論文 88-25-2-14

이동벡터 추정을 위한 고속 Block Matching Algorithm에 관한 연구

(A Study on Fast Block Matching Algorithms for the Motion Vector Estimation)

李 仁 弘*, 朴 來 弘**

(In Hong Lee and Rae-Hong Park)

要 約

본 논문에서는 움직임이 있는 화상에서 물체의 이동벡터를 효과적으로 구하는 알고리즘을 제안하였다. MCC(motion compensated coding)에서 이동을 검출하는 방법으로는 pel(pixel element) recursive 방법과 block matching 방법이 있는데, 여기서는 후자에 기초하여 계산량을 줄이는 알고리즘을 제안한다.

계산시간을 줄이기 위해 세가지 방법이 제안되었으며 이로 인해 기존의 방법보다 빠르게 이동벡터를 구할 수 있다. 특히 Koga등이 제안한 three step search 방식에 비해 대략 3～4배 정도 빠르다.

Abstract

In this paper effective block matching algorithms are proposed to find the motion vector. There are two approaches to the estimation of the motion vector in MCC (motion compensated coding), i.e. pel (pixel element) recursive algorithm and block matching algorithm. The search algorithm in this paper is based on the block matching method. The advantage of this algorithm is the reduction of the computation time.

In order to reduce the computation time, three mathods are proposed in this paper. These new algorithms are faster than other methods. Compared with the three step algorithm by Koga et al., the average ratio of the computational savings obtained from the proposed algorithms is about 3-4.

I. 서 론

영상 신호를 디지탈 데이타로 전송하는 경우에 있어서 가장 커다란 문제는 전송 대역폭의 증가이며, 이에 따라 전송 대역폭을 줄이기 위한 여러가지 데이타 압축(data compression) 방식이 연구되어 왔다.[1] 데이타 압축 방식으로는 크게 구성 형태에 따라 intraframe coding과 interframe coding으로 나눌 수 있다. intraframe coding은 단일 영상에 존재하는 중복성(redundancy)을 이용하는 방식이며, 반면에 interframe coding은 연속하는 영상에서 그 영상들 사이에 존재하는 중복성을 이용하는 coding 방식이다.[2]

Interframe coding에는 CRC(conditional replenishment coding)[3,4]와 MCC(motion compensated

*準會員, 金星電氣技術研究所
(R & D Lab., GoldStar Elec. Co., Ltd.)
**正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogang Univ.)
接受日字 : 1986年 10月 8日

coding)[5]가 대표적인 것인데 이중 MCC의 데이타 감축성능이 더욱 우수하다.[6,7] CRC는 연속화상에서 움직이는 부분만을 추출하여 위치정보와 변한 양을 전송하는 방식인 반면 MCC는 이동벡터를 전송하는 방식이다. 따라서 MCC에서 가장 중요한 문제는 이동 벡터를 검출하는 방법이다. 이동 벡터를 검출하는 방법은 PRA(pel recursive algorithm)[8]와 BMA(block matching algorithm)[8-13]이 있는데, 전자는 화소(pel)단위로 이동벡터를 추출하여 정확한 값을 얻을 수 있는반면, 계산량이 많고 복잡하다. BMA는 block 단위로 이동벡터를 추출하는 만큼 정확도는 떨어지지만 계산량이 많지 않아 실시간 처리에 유리하므로 여기서는 BMA를 고려한다.

본 논문에서는 두 화면의 연속 영상을 데이타 베이스로 하여, 제안한 방법을 중심으로 기존의 여러가지 다른 방법과 비교분석 하였다. Ⅱ장에서는 여태까지 발표되었던 대표적인 이동벡터 측정 방법을 알아보고, Ⅲ장에서는 개선된 알고리즘을 살펴보면 Ⅳ장에서는 여러 방법들 사이의 performance를 비교 분석한다. 각 방식의 성능비교에는 영상처리에 많이 쓰이는 NSNR (normalized signal to noise ratio)와 계산시간(computation time)을 사용하였다. 또 각 방식을 사용하여 운동보상한 결과 예측된 영상들의 예측오차(prediction error)들을 비교하였다. 여기서 제안한 방법은 최대 이동 가능거리가 ±6 화소라고 가정할 경우에, 대략 한 부영상(subblock) 당 평균 6-7번의 계산으로 이동벡터를 구할 수가 있음을 보여준다. Ⅴ장에서는 전체적인 검토 및 결론이 맺어진다.

Ⅱ. 기존의 Block Matching Algorithms

본 장에서는 먼저 BMA의 이론적 배경을 설명하고, 지금까지 제안된 계산량을 줄이는 알고리즘[9-13]에 대해 알아본다.

1. BMA란?

연속하는 화상(frame)들 사이의 이동량을 검출함에 있어 시간축으로 인접한 화상들 내에서 부영상사이의 상관계수(correlation)를 비교하여 최대치를 이동 보상 위치로 이용하는 방법을 BMA라 한다.[8] BMA의 처리과정(procedure)은 다음과 같다.

