Transactions Letters

A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation

Jo Yew Tham, Surendra Ranganath, Maitreya Ranganath, and Ashraf Ali Kassim

***Abstract*— The widespread use of block-based interframe motion estimation for video sequence compression in both MPEG and H.263 standards is due to its effectiveness and simplicity of implementation. Nevertheless, the high computational complexity of the full-search algorithm has motivated a host of suboptimal but faster search strategies. A popular example is the three-step search (TSS) algorithm. However, its uniformly spaced search pattern is not well matched to most real-world video sequences in which the motion vector distribution is nonuniformly biased toward the zero vector. Such an observation inspired the new three-step search (NTSS) which has a center-biased search pattern and supports a halfway-stop technique. It is faster on the average, and gives better motion estimation as compared to the well-known TSS. Later, the four-step search (4SS) algorithm was introduced to reduce the average case from 21 to 19 search points, while maintaining a performance similar to NTSS in terms of motion compensation errors. In this paper, we propose a novel *unrestricted center-biased diamond search* (UCBDS) algorithm which is more efficient, effective, and robust than the previous techniques. It has a best case scenario of only 13 search points and an average of 15.5 block matches. This makes UCBDS consistently faster than the other suboptimal block-matching techniques. This paper also compares the above methods in which both the processing speed and the accuracy of motion compensation are tested over a wide range of test video sequences.**

***Index Terms*— Block matching, center-biased search strategy, diamond search pattern, fast motion compensation, video compression.**

I. Introduction

MOTION estimation is central to many interframe video coding techniques. Each frame in a typical video sequence is made up of some changed regions of the previous frame, except at scene cuts where the current frame is unrelated to the previous one. Furthermore, it is observed that the locally changed areas are usually small and restricted, especially for most low motion content videoconferencing and visual telephony sequences. The main objective of any motion estimation algorithm is thus to exploit the strong interframe correlation along the temporal dimension. If we can estimate the set of motion vectors that map the previous frame to the current frame, then we only need to code and transmit the motion vectors, and possibly the error frame associated with the difference between the motion-predicted and the current frames. Since the error frame has a much lower zero-order entropy than the current frame, fewer bits are needed to convey the same amount of information. Hence, compression can be achieved even after coding the motion vectors.

Having recognized the advantages of interframe coding as compared to intraframe coding of video sequences, many different motion estimation and motion compensation techniques were investigated and reported in the literature. Some of the more popular methods include block-matching algorithms (BMA), parametric/motion models, optical flow, and pel-recursive techniques. Among these methods, BMA seems to be the most popular method due to its effectiveness and simplicity for both software and hardware implementations. BMA is also used extensively in all current international video compression standards which include MPEG-1 [3], MPEG-2 [4], H.261 [1], and H.263 [5].

It is obvious that in using BMA, the most accurate strategy is the full-search (FS) method which exhaustively evaluates all possible candidate motion vectors over a predetermined neighborhood search window to find the optimum. The candidate that gives the best match for a given block distortion measure is chosen as the estimated motion vector. Nevertheless, this method has not been a popular choice because of the high computational cost involved. For example, a search window with a maximum motion of $\pm W$ in both the horizontal and vertical directions will require $(2W+1)^2$ candidate search points for each block. As a result, many computationally efficient variants such as the three-step search (TSS) [7], the two-dimensional (2-D) logarithmic search [6], the cross search [2], the conjugate directional search [11], and the dynamic search-window adjustment and interlaced search [8] were proposed. Although they are suboptimal in the sense that they are susceptible to being trapped in local optima, these faster variations are usually employed in practical applications.

Among these suboptimal BMA's, TSS [7] became the most widely used technique mainly because of its faster estimation. It consists of three evaluation steps—each step contains nine uniformly spaced search points which get closer after every step. The best candidate search point in the previous step becomes the center of the current step. Hence, TSS requires a fixed $(9+8+8)=25$ search points per block, which leads to a speedup ratio of 9 over the FS when $W=7$.

Manuscript received October 20, 1996; revised September 2, 1997 and March 30, 1998. This work was supported in part by the Wavelets Strategtic Research Programme, NUS, which is funded by the National Science and Technology Board and the Ministry of Education under Grant RP960 601/A. This paper was recommended by Associate Editor L.-G. Chen.

The authors are with the Department of Electrical Engineering, National University of Singapore (NUS), Singapore 119260.

Publisher Item Identifier S 1051-8215(98)05759-0.

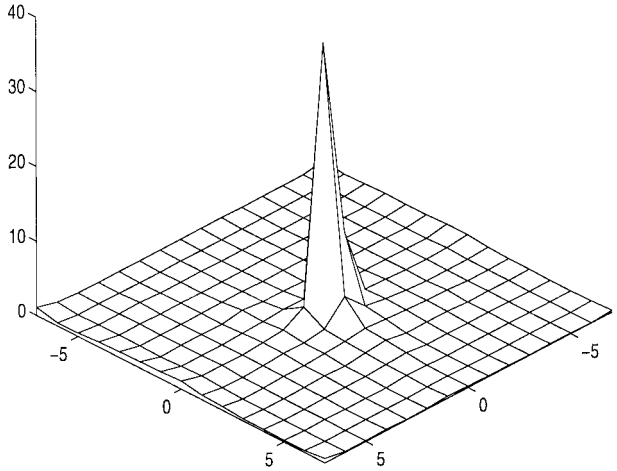

Fig. 1. Center-biased motion vector distribution characteristic of the fast-motion "Football" sequence.

The main drawback of TSS is the relatively large search pattern in the first step which renders it inefficient for finding blocks with small motions. This goes against the fact that most real-world sequences have a centrally biased motion vector distribution. This is depicted in Fig. 1, in which more than 80% of the blocks are stationary or quasi-stationary (within a central 3×3 area) even in the fast-motion "Football" sequence. Smaller-motion sequences such as "Miss America" and "Salesman" contain an average of 90 and 99% of the blocks having motion vectors within ± 3 pixels, respectively.

