論文 87-24-4-18

고속 Block Matching 알고리즘에 관한 연구

(A Study on the Fast Block Matching Algorithm)

李 仁 弘*, 朴 來 弘**

(In Hong Lee and Rae Hong Park)

要 約

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

여기 제안된 가산 투영법을 적용한 알고리즘의 장점은 이동치를 구할때 계산시간을 절약할 수 있다는 점이다. 즉, 기존의 block matching 알고리즘은 이차원적인 계산인데 비해 가산투영을 이용하면 일차원적인 계산으로 간단화 시킬 수 있다. 컴퓨터 시뮬레이션 결과를 살펴보면 기존의 알고리즘에 비해 NMSE 값이 약 0.23db정도 차이가 나지만 시각적으로 거의 느낄 수 없으며 계산시간면에서는 대략 3～4 배정도 빠르게 이동차를 구함을 알 수 있다.

Abstract

In this paper an effective block matching algorithm is 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 a proposed algorithm using integral projections is the reduction of the computation time. While the conventional block matching methods have to be computed in 2-dimensional arrays, the proposed algorithm using integral projections can be computed in 1-dimensional arrays. In comparison with conventional block matching methods, a computer simulation shows that though the prediction error increases 0.23 db, it is not detectable for human eyes and the average reduction ratio of computation time obtained from the proposed algorithm is about 3-4.

*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 12月 17日
(* 본 연구는 서울대학교 공과대학 위탁연구 과제의 일부로 이루어진 것임.)

I. 서 론

영상신호를 디지탈 데이타로 전송하는 경우에 있어서 가장 큰 문제는 전송 대역폭의 증가이며, 이에 따라 전송 대역폭을 줄이기 위한 여러가지 데이타 압축(data compression) 방식이 연구되어 왔다.[1,2]

데이타 압축방식은 크게 구성형태에 따라 intraframe coding과 interframe coding으로 나눌 수 있다. Intraframe coding은 공간적 영역(spatial domain)에 존

재하는 중복성(redundancy)을 제거하여 데이타양을 줄이는 방법이고, 반면에 interframe coding은 시간적으로 연속하는 영상들에의 시간적영역(temporal domain)에 존재하는 중복성을 제거하여 데이타양을 줄이는 coding방식이다.[3]

Interframe coding에는 여러가지 방식이 있는데 이중 MCC(motion compensated coding)의 데이타 감축능력이 가장 우수한 것으로 알려져 있다.[4,5] MCC는 영상사이의 이동값을 추정하고 이를 예측에 이용하여 이동값과 예측오차를 전송하는 방식으로 움직임이 큰 영상에서도 좋은 효과를 얻을 수 있다. 그러므로 MCC에서 가장 중요한 문제는 이동값을 검출하는 방법이다.

이동값을 검출하는 방법에는 PRA(pel recursive algorithm)[6]와 BMA(block matching algorithm)[7-12]가 있는데, 전자는 화소단위로 이동값을 추출하여 정확한 값을 얻을 수 있는 반면에 계산량이 많고 복잡하다. BMA는 block 단위로 이동값을 추출하는 만큼 정확도는 떨어지지만 계산량이 많지 않아 실시간 처리에 유리하므로 본 논문에서는 BMA에 근거하여 고속 알고리즘을 제안한다.

본 논문에서는 두 화면의 연속 영상을 데이타 베이스로 하여, 제안한 알고리즘을 중심으로 기존의 여러가지 다른 알고리즘과 비교분석 하였다. Ⅱ장에서는 여태까지 발표되었던 대표적인 BMA를 알아보고, Ⅲ장에서는 기존의 BMA에 가산투영법(integral projection)을 적용한 고속 알고리즘을 소개한 후, Ⅳ장에서는 기존의 알고리즘들과 제안한 알고리즘과의 성능을 컴퓨터 시뮬레이션을 통해 비교 분석하였다. 각 알고리즘의 성능비교에는 영상처리에 많이 쓰이는 NMSE(normalized mean square error)와 계산시간(computation time)을 사용하였다. 또한 각 알고리즘을 사용하여 운동보상한 결과예측된 영상들의 예측오차(prediction error)들을 비교하였다. 끝으로 V장에서는 전체적인 검토 및 결론을 제시하였다.

