2008년도 대한전기학회 하계학술대회 논문집 2008. 7. 16 - 18

빠른 블록 정합 움직임 추정을 위해 보완된 다이아몬드 탐색 알고리즘

노형석*, 좌동경**, 홍석교***
아주대학교*, 아주대학교**, 아주대학교***

Modified Diamond Search Algorithm For Fast Block-Matching Motion Estimation

Hyung-Suk Lho*, Dongkyung Chwa**, Suk-Kyo Hong***
Ajou University*, Ajou University**, Ajou University***

Abstract - 이 논문에서는 영상 프레임사이의 움직임 벡터 연구에 기반을 둔 빠른 블록 정합 움직임 추정을 위해 보완된 다이아몬드 탐색 알고리즘을 제안한다. 실험 결과는 제안된 알고리즘이 기존의 다이아몬드 탐색 알고리즘 보다 더 적은 탐색으로 좀 더 빠르게 동작하는 것이 가능하다. 본 알고리즘의 특징은 움직임이 적은 블록에 대해서는 CDS(conjugate direction search) 알고리즘을 적용하여 탐색함으로써 탐색 점의 수를 줄일 수 있어 좀 더 빠르게 움직임 추정을 할 수 있다.

1. 서 론

영상 데이터의 양은 방대하다. 이러한 데이터를 가공하지 않고 보내는 것은 실시간으로 처리하기에 문제가 있다. 따라서 동영상 압축 분야에서는 중요한 정보를 압축해서 보내는 기술이 계속 발전해 오고 있다. 그 중에서도 가장 대표적인 방법이 서로 인접한 프레임들 간의 블록별 움직임 정보를 추정하여 이용하는 것이다. 즉, 그 움직임 정보를 움직임 벡터로 추정하여 보상 후 그 오차만 전송하여 데이터 전송량을 줄이게 된다. 블록 정합 탐색 알고리즘은 CCITT(현재 ITU-T), H.261, H.263, MPEG-1, MPEG-2, MPEG-4와 같은 다양한 비디오 코딩 표준에서 광범위하게 적용되고 있다[1]. 빠르고 정확한 블록 정합 움직임 추정 방법(BMME: Block-Matching Motion Estimation)은 처리시간을 줄이기 위해 연구된다. 탐색 윈도우 안에서 모든 점을 탐색하는 전역 탐색(FS) 알고리즘은 최적의 벡터를 찾지만 속도가 느리다는 단점이 있다. 이러한 문제점을 보완하기 위해 2차 대수 함수적 탐색(LOGS) 알고리즘[2], 크로스 탐색(CS) 알고리즘[3], CDS 알고리즘[4], block based gradient descent 탐색(BBGDS) 알고리즘[5] 등이 개발되었다. 이러한 블록 정합 움직임 추정 알고리즘들은 전역 탐색 알고리즘과 비교하여 탐색 점들의 수를 줄이고 전역 탐색 알고리즘에서 구해진 최적의 벡터와 거의 비슷한 벡터를 추정하기 위해 서로 다른 탐색 스타일과 탐색 전략을 사용한다[6].

이 논문에서 제안하는 알고리즘은 이전 프레임에서 움직임 벡터 크기가 큰 블록에 대해서는 DS 알고리즘을 적용하고 비교적 움직임 벡터의 크기가 작은(혹은 움직임 벡터의 크기가 0인) 블록에 대해서는 CDS 알고리즘을 적용하여 탐색 점의 수를 DS 알고리즘에 비해 줄이고 CDS 알고리즘에 비해 평균제곱에러(MSE: mean square error)를 줄이는 성능을 보이게 된다.

