평 방향 그리고 세 번째는 수직,수평 1/2화소인 경우이다. f는 고정된 상수 행렬 값이며 $f_{hr}=f_{hl}^T$, $f_{mr}=f_{ml}^T$ 이고, $f_l^s=f_{hl}-16\cdot f_{ml}$, $f_r^s=f_{hr}-16\cdot f_{mr}$ 이다.

$$f_{hl}=\begin{pmatrix} 1 & -5 & 20 & 20 & -5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -5 & 20 & 20 & -5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -5 & 20 & 20 & -5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -5 & 20 & 20 & -5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -5 & 20 & 20 & -5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -5 & 20 & 20 & -5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -5 & 20 & 20 & -5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -5 & 20 & 20 & -5 \end{pmatrix}$$

$$f_{ml}=\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

이렇게 구해진 보정 블록(C)은 입력 블록($\tilde{B}$)에 더해 주면 보정이 완료된 새로운 residual 값을 구할 수 있다.

$$N=\tilde{B}-C \tag{12}$$

수식 (10), (11)에서 E_L, E_R, F_L^S, F_R^S은 고정된 행렬이기 때문에 수식 (13)과 같이 나타낼 수 있으며 계산량을 줄일 수 있다.

$$C=\sum_{i=1}^{n} W_L \cdot B_i \quad n \in S, \quad S=\{2,3,4,6,9\}$$

$$C=\sum_{i=1}^{n} B_i \cdot W_R \quad n \in S, \quad S=\{2,3,4,6,9\}$$

$$W_L=F_L \cdot E_L \gg 5 \tag{13}$$
$$W_R=E_R \cdot F_R \gg 5$$

새롭게 구한 residual 블록(N)은 두 표준간의 transform 차이를 수정하기 위해서 DCT 변환 단계의 입력으로 들어가게 된다.

4. 실험 결과

제안된 기법의 성능을 평가하기 위해 그림 3과 같은 구조의 트랜스코더를 구현하고 그림 6의 픽셀 기반 트랜스코더[4]와 화질 및 계산량을 비교하였다. H.264는 최신의 동영상 압축 표준으로 기존 표준과 H.264간의 DCT 기반 트랜스코딩 기법 및 보정 기법이 전무하므로 픽셀 기반의 트랜스코더와 성능을 비교한다. 동일한 조건에서의 실험을 위해 H.264에서는 Inter 프레임의 움직임 예측 블록 크기를 MPEG-2와 동일한 16×16으로 고정하고 1/2 화소까지 지원한다. 트랜스코더에 사용된

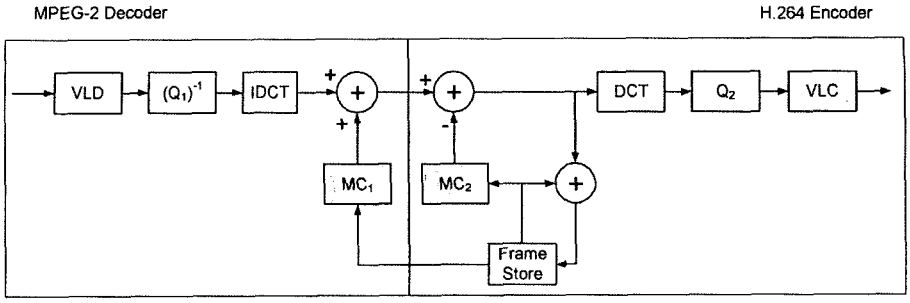


그림 6 비교대상인 픽셀 기반 트랜스코더 구조

표준은 MPEG-2의 TM5[13]와 H.264의 JM8.2[14]이며 트랜스코더 구조에 맞게 수정하였다. 실험에 사용된 동영상은 움직임이 많은 Mobile과 상대적으로 움직임이 적은 Tempete CIF 파일을 사용하였으며 압축 시 사용된 프레임율은 30Hz이다. 또한 실험의 편리를 위해 B 프레임은 사용하지 않았으며 GOP의 크기를 15, I와 P 프레임 간격을 1로 설정하였다.

