G60/86

MONASH UNIVERSITY
THESIS ACCEPTED IN SATISFACTION OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ON.....[redacted].....

.....[redacted].....

Sec. Research Graduate School Committee

A Novel Distance-Dependent Thresholding Strategy for Block-based Performance Scalability and True Object Motion Estimation

by

Golam Sorwar

B. Sc. Eng., Bangladesh University of Eng. and Tech., Dhaka, Bangladesh, 1994

M. Sc. Eng., Universiti Kebangsaan Malaysia, Malaysia, 1998

A dissertation submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Gippsland School of Computing and Information Technology

Monash University, Melbourne, Australia.

April 2003

Abstract

Motion estimation is an important issue in the field of video and image processing. In video compression, it is often combined with motion compensation to exploit the spatio-temporal correlation of video sequences along the motion trajectory. In video coding, block-matching algorithms (BMA) are used for motion estimation, with the full search (FS) method guaranteeing optimal performance in terms of minimum prediction error and hence, picture quality, though at the expense of a high computational overhead. Many sub-optimal, directionally-based fast searching techniques have been proposed to reduce this computation cost; however, they are generally based on assumptions about the video sequence that are often either inaccurate or inappropriate. Performance is thus very dependent on the motion content within the sequence. As yet, there is no unified fast searching approach that provides sufficient flexibility to be able to automatically adapt system parameters, in relation to a particular application, or a preset user-defined picture quality or computational complexity.

The research presented in this thesis details the development of an application-independent, non-directional, block-based motion estimation system, which provides such flexibility for video coding applications. The fully adaptive motion estimation framework guarantees any level of quality of service (QoS) in terms of prediction quality and processing speed. The approach has, as its basis, the innovative concept of a *Distance-dependent Thresholding Search* (DTS) which exploits statistical analysis of the distortion characteristics of real world video sequences. A full qualitative and quantitative evaluation of the system is provided, together with a computation complexity analysis of the various constituent algorithms. A comparison with other fast directionally-based search algorithms is also presented.

The flexibility of the DTS algorithm is underscored by exploiting its non-directional characteristics to provide significant improved estimates of block-based *true* object motion in object-based video analysis applications. A special filter called the *Mean Accumulated Thresholded* (MAT) filter has been designed specifically to eliminate spurious motion vectors introduced as a result of the limitations of conventional block-based motion estimation methods. Integration of this filter with the proposed motion estimation algorithm demonstrates that it is a very powerful tool for block-based *true* object motion estimation applications.

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

[redacted]

Date: 8/4/03

Acknowledgements

I would like to express my sincere gratitude and profound indebtedness to my supervisors, Professor Lawrence S. Dooley and Dr. Manzur Murshed for their constant guidance, insightful advice, helpful criticism, valuable suggestions, commendable support, and endless patience towards the completion of this thesis. I feel very proud to have worked with them and without their inspiring enthusiasm and encouragement, this work could not have been completed.

I am also grateful to Professor Syed Mahbubur Rahman for his good wishes, encouragement, support, and valuable suggestions.

I thank all the staff at Gippsland School of Computing and Information Technology (GSCIT), Monash University. I would like to thank all the research students of the Multimedia Research Group (MRG), and all other graduate students and friends at GSCIT, Monash University, for their moral support and encouragement. I would also like to extend my thanks to both Julie Murray and Harriet Searcy, Language & Learning Services Unit, Monash University, for their kind assistance in proof reading this thesis.

I wish to express my gratitude to the Gippsland School of Computing and Information Technology for providing an excellent environment for research and financial support in the form of teaching assistance and scholarships. The support I have received from Monash University in the form of scholarships and various professional development programs is gratefully acknowledged.

My heartfelt thanks go to my wonderful wife Shelly for her love, care, patience, and understanding during this work, and without whose encouragement and moral support this dissertation would have been impossible. I would also like to thank my lovely son, Tahseen, my mother and my mother-in-law, and my beloved friend, Polash, for their sacrifice and inspiration.

Last, but by no means least, I thank God for the talents and abilities I was given that made it possible to undertake this research.

Acronyms and Abbreviations

ACDSDTS	Adaptive-Centre Diamond Search Distance-dependent Thresholding Search
ACDTS	Adaptive-Centre Distance-dependent Thresholding Search
ACNTSS	Adaptive-Centre New Three Step Search
ANN	Artificial Neural Network
BBGDS	Block-based Gradient Descent Search
BDM	Block Distortion Measure
BLMS	Block Least Mean Square
BMA	Block-Matching Algorithm
CCF	Cross Correlation Function
CIF	Common Intermediate Format
CODEC	Encoder and Decoder
CSA	Cross Search Algorithm
DC	First Coefficient of Discrete Cosine Transform
DCT	Discrete Cosine Transform
DFD	Displaced Frame Difference
DS	Diamond Search
DTS	Distance-dependent Thresholding Search
ET	Exponential Thresholding
FADTS	Fully Adaptive Distance-dependent Thresholding Search
fps	Frames per second
FS	Full Search
FLMS	Fast Least Mean Square
FSS	Four Step Search
GME	Global Motion Estimation
H.261	Standard for video coding
H.263	Standard for video coding
ILSE	Iterative-Least-Square Estimation
JPEG	Joint Photographic Experts Group
LMS	Least Mean Square

LS	Least Square
LT	Linear Thresholding
MAD	Mean Absolute Difference
MAE	Mean Absolute Error
MAT	Mean Accumulated Thresholded
MC	Motion Compensation
ME	Motion Estimation
MILSE	Modified Iterative Least-Square Estimation
MPC	Matching Pel-Count
MPEG	Motion Picture Experts Group
MSD	Mean Square Difference
MSE	Mean Square Error
MV	Motion Vector
NBLMS	Normalized Block Least Mean Square
NLMS	Normalized Least Mean Square
NTSS	New Three Step Search
OFE	Optical Flow Equation
OSA	Orthogonal Search Algorithm
PDC	Pixel Difference Classification
PSNR	Peak Signal-to-Noise Ratio
QCIF	Quarter Common Intermediate Format
QoS	Quality of Service
RLS	Recursive Least Square
SD	Search Diamonds
SIF	Standard Interchange Format
SP	Search Points
SS	Search Squares
TDL	2-D Log search
TSS	Three Step Search
UESA	Unimodal Error Surface Assumption

Nomenclature

Symbols	Denotation
b	Number of bits in gray level image intensity.
C	Threshold control parameter.
C_d	The number of correct shot detections.
C_E	Exponential threshold control parameter.
C_L	Linear threshold control parameter.
$C_{L_{ini}}$	Initial value of threshold control parameter C_L for each shot of a video sequence calculated from first three frames.
$C_{L_{max}}$	Maximum value of threshold control parameter C_L used in the adaptation process in the FADTS algorithm.
$C_{L_{min}}$	Minimum value of threshold control parameter C_L used in the adaptation process in the FADTS algorithm.
$Complx_{MB}$	The computational complexity in motion vector calculation of a macroblock.
$Complx_p$	The computational complexity in evaluating one pixel match between current and candidate macroblocks.
d	Maximum displacement of a macroblock between the current and reference frames.
$\vec{D}_{mv}$	The displacement between motion vectors of a current block and its neighbouring blocks.
$e^{[m]}$	The error signal between target and actual output at iteration m.
ET(C_E)	Exponential thresholding function with control parameter C_E.
F	Camera focal length.
F_a	Camera focal length after zooming.
F_b	Camera focal length before zooming.
$F_n(i,j)$	Intensity of pixel with coordinate (i,j) in frame n.
F_p	The number of false positives.

Symbols	Denotation
LT(C_L)	Linear thresholding function with control parameter C_L.
$MAE_{k,l}(u,v)$	Mean absolute error between current and candidate macroblocks with upper left coordinate (k,l) and displacement (u,v).
$MSE_{C_{L_{max}}}$	Maximum prediction error in terms of MSE for $C_{L_{max}}$.
$MSE_{C_{L_{min}}}$	Minimum prediction error in terms of MSE for $C_{L_{min}}$.
$MSE_{k,l}(u,v)$	Mean square error between current and candidate macroblocks with upper left coordinate (k,l) and displacement (u,v).
$MV_{predicted}$	Motion vector for the current block predicted from the neighbouring blocks' motion vectors.
MV(u,v)	Motion vector with x and y components (u,v).
$[N, N]$	Macroblock size.
$[N_h, N_v]$	Frame size.
$p_{cx,cy}$	Search centre of a search space.
R	The number of operations required in predicting the search centre for the current block from the neighbouring blocks.
SD_τ	Search diamonds with index τ.
SS_τ	Search squares with index τ.
$SP_{C_{L_{max}}}$	Maximum search speed with threshold control parameter $C_{L_{max}}$.
$SP_{C_{L_{min}}}$	Minimum search speed with threshold control parameter $C_{L_{min}}$.
T_f	False motion vector elimination threshold.
$Threshold(\tau, C)$	Parametric threshold function.
$Threshold(\tau, C_E)$	Parametric exponential threshold function.
$Threshold(\tau, C_L)$	Parametric linear threshold function.
$T_{out(MSE)}$	Target prediction quality in terms of average MSE.
$T_{out(SP)}$	Target search speed in terms of number of search points (SP).
$\bar{V}_{init}$	The initial search window's centre from the origin of the current block.

Symbols	Denotation
$\vec{V}_m$	Mean motion vector.
$\vec{V}_p$	Initial search centre in ACDTS and ACDSDTS algorithms.
ς	Number of macroblocks go in the search process per second.
Ω	Number of operations required for BDM calculation at each search point.
λ	Step size without normalization.
μ	Step size with normalization.
τ	Search square or diamond index in the search space.
ξ	Total number of operations required in the first iteration for calculating the camera parameters for each macroblock.
ψ	Number of operations required per second for the FS algorithm with integer-pel accuracy.
$\Re$	Kernel size used in the MAT filter.