2. 본 론

2.1 관계된 논문 분석

블록 정합 움직임 추정 알고리즘에서 탐색 패턴의 모양과 크기는 탐색 속도뿐만 아니라 정확한 움직임 벡터를 추정하는 것을 결정한다. 탐색 패턴의 모양과 크기에 따라 움직임 벡터가 지역 최적 점에 빠질 우려가 있을 수도 있다. 두 프레임간의 에러 평면은 단조롭지 않기 때문에 지역 최적점이 여러개 있기 마련이다. 따라서 탐색 패턴의 모양과 크기를 정하는데 신중을 가하여야 한다. BBGDS 알고리즘과 같은 경우에는 탐색 크기가 3×3이므로 움직임이 큰 영상에 대해서는 지역 최적 점에 빠지게 된다. 반면에 9×9와 같이 너무 큰 탐색 크기는 움직임 벡터를 잘못 인도할 수 있다. 이에 반해 DS 알고리즘은 두 크기의 중간 크기로 지역 최적점에 빠지는 확률이 BBGDS에 비해 작다. CS 알고리즘은 탐색 패턴의 모양이 X자 모양이라서 상하좌우 움직임 추정에 취약함을 보인다. LOGS 알고리즘은 탐색 크기가 탐색 윈도우 사이즈에 따라 변하므로 성능이 탐색 윈도우 크기에 영향을 받는다. DS 알고리즘의 경우에는 다이아몬드 형태의 탐색 패턴의 모양을 가지고 탐색 패턴의 크기도 BBGDS 알고리즘보다 크기 때문에 FS에서 얻어진 최적의 움직임 벡터와 비슷한 움직임 벡터를 얻을 수 있지만, 초기 탐색 점이 최소 13개나 필요하다는 단점이 있다. 따라서 이 단점을 극복하기 위해 움직임 벡터의 크기가 작은 블록에서는 초기 탐색 점을 적게 사용하는 CDS 알고리즘을 적용한다는 것이 이 논문에서 제안하는 알고리즘이다.

2.2 기존의 탐색 알고리즘

본 논문에서 제안하는 알고리즘의 기초가 되는 DS 알고리즘과 CDS 알고리즘에 대해 방법을 간단하게 알아본 후 우리가 제안한 알고리즘을 설명한다.

2.2.1 DS 알고리즘

DS 알고리즘은 큰 다이아몬드 탐색 패턴(LDSP: large diamond search pattern), 작은 다이아몬드 탐색 패턴(SDSP: small diamond search pattern)을 가진다[6]. DS 알고리즘은 다음과 같이 요약할 수 있다.

Step 1) 초기 LDSP는 탐색윈도우의 중심이 되고 9개의 점에 대해 MSE를 구한다. 만약에 가장 작은 MSE가 가운데 점이면 Step 3으로 가고, 그렇지 않은 경우엔 Step 2로 간다.

Step 2) MSE가 가장 작았던 점을 중심으로 하고 각 점에 대해 LDSP를 구한다. 만약에 가장 작은 MSE가 가운데 점이면 Step 3으로 가고, 그렇지 않으면 이 단계를 반복한다.

Step 3) LDSP에서 SDSP로 탐색 방법을 바꾸고 MSE를 구한다. MSE 값이 가장 작은 점과 초기 중심점과의 차이로 움직임 벡터를 구한다.

2.2.2 CDS 알고리즘

CDS 알고리즘은 단계에 따라 방향을 바꾸면서 탐색하는 방법이다. 다시 말하면, 가로를 먼저 탐색하고 최적점을 찾으면 그 점에서부터 세로로 탐색하는 방법이다[4]. CDS 알고리즘은 다음과 같이 요약할 수 있다.

Step 1) 탐색 윈도우에서 중심점을 (x, y)라고 했을때, (x-1, y), (x, y), 그리고 (x+1, y) 에 대해 MSE를 구한다. 만약에 (x, y)가 최소값이면 Step 3으로 가고, 그렇지 않으면 Step 2로 간다.

Step 2) 중심점을 최소값이 나온 점으로 옮기고 (x-1, y), (x, y), 그리고 (x+1, y)에 대해 MSE를 구한다. 만약에 최소값이 (x, y)이면 Step 3으로 가고, 그렇지 않으면 이 단계를 반복한다.

Step 3) (x, y-1), (x, y), 그리고 (x, y+1)에 대해 MSE를 구한다. 만약에 최소값이 (x, y)이면 Step 5로 가고, 그렇지 않으면 Step 4로 간다.

Step 4) 중심점을 최소값이 나온 점으로 옮기고 (x, y-1), (x, y), 그리고 (x, y+1)에 대해 MSE를 구한다. 만약에 최소값이 (x, y)이면 Step 5로 가고, 그렇지 않으면 이 단계를 반복한다.

Step 5) 현재의 중심점과 초기 중심점과의 차이로 움직임 벡터를 구한다.

2.3 MDS(modified diamond search) 알고리즘

위에서 설명한 두 알고리즘을 융합하여 CDS 알고리즘보다 좀 더 좋은 움직임 벡터 추정을 하고, DS 알고리즘보다 적은 점을 탐색한다. 이 논문에서 제안된 알고리즘은 다음과 같이 요약할 수 있다.

Step 1) 모든 블록에 대해 DS 알고리즘을 적용한다.

