2000년 한국정보처리학회 추계 학술발표논문집 제7권 제2호

블록 정합 방법을 이용한 움직임 추정 : 분류 및 비교

최경주, 이일병
연세대학교 컴퓨터과학과
e-mail:kjcheoi@csai.yonsei.ac.kr

Block-Matching Motion Estimation : Classification and Comparison

Kyungjoo Cheoi, Yillbyung Lee
Dept of Computer Science, Yonsei University

요약

움직임 추정 및 보상을 위한 방법 중 가장 많이 사용하는 블록 정합 방법은 어떤 평가 함수와 탐색 방법(Search Procedure)을 사용했느냐에 따라 그 성능이 달라지게 된다. 본 논문에서는 평가 함수로써 평균 제곱 오차(Mean Squared Error; MSE), 평균 절대값 오차(Mean Absolute Error; MAE), 화소 차 분류(Pel Difference Classification; PDC)을, 탐색 방법으로써 전체 탐색 방법(Full Search Method ; FSM), 3단계 탐색 방법(Three Step Search ; TSS), 대각 탐색 방법(Cross Search Algorithm ;CSA)을 사용하여 이들의 성능을 각각 비교 분석하여 봄으로써 블록 정합 방법을 이용한 움직임 추정에 대한 전반적인 이해를 도모하고자 한다.

1. 서론

동영상의 움직임 추정 및 보상을 위한 방법에는 광류(Optical flow) 방법, 블록 기반 방법, 화소 반복(Pel-Recursive) 방법, 베이시안 방법 등 여러 가지가 있는데, 일반적으로 블록 정합 방법이 가장 많이 사용된다[1]. 블록 정합 방법은 탐색 방법과 평가 함수에 따라 그 성능이 크게 좌우되는데, 본 논문에서는 탐색 방법으로써 전체 탐색 방법, 3단계 탐색 방법, 대각 탐색 방법을, 평가 함수로써 평균 절대값 오차(Mean Absolute Error; MSE), 평균 제곱 오차(Mean Squared Error; MSE), 화소 차 분류(Pel Difference Classification; PDC)를 사용하여 이들의 성능을 각각 비교 분석하여 봄으로써 블록 정합 방법을 이용한 움직임 추정에 대한 전반적인 이해를 도모하고자 한다. 다음 2장에서는 전반적인 블록 정합 알고리즘을, 3장에서는 실험 결과를 설명하고, 4장에서 결론을 내린다.

2. 블록 정합 알고리즘

블록 정합 알고리즘은 변화가 일어난 블록이 움직임이 일어나기 이전 프레임의 어느 위치에 있는 블록과 가장 일치하는가를 추정하는 방법[1]으로, 움직임을 추정하거나 보상할 때 가장 많이 사용하는 방법이다. 이 방법의 기본적인 아이디어는 그림 1과 같다. 우선 현재 프레임을 일정한 블록 N으로 나눈 뒤에, 이전 프레임의 탐색 영역(search window)안에서 가장 근사한 블록을 찾고, 이 두 블록간의 거리를 계산하면 이것이 현재 프레임과 이전 프레임 사이에 이동한 움직임 벡터가 된다. 이 때 탐색은 그림 2에서 보이는 것과 같이 (N+2w × N+2w) 영역으로 제한되어 이루어지며, 가장 근사한 블록을 찾아내는 근거가 평가 함수이며, 이 평가 함수는 움직임 벡터에 영향을 미치는 요소로써 블록 정합 알고리즘의 성능과 밀접한 관계가 있다. 또한 블록 정합을 할 때 탐색 영역 안의 어떤 블록의 값에 대해 정해진 평가 함수를 적용하여 최소의 오차를 나타내는 블록의 위치를 움직임 벡터로 산정할 것이지를 결정하는 탐색 방법에 따라서도 성능이 달라지게 된다.

