New Results on Exhaustive Search Algorithm for Motion Estimation using Adaptive Partial Distortion Search and Successive Elimination Algorithm

Man-Yau Chiu and Wan-Chi Siu*
Centre for Multimedia Signal Processing
Department of Electronic and Information Engineering
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
Tel: (852) 27666229 Fax: (852) 23624741 email: enwcsiu@polyu.edu.hk

Abstract— **Motion estimation is one of the most computational-intensive tasks in video compression. In order to reduce the amount of computation, various fast motion estimation algorithms have been developed. These fast algorithms can be classified into two groups. One is the lossy motion-estimation approach, which may have some degradation of predicted images, and the other is lossless, which means that the quality of the predicted images is exactly the same as those obtained by the conventional full search algorithm. The partial distortion search and successive elimination algorithm are two well-known techniques belonging to the second kind of approach. These two algorithms use different checking criteria to eliminate as much redundant computations as possible. Actually, the working principles of these methods are independent to each others and it is possible to apply them sequentially in order to achieve greater saving in computation. In this paper, we propose a new fast full-search motion estimation algorithm which can exploit fully the advantages of adaptive partial distortion search and successive elimination algorithm. Experimental results show that this proposed algorithm has an average speed-up of 13.31 as compared with the full search algorithm in terms of computational efficiency. This result is much better than the method simply combing both partial distortion search and successive elimination algorithm, which has an average computational speed-up of 10.60. For a practical realization using a PC, the average execution time speed-up for our algorithm is 4.96, which is also the best performance among all algorithms tested.**

I. Introduction

From various approaches for motion estimation, block-matching algorithm is very popular in the framework of generic coding (e.g. MPEG-1, MPEG-2, MPEG-4, H.261 and H.263) [1, 2]. Block-based matching algorithms are to look for the optimal motion vectors, which minimize the matching difference between candidate blocks in the reference frame and the target block in the current frame. A common comparison criterion for measuring the block difference is to calculate the sum of absolute differences (SAD) between an *NxN* block in the current frame and an *NxN* block in the reference frame as follows.

$$SAD=\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left|f_c(i,j)-f_r(i,j)\right| \qquad (1)$$

where $f_r(i,j)$ = sample of the reference block at location (i,j); $f_c(i,j)$ = sample of the current block at location (i,j) and $f_r(0,0)$, $f_c(0,0)$ are the top-left samples in the reference and current blocks respectively.

In order to find the best matching block in the reference frame, in theory, it is necessary to carry out a comparison of the current block with every possible region in the reference frame. This is usually too time-consuming because of the large number of operations required. Full search motion estimation calculates the comparison criterion (e.g. SAD as shown in (1)) at each possible location in the search window. The locations may be processed in a "raster" order starting from the top-left corner of the search window or in a "spiral" order starting from the centre position of the search window.

Owing to the computational complexity of the full search algorithm, many alternative "fast search" algorithms have been developed for motion estimation. These fast algorithms can be classified into two groups. One is the lossy motion-estimation algorithm [3] in which these algorithms can reduce the amount of computations as compared with the full search by just sampling a few points in the SAD map, and this may not be the global minimum. The other is lossless which gives exactly the same video quality as those image frames predicted by the conventional full search algorithm. For lossless motion estimation, typical examples are partial distortion search (PDS) [4, 5], and successive elimination algorithm (SEA) [6], in which they can achieve the same compression performance as the full search but with less computation as required in full search. Based on these methods, various derivatives have been published in literature, such as adaptive PDS [7, 8]; multi-level SEA [9, 10, 11]; combination of SEA and PDS [12] and etc.

In this paper, the partial distortion search algorithm (both conventional and adaptive approaches) and successive elimination algorithm are firstly described in Section II. Then, our proposed new fast full-search motion estimation algorithm combining the merits of these two algorithms is described in Section III. A comparison of computational efficiency and the motion estimation execution time of various lossless fast full-search algorithms and our proposed algorithm are described in Section IV. Finally, a conclusion is given in Section V.

II. FAST FULL-SEARCH ALGORITHMS: PARTIAL DISTORTION SEARCH AND SUCCESSIVE ELIMINATION ALGORITHM

A. Partial Distortion Search

1) Conventional Partial Distortion Search

Similar to the conventional full search (FS) algorithm, partial distortion search (PDS) also needs to evaluate the block differences (SAD) between the target block in the current frame with all the possible candidate blocks within the search window in the reference frame. But, PDS uses the partial sum of absolute differences between two blocks to eliminate the impossible candidates before the complete calculation of SAD. That is, if an intermediate sum of absolute differences is larger than the minimum value of SAD known at that time, the remaining computation for calculating the SAD can be abandoned.