Ⅱ. 기존의 Block Matching Algorithms

본 장에서는 먼저 BMA의 이론적 배경을 설명하고, 지금까지 제안된 계산량을 줄이는 알고리즘에 대해 알아 본다.[8-12]

1. BMA의 이론적 배경

연속하는 화상(frame)를 사이의 이동량을 검출함에 있어 시간축으로 인접한 화상들내에서 부영상 사이의 상관관계(correlation)를 비교하여 최대치를 이동보상 위치로 이용하는 방법이 BMA이다.[7] (그림1 참조)

BMA의 처리과정은 다음과 같다. 우선 화상을 고정

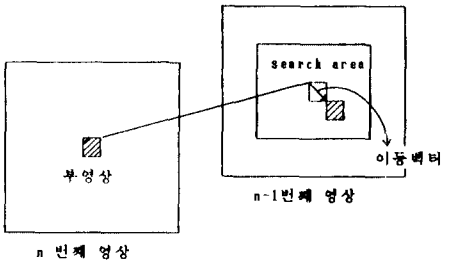

그림 1. BMA에서 이동벡터 검출
Fig. 1. Detection of the Motion Vector in BMA.

된 크기의 부영상으로 나눈다. 이때 이전 화상내의 부영상과의 상관계수 값이 최대가 되는 위치(오차가 제일 작게 일어나는 위치)를 구하기 위해 다음과 같은 함수 D(.)을 정의한다

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

여기서, G(.) : error power를 구하는 비선형 함수
U(m, n) : 현 화상내에서 M×N크기의 부영상으로 구성된 화상
Ur : 이전화상 내에서 (M+2p)×(N+2p) 크기의 search area
p : 최대 이동가능거리, $-p \le i, j \le p$

이다. 이때 이동치는 D(i, j)를 최소로 하는 (i, j)로 주어진다. 이것은 D(.)값이 최적위치(optimal position)에서 멀어질수록 증가함을 보여준다. 즉 정확한 이동치를 갖는 위치에서 멀어지면 멀어질수록 오차가 커지므로 이동값을 구하는 것은 D(.)값을 작게하는 위치로 찾아가는 것이다. 이와같은 가정하에 계산량을 줄일 수 있는 효과적인 BMA를 제안할 수 있다.

2. DMD방식

이 방법은 Jain & Jain이 제안한 것인데 2-D logarithmic 알고리즘이라고도 한다. DMD(direction of the minimum distortion)방식에서 cost function으로 식(2)로 정의된 MSE(minimum mean square error)를 이용한다.[8]

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

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

이다.

이 search 방법의 각 단계는 SR의 중심을 포함하여

5개의 위치에 대한 MSE값을 계산한다. 이중 가장 작은 MSE값을 갖는 위치를 중심으로 하여 다시 순환적으로 반복한다. 이 과정은 SR의 크기가 3×3이 될 때까지 계속되며 마지막 단계에서는 9개의 위치에 대해 MSE값을 비교 비교하여 가장 작은 MSE값을 갖는 위치를 구한다. 이때 이 위치가 바로 이동치가 된다(그림 2(a) 참조).

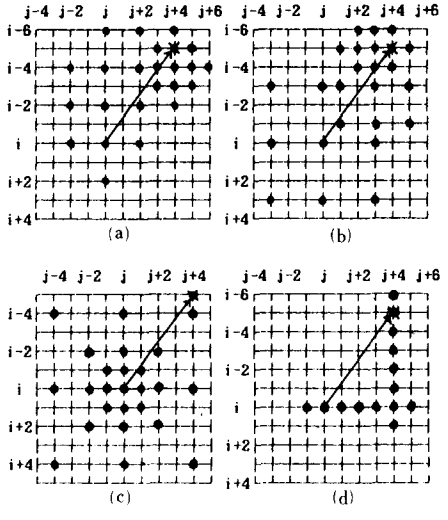

그림 2. 기존의 BMA방식, *: 찾고자 하는 위치
(a) DMD방식 : (i−2, j) → (i−4, j) → (i−4, j+2) → (i−4, j+4) → (j−5, j+4)
(b) Three step방식 : (i−3, j+3) → (i−5, j+3) → (i−5, j+4)
(c) Menu vector방식 : (i−4, j+4)
(d) OTS 방식 : (i, j+1) → … → (i, j+5) → (i−1, j+4) → … → (i−5, j+4)

Fig. 2. Conventional BMA Methods. *: 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) Three Step Method : (i−3, j+3) → (i−5, j+3) → (i−5, j+4).
(c) Menu Vector Method : (i−4, j+4).
(d) OTS Method : (i, j+1) → … → (i, j+5) → (i−1, j+4) → … → (i−5, j+4).

