Picture Coding Symposium
Portland, OR, April 21-23 1999

Correlation Based Search Algorithms for Motion Estimation[†]

Mohamed Alkanhal, Deepak Turaga and Tsuhan Chen
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: {alkanhal, dturaga, tsuhan}@ece.cmu.edu

Abstract

The measure of the 'goodness' of a motion estimation algorithm is governed by it's speed, the quality of motion compensation it provides, and the size of the resulting bitstream. Hence, algorithms should be evaluated based on this 'speed-quality-bitrate' tradeoff. Previously introduced fast motion estimation algorithms focus mainly on the speed vs. quality of motion compensation. In this paper, we introduce several new algorithms and evaluate them based on all three parameters. All these new algorithms exploit spatial correlation of motion vectors. These algorithms include a MAD (Mean Absolute Distortion) based spiral search, an Adaptive Window Size algorithm and two Majority Voting schemes. The algorithms are evaluated on several test sequences in the H.263 framework and the results obtained are very encouraging.

1. Introduction

Block-based motion estimation forms the base of all video coding schemes. It involves finding a candidate block in a specified search area, in the previous frame that is most similar to the current block in the current frame. The exhaustive or full search (ES) block matching algorithm does a search over the entire search space to come up with the optimal solution in terms of quality of motion compensation. This is, measured in terms of the MAD (sum of absolute difference between corresponding pixels of the current block and the test block). The ES is computationally very intensive and several sub-optimal search algorithms have been developed. These reduce computational complexity while trying to approximate the optimal solution. The choice of an algorithm for motion estimation is governed by the 'speed-quality-bitrate' tradeoff. A good algorithm is one that has a low computational complexity, provides a high quality of motion compensation and also ensures that the bitstream is as small as possible. The focus of this paper is on this tradeoff and several new algorithms are introduced to exploit such tradeoff. Previous results in literature do not always consider all the three factors.

1.1 Existing Techniques

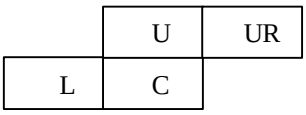
There are a large number of different sub-optimal search algorithms for block based motion estimation. These include: Three Step Search (TSS)[1], Four Step Search (FSS)[2], Spiral Search [3], Two-D Logarithmic Search [4], Orthogonal Search [5], Cross Search [6], One-at-Time Search [7] etc. Among these algorithms, the TSS and the FSS perform best in terms of coding speed and resulting quality of motion compensation.

These algorithms do not directly exploit spatial correlation between neighboring motion vectors. The algorithms introduced in this paper use such spatial correlation to obtain better performance. The use of such spatial correlation helps in reducing the search space. It also helps in reducing the bit rate for motion vectors because standards like H.263 use differentially coded motion vectors. The algorithms introduced in this paper try to achieve better performance in terms of all three parameters in the speed-quality-bitrate tradeoff.

The first group of algorithms introduced use spatial correlation for both selecting the starting point for the search and restricting the search space around the starting point. Two different ways of altering the search space dynamically are introduced. These are the MAD-based spiral search and the Adaptive Window size search. The second group includes two majority voting schemes. These algorithms use spatial correlation information to decide between FSS and TSS for each block in the frame.

The spatial correlation information for all of the above algorithms is picked from the same configuration of neighbors that the H.263 standard uses for differentially coding the motion vectors. This is shown in the following figure.

[†] Work supported in part by Institute for Information Industry.



C: Current Block U: Upper Block
L: Left Block UR: Upper Right Block

Fig.1 Motion Vector Prediction in H.263

These neighboring blocks are called the predictor blocks. The median of these three motion vectors is used to predict the motion vector of C.

2 The Techniques

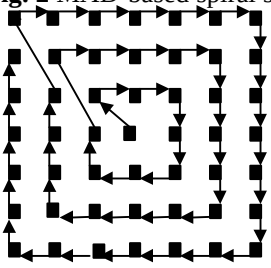
In this section, new techniques are proposed and discussed. These techniques are MAD-based spiral search, Adaptive Window Size, majority voting and extended majority voting.

2.1 MAD-Based Spiral Search (M-BSS)

Most image sequences have smooth motion and high spatial correlation. It is quite likely that the motion vector of a block is close to the motion vectors of its neighbors. Hence, the search window center can be predicted using the motion vectors of the predictor blocks. If the spatial correlation information is right, the best match should be around this predicted center.

The M-BSS uses the motion vectors of the predictor blocks to get a predicted search window center. It then uses the MAD values of these predictor blocks to achieve a variable window size. The MAD is computed starting at the center of the search window, and moving outward spirally (Fig. 2). This process is stopped once the MAD falls under a threshold value. This threshold is clearly the parameter that controls the size of the window and it is obtained from the MAD values of the predictor blocks. We choose the threshold to be the same as the median of the MAD values of the predictor blocks. A median operation helps suppress the effect of the MAD value of any uncorrelated block in the neighborhood.

Fig. 2 MAD-based spiral search.



