IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 2, Issue 6 (Jul. – Aug. 2013), PP 20-28
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

High Performances Block Matching Algorithms for Motion Estimation

[1]Moussa Bellal, [1]Belloulata Kamel, [2]Shiping Zhu
[1]*Department of Electronics, School of Engineering, University of Sidi Bel Abbès, Sidi Bel Abbès 22000, Algeria.*
[2]*Department of Measurement Control and Information Technology, School of Instrumentation Science and Optoelectronics Engineering, Beihang University, Beijing 100191, China.*

Abstract: *This paper is a review of the block matching algorithms used for motion estimation in video compression. It implements and compares 8 different types of block matching algorithms that range from the very basic Exhaustive Search to the recent fast adaptive algorithms like Adaptive Rood Pattern Search. The algorithms that are evaluated in this paper are widely accepted by the video compressing community and have been used in implementing various standards, ranging from MPEG1 / H.261 to MPEG4 / H.263. The paper also presents a very brief introduction to the entire flow of video compression.*
Index Terms: *Block matching, Estimation / motion compensation and disparity, H.264/MPEG-4 AVC.*

I. Introduction

WITH the advent of the multimedia age and the spread of Internet, video storage on CD/DVD and streaming video has been gaining a lot of popularity. The ISO Moving Picture Experts Group (MPEG) video coding standards pertain towards compressed video storage on physical media like CD/DVD, where as the International Telecommunications Union (ITU) addresses real-time point-to-point or multi-point communications over a network. The former has the advantage of having higher bandwidth for data transmission. In either standard the basic flow of the entire compression-decompression process is largely the same and is depicted in Fig. 1. The encoding side estimates the motion in the current frame with respect to a previous frame. A motion compensated image for the current frame is then created that is built of blocks of image from the previous frame. The motion vectors for blocks used for motion estimation are transmitted, as well as the difference of the compensated image with the current frame is also JPEG encoded and sent. The encoded image that is sent is then decoded at the encoder and used as a reference frame for the subsequent frames. The decoder reverses the process and creates a full frame. The whole idea behind motion estimation based video compression is to save on bits by sending JPEG encoded difference images which inherently have less energy and can be highly compressed as compared to sending a full frame that is JPEG encoded. Motion JPEG, where all frames are JPEG encoded, achieves anything between 10:1 to 15:1 compression ration, where as MPEG can achieve a compression ratio of 30:1 and is also useful at 100:1 ratio [1].

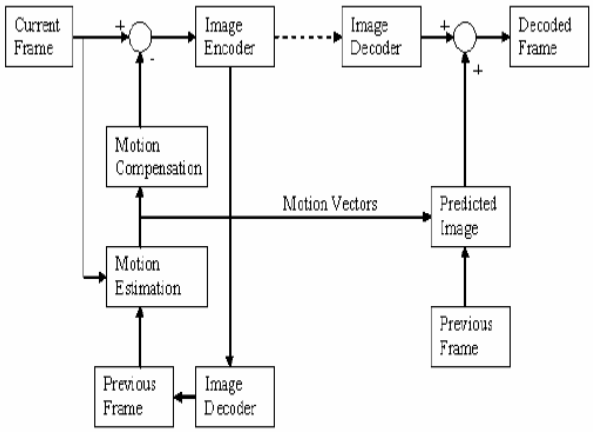

Fig.1. MPEG / H.26x video compression process flow.

It should be noted that the first frame is always sent full, and so are some other frames that might occur at some regular interval (like every 6^{th} frame). The standards do not specify this and this might change with

every video being sent based on the dynamics of the video.

The most computationally expensive and resource hungry operation in the entire compression process is motion estimation. Hence, this field has seen the highest activity and research interest in the past two decades.

This paper implements and evaluates the fundamental block matching algorithms from the mid-1980s up to the recent fast block matching algorithms of year 2002. It also presents a literature review of few papers from the last 3 years. The algorithms that have been implemented are Exhaustive Search (ES), Three Step Search (TSS), New Three Step Search (NTSS), Simple and Efficient TSS (SES), Four Step Search (4SS), Diamond Search (DS), Hexagon-Based Search Algorithm (HEXBS) , and Adaptive Rood Pattern Search (ARPS). Section II explains block matching in general and then the above algorithms in detail. Section III compares them and presents some simulation results. Section IV is a literature survey of the more recent algorithms, followed by summary and references.

