12

Block Based Motion Vector Estimation Using FUHS16, UHDS16 and UHDS8 Algorithms for Video Sequence

S. S. S. Ranjit
Universiti Teknikal Malaysia Melaka (UTeM)
Malaysia

1. Introduction

1. Fast Unrestricted Hexagon Search (FUHS16) algorithm

In this section a short description about the Fast Unrestricted Hexagon Search (FUHS16) algorithm for motion estimation development based on some of the existing algorithms that have been discussed and simulated. Comparison of the performance among techniques is conducted as part of experimental result preparation.

FUHS16 algorithm is developed based on 16 × 16 pixels in a block size and two different models of hexagon sizes are applied to perform the motion vector search. Figure 1 shows how a single frame is extracted into required block size, where in FUHS16 algorithm each frame size is represented by 176 × 144 pixels. This means, that each frame will have 9 blocks horizontally and 11 blocks vertically. Hence, there are 99 extracted blocks in a video frame.

Assumed has the following parameters

i = horizontal (9 blocks),
j = vertical (11 blocks),
i = 1: (r / bsize),
j = 1: (c / bsize),

where, r = 144, c = 176.

B = Block,
CF = current_frame,
BZ = Block_Size

Eq. (1) shows the formula to extract the video frame into 16 × 16 block size.

$$B=CF(1+BZ*(i-1):BZ*i,1+BZ*(j-1):BZ*j) \quad (1)$$

2. Fast unrestricted hexagon search algorithm search procedure

In the first step, large hexagon search shape with seven checking points are used to perform the search for the best-matched motion vector from the inner large hexagon search shape. If the best-matched motion vector is found at the center of large hexagon, the large hexagon search shape switches to small hexagon search shape that includes four checking points for the focused inner search.

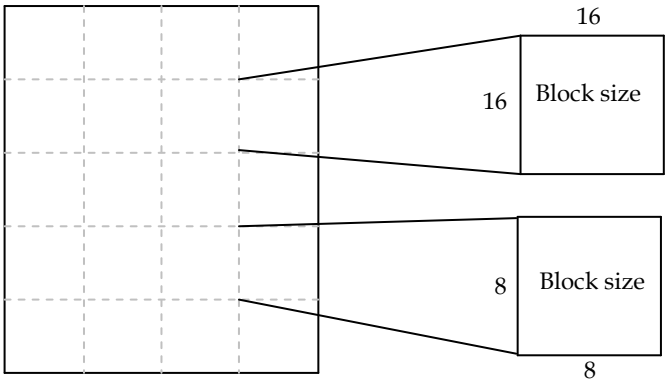


Fig. 1. Extraction 16 × 16 and 8 × 8 Pixels Block Size from Single Frame

These four checking points are compared in order to determine the final best-matched motion vector coordinate. Otherwise, the search continues around the point with the smallest MAD by using the same large hexagon search shape. This process continues till the large hexagon search shape moves along the direction of decreasing distortion. It is noted that a small hexagon search shape is applied in the final step after the decreasing distortion reaches optimum of the motion vector for large hexagon search shape. Then the small hexagon search shape will focus on the final search for the best-matched motion vector coordinate.

The proposed FUHS16 algorithm can be described further in the following three steps.

i. Starting

The large hexagon search shape with seven checking points are centered at (+8, +8) and it is assumed as (0, 0). We name this as the predefined search window in the motion field. If the smallest MAD point with best-matched motion vector is found to be at the center of the large hexagon, we will proceed to Step (iii); otherwise Step (ii) will be proceed.

ii. Searching

Since the MAD point in the previous search step is not located at the center, a new large hexagon search shape is formed to perform new checking. It confines of seven checking points. Now the new MAD point is identified. If the MAD point is located at the center of newly to form large hexagon search shape, we proceed to Step (iii); otherwise, this step is repeated continuously till the next smallest MAD is again found at the center of large hexagon search shape.

iii. Ending

For the final search, large hexagon search shape determines the best-matched motion vector which is located at the center inner large hexagon search shape. After this, it will then switch to the small hexagon search shape to perform the final best-matched motion vector coordinate, MAD search point. The four points in the small hexagon search shape are evaluated to compare with the current MAD point. The MAD point is the final solution of best-matched motion vector coordinate location.

From the above procedure, it can be easily derived that the total number of search points per frame are,

$$N_{FUHS16(m_x,m_y)} = (LHS + SHS) + 3n, \quad (2)$$

Where, (m_x, m_y) = final best-matched motion vector coordinate,

n = number of execution of Step (ii),
LHS = Large hexagon shape search points,
SHS = Small hexagon shape search points.

In Figure 2, the motion vector is predicted at MAD_4 after emerging 3 hexagon search shape. Based on the Equation 2, the $N_{FUHS(m_x m_y)} = 7 + 3(3) + 4 = 20$. The FUHS16 needs 20 search points to predict final best-matched motion vector at MAD_4.

MAD_0 is the starting search point in the large hexagon search shape, center at coordinate (+8, +8) – it is then assumed as coordinate (0, 0). The outer six points in the large hexagon search shape are evaluated to compare to the optimal MAD in the first search. If the optimal MAD is found to be at the center, then small hexagon search shape will take place to focus on the fine resolution search to predict the optimum motion vector in that area.

If the smallest MAD is found at one of the outer search point of large hexagon search shape, then three new search points are emerged to form a new large hexagon search shape as shows in Figure 2. The current optimal MAD is known as MAD_1 and is positioned at the center of newly form large hexagon search shape. The current coordinate of MAD_1 is at (+7, +10). All the points in the large hexagon search shape are again evaluated to predict the optimal MAD in the second search.

In the second search, the optimal MAD is MAD_2 which is located at one the six outer search points. Again three new search points are emerged to form a new large hexagon search shape and repositioned MAD_2 to be at the center of newly formed large hexagon search shape. The newly form large hexagon search shape is centered at coordinate (+6, +12). All the search points surrounding the large hexagon search shape are evaluated again to predict the optimal MAD.

In the third search, MAD_3 is found at the outer search point of large hexagon search shape. Three new search points are emerged to form a new large hexagon search shape and MAD_3 is repositioned at the newly form large hexagons search shape. The new coordinate of MAD_3 is (+7, +14).

All the points surrounding the MAD_3 are evaluated again (assigned with number 2) and the optimal MAD is located at MAD_3 which is at the center of the large hexagon search shape. Then the small hexagon search shape will take place surrounding MAD_3 to conduct fine resolution search at the inner search. All the four points in the small hexagon search shape are evaluated again to find the best-matched motion in that block. So, the MAD_4 is the optimal MAD found in the fine resolution search and is coordinated at (+8, +15).

The final coordinate is considered the best-matched motion vector coordinate in the block of the current frame. This process is repeated in every single frame to predict the best-matched motion estimation of a current frame.

The preliminary development of FUHS16 technique is described in this section. The FUHS16 technique is then used as a baseline to enhance or develop our next algorithm. The FUHS16 algorithm is simulated to obtain the motion vector estimation search point, performance analysis compare to the other superior algorithms. The obtained results are analysed and the algorithm has been improved with some changes. These changes will be further discussed in next section.

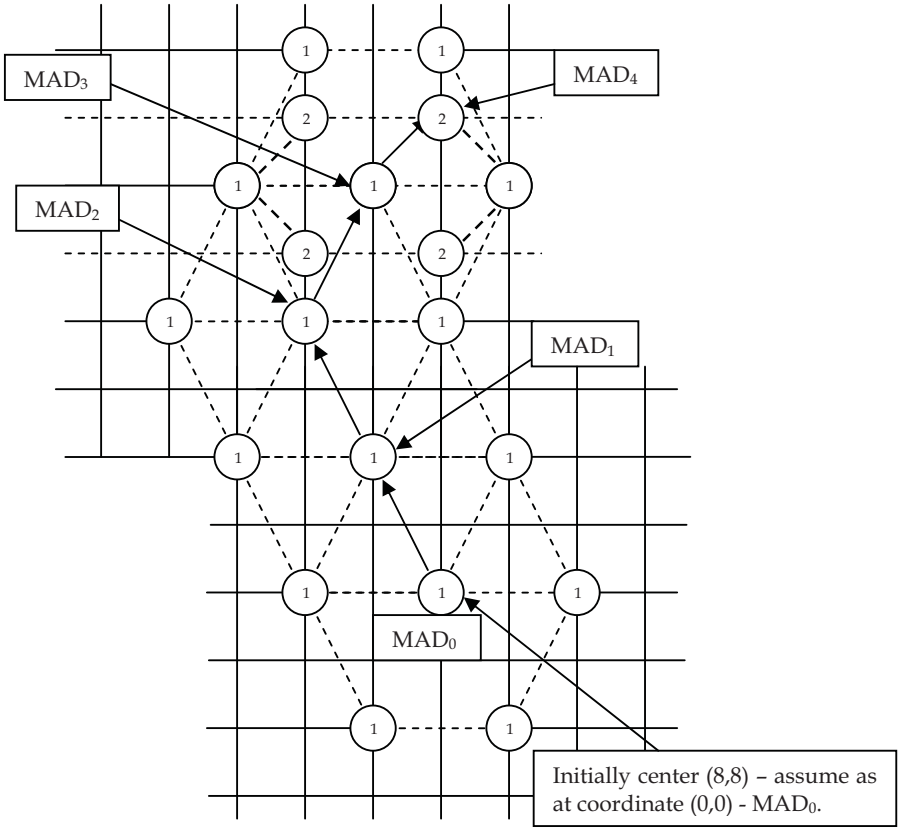


Fig. 2. Hexagon 3 new check points are formed and evaluated as new candidates to predict the motion vectors

3. Unrestricted Hexagon Diamond Search (UHDS16) algorithm

This section starts with some modifications from the FUHS16 algorithm. In this section, the UHDS16 technique is introduced. This technique is developed to have unrestricted search. To achieve this, a simple and efficient fast block-matching algorithm based on hexagon-diamond search shape is proposed. UHDS16 is designed uniquely with a large hexagon shape and shrink diamond search step (SDSS). Large hexagon is more unique to identify the motion vector in the small region of large hexagon shape. Finally, the shrink diamond search step is to locate the best-matched motion vector in the large hexagon small region. Experimental results show that the proposed UHDS16 algorithm significantly produces smaller computation complexity.

The speed and accuracy of the rood pattern based search algorithm are highly related to the size of the pattern. First step of the proposed method permits the algorithm to adapt itself to the content of motion. In most cases, adjacent blocks belong to the same moving object that has similar motions. Therefore, it is reasonable to predict the motion of current blocks from motion vectors of adjacent blocks.