Basically, after updating the current SAD at every sample location, we can check whether the accumulated SAD so far is larger than the current minimum SAD ($\mathrm{SAD}_{\min}$) or not. But, the checking itself also takes processing time, so it is not efficient to perform this test after adding the absolute value of the difference of each pair of samples. Instead, a good approach is to include this checking every time after a row of samples has been evaluated for updating the current SAD. This idea can be formulated by checking the k^{th} partial SAD (SAD_k) as shown in (2). Once the partial sum SAD_k exceeds the current $\mathrm{SAD}_{\min}$, we can terminate the remaining summation of the absolute differences and remove this impossible candidate motion vector (x, y).

$$SAD_k = \sum_{j=0}^{k-1}\sum_{i=0}^{N-1}\left|f_c(i,j) - f_r(i+x, j+y)\right| \quad k = 1,2,\ldots N. \tag{2}$$

2) Adaptive Partial Distortion Search using Clustered Pixel Matching Error Characteristic

As discussed previously, PDS reduces the computation complexity by terminating the SAD calculation early when it finds that a partial SAD is already greater than the current minimum SAD. Based on (2) let us define a generalized form of the partial SAD as follows:

$$SAD_p = \sum_{j=0}^{p-1}\sum_{i=0}^{N-1}\left|f_c(k_n,l_n) - f_r(k_n+x, l_n+y)\right| \quad p = 1,2,\ldots N. \tag{3}$$

where $\{(k_n, l_n) \mid n=0,\ldots, j*N+i,\ldots, N*N-1\}$ is an index set of all pixels in a MB. The index set determines the order of each pixel in the MB to be evaluated in calculating the SAD_p. If an index set can be formed in such a way that a pixel with greater matching error is accumulated to SAD_p earlier than other pixels, then SAD calculation can be terminated earlier. A possible way is to use an adaptive PDS using the characteristic of clustered pixel matching errors (CPME) [8].

Ideally, the index set should be formed in the descending order of pixel matching errors. For this adaptive PDS algorithm (denoted as CPME-PDS), it makes use of the phenomenon that pixel matching errors with similar magnitude tend to appear in clusters in natural video sequences. So, this algorithm at first tries to estimate the pixel matching errors for each pixel within the target MB, and then uses these predicted errors to form an index set for calculating the partial SAD. The major steps of this algorithm are outlined as follows.

Step 1: For each target MB under consideration, calculates the mean of pixel values in the initial candidate MB which is at the centre of the search window as follows:

$$m = \frac{1}{N*N}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left|f_r(i+u_{med}, j+v_{med})\right| \tag{4}$$

where (u_{med}, v_{med}) is the median predictor of three adjacent blocks, left, top and top right blocks to the current position as described in [13].

Step 2: Initializes an index set, $S'=\{(k'_n, l'_n) \mid n=0,\ldots, j*N+i,\ldots, N*N-1\}$

Step 3: Uses m as reference for calculating the predicted pixel matching error, $p_{exp}(n)$, at each pixel location in the target MB as follows:

$$p_{\exp}(n) = \left|f_c\left(k'_n, l'_n\right) - m\right| \qquad n = 0,\ldots,N*N-1 \tag{5}$$

Step 4: Re-arranges the order of pixels in S' in descending order of predicted pixel matching errors to obtain S and then uses S as an index set in calculating the partial SAD at every search location in the search window.

In order to make full use of the advantage of clustering characteristic, we may also sort consecutive number of pixels in a row instead of each individual pixel in forming the index set. In this way, consecutive number of pixels (4, 8 or 16) will be accumulated to partial SAD each time. Also, it can help to reduce the amount of non-uniform memory access in calculating the partial SAD. Such row-based adaptive PDS algorithm is denoted as CPME-PDS$_4$, CPME-PDS$_8$ and CPME-PDS$_{16}$ when consecutive 4, 8, 16 pixels are accumulated to partial SAD each time respectively.

B. Successive Elimination Algorithm

In conventional full search, it calculates the comparison criterion, e.g. SAD, at each possible location within the search window and then checks whether the SAD just obtained is smaller than the current $\mathrm{SAD}_{\min}$ or not. If the newly obtained SAD is smaller than the current $\mathrm{SAD}_{\min}$, $\mathrm{SAD}_{\min}$ and the best match location will be updated accordingly. Actually, SAD computation at a search location can be totally avoided if we can know in advance that its SAD is impossible to be smaller than the current $\mathrm{SAD}_{\min}$.