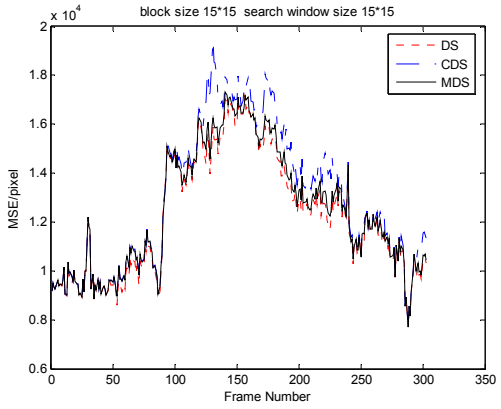
Step 2) 움직임 벡터의 크기가 경계값 이하이면 Step 3으로 가고, 그렇지 않으면 Step 4로 간다.

Step 3) 해당 블록에 대해 CDS 알고리즘을 적용한다. 움직임 벡터의 크기가 경계값 이하이면 이 단계를 반복하고, 그렇지 않으면 Step 4로 간다.

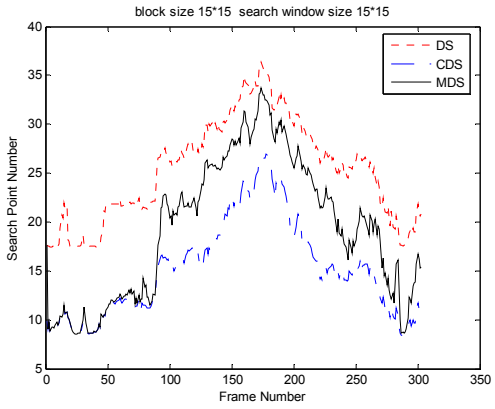
Step 4) 해당 블록에 대해 DS 알고리즘을 적용한다. 움직임 벡터의 크기가 경계값 이하이면 Step 3, 그렇지 않으면 이 단계를 반복한다.

이 알고리즘을 수식적으로 나타내면 다음과 같다.

$$d_{t+1}(n)=\begin{cases} DS\,Algorithm, & \text{if } d_t(n) > Threshold\,Value \\ CDS\,Algorithm, & otherwise \end{cases} \quad (1)$$



〈그림 1〉 자동차 주행 영상의 MSE 비교



〈그림 2〉 자동차 주행 영상의 평균 탐색 점의 수

〈표 1〉 픽셀당 평균 MSE

Algorithm	Frame Distance = 1	Frame Distance = 2
DS	12257.472	15942.497
CDS	12943.162	17316.959
MDS	12464.652	16353.148

〈표 2〉 움직임 벡터 당 탐색 점의 수의 평균

Algorithm	Frame Distance = 1	Frame Distance = 2
DS	25.296	28.976
CDS	15.165	18.044
MDS	19.131	23.829

여기서 d는 추정된 움직임 벡터를 의미하고, t는 프레임을 나타낸다. 그리고 n은 해당 프레임에서의 블록을 나타낸다. 예를 들어, $d_t(n)$는 t 프레임에서 n번째 블록의 움직임 벡터이다.

DS 알고리즘은 CDS 알고리즘에 비해 초기의 탐색점이 많고 탐색 패턴 모양이 크다. 따라서 CDS에 비해 큰 모션을 잡아내기에 더 좋기 때문에 이런 알고리즘을 구상하였다. 반면 움직임이 거의 없는 블록에 대해서는 CDS 알고리즘을 적용함으로써 DS 보다 빠른 속도를 확보할 수 있다는 장점이 있다.

2.4 제안된 알고리즘 수행에 관한 설명

첫째, DS 알고리즘과 CDS 알고리즘은 탐색하는 점이 제한이 되어 있지 않는 공통 특징을 가지고 있으므로, 본 알고리즘도 탐색하는 점에 제한이 되어 있지 않다.

둘째, DS 알고리즘은 수행도중 이전 단계에서 MSE를 구한 점을 다음 단계에서도 사용해야 하는 경우가 있는데, 그것을 계산 비용을 줄이기 위해 다시 계산하지 않고 재사용하였다[6]. CDS 알고리즘 같은 경우에도 마찬가지로 이전 단계와 다음 단계에서 겹치는 점을 재사용하였다.

셋째, LOGS 알고리즘이나 CS 알고리즘 같은 경우에는 추정되는 윈도우 벡터가 탐색 윈도우 크기를 벗어날 수 없지만, DS 알고리즘이나 CDS 알고리즘 같은 경우에는 탐색 윈도우를 벗어나게 구현할 수도 있다. 실제로 본 논문에서의 실험은 이러한 제약 없이 테스트 하였다.

