움직임 벡터들의 방향과 크기를 고려한 프레임율 증가 기법

*박종근 **배창영 ***이경준 ****정제창[a)‡]

한양대학교 전자컴퓨터통신공학과

*parkjk14@hanyang.ac.kr, **baepang@hanyang.ac.kr, ***kjlee888@naver.com, ***jjeong@hanyang.ac.kr

Frame Rate Up-Conversion Considering The Direction and Magnitude of Motion Vectors

*Jonggeun Park **Changyoung Bae ***Kyungjun Lee ****Jechang Jeong

Department of Electronic and Computer Engineering, Hanyang University

요약

본 논문은 EBME(Extended Bilateral Motion Estimation) 알고리듬에서 움직임 벡터들의 방향과 크기를 고려한 알고리듬을 제안하였다. EBME는 높은 연산량을 요구하기 때문에 프레임 내의 x, y방향 각각의 평균 움직임 벡터크기를 이용하여 동적프레임과 정적프레임을 판단하고, EBME 수행여부를 결정하여 연산량을 줄인다. 또한 동일한 움직임 벡터들의 방향과 크기를 비교하여 MVS(Motion Vector Smoothing)단계 수행여부를 판단함으로써 연산량을 줄인다. 제안하는 알고리듬을 적용한 실험 결과 기존의 EBME 알고리듬에 비해 수행시간은 단축되었으나 PSNR(Peak Signal to Noise Ratio)은 향상 되었다.

1. 서론

최근 UHD TV(Ultra High Definition TV)가 대중화 되면서 가격이 저렴해지고, 보급률이 높아짐에 따라 고화질영상의 활용도가 높아지고 있다. 하지만 UHD TV의 해상도가 높아지고, 사이즈가 커질수록 주파수가 낮아져 화면이 깜박 꺼리는 flicker현상이 발생한다. 이런 문제를 해결하기 위해서 주파수를 증가시키는 FRUC(Frame Rate Up-Conversion)를 이용하여 프레임수를 증가시키는 방법이 제안되었다[1-3].

초기의 FRUC 알고리듬은 이전프레임과 현재프레임의 평균값을 이용하여 보간프레임을 생성한다. 하지만 이 방법은 영상 내 물체의 움직임을 고려하지 않기 때문에 열악한 결과 이미지를 보인다. 이를 보완하기 위해 제안된 MC-FRUC(Motion- Compensated FRUC)는 움직임 예측(Motion Estimation) 과정에서 물체의 움직임을 찾아 움직임 벡터(Motion Vector)를 구하고 이를 이용하여 MCI(Motion -Compensated Interpolation) 과정에서 보간프레임을 생성한다[4].

초기에 제안된 FRUC 기술은 블록 정합 알고리듬(Block Matching Algorithm)으로 단방향 움직임 예측(Unilateral Motion Estimation)을 이용하여 움직임을 추정하였다. 이는 현재프레임의 블록을 중심으로 이전프레임으로부터 움직임 벡터를 찾기 때문에 보간프레임 생성 시 홀(hole)과 중첩(overlap) 영역이 생기는 단점이 있다.

a)한양대학교 전자컴퓨터 통신공학과(Department of Electronic and Computer Engineering, Hanyang University)

‡ Corresponding Author : 정제창(Jechang Jeong)

E-mail: jjeong@hanyang.ac.kr

Tel: +82-2-2220-4370

이런 문제를 해결하기 위해서 보간프레임의 각 블록을 기준으로 이전프레임과 현재프레임으로부터 대칭성을 이용하여 움직임 벡터를 찾는 BME(Bilateral Motion Estimation)이 제안되었다[6]. 또한 MC-FRUC의 문제점을 보완하기 위해 향상된 신뢰도의 움직임 벡터를 구하기 위한 다양한 FRUC 알고리듬이 제안되었다[2-4].