Contents

Chapter 1

Introduction

1.1 Background: Motion Estimation

Video sequences are much richer source of visual information than still image because of the representation of motion. While a single image provides a snapshot of a scene, a sequence of images (widely termed as frames) will register the dynamics within it. The registered motion is a very strong cue which allows human vision to recognise objects as soon as they move, even if they are inconspicuous when still. Motion is, therefore, the most obvious and effective feature in providing global and local understanding as well as describing the dynamic content within a video sequence. In Fig. 1.1, a reference frame, (n-1), shows an object on a white background. The following frame, n (called the current frame), shows the same object but in a different position. The offset between these two positions is called the *motion vector* (MV), which defines how to move the object in the reference frame to its new position in the current frame. The motion vector can be estimated for either each pixel, or for a block of pixels (block-based), in a given frame.

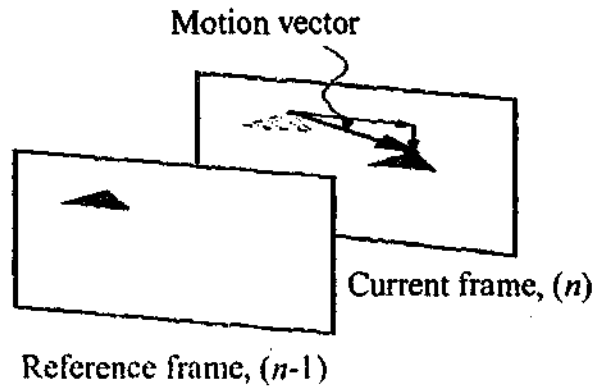


Fig. 1.1: Representation of the motion vector.

Motion is primarily governed by movement due to the camera being used (pan and/or zoom), referred to as *global* motion and movement of objects referred to as either *local or true* object motion, or both. Fig. 1.2 shows the motion vector (block-based) needle diagram for frame n with respect to frame n-1, where n = 32 in the *Table Tennis* sequence, one of the standard video sequences used throughout this research, and whose parameters are detailed in

Appendix B. It is clear that the only moving objects are the bat, ball, and the player's hand. It is also obvious that besides the movement of these objects, there is widespread motion across the frame, which indicates the camera zooming out, with some panning as well. This motion does not depend on any particular object movement and is global over the entire frame; it is therefore referred to as *global* (camera) motion, as shown in Fig. 1.3(a). Fig. 1.3(b) shows *true* object motion, after camera motion cancellation of the aforementioned three moving objects, based on their velocities and directions. The overall motion, if both object and camera motion is present, is the vector sum of the *true* object and *global* motion components.

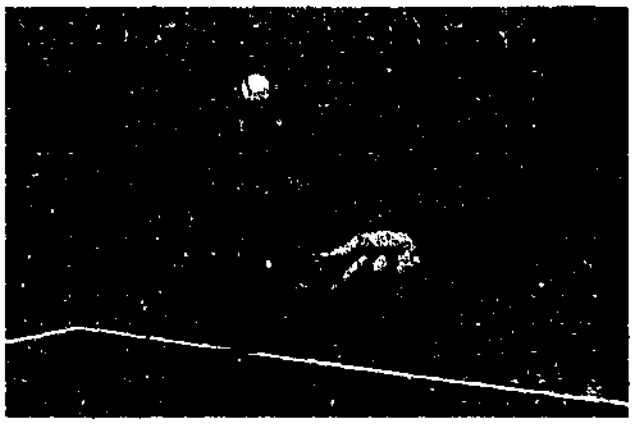


Fig. 1.2: Block motion vector needle diagram showing both *global* and *true* object motion.

(a) *Global* motion.

(b) *True* object motion.

Fig 1.3: Block motion for (a) camera and (b) objects.

The extraction of motion information from sequences of time-varying images has numerous applications in a wide range of areas especially computer vision [8] and image processing. Some of the current applications are: -

- *Video compression*: Perhaps the most important application of motion is in video data compression [9-13]. In the evolving digital technology era, video compression has become an integral part of multimedia applications for both communication and entertainment purposes. In video compression, motion information is used to reduce

inter-frame redundancy; instead of coding every new frame in isolation, references are sought from previously coded frames.

- *Multimedia systems*: Another important application of motion is where the information is used to characterise video clips for automated indexing and retrieval from large databases [14-24].
- *Mobile robotics*: Motion analysis plays an important role in mobile robotics [25, 26] by aiding navigation, obstacle detection and avoidance, and tracking of moving objects.
- *Satellite imagery*: In this area, motion analysis is used to measure cloud movements and establish wind maps for weather prediction [27].
- *Biomedical applications*: In this area for example, motion can be used to monitor movement patterns of the heart using *Magnetic Resonance* (MR) imagery [28-30], or to enhance and interpret ultrasound scans.
- *Surveillance*: This includes applications for urban and road traffic monitoring, and protection of sites from intrusion [31].
- *Image restoration and enhancement*: In this area, motion analysis can be exploited to remove telecine flicker or motion blur from old movies [32, 33].

As the diversity of these applications indicate, *motion estimation* (ME) has been the focus of extensive research over many years [34-37], and this is reflected in the plethora of motion estimation and analysis techniques that have been proposed. Existing motion estimation techniques may be broadly classified into three distinct classes: -

1. Gradient-based [34, 37-40],
2. Pel (pixel)-recursive [41-43],
3. Block-based [36, 44-54].

These are discussed below:

1.1.1 Gradient-based Motion Estimation

Pixel-based motion estimation is a gradient-based method [34, 37-40] which focuses on estimating the apparent motion of intensity patterns in a video sequence, known as *optical flow*, and is based on two assumptions. First, that the b ness of an object stays constant over time. This assumption is called the *data constancy constraint* [40]. Second, that pixels in a given small image neighbourhood are likely to correspond to points on the same 3-dimensional (3-D) surface, the so-called spatial coherence assumption. Since the projected motion of points on a 3-D surface usually varies gradually, a correspondence of this assumption is to impose a smoothness constraint on the *optical flow*.

The *data constancy constraint* assumes that any observable change in image intensity over time is due only to camera and object movement. This constraint has been formulated as a single differential equation in terms of the partial derivative of the image intensity function, known as the *optical flow equation* (OFE). Since the optical flow at each image point is 2-dimensional (2-D), it contains two variables. Thus, to fully constrain the equation, the *spatial coherence assumption* is used, either locally, by requiring that the optical flow over some arbitrary small region is constant [38], or globally, by minimising the total pixel-to-pixel variation of the estimated motion vector [37]. As these techniques generate a velocity vector for each pixel in the image, they generate a dense motion field, which is useful in computer vision tasks where a large set of motion vectors are often required. However, these techniques have the following drawbacks: -

1. They require estimation of spatial and temporal gradients, and this is often noise sensitive.
2. The intensity derivatives are numerically approximated. This requires local spatio-temporal linearity of intensity. In image sequences with high motion, local linearity is violated [55].
3. OFE is ambiguous in relation to the projected motion; that motion can only be defined in a direction perpendicular to a gradient means that the gradient-based methods suffer from *aperture* problems [37, 56].

From a video coding perspective, there are also two other fundamental drawbacks:-

4. The smoothness constraint leads to an increased prediction error energy.
5. A dense motion field requires a large information overhead.

For these various reasons, gradient-based techniques for motion estimation will not be considered any further in this thesis.

1.1.2 Pel-Recursive Motion Estimation

Pel–recursive methods [41-43] can be considered as a subset of gradient-based methods, and have been developed for image-sequence coding. They obtain a dense optical flow by raster scanning, that is, they start the estimation at the top-left pixel and end at the bottom-right pixel. The luminance of pixel x in the current image is predicted from the reference image by means of the correspondence vector found at the previous pixel in the current image, by recursively minimising certain prediction error criteria, commonly known as the *Block Distortion Measure* (BDM). It is assumed that the previous vector is a good estimator of the new vector and thus, only small changes are allowed between the two vectors. The advantages of these methods are: -

1. As the update of the motion vector is only based on previously transmitted data (causality), the decoder is able to estimate the same motion field as is usually estimated by the encoder, so requiring no motion information overhead [57].
2. The regular structure and causality of these methods allows for efficient implementation in hardware.

There are however, a number of disadvantages when applying these methods to video compression applications: -

1. The first of the above advantages is obtained at a cost of increased complexity at the decoder, as the encoder has to also estimate the motion field [57].
2. The causality constrains these algorithms and reduces their prediction capability compare with non-causal methods.
3. As the error function for minimising generally contains many local minima, the iterative procedure may converge to local, rather than global minima. In particular, these algorithms are very sensitive to noise.
4. Large displacements and discontinuities in the motion field cannot be efficiently processed.
5. The pel-recursive motion estimation technique (with recursion on pel) is not compatible with transform coding of BDM, as, in this case, the decoder is unable to reconstruct the motion vector.

Due to these limitations, pel-recursive techniques will not be considered any further in this thesis. To address the shortcomings of both the gradient-based and pel-recursive methods, the obvious alternative is to consider a block of pixels, rather than estimating motion on a pixel-by-pixel basis. The superiority of block-matching algorithms for motion estimation in video coding applications will now be examined.

1.1.3 Block-based Motion Estimation

Block-based methods represent, in certain respects, an opposite philosophy from the previously discussed gradient-based estimation techniques because larger analysis windows are used to avoid some of the problems identified in previous sections. Besides the *data constancy constraint*, these methods also assume that objects move in a translation movement for, at least, a few frames. Based on this assumption, the idea in block-matching algorithms is that the image sequence should consist of a set of regions each undergoing a single motion between frames. Each frame is then divided into a set of non-overlapped, equally spaced, fixed size, small rectangular blocks called macroblocks, and the translation motion within each block is assumed to be uniform. The motion vector of each block is then found by searching for the

corresponding blocks in the reference frame by minimising certain matching criteria between the gray levels, such as the *mean absolute error* (MAE). Such block-based motion estimation methods are popularly termed *block-matching algorithms* (BMAs).

BMAs have a number of advantages compared to other techniques: -

1. They are very simple, straightforward, and yet very efficient.
2. Their regular structure and causality allow for efficient implementation.
3. They are less sensitive to *aperture* problems since, with a suitable block size, each block is likely to contain several image gradients. In general, block-matching methods will also be less sensitive to noise since more image data is used in the motion estimation process.
4. They outperform other methods in capturing *true* motion in high motion video sequences [57].
5. Although the simple BMA model considers translation motion only, other types of motion, such as rotation and zooming of large objects, may be closely approximated by the piecewise translation of these small blocks, provided the blocks are small. This important observation, originally made by Jain *et al.* [36], has been frequently confirmed [58].