우선 화상을 고정된 크기의 부영상으로 나눈다. 이때 이전 화상내의 부영상과의 상관계수값이 최대가 되는 위치(즉, error가 제일 작게 일어나는 위치)를 구하기 위해 다음과 같은 함수D(·)를 정의한다.

$$D(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N} G(U(m,n)-U_r(m+i,n+j)),\quad -p\le i,j\le p \qquad (1)$$

여기서,

G(·) : error power를 구하는 비선형함수 (nonlinear function)

U : 현 화상내에서 M×N 크기의 부영상으로 구성된 화상

U_r : 이전 화상내에서 (M+2p)×(N+2p) 크기의 search area(SR)

p : 최대 이동가능 거리

이다. 이때 이동벡터는 D(i, j)를 최소로 하는 (i, j)로 주어진다.

만약 최대 이동가능 거리를 ±6 화소라하면 한 부영상에 필요한 계산수가 (2×6+1)×(2×6+1)=169가 되며 전체 화상에 대해 필요한 계산량은 엄청나게 많아져서 실시간 처리가 어렵게 된다. 따라서, 다음과 같은 가정하에 search step수를 줄이게 된다.[9]

즉, D0(q, 1)=min{D(i, j)}라 하고 다음을 정의하자.

D1(|m|, |n|) = D(i, j) − D0(q, 1), m≥0, n≥0 (2a)

D2(|m|, |n|) = D(i, j) − D0(q, 1), m≥0, n≤0 (2b)

D3(|m|, |n|) = D(i, j) − D0(q, 1), m≤0, n≤0 (2c)

D4(|m|, |n|) = D(i, j) − D0(q, 1), m≤0, n≥0 (2d)

여기서, (q, 1)은 D(·)를 최소로 하는 위치고, (|m|, |n|)은 임의의 위치와 최적 위치와의 거리를 나타낸다.

그러면 다음과 같이 가정할 수 있다.

Dk(|m|, |n|)<Dk(|m′|, |n′|), k=1, 2, 3, 4

if |m|<|m′| and |n|≤|n′| or |m|≤|m′| and |n|<|n′| (3)

이것은 D(·)값이 최적위치(optimal position)에서 멀어질수록 증가함을 보여준다. 이와같은 가정하에 계산량을 줄일 수 있는 효과적인 BMA를 제안할 수 있다.

2. 2-D Logarithmic 방법

이 방법은 Jain이 제안한[9] 것인데 cost function으로 식(4)로 정의된 MSE(minimum mean square error)를 이용한다.

$$MSE(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}(S_{t,k}(m,n)-S_{t-1,k}(m+i,n+j))^2,\quad -p\le i,j\le p \qquad (4)$$

여기서,

$S_{t,k}$: t번째 화상에서 k번째 부영상

$S_{t-1,k}$: t−1번째 화상에서 k번째 SR(search area)

이다.

첫 단계는 SR의 중심을 포함하여 5개의 위치에 대한 MSE 값을 계산한다. 이중 가장 작은 MSE 값을 갖는 위치를 중심으로 하여 다시 순환적으로 반

복한다. 이 과정은 최소 MSE값을 갖는 위치가 연속적으로 동일할 때까지 계속되며, 마지막 단계에서는 9개의 위치에 대해 MSE 값을 비교하여 가장 작은 MSE 값을 갖는 위치를 찾는다. 이 위치가 바로 이동벡터이다(그림1(a)참조).

DMD(direction of the minimum distortion) 방식을 약간 변형시켜 search point의 수를 줄이고 threshold 개념을 사용한 modified DMD 방식도[13] 계산량을 줄이는데 효과적이다.

3. Menu Vector 이용법

이 방법은 Ninomiya와 Ohtsuka가 제안한[10] 것으로 미리 menu vector의 집합을 만든 다음 그 menu vector에 대해 search함으로써 이동벡터를 찾는다. Menu vector의 집합Y는 다음과 같이 정의한다.

$$Y_i = \{(y_1, y_2) \mid y_1, y_2 : 0, \pm 2^{n-i}, \pm 2^{n-i+1}\} \quad (5)$$

여기서 (y_1, y_2)는 이동벡터를 나타내고, i는 i번째 search stage를 나타낸다. 그리고 n은 전체 stage 의 수를 나타낸다.

그림 1(b)는 menu vector를 이용하여 이동량을 구하는 방법을 보여주는데, 점(·)으로 표시한 위치가 menu가 된다. 여기 표시한 25개의 menu는 전형적인 menu들의 위치를 나타낸다. 여러 menu들 중에서 가장 오차가 적은 menu를 선택하여 이 위치를 이동벡터로 취한다. 또 이 방법은 temporal recursion을 도입하여 search step을 단 한 번으로 줄일 수 있다.