For successive elimination algorithm (SEA), the main concept is to establish a testing criterion in such a way that it can be used to indicate whether the candidate block in the reference frame will have the SAD value which must be greater than or equal to the current $\mathrm{SAD}_{\min}$ or not. So, if this testing criterion indicates that the candidate block under consideration will have the SAD value which must be greater than or equal to the current $\mathrm{SAD}_{\min}$, then this block will be rejected immediately. Otherwise, SAD computation will be performed on this candidate block for further checking. The testing criterion to determine whether the actual SAD computation is needed or not is established based on an inequality relationship between the target block in the current frame and the candidate block in the reference frame.

Assume that the search window is of size $(2M+1)\text{x}(2M+1)$; where x, y represents two components of a motion vector, $-M \le x, y \le M$ and we use the same notations as described previously. Then, the SAD between the target block in the current frame and the candidate block at search location (x, y) in the reference frame is as follows:

$$SAD(x,y) = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1}\left|f_c(i,j) - f_r(i+x, j+y)\right| \tag{6}$$

By using the inequality:

$$\left\|f_c(i,j)\right| - \left|f_r(i+x,j+y)\right\| \le \left|f_c(i,j) - f_r(i+x,j+y)\right| \tag{7}$$

It can be shown that the following relation holds [6]:

$$\left|R - M(x,y)\right| \le SAD(x,y) \tag{8}$$

where R and $M(x,y)$ represents the sum norms of the two blocks defined as follows:

$$R = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} f_c(i,j), \quad M(x,y) = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} f_r(i+x, j+y) \tag{9}$$

Assume we have obtained the current $\text{SAD}_{\min}$ $(=\text{SAD}(m,n))$ at location (m,n), then we need to compute the SAD at location (x,y) only if $\text{SAD}(x,y) < \text{SAD}_{\min}$. From (8), it is observed if $|R - M(x,y)| \ge \text{SAD}(m,n)$, it is impossible for $\text{SAD}(x,y)$ to be smaller than $\text{SAD}(m,n)$, and so this candidate block should be rejected immediately. Thus, the SAD computation is needed to be performed only if the following inequality holds:

$$\left|R - M(x,y)\right| < SAD(m,n) \tag{10}$$

Then, at each location within the search window, the testing criterion as described in (10) will be used to determine whether the SAD calculation is needed or not.

III. Proposed Hybrid Fast full-search algorithm combining Partial Distortion Search with Successive Elimination

Since SEA and PDS utilize different approaches to remove redundant computations involved in conventional full search, so it is possible to combine these two techniques together to further speed up the motion estimation process. On top of this, the idea of using an adaptive index set can be incorporated to further improve its performance. An analytical approach in deriving a new fast full-search algorithm is given as follows.

Our algorithm uses the following tools to reduce the redundant computations:

- pre-filtering of impossible candidates within the search window without calculating their actual SAD values;
- adaptive approach in calculating the SAD such that the big pixel differences can be accumulated as soon as possible; and
- early termination in calculating the SAD

and in particular, these tools can be realized by applying SEA, using an adaptive index set in calculating the partial SAD and applying PDS sequentially for each target MB under consideration. In forming the adaptive set, we can use CPME approach described previously. Furthermore, in order to reduce the effect of non-uniform memory access, we can use the row-based CPME approach to determine the adaptive index set for each target MB.

To summarize, the new proposed algorithm which combines SEA, and CPME-PDS or its row-based approach CPME-PDS_4 (denoted as SEA+CPME-PDS, SEA+CPME-PDS_4) consists of three major steps.

Step 1: At each search location, check whether it is a potential candidate or not by applying the SEA algorithm.

Step 2: If it is a potential candidate, use the adaptive set to calculate the partial SAD.

Step 3: Check the PDS termination criterion for each partial SAD obtained and terminate the SAD calculation once the criterion is matched.

IV. Experimental results and discussion

Totally, six frame-based video sequences were tested, and the size of two of them (Car phone and Trevor) is 176x144 pixels and the four of them (Stefan, Table-tennis, Coastguard and Football) is 352x240 pixels.

To evaluate the performance of the SEA+CPME-PDS and SEA+CPME-PDS_4, we have tested these video sequences with several algorithms and they are: the conventional full search algorithm (FSA), the SEA, the conventional PDS, the SEA+PDS, the CPME-PDS and its row-based sorting version CPME-PDS_4. The outward spiral scanning was applied to all these algorithms. In addition, a median predictor was used as an initial searching centre to exploit the correlation in the motion field for all tested algorithms.