본 논문에서 EBME(Extended Bilateral Motion Estimation) 알고리듬은 BME를 두 번 수행하고, MVS(Motion Vector Smoothing)단계에서 모든 이상 벡터가 수정될 때까지 반복하기 때문에 높은 연산량을 요구한다. EBME의 연산량을 줄이기 위해 제안하는 알고리듬은 프레임 내의 x, y방향 각각의 평균 움직임 벡터크기를 이용하여 동적프레임과 정적프레임을 판단하고, EBME 수행여부를 결정하여 연산량을 줄인다. EBME단계를 거친 최종적인 움직임 벡터들에서 주변의 동일한 방향과 크기를 비교하여 MVS단계 수행여부를 판단함으로써 연산량을 줄인다.

본 논문의 구성은 다음과 같다. 2장에서는 EBME 알고리듬을 설명하고, 3장에서는 제안한 알고리듬에 대해 설명한다. 그리고 4장에서는 실험결과를 제시하고, 마지막으로 5장에서 결론을 맺는다.

2. 기존 알고리듬

2-1. EBME(Extended Bilateral Motion Estimation)

EBME는 BME를 한번 수행한 후 다음 블록의 그리드(grid)에서 영상의 가로축과 세로축 방향으로 블록의 크기의 반만큼 이동하여 새로운 블록을 형성하고, 다시 쌍방향 움직임예측을 수행하여 중첩된 영역에 더 정확한 움직임 벡터를 사용한다[6].

그림 1의 Original block에 대하여 쌍방향 움직임 예측으로 식 (1), 식 (2)를 이용하여 움직임 예측을 한다. 정확한 움직임 벡터를 구하기

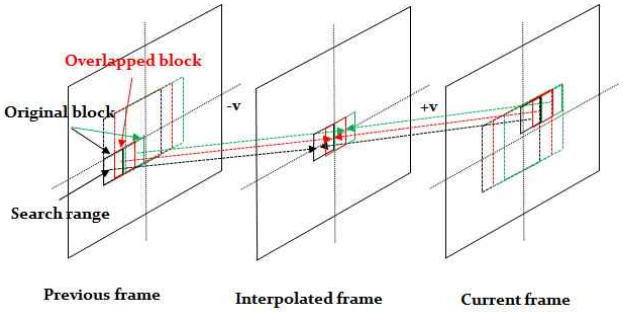

그림 1. EBME에서의 움직임 예측

위해 추가적으로 Overlapped block을 식 (1), 식 (2)를 이용하여 SBAD(Sum of Bilateral Absolute Differences)값이 최소가 되는 위치로 정한다. 그리고 정확한 움직임 벡터를 구하기 위해 추가적으로 Overlapped block을 식(1), 식(2)를 이용하여 SBAD(Sum of Bilateral Absolute Differences)값이 최소가 되는 위치로 정한다.

$$SBAD(dx,dy) = \sum_{x\in S_x}\sum_{y\in S_y}|f_{n-1}(x-dx,y-dy)-f_n(x+dx,y+dy)| \quad (1)$$

$$v = \arg\min_{(dx,dy)\in S_{x,y}} SBAD(dx,dy) \quad (2)$$

식 (1), 식 (2)에서 (dx,dy)는 움직임 벡터 후보를 나타내며, f_{n-1}은 이전 프레임, f_n은 현재 프레임을 나타낸다.

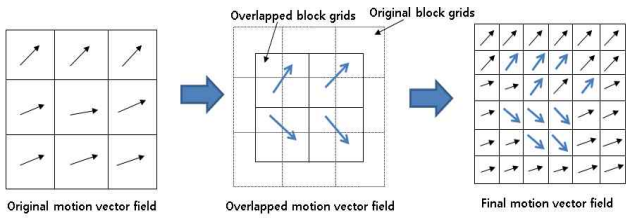

그림 2. EBME에서 중복된 격자 움직임 벡터 선택 과정

다음으로 original block과 overlapped block의 겹쳐지는 영역에 대하여 움직임 벡터를 비교한다. 각각의 블록은 그림 2와 같이 16×16 크기 단위로 구한 움직임 벡터를 네 개의 8×8 단위 블록으로 나누어 겹쳐지는 부분에 대하여 SBAD가 최소가 되는 움직임 벡터로 수정한다.