2.1 평가 함수

(1) Minimum Squared Error(MSE)

가장 기본적인 평가 함수로써 가장 정확한 움직임

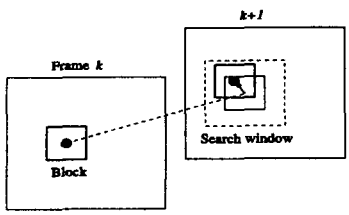

그림 1 블록 정합 알고리즘

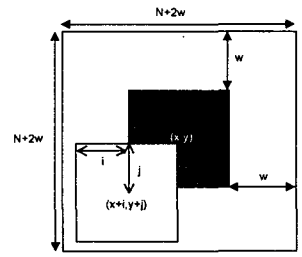

그림 2 탐색 윈도우

벡터를 찾을 수 있으나, 계산량이 많고 수행속도가 느리다[1]. MSE값이 작을수록 가장 정합이 잘 되는 것이므로 MSE를 최소화하는 (i, j)를 움직임 벡터로 채택한다. MSE는 식 (2)와 같이 표현될 수 있다.

$$MSE(i,j)=\frac{1}{N^2}\sum_x\sum_y\left[S_f(x,y)-S_{f-1}(x+i,y+j)\right]^2 \quad (1)$$

단, f : 현재 프레임, $f-1$: 이전 프레임, N : 블록의크기

(2) Minimum Absolute Error(MAE)

실제적으로 가장 많이 사용하는 평가 함수로서, 각 블록에 대응되는 화소들이 비교되고 그 차이가 더해지는데[1], 이는 식 (1)과 같이 표현될 수 있다. MSE를 사용하는 것보다 정확도는 떨어지지만, 계산 과정이 단순하고 같은 계산이 반복되므로 하드웨어 구현이 용이하고 VLSI 구성이 쉽다. MAE값이 작을수로 가장 정합이 잘 되는 것이므로 MAE를 최소화하는 (i, j)를 움직임 벡터로 채택한다.

$$MAE(i,j)=\frac{1}{N^2}\sum_x\sum_y\left|S_f(x,y)-S_{f-1}(x+i,y+j)\right| \quad (2)$$

(3) Pel Difference Clssification(PDC)

기존의 평가 함수와는 달리 pel 하나 하나의 움직임을 모두 고려하는 방법[2]으로서, 블록 안에 있는 각 화소가 정합되는 화소(matching pel)과 그렇지 않은 화소(mismatching pel)의 두 부류로 나뉘어 분류된다. 이때 미리 설정된 임계치가 사용되는데, 실험에서는 4를 설정하였다. 블록 안에 있는 정합되는 화소의 수가 가장 많이 있는 것이 가장 정합이 잘 되는 것이므로, PDC를 최대화하는 (i, j)를 움직임 벡터로 채택한다.

$$PDC(i,j)=\sum_x\sum_y T(x,y,i,j)$$

$$T(x,y,i,j)=\begin{cases}1 & \text{if } \left|S_f(x,y)-S_{f-1}(x+i,y+j)\right|\le t\\ 0 & \text{otherwise}\end{cases} \quad (3)$$

2.2 탐색 방법(Search Procedures)

가장 잘 일치되는 블록을 찾기 위해 각 화소 (i, j)에서의 가능한 모든 변위 벡터의 후보들에 대한 평가 함수를 최적화하여야 한다. 다음은 본 논문에서 사용한 3가지 탐색 방법이다.

(1) 전체 탐색 방법 : Full Search Method(FSM)

탐색 영역 안의 모든 블록의 값들에 대해 정해진 평가 함수를 적용하여 최소의 오차를 나타내는 블록의 위치를 움직임 벡터로 산정하는 방식으로 정확한 움직임 벡터를 찾는데는 가장 적합한 방법이지만 계산량이 많아 수행 속도가 느리다.

FSM의 알고리즘[1]은 다음과 같다.

[단계 1] 이전 프레임에서 기준 블록과 동일한 위치에 가상 블록을 정하고, 이 가상 블록 주변에 수평 및 수직 방향으로 일정한 변위(w)의 탐색 영역을 정한다.