4. Three Step 알고리즘

이 방법은 Koga가 제안한[11] 것으로 (6)식과 같이 정의된 MAD(mean of the absolute frame difference)를 cost function으로 이용한다.

$$MAD(i,j) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N} \mid S_{t,k}(m,n) - S_{t-1,k}(m+i, n+j) \mid, \quad -p \le i, j \le p \quad (6)$$

만약 최대 이동가능 거리를 ±6 화소라고 가정한다면 세 단계를 거쳐서 이동벡터를 구하게 된다.[11] 먼저 기준점을 중심으로 좌, 우, 상, 하, 사선 방향으로 3화소씩 떨어진 위치에 대해 MAD를 구한다. 이때 가장 최소의 MAD 값을 나타내는 위치(i−3, j+3)를 구하여 save시킨다. 그 다음에는 이 위치를 기준으로 하여 좌, 우, 상, 하, 사선 방향으로 2화소씩 떨어진 위치에 대해 MAD를 구하고, 가장 최소의 MAD 값을 나타내는 위치(i−5, j+3)를 구한다. 마지막으로 이 위치를 중심으로 좌, 우, 상, 하, 사선 방향으로 1화소씩 떨어진 위치에 대해 MAD를 구하고, 가장 최소의 MAD값을 나타내는 위치(i−5, j+4)을 구한다. (그림1(c)참조) 이 위치(i−5, j+4)가 부영상의 이동벡터가 된다.

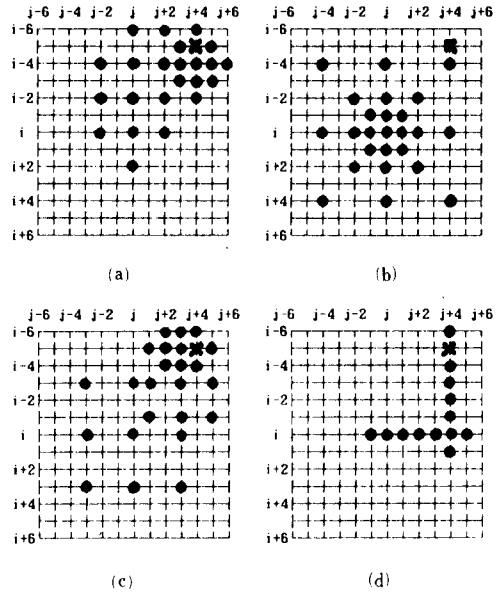

그림 1. 기존의 BMA를 이용한 search 알고리즘,
*: 찾고자 하는 위치
(a) DMD방식 : (i−2, j)→(i−4, j)→(i−4, j+2)→(i−4, j+4)→(i−5, j+4)
(b) menu vector방식 : (i−4, j+4)
(c) three step방식 : (i−3, j+3)→(i−5, j+3)→(i−5, j+4)
(d) OTS 방식 : (i, j+1)→…→(i, j+5)→(i−1, j+4)→…→(i−5, j+4)

Fig. 1. Conventional BMA search algorithms.
*: searching point.
(a) DMD search method : (i−2, j)→(i−4, j)(i−4, j+2)→(i−4, j+4)→(i−5, j+4)
(b) menu vector method : (i−4, j+4)
(c) three step method : (i−3, j+3)→(i−5, j+3)→(i−5, j+4)
(d) OTS method : (i, j+1)→…→(i, j+5)→(i−1, j+4)→…→(i−5, j+4).

5. One at a time search(OTS) 알고리즘

이 방법은 2-D search를 1-D search로 간단화 한 것이다. 즉, SR의 두 축중 하나를 고정시켜가며 optimal value를 찾아간다.[12] 그림 1 (d)에 search과정이 잘 나타나 있다. 1-D로 search하기 때문에 알고리즘이 간단하여 계산량을 줄일 수 있는 반면에 line search가 정확해야만 optimal value를 찾을 수 있다는 단점이 있다.

Ⅲ. 개선된 Fast BMA

여기서는 세가지 방법을 제안하여 계산량을 많이 절

약하였다. 즉, 최대 이동가능 거리가 ±6 화소라고 가정할 경우, 최대 5번+5번+5번=15번의 계산을 하여 error면에서 약간의 희생을 치르고 계산시간의 향상을 꾀하였다. 또 화상을 움직이는 부분과 정지된 부분으로 나누어 위의 계산을 단지 움직이는 부분에만 적용함으로써 불필요한 계산을 줄였다. 그리고 이렇게 구한 이동벡터의 성질을 살펴본 결과, 이웃한 부영상사이의 이동벡터들은 유사한 값을 갖는다는 사실을 알 수 있었다. 이러한 이동벡터사이의 유사성을 이용함으로써 보다 효과적인 이동검출이 가능하게 되었다.

1. 부영상내에서 고속 처리방법

A. Searce point의 수를 줄이는 방법

여기서 사용되는 matching criterion은 Koga 방식과 마찬가지로 식(6)으로 주어진다. 이 방법은 타 방법과 마찬가지로 최적위치에서 멀어질수록 오차가 증가한다는 가정에 입각한 것이다. 오차함수로는 MAD를 이용하고 Koga방식처럼 세 단계로 나누어서 오차가 줄어드는 위치를 추적함으로써 최적위치에 접근해 간다.

다음은 이 방법을 세단계로 나누어 이동벡터를 구하는 과정을 보여준다. Koga 방법은 각 단계마다 9개 위치에 대해 MAD값을 구해 search하지만 제안한 방법은 각 단계마다 5개 위치에 대해서만 search한다.