2-2. MVS(Motion Vector Smoothing)

최종적으로 구해진 움직임 벡터에는 잘못 판단된 움직임 벡터들이 존재 한다. 따라서 식 (3)을 이용하여 $D_o > D_n$인 경우 이상 벡터로 판단하며, 그림 3과 같이 이상 벡터인 v_0는 정확한 움직임 벡터(correct motion vector)들이 median필터를 거쳐 수정된다. 이상 벡터인 v_0가 한 번의 수행을 거쳐도 남아있으면, 그림 3과 같이 다시 똑같은 과정을 반복하여 모든 이상 벡터가 수정될 때까지 반복한다. 식 (3)에서 v_m은 평균 움직임 벡터이며, v_0는 이상벡터를 나타낸다.

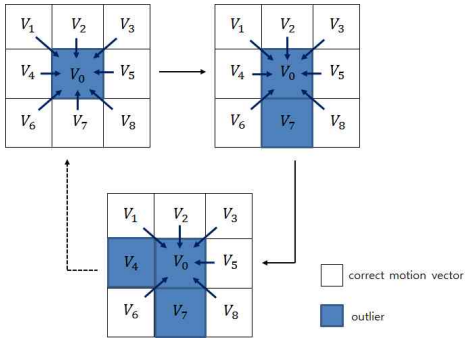

그림 3. MVS 과정

$$\begin{aligned} v_m &= \frac{1}{9}\sum_{i=0}^{8} v_i \\ D_o &= abs(v_m - v_0) \\ D_n &= \frac{1}{8}\sum_{i=1}^{8} abs(v_m - v_i) \end{aligned} \quad (3)$$

2-3. MCI(Motion Compensated Interpolation)

MCI는 가중치를 사용하여 이전프레임과 현재프레임 중 일치하는 영역이 많은 프레임을 찾아 높은 가중치를 부여한다.

$$SOAD = \sum_{x\in O_x}\sum_{y\in O_y}|f_c(x,y)-f_v(x,y)| \quad (4)$$

먼저, 그림 4와 같이 확장된 블록이 식 (4)의 SOAD (Sum of Overlapped area Absolute Difference)를 이용하여 확장된 블록의 오차 크기를 비교 한다. f_c는 현재프레임의 확장된 블록을 나타내고, f_v는 가상 보상프레임의 확장된 블록을 나타낸다. O_x와 O_y는 확장된 블록의 영역에 대한 x와 y의 위치를 나타낸다.

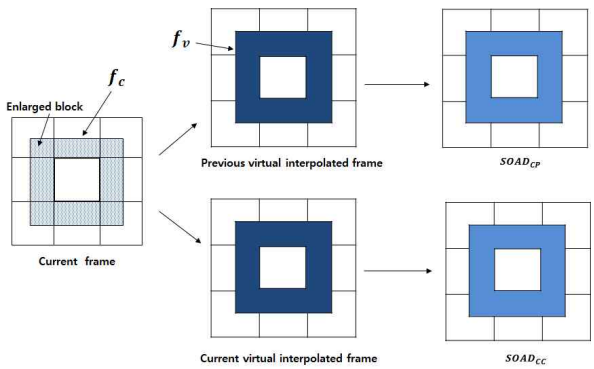

그림 4. $SOAD_{\alpha}$, $SOAD_{cp}$에 대한 진행 과정

그림 4에서 $SOAD_{cp}$는 현재프레임에서 확장된 블록과 이전 가상 보상 프레임에서 확장된 블록의 차에 대한 절대값이고, $SOAD_{\alpha}$는 현재프레임에서 확장된 블록과 현재 가상 보상프레임에서 확장된 블록의 차에 대한 절대값이다. 그림 4를 통해 $SOAD_{cp}$와 $SOAD_{\alpha}$는 현재프레임에서 겹쳐지는 영역을 판단한다. $SOAD_{pp}$는 이전프레임에서 확장된 블록과 이전 가상 보상프레임에서 확장된 블록의 차에 대한 절