This algorithm suffers from the drawback that errors in the threshold value can propagate through the sequence. It is possible that the MAD threshold computed is much larger than the achievable MAD. This leads to accumulation of errors over time. Hence, the threshold must be updated frequently, using a full search. We tried doing an update every first block of the frame, every first row of the frame and finally every n blocks. The last update method outperformed the other two in terms of quality without sacrificing too much on speed

One parameter that also needs to be defined is the largest size of the variable window. A window with size (± 16, ± 16) and a window with size (±3, ±3) were tried. The tradeoff between quality of motion compensation and speed is indicated in the results.

2.2 Adaptive Window Size Search (AWSS)

This algorithm also uses the motion vectors of the predictor blocks to predict the center of the search space. It, however, also derives the search window size from the motion vectors of the predictor blocks. Motion vectors are classified as small, medium and large. The predictor blocks with small motion vectors vote for a small window size, while those with medium size motion vectors vote for a medium search window and so on. Small medium and large are defined as follows:

a) Small: $|mvx|, |mvy| \le 4$;Window = 4

b) Large: $|mvx| or |mvy| > 8$;Window = 16

c) Medium: all other *mvx, mvy;* Window = 8

The window size with a majority of votes is picked. In the absence of a clear winner, the medium size window is selected. A full search is done in the region of the intersection of this window and the original search space.

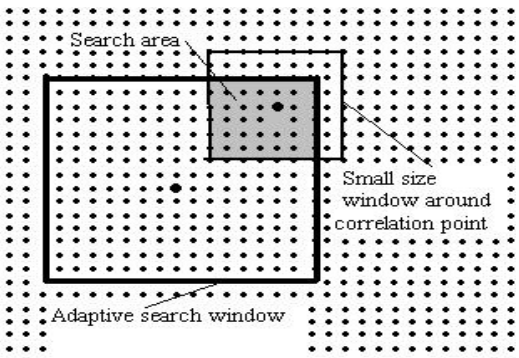


Fig. 3 Search area: Intersection of adaptive search window around block center and small window around predicted center.

This algorithm also suffers from the accumulation of errors, and the window might get trapped to have a small size. Hence, a dynamic update for the window size is needed. For this paper, a full search was implemented for the first block of every frame. This update gave satisfactory results both in terms of computation and quality of motion compensation. Future work might focus on the selection of the update procedure. Better performance may also be achieved by combining the M-BSS with this algorithm.

2.4 Majority Voting Algorithm (MVA)

The TSS and the FSS perform really well for different kinds of sequences. While the TSS is better for large motion, the FSS is more efficient for small motion. We need to exploit these good features in order to get a good performance over many different sequences. This algorithm decides for every block whether it wants to do a TSS or a FSS, depending on whether the current scene is high motion or low motion. Each of the predictor blocks votes for either the TSS or the FSS. The vote is cast depending on the motion vector of the voting block. If the voting block has a large motion vector, then it will vote for the TSS, otherwise it will vote for the FSS. A majority of the three votes is accepted and the corresponding winning algorithm is picked. A large motion vector is one that is at least a certain threshold distance away from the center. This threshold can be varied. Also, other criteria for the voting scheme are also being considered.

2.5 Extended Majority Voting (EMV)

An extension to the majority voting scheme has also been implemented. In this modification to the voting scheme, not only does the algorithm pick between the FSS and the TSS, but also the starting point of the search is moved based on spatial correlation. This exploits spatial correlation more effectively. Both the TSS and FSS are center biased and hence perform more efficiently if the predicted center is close to the best match. The moving of the starting point can also help avoid getting trapped in local minima.

3.0 Simulation Results and Discussions

Simulations have been performed on four video sequences. Children and Weather are QCIF streams at 10Hz that have 10 frames each. Coastguard is QCIF with 300 frames at 30Hz, while Stefan is a CIF sequence with 300 frames at the 30Hz. All algorithms are compared in terms of the Average MAD, the CPU time and the size of the resulting bitstream. The bitstream includes both the Motion Vector Difference (MVD) bits and the error bits. The first frame is not included in the total bits calculations.

The conversion of the error image and motion vectors to bitstream has all been done in conformance with the H.263 standard. While deriving the motion vectors for a particular frame, the best match is located in the original previous frame and not a coded version of it. Also, the error bits are based on coding of the difference between the predicted frames using original previous frame and the current frame.

All these algorithms are evaluated based on the following criteria:

a) The CPU time taken by the algorithm, as measured on a Pentium 200 MHz processor.
b) The average MAD for different sequences (as a measure of the quality of motion compensation).
c) The total number of bits to code the sequence (the error bits and the motion vectors).

Table 1 shows the results for the MAD based spiral search algorithm. The algorithm was implemented with the largest window size set at ± 16. Results for an update done every n = 10 blocks are included. It can be seen that this performs better for sequences with correlated motion like Coastguard.

Table 1

Sequence	MAD	CPU Time	**Error bits**	**MVD bits**	**Total bits**
Children	7.88	16.24	828K	3,307	832K
Weather	2.57	20.08	533K	1,687	535K
Coastguard	3.63	449.05	11.9M	123K	12M
Stefan	9.13	1604.36	140M	894K	141M

Table 2 shows results of the MAD-based spiral search with a small window size. In order for the comparison to be valid, it is compared with an exhaustive search over the small window. It can be seen that the MAD based spiral search is much better in terms of speed, while the quality of motion compensation is worse (especially for uncorrelated sequences like Children).