Step 1 : 여기서는 기준점을 중심으로 사선 방향으로 3화소씩 떨어진 4개의 위치에 대한 부영상과 현 부영상과의 MAD를 비교하여 이 값이 최소가 되는 위치를 찾는다. 이것은 Koga방식의 경우 9개 위치에 대해 search함으로 생기는 계산의 중복을 피한다.

Step 2 : 기준점을 앞에서 구한 위치로 하고 이를 중심으로 좌, 우, 상, 하로 2화소씩 떨어진 4개의 위치에 대한 MAD값을 비교하여 최소가 되는 위치를 찾는다. 그리고 이 점을 중심으로 하여 한 점을 더 search한다. 이 새로운 점의 선택은 이미 구한 점들의 MAD 값을 비교하여 찾는다. 만약 그림 2(a)에서처럼 (i−5, j+3)이 (i−5, j+3), (i−3, j+1), (i−3, j+3), (i−3, j+5), (i−1, j+3)중에서 MAD 값이 가장 작은 위치로 일단 찾아졌다고 하자. 그러면 이 점과 (i−3, j+1), (i−3, j+5)점에서의 MAD값을 이용하여 새로운 점을 찾게 된다. 즉, D1= | MAD(i−5, j+3) −MAD(i−3, j+1) | 과 D2= | MAD(i−5, j+3) −MAD(i−3, j+5) | 를 구하여 D1>D2라면 (i−5, j+5)가 선택되고, D2>D1이라면 (i−5, j+1)이 선택된다. 그림 2(a)에서는 D1>D2이므로 (i−5, j+5) 가 선택되고 이 위치에서 MAD값을 구하여 다시 MAD (i−5, j+3)와 MAD(i−5, j+5)의 값을 비교하여 최종적인 위치가 결정된다.

Step 3 : Step2와 같으며 단지 1화소 거리 떨어진 위치에 대해 search 알고리즘을 적용한 것만 다르다.

이 알고리즘을 쓰면 13−15번의 계산으로 이동벡터를 구할 수 있다. 이것은 Koga방식[11]과 비교했을때 약 40%의 계산량 감축을 뜻한다. 또 modified DMD 방식[13]에서의 search과정(threshold 개념은 제외함)과 비교했을 때 약 20% 정도의 계산량 감축을 보여준다.

B. 이동벡터 사이의 유사성을 이용하는 방법

이 방법은 인접한 부영상의 이동치가 서로 비슷하다는 점을 이용한 것이다. 즉, 현재 부영상의 이동치를 $\vec{X}_k$라 하고 이전 부영상의 이동치를 $\vec{X}_{k-1}$이라 하면, 다음과 같이 쓸 수 있다.

$$\vec{X}_k = \begin{cases} \vec{X}_{k-1}, \ \text{if} \sum_{m=1}^{M}\sum_{n=1}^{N} | S_{t,k}(m,n) - S_{t-1,k}((m,n) + \vec{X}_{k-1}) | < \text{THSD3} & (7a) \\ \vec{X}_{k-1} + \vec{y}, \ \vec{y} \in Y(\cdot), \ \text{if} \sum_{m=1}^{M}\sum_{n=1}^{N} | S_{t,k}(m,n) - S_{t-1,k}((m,n) + \vec{X}_{k-1} + \vec{y}) | < \text{THSD3} & (7b) \\ \vec{X}_k, \ \text{otherwise} \end{cases}$$

여기서, $((m,n)+\vec{X}_{k-1})$: 원래 (m, n)의 위치에 $\vec{X}_{k-1}$

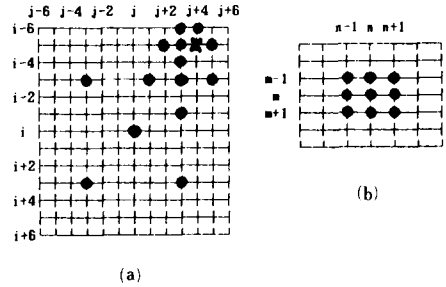

그림 2. 제안한 BMA 방식
(a) 계산해야 할 위치를 줄이는 방법 : (i−3, j+3)→(i−5, j+3)→(i−5, j+4)
(b) 이전의 이동벡터로 부터 현재의 이동벡터를 예측하는 방법
(m, n) : 이전 부영상의 이동벡터,
(m, n)+Y(·) : 예측된 이동벡터

Fig. 2. Proposed new BMA methods.
(a) new BMA search procedure : (j+3)→(i−5, j+3)→(i−5, j+4).
(b) prediction method using the previous motion vector. (m, n) : motion vector in the previous subblock, (m, n)+Y(·) : predicted motion vector.

을 보상하여 얻은 새로운 위치

$Y(\cdot)$: {(−1, −1), (−1, 0), (−1, 1), (0, −1), (0, 1), (1, −1), (1, 0), (1, 1)}

이다.