In order to exploit the characteristics of the center-biased motion vector distribution, a new three-step search (NTSS) algorithm [9] was introduced. It employs a center-biased search pattern in the first step by adding a smaller central eight-point pattern to that of the TSS. As a result, the worst case scenario of the NTSS will require $(25 + 8) = 33$ block matches. However, the NTSS also allows termination of the search after the first or second step. Using this technique, only 17 search points are needed for stationary blocks, and either 20 or 22 search points for quasistationary blocks (within ± 2 pixels). According to the results in [9], the speed of NTSS is within $\pm 18\%$ of TSS, but it gives almost consistently better motion estimates.

However, the computational requirement of NTSS may be higher than the TSS for sequences which have a lot of large motion vectors, for example, due to fast camera panning or accelerating objects in the scene. Recently, a new four-step search (4SS) algorithm [10] was proposed to speed up both the worst case and average-case computational requirements of NTSS. It also exploits the center-biased motion vector distribution characteristic by utilizing a nine-point search pattern on a 5×5 grid in the first step instead of a 9×9 grid as in the TSS. As a result of starting with a smaller search grid pattern, 4SS requires four search steps as compared to only three steps in both the TSS and NTSS for the same search window of $W = 7$. Nevertheless, simulation results in [10] show that the total number of candidate search points in 4SS actually ranges from the best case of 17 to the worst case of 27 points. According to [10], 4SS gives a speedup of six block matches for the worst case, and an average of two block matches less than the NTSS. More importantly, 4SS still manages to maintain motion estimation performance comparable to the NTSS, which in turn is better than the TSS.

In this paper, we propose a novel unrestricted center-biased diamond search (UCBDS) algorithm for suboptimal block motion estimation. Section II first presents the new diamond search pattern, and explains the algorithm development of UCBDS. Then a theoretical analysis is carried out to investigate the fast and effective search of UCBDS. Section III presents some simulation results of our proposed UCBDS scheme in comparison with the FS, TSS, NTSS, and 4SS. Finally, the conclusions are drawn in Section IV.

II. Unrestricted Center-Biased Diamond Search

In most practical applications such as video telephony, real-time software or hardware implementation of the video codec is indispensable. The fact is that a significant portion of the processing time of an interframe video encoder is dedicated to performing motion estimation. This really motivates the need for a fast and effective motion estimation algorithm. We have reviewed a few popular block-based motion estimation algorithms in the previous section. In this section, we will explain a more efficient, effective, and flexible solution to suboptimal block-based interframe motion estimation.

A. Algorithm Development of UCBDS

To begin any block-based motion estimation algorithm, each frame is first divided into blocks of size $N \times N$ pixels. Block sizes of 16×16 are used for MPEG-1 [3], MPEG-2 [4], H.261 [1], and H.263 [5].[1] Furthermore, in low and very low bit-rate video applications, the search for each block match is usually performed over a 15×15 search area,[2] requiring 225 possible candidate search points per block when the FS is used. As mentioned earlier, this is too computationally expensive. Hence, our main objective is concerned with choosing a suitable subset of these 225 points for a suboptimal version of the search algorithm.

Fig. 2(a) depicts a basic diamond search-point configuration used in UCBDS. It consists of nine candidate search points. This pattern is inspired by its compact[3] structure which is very suitable for exploiting the center-biased characteristic of motion vector distribution. Fig. 2(b) and (c) shows the positions of the diamond, with respect to the previous position, for the next search step along the diamond's vertex and face, respectively. Note that a maximum overlapping region is chosen so that there are either five or three new candidate search points to be evaluated at every next step. Maximum overlapping is required to minimize the number of search points at each step. However, UCBDS also attempts to reach

[1] Although H.263 has an advanced option to switch to four 8×8 blocks.

[2] However, as will be explained later, our UCBDS scheme is flexible enough to operate on any size search window.

[3] More compact structures are either the smaller five-point diamond or the nine-point square within ± 1 units. However, simulations show that a larger size search pattern is necessary to reduce the chances of being trapped in local optima. This is especially true for larger motion blocks whereby the central block motion field surface can be relatively smooth and misleading.