[단계 2] 이 탐색 영역 내에서 기준 블록과 동일한 크기를 가지고 수직 및 수평 방향으로 한 화소만큼 변화되어 중첩되는 후보 블록 전체에 대해 기준 블록과 평가 함수를 비교하여 움직임 벡터를 찾는다.

(2) 3단계 탐색 방법 : Three Step Search(TSS)

FSM를 보완하기 위한 고속 탐색 방법[2]으로써 탐색 영역 안에 있는 모든 블록의 값을 탐색하는 것이 아니라 9개 블록의 값을 탐색하고, 평가 함수를 적용하여 그 중 가장 잘 정합이 되는 블록에서 다시 탐색을 시작하는데, 첫 번째 단계에서부터 이 때 w가 반으로 줄어들게 되며, w가 1이 될 때까지 위의 과정을 반복한다.

TSS의 자세한 알고리즘은 다음과 같다.

[단계 1] 정합하려는 블록의 중심화소를 (i, j)라 하면, p를 w/2로 설정한 후, (i-p, j-p), (i-p, j), (i-p, j+p), (i, j-p), (i, j), (i, j+p), (i+p, j-p), (i+p, j), (i+p, j+p)의 9개의 점을 중심 화소로 하는 블록에 대한 탐색을 수행한 후, 평가 함수를

적용하여 가장 잘 정합이 되는 블록을 찾는다.

[단계 2] 가장 잘 정합이 되는 블록의 중심 화소를 시작으로 하여 단계 1의 과정을 반복한다.

[단계 3] 위의 과정을 w가 1이 될 때까지 반복한다.

(3) 대각 탐색 방법:Cross Search Algorithm(CSA)

Logarithmic 탐색 방법[3]으로써, 탐색 영역 안에 있는 4개의 블록의 값을 탐색하게 되는데, 이 때 각 점의 끝점에서 (×)모양으로 대각 탐색하고, 평가 함수를 적용하여 그 중 가장 잘 정합되는 블록에서 다시 탐색을 시작하는데, 이때 w가 반으로 줄어들게 되며, 이 w가 1이 될 때까지 위의 과정을 반복한다. 마지막 단계에서는 각 끝점마다 (×)모양 또는 (+)모양으로 대각탐색을 수행한다.

CSA의 자세한 알고리즘은 다음과 같다.

[단계 1] 정합하려는 블록의 중심화소를 (i, j)이라 하면, p를 w/2로 설정한 후, (i-p, j-p), (i-p, j+p), (i+p, j-p), (i+p, j+p)의 4개의 점을 중심화소로 하는 블록에 대한 탐색을 수행하여 평가 함수 적용하여 가장 잘 정합이 되는 블록을 찾는다.

[단계 2] 가장 잘 정합이 되는 블록의 중심화소를 시작으로 하여 단계 1의 과정을 반복한다.

[단계 3] 위의 과정을 w가 1이 될 때까지 반복한다.

[단계 4] 윗 단계에서 가장 잘 정합이 되는 곳이 윗쪽의 오른쪽 편과 아래쪽의 왼쪽편에 있다면, (i, j)를 기준으로 했을 때 (i-1, j), (i, j-1), (i+1, n), (i, j+1)에 대해 p=1로 설정하여 탐색을 수행하여 가장 잘 정합이 되는 곳을 찾는다. 만일 윗 단계에서 가장 잘 정합되는 곳이 위쪽의 왼쪽 편과 아래쪽의 오른쪽편에 있다면, (i, j)를 기준으로 했을 때 (i-1, j), (i, j-1), (i+1, j), (i, j+1)에 대해 p=1로 설정하여 탐색을 수행하여 가장 잘 정합이 되는 곳을 찾는다.

3. 실험 내용 및 결과

윗절에서 언급한 3가지 탐색 방법에 대해 3가지 평가 함수를 각각 적용시켜 각각의 탐색점 수, 탐색 시간, 평가 함수 별 오차 및 정합된 화소 수를 분석하였는데, 실험 영상으로써 256×256의 trevor를 사용하였으며, 탐색 시간, 탐색점 수, 평가 함수 별 오차 및 정합된 화소 수를 성능 평가 기준으로 삼았다. 본 실험은 PentiumII 350Mhz에서 실험되었다.