이전에 구한 이동량을 보상하여 구한 MAD 값이 어떤 기준치(THSD3) 이하면, 현 부영상의 이동량은 이전 부영상의 이동량과 같은 값을 갖는다. 만약 이 기준치를 초과하면 식(7b)에 있는 것처럼 새 벡터 $\vec{y}$를 첨가하여 이동량을 보상한 다음 기준치와 비교한다. 즉, 그림 2(b)를 보면 $\vec{X}_{k-1}$로부터 얻은 기준점을(m, n) 이라할 때 $Y(\cdot)$는 그 주위의 8개 위치가 된다. $\vec{y}$와 $\vec{X}_{k-1}$을 동시에 보상하여 얻은 MAD값이 기준치 이하이면 현 부영상의 이동값은 $\vec{X}_{k-1}+\vec{y}$가 되며, 그렇지 않은 경우에는 Ⅲ장 1절 A에서 제안한 search 알고리즘을 적용하여 이동벡터를 구한다.

2. Frame간의 성질을 이용한 방법

이전의 BMA는 모든 영역에 대해 search 알고리즘을 적용하여 비효율적이었지만 여기서는 이동부영상과 정지부영상으로 나누어 search 알고리즘을 단지 이동부영상에만 적용하여 계산상의 효율을 꾀하였다. 이와 비슷한 방법은 동물체 추적에 이용되었으나,[14] 여기서는 직접 BMA에 적용한 것이다. 이동화소(moving pel)와 이동부영상의 구분은 다음과 같다.

우선 화상내의 모든 화소는 다음과 같은 기준하에 이동화소와 정지화소(stationary pel)로 구분된다.

$$\frac{|S_{t,k}(m,n)-S_{t-1,k}(m,n)|}{S_{t-1,k}(m,n)}\times 100\,(\%) > \mathrm{THSD1} \qquad (8)$$

여기서, THSD1는 이동화소를 결정하는 기준치로 퍼센트(%)로 주어진다. 즉, 이전화상(previous frame)과 현화상(present frame)에서 동일위치의 점의 밝기가 THSD1 퍼센트 이상 변하면 이동화소로 결정되고 그 이하면 정지화소로 결정된다. 또 이동부영상은 그 부영상내의 전체화소 갯수중 이동화소의 갯수가 미리 정한 기준치(THDS2) 이상인 것으로 한다. 이렇게 하여 이동 부영상으로 정의된 영역만 search 알고리즘을 적용하여 이동벡터를 구하며, 정지 부영상으로 정의된 영역은 이동벡터에 0을 부여한다.

Ⅳ장에서 이동화소와 이동부영상을 정하는 기준치(THSD1, THSD2)를 변수로 하여 NSNR(dB)와 계산시간사이의 관계를 그래프로 나타내었다. 이로부터 적절한 THSD1 및 THSD2 값을 얻을 수 있다.

3. 전체적인 알고리즘

다음은 앞에서 제안한 세가지 방법을 합하여 얻은 알고리즘이다.

Step 1 : 입력화상을 받아 들인다.

Step 2 : 입력화상을 M×N 크기의 부영상으로 나눈다.

Step 3 : 입력화상간의 difference picture를 구한다.

Step 4 : 만약 difference picture의 subblock error가 미리 정한 값(THSD2)보다 작으면 이동치는 '0'으로 두고 step10으로 간다. 그렇지 않으면 step 5로 간다.

Step 5 : 임의의 k번째 부영상에 대한 이동치는 이전의 부영상에서 구한 이동치 값을 그대로 사용하여 만약 error가 미리 정한 값(THSD3)보다 작으면 $\vec{X}_k=\vec{X}_{k-1}$이 되어 Step10으로 가고, 그렇지 않을 경우 Step 6로 간다.

Step 6 : $\vec{y}\in Y(\cdot)$인 $\vec{y}$를 첨가하여 새로운 이동치($\vec{X}_{k-1}+\vec{y}$)를 예측한다. 다음에 Step 5의 과정을 반복하여 조건을 만족하면 $\vec{X}_k=\vec{X}_{k-1}+\vec{y}$가되어 Step10으로 가고, 그렇지 않으면 Step 7로 간다.

Step 7 : 기준점을 기준으로 3화소씩 떨어진 위치중 4군데 사선 방향의 점에 대해 MAD값을 구하여 최소의 MAD값을 갖는 위치를 save시킨다.

Step 8 : 위에서 구한 점을 기준으로 2화소 떨어진 상, 하, 좌, 우 점 중 최소 MAD 값을 갖는 (i,j)를 구한다. 이 점을 중심으로 다른 한 점을 선택하여 이 점에서의 MAD를 구하고, (i, j)에서의 MAD와 비교하여 둘 중 더 작은 MAD 값을 갖는 위치를 save시킨다.

Step 9 : 이 점을 기준으로 1화소씩 떨어진 상, 하, 좌, 우 4군데 점에 대하여 Step8에서와 같은 작업을 한다.

Step 10 : 마지막 부영상이라면 끝내고, 그렇지 않으면 Step3 부터 반복한다.

이렇게 하여 구한 부영상의 이동벡터는 한 block에 대해 대략 6−7번의 search 계산을 필요로 한다.