DMD 방식을 약간 변형시켜 search point의 수를 줄이고 임계값(threshold value) 개념을 사용한 modified DMD 방식도 계산량을 줄이는데 효과적이다.[12]

3. Three Step Search 알고리즘

이 방법은 Koga등이 제안한 것으로 (3)식과 같이 정의된 MAD(mean of the absolute frame difference)를 cost function으로 이용한다.[9]

$$MAD(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}: S_{t,k}(m,n)-S_{t-1,k}(m+i,n+(m+i,\ n+j): \qquad (3)$$

Three step방식은 각 search단계 마다 SR의 크기를 줄여가며 9개 위치에 대해 MAD 값을 비교해서 최소가 되는 위치를 찾는다. 이 위치가 이동벡터가 된다. (그림2(b) 참조)

이 방법은 세 단계에 걸쳐 SR을 줄여가는 것으로 비교적 정확한 값을 구할 수가 있다.

4. Menu Vector 이용법

이 방법은 Ninomiya & Ohtsuka가 제안한 것으로 미리 menu vector의 집합을 만든 다음 그 menu vector에 대해 search함으로써 이동치를 찾는다.[10] Menu vector의 집합 Y_i는 다음과 같이 정의된다.

$$Y_i=\{(y_1,y_2)\mid y_1,y_2: 0,\ \pm 2^{n-i},\ \pm 2^{n-i+1}\}$$
$$i=1,\cdots,n \qquad (4)$$

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

그림 2(c)는 menu vector를 이용하여 이동량을 구하는 방법을 보여 주는데 점(.)으로 표시한 위치가 menu를 나타낸다. 여러 menu들 중에서 가장 오차가 작은 menu를 선택하여 이 위치를 이동값으로 취한다.

5. OTS 알고리즘

Srinivasan & Rao가 제안한 OTS(one at a time search) 방식의 search 과정은 다음과 같다.[11]

우선 어느 한 축을 따라서 원점을 중심으로 3개의 위치에 대해 MAD를 계산하고, MAD가 감소하는 방향으로 search한다. 이 방향으로 계속 search하다가 최소가 되는 점이 가운데 점이면 처음 search한 방향의 conjugate 방향으로 search 과정을 반복한다(그림 2(d) 참조).

Ⅲ. 가산 투영법을 이용한 고속 알고리즘

본 장에서는 기존의 BMA에 본 논문에서 제안하는 가산 투영법(integral projection)을 적용하여 물체의 이동정보를 구하는 고속 알고리즘에 대해 설명한다. 가산투영을 이용하면 기존의 2차원적인 계산을 1차원적인 계산으로 줄일 수가 있어서 많은 계산상의 이득을 얻을 수 있다.

1. 가산 투영을 이용한 계산량 감축

가산 투영법은 화상신호가 있을 때 임의의 방향에서 투영하여 그 투영선상에서 각 화소가 갖고 있는 gray 값을 모두 합하는 것이다. 디지탈 화상 F의 열(column) 성분 화소들을 더하여 얻은 벡터를 수직 방향의 가산투

영, 행(row) 성분 화소를 더하여 얻은 백터를 수평 방향의 가산투영이라 한다. 수직(수평) 가산투영은 그 방향의 F의 밝기 분포를 나타내므로 가산 투영법은 화상들을 구별하는데 사용할 수 있다. 본 논문에서는 수직 가산투영과 수평 가산투영을 이용하는데 그림4는 수직(수평) 가산투영값을 구하는 방법을 나타낸다.

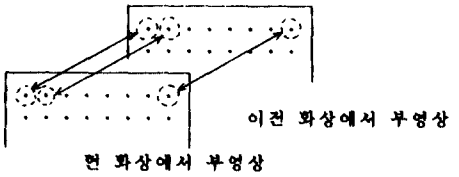

그림3. MAE나 NMSE에 의한 cost 함수계산
Fig. 3. Calculation of the Cost Function by Using MAE or NMSE.