그림 7은 프레임 변화에 따른 PSNR 변화이다. 각 그래프에서 "Proposed"는 제안된 기법을 사용하는 트랜스코더를 나타내고 "Pixel"는 픽셀 기반 트랜스코더를 나타낸다. 실험 그래프에서 볼 수 있듯이 움직임이 클수록 PSNR 차이가 커지는 것을 볼 수 있다. 그림 7의 평균 PSNR을 수치적으로 나타낸 것이 표 1이며 평균 PSNR이 약 0.62dB 정도 더 높은 것을 알 수 있다.

실험을 통하여 논문에서 제안하는 기법이 더 좋은 화

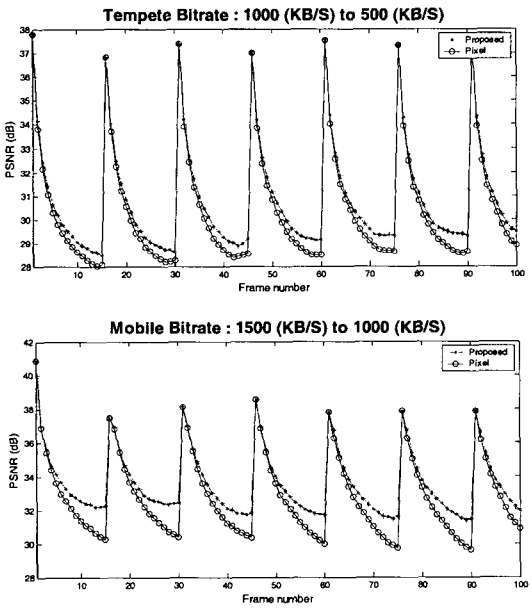


그림 7 프레임 변화에 따른 PSNR 변화

표 1 그림 7에서의 두 트랜스코더 PSNR

	(a) Tempete Bitrate 1000 (KB/S) to 500 (KB/S)	(b) Mobile Bitrate 150 (KB/S) to 1000 (KB/S)
Proposed PSNR	33.61	30.85
Pixel PSNR	32.79	30.43

질을 보이는 것을 확인하였다. 그 이유는 그림 6에서 볼 수 있듯이 픽셀 기반 트랜스코더는 두 번의 MC 과정을 거치게 된다. 복호화를 위해 MC_1에서 MPEG-2 방식의 1/2화소를 계산을 한다. 이 과정에서 반올림 에러 값이 생긴다. 새로운 residual 값을 구하기 위해 MC_2에서 H.264 방식의 1/2 화소를 계산을 하게 되며 이 과정에도 반올림 값 에러가 생긴다. 즉 두 곳에서 반올림 에러 값이 생기게 된다[3]. 하지만 제안 하는 트랜스코더는 행렬 연산으로 두 표준 사이의 차이 값을 바로 구하게 되므로 중간 과정에서 생기는 반올림 에러 값을 한번으로 줄였다. 또한 DCT 상에서 트랜스코딩이 이루어지므로 DCT, IDCT에서 생기는 오류 값 또한 제거 하였다.

그림 8은 MPEG-2의 비트율을 고정하고 H.264로 부호화하기 위한 새로운 비트율을 200~1500까지 조절하여 화질의 변화를 PSNR 값으로 나타내었으며 앞의 실험과 동일한 이유로 제안하는 방법이 더 좋은 화질을 보이는 것을 알 수 있다.

그림 9에서 트랜스코딩 후의 실제 이미지를 볼 수 있다.

두 트랜스코더의 계산량을 비교하기 위해 그림 10에서 트랜스코딩 과정의 시간 비율을 보여주고 있다. 픽셀 기반 트랜스코더 시간을 100으로 두었을 때 동일한 조건으로 제안하는 트랜스코더의 시간을 비율로 나타내었다. 두 트랜스코더에서 VLD, Q_1, Q_2, VLC 과정은 동일한 과정이므로 계산량 과정에서 제외하였다. 표에서 볼 수 있듯이 움직임이 작은 Tempete의 경우는 계산량이 작지만 움직임이 많은 영상에 대해서는 계산량이 많은 것을 알 수 있다. 역 움직임 추정을 위한 수식 (8)에서 많은 행렬 곱을 요구하기 때문이며 이 부분이 대부분의