UHDS16 is designed to have repetitive search in the small region of large hexagon search shape. The large hexagon search shape is to locate the best motion vector before switching to shrink diamond search step for the final best-matched motion vector coordinate.
The UHDS algorithms are implemented using two different block sizes. Initially, the UHDS16 algorithm is developed using the 16 × 16 block size with search windows 15 × 15 and then the same technique and ideology is used for 8 × 8 block size with search windows size 7 × 7. The difference between UHDS16 algorithm and UHDS8 algorithm is the block size.
Figure 1 illustrates the extraction of 8 × 8 pixels block size from a single frame. The FUHS16 algorithm block extraction procedure is applied in the UHDS8 algorithm. This means, that each frame will have 18 blocks horizontally and 22 blocks vertically.

4. Unrestricted Hexagon Diamond Search (UHDS16) and (UHDS8) algorithm search procedure

Based on the switching strategy with two different shape searches in Figure 3, we develop the following search methodology as depicted in Figure 4.
The UHDS algorithm employs two search procedures as depicted in Figure 3. Large hexagon is assigned with a signed number '1'. The hexagon shape procedure is to locate the final best-matched motion vector coordinate in the search area. The coarse hexagon shape continues to search till the motion vector found in the hexagon area is an optimal MAD point. This is then followed by the shrink diamond shape which checks all four points with number assigned with '2' in Figure 3. The four search points are evaluated and compared to the center point in order to locate the final best-matched motion vector coordinate.
Figure 4 describes the basics of the UHDS16 and UHDS8 search algorithm. The number of search points needed for UHDS16 and UHDS8 algorithm is 12. The large hexagon is confined with seven outer search points, while the SDSS is confined with five search points.

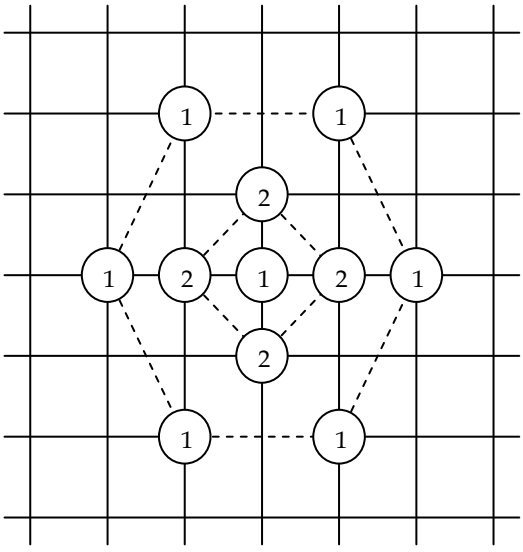


Fig. 3. Hexagon-Diamond Search Modelling Shape

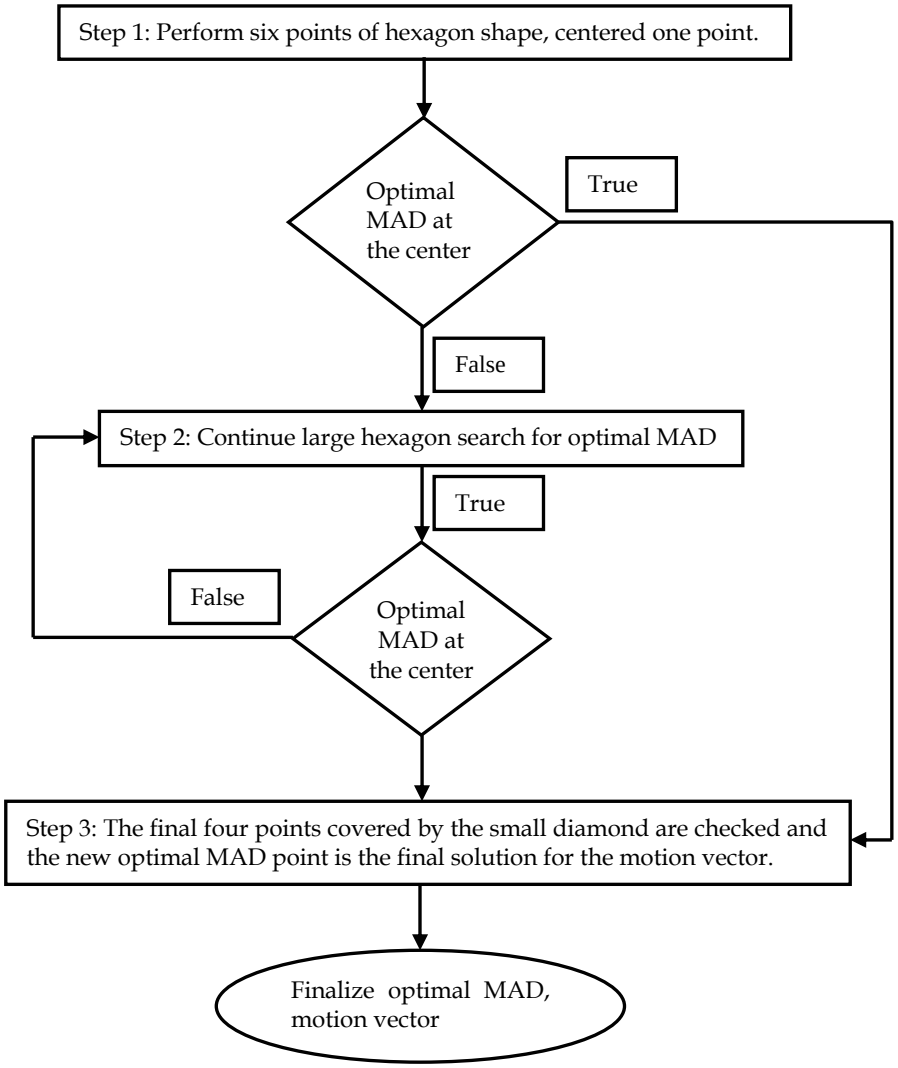


Fig. 4. Motion Estimation Search Procedure of Unrestricted Hexagon Diamond Search Algorithm

In the first step, seven checking points of the hexagon, within the search window around the motion vector predictor are compared to obtain the best motion vector. MAD is positioned at the center (0, 0) and served as a reference point to determine the final best motion vector in the SDSS. If the MAD point is found to be at the center of the hexagon search, then the hexagon search is switched to the small diamond search pattern for the final motion vector.
In the second step, MAD is the motion vector found by comparing the motion vector at step one MAD. If the MAD point is not located at the center and has best motion vector compared to the center one in step one, a new hexagon is formed and current MAD point

becomes the center. All the six points surrounding MAD points will be compared again to relocate the best motion vector before switching to the small diamond search step.
In the third step, SDSS will finalize the best motion vector and make comparison to determine the greatest motion vector amongst the four MAD points. Otherwise, the second step is repeated until the best optimal MAD distortion and best motion vector are found.

5. Peak Signal-to-Noise Ratio

The PSNR is a method used for objective quality comparison between two values for different reconstructed images. It gives one measure of quality which is applied in image processing perspective. PSNR analysis uses a standard mathematical model to measure the difference between two images in video sequence. It is commonly used in the development and analysis of algorithms, and comparing image quality between different compression systems. The PSNR Equation (3) is mostly used as a measure of quality of reconstruction within the compression of images. The peak in PSNR refers to the maximum pixel value. The following formula is used to calculate the PSNR value:

$$PSNR = 10\log_{10}\frac{255*255}{MSD\_pred}, \tag{3}$$

The PSNR measured usually is in decibels (dB). The higher the PSNR, the better quality will be produced for a compressed or reconstructed image.
Where

$$MSD\_pred = \frac{\sum_{M,N}\left[I_1(m,n) - I_2(m,n)\right]^2}{N \times N}. \tag{4}$$

$I_1(m,n)$ = pred_frame,

$I_2(m,n)$ = current_frame,

$N \times N$ = block pixels.

Equation (4) describes the cumulative squared error between the compressed and the reference image. If the value of Equation (4) is low, then the squared error accumulated will be low.

5. Experimental result and discussion for FUHS16, UHDS16 and UHDS8 algorithm

This section describes about the experimental results for FUHS16, UHDS16, and UHDS8 algorithms. Each algorithm is conducted using ten different video sequences with size of 176 × 144 pixels. Each video sequence is represented with ten video frames for simulation purposes. The experimental results are measured using the MATLAB and described accordingly based on quality performance in terms of PSNR points, computational complexity in terms of search points and elapsed processing time for each algorithm.

5.1 Results for FUHS16 and UHDS16 algorithm

In this section, results for FUHS16 and UHDS16 algorithm are presented. The presented figures represent the original frame and predicted frame of "Claire" – Slow Motion (Lee et

al., 2005), "News" – Slow Motion (Wu et al., 2010), "Mother" – Slow Motion (Yang et al., 2007), "Salesman" – Large Motion (Shilpa et al., 2010), "Container" - Slow Motion (Wu et al., 2010), "Coastguard" - Large Motion (Wu et al., 2010), "Foreman" – Medial Motion (Wang et al., 2010), "Table Tennis" – Large Motion (Chen et al., 2002), "Akiyo" – Slow Motion (Wang et al., 2008) and "Hall" – Slow Motion (Wu et al., 2010). Each of these video sequences have