These advantages are counterbalanced by two key limitations: -

1. During motion estimation, a single motion vector is considered for all pixels within one block and the motion vectors of partitioned blocks are estimated independently of each other, leading to picture artifacts.
2. The fixed block size also imposes a limit on the accuracy of the estimated motion field since the regions are unable to adapt to the underlying image data.

To address these shortcomings, one extension to the basic block-matching algorithm is to consider sub-pel (half-pel or quarter-pel) accuracy in motion estimation which leads to a significant improvement when compared to integer pel accuracy [59]. The half-pel approach is detailed in Chapter 2 and is used throughout the thesis for all experimental results in video coding applications.

From this discussion, it can be concluded that among all different motion estimation techniques, the BMA is the most effective, especially from a video coding point of view, and for this reason has been adopted by all the international video coding standards, including MPEG-1/2 [9, 10] and H.261/263 [11, 12]. Recently, block-matching techniques for motion estimation have been exploited for their simplicity and ease of implementation in many other video processing applications including video object segmentation for object based video coding [60-62], video object segmentation, detection and tracking for content based indexing,

querying, browsing, and searching of video objects [17-24]. Two of these areas of application are further explored below.

Block Motion for Video Coding

Video coding techniques take advantage of data redundancies in order to reduce the bit-rate/bandwidth needed to represent visual information. There are many methods of compressing video data but most follow a common structure which incorporates two types of compression: intraframe and interframe coding. The former exploits the spatial characteristics of a single video frame while the latter exploits the temporal characteristics between two or more neighbouring video frames.

Among the different intraframe coding techniques, predictive, transform, subband wavelet, and second generation coding [63] are the most popular. Interframe coding can be considered as a particular case of predictive coding where the prediction is based on pixel values from the reference frames. For instance, in the portion of a scene with very low motion, pixels can be precisely predicted from the pixel at the same location in the reference frame. However, this is not valid in scenes with high motion. In this case, pixels in the reference frames spatially displaced by the appropriate vector are more efficient for prediction. This is known as *motion compensated* prediction. The difficulty of this approach lies in estimating accurately the motion between two frames, which is the sole aim of *motion estimation* (ME), often referred to as *motion compensation* (MC). In any coding strategy based on this principle, the motion compensated prediction error (residual error) as well as the motion vectors are transmitted, instead of the frame itself.

There have been numerous contributions in the literature that aim to estimate block motion vectors for video coding applications. A comprehensive review is provided in Chapter 2. Among existing techniques, the *full search* (FS) algorithm [36] is a brute force BMA method which searches all possible locations inside the search window and produces an optimal solution in terms of prediction quality. If the performance in terms of BDM is the only criterion, FS is obviously the best and simplest approach to use. However, its high computational complexity often makes it unsuitable for real-time implementation. This has led to the development of fast BMAs such as the *2-D logarithmic search* [36], *three-step search* (TSS) [44], *new three-step search* (NTSS) [45], *four-step search* (FSS) [46], *advanced centre biased search* [47], *cross-search* [48], *prediction search algorithm* [49], *orthogonal search algorithm* [50], *simple and efficient search* [51], *block-based gradient descent search algorithm* [52], and *diamond search* [53]. All these fast BMAs have been based on a *unimodal error surface assumption* (UESA) which implies that the BDM increases monotonically as the search point

moves away from the global minimum. These algorithms perform well only if this assumption is upheld. There have also been a number of methods developed using pixel subsampling [54, 64, 65] and spatio-temporal correlation [49, 66-68]. The aim of these particular motion estimation algorithms is to reduce the computational cost of FS algorithm in terms of either the number of operations, by considering a subset of pixels instead of all pixels during the BDM calculation, or reducing the number of search points by predicting the starting centre of the neighbouring block's motion vectors. Although the error performance of these algorithms is comparable to the FS algorithm, the computational cost is also higher compared with other fast algorithms. All these existing fast algorithms, however, have some inherent limitations: -

1. Due to the highly non-stationary characteristics of the video signal, the *unimodal error surface assumption* is generally invalid for many video sequences. Moreover, the search direction of fast algorithms can be ambiguous, leading to the motion vectors becoming entrapped in a local minimum, with a resulting degradation in predictive performance.
2. Most fast algorithms are application and/or system dependent. For example, algorithms such as NTSS and FSS are intended and optimised for low bit-rate video coding applications (video conferencing or videophone), whereas others such as TSS are optimised for high quality video in the context of the MPEG-2 standard.
3. None of the fast algorithms have been designed to provide flexibility in controlling the performance in terms of predicted picture quality and processing time (speed). They do not allow any performance scalability in motion estimation, have no facility to trade system parameters depending upon a particular application, or to preset a user-defined level of *Quality of Service* (QoS) in terms of predicted picture quality or computational complexity.

The above discussion highlights that currently there is no single block-based motion estimation solution that exhibits good performance at both low bit-rate and high quality video coding cases, while providing flexibility in performance management in terms of either predicted image quality or processing speed. Such a generic solution would be very advantageous in facilitating complexity management in video coding, especially in real-time software-only video CODECs (Coder and Decoder) or low-power video CODECs for mobile or hand-held computing platforms which particularly require a more flexible trade-off between complexity and quality [59]. The development of such a novel system that addresses many of the above challenging issues is a key motivation for this thesis.

An important yet disparate area where block motion estimation theories are being increasingly utilised will now be explored.

Block Motion for *True* Object Motion Estimation

In coding applications, motion vectors are obtained to minimise the BDM irrespective of whether the vector points in the direction of the moving objects. A motion vector that represents the motion of a foreground moving object is termed *true* object motion (Fig. 1.3). In estimating block-based motion vectors, errors may be introduced not only because of representing all the pixels of a macroblock by a single vector (Fig.1.2), but also in seeking the minimum BDM. These errors following *global* motion compensation will lead to *false* object motion.

With the rapid growth in multimedia and Internet applications, there is a huge amount of video data available, which highlights the need for efficient representation of video information to allow content-based functionality. As motion provides one of the easiest cues to a sequence's temporal dimension [69], it is one of the most important visual features for content-based video representation and is increasingly becoming an essential part of several applications, including content-based video indexing for browsing and retrieval [18-24], video surveillance systems [31], video object segmentation and tracking [20, 23], and object based coding [13]. Amongst these different applications, one of the most interesting is using object motion in video indexing for accessing large amounts of multimedia data over the Internet.

Various algorithms [15, 70, 71] have been proposed to index video by dense motion field using OFE [37, 38] where the apparent velocity and direction of every pixel in the frame has to be computed. Although it is an effective method, it is computationally intensive and very complex. The OFE method also does not cope well with high motion video sequences [57].

To overcome this problem, many recent video processing applications have explored block-based motion estimation techniques to estimate *true* object motion. In Tancharoen *et al.* [60] and Ji and Park [61, 62], BMA techniques were used to estimate the motion for moving object segmentation from the background for object-based video coding and video analysis. As the motion vector information of the macroblocks is available in MPEG coded video streams, an alternative to video representation [18-23] is provided by extracting this information, thus avoiding time consuming computation of optical flow. However, there are some drawbacks associated with these approaches: -

1. The motion vectors in a coded bit stream do not always represent *true* object motion since motion estimation is performed solely from the coding efficiency point of view, where minimum error matching is the only criterion.
2. As the block-matching method captures both camera (*global*) as well as *true* object motion, the block motion available in the video stream does not directly provide the *true* object motion.

3. Block-based motion estimation also introduces *false* motion vectors in the form of noise, especially in uniform regions [20]. To eliminate *false* vectors, a non-linear *median* filter can be applied [17, 21, 31, 72] for motion field smoothing. Though *median* filters work well for image processing applications such as noise reduction, image enhancement and restoration, they are inefficient for eliminating *false* motion vectors, because they tend to remove significant numbers of *true* object motion vectors along the edge of the objects, especially when objects are relatively small compared with the size of the frame.

To highlight the weakness of current BMAs, Fig. 1.4 (reproduced from Fig. 6.16 in Chapter 6) shows the average percentage of *true* object motion vectors captured by different BMAs for a range of standard and non-standard video sequences (Appendix B). It reveals that while FS is the optimal algorithm for predicted image quality, performance is very similar to NTSS, and overall, no fast search algorithm captures more than 60% of the *true* object motion vectors. This is because existing BMAs only capture motion based upon a minimum matching criterion.

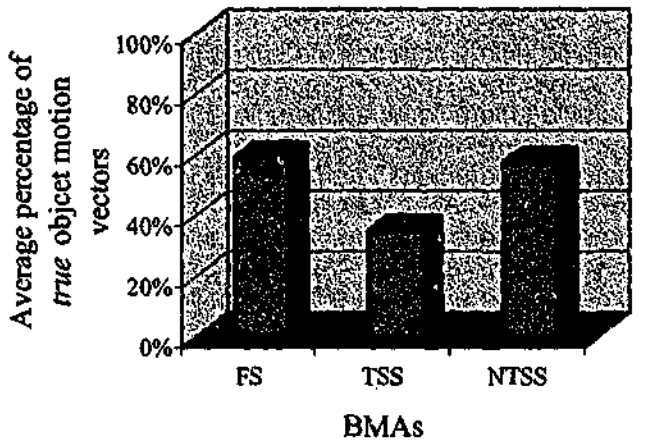


Fig. 1.4: Average percentage of *true* object motion vectors captured by different BMAs.

This discussion indicates that while block-based motion can be applied for *true* object-based video representation, to effectively capture *true* object motion using the BMA approach is a challenging task. To place this in context, many of the desirable features identified in the previous section on block motion estimation for video coding are equally applicable for *true* object motion estimation. It is the potential extension of these new block-based motion principles into the area of *true* object motion estimation that is one focus of the present research.

1.2 Fundamental Premise

It has been shown in Feng *et al.* [73] and Lim and Ho [74] that the magnitude of a motion vector is proportional to the BDM, an observation that will be explored further in Chapter 3.

Fig. 1.5 shows a simplified version of Fig. 3.3(a) in Chapter 3, which was obtained after processing the high motion *Football* video sequence as follows: -

(i) The motion vectors were calculated using the FS algorithm for all blocks in the first 80 frames.

(ii) The frequencies of each distinct minimum MAE of similar length motion vectors were calculated.