(1) 각 탐색 방법에 대한 3가지 평가 함수의 탐색 시간 비교

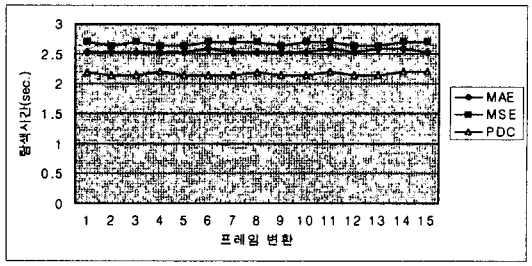

그림 6 FSM에서의 3가지 평가 함수 별 탐색 시간

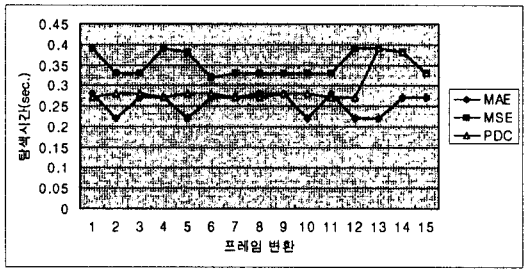

그림 7 TSS에서의 3가지 평가 함수 별 탐색 시간

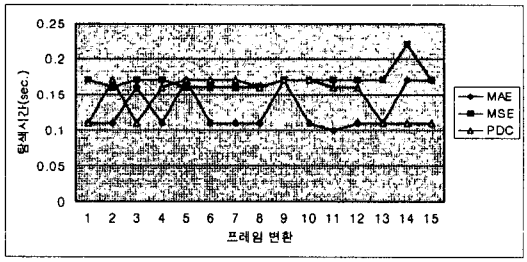

그림 8 CSA에서의 3가지 평가 함수 별 탐색 시간

(2) 3가지 평가 함수에 따른 탐색 방법의 탐색점 수 비교

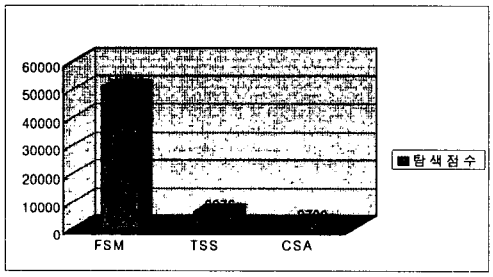

그림 9 3가지 평가방법의 탐색점 수 비교

(3) 각 평가 함수에 대한 3가지 탐색 방법의 탐색시간 비교

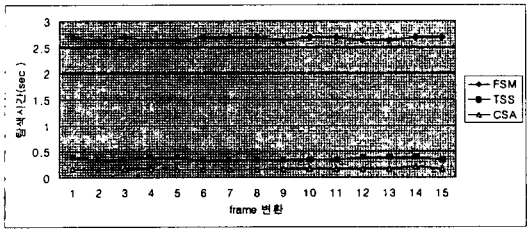

그림 10 MSE를 평가 함수로 사용

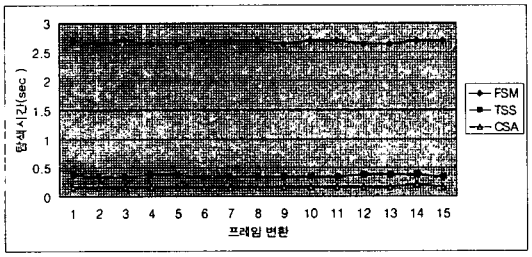

그림 11 MAE를 평가 함수로 사용

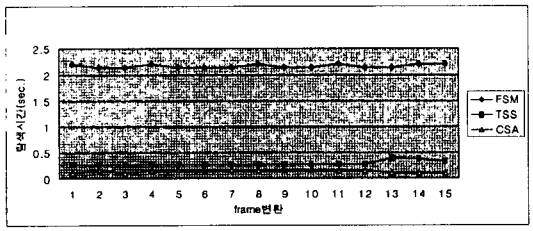

그림 12 PDC를 평가 함수로 사용

(4) 3가지 탐색 방법에 대한 오차 및 정합 화소 수

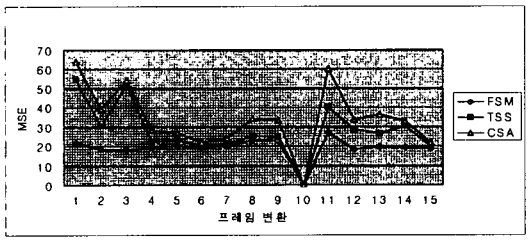

그림 13 3가지 탐색 방법에 대한 MSE

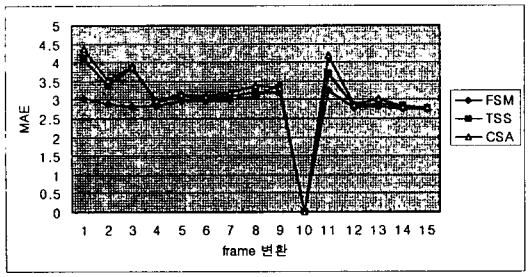

그림 14 3가지 탐색 방법에 대한 MAE

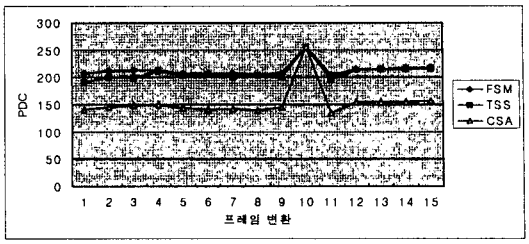

그림 15 3가지 탐색 방법에 대한 PDC

4. 결론

본 논문에서는 블록 정합 알고리즘을 사용한 움직임을 추정하기 위해 탐색 방법으로써 FSM, TSS, CSA를 사용하였고, 평가 함수로써 MAE, MSE, PDC를 사용하였다. 