즉 그림3에서 처럼 화소 개개에 대하여 뺄셈, 절대값 계산, 덧셈을 취하는 것이 아니라 일단 기준되는 search point에 대해 수직 및 수평 가산투영을 통한 값을 구한 다음 이 값들을 가지고 뺄셈, 절대값 계산, 덧셈을 하는것이다. 기존의 방식대로 한다면 한 search point에 대해 8×8 부영상의 경우 64번의 뺄셈계산, 64번의 절대값계산, 63번의 덧셈계산이 필요하지만 가산투영을 이용하면 처음의 가산 투영값이 필요할 뿐 나머지 search 과정에 대해서는 다시 계산할 필요없이 필요한 값만 update 해주면 된다. 이것은 원래 MAD나 NMSE로 cost를 계산할 경우 2-D로 해야 하는 것을 가산투영을 이용하여 1-D인 한 방향으로만 고려하는 것이다.

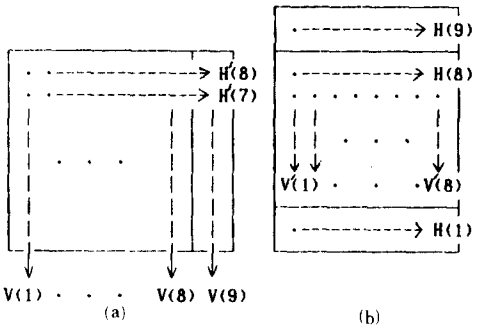

그림4. 8×8 크기의 가산 투영을 이용한 경우
(a) 수평방향으로 Search할 경우
(b) 수직방향으로 Search할 경우
Fig. 4. Integral Projection in 8×8 Subblock Size.
(a) Search in Horizontal Direction.
(b) Search in Vertical Direction.

그림4(a)는 search방향이 수평일 경우 사용되는 가산투영 방식이다. 이 경우 바로 옆의 search point에 대해서는 수직 가산투영일 경우 단지V(9)란 값만 update하면 된다. 그리고 수평 가산투영일 경우에는 기존의 수평 가산투영값에서 맨 왼쪽의 각 화소들의 값을 빼주고 맨 오른쪽의 화소들의 값을 더해주면 된다.

그림4(b)는 search방향이 수직일 경우 사용되는 가산투영 방식이다. 수직 가산투영값을 구할 경우 바로 위의 search point에 대해서는 맨 아래행의 값을 빼주고 위 행의 값을 더해 주면 되고, 수평 가산 투영값을 구할 경우에는 H(9)만 update하면 되므로 많은 계산상의 이득이 있음을 쉽게 알 수 있다.

2. 가산투영을 적용한 BMA

본 절에서는 기존의 방법들에 가산투영을 적용하는 것을 보여준다. 가산투영을 적용하는 경우 우선적으로 구해야 하는 값들은 다음과 같다. 먼저 현화상에서 고려대상이 되는 부영상의 가산 투영값을 구해야 한다. 그 다음 이전 화상에서 같은 위치에 있는 부영상의 가산 투영값을 그림4에서 처럼 구한다.

알고리즘 B-1(DMD방식에 가산투영을 적용한 경우)에서 나머지 search point들의 가산투영값들은 다음과 같이 구한다. 각 단계에서 좌, 우로 2화소씩 떨어진 search point들의 수직 가산 투영값들은 처음 구한 기준 가산투영값에다 단지2개의 열에 대한 수직 가산투영값만 update하면 된다. 그리고 상, 하로 2화소씩 떨어진 search point들의 수직 가산투영값들은 이미 구한 수직 가산 투영값에 2개 화소의 gray 값을 빼주고 새로운 2개화소의 gray값을 더해주면 된다.