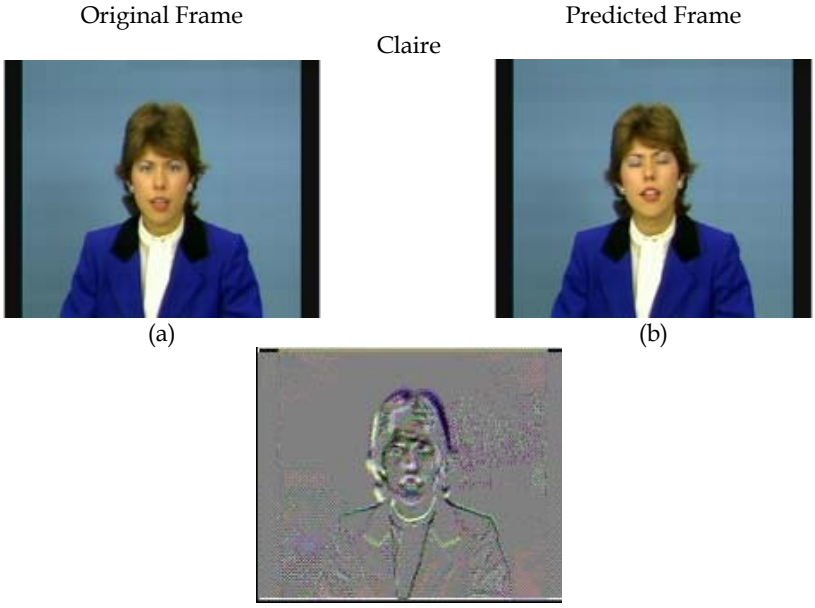
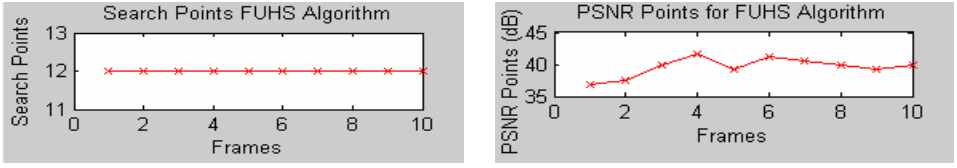
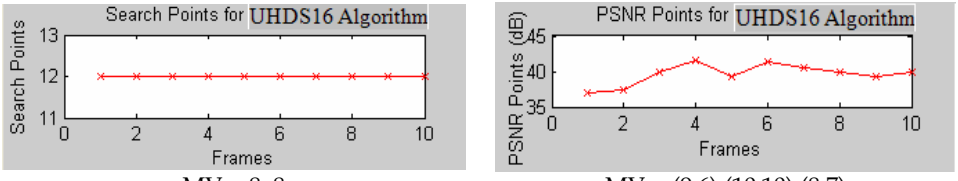


Fig. 5.1. (a) Claire Original Frame; (b) Claire Predicted Frame and (c) Claire Frame Difference



MV = 8, 8 MV = (9,5) (10,7)(8,6)

Fig. 5.2. Search Points and PSNR Points for FUHS16 Algorithm (Claire)



MV = 8, 8 MV = (9,6) (10,10) (8,7)

Fig. 5.3. Search Points and PSNR Points for UHDS16 Algorithm (Claire)

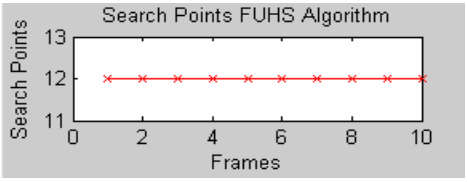
News

(a)

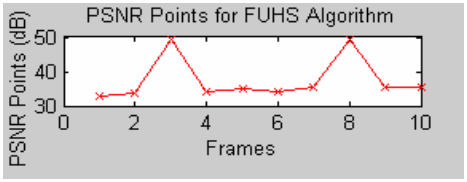
(b)

(c)

Fig. 5.4. (a) News Original Frame; (b) News Predicted Frame and (c) News Frame Difference

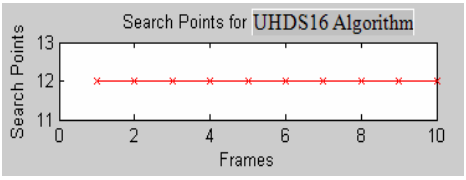


MV = (8,8)

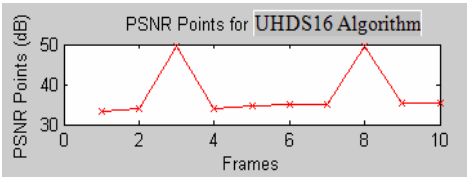


MV = (10,6)(10,7)(9,7)

Fig. 5.5. Search Points and PSNR Points for FUHS16 Algorithm (News)



MV = (8,8)



MV = (11,6) (8,9) (10,7)

Fig. 5.6. Search Points and PSNR Points for UHDS16 Algorithm (News)

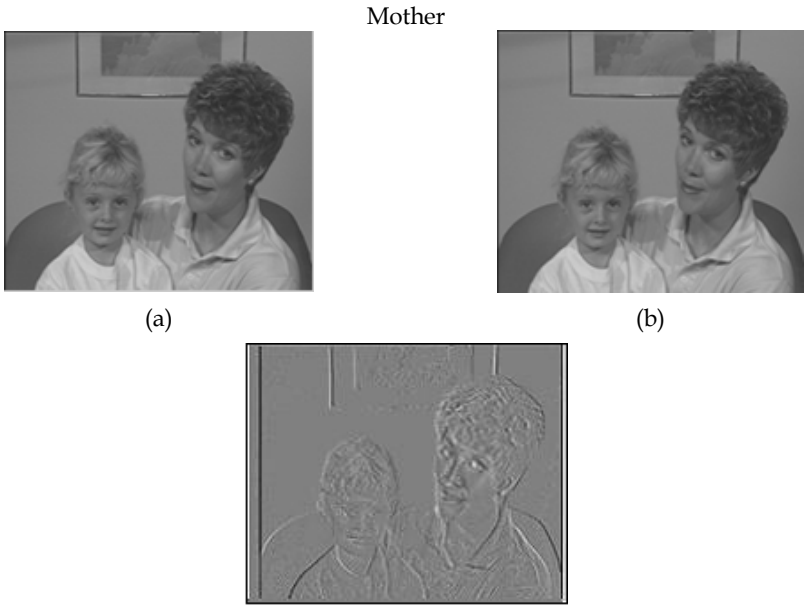
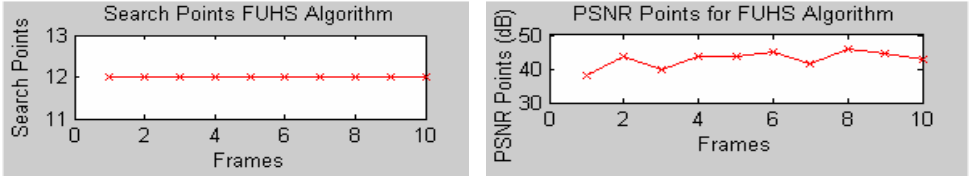
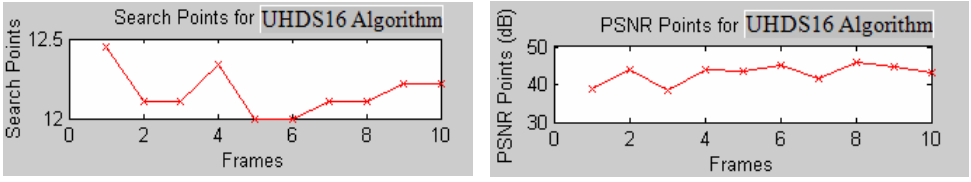


Fig. 5.7. (a) Mother Original Frame; (b) Mother Predicted Frame and (c) Mother Frame Difference



MV = (8,8) MV = (8,6)(10,7)(7,7)

Fig. 5.8. Search Points and PSNR Points for FUHS16 Algorithm (Mother)



MV = (8,8) MV = (8,6) (7,6) (7,7)

Fig. 5.9. Search Points and PSNR Points for UHDS16 Algorithm (Mother)

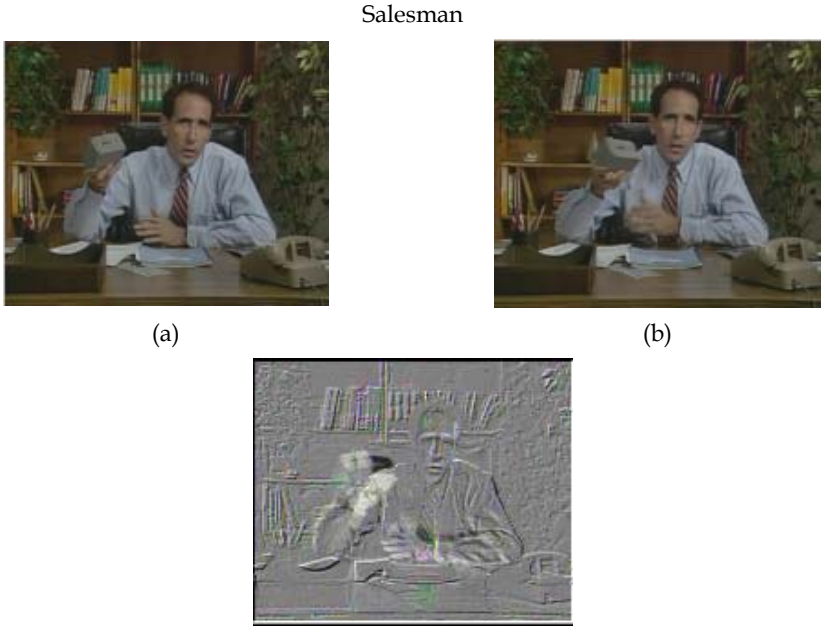
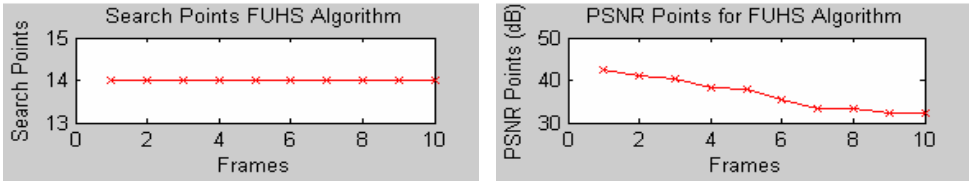
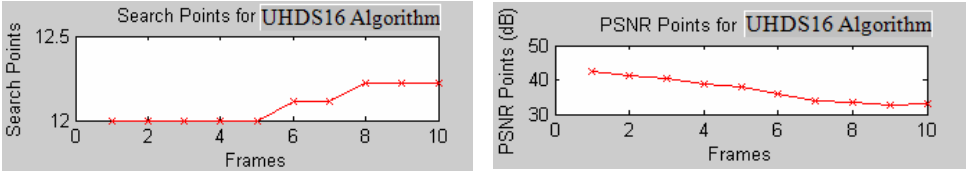


Fig. 5.10. (a) Salesman Original Frame; (b) Salesman Predicted Frame and (c) Salesman Frame Difference



MV = (8,8) MV =(9,5) (10,7) (8,6)

Fig. 5.11. Search Points and PSNR Points for FUHS16 Algorithm (Salesman)



MV = (8,8) MV =(9,5) (10,6) (8,6)

Fig. 5.12. Search Points and PSNR Points for UHDS16 Algorithm (Foreman)

