Module 7

VIDEO CODING AND MOTION ESTIMATION

Lesson 22

Other fast search motion estimation algorithms

At the end of this lesson, the students should be able to:

1. Provide an overview of the following fast search motion estimation algorithms.

   - Cross search
   - Three step-search
   - New three-step search
   - Diamond search
   - Gradient –descent search

2. Determine the computation complexity of the above algorithms

3. Compare between FSBM and fast search motion estimation algorithms.

22.0 Introduction

In lesson 21 we had presented the full search block motion (FSBM) estimation algorithm, which necessitated development of fast and efficient non exhaustive search methodologies for real time implementations. We had seen that such fast and efficient search technologies are based on a heuristic assumption that as we move away from the point of distortion minima, the distortion value monotonically increases. In lesson 21, we had presented only one such efficient search, i.e., 2-D Logarithmic search algorithm. More search algorithms with better computational complex than 2-D logarithmic search and more efficient in motion compensation are presented in this lesson.

22.1 Cross search algorithm (CSA)

The cross search algorithm follows the ideas similar to that of logarithmic search, but some differences lead to fewer computational search points. The major difference is that the search locations are the end of a cross (“X”), rather than a plus (“+”). Step size is reduced at each iteration and at the final stage, the search points can be either the end –point of “X” or “+”.

If the reference point of a block is taken as it upper left-hand corner (i, j), the candidate block position is taken as (0,0) and the sum of absolute difference (*SAD*) is taken as the distortion measure, the cross search algorithm may be summarized as follows:

Step- 1 **:** If $|SAD_k(0,0) - SAD_{k-1}(0,0)| < T$, (where $SAD_k(.,.)$ refers to the k^{th} frame and *T* is a predefined threshold), then the candidate block is taken as a non moving block and the search stops.

Step-2 : Initialize the minimum *SAD*- position (*m,n*) at *m* = 0, *n* = 0 and set the search step size *p* = *w*/2, where *w*is the maximum displacement possible in either directions.

Step-3 : Set the position (i, j) to the minimum position *(m,n).*

Step –4 : Find the minimum *SAD*-position *(m,n)*, considering the positions (i, j), $(i-p, j-p)$, $(i-p, j+p)$, $(i+p, j-p)$ and $(i+p, j+p)$.

Step –5 : If *p* =1, go to *step-6*, otherwise, set *p* as *p*/2 and go to *step-3*.

Step-6 : If the final minimum position (*m,n*) is either (i, j) , $(i-1, j-1)$ or $(i+1, j+1)$ go to *step-7*, otherwise go to *step-8*.

Step-7 : Search for the minimum position at (*m,n*) (*m*-1,*n*) (*m,n*-1) and (*m, n*+1). These are the end points of a "+".

Step-8. Search for the minimum position at (*m,n*), (*m*-1, *n*-1), (*m*-1, *n*+1), (*m*+1, *n*-1) and (*m*+1, *n*+1), In this case, the search positions are the end points of a "X".

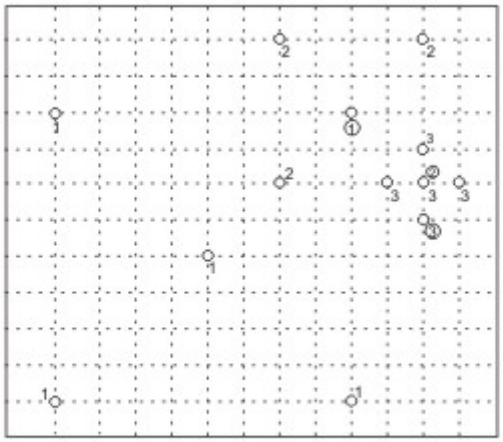

Figure 22.1 An example of Cross Search Algorithm (CSA)

The cross search algorithm is illustrated in Fig.22.1. The numbers written correspond to the search step(iteration) number. The minimum *SAD* position is shown by circling the number. In this example, the last stage (*p* =1) requires the search around the end-points of "+".

22.1.1 Computational complexity of CSA:

We find from the algorithms that there are altogether $\log_2 w$ search stages and each stage requires 4 search positions. In addition, there is an extra computation at position (0,0) in the first stage. The final stage requires 4 extra search positions. Thus, the total number of computations becomes $5 + 4\log_2 w$.

22.2 Three step search (TSS) algorithm

The *TSS* algorithm is yet another variant of the 2-D logarithm search with the major difference that there are nine checking points in each stage, with the current minimum distortion position at the centre. The eight other checking points correspond to the end of a "+" and "X", as illustrated in fig 22.2.