The computational efficiencies of the algorithms were assessed in terms of the number of operations and the execution time required for the searching. In our implementation, quick sort was used as the sorting approach for CPME-PDS and CPME-PDS_4 in forming the index set to calculate the partial SAD. TABLE I shows a comparison of the average number of total operations per MB of the tested algorithms. The results show that the SEA+CPME-PDS can successfully improve the computational efficiency of the conventional PDS and SEA algorithms. In terms of speed-up ratios, the SEA+CPME-PDS can achieve a speed-up ranging between 4.85 and 34.48 of the FSA or an average speed-up of 13.31 of the FSA. TABLE II shows a comparison of the execution time per frame of the tested algorithms. In particular, for CPME-PDS, CPME-PDS_4, SEA+CPME-PDS and SEA+CPME-PDS_4, the execution time already includes the time required for sorting process to form the index set for each target MB. Again, the results show that the SEA+CPME-PDS can successfully reduce the execution time required as compared with the case when only conventional PDS or SEA algorithm is employed. In terms of speed-up ratios, the SEA+CPME-PDS can achieve a speed-up ranging between 2.8 and 6.7 of the FSA or an average speed-up of 4.23 of the FSA.

Furthermore, it is found that the SEA+CPME-PDS_4 can achieve a higher speed-up in execution time as compared with that of the SEA+CPME-PDS. For SEA+CPME-PDS$_4$, it can achieve an average speed-up of 4.96 of the FSA in terms of the execution time which is also the best among all the algorithms tested. Comparing the performance of SEA+PDS and SEA+CPME-PDS, the SEA+CPME-PDS requires fewer number of operations per MB but in general the SEA+CPME-PDS requires longer execution time (except for "Car phone" sequence) as compared with that of the SEA+PDS. It is because though the adaptive CPME-PDS approach can successfully reject the impossible candidates at an earlier stage, it requires random access of pixel values in computing the partial sum. Such non-uniform memory access may result in longer overall execution time. To compromise the effect of non-uniform memory access of CPME-PDS, its row-based version, CPME-PDS$_4$, is employed instead and from TABLE I and TABLE II, the SEA+CPME-PDS$_4$ requires fewer number of operations per MB and in general requires shorter execution time as compared with that of the SEA+PDS.

TABLE I. AVERAGE NUMBER OF TOTAL OPERATIONS PER MB OF THE TESTED ALGORITHMS IN A SEARCH WINDOW WITH A SEARCH RANGE EQUAL TO 15 (I.E. $-15 \le u, v \le 15$)

Tested Algorithms \ **Video Sequences**		**Car phone**	**Trevor**	**Stefan**	**Table-tennis**	**Coast-guard**	**Foot-ball**
FSA	No. of Opr.	738048	738048	738048	738048	738048	738048
	Speed-up	1.00	1.00	1.00	1.00	1.00	1.00
SEA	No. of Opr.	116542	86114	308250	328831	356370	355991
	Speed-up	6.33	8.57	2.39	2.24	2.07	2.07
PDS	No. of Opr.	158078	110638	241830	219509	246280	299882
	Speed-up	4.67	6.67	3.05	3.36	3.00	2.46
CPME-PDS	No. of Opr.	116173	81075	169451	148893	179450	232582
	Speed-up	6.35	9.10	4.36	4.96	4.11	3.17
CPME-PDS$_4$	No. of Opr.	119355	88555	179861	165826	187950	242991
	Speed-up	6.18	8.33	4.10	4.45	3.93	3.04
SEA+ PDS	No. of Opr.	48113	25736	134725	129517	162125	191212
	Speed-up	15.34	28.68	5.48	5.70	4.55	3.86
SEA+ CPME-PDS	No. of Opr.	**39120**	**21404**	**99660**	**91056**	**120416**	**152216**
	Speed-up	**18.87**	**34.48**	**7.41**	**8.11**	**6.13**	**4.85**
SEA+ CPME-PDS$_4$	No. of Opr.	*40242*	*22133*	*104994*	*102470*	*126001*	*158389*
	Speed-up	*18.34*	*33.35*	*7.03*	*7.20*	*5.86*	*4.66*

Note: Figures which are in **bold** are the best results; while figures which are in *italic* are the second best results. No. of Opr. = Number of Operations

TABLE II. THE EXECUTION TIME (SECONDS) PER FRAME OF THE TESTED ALGORITHMS IN A SEARCH WINDOW WITH A SEARCH RANGE EQUAL TO 15 (I.E. $-15 \le u, v \le 15$)