Ⅳ. 실험 및 결과

본 실험에서는 입력으로 크기가 256×256인 연속화면을 이용하였다. 입력 1은 움직임이 빠른 연속화면이고, 입력 2는 움직임이 느린 연속화면이다. 즉, significant pixel을[11] 연속화면에서 gray level 변화가 일정값(4) 이상인 것으로 정의한다면 입력 1은 20%, 입력 2는 12% 정도의 significant pixel을 함유한다. Simulation시 물체의 최대 이동가능 거리는 ±6 화소로 제한하였다. 또 performance비교는 계산시간(computation time), NSNR등을 이용하였으며, 각 방

식에 의해 예측된 영상을 원화상과 비교해본다. NSNR의 정의는 아래와 같다.

$$NSNR = -10 \times \log \frac{E\{(Si-Ti)^2\}}{E\{Si^2\}} \quad (9)$$

여기서 Si는 원화상의 gray level이고, Ti는 운동보상 결과 얻은 영상의 gray level이다. 그리고 Ti=Si를 예측가능화소(predictable pel)라 정의하여 각 방법들이 어느정도 정확하게 이동벡터를 구했는지 비교하였다.

그림 3과 그림 4는 부영상의 크기를 4×4로 하고 Ⅲ장에서 언급한 threshold값들을 변화시켜 가며 계산시간과 운동보상 결과 예측된 영상의 NSNR 값을 살펴본 것이다. 우선 그림 3은 Frame간의 성질을 이용한 방법에서 제시한 이동화소를 결정하는 THSD1 을 변수로 하고 횡축을 이동부영상을 결정하는 THSD 2로 하였다. 이때 THSD1은 이전화소의 밝기에 비해 몇 퍼센트 변한 화소를 이동화소로 삼는가를 결정하는 기준치이며, THSD2는 부영상내의 전체 화소 갯수중 이동화소의 갯수를 나타낸다. 즉, THSD 1을 3~25%까지 변화시키고 THSD2를 2 - 6으로 변화시킬 때 계산시간과 NSNR 값의 변화를 살펴본 것이다. 계산시간과 NSNR 값을 모두 고려한 결과 THSD1은 12.5%, THSD2는 4정도가 적절하다. 그림4는 인접한 이동벡터 사이의 유사성을 이용한 제안방법에서 제시한 THSD3(허용할 수 있는 예측오차의 크기)와 계산시간 및 NSNR값과의 관계를 보여준다. 여기서 THSD3는 18정도가 적절함을 알 수 있다. 또 움직임이 많은 화상(입력 1)에 비해 움직임이 적은 화상(입력 2)이 계산시간이나 NSNR면에서 더 좋음을 알 수 있다.

그림 5와 그림 6은 부영상의 크기가 4×4일 경우에 운동보상 결과 생기는 예측오차(prediction error)를 영상으로 보여주는데, 각각 입력1, 2에 대한 실험결과이다. 여기서 흰 부분이 운동보상 결과 생긴 예측오차를 나타낸다. 제안한 방법을 사용하여 운동보상 결과 얻은 예측된 영상을 살펴보면 타 방법들과 마찬가지로 움직임이 적은 영상보다 많은 영상에서 예측오차가 더 많이 생김을 알 수 있다.

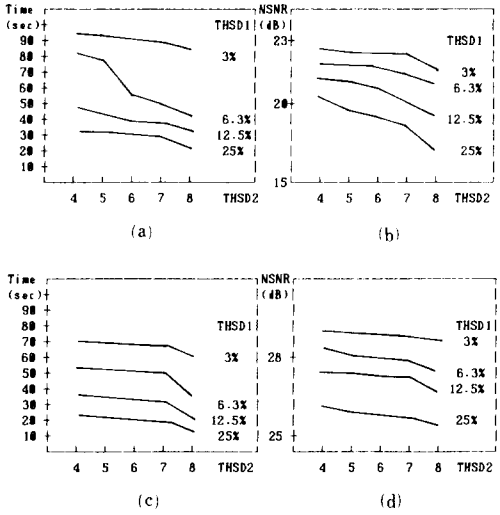

그림 3. Frame 간의 성질을 이용한 방식에서 threshold값의 변화에 따른 performance 비교
THSD1 : 이동화소를 결정하는 threshold 값
THSD2 : 이동부영상을 결정하는 threshold값
(a) 입력 1에서 threshold값과 계산시간과의 관계
(b) 입력 1에서 threshold값과 NSNR와의 관계
(c), (d) 입력 2에서의 관계

Fig. 3. Relation between threshold values and performance in the proposed method that uses the frame differences.
THSD1 : threshold value to segment a moving pel.
THSD2 : threshold value to segment a moving subblock.
(a) relation between thresholds and computation time for input 1.
(b) relation between thresholds and NSNR for input1.
(c), (d) relations for input2.

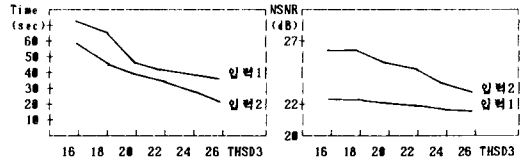

그림 4. 이동벡터 사이의 유사성을 이용한 방식에서 threshold값의 변화에 따른 performance 비교
THSD3 : 한 부영상 내에서 허용할 수 있는 error의 크기
(a) threshold값과 계산시간과의 관계
(b) threshold값과 NSNR와의 관계

Fig. 4. Relation between threshold value and performance in the proposed method that uses the similarities of adjacent motion vectors.
THSD3 : tolerable error magnitude in a subblock.
(a) relation between threshold and computation time.
(b) relation between threshold and NSNR.