(iii) These frequencies were then translated into a cumulative probability (Definition 3.3) of minimum MAEs.

Fig. 1.5 clearly reveals that the minimum MAE for a particular cumulative probability, for example, 0.86, increases from 25, when the motion vector length is 0, through to 60, when the motion vector length has increased to the range $[7,7\sqrt{2}]$. The graph also confirms that there is a higher probability of terminating the FS algorithm at a higher MAE value as the motion vector length is increased.

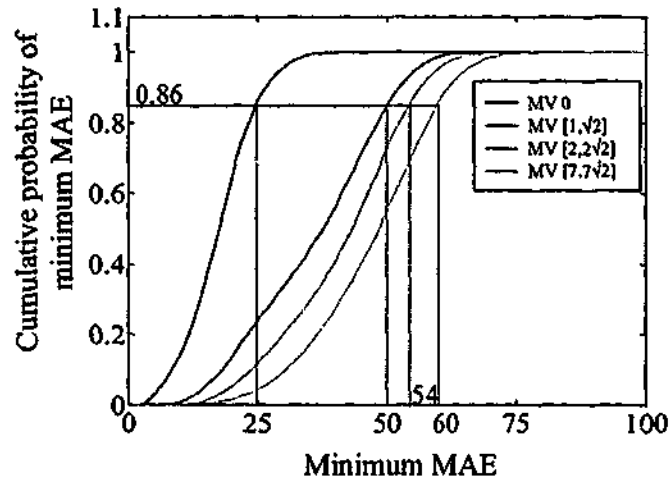


Fig. 1.5: The cumulative probability of minimum MAE for the *Football* video sequence.

This leads to a fundamental premise, which is that the distortion of an object in a video frame increases with its velocity as well as the zoom and pan factors of the camera. Thus, as the length of the motion vector grows, so does the distortion error. Based on this tenet, it can be concluded that locating a block with the minimum prediction error but with a motion vector of high magnitude, is not only ineffectual in the prevailing distorted search space, but will inevitably lead to many *false* motion vectors being erroneously selected. Designing a new BMA that seeks to exploit this feature has a number of potential advantages.

1. It allows a search to be restricted for any given length of motion vector beyond a certain threshold of minimum BDM. This leads to the novel concept of a fast non-directional BMA.

2. A variable threshold in the search process, which increases as the search expands outwards, will enable the user to restrict the search boundary so that it can be used as an effective control parameter for performance scalability and QoS in terms of predicted image quality as well as processing time in motion estimation.
3. Unlike directional fast algorithms, a non-directional search means it is unlikely that the search will be trapped in either a global minimum or a minimum along any specific direction, especially when the search progresses away from the centre. A global minimum does not always represent the *true* motion vector, especially if it is far from the search centre, as it may be introduced by a different object or *global* motion. However, such an approach has the potential to improve the ratio of captured *true* object motion vectors.

The above indicates that this strategy affords a promising control mechanism which can be used for performance management in motion estimation for coding applications as well as for capturing more *true* object motion vectors than other fast search algorithms.

1.3 Motivation and Contribution

While extensive work has been done in block-based motion estimation for video coding applications, there are still many issues left unresolved by existing techniques, specifically for real-time software-only and low power video coding applications. Performance scalability, application independency, and QoS in terms of prediction error or processing time also remain challenging issues.

It is in this context, therefore, that this dissertation presents the novel system shown in Fig. 1.6, which is characterised by being: -

1. A fully adaptive system which can efficiently provide flexibility in controlling the complexity of motion estimation for software-only or low-power video coding by trading between picture quality and complexity.
2. An adaptive system for performance scalable motion estimation that provides QoS by satisfying any level of user demands in terms of image prediction quality or computational complexity for video coding applications.
3. A generic system that exhibits consistent performance for all types of video sequence, including high or low motion.
4. A non-directional block-based motion estimation algorithm that addresses the drawbacks of existing BMAs for both video coding and *true* motion estimation applications.

Blocks 1, **2**, and **3** in the block diagram of Fig. 1.6 are the key components of the new motion estimation system. This flexible system also provides an opportunity to exploit the new non-directional, variable thresholding search method in capturing increased numbers of *true* block-based object motion vectors. **Block 4** represents the additional modules that need to be integrated to achieve this objective. It is important to emphasise that while **Block 4** is not the primary focus of the research, it demonstrates the potential of using this new search method for block-based *true* object motion estimation and provides both improved qualitative and quantitative performance.

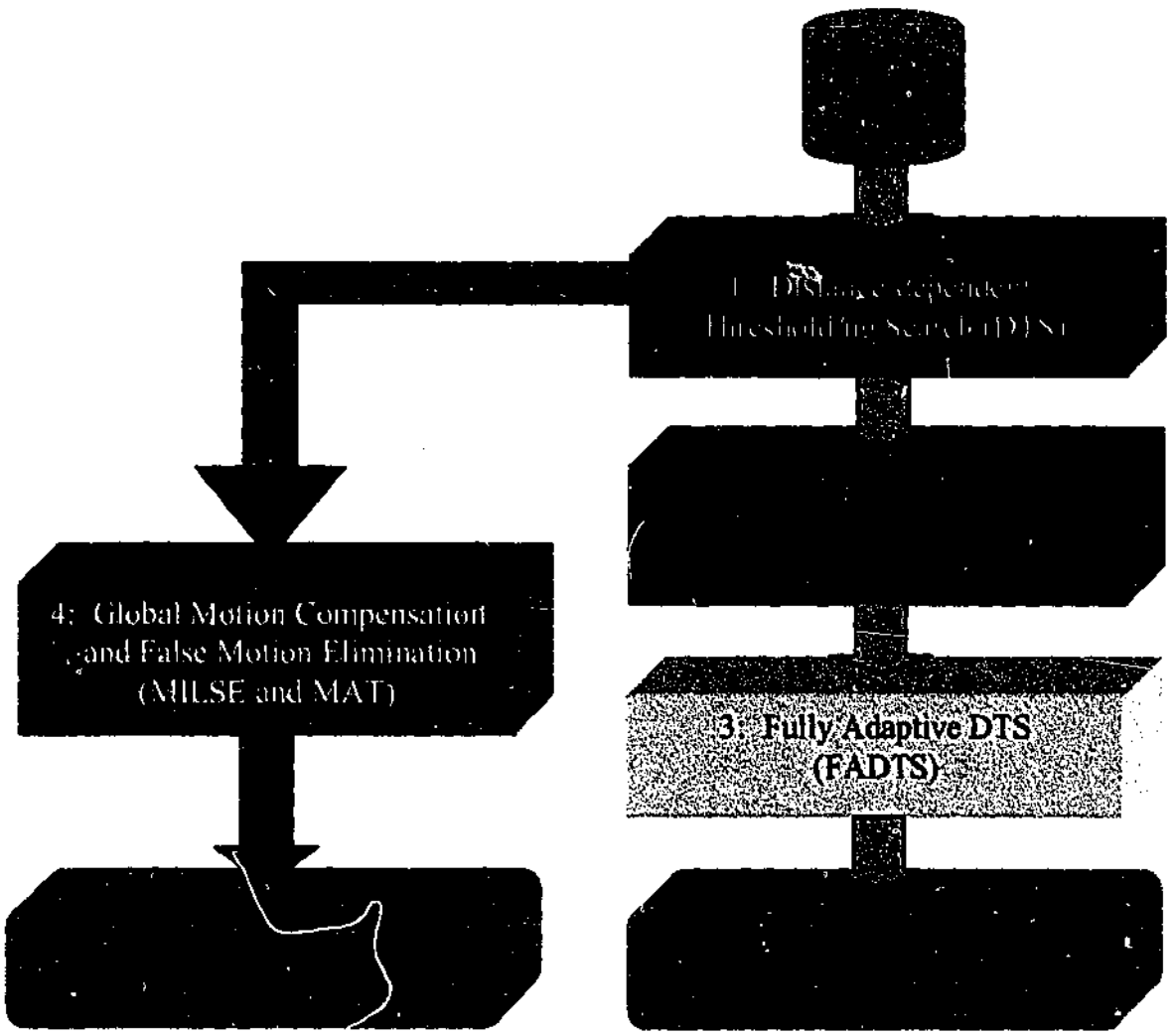


Fig 1.6: System block diagram.

The main contributions of this research are: -

- Development of a novel variable *distance-dependent thresholding search* (DTS) block based motion estimation algorithm for real-time video coding (**Block 1** in Fig. 1.6). This algorithm is independent of the ubiquitous *unimodal error surface assumption*, and a unique feature is that it can be used both as the FS algorithm for optimum quality, as well as a fast BMA. It can therefore be applied for performance management motion estimation [1, 2]. In obtaining *true* object motion, the DTS

algorithm has also demonstrated good performance by capturing significantly more *true* object motion vectors compared to existing FS and fast BMAs [3, 4].

- While the DTS algorithm has provided better search efficiency compared to existing fast BMAs such as TSS or NTSS for low motion video sequences, it was found to be not so effective for highly complex motion sequences. To improve efficiency, a fast *adaptive-centre DTS* (ACDTS) algorithm has been developed by integrating the concept of spatio-temporal motion correlation of the neighbouring blocks' motion vectors with the DTS algorithm (**Block 2**). The ACDTS increased the likelihood of finding motion vector prediction quality with fewer search points compared to the DTS algorithm.
- As the actual search pattern used has a strong impact on the performance of a BMA, to enhance the search efficiency of the ACDTS algorithm, a *diamond search* pattern instead of the usual rectangular pattern has been implemented (**Block 2**). This algorithm, called the *adaptive-centre diamond search DTS* (ACDSDTS), outperformed both the DTS and ACDTS algorithms by trading off quality with complexity. Because of the relatively large step size used in the horizontal and vertical directions, such a pattern is able to find high motion blocks with fewer search points, and also reduces its susceptibility to being trapped in local minima.
- While DTS, ACDTS and ACDSDTS all provided performance scalability in motion estimation in terms of prediction quality and processing speed with different threshold settings, these thresholds had to be manually defined. To fully automate the system, the ACDSDTS algorithm has been extended to a *fully adaptive distance dependent thresholding search* (FADTS) block motion estimation algorithm that can satisfy any level of user demand in terms of predicted image quality and processing speed (**Block 3**) [5]. A unique feature of FADTS is that it adjusts the threshold automatically using the desired target and the content from the actual video sequence, to achieve a guaranteed level of QoS in terms of image quality or processing complexity. This is very effective in facilitating performance management of motion estimation, especially for low bit rate and software-only video coding. As the motion estimation always considers the first frame of each shot as the reference frame, a shot detection technique using an *Artificial Neural Network* (ANN) has been developed [6] which can be embedded in the proposed FADTS system to detect a shot change (**Block 3**) for non-real-time coding applications. A simple, yet elegant solution is proposed, which will allow shot changes to be detected in real-time.