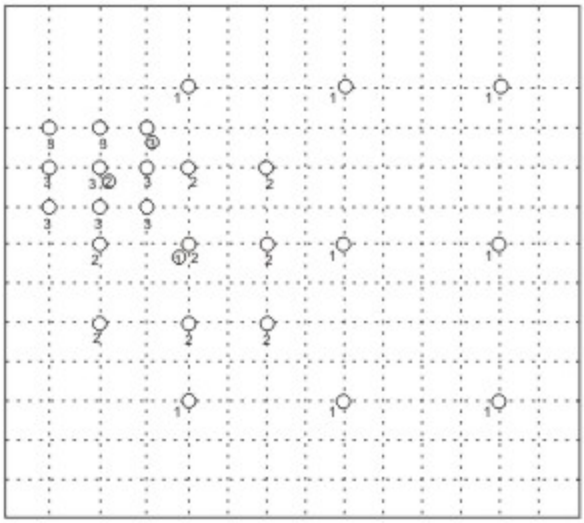

Figure 22.2 A example of Three-Step Search (TSS)

In the first stage, the centre of the search position is (0,0), i.e., the current candidate block position. Initially, the search step *p* is set as *p* = *w*/2, where *w* is the maximum possible search range. The nine positions to be searched are (*-p*,*-p*), (0,*-p*), (*p*,*-p*) (*-p*,0), (0,0). Let's say that the minimum out of these nine position is (*m,n*). In the next stage of search, (*m,n*) is taken as the search centre, the search. However, in the first stage, position (0,0) involves an extra computation. Hence, the total number of computations is $1 + 8\log_2 w$.

22.3 New three-step search (NTSS) algorithm

The *NTSS* algorithm involves a significant addition to the *TSS* algorithm in following respects –

a) a centre biased checking point pattern is introduced in the first stage and

b) a half-way stop techniques is used for stationary or the quasi stationary block.

When the motion vector distribution patterns on typical video sequences are studied, it is observed that the global minimum distribution is centre biased (i.e., majority of the blocks are stationary or quasi-stationary. With such a particular from of distribution, the search point pattern should also be centre biased. Thus, the modifications made in *NTSS* over *TSS* are:

a) In the first step, in addition to the original nine checking points in *TSS*, eight extra search points are added, which are eight neighbours of search window centre, as illustrated in fig 22.3. The first step therefore has 17 search positions.

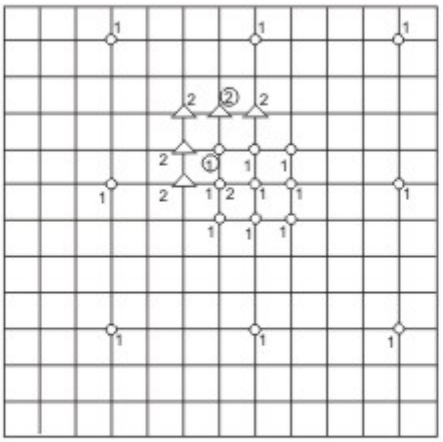

Figure 22.3 An example of new three-step search (NTSS)

b) A half-way stop techniques is used as follows –

i) If the minimum distortion position in the first step is (0,0), i.e., centre of the search, the block is considered stationary and further search is stopped.

ii) If the minimum distortion position in the first step is one of the eight neighbors of the centre of the search, the block is considered quasi-stationary. In such cases, a second and final stage of search is also carried out by considering eight neighbors of the minimum distortion position, which may include some of the points already checked.

If the minimum distortion position does not satisfy the stationary or the quasi-stationary conditions, the search in the second and the subsequent stages are done exactly as per the *TSS*.

22.3.1 Computational complexity of NTSS:

The computational complexity of the NTSS algorithm is expressed in terms of the following probabilities----

P_1 = Probability that the block is stationary.

P_2 = Probability that the block is quasi stationary and the minimum is one of the four neighboring positions around the vertical or the horizontal.

$P_2^{'}$ = Probability that the block is quasi stationary and the minimum is one of the four diagonal neighbors.

When P_1 condition occurs, only 17 computations are involved and there is no further computation beyond the first stage.

When P_2 condition occurs, five out of the eight second stage searches are saved, ie, there are total 20 computations, comprising of 17 in the first and 3 in the second stage.

When $P_2^{'}$ condition occurs, three out of the eight second stage searches are saved. Thus, there are 22 computations – 17 in the first and 5 in the second stage.

When neither of these conditions occurs, we have *TSS* like computations. If it really involves three stages, there are 17 computations in the first stage, 8 new search positions in each of the second and third stages involving 17 + 2 x 8 = 33 computations.