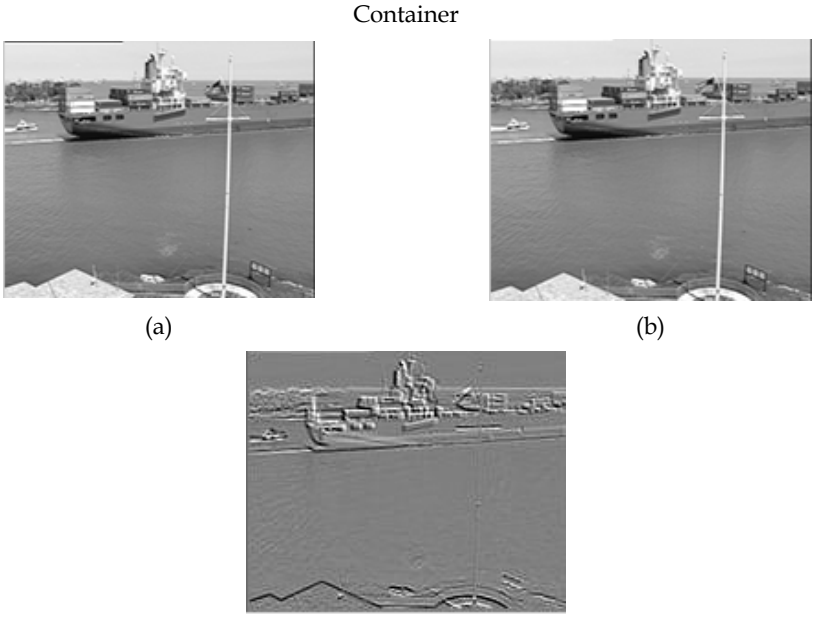
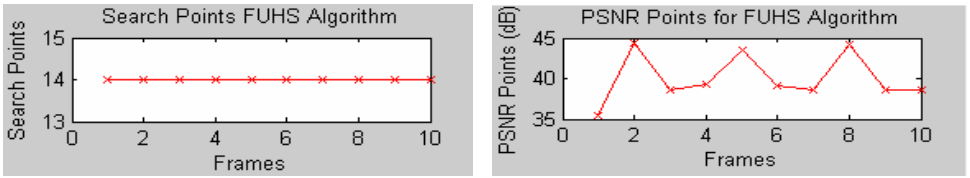
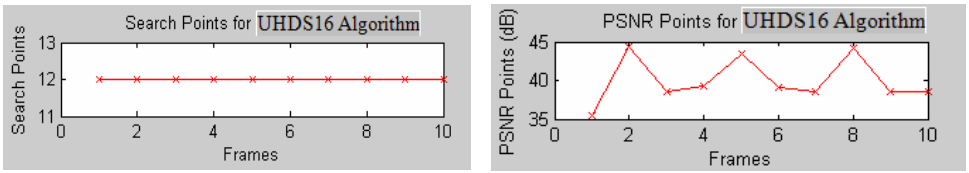


Fig. 5.13. (a) Container Original Frame; (b) Container Predicted Frame and (c) Container Frame Difference



MV = (8,8) MV = (9,6) (10,7) (8,7)

Fig. 5.14. Search Points and PSNR Points for FUHS16 Algorithm (Container)



MV = (8,8) MV = (9,6) (10,6) (8,7)

Fig. 5.15. Search Points and PSNR Points for UHDS16 Algorithm (Container)

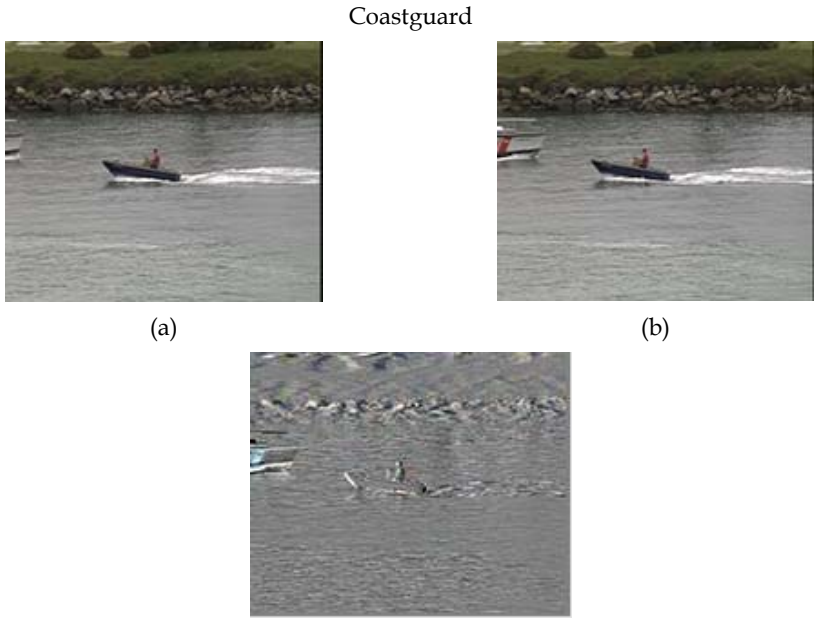
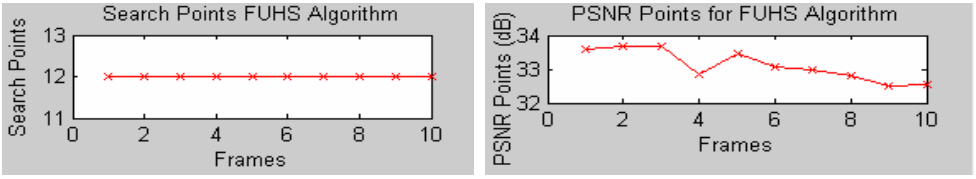
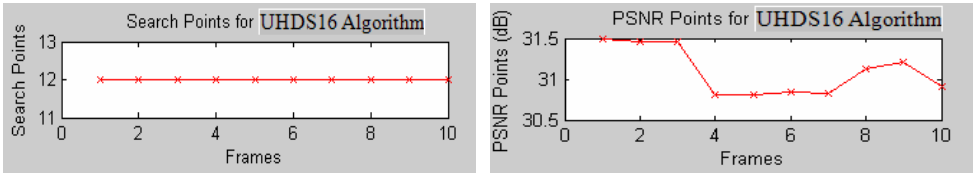


Fig. 5.16. (a) Coastguard Original Frame; (b) Coastguard Predicted Frame and (c) Coastguard Frame Difference



MV = (8,8) MV = (9,5)(10,7)(8,6)

Fig. 5.17. Search Points and PSNR Points for FUHS16 Algorithm (Coastguard)



MV = (8,8) MV = (9,5)(10,7)(8,6)

Fig. 5.18. Search Points and PSNR Points for UHDS16 Algorithm (Coastguard)

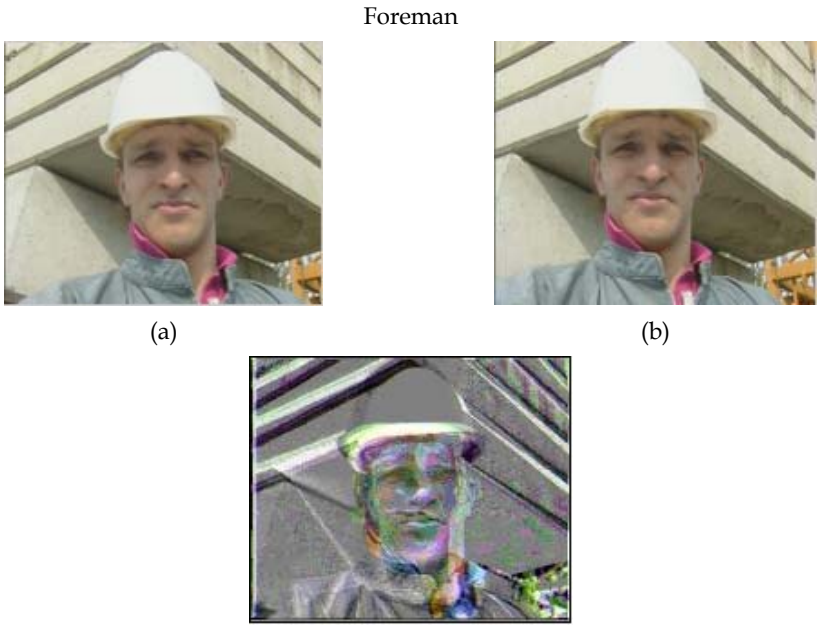
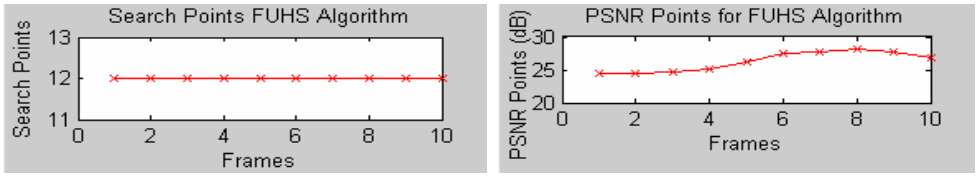
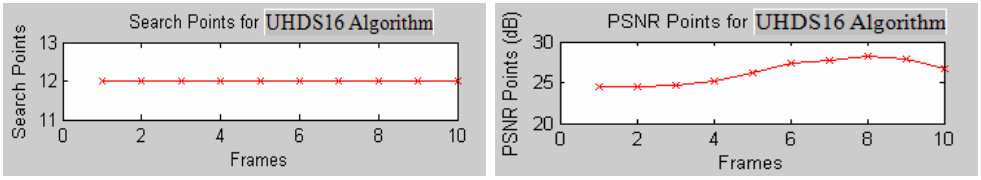


Fig. 5.19. (a) Foreman Original Frame; (b) Foreman Predicted Frame and (c) Foreman Frame Difference



MV = (8,8) MV= (8,6) (10,7) (7,7)

Fig. 5.20. Search Points and PSNR Points for FUHS16 Algorithm (Foreman)



MV = (8,8) MV= (8,6) (10,7) (7,7)

Fig. 5.21. Search Points and PSNR Points for UHDS16 Algorithm (Foreman)