Table 2

Sequence	Algo.	MAD	CPU Time	**Error bits**	**MVD bits**	**Total bits**
Children	ES	8.75	1.36	831K	2421	833K

	M-BSS	9.58	1.01	863K	2828	866K
Weather	ES	2.67	1.41	533K	1629	535K
	M-BSS	2.73	1.25	535K	1560	536K
Coastguard	ES	3.52	44.3	11.7M	130K	11.8M
	M-BSS	3.66	26.2	11.9M	122K	12.1M
Stefan	ES	12.47	237.6	178M	898K	179M
	M-BSS	13.57	106.1	176M	916K	177M

Table 3 includes the results for the adaptive window size algorithm. The performance of this algorithm is between the small window full search and the small window MAD based spiral search. It is however, much worse than the MAD based spiral search in terms of computation for the Coastguard sequence.

Table 3

Sequence	MAD	CPU Time	**Error bits**	**MVD bits**	**Total bits**
Children	8.78	1.23	831K	2391	833K
Weather	2.70	1.32	534K	1536	536K
Coastguard	3.58	38.99	11.8M	129K	11.9M
Stefan	14.19	146.9	198M	636K	199M

Table 4 shows results of implementing the majority voting and the extended majority voting algorithms. It can be seen that both these algorithms perform better in terms of MAD for sequences with correlated motion such as Weather and Coastguard. The EMV in fact does better in term of speed also. Both of the algorithms do better than the TSS in terms of the number of bits. The EMV does better than the FSS in terms of the number of bits for Weather and does around the same number for Coastguard. In sequences with large motion, such as Children, both these algorithms perform better than the FSS in terms of speed and MAD. They however, perform worse than the TSS. For sequences with uncorrelated motion, such as Stefan, these sequences do not perform as well. They, however, do better than the FSS, which seems to get trapped in local minima, thereby giving a high speed, but poor MAD. Overall, it can be seen that both these algorithms try to exploit the nice features of the TSS and the FSS to give a good performance for sequences with correlated motion.

Table 4

Sequence	Algo.	MAD	CPU Time	**Error bits**	**MVD bits**	**Total bits**
Children	FSS	8.64	1.04	825K	3424	828K
	TSS	8.41	1.73	821K	3859	825K
	MVA	8.47	1.11	825K	3423	828K
	EMV	8.44	0.84	824K	3422	828K
Weather	FSS	2.83	1.0	532K	2372	534K
	TSS	2.78	1.8	534K	2429	536K
	MVA	2.59	1.1	532K	2374	534K
	EMV	2.56	1.0	532K	2334	534K
Coastguard	FSS	3.67	41.8	11.7M	78K	11.8M
	TSS	3.76	62.5	11.9M	83K	12.0M
	MVA	3.58	41.3	11.8M	80K	11.9M
	EMV	3.56	41.1	11.8M	79K	11.9M
Stefan	FSS	11.20	197.9	165M	839K	166M
	TSS	10.42	258.7	159M	943K	160M
	MVA	10.4	217.6	157M	891K	158M
	EMV	10.9	229.8	170M	1.0M	171M

4.0 Conclusion

In this paper, a number of new algorithms were proposed and implemented. All of these algorithms use spatial correlation information. These algorithms were designed taking into account the "speed-quality-bitrate" tradeoff. The best algorithm is the Extended Majority Voting. It is very attractive for most sequences as it uses the TSS, the FSS and spatial information most effectively to perform well in terms of all three of the "speed-quality-bitrate" tradeoff.

References

[1] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation", IEEE Trans. on Circuits and Systems for Video Technology, vol. 4, no. 4, pp. 438-442, Aug. 1994.

[2] L. Po and W. Ma, " A Novel Four-Step Search Algorithm for Fast Block Motion Estimation", IEEE Trans. on Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 313-317, June 1996.

[3] T. Zahariadis and D. Kalivas, "A Spiral Search Algorithm for Fast Estimation of Block Motion Vectors", Signal Processing VIII, theories and applications, proceedings of the EUSIPCO 96, Eighth European Signal Processing Conference p.3 vol. Lxiii + 2144, pp.1079-1082, vol.2.

[4] J. Jain and A. Jain, "Displacement Measurement and its application in interframe image coding", IEEE Trans. on Communications, vol. COM-29, pp. 1799-1808, Dec. 1981.

[5] A. Puri, H. Hang and D. Schilling, "An efficient block-matching algorithm for motion compensated coding", Proceedings IEEE ICASSP, pp. 25.4.1 - 24.4.4, 1987.

[6] M. Ghanbari, " The Cross-Search Algorithm for Motion Estimation", IEEE Trans. on Communications, vol. COM-38, no. 7, pp. 950-953, July 1990.

[7] R. Sinivasan, K. Rao, "Predictive coding based on efficient motion estimation", International Conference on Communications, Part 1, pp. 521-526, 1988, Amsterdam.