대값 이고, $SOAD_{pc}$는 이전프레임에서 확장된 블록과 현재 가상 보상 프레임에서 확장된 블록의 차에 대한 절대값이다.

$$\alpha = \frac{SOAD_{\alpha c} + SOAD_{cp}}{SOAD_{\alpha c} + SOAD_{cp} + SOAD_{pc} + SOAD_{pp}} \tag{5}$$

식 (5)에서 현재프레임과 이전프레임의 확장된 블록의 겹쳐지는 영역에 따라 가중치가 변한다.

$$\begin{aligned} B_{n-\frac{1}{2}}(x,y) = \alpha \cdot\ & w(x,y) \cdot\ B_{n-1}(x-dx, y-dy) \\ & + (1-\alpha) \cdot\ w(x,y) \cdot\ B_n(x+dx, y+dy) \end{aligned} \tag{6}$$

식 (6)은 보상프레임의 최종식을 나타내며, $w(x,y)$는 OBMC(Overlapped Block Motion Compensation)를 위한 함수를 나타낸다[7]. 그리고 B_{n-1}은 확장블록의 이전프레임, B_n은 확장블록의 현재프레임을 나타낸다.

3. 제안하는 알고리듬

제안하는 알고리듬의 흐름도는 그림 5이며, 프레임 내의 x, y방향 각각의 평균 움직임 벡터크기를 이용하여 동적프레임과 정적프레임을 판단하고, EBME 수행 여부를 결정한다. 또한 움직임 벡터들의 방향과 크기를 비교하여 동일한 움직임 벡터를 판단하고 MVS 수행 여부를 결정한다.

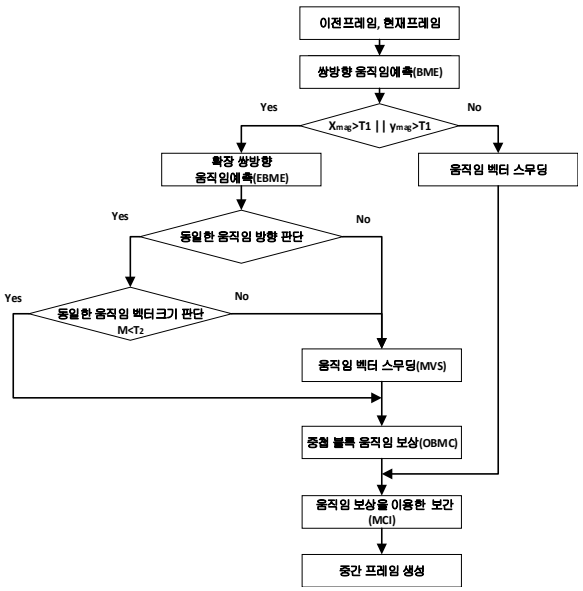

그림 5. 제안하는 알고리듬의 흐름도

3-1. 프레임 내의 움직임 정도 판별

프레임 내의 움직임 정도에 따라 동적프레임과 정적프레임으로 분류한다. 식 (7), 식 (8)에서 동적프레임인 경우 움직임 벡터의 변화가 많아 정확한 벡터를 찾는 EBME를 이용하고 정적프레임인 경우 움직임 벡터의 변화가 적어 연산량이 낮은 BME를 이용한다. 식 (7)에서 N은 프레임 내의 블록의 총 개수이고, x_{mag}과 y_{mag}은 프레임 내의 x, y 방향 각각의 평균 움직임 벡터크기이다. DF는 동적프레임이고, SF는 정적프레임을 나타낸다.