Thus, the total number of computations can be expressed as

$$17 P_1 + 20 P_2 + 22 P_2^{'} + 33 (1-P_1 - P_2 - P_2^{'})$$

These probabilities are dependent on how many stationary or quasi stationary blocks a video frame contains. For videophones and videoconferencing applications, the block motion is gentle and the background is stationary. Thus, the saving is quite substantial.

22.4 Diamond- search (DS) algorithms

The *DS* algorithm, proposed by Zhu and Ma employs two search patterns, as shown in *fig 22.4 (a) and (b).* The first pattern [fig.22.4 (a)] consists of nine checking points from which eight points surround the centre, forming a diamond shape. This pattern is referred to as *large diamond search pattern* (*LDSP*). The second pattern, shown in fig 22.4 (b) is referred to as *small diamond search pattern* (*SDSP*). The *SDSP* consists of five checking points as shown.

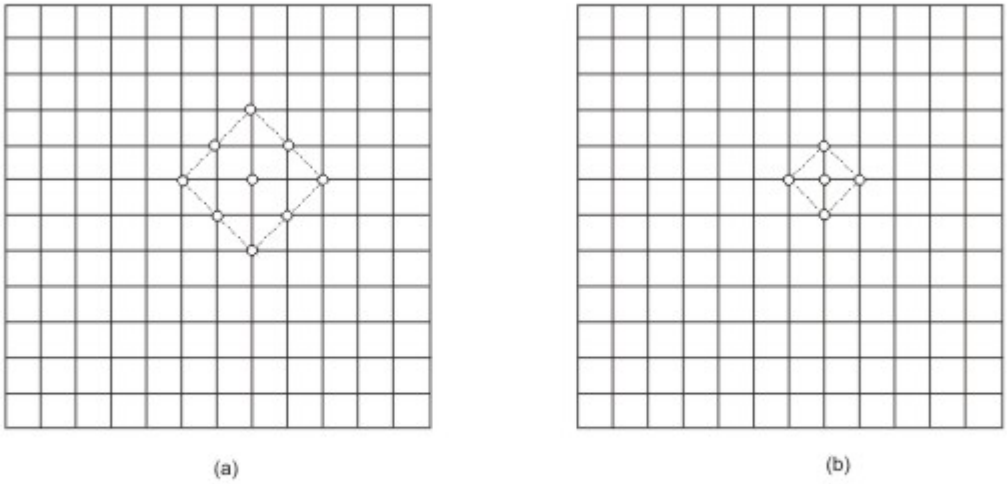

Figure 22.4 (a) large-diagonal search pattern (LDSP) and (b) small-diagonal search pattern (SDSP)

In *DS* algorithms, first *LDSP* is repetitively applied with the current minimum distortion position as centre which is initially assumed to be the position corresponding to the candidate block. If the minimum distortion position after the *LDSP* is still the centre of the *LDSP,* no more *LDSP* can be applied with that position as the centre and the search pattern is switched from *LDSP* to *SDSP*, i.e., a finer search applied only once as the final stage. The minimum distortion position obtained out of these five positions provides the motion vector for the best matching block.

An example of applying the *DS* algorithm is illustrated in fig.22.5. In this example, the numerical values written at the grid intersections are the distortion values. *LDSP* is first applied at the following positions : (-2,0), (-1,1), (0,2), (1,1), (2,0), (1,-1), (0,-2), (-1,-1) with (0,0) as the center. In this case, the minimum distortion value is observed at (1,-1), corresponding to the distortion value of 15. Since the minimum position is not the center of *LDSP*, we have to apply *LDSP* once more with (1,-1) at the center. The new *LDSP* search positions are (-1,-1), (0,0), (1,1), (2,0), (3,-1), (2,-2), (1,-3) and (0,-2). The reader may verify that this time, the minimum value is obtained at the *LDSP* center, i.e, at (1,-1). We therefore switch from *LDSP* to *SDSP* with (1,-1) as the center and carry out the searches at (0,-1),

(1,0), (2,-1) and (1,-2). The minimum value is obtained at (0,-1), corresponding to a value of 13 and this is the final position for motion vector search.

22.4.1 Computational complexity of DS algorithm :

If we have to apply *LDSP N* times, then for the first time, we have nine different search points and for the remaining (*N*-1) times, eight new search points are added each time. *SDSP* is applied only once, adding four new search points. Thus the total number of computations is given by

$$9 + 8(N\text{-}1) + 4 = 5 + 8N$$

N depends on the motion content of the block. If *N* is small, as is the case for stationary and quasi stationary blocks, *DS* algorithm is computationally advantageous.