2.5 실험 결과

우리가 수행한 실험에서는 블록크기를 16×16으로 고정하였다. 정확한 비교 자료를 만들기 위해 탐색 윈도우의 크기를 15×15로 하였다. 이것은 중심점으로부터 가로, 세로 모두 7의 크기로 한 것이다. 하지만 DS의 알고리즘과 CDS 알고리즘은 탐색 윈도우 크기의 제한이 없으므로 탐색 윈도우의 크기를 제약을 둘 필요가 전혀 없다. 프레임 간격은 1, 2로 실험하였다. 에러 값은 MSE 대신에 평균 절대값 차이(MAD)를 사용하였다. t번째 프레임 I_t에서 좌표(k,l)에 M×N 크기의 블록이 있다고 가정하고, 이 블록과 $t-1$번째 프레임 I_{t-1}에서 좌표 $(k+x,l+y)$에 위치한 블록과의 MAD 식은 다음과 같다.

$$MAD_{(k,l)}(x,y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}|I_t(k+i,l+j)-I_{t-1}(k+x+i,l+y+j)| \quad (2)$$

여기서 MAD는 평균 절대값 차이이고, M과 N는 각각 블록의 가로, 세로 크기를 의미하고, t는 프레임을 나타낸다. 식 (2)을 이용하여 알고리즘의 성능을 비교 할 수 있다.

테스트 영상은 자동차가 주행하는 것을 찍은 영상을 사용하였다. 테스트 프로그램은 [7]에서 참조하여 작성하였다. 테스트 프로그램은 Visual Studio 6.0을 이용하여 작성하였고, 논문에 소개된 LOGS, CS, BBGDS 알고리즘도 구현하였으나 지면상 결과를 생략하였다.

<그림 1>은 DS 알고리즘, CDS 알고리즘, 그리고 이 두 알고리즘을 융합한 알고리즘의 식 (2)을 통해 구한 MSE를 비교한 그래프이다. 거의 유사한 그래프를 띄지만 <표 1>에서 보는 것과 같이 MSE 값이 CDS 알고리즘에 비해 약 10 정도가 줄어든 것을 볼 수 있다. <그림 2>는 DS 알고리즘, CDS 알고리즘, 그리고 이 두 알고리즘을 융합한 MDS 알고리즘의 평균 탐색 점의 수를 비교한 그래프이다. <표 2>를 보면, DS 알고리즘을 표현한 그래프보다는 월등히 낮고, CDS 알고리즘과 거의 흡사하다는 것을 알 수 있다. 이처럼 두 알고리즘의 융합으로 더 CDS 알고리즘보다 더 정확한 움직임 벡터 추정과 DS 알고리즘보다 더 적은 탐색 점의 수를 사용하여 좋은 결과를 얻을 수 있다는 것을 실험을 통해서 보였다.

3. 결 론

이 논문에서는 하나의 빠른 움직임 벡터 추정을 위한 보완된 다이아몬드 탐색 알고리즘을 소개한다. 이 알고리즘은 DS 알고리즘을 기반으로 하여 그 알고리즘에 CDS 알고리즘을 융합함으로써 DS 알고리즘보다 더 좋은 성능을 낸다. 이전 블록에서의 움직임 벡터의 크기가 작으면 CDS 알고리즘을 사용하고, 크면 DS 알고리즘을 사용함으로써 속도를 향상시켰다.

[참 고 문 헌]

[1] K. R. Rao and J. J Hwang, “Techniques and Standards for Image, Video and Audio Coding”, Englewood Cliffs, NJ: Prentice Hall 1996.
[2] J. Jain and A. Jain, “Displacement measurement and its application in interframe image coding”, IEEE Transaction on Communication, vol. COMM-29, pp. 1799-1808, Dec., 1981.
[3] M. Ghanbari, “The cross-search algorithm for motion estimation”, IEEE Transaction on Communication, vol. 38, pp. 950-953, July, 1990.
[4] R. Srinivasan and K. R. Rao, “Predictive coding based on efficient motion estimation”, IEEE Transaction on Communication, vol. COMM-33, pp. 888-896, Aug., 1985.
[5] L. K. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding”, IEEE Transaction on Circuits System Video Technology, vol. 6, pp. 419-423, Aug., 1996.
[6] Shan Zhu and Kai-Kuang Ma, “A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation”, IEEE Transaction on Image Processing, vol. 9, No. 2, Feb., 2000.
[7] 황선규, “영상처리 프로그래밍 by Visual C++”, 한빛미디어(주), pp. 778-780, May, 2007.
[8] 이지홍, 고윤호, (주) 한백전자 기술연구소, “TI DSP TMS320C64xx를 이용한 디지털 영상처리”, ITC, pp. 223-229, Oct., 2006.