이렇게 구한 가산투영값을 이용하여 cost를 계산하고 가장 최소 cost를 갖는 search point를 기준으로 하고 이 search point의 가산투영값을 이용하여 위의 과정을 반복한다. 마지막 단계에서는 기준 search point의 가산 투영값을 이용하여 좌, 우로 1화소 떨어진 search point들에 대해서는 각각 한 열씩만 더하여 update하면 되고 상, 하로 1화소 떨어진 search point들에 대해서는 각각 1화소의 gray값을 빼주고 새로운 한 화소의 gray 값을 더해주면 된다. 수평 가산투영값은 행성분으로 고려하는 것만 제외하고는 수직 가산 투영값을 구하는 방법과 같다. 이렇게 구한 가산 투영값을 이용하여 cost를 계산하면 많은 계산량의 감축을 얻는다.

알고리즘 B-2(three step방식에 가산투영을 적용), 알고리즘B-3(Menu vector 방식에 가산투영을 적용), 알고리즘B-4(OTS 방식에 가산투영을 적용)의 경우도 위와 비슷한 방법으로 가산 투영값을 구하여 cost

계산에 이용할 수 있다. 우선 기준이 되는 search point의 가산투영값을 구한다. 그 다음 좌, 우로 2화소(3화소) 떨어진 search point에 대해서는 수직 가산투영일 경우 단지 2개(3개)의 열에 대한 수직 가산투영값만 update하면 된다. 그리고 수평 가산투영일 경우에는 기존의 수평 가산 투영값에서 2개(3개) 화소의 gray값을 빼주고 새로운 2개(3개) 화 소의 gray 값을 더해주면 된다. 또 상, 하로 2화소(3화소) 떨어진 search point에 대해서는 수직 가산투영일 경우 이미 구한 수직 가산투영값에 2개(3개) 화소의 gray 값을 빼주고 새로운 2개(3개) 화소의 gray 값을 더해주면 된다. 그리고 수평 가산투영일 경우에는 단지 2개(3개) 행에 대한 수평 가산 투영값만 update 해주면 되므로 많은 계산상의 이득이 있다.

계산상으로 기존의 BMA와 가산투영을 적용한 경우를 비교하면 표1과 같다. 여기서 한 부영상당 필요한 search point의 수는 평균적으로 DMD방식의 경우 22번,[8] three step방식[9]과 menu vector방식[10]의 경우 25번, OTS방식의 경우 12번[11]으로 고려하였다.

표 1. 기존의 방법과 제안한 방법과의 계산수 비교
Table 1. Comparison Between Conventional BMA and the Proposed Algorithm in Computation Amount.

	덧 셈	뺄 셈	절대값
DMD 방식	1386	1408	1408
알고리즘B-1	904	520	288
Three step방식	1575	1600	1600
알고리즘B-2	1298	736	432
Menu vector방식	1575	1600	1600
알고리즘B-3	942	656	400
OTS방식	756	768	768
알고리즘B-4	558	288	192

Ⅳ. 컴퓨터 시뮬레이션 결과 및 검토

본 장에서는 기존의 BMA에 가산투영법을 적용한 컴퓨터 시뮬레이션의 결과를 나타낸다. 즉, 알고리즘B-1, 알고리즘 B-3, 알고리즘 B-4 각각에 대해 시뮬레이션을 하여 기존의 방법과 비교한다.

본 시뮬레이션에서는 입력으로 크기가 256×256인 연속화면을 이용하였다. 입력1은 움직임이 빠른 연속화면이고, 입력2는 움직임이 느린 연속화면이다. 즉, significant pixel[9]을 연속화면에서 gray level 변화가 일정값(4) 이상인 것으로 정의한다면 입력1은 20%, 입력2는 12% 정도의 significant pixel을 함유한다 (그림5 참조).