- From a *true* object motion estimation perspective, while the DTS algorithm (**Block 1**) provided better performance compared to existing BMAs [3, 4], it did not consider *global* motion cancellation (compensation), i.e. the capture of *true* object motion where both object and camera motion are involved. To resolve this issue, a *Modified Iterative Least-Square Estimation* (MILSE) *global* motion estimation method has been designed and implemented which reduces computational complexity without degrading the performance in terms of *global* motion parameter estimation (**Block 4**).
- Having applied *global* motion cancellation (**Block 4**), the resulting motion field of the DTS algorithm (**Block 1**) possesses a number of spurious motion vectors because of block-based estimation limitations. A new non-linear filter, called the *Mean Accumulated Thresholded* (MAT) filter has been designed (**Block 4**) to eliminate these *false* motion vectors while retaining only *true* object motion vectors [7].
- Finally, an analysis of computational complexity for each of the algorithms in the system framework shown in Fig. 1.6 has been undertaken, and is presented in each relevant chapter.

1.4 Structure of the Dissertation

In Chapter 2, the importance of block-based motion for video coding is discussed and the different block-matching criteria used in motion estimation are described. A contemporary review of existing block-based motion estimation methods is presented. The relative merits and shortcomings of each method are highlighted to set in context the research detailed in this thesis.

In Chapter 3, a new *variable distance dependent thresholding search* (DTS) block-based motion estimation algorithm is introduced for real-time video coding and *true* object motion estimation based on the key premise that the distortion of an object in a video frame increases with the velocity of the moving object and camera factors. Both linear and non-linear (exponential) threshold functions for the DTS algorithm are evaluated, and its performance is compared to existing fast and exhaustive BMAs using both integer-pel and half-pel accuracy. The performance of the DTS algorithm, in terms of capturing *true* object motion, is also qualitatively assessed, with a more comprehensive evaluation being provided in Chapter 6. Work from this chapter has been published by Sorwar *et al.* in [1-4].

In Chapter 4, the DTS algorithm is extended to an *adaptive-centre* DTS (ACDTS) algorithm to improve performance. A spatial motion correlation between neighbouring blocks has been integrated to automatically predict the best search starting point in each window. As this predicts a starting point closer to the global minimum point, it reduces the computational

cost with better prediction quality. The performance of the ACDTS algorithm is compared with TSS and NTSS as well as the *adaptive-centre* NTSS (ACNTSS) algorithms. The search efficiency of the ACDTS algorithm is further improved by considering an efficient *diamond search* pattern (ACDSDTS) based on the central-biased motion distribution characteristic of a video sequence. The computational overhead introduced by using the centre prediction and diamond pattern search is also analysed.

In Chapter 5, the ACDSDTS algorithm has been extended to a fully adaptive distance dependent thresholding search (FADTS) algorithm. A brief review of existing adaptive algorithms is presented and a new model for threshold adaptation proposed. The different parameters related to this model are analysed to achieve optimum performance of the FADTS algorithm. As shot changes in a video sequence are used as reference frames for the adaptive process, existing shot detection techniques are also briefly outlined. An integrated shot detection (camera break) technique based on an artificial neural network (ANN) and BDM thresholding are described for non-real-time and real-time applications respectively. Work from this chapter has been published by Sorwar *et al.* in [5, 6].

Chapter 6 examines the potential of applying the DTS algorithm to capture an improved number of *true* object motion vectors by using *global* motion estimation and compensation, and *false* motion vector elimination. A new *false* motion elimination filter called the *mean accumulated threshold* (MAT) filter is proposed to extract *true* object motion vectors from the motion vector field. Existing *global* motion (camera pan, zoom) estimation models and methods are also reviewed, and a *global* motion estimation method based upon a *modified iterative-least-square estimation* (MILSE) is presented to reduce the computational cost. Finally, the performance of the DTS algorithm for *true* object motion vector estimation is quantitatively analysed. Material from this chapter has been published by Sorwar *et al.* in [7].

Finally, the main conclusions from this research and proposals for future research directions are given in Chapter 7.

Chapter 2

Motion Estimation: A Review

2.1 Introduction

The extraction of motion information from a sequence of time-varying images has numerous applications in the field of image processing and video coding applications. Among the different applications, the most important application is video compression.

Video compression has become an integral part of multimedia applications for both communication and entertainment purposes. It takes advantage of data redundancies in order to reduce the bandwidth needed to represent the visual information. In the framework of video coding, the redundancies arise from both spatial correlation within an image and temporal correlation between successive images. Due to the different nature of the video signal in the spatial and temporal dimensions, spatial and temporal correlations are usually processed separately. Coding techniques that reduce the spatial correlation are referred to as *intraframe coding*, whereas those that reduce the temporal correlation are called *interframe coding*. A review of intraframe and interframe coding is given by Netravali and Limb in [75] and Jain in [76].

Among the various inter/intra-frame compression techniques [56, 77] the motion compensated transform coding technique is the most popular, and has been adopted in many video coding standards such as MPEG-1/2 [9, 10] and H.261/263 [11, 12] owing to its high compression efficiency. The latter belong to the class of nonlinear interframe predictive coding where the prediction is based on pixel values from the previous frame. In the first stage, the displacement of objects between successive frames is estimated (*motion estimation*). The resulting motion information is then exploited in efficient interframe predictive coding, known as *motion compensated prediction*. The difficulty of this approach lies in estimating accurately the motion between two frames. In any coding scheme based on the above principle, the motion compensated prediction error, more commonly called *displaced frame difference* (DFD), is transmitted instead of the frame itself. It results in a more efficient representation of the visual data. The motion information also has to be transmitted as an overhead.

The previous chapter confirmed that block-matching motion estimation algorithms have already been adopted in different video coding standards because of their simplicity and ease of implementation. It was also shown in Section 1.1 that block motion has recently been used for video object analysis, in particular, the block motion available in the MPEG video stream. As the primary focus of this thesis is on block-based motion estimation, an extensive review will be provided in this chapter, while applications of block motion as *true* object motion will be discussed in Chapter 6.

This chapter is organised as follows. In Section 2.2, the reasons why block-matching is considered as a generic and efficient technique for motion compensation video coding are presented, with some issues relating to block-matching motion estimation given in Section 2.3. In Section 2.4, existing block-matching motion estimation techniques, with their advantages and disadvantages, are analysed from a computational point of view. Half-pel accuracy in motion estimation is also discussed in this section. This chapter concludes with a summary of the key problems associated with existing BMAs in Section 2.5.

2.2 Importance of Block-Matching in Video Coding

Video coding exploits temporal redundancy in order to reduce the bandwidth while preserving the quality of the receiver-reconstructed images. This has resulted in many motion based video compression strategies. Simple *frame-differencing* strategies assume that the average motion is small, and simply compress the pixel-by-pixel difference between two frames.

Vector quantization [78] is an alternative strategy where a *codebook* of commonly occurring pixel patterns is constructed. The compression process replaces a pixel pattern with its corresponding codeword. While this technique results in superior compression, construction of the codebook is a difficult problem. Techniques that attempt to recognise individual objects as they move from frame to frame have been used to construct effective codebooks. Practically, to reduce computation and storage complexity, motion parameters of objects in a picture are estimated based on two or three adjacent frames. Most of the motion estimation algorithms are based on the following assumptions:-

- Objects are rigid bodies; hence object deformation can be neglected for adjacent frames.
- Illumination is uniform along the motion trajectory (Section 1.1.1).
- Objects move in a translational movement for at least a few frames.
- Occlusion of one object by another, and uncovered background, are neglected.

The problem with motion estimation, in fact, consists of two related sub-problems: -

- identification of the *moving object* boundaries, or so-called motion segmentation.

- estimation of the motion parameters of each moving object.

Here, a *moving object* is a group of contiguous pixels that share the same set of motion parameters. This definition does not necessarily match the ordinary meaning of object. For example, in a videophone scene, the still background might include a wall, bookshelf, or decorations. As long as these items are stationary (sharing the same motion vector), they can be considered as a single object in the context of motion estimation and compensation. The smallest object may contain a single pel. One difficulty in using small objects (or evaluation windows) is the ambiguity problem. Similar objects (image patterns) may appear at multiple locations inside a picture and may lead to incorrect displacements vectors. Also, statistically, estimates based on a small set of data are more vulnerable to random noise than those based on a larger data set.

Alternatively, if a large number of pels are treated as a single unit for the estimation of their motion parameters, it is important to know precisely the boundaries of the moving objects, otherwise these may cause accuracy problems. Pixels inside an object, or evaluation window, do not share the same motion parameters and, therefore, the estimated motion parameters are not accurate for some, or all, pels in the object or window. On the other hand, the criterion of grouping pels into moving objects, no matter which scheme is in use, must be consistent with the motion information of every pel.

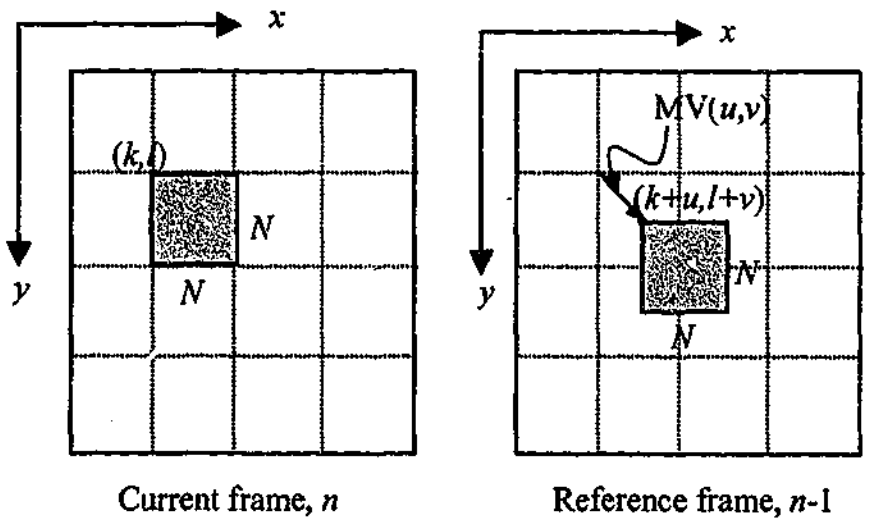


Fig. 2.1: Block motion estimation and compensation.