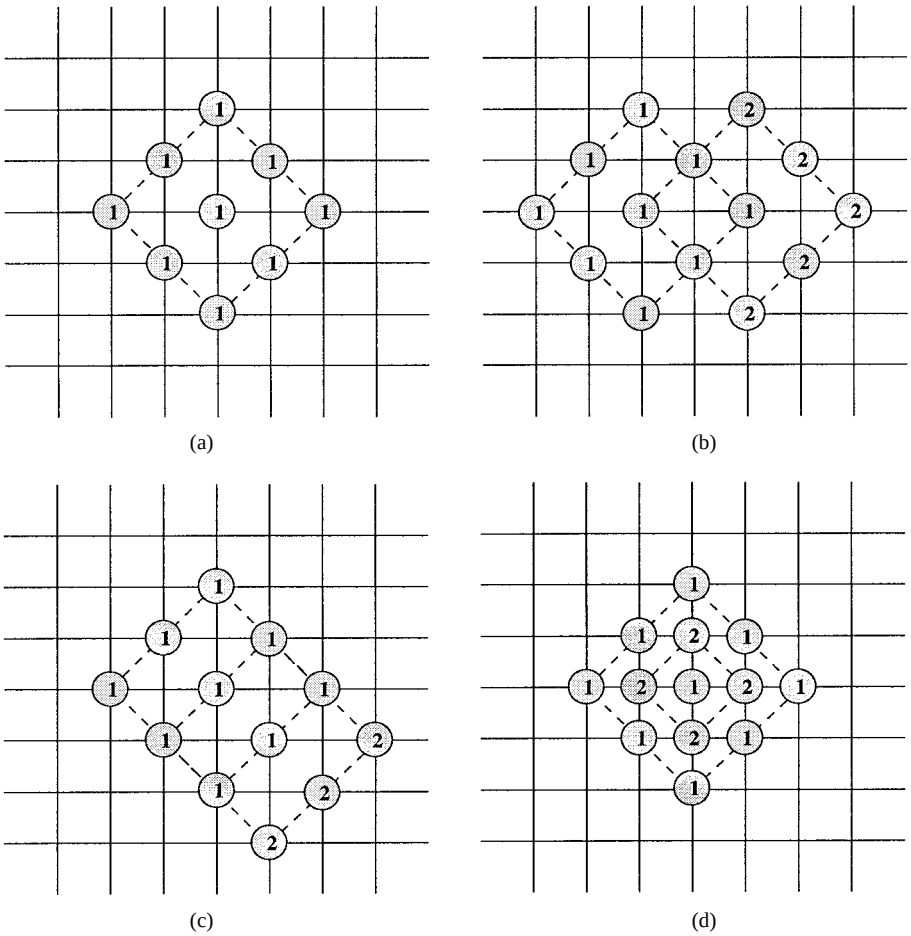

Fig. 2. Diamond search pattern. (a) Original diamond search-point configuration. (b) Next step along a diamond's vertex. (c) Next step along a diamond's face. (d) Final step with a shrunk diamond.

out as far as possible in each step to search for larger motion blocks. Such a strategy is critical to reduce its susceptibility to being trapped in local optima. Fig. 2(d) illustrates the final search step where the diamond is shrunk to only four new candidates for internal-point checking.

In order to describe the UCBDS algorithm, we have to define a block distortion measure (BDM) which forms the objective function for each block's evaluation. For simplicity, we choose the sum of absolute difference (SAD) given by

$$\mathrm{SAD}_{(p,q)}(m_x, m_y) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |f_t(p+i, q+j) - f_{t-\triangle t} (p+i+m_x, q+j+m_y)| \quad (1)$$

where $-W \le m_x, m_y \le +W$. Here, each $N \times N$ block with its upper left corner at a position (p, q) in the current frame f_t is estimated/predicted from a block of the same size in the previous (reconstructed) frame $f_{t-\triangle t}$ by means of a motion vector (m_x, m_y). Other BDM's such as sum of square error (SSE) and maximum pixel count (MPC) can also be used. However, as will be shown by the simulation results later, the choice of the simple SAD measure will give an equivalent estimation result like that obtained by using the SSE measure.

Fig. 3 illustrates an example of the unrestricted search path strategy using UCBDS. Assume that the true motion vector of the block is $(m_x, m_y) = (+7, -2)$. We begin at (0, 0) with an original diamond pattern marked as 1. For each of the nine candidate search points, the BDM is computed and compared. If the minimum BDM was found at (0, 0), the four points at

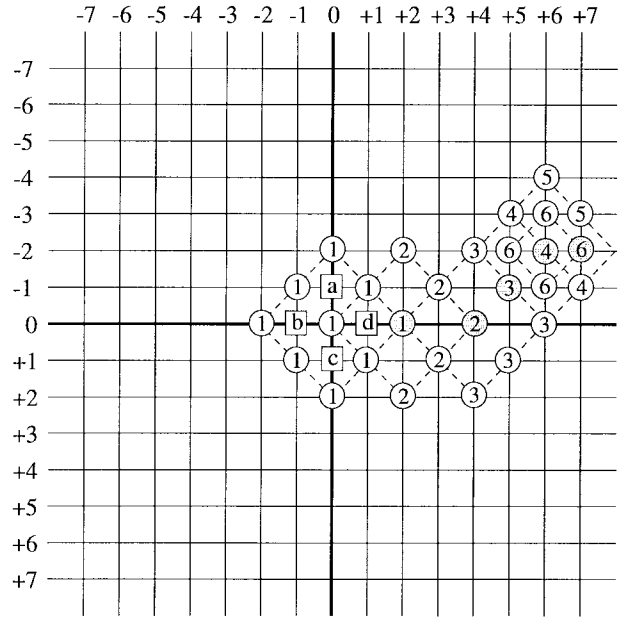

Fig. 3. Example of an unrestricted search path strategy using UCBDS to locate a motion vector at $(+7, -2)$. The best candidate search point at step 5 coincides with that of step 4; hence, a shrunk diamond is used for step 6.

a, b, c, and d are compared. Suppose that the lowest BDM is found at (+2, 0) We then proceed to the next step (marked as 2) in which the new diamond is now centered at (+2, 0). In this example, we require six search steps, where the shaded candidate points are the best points in each step. Notice that the best point in step 5 coincides with that of step 4. This signals us to shrink the diamond pattern for internal-point checking. Altogether, we have performed 28 block evaluations for this example.

The above unrestricted search path strategy using the UCBDS algorithm can be summarized as follows.

- ***Starting:*** The original diamond pattern [Fig. 2(a)] is placed at (0, 0), the center of the search window. The BDM is evaluated for each of the *nine* candidate search points. If the minimum BDM point is found to be at the center (c, c) of the diamond, proceed to **Ending**; otherwise, proceed to **Searching**.
- ***Searching:*** If the minimum BDM point in the previous search step is located at one of the four vertices [i.e., either $(c-2, c)$, $(c+2, c)$, $(c, c-2)$, or $(c, c+2)$], then proceed to **Vertex Search**. Else, if it is located at one of the four possible faces of the previous diamond [i.e., either $(c-1, c+1)$, $(c-1, c-1)$, $(c+1, c-1)$, or $(c+1, c+1)$], then proceed to **Face Search**.
  - — ***Vertex Search:*** The diamond pattern of Fig. 2(b) is used with the center of the new diamond coinciding with the lowest BDM point [i.e., updating the center (c, c)]. *Five* new candidate search points are evaluated.
  - — ***Face Search:*** The diamond pattern of Fig. 2(c) is used with the center of the new diamond coinciding with the lowest BDM point [i.e., updating the center (c, c)]. *Three* new candidate search points are evaluated.