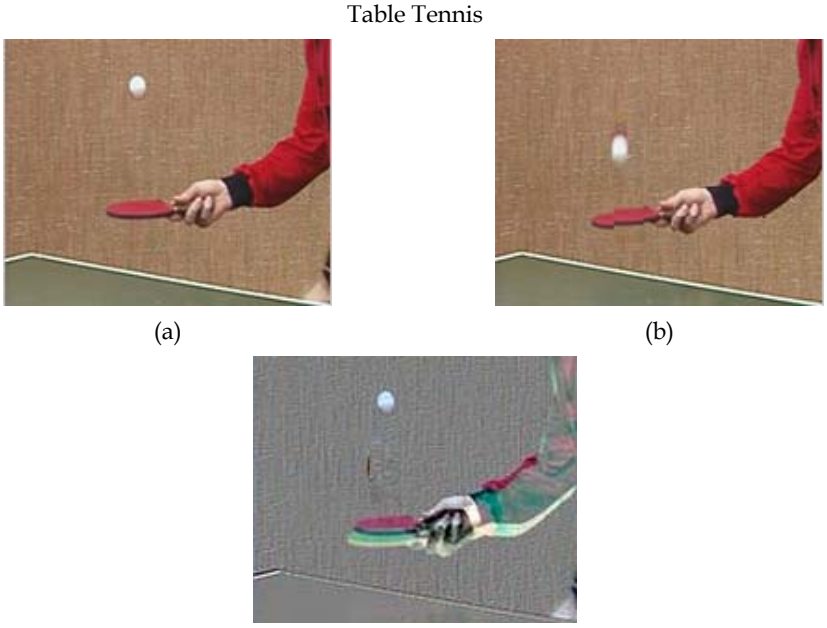
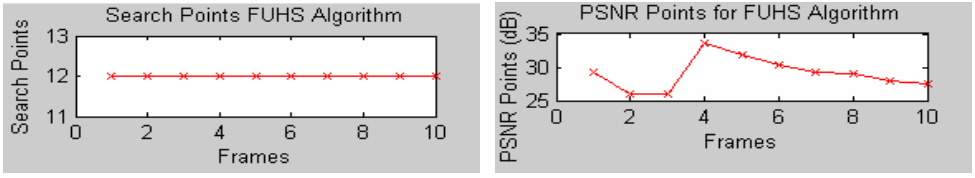
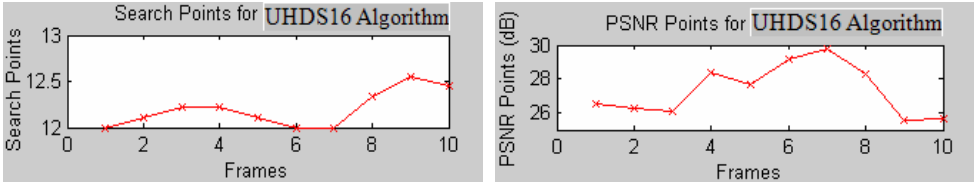


Fig. 5.22. (a) Table Tennis Original Frame; (b) Table Tennis Predicted Frame and (c) Table Tennis Frame Difference



MV = (8,8) MV = (10,7) (10,7) (9,8)

Fig. 5.23. Search Points and PSNR Points for FUHS16 Algorithm (Table Tennis)



MV = (8,8) MV = (5,14) (5,14) (4,15)

Fig. 5.24. Search Points and PSNR Points for UHDS16 Algorithm (Table Tennis)

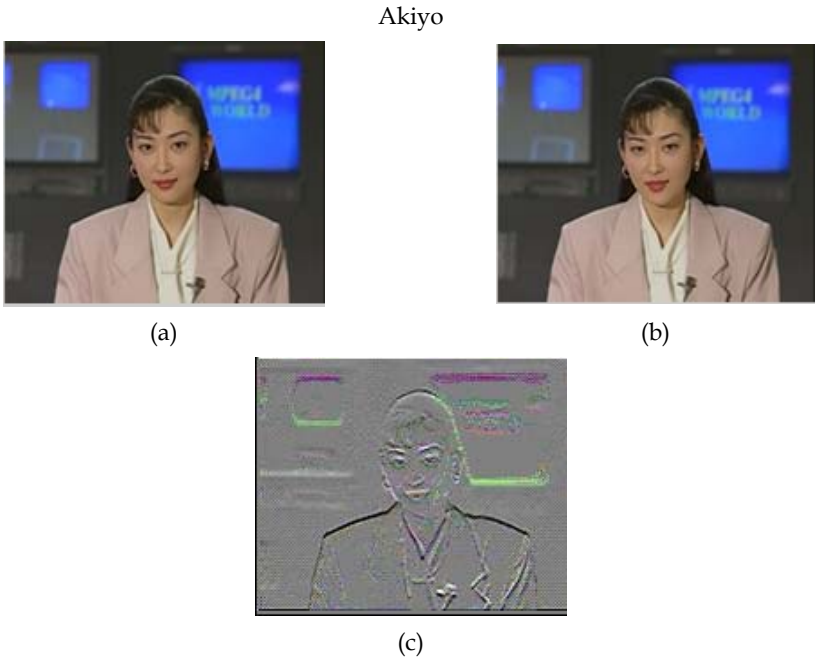
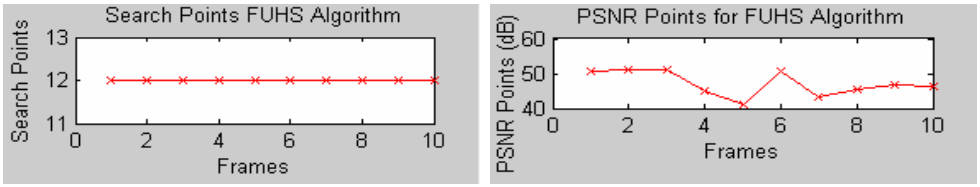
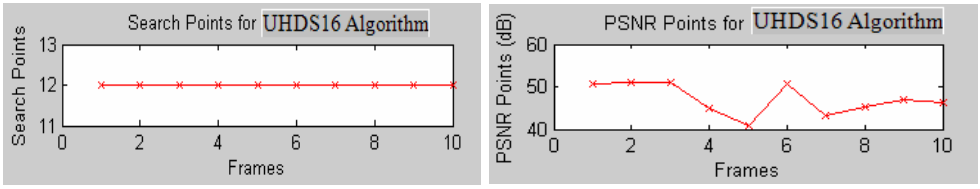


Fig. 5.25. (a) Akiyo Original Frame; (b) Akiyo Predicted Frame and (c) Akiyo Frame Difference



MV = (8,8) MV = (9,6) (10,7) (8,7)

Fig. 5.26. Search Points and PSNR Points for FUHS16 Algorithm (Akiyo)



MV = (8,8) MV = (9,6) (10,7) (8,7)

Fig. 5.27. Search Points and PSNR Points for UHDS16 Algorithm (Akiyo)

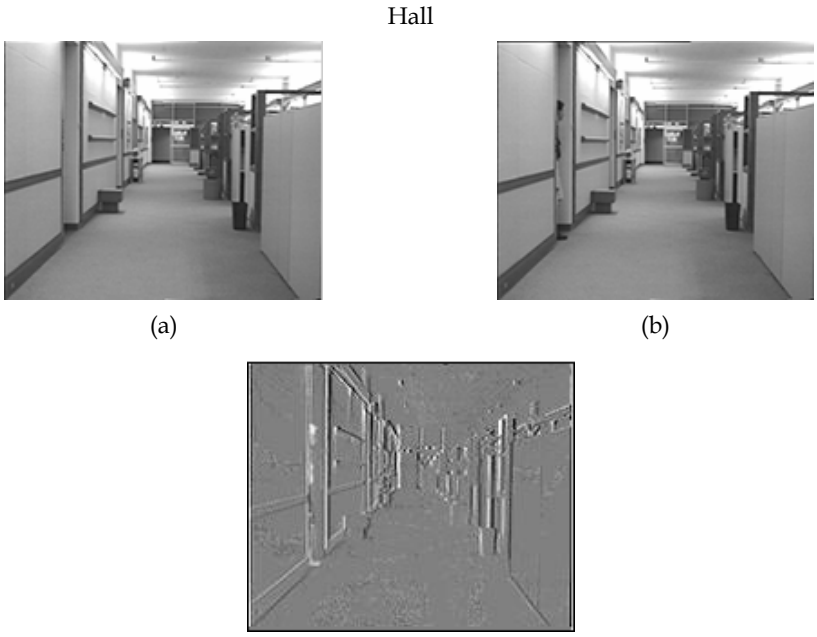
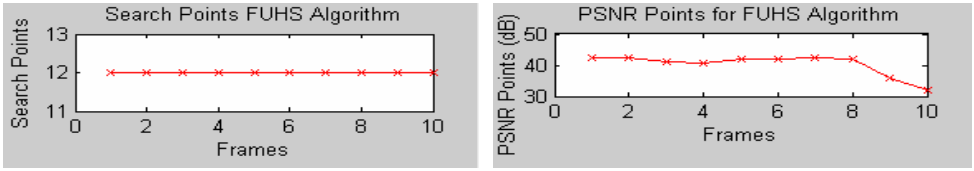
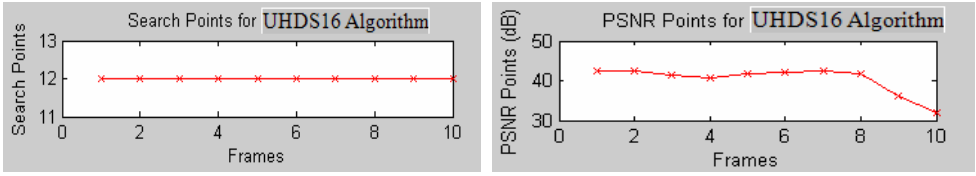


Fig. 5.28. (a) Hall Original Frame; (b) Hall Predicted Frame and (c) Hall Frame Difference



MV = (8,8) MV = (9,6) (10,7) (8,7)

Fig. 5.29. Search Points and PSNR Points for FUHS16 Algorithm (Hall)



MV = (8,8) MV = (9,6) (10,7) (8,7)

Fig. 5.30. Search Points and PSNR Points for UHDS16 Algorithm (Hall)

300 raw video frames, the video format of Quarter Common Intermediate Format (QCIF) and can be categories into different motion varying from small to large. Based on the presented results, original frame and predicted frame is not similar to the matched frame. This is to reveal that there is object translation between the original frame and predicted frame.
The motion vector coordinates shown are for ten video frames that have been conducted in FUHS16 and UHDS16 algorithm. Based on each vector coordinate, the motion represents the object translation between two frames. The first coordinate shows the first search of large hexagon shape, second vector coordinate represents the repetitive search of large hexagon shape and the best-matched motion vector coordinate represent the search for small hexagon shape. The repetitive search of large hexagon search is to gauge for the best-matched zero motion in the large hexagon region. While the small hexagon shape will finalized the best-matched motion vector coordinate among the four search points in the small region.
The results shown in Table 1 are the average PSNR points for FS, NTSS, TSS, DS, HEXBS, FUHS16 and UHDS16 algorithms. The average PSNR points values for all ten video sequences. These results are used for analysis purposes and for further improvement.