There exist practical solutions to circumvent the aforementioned motion segmentation and estimation dilemma. One solution is partitioning images into regular, non-overlapped macroblocks, assuming that the moving objects can be approximated reasonably well by regular shaped blocks. A single motion vector (MV) is then estimated for each macroblock, under the assumption that all the pels in the block share the same MV, as illustrated in Fig. 2.1, where the macroblock size is $N \times N$, (k,l) represents the x and y coordinates of the upper left pixel position

of the current macroblock in the current frame, and (u,v) represents the x and y components of MV where the directions of x and y in Figs 2.1 and 2.2 represent the positive directions of (u,v).

This assumption of a single motion vector may not always be true because an image block may contain more than one moving object. In image sequence coding, however, prediction errors due to imperfect motion compensation are coded and transmitted. Hence, because of its simplicity and small overhead, the block-based motion estimation-compensation method is widely adopted in real video coding systems.

2.3 Block-based Motion Estimation Techniques

This section reviews the main approaches to block-based motion estimation that currently exist, identifying key problem areas. In a broad sense, motion estimation techniques can be classified into 3-dimensional (3-D) methods [79-81] and 2-dimensional (2-D) methods. 3-D methods attempt to determine the motion by solving the projection equations directly by making use of feature correspondences between frames, whereas 2-D methods estimate optical flow. As this thesis predominantly focuses on the 2-D methods, 3-D methods will not be discussed further, although a comprehensive review of techniques is provided in [79, 80].

2.3.1 Block-Matching Methods

Jain and Jain [36] first used a block-matching motion estimation for an interframe coding structure and proposed a fast search algorithm to reduce computation. Extensive work has since been undertaken to extend their method. Before reviewing the various existing techniques, some of the important issues related to block-matching will be discussed.

2.3.1.1 Basic Concepts

Block matching is a correlation technique that searches for the best match between the current image block and candidates in a confined area of the reference frame. Fig. 2.1 illustrates the basic operations of this method. In a typical use of this method, images are partitioned into non-overlapped rectangular blocks. Each block is viewed as an independent object and it is assumed that the motion of pixels within the block is uniform. The MV is the by-product when the new location of the object (block) is identified. The size of the block affects the performance of motion estimation. Small block sizes afford good approximation of the natural objects' boundaries, and of real motion, which is now approximated by a piecewise translation movement. However, small block sizes produce a large amount of raw motion information, which increases the number of transmission bits or the required data compression complexity, in

condensing this motion information. The international video transmission standards, H.261, H.263, MPEG-1, MPEG-2, and MPEG-4 all adopt the block size of 16x16 pixels.

The basic operation of block-matching involves selecting a candidate block and calculating the matching function (usually a non-negative function of the intensity difference) between the candidate and the current block. This operation is repeated until all candidates have been processed and the best match identified. The relative distance between the best candidate and the current block is the estimated MV.

Several parameters are involved in the searching process and all have an impact on both accuracy and complexity: -

- the number of search points (candidate blocks).
- the matching function.
- the search order of candidates.

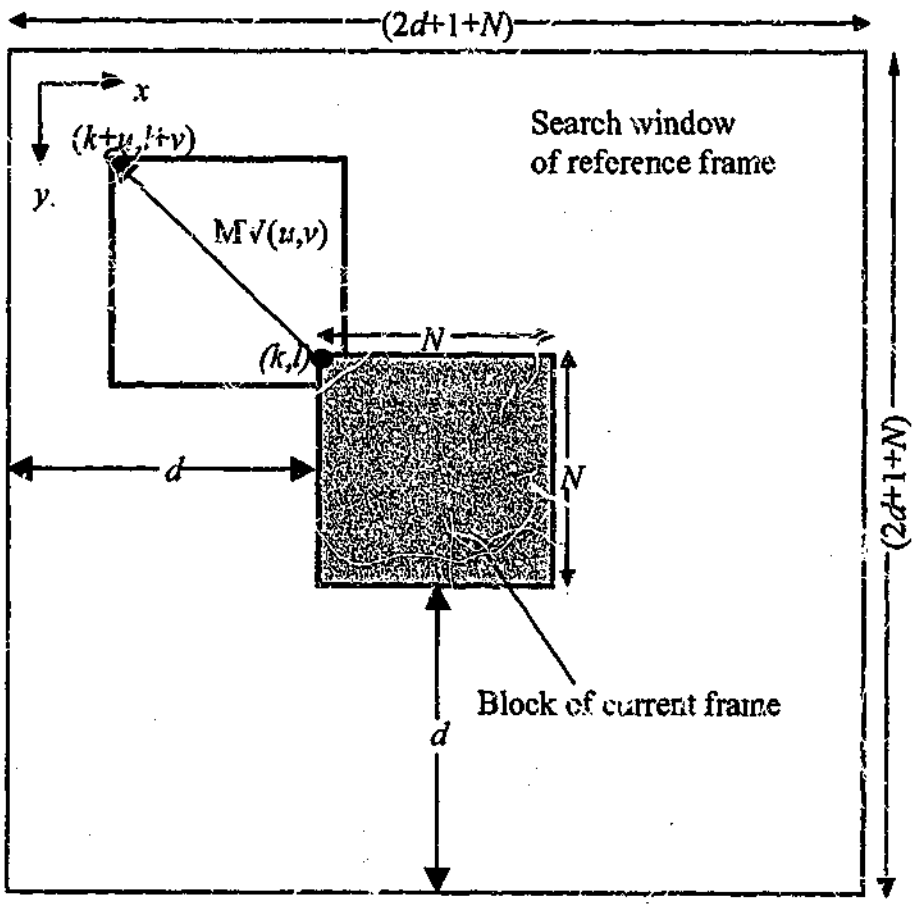


Fig. 2.2: Search space of block-matching algorithms.

Assume that the maximum displacement of a motion vector is $\pm d$ in both the horizontal and vertical directions as shown in Fig. 2.2 (throughout this thesis, pixels of a frame are numbered using the Cartesian coordinate system with the origin starting in the upper-left corner. Thus far, integer-pel MV is considered, although in Section 2.4.10, half-pel motion accuracy will be discussed). Except for the blocks at the image boundaries, the size of the search space is

(N+2d+1)×(N+2d+1), and therefore a MV(*u,v*) is obtained by finding a matched block within the above mentioned search space in the reference frame by using a predetermined matching criterion, where the number of possible search points is *(2d+1)×(2d+1)* for the best match of the current block, as shown by Fig. 2.2. The algorithm, which examines all these locations, is called the *Exhaustive Search*, or *Full Search* (FS). If more than one block generates a minimum BDM, the FS algorithm selects the block whose MV has the smallest magnitude, to exploit the centre-biased motion vector distribution characteristics of real video sequences [45, 46]. To achieve this, checking points trace a spiral trajectory starting at the centre of the search space. This trajectory is used by the FS algorithm, with the example of a maximum displacement d = ±7 shown in Fig. 2.3. The centre of the search window is equal to the location of the searching block (current block) of the current frame.

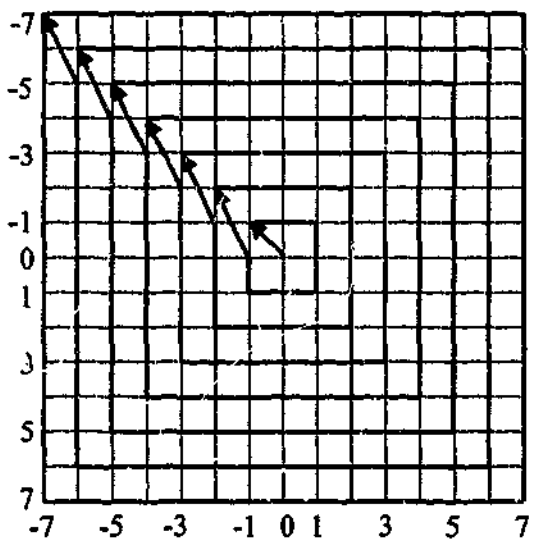


Fig. 2.3: The spiral trajectory of the checking points in the FS algorithm.

2.3.1.2 Block-Matching Criteria

Choosing an appropriate matching function is an important part of the process of searching for the best matching block. The selection of the matching function has a direct impact on computational complexity and coding efficiency. Several popular matching functions that appear in the literature are *Mean Absolute Error* (MAE), *Mean Square Error* (MSE), *Cross Correlation Function* (CCF), and *Matching Pel-Count* (MPC).

If $F_n(i,j)$ denotes the intensity of the pixel with coordinates (i,j) for the current frame, then the MAE, MSE, CCF, and MPC matching criteria are defined as follows:

***Mean Absolute Error* (MAE):** In this criterion, the pixels from each block in the current frame *n*, are compared with the corresponding candidate block in the search area in the reference

frame, n-1, and their absolute differences are summed and averaged. The MAE criterion is given as follows:

$$MAE_{(k,l)}(u,v)=\frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|F_n(k+i,l+j)-F_{n-1}(k+u+i,l+v+j)| \text{ for } -d\le u,v\le d \quad (2.1)$$

where $F_{n-1}(u,v)$ is a candidate block in the search space in the reference frame, and (u, v) is a motion vector representing the search location. The search space is specified by $u = (-d, +d)$ and $v = (-d, +d)$. The candidate block with the minimum MAE is considered to give the best match. The MAE is also known as *Mean Absolute Difference* (MAD).

Mean Square Error **(MSE):** This is similar to the MAE function, except that the difference between pixels is squared before addition. The MSE function is defined as:

$$MSE_{(k,l)}(u,v)=\frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}[F_n(k+i,l+j)-F_{n-1}(k+u+i,l+v+j)]^2 \quad (2.2)$$

The candidate block with the minimum MSE is considered to give the best match. The *Mean Square Error* (MSE) function is also known as the *Mean Square Difference* (MSD).