    Note that any candidate point that extends beyond the search window is ignored. The minimum BDM is again identified. If the minimum BDM is found at (c, c), then proceed to **Ending**; otherwise, proceed to **Searching** to continue the next search step.
- ***Ending:*** The shrunk diamond pattern of Fig. 2(d) is used with the same center (c, c). Now, the final *four* internal points of the previous diamond are evaluated. Similarly, any internal candidate point that extends beyond the search window is also ignored. The candidate point that gives the lowest BDM is chosen as the estimated motion vector $(\hat{m}_x, \hat{m}_y)$. The current block's search process is completed. Proceed to **Starting** for the next block, if any.

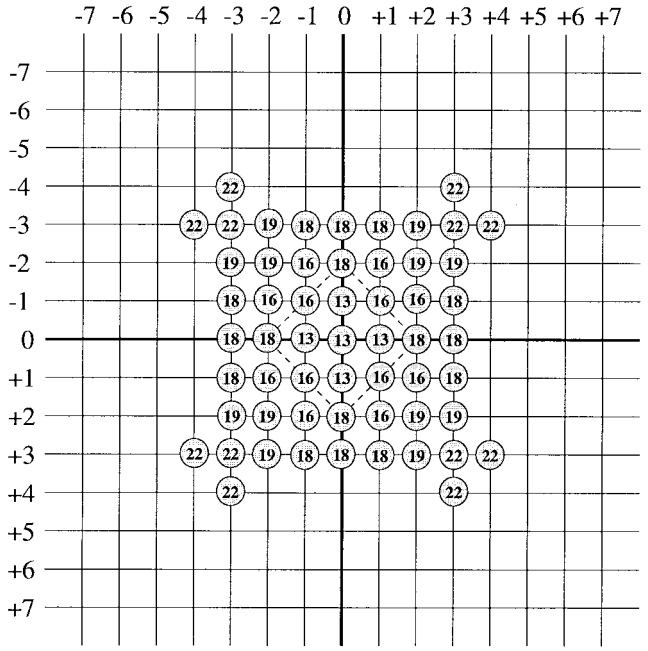

Fig. 4. Minimum possible number of search points for each motion vector location using UCBDS.

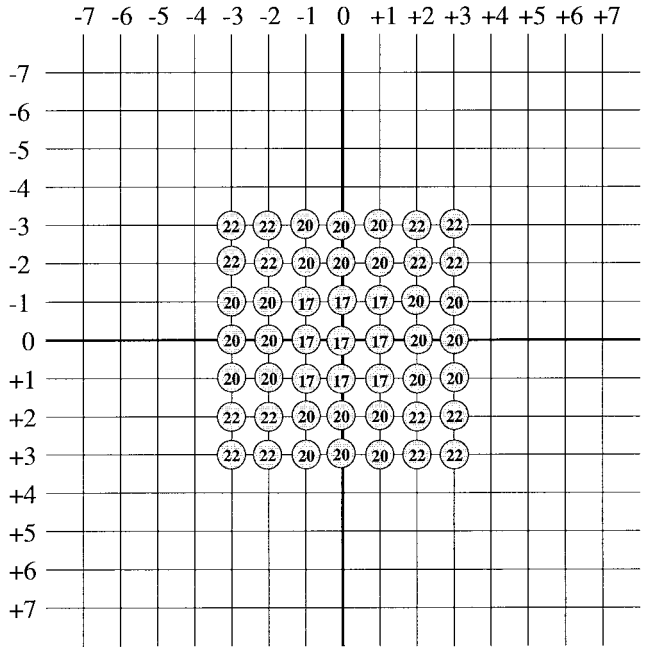

Fig. 5. Minimum possible number of search points for each motion vector location using 4SS.

B. Theoretical Analysis of UCBDS

We have explained earlier the motivations for having center-biased search algorithms such as the NTSS, 4SS, and UCBDS. This subsection aims to investigate theoretically why UCBDS is truly center biased, and how speed improvement can be obtained over other search algorithms. In particular, we are comparing UCBDS with the fast 4SS (comparisons with the TSS, NTSS, and FS are similar).

Our main argument in this analysis is based heavily on the observed center-biased motion vector distributions. To begin, we first analyze the minimum[4] number of search points N_s within a region of ± 3 pixels about the stationary motion vector $(0, 0)$. This is depicted in Fig. 4. Similarly, Fig. 5 illustrates

[4]Here, we refer to the *minimum* possible number of search points needed to conclude that a candidate point (m_x, m_y) is the estimated motion vector. In practice, the same motion vector may need more than this minimum value. Actually, it depends heavily on the gradient of the block motion field surface. For example, the motion vector at $(0, +1)$ will need 13 block matches if $(0, 0)$ was chosen, but it will need 18 block matches if $(0, +2)$ was chosen instead in the first search step. The same problem also plagues all other block-based search algorithms.

TABLE I
PERFORMANCE COMPARISONS (PER 16 × 16 BLOCK) OF FS, TSS, NTSS, 4SS, AND UCBDS USING DIFFERENT VIDEO SEQUENCES; THE AVERAGE NUMBER OF SEARCH POINTS PER BLOCK WOULD BE DIRECTLY PROPORTIONAL TO CPU TIME

Using "Trevor White" Sequence

	Search points				Avg. BDM		Avg. dist.(x10^{-2})		Avg. prob.	
	Min.	Max.	Avg.	Speedup	SSE	SAD	SSE	SAD	SSE	SAD
FS	225	225	225	1.00	6089	788	–	–	–	–
TSS	25	25	25.0	9.00	6535	802	12.8	11.5	0.958	0.961
NTSS	17	33	18.2	12.3	6178	791	5.91	6.41	0.981	0.979
4SS	17	27	17.5	12.9	6389	800	9.95	10.1	0.962	0.964
UCBDS	13	36	13.8	16.3	6212	792	6.91	7.21	0.978	0.978