22.5 Block based gradient descent search (BBGDS) algorithm

The *BBGDS* algorithm proposed by Liu and Fig evaluates an objective function (Mean of absolute difference as per their implementation) starting from a small centralized checking block. The minimum value within that checking block is found and the search proceeds in the gradient descent direction, ie, direction in which the minimum is found.

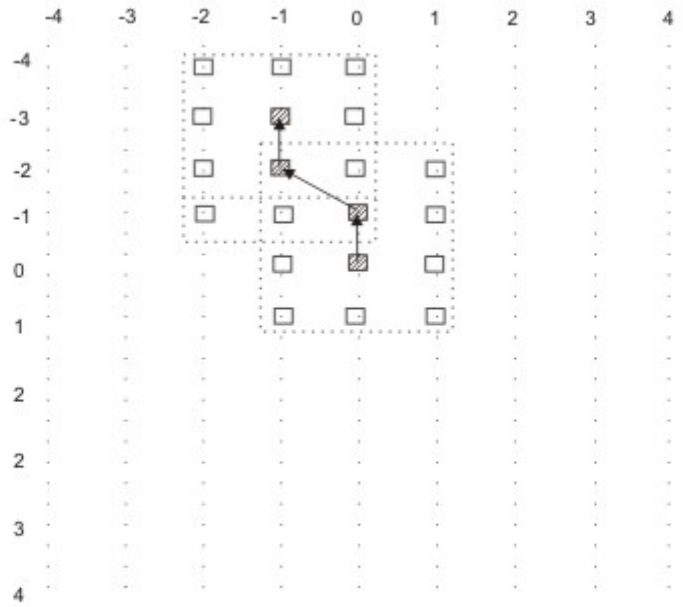

Figure 22.5 Block based gradient descent search (BBGDS) algorithm

As illustrated in fig.22.6, the starting search centre is (0,0), which corresponds to the position of the candidate block. The minimum is determined by searching within a 3x3 checking block around (0,0). In this example, the minimum is found as (0,-1), which becomes the new centre of the checking block. The trajectory of the position of minimum is indicated by the arrows. If within any checking block, the position of minimum is the centre, further search stops. In this example, the motion vector determined is (-1,-3).

The BBGDS algorithm primarily works under the assumption that the global minimum has a monotonic distortion in its neighborhood.

22.5.1 Computational complexity of BBGDS:

In this algorithm, each search step has nine computations in the 3 x 3 checking block, out of which maximum four search positions are new. If we assume a total search range of $\pm w$ pixels, then the maximum number of search stages in $2w$ and thus the upper bound on number of computations is $5+ 8w$, since for the first search stage, all nine positions are to be computed new. For quasi-stationary and stationary blocks, the number of search stages will be too small and *BBGDS* involves small number of computations.

References

1. J. R. Jain and A. K. Jain, " Displacement measurements and its application in interframe image coding", IEEE Transaction on Communication, Vol. COM-29, No. 12, December, 1981, pp. 1799-1808.

2. M. Ghanbari, " The cross-search algorithm for motion estimation", IEEE Transactions on Communication Vol. COM-38, no 7, July, 1990, pp 950-953.

3. T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro. " Motion compensated interframe coding for video conferencing". Proc NTC 81, New Orlean, LA, Nov/Dec. 1981; pp C96.1 – 9.6.5.

4. R. L., B. Zeng and M. L. Liou, " A new three-step search algorithm for block motion estimation". IEEE Transactions on circuits and systems for Video Technology, Vol. 4, no. 4, August 1994, pp. 438-442.

5. S. Zhu and K. Ma, " A new diamond search algorithm for fast block-matching motion estimation". IEEE Transactions on Image Processing, Vol. 9, no.2, February 2000, pp. 287-290.

6. L. Liu and E. Feig, " A block-based gradient descent search algorithm for block motion estimation in video coding", IEEE Transactions on circuits and systems for Video Technology , Vol. 6, no. 4, August, 1996, pp 419-422.

7. L. Po and W. Ma, “ A novel four-step search algorithm for fast block motion estimation. “ IEEE Transaction on Circuits and system for Video Technology, Vol. 6, No.3, June 1996, pp 313-317.

8. X. Jing and L. Chau, “ An efficient three step search algorithm for block motion estimation,. “ IEEE Transactions of Multimedia, Vol. 6, no. 3, June 2004, pp 435-438

9. R. Srinivasan and K. R. Rao. “ Predictive coding based on efficient motion estimation, “ IEEE transaction on communication Vol. COM-33, August, 1985, pp 888-896.