Cross Correlation Function **(CCF):** The *Cross Correlation Function* (CCF) for the block-matching criterion is derived from the following equation:

$$CCF_{(k,l)}(u,v)=\frac{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}F_n(k+i,l+j)F_{n-1}(k+u+i,l+v+j)}{\sqrt{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}F_n^2(k+i,l+j)}\sqrt{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}F_{n-1}^2(k+u+i,l+v+j)}} \quad (2.3)$$

The candidate block with the maximum value of CCF is considered to give the best match. This is also referred to as a *Normalized Cross-correlation Function* (NCF) [82].

Matching Pel-Count **(MPC):** The *Matching Pel-Count* (MPC) function compares each pixel of the target block of the current frame, n, with the corresponding pixel block within the search space of the reference frame, n-1. If the pixels are similar to each other, the pixel pair is classified as a matching pixel; if not, then it is a mismatching pixel. The matching and the mismatching classifications are done with respect to a pre-defined threshold value. The MPC criterion is as follows:

$$MPC(u,v)=\sum_{i=1}^{N}\sum_{j=1}^{N}P(u,v,i,j) \text{ for } -d\le u,v\le d \quad (2.4)$$

$P(u,v,i,j)$ is the binary representation of the pixel difference defined as :

$$P(u,v,i,j)\begin{cases}=1 & if\left|F_n(k+i,l+j)-F_{n-1}(k+u+i,l+v+j)\right|\le T_p\\ =0 & \text{otherwise}\end{cases} \tag{2.5}$$

where T_p is a pre-defined threshold value. For a matching pixel, P = 1, while for a mismatching pixel, P = 0. The greater the number of matching pixels, the better the match.

Among the above matching criteria, MSE and CCF require multiplication and accumulation, while MAE and MPC require comparison and accumulation, and a multiplication is always more computationally expensive than either a comparison or accumulation. Although MPC is computationally less expensive compared with the MAE, its performance is highly sensitive to the choice of the threshold value T_p [74]. The optimum threshold selection is a difficult task which entirely depends on the video sequence, and therefore, its performance is not guaranteed.

Among the various matching criteria therefore, MAE is the most popular and widely used in block motion estimation due to its low complexity, while its performance is comparable to that of MSE. For this reason the MAE function will be used throughout the thesis for BDM calculations, while the MSE function, as is the convention, will be used for prediction error performance analysis of BMAs [45-47, 51-54].

2.3.1.3 Motion Estimation Algorithm Complexity

Motion estimation algorithm optimisation has been widely studied because of its fundamental impact on compression efficiency and its high requirements in both processing power and data bandwidth.

In this section, a complexity analysis of motion estimation is reviewed. At the same time, the advantages and disadvantages of the different approaches, both in terms of compression quality and processing speed efficiency, are characterised. Fig. 2.2 showed the motion estimation process for a current macroblock of size $N\times N$ within a search range of $\pm d$ (horizontally and vertically) in the reference frame. To find a motion vector with minimum BDM for this current macroblock, the computational complexity of performing the motion estimation is given by:

$$Complx_{MB} = SP\times\left[(N\times N)\times Complx_p\right] \tag{2.6}$$

This shows that the complexity, $Complx_{MB}$, is proportional to the number of *search points*, SP, the number of pixels used to perform the matching ($N\times N$), and the complexity involved in evaluating one pixel match, $Complx_P$. To illustrate the complexity, consider a typical application of a $\pm d$ pixels maximum displacement used for a video sequence with frame size $[N_h, N_v]$ and a frame rate of f fps (frames per second). The total number of integer arithmetic operations per second required for an MAE-based FS algorithm is

$$\left(\frac{N_h \times N_v}{N^2}\right) \times (2d+1)^2 \times 3N^2 \times f \tag{2.7}$$

For example, for a typical MPEG-1 application of a 15×15 pixels search space used for a video sequence with SIF format (352×240 pixels) and 30 fps, based on (2.7), the FS algorithm would require about 1.71 billion integer arithmetic operations per second, which can consume up to 90% of the computational power of the whole encoding system [82]. When considering applications which require encoding at higher resolutions and at higher quality, it is evident that the FS algorithm has severe limitation in a real-time implementation.

Several fast algorithms, which will be discussed in next section, have thus been devised to save computational complexity in the FS algorithm but at the price of impaired quality performance. The most common approach is to lower the search computation by reducing the number of SP in (2.6) inside the defined search space. Normally, a fast search algorithm starts with a rough search, computing a set of scattered points. The distance between two nearby search points is called the search *step size*. After the first step is completed, the search moves to the most promising search point and continues with the next step. This process is continued until satisfaction with some predefined conditions for motion estimation is established.

2.4 Fast Search Motion Estimation Algorithms

Some of the well-known types of BMAs are now reviewed, where only integer-pel accuracy is considered. All fast search algorithms assume that either the error surface is *unimodal* over the entire search area (i.e. there is only one global minimum) or the MV is centre-biased. These assumptions essentially require that either the BDM increases monotonically as the search point moves away from the global minimum position, or the MV is centrally distributed.

2.4.1 2D-log Search

The first fast search block-matching algorithm was the *2-D log search* (TDL) proposed by Jain and Jain in [36], and is an extension of the 1-D binary logarithm search. It uses the St. Andrew's cross (+) search pattern in each step. The initial step size is equal to $\max(2, 2^{m-1})$, where $m = \lfloor \log_2 d \rfloor$ and $\lfloor \cdot \rfloor$ is a lower integer truncation function, and d is the maximum displacement. The step size is reduced by half only when the minimum BDM point of the previous step is found at the centre of that step, or the current minimum BDM point reaches the search window boundary, otherwise, the step size remains constant. When the step size is reduced to one, all the eight checking points adjacent to the centre checking point of that step are searched. For example, two different search directions are shown in Fig. 2.4 for the case $d = \pm 7$. The top

search path requires (5+3+3+8)=19 checking points, while the lower-right path requires (5+3+2+3+2+8)=23 points. The number of search steps is dependent on the size of the search area, and is not given by the definition of the algorithm. According to [48], for the general case, the TDL algorithm requires $\left(1+7\lceil \log_2(d+1) \rceil\right)$ checking points. This technique has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of candidates to evaluate.
- Reasonable performance for high motion sequences.

Disadvantages:

- Poorer performance for low motion sequences as the initial step size is large and increases the chance of its being trapped in local minima.
- Complexity increase with increased search area.
- Fixed performance for a particular video sequence.
- Lack of a control mechanism for performance scalability.

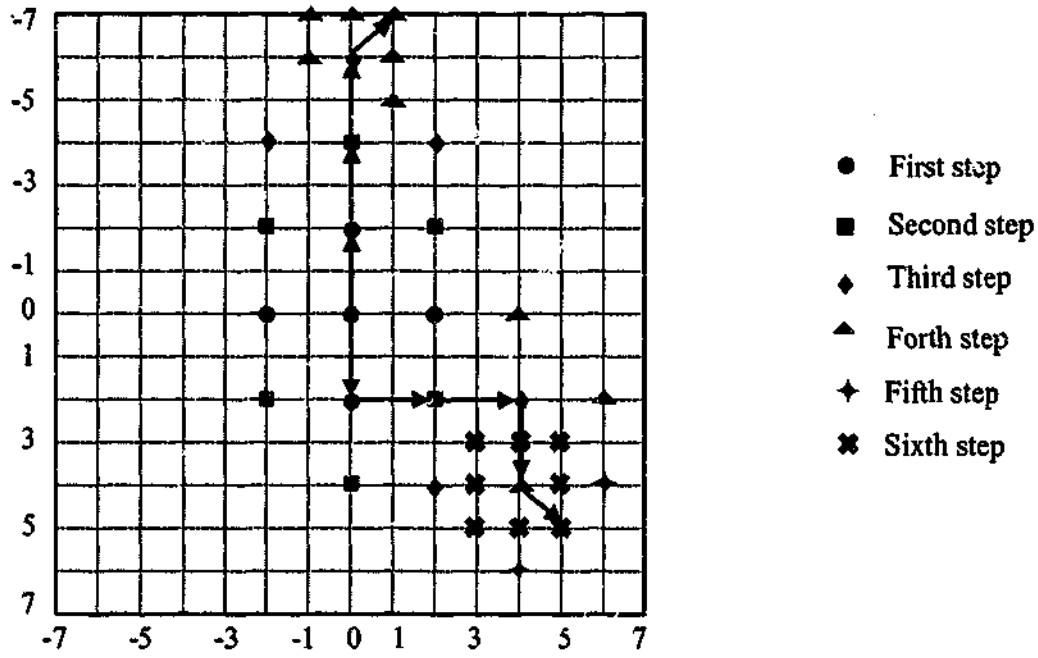


Fig. 2.4: 2D-log search.

2.4.2 Three Step Search

The *three step search* (TSS) algorithm proposed by Koga *et al.* in [44] is probably the most well-known and popular technique and is even recommended in several standards because of its simplicity and effectiveness. This method is based on a coarse-to-fine approach with logarithmically decreasing step sizes as shown in the example of Fig. 2.5 for a maximum displacement of $d = 7$. The initial step size is half of the maximum motion displacement d (i.e.

$\lceil d/2 \rceil$ where $\lceil \cdot \rceil$ is the upper integer truncation function). At each step, nine checking points are matched and the point with the minimum BDM is chosen as the starting centre of the next step. The process repeats until the distance between these checking points is equal to one. For d =7, the number of checking points required is (9+8+8)=25. For a larger search window (i.e. larger d), TSS can be easily extended to n-steps using the same searching strategy with the number of checking points required defined as $SP = [1 + 8\lceil \log_2 (d+1) \rceil]$. This technique has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of candidates to evaluate.
- Good regularity in terms of motion vector generation.
- Reasonable performance with high motion sequences as it considers the uniform motion distribution.

Disadvantages:

- Poorer performance with low motion sequences as the initial step size increases linearly with maximum displacement that increases the chance of its being trapped in local minima.
- Complexity factor increase with increased search area.
- Lack of a control mechanism to provide performance scalability.
- Complexity is fixed for a particular video sequence with maximum displacement.