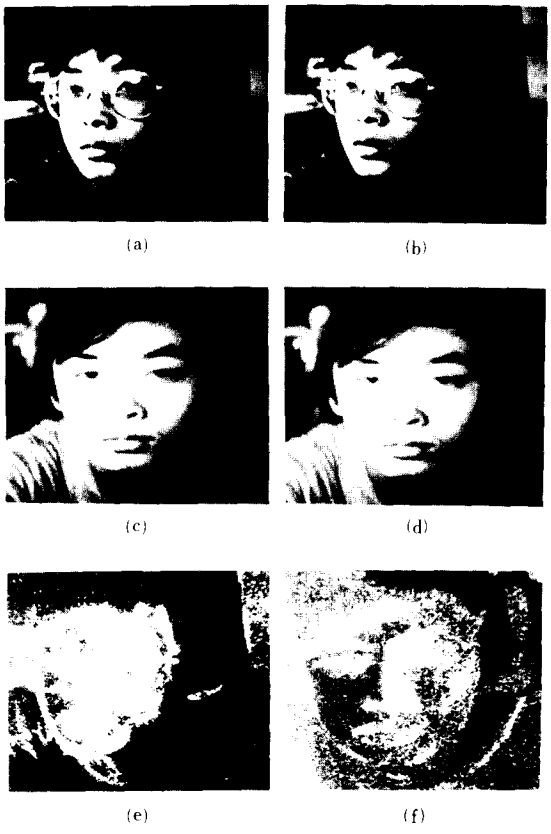

그림 5. 시뮬레이션에 사용된 입력영상
(a) 입력1에서 이전 영상
(b) 입력1에서 현재 영상
(c) 입력2에서 이전 영상
(d) 입력2에서 현재 영상
(e) 입력1의 움직임 정도
(f) 입력2의 움직임 정도

Fig. 5. Input Images for a Computer Simulation.
(a) Previous Frame in input 1.
(b) Present Frame in Input 1.
(c) Previous Frame in Input 2.
(d) Present Frame in Input 2.
(e) Difference Image for Input 1.
(f) Difference Image for Input 2.

시뮬레이션은 IBM-PC/XT를 이용하였으며 부영상의 크기는 8×8로 고정시켰고, 물체의 최대이동 가능거리는 6화소로 제한하였다. 또 성능비교는 계산시간(computation time)과 NMSE등을 이용하였으며, 각 알고리즘에 의해 예측된 영상을 원화상과 비교하여 오차를 영상으로 나타내었다. 본 논문에서 사용한NMSE의 정의는 다음과 같다.

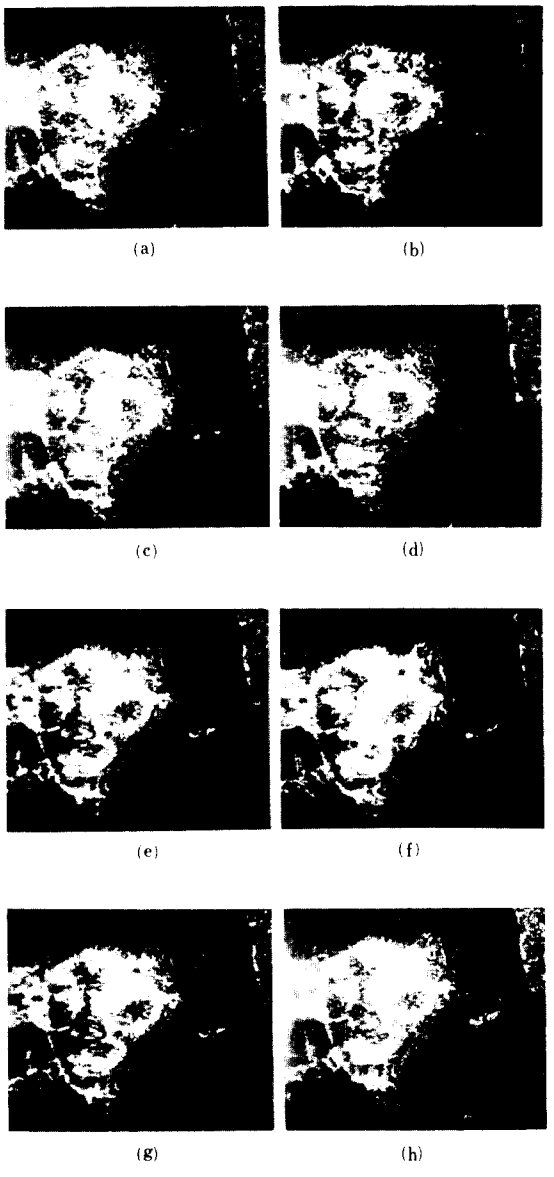

그림 6. 입력 1 에서 예측오차의 비교
(a) DMD방식
(b) 알고리즘B-1
(c) Three Step방식
(d) 알고리즘B-2
(e) Menu Vector방식
(f) 알고리즘B-3
(g) OTS방식
(h) 알고리즘 B-4