$$MV_{mag} = \frac{1}{N}\sum_{i=1}^{N}|MV_i| \tag{7}$$

$$\begin{aligned} & MV_{mag}(x_{mag}, y_{mag}) \\ & = \begin{cases} DF, & \text{if } x_{mag} > T_1 \text{ or } y_{mag} > T_1 \\ SF, & otherwise. \end{cases} \end{aligned} \tag{8}$$

3-2. 동일한 움직임 벡터들의 방향과 크기 판단

동적프레임과 정적프레임을 판단 후, 프레임 내의 움직임 벡터에서 그림 6과 같이 3×3마스크에 포함된 동일한 움직임 벡터들이 존재한다면, 3×3마스크에 포함된 움직임 벡터들을 정확한 움직임벡터로 판단을 하여 MVS단계를 수행하지 않는다.

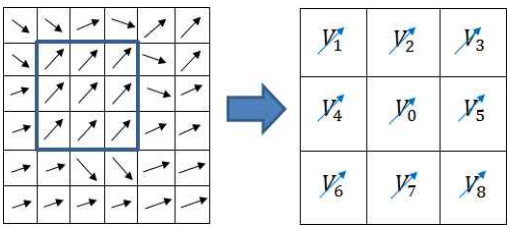

그림 6. 3×3마스크에 포함된 동일한 움직임 벡터일 경우

동일한 움직임 벡터들의 방향과 크기 판단 과정은 먼저, 3×3마스크에 포함된 움직임 벡터들을 그림 7과 같이 4개의 방향을 판단을 하고, 3×3마스크에 포함된 움직임 벡터들이 4개의 방향 중 동일한 방향에 다 포함이 안 될 경우, MVS 단계를 거치며, 동일한 방향에 다 포함이 될 경우에는 식 (9)를 통하여 크기를 구한다. 식 (10)에서 M이 임계치 T_2 보다 작으면 3×3마스크에 포함된 움직임 벡터들을 정확한 움직임 벡터로 판단하고, MVS단계를 수행하지 않는다. MV_m는 움직임 벡터들의 크기 차이의 합을 나타내며 x_m, y_m는 x, y방향 각각의 움직임 벡터들의 크기차이의 합이다.

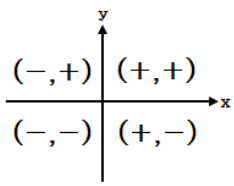

그림 7. 움직임 벡터 방향 판단

$$\begin{aligned} MV_m &= \sum_{i=1}^{8}|V_i - V_0| \\ MV_m &= (x_m, y_m) \\ M &= \sqrt{(x_m)^2 + (y_m)^2} \end{aligned} \tag{9}$$

$$flag = \begin{cases} 0, & \text{if } M < T_2 \\ skip, & otherwise. \end{cases} \tag{10}$$

4. 실험결과

실험에는 CIF(352×288) 크기의 8개 영상(*Bus, Coastguard, Container, Flower, Foreman, Mobile, Mother_Daughter, Table*)을 사용하였고, 총 50개의 짝수프레임을 생성하였다. 실험조건은 기본 블록 크기를 16×16으로 하였고, 탐색 범위는 ±8로 하였으며, T_1=0.6, T_2=0.3로 설정하였다. 화질비교 방법은 PSNR로 비교 하였다.

표 1에서 EBME 알고리듬 보다 제안한 알고리듬이 평균 0.008dB PSNR이 증가함을 볼 수 있고, 표 2에서 제안하는 알고리듬의 수행속도가 단축되었다.