이들의 성능분석을 위해 위 3가지 탐색 방법 당 3가지 평가 함수를 모두 적용하여 실험을 하였으며, 그 결과를 탐색점 수, 탐색시간, 평가 함수 별 에러 및 matching pel 수에 따라 분석하였다. 실험 결과 3가지 탐색 방법 중 최소오차를 보이며 가장 matching pel수가 많은 것은 역시 모든 점을 탐색하는 FSM이였으며, TSS, CSA가 그 뒤를 이었지만, 탐색시간 면에서는 그 역순임을 알수 있었다. 평가 함수 별 성능을 분석한 결과 탐색시간은 PDC,MAE, MSE 순으로 빨랐다. 탐색점 수면에서는 CSA가 가장 적었고, TSS, FSM이 그 뒤를 이었다. 결국 탐색 방법은 CSA를 사용하고, 평가 함수로써 PDC를 사용하면 탐색 시간이 가장 빨라지고, 탐색점 수는 가장 적게 나올 수 있으나, 그 움직임 벡터를 구할 때 오차는 가장 커지게 된다. 따라서 움직임을 추정할 때 하나 하나의 움직임이 모두 중요한 변인이 되는 응용 분야에는 FSM에 PDC을, 탐색시간이 중요한 변인이 되는 응용 분야에는 CSA에 PDC를 사용하면 좋을 것이다.

참고문헌

[1] A.Murat Tekalp, Digital Video processing , Prentice-Hall,1995.

[2] H.Gharavi and M.Mills, Block-matching motion estimation algorithms : New results, *IEEE Trans.* Circ. And Syst., vol.37, pp.649-651, 1990.

[3] M.Ghanbari, The cross-search algorithm for motion estimation, *IEEE Trans. Commun.*, vol. 38, pp.950-953, 1990.

[4] V.seferidis and M.Ghanbari, General approach to block-matching motion estimation, *Optical Enginerring*, vol. 32, pp. 1464-1474, july 1993.