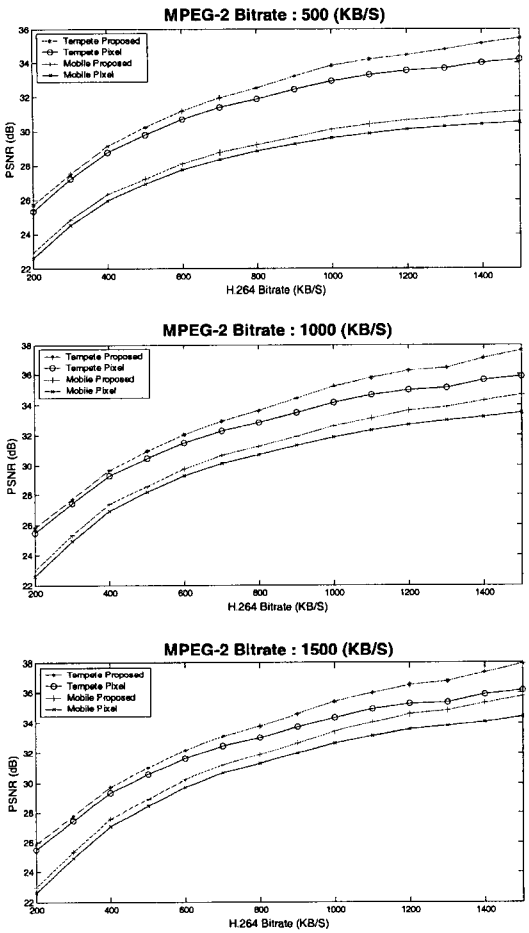


그림 8 MPEG-2 비트율을 고정시킨 후 H.264의 비트율 변화에 따른 PSNR 비교

계산량을 차지하는 것을 알 수 있다.

5. 결론 및 향후 연구

본 논문에서는 MPEG-2에서 H.264로의 트랜스코딩에서 DCT 상에서 1/2 화소 보정 방법을 제안하였다. 제안하는 기법에서는 DCT 상태의 참조 프레임에 특정 행렬 연산을 거쳐 두 표준 사이의 1/2 화소 차이를 구한 뒤 입력 블록에 더하여 보정작업을 수행하였다. 이 과정 에서 사용되는 행렬은 미리 정해지는 상수 값이므로 메모리에 저장을 해 둘 수 있으므로 계산량이 감소한다. 제안하는 기법은 MC 과정에서 생기는 오류 값을 줄임으로써 화질에서 픽셀 기반 트랜스코더보다 나은 성능을 보이며 실제 실험에서도 그와 같은 사실을 확인할 수 있었다. 그러나 움직임이 많은 동영상은 1/2 화소가 많고 그 계산량이 픽셀기반 트랜스코딩보다 높아 전

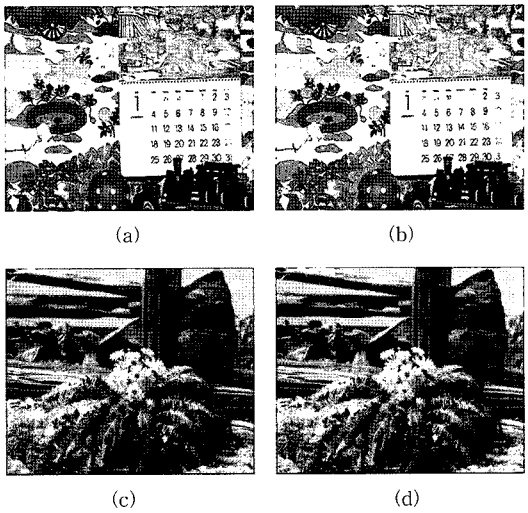


그림 9 실제 이미지 비교. 왼쪽 줄은 제안하는 트랜스코더 이미지, 오른쪽 줄은 픽셀 기반 트랜스코더. : (a)(b) Mobile 영상에 대해서 MPEG-2의 비트율이 1000(KB/S), H.264의 비트율이 500(KB/S)인 경우, (a)(b) Tempete 영상에 대해서 MPEG-2의 비트율이 1500(KB/S), H.264의 비트율이 1000(KB/S)인 경우

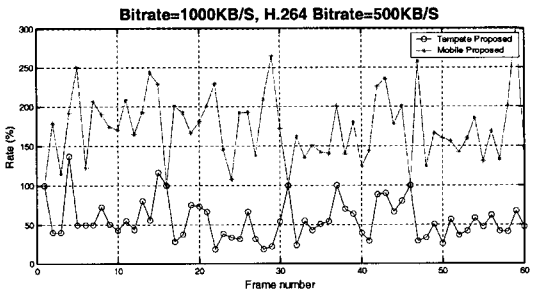


그림 10 두 트랜스코더의 트랜스코딩 시간 비교

체 계산량이 증가함을 알 수 있었다. 움직임이 많은 비디오에 대한 계산 복잡도를 줄이는 향후 연구가 필요하다.