Tested Algorithms \ **Video Sequences**		**Car phone**	**Trevor**	**Stefan**	**Table-tennis**	**Coast-guard**	**Foot-ball**
FSA	Exec. Time	0.1698	0.1746	0.5617	0.5621	0.5622	0.5602
	Speed-up	1.00	1.00	1.00	1.00	1.00	1.00
SEA	Exec. Time	0.0353	0.0315	0.2452	0.2626	0.2795	0.2795
	Speed-up	4.81	5.55	2.29	2.14	2.01	2.00
PDS	Exec. Time	0.0575	0.0483	0.2384	0.2253	0.2415	0.2779
	Speed-up	2.95	3.61	2.36	2.49	2.33	2.02
CPME-PDS	Exec. Time	0.0584	0.0495	0.2341	0.2174	0.2417	0.2859
	Speed-up	2.90	3.53	2.40	2.59	2.33	1.96
CPME-PDS$_4$	Exec. Time	0.0458	0.0404	0.1937	0.1832	0.2009	0.2419
	Speed-up	3.71	4.33	2.90	3.07	2.80	2.32
SEA+ PDS	Exec. Time	0.033	**0.0182**	*0.1405*	*0.1387*	*0.1625*	*0.1818*
	Speed-up	5.15	**9.57**	*3.99*	*4.05*	*3.46*	*3.08*
SEA+ CPME-PDS	Exec. Time	*0.0305*	0.0261	0.1544	0.1488	0.1742	0.2001
	Speed-up	*5.56*	6.70	3.64	3.78	3.23	2.80
SEA+ CPME-PDS$_4$	Exec. Time	**0.0264**	*0.0224*	**0.1328**	**0.1309**	**0.1493**	**0.1746**
	Speed-up	**6.43**	*7.81*	**4.23**	**4.29**	**3.77**	**3.21**

Note: Figures which are in **bold** are the best results; while figures which are in *italic* are the second best results. Exec. Time = Execution time

V. CONCLUSION

In this paper, we have proposed a new fast full-search motion estimation algorithm which utilizes different tools in eliminating as much redundant operations as possible. This algorithm can be considered as a scheme which makes use of a pre-filtering step to get rid of impossible candidates sooner within the search window without calculating the actual SAD values, use an adaptive index set in calculating the SAD such that big pixel differences can be accumulated as early as possible, as well as an early termination criterion in calculating the SAD. Experimental results shows that the new proposed algorithm SEA+CPME-PDS$_4$ can achieve a better overall performance, both in terms of the average number of operations and the execution time required, among all the tested algorithms.

It is also interesting to note that this paper also outlines a framework in utilizing various tools together in order to achieve the best performance in motion estimation. In principle, whenever any improvement can be achieved in anyone of these tools, the improved version of this tool can be used in this framework to replace the old one since the tools used are independent to each others.

VI. REFERENCES

[1] *Video Coding for Low Bit Rate Communication*, ITU-T Standard H.263, March 1996.

[2] *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC Std. 13818-2, 1996.

[3] Jo Yew Tham, Surendra Ranganath, Maitreya Ranganath, and Ashraf Ali Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp.369-377, August 1998.

[4] ITU-T Recommendation H.263 Software Implementation, Digital Video Coding Group, Telenor R&D, 1995.

[5] S. Eckart and C. Fogg, "ISO/IEC MPEG-2 software video codec," *Proc. SPIE*, Vol. 2419, pp. 100–118, 1995.

[6] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, no. 1, pp.105-107, January 1995.

[7] Jong-Nam Kim and Tae-Sun Choi, "A fast full-search motion-estimation algorithm using representative pixels and adaptive matching scan," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 10, no. 7, pp.1040-1048, October 2000.

[8] Ko-Cheung Hui, Wan-Chi Siu, and Yui-Lam Chan, "New adaptive partial distortion search using clustered pixel matching error characteristic," *IEEE Transactions on Image Processing*, vol. 14, no. 5, pp.597-607, May 2005.

[9] X.Q. Gao, C.J. Duanmu, and C.R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp.501-505, March 2000.

[10] Ce Zhu, Wei-Song Qi, and Wee Ser, "Predictive fine granularity successive elimination for fast optimal block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 14, no. 2, pp.213-221, February 2005.

[11] Byung Cheol Song, Kang-Wook Chun, and Jong Beom Ra, "A rate-constrained fast full-search algorithm based on block sum pyramid," *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp.308-311, March 2005.

[12] Huan-Sheng Wang and R.M. Mersereau, "Fast algorithms for the estimation of motion vectors," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp.435-438, March 1999.

[13] *Information Technology—Coding of Audio-Visual Objects: Visual*, ISO/IEC Standard 14496-2, 1998.