Using "Flower Garden" Sequence

	Search points				Avg. BDM		Avg. dist.(x10^{-1})		Avg. prob.	
	Min.	Max.	Avg.	Speedup	SSE	SAD	SSE	SAD	SSE	SAD
FS	225	225	225	1.00	32553	1585	–	–	–	–
TSS	25	25	25.0	9.00	40821	1790	6.14	4.96	0.804	0.823
NTSS	17	33	22.6	9.96	34205	1620	1.83	1.62	0.925	0.931
4SS	17	27	20.1	11.2	38708	1748	4.17	3.49	0.840	0.852
UCBDS	13	33	17.8	12.6	33781	1619	1.69	1.47	0.947	0.950

Using "Football" Sequence

	Search points				Avg. BDM		Avg. dist.(x10^{-1})		Avg. prob.	
	Min.	Max.	Avg.	Speedup	SSE	SAD	SSE	SAD	SSE	SAD
FS	225	225	225	1.00	70035	2395	–	–	–	–
TSS	25	25	25.0	9.00	74115	2442	6.75	5.47	0.864	0.890
NTSS	17	33	23.2	9.69	73393	2440	6.18	5.48	0.871	0.886
4SS	17	27	20.1	11.2	74195	2459	7.30	6.99	0.867	0.877
UCBDS	13	40	18.3	12.3	74475	2461	7.53	7.05	0.888	0.896

the corresponding minimum N_s for 4SS over the same region. It can easily be observed that, within this same region, UCBDS gives lower values of N_s as compared to 4SS. Furthermore, UCBDS covers a slightly larger area for $N_s \leq 22$ search points. However, beyond the ± 3 region, it is noted that 4SS can become more efficient. To get a better picture of the gain in N_s, we subtract the corresponding candidate points of UCBDS from 4SS over this region. By doing so, we can obtain a saving of as high as four block matches per block.

To further quantify this gain in N_s for block estimation, we define the following probabilities of occurrence:

- P_0—probability of stationary blocks [i.e., the motion vector is (0, 0));
- P_1—probability of quasi-stationary blocks within ± 1, but *excluding* (0, 0);
- P_2—probability of quasi-stationary blocks within ± 2, but *excluding* the ± 1 region at the center;
- P_3—probability of quasi-stationary blocks within ± 3, but *excluding* the ± 2 region at the center;
- P'—probability of blocks in the region where $4 \leq |m_x|, |m_y| \leq W$.

By taking the average of the differences in N_s between 4SS and UCBDS over each of the above regions, the statistical average gain of UCBDS over 4SS can be represented as

$$\text{gain in } N_s = 4P_0 + \left(\tfrac{20}{8}\right)P_1 + \left(\tfrac{52}{16}\right)P_2 + \left(\tfrac{48}{24}\right)P_3 + nP' \tag{2}$$

where $P' = 1 - (P_0 + P_1 + P_2 + P_3)$, and n is some negative number. Suppose further that we assume a uniform probability distribution over the ± 3 region at the center, and that no motion vectors lie outside of this region. Then from (2), we will have a uniformly distributed average gain of

$$\text{uniform gain in } N_s = 0.25 \times (4 + 2.5 + 3.25 + 2) = 2.94 \text{ search points per block.} \tag{3}$$

However, observations from most real-world sequences show very peaked probabilities around P_0 and P_1, as depicted in Fig. 1. This means that an average gain of more than 2.94 search points per block can be expected. More simulation results later will justify this statement. At the other extreme, if all blocks are stationary or have motion vectors of either (0, −1), (0, +1), (+1, 0), or (−1, 0), then we can have the maximum possible gain of four search points, or a 31% speed improvement, per block over the fast 4SS.

TABLE II
PERFORMANCE OF UCBDS VERSUS OTHER SEARCH TECHNIQUES USING THE SAD MEASURE FOR THE "FOOTBALL" SEQUENCE

	TSS	NTSS	4SS	FS
Percentage Increase in Distortion	0.78	0.85	0.08	2.68
Percentage Speed Improvement	36.7	26.8	9.84	1130

III. SIMULATION RESULTS AND COMPARISONS

As highlighted earlier, only the *minimum* number of search points per block was considered in the theoretical analysis. In practice, this may not always be the case. This section, therefore, aims to investigate the actual experimental performance of UCBDS. In all of our simulations, the SAD block