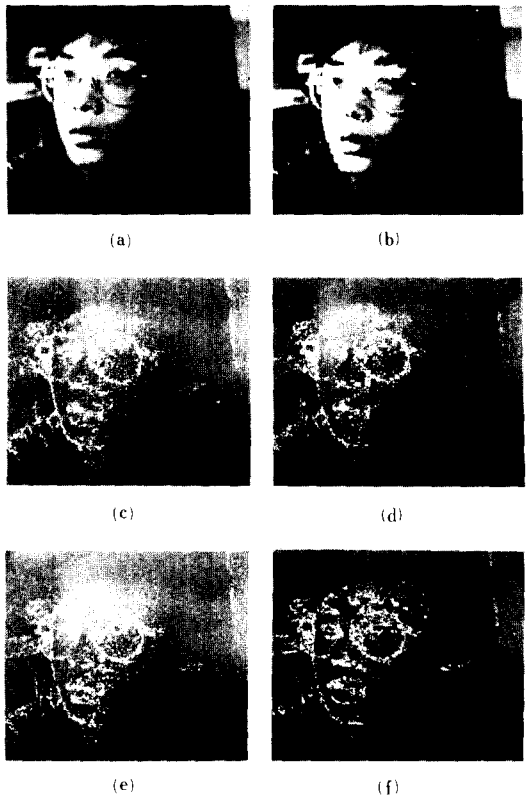

그림 5. 입력 1에서 운동보상 결과 생긴 예측오차의 비교
(a) 원화상
(b) 제안한 방법을 사용하여 얻은 prediction 화상
(c) Three step방식
(d) DMD방식
(e) Menu vector 이용법
(f) 제안한 방식

Fig. 5. Comparison of the prediction error with motion compensation for input 1.
(a) original image.
(b) prediction image using the proposed BMA.
(c) Three step method.
(d) DMD method.
(e) Menu vector method.
(f) proposed method.

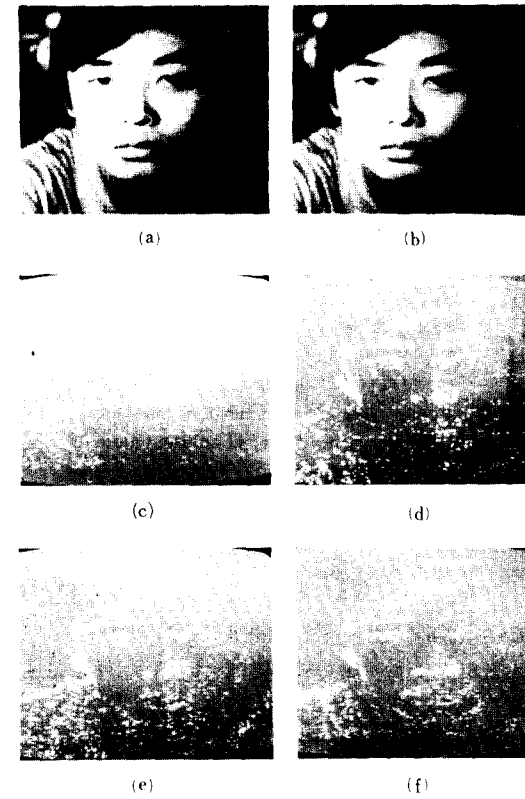

그림 6. 입력 2에서 운동보상 결과 생긴 예측오차의 비교
(a) 원화상
(b) 제안한 방법을 사용하여 얻은 prediction 화상
(c) Three step 방식
(d) DMD 방식
(e) Menu vector 이용법
(f) 제안한 방식

Fig. 6. Comparison of prediction error with motion compensation for input 2.
(a) original image.
(b) prediction image using the proposed BMA.
(c) Three step method.
(d) DMD method.
(e) menu vector method.
(f) proposed method.

만약 부영상의 크기를 8×8로 한다면 three step search 방식에 비해 예측 오차는 1.9dB 정도 증가하고, 예측가능화소는 23.86%로 거의 비슷하며(three step search 방식의 경우 부영상의 크기를 8×8로 하면 예측가능화소는 23.30% 정도), 계산시간은 2.2배 정도 단축된다.

그림 7은 Koga의 three step방법과 제안한 방법과의 운동보상 결과 생긴 예측오차 발생빈도를 비교한 것이다. 여기서 예측오차 $|e|$는 원화상의 gary값(Si)과 운동보상 결과 얻은 화상의 gray값(Ti) 사이의 차의 절대값을 나타낸다. 또 발생빈도는 $|e|$의 값이 발생할 확률로 나타낸다. 제안한 방법은 예측오차면에서

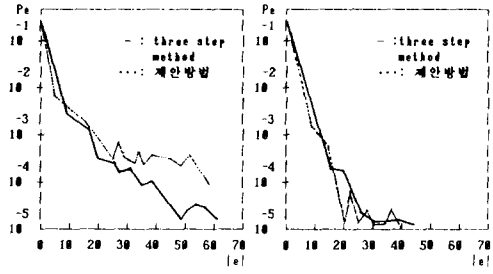

그림 7. 예측오차 발생 빈도비교, |e| : 예측오차의 크기, Pe : |e|의 확률,
(a) 입력1, (b) 입력2

Fig. 7. Comparison of the motion tracking performance. |e| : absolute value of the prediction error.
Pe : probability of the absolute error.
(a) input1, (b) input2.