Fig. 6. Comparison of the Prediction Error for Input 1.
(a) DMD Method. (b) Algorithm B-1.
(c) Three Step Method. (d) Algorithm B-2.
(e) Menu Vector Method. (f) Algorithm B-3.
(g) OTS Method. (h) Algorithm B-4.

$$NMSE = -10 \times \log \frac{E\{\{Si-Ti\}^2\}}{E\{Si^2\}} \quad (5)$$

여기서 Si는 원화상의 gray level이고, Ti는 운동보상 결과 얻은 영상의 gray level이다. 그리고 정확하게

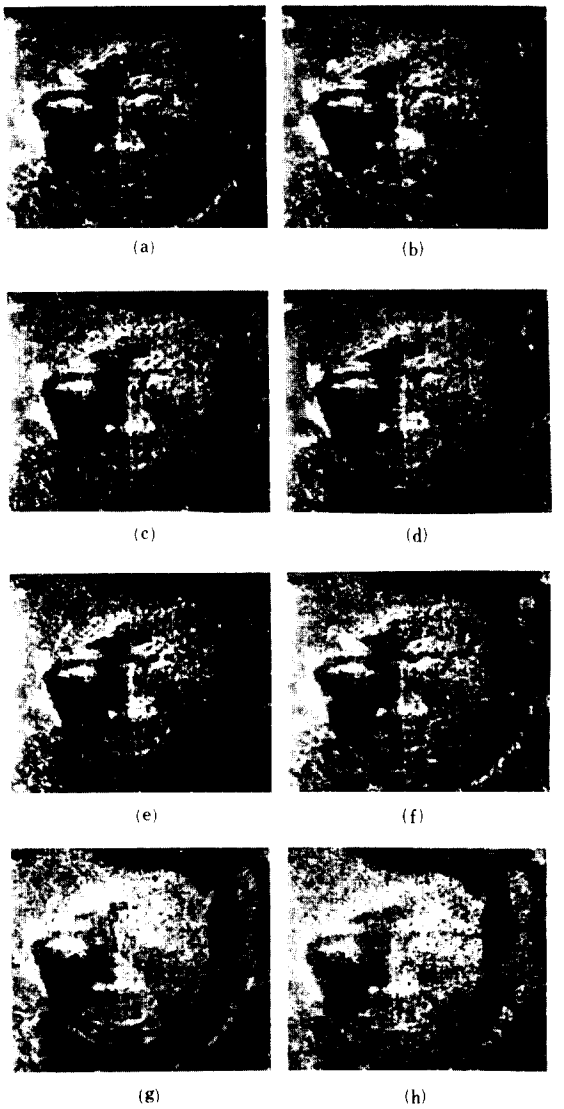

그림 7. 입력 2 에서 예측오차의 비교
(a) DMD방식 (b) 알고리즘B-1
(c) Three Step방식 (d) 알고리즘B-2
(e) Menu Vector방식 (f) 알고리즘 B-3
(g) OTS방식 (h) 알고리즘 B-4

Fig. 7. Comparison of the Prediction Error for Input 2.
(a) DMD method (b) Algorithm B-1.
(c) Three Step Method. (d) Algorithm B-2.
(e) Menu Vector Method. (f) Algorithm B-3.
(g) OTS Method. (h) Algorithm B-4.

운동보상 결과 예측된 화소(Ti=Si인 화소)를 예측가능화소(predictable pixel)로 정의하여 각 방법들이 어느 정도 정확하게 이동값을 구했는지 비교하였다.