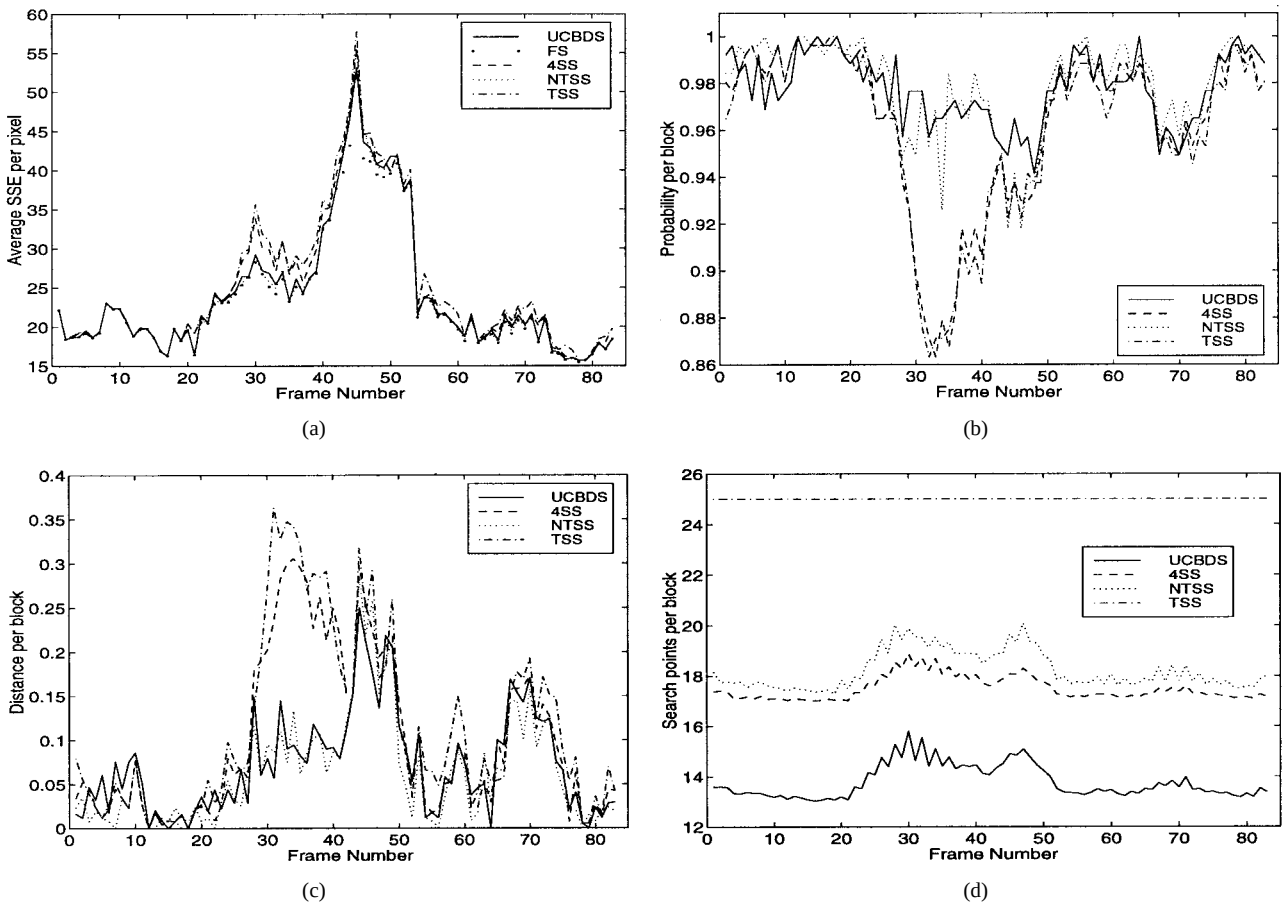

Fig. 6. Performance comparisons of FS, TSS, NTSS, 4SS, and UCBDS over all four test criteria using "Trevor White" sequence.

distortion measure, block size $N = 16$, and search window size $W = 7$ were used. For vigorous testing, a total of six sequences with different degrees and types of motion content was used; however, due to space limitations, we will only present the results of three representative sequences. First, we used 90 frames of the "Trevor White" sequence, which is a typical videoconferencing scene with limited object motion and a stationary background. Second, we chose 100 frames of the "Flower Garden" sequence, which consists mainly of stationary objects, but with a fast camera panning motion. Third, we selected 80 frames of the "Football" sequence, which contains large local object motion.

We compared the UCBDS against four other block-based motion estimation methods—FS, TSS, NTSS, and 4SS—using the following four test criteria.

1) **Average SSE per pixel**—This shows the magnitude of distortion per pixel; using SAD for the BDM gives similar results.
2) **Probability of finding true motion vector per block**—This gives the likelihood of the suboptimal predicted block motion vectors to be the same as those found using the optimum FS; this also provides an indication of the susceptibility of each suboptimal search method being trapped in local optima.
3) **Average distance from true motion vector per block**—This measures the Euclidean distance of a block's predicted motion vector from that obtained using FS.
4) **Average number of search points N_s per block**—This provides an equivalent measure of the actual CPU run time, as justified below.

Using SAD in (1) as the BDM, we need $X = (N^2 Abs + (2N^2 - 1)Add)$ operations per search point. If each frame is partitioned into B blocks of $N \times N$ pixels each, then the total number of operations per frame is represented by $\Sigma_{b=1}^{B} N_s^{(b)} X$, where $N_s^{(b)}$ is the total number of search points of the bth block. Since we employed the same SAD measure and the same number of blocks B for each of the block-matching schemes, the average number of search points per block, therefore, provides exactly an equivalent measure of the actual CPU search time. Furthermore, measurements of CPU run times are highly dependent on system loads and algorithm implementation.

Table I summarizes the experimental performance of each search technique over the four test criteria, for both the SSE and SAD block distortion measures, using three representative sequences. The first column tabulates the search speed criterion,[5] whereby the minimum, maximum, average, and speedup factor with respect to FS are reported. It is worthwhile to note

[5]For conciseness, only the search speed results obtained using the SAD measure are shown as similar positively correlated results were obtained using the SSE block distortion measure. Furthermore, SAD is preferred because of its computational simplicity.