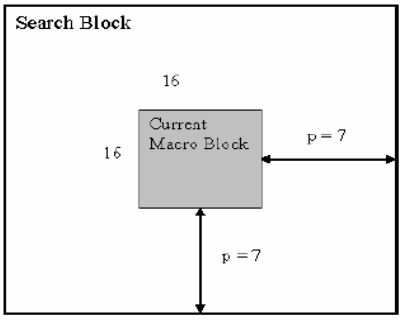

Fig. 2. Block Matching a macro block of side 16 pixels and a search parameter p of size 7 pixels.

II. Block Matching Algorithms

The underlying supposition behind motion estimation is that the patterns corresponding to objects and background in a frame of video sequence move within the frame to form corresponding objects on the subsequent frame. The idea behind block matching is to divide the current frame into a matrix of 'macro blocks' that are then compared with corresponding block and its adjacent neighbors in the previous frame to create a vector that stipulates the movement of a macro block from one location to another in the previous frame. This movement calculated for all

The macro blocks comprising a frame, constitutes the motion estimated in the current frame. The search area for a good macro block match is constrained up to p pixels on all fours sides of the corresponding macro block in previous frame. This 'p' is called as the search parameter. Larger motions require a larger p, and the larger the search parameter the more computationally expensive the process of motion estimation becomes.

Usually the macro block is taken as a square of side 16 pixels, and the search parameter p is 7 pixels.

The idea is represented in Fig 2. The matching of one macro block with another is based on the output of a cost function. The macro block that results in the least cost is the one that matches the closest to current block. There are various cost functions, of which the most popular and less computationally expensive is Mean Absolute Difference (MAD) given by equation (i).

Another cost function is Mean Squared Error (MSE) given by equation (ii).

$$MAD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|C_{ij} - R_{ij}\right| \quad \textbf{(i)}$$

$$MSE = 1/N^2\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(C_{ij} - R_{ij}\right)^2 \quad \textbf{(ii)}$$

Where N is the side of the macro bock, Cij and Rij are the pixels being compared in current macro block and reference macro block, respectively.

Peak-Signal-to-Noise-Ratio (PSNR) given by equation (iii) characterizes the motion compensated image that is created by using motion vectors and macro clocks from the reference frame.

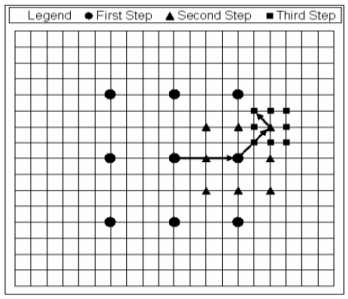

Fig. 3. Three Step Search procedure. The motion vector is (5, -3).

$$PSNR = 10 \log_{10} \left[\frac{(\text{Peak to value of original data})^2}{MSE} \right] \quad \textbf{(iii)}$$

A. Exhaustive Search (ES)

This algorithm, also known as Full Search [1], is the most computationally expensive block matching algorithm of all. This algorithm calculates the cost function at each possible location in the search window. As a result of which it finds the best possible match and gives the highest PSNR amongst any block matching algorithm. Fast block matching algorithms try to achieve the same PSNR doing as little computation as possible. The obvious disadvantage to ES is that the larger the search window gets the more computations it requires.

B. Three Step Search (TSS)

This is one of the earliest attempts at fast block matching algorithms and dates back to mid 1980s [1]. The general idea is represented in Figure 3. It starts with the search location at the center and sets the 'step size' S = 4, for a usual search parameter value of 7. It then searches at eight locations +/-S pixels around location (0,0). From these nine locations searched so far it picks the one giving least cost and makes it the new search origin. It then sets the new step size S = S/2, and repeats similar search for two more iterations until S = 1. At that point it finds the location with the least cost function and the macro block at that location is the best match.

The calculated motion vector is then saved for transmission. It gives a flat reduction in computation by a factor of 9. So that for p = 7, ES will compute cost for 225 macro blocks whereas TSS computes cost for 25 macro blocks.