Average PSNR Points (dB)							
Algorithms	FS	TSS	NTSS	DS	HEXBS	FUHS16	UHDS16
Claire	39.93	38.91	38.91	39.04	39.20	39.23	39.89
News	37.65	36.95	37.26	37.34	37.35	37.21	37.57
Mother	43.21	42.70	42.70	42.79	42.75	42.70	42.92
Salesman	37.20	36.40	36.55	36.74	36.66	36.57	37.07
Container	40.02	40.02	40.02	40.02	40.02	40.02	40.02
Coastguard	33.11	31.00	31.03	31.09	31.03	31.00	33.11
Foreman	26.40	24.25	24.78	24.46	25.76	25.74	26.28
Tennis	32.47	26.39	28.00	27.52	28.95	29.26	29.07
Akiyo	47.06	47.06	47.06	47.06	47.05	47.05	47.06
Hall	40.30	40.29	40.30	40.30	40.29	40.29	40.30

Table 1. Average PSNR Points

Based on the average PSNR points, FUHS16 and UHDS16 algorithms produces similar PSNR points compared with the other algorithms. However, UHDS16 algorithm shows significant improvement in measuring the average PSNR point's compares with FUHS16 algorithm.
UHDS16 algorithm has outperformed all the other algorithms except for the FS algorithm in terms of PSNR point measurement. The overall average PSNR point's difference between UHDS16 and FS algorithm is approximately 0.31 dB. The importance to calculate the overall average PSNR points is to show the individual comparison of each algorithm.
Table 2 presents average search points for FS, TSS, NTSS, DS, HEXBS, FUHS16 and UHDS16 algorithms. The average search points calculated are for 990 blocks for ten video frames. As shown, FUHS16 algorithm and UHDS16 algorithm have the same number of average search points as HEXBS algorithm to obtain the similar algorithm performance with all the other algorithms. FUHS16 algorithm and UHDS16 algorithm have approximately improved 2 search points (17 percent) in terms of motion vector search points compared with DS algorithm. FS algorithm requires 213 extra search points (94.7 percent), TSS requires 13 extr

Average Search Points							
Algorithms	FS	TSS	NTSS	DS	HEXBS	FUHS16	UHDS16
Claire	225	25	27	14	12	12	12
News	225	25	25	14	12	12	12
Mother	225	25	25	14	12	12	12
Salesman	225	25	25	14	12	12	12
Container	225	25	25	14	12	12	12
Coastguard	225	25	25	14	12	12	12
Foreman	225	25	25	14	12	12	12
Tennis	225	25	28	14	12	12	12
Akiyo	225	25	25	14	12	12	12
Hall	225	25	26	14	12	12	12

Table 2. Average Search Points

a search points (52 percent) and NTSS requires 16 extra search points (57.1 percent) to gauge the similar motion vector coordinate as FUHS16 algorithm and UHDS16 algorithm.

Based on average PSNR points and average search points, FUHS16 algorithm and UHDS16 algorithm maintains the similar PSNR point's quality even though the search points are reduced 94.7 percent compare with FS algorithm. This also shows that reducing the search points can still maintain the similar performance and quality compare with the other algorithms. And, reducing the search points also reduces the computational complexity in terms of the MAD calculation (search points) and increases the best-matched motion vector coordinate estimation performances.

Table 3 shows the average elapsed processing time taken for FS, TSS, NTSS, DS, HEXBS, FUHS16 and UHDS16 algorithms. Based on the presented result, UHDS16 algorithm has the lowest average elapsed processing time compared with all the other algorithms. FUHS16 algorithm saved a small relative average elapsed processing time compared with HEXBS algorithm and DS algorithm. UHDS16 algorithm and FUHS16 algorithm approximately saved 14 percent, 23 percent and 55 percent of average elapsed processing time compared with NTSS, TSS and FS algorithms respectively. The result for UHDS16 algorithm shows, even though the search points are reduced and repetitive search pattern is developed, the average elapsed processing time still can be reduced.

Average Elapsed Processing Time (Sec)							
Algorithms	FS	TSS	NTSS	DS	HEXBS	FUHS16	UHDS16
Claire	3.28	1.86	1.70	1.63	1.55	1.53	1.47
News	3.14	1.84	1.77	1.63	1.58	1.61	1.45
Mother	3.58	2.01	1.71	1.68	1.59	1.62	1.38
Salesman	3.38	1.88	1.74	1.62	1.61	1.61	1.48
Container	2.86	2.02	1.92	1.72	1.70	1.64	1.53
Coastguard	3.23	1.81	1.77	1.64	1.62	1.64	1.46
Foreman	3.42	1.84	1.77	1.70	1.68	1.69	1.49
Tennis	3.61	2.00	1.74	1.70	1.59	1.49	1.43
Akiyo	3.48	1.81	1.61	1.59	1.57	1.57	1.46
Hall	2.96	2.08	1.79	1.65	1.55	1.54	1.43

Table 3. Average Elapsed Processing Time

5.2 Results for UHDS8 Algorithm

In order to evaluate the performance of UHDS8 algorithm, the UHDS8 algorithm is compared with the FS, TSS, NTSS, DS, and HEXBS algorithms in terms of the average PSNR points, computation complexity in terms of average search points and average elapsed processing time.

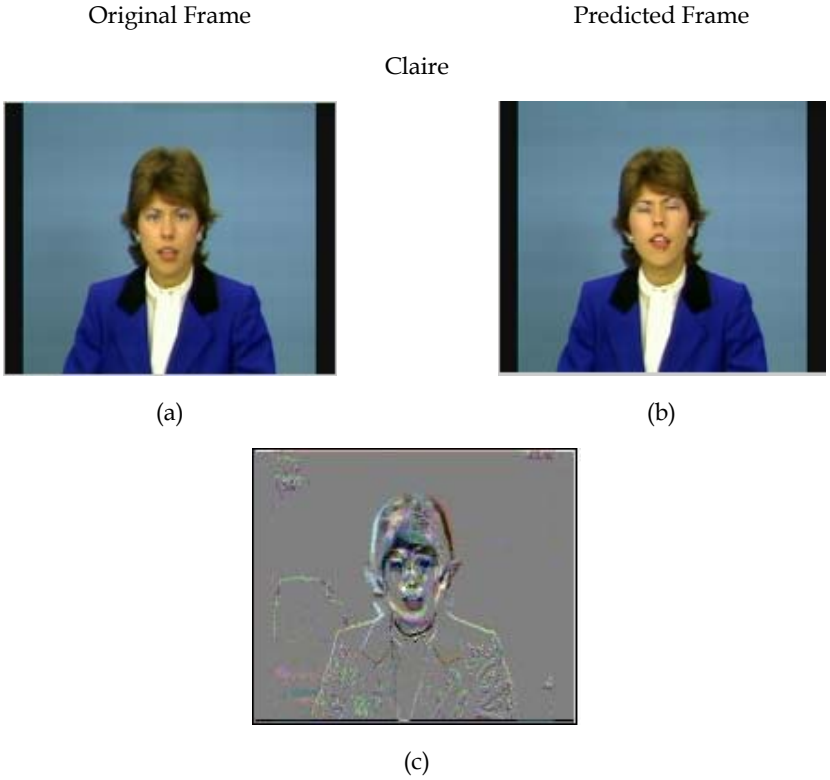
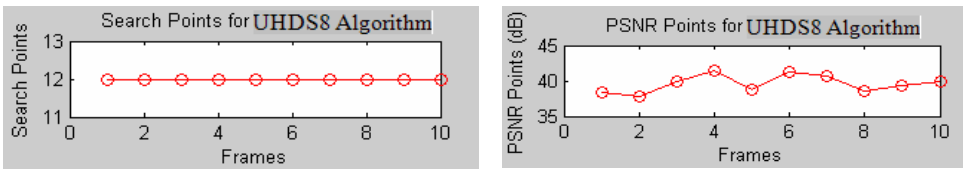


Fig. 5.31. (a) Claire Original Frame; (b) Claire Predicted Frame and (c) Claire Frame Difference



MV = (4,4) MV = (5,2) (6,4) (4,3)

Fig. 5.32. Search Points and PSNR Points for UHDS8 Algorithm (Claire)

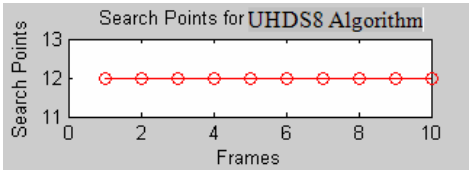
News

(a)

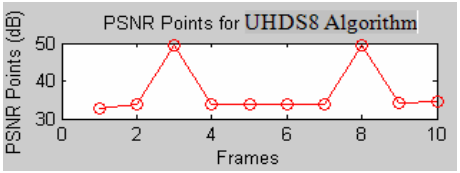
(b)

(c)

Fig. 5.33. (a) News Original Frame; (b) News Predicted Frame and (c) News Frame Difference



MV = (4,4)



MV = (5,2) (5,2) (2,4)

Fig. 5.34. Search Points and PSNR Points for UHDS8 Algorithm (News)

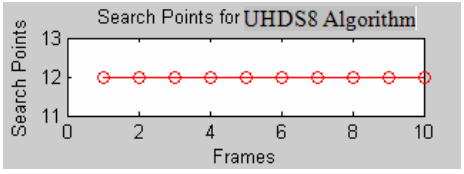
Mother

(a)

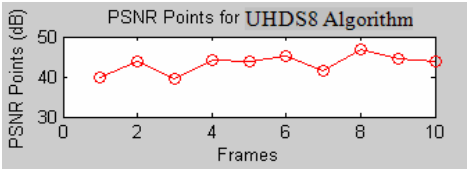
(b)

(c)

Fig. 5.35. (a) Mother Original Frame; (b) Mother Predicted Frame and (c) Mother Frame Difference



MV = (4,4)



MV = (4,2) (3,2) (3,3)

Fig. 5.36. Search Points and PSNR Points for UHDS8 Algorithm (Mother)

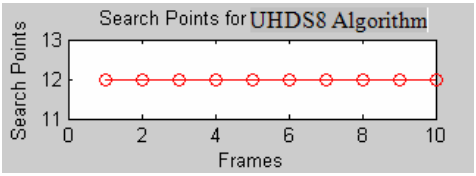
Salesman

(a)

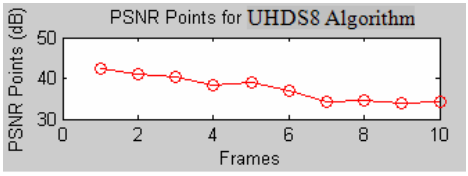
(b)