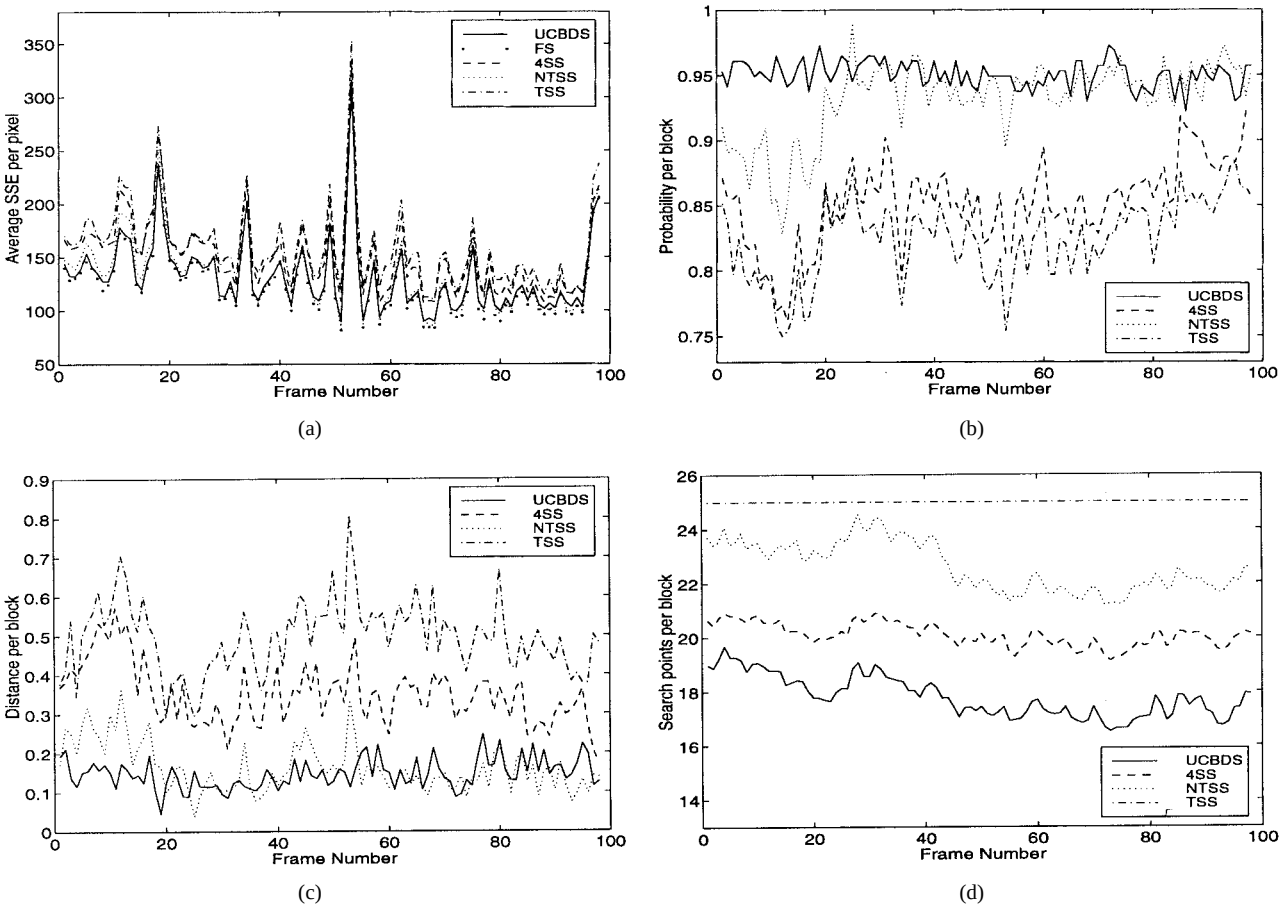

Fig. 7. Performance comparisons of FS, TSS, NTSS, 4SS, and UCBDS over all four test criteria using "Flower Garden" sequence.

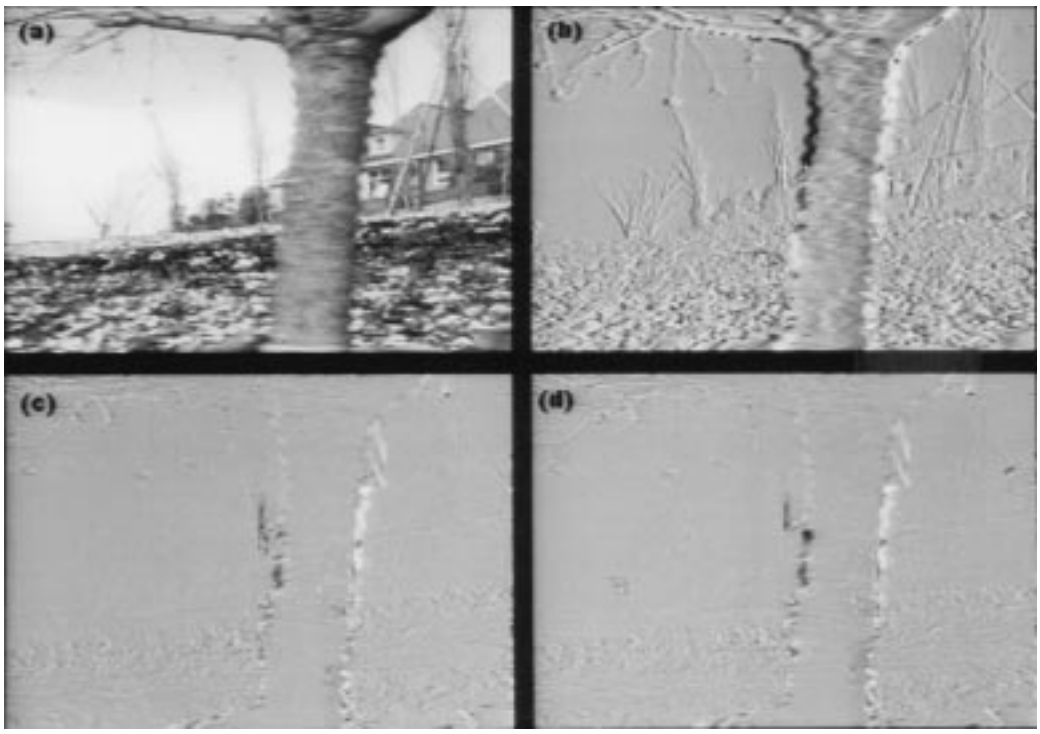

Fig. 8. Performance evaluations. (a) Original frame of "Flower Garden." (b) Uncompensated frame difference (average SSE per pixel: 1115.6). Motion-compensated frame differences using (c) FS (SSE: 142.4) and (d) UCBDS (SSE: 146.4).

that UCBDS has both the minimum and maximum numbers of search points per block due to its center-biased and unrestricted search strategy, respectively. However, the average N_s per block with $\mathrm{UCBDS} < \mathrm{4SS} < \mathrm{NTSS} < \mathrm{TSS} < \mathrm{FS}$; such observations were true for all of the test sequences we used. This shows that UCBDS is generally more efficient (i.e., it has a faster search) than the other schemes, regardless of the presence of panning, zooming, small, or large motions in the sequence.