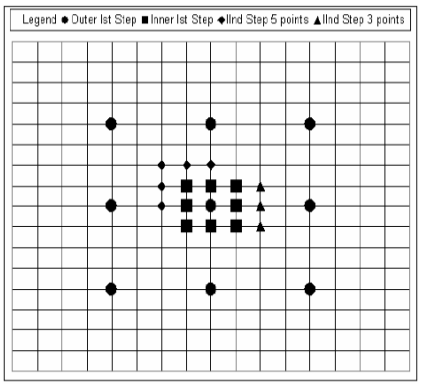

Fig. 4.New Three Step Search block matching: Big circles are checking points in the first step of TSS and the squares are the extra 8 points added in the first step of NTSS. Triangles and diamonds are second step of NTSS showing 3 points and 5 points being checked when least weight in first step is at one of the 8 neighbors of window center.

The idea behind TSS is that the error surface due to motion in every macro block is unimodal. A unimodal surface is a bowl shaped surface such that the weights generated by the cost function increase monotonically from the global minimum.

C. New Three Step Search (NTSS)

NTSS [2] improves on TSS results by providing a center biased searching scheme and having provisions for half way stop to reduce computational cost. It was one of the first widely accepted fast algorithms and frequently used for implementing earlier standards like MPEG 1 and H.261.

The TSS uses a uniformly allocated checking pattern for motion detection and is prone to missing small motions. The NTSS process is illustrated graphically in Fig 4. In the first step 16 points are checked in addition to the search origin for lowest weight using a cost function. Of these additional search locations, 8 are a distance of S = 4 away (similar to TSS) and the other 8 are at S = 1 away from the search origin. If the lowest cost is at the origin then the search is stopped right here and the motion vector is set as (0, 0). If the lowest weight is at any one of the 8 locations at S = 1, then we change the origin of the search to that point and check for weights adjacent to it. Depending on which point it is we might end up checking 5 points or 3 points (Fig 7(b) & (c)). The location that gives the lowest weight is the closest match and motion vector is set to that location. On the other hand if the lowest weight after the first step was one of the 8 locations at S = 4, then we follow the normal TSS procedure. Hence although this process might need a minimum of 17 points to check every macro block, it also has the worst-case scenario of 33 locations to check.

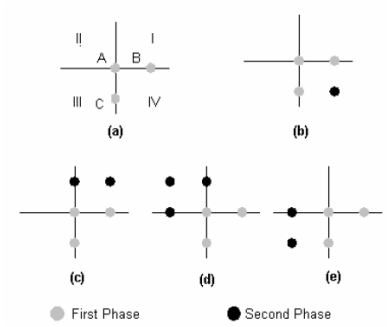

Fig. 5. Search patterns corresponding to each selected quadrant: (a) Shows all quadrants (b) quadrant I is selected (c) quadrant II is selected (d) quadrant III is selected (e) quadrant IV is selected

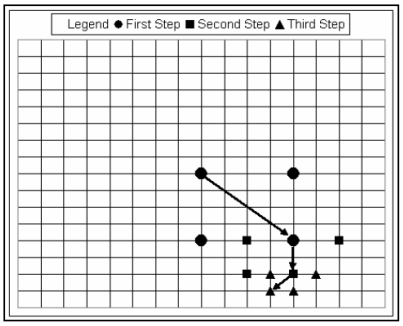

Fig. 6. The SES procedure. The motion vector is (3, 7) in this example.

D. Simple and Efficient Search (SES)

SES [3] is another extension to TSS and exploits the assumption of unimodal error surface. The main idea behind the algorithm is that for a unimodal surface there cannot be two minimums in opposite directions and hence the 8 point fixed pattern search of TSS can be changed to incorporate this and save on computations. The algorithm still has three steps like TSS, but the innovation is that each step has further two phases. The search area is divided into four quadrants and the algorithm checks three locations A,B and C as shown in Figure Y. A is at the origin and B and C are S = 4 locations away from A in orthogonal directions. Depending on certain weight distribution amongst the three the second phase selects few additional points (Fig 5). The rules for determining a search quadrant for seconds phase are as follows:

If MAD(A) ≥MAD(B) and MAD(A) ≥MAD(C), select (b);
If MAD(A) ≥MAD(B) and MAD(A) ≤MAD(C), select (c);
If MAD(A) < MAD(B) and MAD(A) < MAD(C), select (d);
If MAD(A) < MAD(B) and MAD(A) ≥MAD(C), select (e);