(c)

Fig. 5.37. (a) Salesman Original Frame; (b) Salesman Predicted Frame and (c) Salesman Frame Difference



MV = (4,4)



MV = (5,2) (5,2) (2,4)

Fig. 5.38. Search Points and PSNR Points for UHDS8 Algorithm (Salesman)

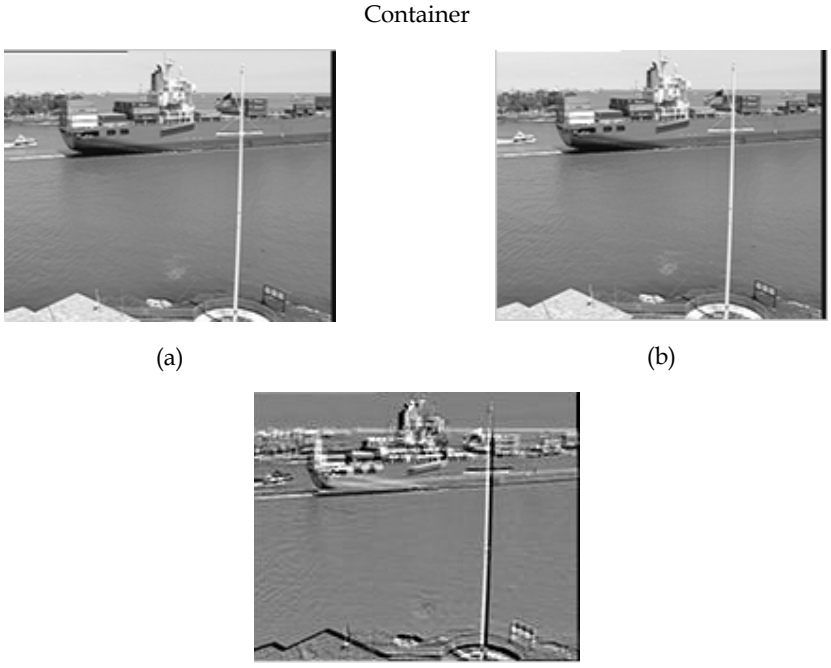
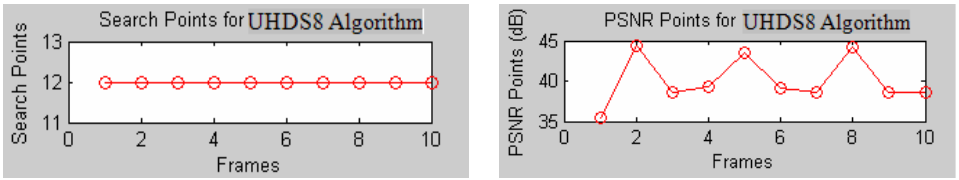


Fig. 5.39. (a) Container Original Frame; (b) Container Predicted Frame and (c) Container Frame Difference



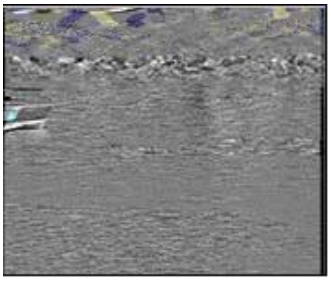
MV = (4,4) MV = (5,2) (5,3) (4,3)

Fig. 5.40. Search Points and PSNR Points for UHDS8 Algorithm (Container)

Coastguard

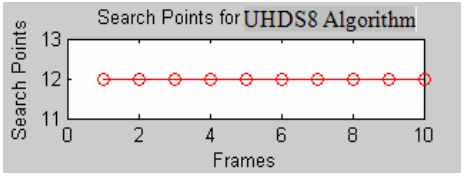
(a)

(b)

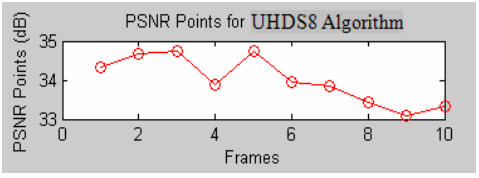


(c)

Fig. 5.41. (a) Coastguard Original Frame; (b) Coastguard Predicted Frame and (c) Coastguard Frame Difference



MV = (4,4)



MV = (5,2) (6,4) (4,3)

Fig. 5.42. Search Points and PSNR Points for UHDS8 Algorithm (Coastguard)

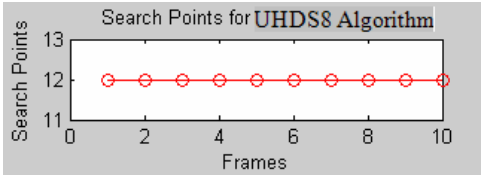
Foreman

(a)

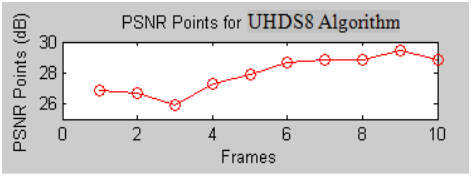
(b)

(c)

Fig. 5.43. (a) Foreman Original Frame; (b) Foreman Predicted Frame and (c) Foreman Frame Difference



MV = (4,4)



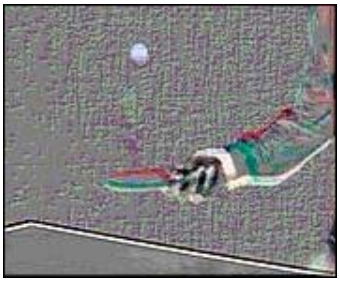
MV= (4,3) (5,-5) (3,4)

Fig. 5.44. Search Points and PSNR Points for UHDS8 Algorithm (Foreman)

Table Tennis

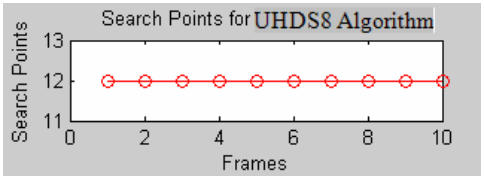
(a)

(b)

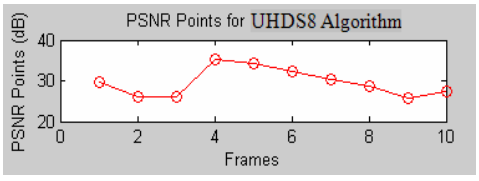


(c)

Fig. 5.45. (a) Table Tennis Original Frame; (b) Table Tennis Predicted Frame and (c) Table Tennis Frame Difference



MV = (4,4)



MV = (5,2) (5,2) (2,4)

Fig. 5.46. Search Points and PSNR Points for UHDS8 Algorithm (Table Tennis)

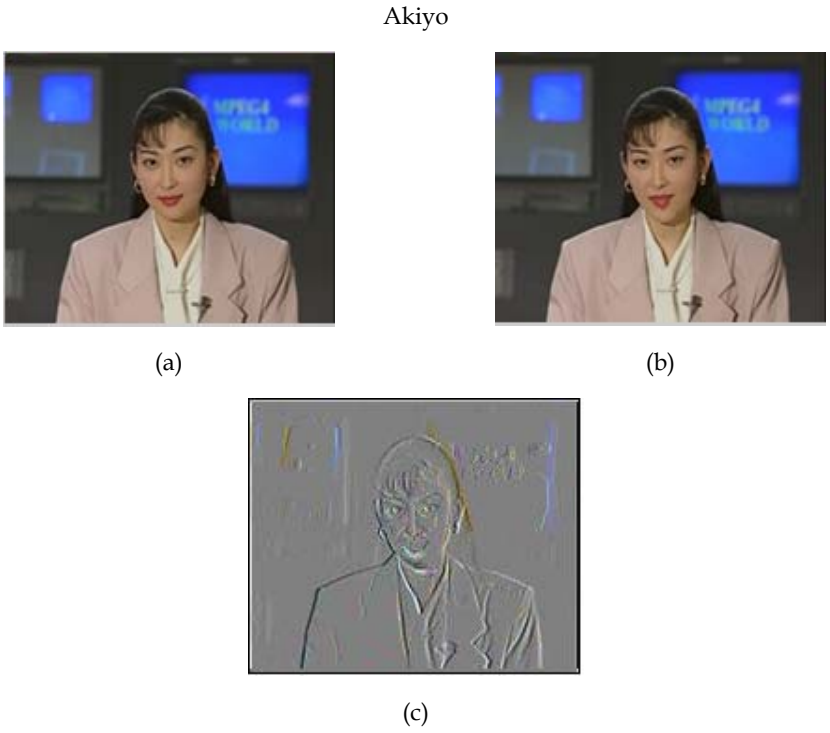
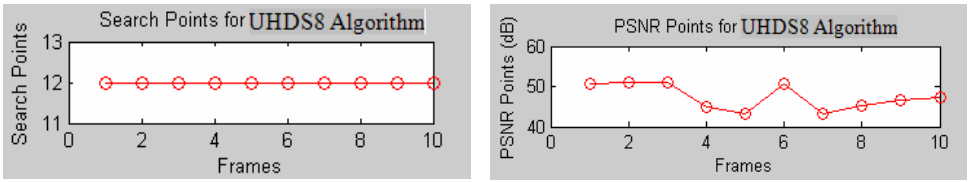


Fig. 5.47. (a) Akiyo Original Frame; (b) Akiyo Predicted Frame and (c) Akiyo Frame Difference



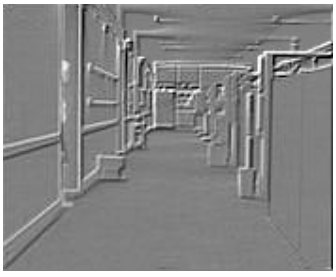
MV = (4,4) MV = (5,2) (6,4) (3,6)

Fig. 5.48. Search Points and PSNR Points for UHDS8 Algorithm (Akiyo)

Hall

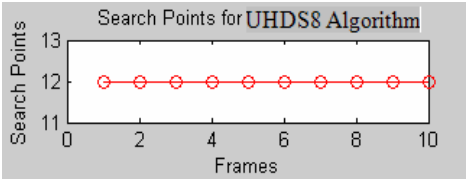
(a)

(b)

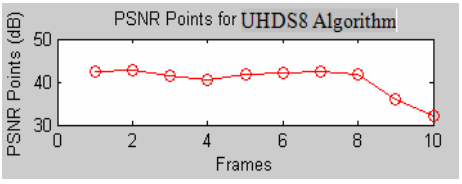


(c)

Fig. 5.49. (a) Hall Original Frame; (b) Hall Predicted Frame and (c) Hall Frame Difference



MV = (4,4)



MV = (4,2) (5,4) (4,3)

Fig. 5.50. Search Points and PSNR Points for UHDS8 Algorithm (Hall)

In this section, the proposed UHDS8 algorithm is compared with the popular FS, NTSS, TSS, DS, HEXBS and FUHS8 algorithms. In the first stage, ten video frames from each video sequence are used to simulate the coded algorithm. The results using UHDS8 algorithm and all the other algorithms are shown in Table 4 for average PSNR points, Table 5 for average search points and Table 6 for average elapsed processing time.