표 1. PSNR 비교

Sequence	EBME(dB)	Proposed(dB)	Difference
Bus	25.873	**25.878**	0.005
Coastguard	**30.340**	30.318	-0.022
Container	39.290	**39.313**	0.023
Flower	30.812	30.812	0.000
Foreman	**33.652**	33.632	-0.020
Mobile	25.790	**26.186**	0.396
Mother_ Daughter	**40.805**	40.675	-0.130
Table	**30.185**	29.990	-0.195
Average	32.093	32.101	0.008

표 2. 수행시간

Sequence	EBME(sec)	Proposed(sec)	Gain(times)
Bus	23.857	**23.842**	1.001
Coastguard	22.764	**22.194**	1.026
Container	23.883	**12.709**	1.879
Flower	24.721	**24.320**	1.016
Foreman	22.345	**21.506**	1.039
Mobile	25.224	**21.584**	1.169
Mother_ Daughter	23.146	**12.976**	1.784
Table	25.532	**22.273**	1.146
Average	23.934	20.176	1.186

5. 결론

EBME알고리듬은 BME과정을 두 번수행하며, MVS단계에서 이상 벡터를 판단하고, 모든 이상 벡터가 수정될 때 까지 반복하기 때문에 높은 연산량을 요구한다. 그래서 제안한 알고리듬은 프레임 내의 x, y 방향의 각각의 평균 움직임 벡터 크기를 이용하여 동적프레임과 정적프레임을 판단한다. 동적프레임인 경우, 움직임 벡터 변화가 많아 EBME를 사용하여 정확한 움직임 벡터를 찾고, 정적프레임인 경우, 움직임 벡터 변화가 적기 때문에 연산량이 적은 BME를 이용하였다. 동적프레임과 정적프레임을 판단 후, 프레임 내의 여러 개의 동일한 움직임 벡터들이 존재한다면, 여러 개의 동일한 움직임 벡터들의 방향과 크기를 비교하여 MVS단계를 통과시켜 연산량을 줄였다. 실험 결과 EBME 알고리듬 보다 평균 0.008dB의 PSNR 향상을 가져왔으며, 수행시간은 평균 Gain이 1.186 으로 EBME 알고리듬 보다 수행시간이 감소하였다. *Coastguard, Mother_Daughter*에서는 동적프레임을 많이 포함하여 수행시간이 많이 감소하였다.

감사의 글

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터 육성지원사업의 연구결과로 수행되었음"
(IITP-2015-H8501-15-1005)

참고문헌

[1] S. H. Lee, O. Kwon, and R. H. Park, "Weighted-adaptive motion compensated frame rate up-conversion," *IEEE Trans. Consumer Electronics*, vol. CE-49, no. 3, pp. 485-492, Aug. 2003.

[2] S. J. Kang, K. R. Cho, and Y. H. Kim, "Motion Compensated Frame Rate Up-Conversion Using Extended Bilateral Motion Estimation," *IEEE Trans. Consumer Electronics*, vol. 53, no. 4, pp. 1759-1767, Nov. 2007.

[3] S. J. Kang, D. G. Yoo, S. K. Lee, and Y. H. Kim, "Multiframe-based Bilateral Motion Estimation with Emphasis on Stationary Caption Processing for Frame Rate Up-conversion," *IEEE Trans. Consumer Electronics*, vol. 54, no. 4, pp. 1830-1838, Nov. 2008.

[4] D. J. Park, and J. C. Jeong, "Adaptive Extended Bilateral Mtion Estimation Considering Block Type and Frame Motion Activity," *JBE*, vol. 18, no. 3, May 2013.

[5] V. Seferidis and M. Ghanbari, "General approach to block-matching motion estimation," *Journal of Optical Engineering*, vol. 32, pp. 1464-1474, July 1993.

[6] B. D. Choi, J. W. Han, C. S. Kim, and S. J. Ko, "Motion-Compensated Frame Interpolation Using Bilateral Motion Estimation and Adaptive Overlapped Block Motion Compensation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 17, no. 4, pp. 407-416, Apr. 2007.

[7] M. T. Orchard, G. J. Sullivan, "Overlapped Block Motion Compensation: An Estimation-Theoretic Approach," *IEEE Trans. Image Processing*, vol. 3, no. 5, pp. 693-699, Sep. 1994.