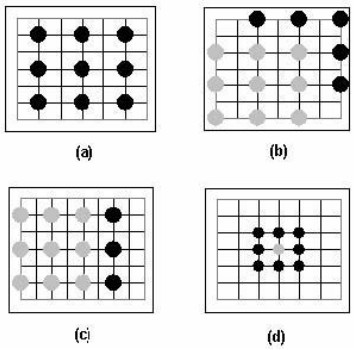

Fig. 7. Search patterns of the FSS. (a) First step (b) Second/Third step (c)Second/Third Step (d) Fourth Step

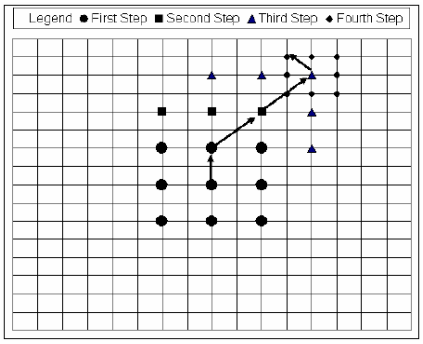

Fig. 8. Four Step Search procedure. The motion vector is (3, -7).

Once we have selected the points to check for in second phase, we find the location with the lowest weight and set it as the origin. We then change the step size similar to TSS and repeat the above SES procedure again until we reach S = 1. The location with the lowest weight is then noted down in terms of motion vectors and transmitted. An example process is illustrated in Fig 6.

Although this algorithm saves a lot on computation as compared to TSS, it was not widely accepted for two reasons. Firstly, in reality the error surfaces are not strictly unimodal and hence the PSNR achieved is poor compared to TSS. Secondly, there was another algorithm, Four Step Search, that had been published a year before that presented low computational cost compared to TSS and gave significantly better PSNR.

E. Four Step Search (4SS)

Similar to NTSS, 4SS [4] also employs center biased searching and has a halfway stop provision. 4SS sets a fixed pattern size of S = 2 for the first step, no matter what the search parameter p value is. Thus it looks at 9 locations in a 5x5 window. If the least weight is found at the center of search window the search jumps to fourth step. If the least weight is at one of the eight locations except the center, then we make it the search origin and move to the second step. The search window is still maintained as 5x5 pixels wide. Depending on

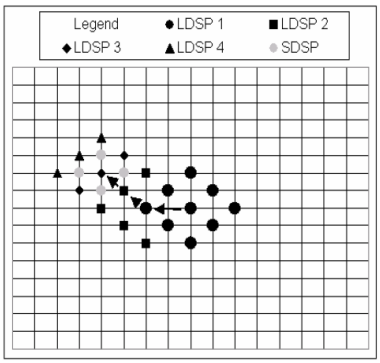

Fig. 9. Diamond Search procedure. This figure shows the large diamond search pattern and the small diamond search pattern. It also shows an example path to motion vector (-4, -2) in five search steps-four times of LDSP and one time of SDSP.

where the least weight location was, we might end up checking weights at 3 locations or 5 locations. The patterns are shown in Fig 7. Once again if the least weight location is at the center of the 5x5 search window we jump to fourth step or else we move on to third step. The third is exactly the same as the second step. IN the fourth step the window size is dropped to 3x3, i.e. S = 1. The location with the least weight is the best matching macro block and the motion vector is set to point o that location. A sample procedure is shown in Fig 8. This search algorithm has the best case of 17 checking points and worst case of 27 checking points.

F. Diamond Search (DS)

DS [5] algorithm is exactly the same as 4SS, but the search point pattern is changed from a square to a diamond, and there is no limit on the number of steps that the algorithm can take. DS uses two different types of fixed patterns, one is Large Diamond Search Pattern (LDSP) and the other is Small Diamond Search Pattern (SDSP). These two patterns and the DS procedure are illustrated in Fig. 9. Just like in FSS, the first step uses LDSP and if the least weight is at the center location we jump to fourth step. The consequent steps, except the last step, are also similar and use LDSP, but the number of points where cost function is checked are either 3 or 5 and are illustrated in second and third steps of procedure shown in Fig.9. The last step uses SDSP around the new search origin and the location with the least weight is the best match. As the search pattern is neither too small nor too big and the fact that there is no limit to the number of steps, this algorithm can find global minimum very accurately. The end result should see a PSNR close to that of ES while computational expense should be significantly less.