Average PSNR Point (dB)						
Algorithms	FS	NTSS	TSS	DS	HEXBS	UHDS8
Claire	40.46	38.91	38.91	39.47	39.48	39.76
News	38.47	37.46	36.95	37.69	37.72	37.73
Mother	43.88	42.72	42.69	42.89	43.22	43.29
Salesman	38.11	36.83	36.40	36.80	36.82	37.48
Container	40.03	40.02	40.02	40.02	40.02	40.03
Coastguard	33.95	31.16	30.10	31.16	32.98	33.94
Foreman	28.40	25.06	24.25	25.01	27.65	27.96
Tennis	32.64	27.10	26.40	28. 60	29.05	29.15
Akiyo	47.48	47.06	47.06	47.25	47.28	47.42
Hall	40.45	40.34	40.29	40.30	40.31	40.31

Table 4. Average PSNR Points

Table 4 shows the average PSNR points for FS, TSS, NTSS, DS, HEXBS, FUHS8 and UHDS8 algorithms. The FS algorithm has the most promising result among all the developed algorithms. This is because FS algorithm does brute search among all the search points. In Table 5.4, the average PSNR points shows that the UHDS8 algorithm improves the average PSNR points compared with the NTSS, TSS, DS, HEXBS, and FUHS8 algorithms. Analysis shows "Claire" video sequence have improved 0.13 dB of average PSNR point's compared with HEXBS algorithm and DS algorithm. While, "Coastguard" have improved approximately 3 dB of average PSNR point's compared with TSS, NTSS and DS algorithms. "Akiyo" and "Container" video sequences produces the similar average PSNR point's compared with all the other algorithms. The average PSNR points for "Salesman" and "Tennis" video sequences also improved approximately to 0.55 dB compared with all the other algorithms. This shows that UHDS8 algorithm can also produce similar average PSNR points compared with FS algorithm while locating the best-matched motion vector coordinate.

Based on the ten video sequences overall average PSNR points, UHDS8 algorithm and FUHS8 algorithm have slightly improved the performance compared with the NTSS, TSS, DS and HEXBS algorithms. The overall average PSNR points analysis shows that UHDS8 algorithm have outperformed all the other algorithms except for FS algorithm. The overall average PSNR point's difference between UHDS8 algorithm and FS algorithm is approximately 0.68 dB.

Referring to the Table 5, FUHS8, UHDS8 and HEXBS algorithms have the lowest average search points compared with all the other algorithms. FS and TSS algorithms have 49 and 25 fixed search points respectively. Meanwhile NTSS algorithm search point ranging from 25 to 33 and DS algorithm has 14 search points.

Based on the average search points obtained from Table 5, FS algorithm needs to perform extra numbers of MAD calculations in order to determine the optimal MAD point. This will lead to extra usage of memory and consume 52 percent of extra processing time to predict the similar image as UHDS8 algorithm. UHDS8 algorithm reduces search points and elapsed processing time to perform all the MAD calculations in order to estimate the MAD point.

Hence, this shows that UHDS8 algorithm have the ability to reduce the elapsed processing time while maintaining the global optimal point properties of the image. Besides that, the UHDS8 algorithm approximately can save up to 37 search points compared with FS algorithm, 2 search points compared with DS algorithm, 13 search points compared with TSS algorithm and 21 search points compared with NTSS algorithm. This shows that the UHDS8 algorithm requires less MAD calculation and save memory approximately up to 75.5 percent compared with all the other algorithms.

Table 6 shows the elapsed processing time compared with FS algorithm. UHDS8 algorithm has the lowest average elapsed processing time compared with TSS, NTSS, DS, and HEXBS algorithms. Thus, Table 6 reveals that UHDS8 algorithm approximately has saved 40 percent of processing time compared with FS algorithm. Meanwhile, TSS, NTSS, DS and HEXBS algorithms approximately saved 21 percent, 19 percent and 29 percent average elapsed processing time respectively.

Average search points						
Algorithms	FS	TSS	NTSS	DS	HEXBS	UHDS8
Claire	49	25	26.99	14.70	12	12
News	49	25	25.55	14.27	12	12
Mother	49	25	26.42	14.90	12	12
Salesman	49	25	25.16	14.20	12	12
Container	49	25	25.06	14.00	12	12
Coastguard	49	25	25.00	14.00	12	12
Foreman	49	25	25.37	14.22	12	12
Tennis	49	25	26.81	14.36	12	12
Akiyo	49	25	25.00	14.01	12	12
Hall	49	25	25.78	14.00	12	12

Table 5. Average Search Points

This shows that FS algorithm approximately consumes 31 percent extra computational complexity in terms of search points compared with UHDS8 algorithm. Even though the

average elapsed processing time produced by UHDS8 algorithm is almost similar as FUHS8 algorithm, but UHDS8 algorithm average elapsed processing time is slightly faster. UHDS8 algorithm approximately saved 3 percent of average elapsed processing time compared with UHDS8 algorithm. TSS algorithm and NTSS algorithm shows increment of 14 percent and 18 percent average elapsed processing time respectively.

Average Elapsed Processing Time (Sec)						
Algorithms	FS	TSS	NTSS	DS	HEXBS	UHDS8
Claire	2.84	1.86	1.65	1.53	1.34	1.21
News	2.77	1.75	1.63	1.59	1.37	1.19
Mother	2.92	1.91	1.71	1.67	1.37	1.12
Salesman	2.72	1.82	1.68	1.59	1.39	1.22
Container	2.93	1.94	1.82	1.65	1.47	1.28
Coastguard	3.01	1.89	1.80	1.67	1.42	1.22
Foreman	2.99	1.86	1.66	1.59	1.41	1.19
Tennis	3.11	1.81	1.61	1.52	1.35	1.23
Akiyo	2.97	1.80	1.59	1.54	1.32	1.17
Hall	2.89	1.98	1.69	1.55	1.36	1.20

Table 6. Average Elapsed Processing Time

6. Conclusion

Motion vector estimation is the processes which generates the motion vectors to determine how each motion compensated prediction frame is created from the previous frame. Motion estimation is basically extracting the difference between the reference frame compared with the current frame. Motion vectors are typically used to compress the frame by storing the changes in the current frame. The vectors are used to detect and track the motion and to find the information required in the current frame. In this thesis, the previously developed algorithms were investigated and new algorithms were to design to compare the performance with the discussed algorithms.

Five most widely known motion estimation algorithms such as the full search, three step search, new three step search, diamond search and hexagon based search algorithms were explicated. All of these algorithms are reported to be the baseline algorithms in block-matching motion estimation algorithm development. These algorithms are studied to understand the basis of block-matching motion estimation. The motion vector is determined using the motion estimation block prediction. The knowledge is used to develop the proposed algorithm.

The motion estimation technique is developed to detect the motion in the video sequences. Each of the video sequence applied in the proposed algorithms have motion varies from

slow to fast. All the discussed algorithms are simulated using all the different type's video motion. The purpose of testing each algorithm with different kind of motions is to test the algorithms robustness. This also will explain that each algorithm is suitable to perform motion vector estimation based on different type of motions.

Based on this research and the findings of this study with baseline models, FUHS16 algorithm is selected as the basic algorithm for the proposed algorithm. Other proposed algorithms are used for comparison with all the superior algorithms. Fewer search points, less computational complexity and time savings was consequently proposed.

Validation of results involved three categories which are image PSNR points, computational complexity in terms of search points and elapsed processing time. In this thesis, the proposed algorithms are developed to have similar result as FS algorithm while out performing the other TSS, NTSS, DS and HEXBS algorithms. Additionally, measuring the PSNR points of each proposed algorithm is to determine the efficiency of the proposed algorithm. Based on the accumulated results, the proposed algorithms which are FUHS16, FUHS8, UHDS16 and UHDS8 have improved the search point efficiency and effectively have saved the elapsed processing time. This leads to lesser computational complexity when motion estimation prediction is done.

The proposed algorithm shows favorable characteristics for use in a real-world system. The PSNR point's measurements are similar or equal to that achieved using FS algorithm motion estimation block. Furthermore, the PSNR points measurement achieved by the proposed algorithm has outperformed the TSS, NTSS, DS and HEXBS algorithms. As in section 5.2, the UHDS8 algorithm has very close PSNR points measurement as compared with FS algorithm. Besides that, UHDS8 algorithm has also saved 94.7 percent of the computational complexity in terms of search points. The UHDS8 algorithm has also saved approximately 57 percent of the elapsed processing time to estimate the best-matched block compared with the FS algorithm. The numbers of memory accesses required during operation were also reduced.

7. References

Chen, M. J., Chu, M. C., and Pan, C. W. (2002). Efficient motion-estimation algorithm for reduced frame-rate. *IEEE Transactions on Circuit and Systems for Video Technology*, 12(4), 269 – 275.

Lee, J. K. & Chung, K. D. (2005). Conversion Scheme DCT-Domain Transcoding of MPEG-2 to H.264/AVC. *International conference on image analysis and processing*, pp. 551-558, 3617, Italy, September, 2005, Springerlink, Cagliari.

Shilpa, P. M. & Sanjay, N. T. (2010). Fast Motion Estimation Using Modified Orthogonal Search Algorithm for Video Compression. *Journal of Signal, Image and Video Processing*, 4(1), 123-128.

Wang, P., Zheng, Z. & Li L. (2008). A video watermarking scheme based on motion vectors and mode selection. *International Conference on Computer Science and Software Engineering*, 5, 233-237.

Wu, H., Mark C., and Robert K. (2010). A study of video motion and scene complexity. [Online]. Available: ftp://ftp.cs.wpi.edu/pub/techreports/pdf/06-19.pdf [2010, July 14].

Yang, K. C., Clark C. G., Pankaj K. D. & Khaled El-M. (2007). Perceptual Temporal Quality Metric for Compressed Video. *Journal Multimedia*, Vol.9, No. 7, 1528-1535.