참 고 문 헌

[1] T.Wiegand, G.J.Sullivan, G. Bjontegaard, and A.Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., Vol.13, pp.560-576, July 2003.

[2] J. Ostermann, J.Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video Coding with H.264/AVC: Tools, Performance, and Complexity," IEEE Circuits Syst. Magazine, Vol.4, No.1, pp.7-28, Apr. 2004.

[3] Hari Kalva. "Issues in H.264/MPEG-2 Video

Transcoding," Consumer Communications and Networking Conference, 2004. CCNC 2004. First IEEE, 5-8 Jan. 2004.
[4] A. Vetro, C. Christopoulos, and H. Sun, "Video Transcoding Architectures and Techniques: An Overview," IEEE Signal Processing Magazin, Vol.20, No.2, pp.18-29, March 2003.
[5] ISO/IEC 13818-2:1995(E) pp.83-100.
[6] Iain E.G. Richardson, "H.264 and MPEG-4 Video Compression," WILEY, 2003.
[7] 강진미, "MPEG-2에서 H.264/AVC로의 변환을 위한 DCT 기반 트랜스코더 구조", 공학석사 학위논문, 2005년 02월.
[8] Joo-Kyong LEE, Ki-Dong CHUNG. "Quantization/DCT Conversion Scheme for DCT-domain MPEG-2 to H.264/AVC Transcoding," IEICE Trans. Commun., Vol.E88-B, No.7 pp.2856-2863, JULY 2005.
[9] T. Shanableh and M. Ghanbari, "Hybrid DCT/pixel domain architecture for heterogeneous video transcoding," Signal processing: Image Communication, Vol.18, pp.601-620, 2003.
[10] G.Cao, Z.Lei, J.Li, N.D.Georganas, Z.Zhu. "A Novel DCT Domain Transcoder for Transcoding Video Streams with Half-pixel Motion Vectors," Real-Time Imaging (Elsevier Science) Vol.10. Issue 5, Oc.2004, pp.331-337.
[11] Haiyan Shu and Lap-Pui, "An Efficient Arbitrary Downsizing Algorithm for Video Transcoding," IEEE Trans. Circuits Syst. Video Technol., Vol. 14, No.6, pp.887-891.
[12] S. F. Chang and D. G. Messerschmitt, "Manipulation and composing of MC-DCT compressed video," IEEE JNL. Select. Areas Commun,. Vol. 13, pp.1-11, Jan. 1995.
[13] http://diml.yonsei.ac.kr/~wizard97/mpeg2/mpeg2v12.zip.
[14] http://bs.hhi.de/~suehring/tml/download/old_jm/jm82.zip.

권 순 영
2004년 부산대학교 전자전기정보컴퓨터공학부 졸업(학사). 2004년~현재 부산대학교 대학원 컴퓨터공학과 석사과정. 관심분야는 멀티미디어 데이타 압축, 임베디드 시스템

이 주 경
1996년 부산대학교 전자계산학과 졸업(학사). 1998년 부산대학교 전자계산학과 졸업(석사). 1998년~2001년 한국전력공사 근무. 2001년~현재 부산대학교 대학원 컴퓨터공학과 박사과정. 관심분야는 멀티미디어 데이타 압축, 오류제어, 임베디드 시스템

정 기 동
1973년 서울대학교 졸업(학사). 1975년 서울대학교 대학원 졸업(석사). 1986년 서울대학교 대학원 계산통계학과 졸업(이학박사). 1990년~1991년 MIT, South Carolina 대학 교환 교수. 1995년~1997년 부산대학교 전자계산소 소장. 1978년~현재 부산대학교 전자계산학과 교수. 1997년~현재 부산대학교 대학원 멀티미디어 협동과정학과 교수. 관심분야는 병렬처리, 멀티미디어, 임베디드시스템