G. Hexagon-Based Search Algorithm (HEXBS)

Thus, the algorithm "Hexagon-Based Pattern" (HEXB) proposed in [6] tries to find a solution to these remarks, or at least offers an alternative to the DS algorithm while keeping the same philosophy. Indeed, the algorithm uses, as in the DS, two research models

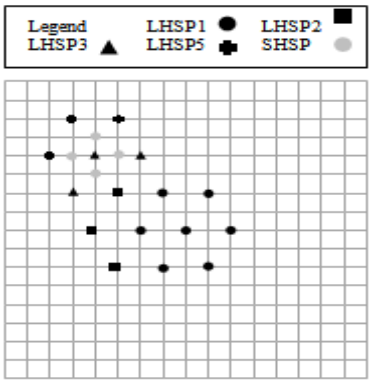

Fig. 10. Hexagon-Based Search Algorithm procedure. This figure shows the large diamond search pattern and the small diamond search pattern. It also shows an example path to motion vector (-3, -2) in five search steps-four times of LDSP and one time of SDSP.

Shown in Fig. 10: a large model ("Large Hexagonal Pattern Search" - LHSP) and a small model ("Small Hexagonal Pattern Search" - SHSP). The first point that we can make is that the SHSP is identical to SDSP. Second, the LHSP contains seven points however LDSP contains nine. This allows a glimpse of the decrease in the number of points tested by the algorithm hexagonal. Finally, the point on the LHSP is located at a distance of 2 or $\sqrt{5} \approx 2.2361$.

The hexagon is the best approximation of the square circle; this is an advantage if the L2 norm is used. Finally, we note that the hexagon is a figure that can cover the space optimally and we believe to increase the performance of the motion estimator.

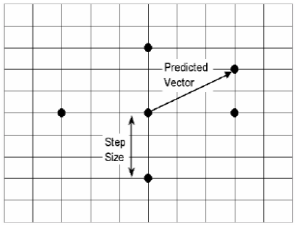

Fig. 11. Adaptive Root Pattern: The predicted motion vector is (3,-2) ,and the step size S = Max(|3|, |-2|) = 3.

G. Adaptive Rood Pattern Search (ARPS)

ARPS [7] algorithm makes use of the fact that the general motion in a frame is usually coherent, i.e. if the macro blocks around the current macro block moved in a particular direction then there is a high probability that the current macro block will also have a similar motion vector. This algorithm uses the motion vector of the macro block to its immediate left to predict its own motion vector. An example is shown in Fig. 10. The predicted motion vector points to (3, -2). In addition to checking the location pointed by the predicted motion vector, it also checks at a rood pattern distributed points, as shown in Fig 11, where they are at a step size of S = Max (|X|, |Y|). X and Y are the x-coordinate and y-coordinate of the predicted motion vector. This rood pattern search is always the first step. It directly puts the search in an area where there is a high probability of finding a good matching block. The point that has the least weight becomes the origin for subsequent search steps, and the search pattern is changed to SDSP. The procedure keeps on doing SDSP until least weighted point is found to be at the center of the SDSP. A further small improvement in the algorithm can be to check for Zero Motion Prejudgment [8], using which the search is stopped half way if the least weighted point is already at the center of the rood pattern.

The main advantage of this algorithm over DS is if the predicted motion vector is (0, 0), it does not waste computational time in doing LDSP; it rather directly starts using SDSP. Furthermore, if the predicted motion vector is far away from the center, then again ARPS save on computations by directly jumping to that vicinity and using SDSP, whereas DS takes its time doing LDSP. Care has to be taken to not repeat the computations at points that were checked earlier. Care also needs to be taken when the predicted motion vector turns our to match one of the rood pattern location. We have to avoid double computation at that point. For macro blocks in the first column of the frame, rood pattern step size is fixed at 2 pixels.

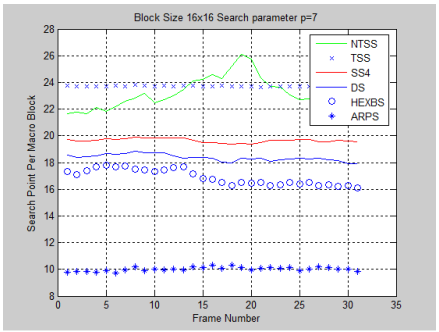

Fig. 12. Search points per macro block while computing the PSNR performance of Fast Block Matching Algorithms. Caltrain Sequence was used with a frame distance of 2.