A natural question now is: How much does UCBDS trade off block distortion for higher search speed? From column 2 of Table I, it can be observed that UCBDS actually performs very competitively in terms of low block distortion, even though it has the lowest average number of search points. For a better comparison of the tradeoff between distortion and search speed, Table II gives the percentage[6] improvement of UCBDS over the other search techniques, using the "Football" sequence as an example. It can be seen that UCBDS has marginally worse BDM performance compared to the other search techniques. However, the speed improvements are quite substantial, and thus justify its use over the other techniques. In another set of comparisons, UCBDS gave both lower BDM and higher search speed for the "Flower Garden" sequence. A possible explanation for the good performance of UCBDS is that it has a very compact search configuration which speeds up the search, while the unrestricted search strategy minimizes the risk of being trapped into a local minimum. From the third and fourth columns of Table I, it can be concluded that UCBDS generally gives a lower average Euclidean distance error from the true motion vectors, and a higher average probability of finding the true motion vectors, when compared with the other suboptimal search techniques.

Figs. 6 and 7 plot the actual performance of each search scheme on a frame-by-frame basis. It is clear that UCBDS performs very well in terms of block distortion, while it consistently outperforms the other methods in terms of search speed. Finally, some results of motion estimation using UCBDS are depicted in Fig. 8 using the "Flower Garden" sequence [Fig. 8(a)]. Fig. 8(b) shows the frame difference (shifted by $+128$, but without scaling) between two frames *without* any motion compensation, while Fig. 8(c) and (d) illustrates the corresponding frame differences *after* performing motion estimation using the FS and UCBDS, respectively. It can be seen that UCBDS gives very good motion estimates (with an average SSE per pixel of 142.4) when compared to the FS (SSE per pixel is 146.4), but UCBDS is about 12.6 times faster than the FS method.

IV. Conclusions and Future Research

In this paper, we proposed a novel *unrestricted center-biased diamond search* (UCBDS) algorithm for fast suboptimal block-based motion estimation. Motivated by the center-biased motion vector distribution characteristics of real-world video sequences, UCBDS was designed with the most (practical) compact diamond search point configuration. We then explained the algorithm development of UCBDS, and performed a theoretical analysis of its efficiency. Simulation results, both objective and subjective, were presented to show that UCBDS is more efficient, effective, and robust as compared to some other popular block-matching algorithms such as the full-search, the three-step search, the new three-step search, and the four-step search. In short, UCBDS has the following advantages over the other search algorithms.

- **Efficiency**—UCBDS is highly center biased, and it has a very compact diamond search point configuration. This allows a minimum of only 13 candidate search points per block, and a speed improvement of up to 31% over the fast four-step search.
- **Effectiveness**—UCBDS has the freedom to search for the true motion vector, which indirectly reduces the chances of being trapped in local optima, and this can lead to lower motion compensation errors.
- **Robustness**—As UCBDS is unrestricted and does not have a predetermined number of search steps, it is flexible enough to work well for any search range/window size.

Lastly, UCBDS still possesses the regularity and simplicity that are useful for hardware implementation. Although UCBDS has demonstrated good results, it is very probable that it may not perform as well if the majority of the motion vectors lie beyond ± 3 pixels. Examples of this scenario include predicted (P) and bidirectional (B) frames in a typical MPEG configuration, where the interframe motion can be very large. In fact, such a problem can be tackled by extending the monogrid UCBDS to a multigrid hierarchical UCBDS (H-UCBDS). Our preliminary simulations [13] have shown that H-UCBDS not only has a fast search for very large motion sequences, but it also generates a multiresolution representation of the motion vectors which is very useful for a scalable video coding framework [12].

[6] A 100% improvement means that it has twice better performance.

Acknowledgment

The authors would like to thank the anonymous reviewers for their useful suggestions and comments.

References

[1] CCITT SGXV, "Description of reference model 8 (RM8)," Document 525, Working Party XV/4, Specialists Group on Coding for Visual Telephony, June 1989.
[2] M. Ghanbari, "The cross-search algorithm for motion estimaton," *IEEE Trans. Commun.*, vol. 38, pp. 950–953, July 1990.
[3] ISO/IEC CD 11172-2 (MPEG-1 Video), "Information technology—Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s: Video," 1993.
[4] ISO/IEC CD 13818-2—ITU-T H.262 (MPEG-2 Video), "Information technology—Generic coding of moving pictures and associated audio information: Video," 1995.
[5] ITU Telecommunication Standardization Sector LBC-95, Study Group 15, Working Party 15/1, Expert's Group on Very Low Bitrate Visual Telephony, from *Digital Video Coding Group, Telenor Research and Development*, or via URL: "http://www.nta.no/brukere/DVC/tmn5."
[6] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799–1808, Dec. 1981.
[7] T. Koga, K. Ilinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. NTC 81*, New Orleans, LA, Nov./Dec. 1981, pp. C9.6.1–C9.6.5.
[8] L. W. Lee, J. F. Wang, J. Y. Lee, and J. D. Shie, "Dynamic search-window adjustment and interlaced search for block-matching algo-

rithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 85–87, Feb. 1993.

[9] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 438–442, Aug. 1994.

[10] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 313–317, June 1996.

[11] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Trans. Commun.*, vol. COM-33, pp. 888–896, Aug. 1985.

[12] J. Y. Tham, S. Ranganath, and A. A. Kassim, "Highly scalable wavelet-based video codec for very low bit-rate environment," *IEEE J. Select. Areas Commun. (Special Issue on Very Low Bit-Rate Coding)*, Jan. 1998.

[13] J. Y. Tham *et al.*, "A fast hierarchical diamond search algorithm for block motion estimation and scalable video compression," in preparation, 1997.