Koga의 방식과 별차이가 없음을 알 수 있다.

끝으로 표1은 부영상의 크기를 4×4로 하고, 입력1과 입력2에 대해서 평균적으로 구한 계산시간, search point의 갯수, 예측가능화소의 백분율(percentage) 및 운동보상 결과 예측된 영상의 NSNR값을 비교한다. 제안한 방법은 NSNR면에서 약간 뒤쳐지지만 계산시간이 Koga등이 제안한 방법에 비해서 약 1/3~1/4 정도로 단축됨을 알 수 있다.

표 1. 실험결과 성능비교(여기서 제안1은 첫번째 제안한 방식만 이용하여 얻은 결과이고, 혼합은 세가지 방법 모두 혼합 사용하여 얻은 결과이다.)

Table 1. Comparison in performance.

	Search point	계산시간 (sec)	예측가능화소의 백분율(%)	예측영상의 NSNR(dB)
DMD방식	22	80	24.88	28.8
Menu vector방식	25	86	22.21	26.6
Three step방식	25	89	25.87	28.1
OTS방식	8	32	23.46	26.8
제 안1	15	52	25.65	28.1
혼 합	6.9	24	23.89	26.0

V. 결 론

움직임이 있는 물체를 가진 화상을 전송하는데 있어 중요한 것은 데이타 압축도와 계산의 간단화이다. 실제 codec을 꾸미기 위해 계산량을 줄이는 알고리즘이 필요하다. 본 논문에서 제안한 방법은 prediction error를 기존의 방법 정도로 유지한 채 계산량을 줄일 수 있었다.

부영상의 크기를 4×4로 하였을 경우 NSNR가 평균 1.2dB 정도 감소하고 속도는 3배 이상 빨라진다. 또 부영상의 크기를 8×8로 하였을경우 NSNR가 평균 1.9dB정도 감소하고 계산속도는 2.2배 정도 빨라진다.

본 논문은 인간의 시력이 움직이는 부분에 대해 민감하지 못하다는 사실에 근거하여, 약간의 오차 증가 대신에 많은 계산시간 단축을 이루어 실시간 처리 시스템에 적용할 수 있음을 보여준다.

參 考 文 献

[1] A.K. Jain, "Image data compression: A review," *Proc. IEEE,* vol. 69, no. 3, pp. 349-389, Mar. 1981.

[2] N.S. Jayant and Peter Noll, *Digital Coding of Waveforms.* Prentice-Hall, New Jersey, pp. 252-338, 1984.

[3] H.H. Bauch et al., "Picture coding," *IEEE Trans. Commun.*, vol. COM-22, no.9, pp. 1158-1166, Sept. 1974.

[4] B.G. Haskell, *Image Transmission Techniques.* Academic Press, New York, pp. 190-217, 1979.

[5] H.G. Musmann et al., "Advances in picture coding," *Proc. IEEE,* vol. 73, no. 4, pp. 523-541, Apr. 1985.

[6] A.N. Netravali and J.D. Robbins," Motion-compensated television coding: Part I," *Bell Syst. Tech. J.*, vol 58, no. 3, pp. 631-670, Mar. 1979.

[7] 장주욱, CRC와 MCC를 이용한 1.5Mbps 비데오 코덱에 관한연구, KAIST 석사 학위논문,pp. 10-50, 1985년.

[8] F. Giorda and A. Racciu," Bandwidth reduction of video signal via shift vector transmission," *IEEE Trans. Commun.*, vol. COM-23, no. 3, pp. 1002-1004, Mar. 1977.

[9] J.R. Jain and A.K. Jain," Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, no. 10, pp. 1799-1808, Dec. 1981.

[10] Y. Ninomiya and Y. Ohtsuka," A motion-compensated interframe coding scheme for television pictures," *IEEE Trans. Commun.*, vol. COM-30, no. 1, pp. 201-211, Jan. 1982.

[11] T. Koga et al., "Motion compensated interframe coding for video conferencing," *Nat. Telecom. Conf.*, pp. G 5.3.1-G 5.3.5, Nov. 29-Dec. 3, 1981.

[12] R. Srinivasan and K.R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Trans. Commun.*, vol. COM-33, no. 8, pp. 888-896, Aug. 1985.

[13] S. Kappagantula and K.R. Rao, "Motion compensated interframe image prediction," *IEEE Trans. Commun.*, vol. COM-33, no. 9, pp. 1011-1014, Sept. 1985.

[14] 김경수, 이상욱, 송유섭, "Fast 2차원동 표적 추적 알고리즘," 전자공학회지, 제22권, 제1호, pp. 75-85, 1985년 1월.