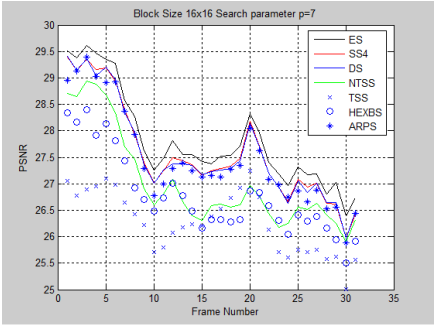

Fig. 13. PSNR performance of Fast Block Matching Algorithms. Caltrain Sequence was used with a frame distance of 2.

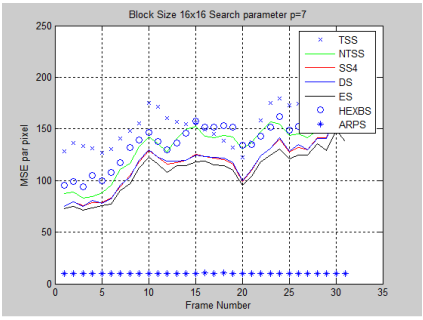

Fig. 14. . MSE performance of Fast Block Matching Algorithms. Caltrain Sequence was used with a frame distance of 2.

III. Simulation Results

The Figure 12, 13.14 illustrate the comparison frame by frame PSNR, MSE and average number of blocks tested by block after the application of DS, 4SS, NTSS, TSS, and the algorithm FS in order to "Caltrain "respectful distance in frame respectively. The results are very similar to the results of [1]. The values of PSNR, MSE and average number of blocks tested by block "Caltrain" are presented in Table 1.If we classify the different methods of block-matching according to the PSNR, we get decreasing order: "Full Search", "Diamond Search", "Four-Step Search", "N Three-Step Search", "Three-Step Search »..... It is obvious that any method can quickly surpass the method qualitatively Full Search. Figure 13 and Table 1 show that the losses are 1.57 dB for "Three-Step Search", 1.06 dB for "Hexagon-Based Pattern" ,0.81 dB for "N Three-Step Search", 0.35 dB for " Adaptive Rood Pattern Search ", 0.26 dB for "Diamond Search", 0.24 dB for "Four-Step Search". This method is fast and the method gives qualitatively better results. Note that this data is related to the sequence "Caltrain" frame with a distance = 2.If we are now considering the speed of research and tested means of blocks, we note the difference between fast algorithms and exhaustive algorithm is considerable. Indeed, the visit FS 207.41 in 4.55 Seconds blocks as, for example, the ARPS did visit 10.01 in 0.30 Seconds blocks a ratio of near 20. In addition, we also note that the HEXBS and ARPS algorithms are faster than block and visit the TSS 1981.Naturellement that date, the number of blocks visited and length of the block-matching are linked and can classify the algorithms by speed descending order: FS, TSS, NTSS, 4SS, DS, HXBS, APRS

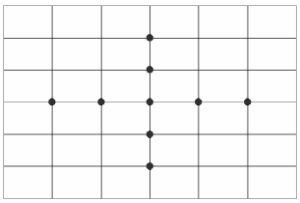

Fig. 15. Cross Search Pattern: This pattern is used by CDS, SCDS, and NCDS as their search start pattern. While CDS uses all 9 points, SCDS and NCDS use only the inner 5 points.

IV. Some Other Recent Algorithms

DS proved to be best block matching algorithm of the last century. Every new algorithm in the new millennia is an improvement on the results of DS. Cross Diamond Search (CDS) [8], Small Cross Diamond Search (SCDS) [9], and New Cross Diamond Search (NCDS) [10], all improve on the performance of DS by modifying the starting search pattern from LDSP to cross search pattern (CSP) shown in Fig 13. Amongst themselves these three algorithms differ with respect to the number of points being used out of the CSP. CDS uses all the 9 points whereas SCDS and NCDS use only the inner 5 points to start and then expand their search. However, analogous to the NTSS that eventually ends up doing similar calculations like TSS, these CSP based variants end up going the DS way. Another reason for their improvement over DS is the provision of multiple half-step stops. It should be mentioned that out of the three CSP based variants only NCDS comes closer to the performance of ARPS. The others although an improvement on DS, do not match the performance of ARPS.