그림6과 그림7은 운동보상 결과 생기는 예측오차(prediction error)를 영상으로 보여주는데 각각 입력1, 입력2에 대한 시뮬레이션 결과이다. 여기서 흰 부분이 운동보상 결과 생긴 예측오차를 나타낸다. 이때 흰 부분의 밝기는 원래 값에 8배한 것이다. 그림6와 그림7(a),(b)는 기존의 DMD 방식과 알고리즘 B-1을 이용하여 운동보상한 결과 생긴 예측오차를 나타낸다. 이 경우 알고리즘 B-1의 NMSE 값이 약 0.5db정도 뒤 떨어지는데, 실제 오차의 증가는 12% 정도이고 시각적으로 거의 오차의 증가를 인식하지 못하였다. 그림6과 그림7의 (c), (d)는 각각 three step 방식과 알고리즘B-2를 이용하여 운동보상한 결과 생긴 예측오차를 나타낸다. 이 경우 알고리즘B-2의 NMSE값이 약 0.2db정도 떨어지고 오차의 증가는 4.7%에 불과하다. 그림6과 그림7의 (e), (f)는 각각 원 menu vector방식과 알고리즘B-3을 이용하여 운동보상한 결과 생긴 예측오차를 보여준다. 이 경우는 오히려 알고리즘B-3의 경우가 NMSE값이 0.1db 정도 더 큰데 시각적으로 별 차이가 없다. 그림6과 그림7의 (g), (h)는 각각 원 OTS방식과 알고리즘B-4를 이용하여 운동보상한 결과 생긴 예측오차를 보여준다. 이 경우 알고리즘B-4의 NMSE값이 약 0.3db정도 뒤떨어지지만 실제 오차의 증가는 7% 정도로 시각상 별 차이를 느끼지 못한다. 이 결과들로 부터 가산투영을 적용한 경우 기존의 방법들에 비하여 예측오차나 화질면에서 거의 뒤쳐지지 않음을 알 수 있다.

표 2. 각 방식의 성능비교
Table2. Comparison in Performance.

	계산시간 (sec)	NMSE (dB)	예측가능 화소(%)
DMD방식	73	27.8	23.20
알고리즘B-1	19.5	27.3	23.12
Three step방식	78	26.8	23.30
알고리즘B-2	29	26.6	23.03
Menu vector방식	77	25.6	22.94
알고리즘B-3	23	25.7	21.37
OTS방식	57	25.7	22.66
알고리즘B-4	14.5	25.4	22.88

끝으로 표2는 입력1과 입력2에 대해서 평균적으로 구한 계산시간, 예측된 영상의 NMSE값과 예측가능 화소의 백분율(%)을 비교한다. 제안한 방법들은 NMSE면에서도 거의 비슷하며 특히 계산시간이 기존의 방법들에 비해 약 1/3~1/4 정도로 단축됨을 알 수 있다.

V. 결 론

움직임이 있는 물체를 가진 화상을 전송하는데 있어서 중요한 것은 움직임의 정확한 검출과 필요한 계산의 간단화이다. 따라서 실시간 동작 영상 부호기를 구현하기 위해서는 계산량을 줄이는 고속 알고리즘이 절대적으로 필요하다.

본 논문에서는 기존의 BMA에 가산투영법을 적용하여 계산을 1차원으로 단순화하여 많은 계산량을 줄였다. 이 경우 오차면에서는 기존의 BMA에 비해 NMSE가 평균 0.23db 정도의 손실은 있으나 계산시간이 기존 방법들의 약 1/3~1/4 정도로 단축되었다. 특히 본 논문에서 사용한 가산투영은 특성상 잡음에 강하므로 실제 영상신호를 전송할 경우 잡음이 있는 영상에 대해서는 기존의 BMA에 비해 오히려 좋은 화질을 얻을 수 있을 것으로 예상된다.

본 논문은 인간의 시력이 움직이는 부분에 대해서는 예리하지 못하다는 사실에 근거하여 약간의 오차를 감수하고 많은 계산시간을 절약하여 실시간 처리에 응용할 수 있는 가능성을 보였다.

參 考 文 献

[1] A.K. Jain, "Image data compression: A review," *Proc. IEEE*, vol. 69, no. 3, pp. 384-389, Mar. 1981.

[2] A.N. Netravali & J.O. Limb, "Picture coding: A review," *Proc. IEEE*, vol. 68, no. 3, pp. 366-406, Mar. 1980.

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

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

[5] 장주욱, CRC와 MCC를 이용한 1.5Mbps 비데오 코덱에 관한 연구, KAIST 석사학위 논문, pp. 10-50, 1985년.

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

[7] F. Giorda & A. Racciu, "Bandwidth reduction of video signal via shift vector transmission," *IEEE Trans. Commun.*, vol. COM-

23, no. 3, pp. 1002-1003, Mar. 1977.

[8] J.R. Jain & 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.

[9] 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.

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

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