V. Summary

The past two decades have seen the growth of wide acceptance of multimedia. Video compression plays an important role in archival of entertainment based video (CD/DVD) as well as real-time reconnaissance / video conferencing applications. While ISO MPEG sets the standard for the former types of application, ITU sets the standards for latter low bit rate applications. In the entire motion based video compression process motion estimation is the most computationally expensive and time-consuming process.

The research in the past decade has focused on reducing both of these side effects of motion estimation. Block matching techniques are the most popular and efficient of the various motion estimation techniques. This paper first describes the motion compensation based video compression in brief. It then illustrates and simulates 8 of the most popular block matching algorithms, with their comparative study at the end. Three more, very recent, block matching algorithms are studied in the end as part of literature review. Of the various algorithms studied or simulated during the course of this final project ARPS turns out to be the best block matching algorithm.

References

[1] Aroh Barjatya,Block Matching Algorithms For Motion Estimation,Student Member, IEEE, DIP 6620 Spring 2004 Final Project Paper.

[2] Renxiang Li, Bing Zeng, and Ming L. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation", IEEE Trans. Circuits And Systems For Video Technology, vol 4., no. 4, pp. 438-442, August 1994.

[3] Jianhua Lu, and Ming L. Liou, "A Simple and Efficent Search Algorithm for Block-Matching Motion Estimation", IEEE Trans. Circuits And Systems For Video Technology, vol 7, no. 2, pp. 429-433, April 1997

[4] Lai-Man Po, and Wing-Chung Ma, "A Novel Four-Step Search Algorithm for Fast Block Motion Estimation", IEEE Trans. Circuits And Systems For Video Technology, vol 6, no. 3, pp. 313-317, June 1996.

[5] Shan Zhu, and Kai-Kuang Ma, " A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation", IEEE Trans. Image Processing, vol 9, no. 2, pp. 287-290, February 2000.

[6] Ce Zhu, Xiao Lin, and Lap-Pui Chau, "Hexagon Search Pattern for Fast Block-Matching Motion Estimation", IEEE Trans. On circuit and systems for video technology, vol 12, no. 5, pp. 349-355, May 2000.

[7] Yao Nie, and Kai-Kuang Ma, "Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation", IEEE Trans. Image Processing, vol 11, no. 12, pp. 1442-1448, December 2002.

[8] Chun-Ho Cheung, and Lai-Man Po, "A Novel Small Cross-Diamond Search Algorithm for Fast Video Coding and Video Conferencing Applications", Proc. IEEE ICIP, September 2002.

[9] Shiping Zhu, Jun Tian, Xiaodong Shen, and Kamel, "A novel cross-hexagon search algorithm based on motion vector field prediction," in Proceedings of the IEEE International Symposium on Industrial Electronics, (ISIE '09), pp. 1870–1874, Seoul, Korea, July 2009.

[10] C Shiping Zhu, Jun Tian, Xiaodong Shen, and Kamel Belloulata, "A New Cross-Diamond Search Algorithm for Fast Block Matching Motion Estimation", in Proceedings of the IEEE International Conference on Image Processing (ICIP '09), vol. 1, pp. 1581–1584, Cairo, Egypt, November 2009.

[11] Kamel Belloulata, Shiping Zhu, and Zaikuo Wang, "A Fast Fractal Video Coding Algorithm Using Cross-Hexagon Search for Block Motion Estimation," ISRN Signal Processing, vol. 2011, Article ID 386128, 10 pages, 2011. doi:10.5402/2011/386128

Table 1: Average quality of the estimate according to the method of block-matching for sequence «Caltrain»

Evaluation / method BM		MSE	PSNR (dB)	Number of search points	Durée block-matching (sec)
Frame Distance =2	FS	148.89	27,83	207.41	4.55
	TSS	204.96	26,26	23,72	0.58
	NTSS	167.40	27,02	23,09	0.55
	4SS	164.18	27,59	19,65	0.51
	DS	164.64	27.57	18.36	0.48
	HEXBS	183.34	26.77	16.89	0.45
	ARPS	10.31	27.48	10.01	0.30