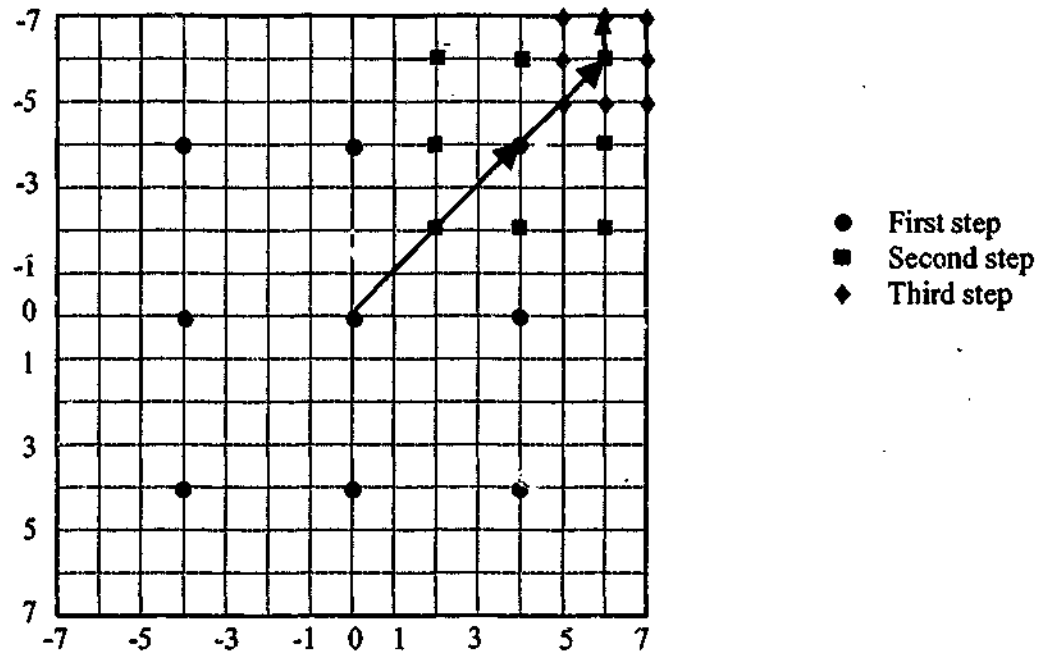


Fig. 2.5: Three step search.

2.4.3 Orthogonal Search

The *orthogonal search algorithm* (OSA) proposed by Puri *et al.* in [50] consists of pairs of horizontal and vertical steps with a logarithmic decrease in step size where the initial step size is $\lfloor d/2 \rfloor$ with $\lfloor \cdot \rfloor$ being the lower integer truncation function. The search paths of an OSA algorithm are shown in Fig. 2.6. Starting from the horizontal searching step, three checking points in the horizontal direction are searched. The minimum BDM point then becomes the centre of the vertical searching step which also consists of three checking points. The step size is halved after each pair of horizontal and vertical steps. The algorithm ends with step size equal to one. For $d = \pm 7$, the OSA algorithm requires a total of (3+2+2+2+2+2)=13 checking points. For the general case, the OSA algorithm requires $(1 + 4\lceil \log_2(d+1) \rceil)$ checking points.

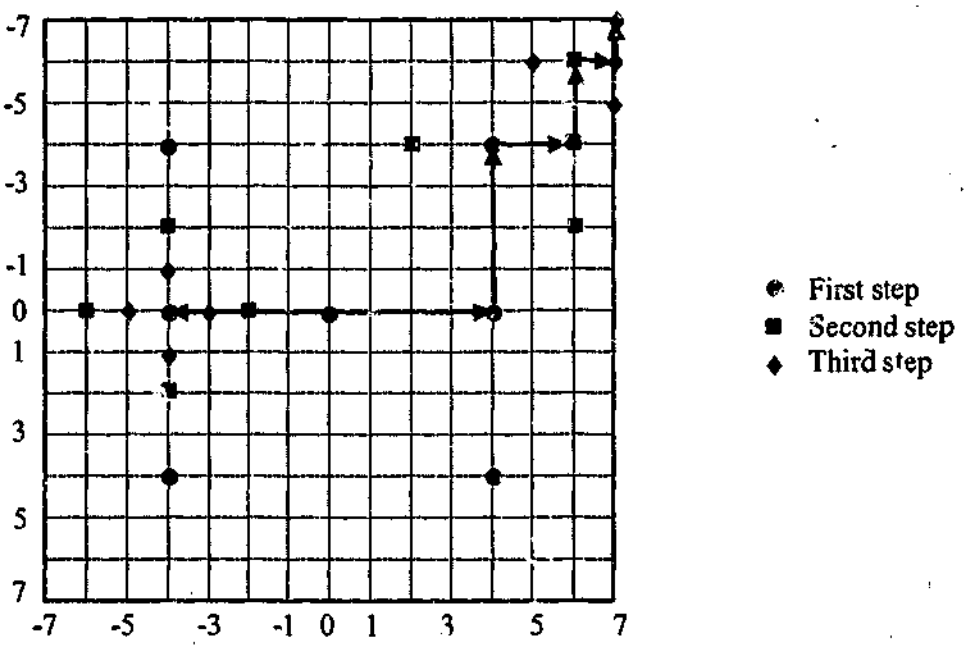


Fig. 2.6: Orthogonal search.

This technique has the following advantages and disadvantages: -

Advantages:

- Lower complexity in terms of search points compared to the TSS or TDL algorithms.
- Reasonable performance with high motion sequences as it considers the uniform motion distribution.

Disadvantages:

- Poorer performance with low motion sequences as the initial step size increases linearly with maximum displacement that increases the chance of its being trapped in local minima.
- Poorer error performance compared to the TSS or TDL algorithms.

- Complexity factor increase with increased search area.
- Lack of a control mechanism to provide performance scalability.
- Complexity is fixed for a particular video sequence with maximum displacement.

2.4.4 Cross Search

The *cross search algorithm* (CSA) proposed by Ghanbari [48] also uses a logarithmic step search algorithm; however, the main difference between this and the logarithmic search method presented in previous sections is that the search location is picked at the end points of a Greek cross (X) rather than a St. Andrew's cross (+) in each step. Fig. 2.7 shows two search paths where there are five checking points placed in a cross pattern at each step. The initial step size is 1/2 of d, and as the step size decreases to one, a (+) cross search pattern (as shown on the lower-left side of Fig. 2.7) is used if the minimum BDM point of the previous step is either the centre of that step, or the upper-left or lower-right checking point. Otherwise, a (X) cross search pattern (as shown on the upper-right side of Fig. 2.7) is used. For $d = 7$, the number of checking points required is (5+4+4+4) =17. For the general case, the number of checking points required is $(5+4\lceil \log_2 d \rceil)$.

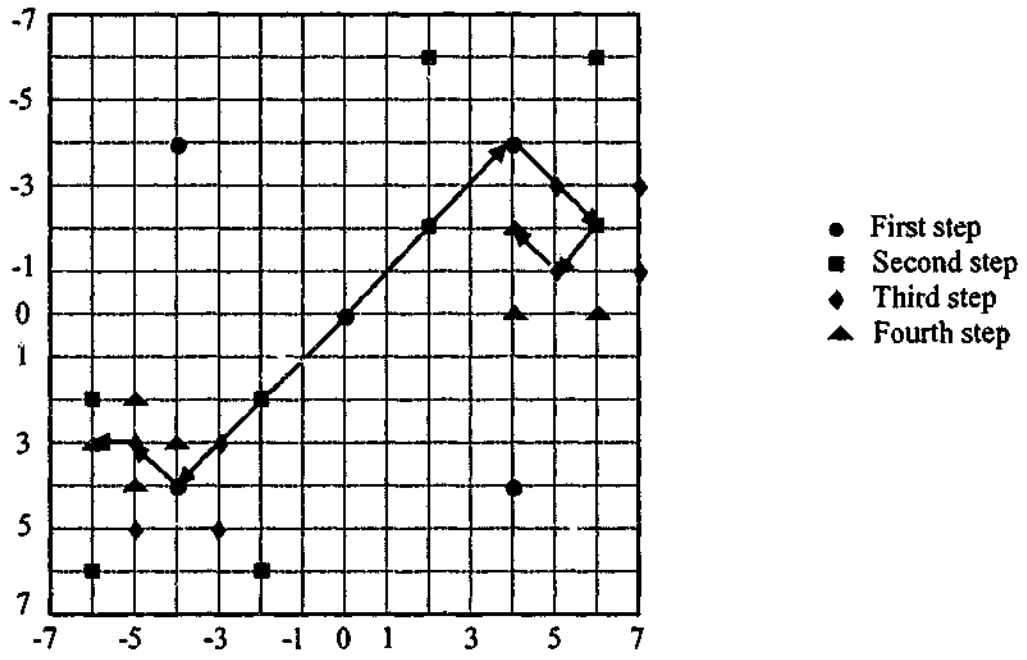


Fig. 2.7: Cross search.

This technique has the following advantages and disadvantages: -

Advantages:

- Low complexity compared to the TDL or TSS algorithms.
- Reasonable performance for high motion sequences.

Disadvantages:

- Not suitable for low motion sequences as the initial step size is high and increases the chance of its being trapped in local minima.
- Complexity increase with increased search area.
- Lack of a control mechanism for performance scalability.
- Prediction error performance is worse compared with the TDL or TSS algorithms.

2.4.5 New Three Step Search

One of the most significant contributions to motion estimation came from Li *et al.* [45], who observed that most real world video sequences usually move slowly and vary gently. There are, for example, low motion video conferencing sequences, where a such motion type is very common. This essentially leads to a centre-biased global minimum motion vector distribution rather than a uniform distribution, which was the assumption used in the TSS algorithm. By employing a centre-biased checking pattern combined with the initial algorithm of TSS, an improvement resulted called the *new three step search* (NTSS) [45]. Compared with the TSS, an additional eight neighbour checking points are searched in the first step of NTSS as shown in Fig. 2.8. The figure shows two search paths with $d = 7$. The centre path shows the case where low motion is searched. In this case, the minimum BDM point of the first step is one of the eight neighbour checking points. The search is stopped *halfway*, with the matching of three more neighbouring check-points at the minimum BDM point of the first step. The number of checking points required for this centre path is (17+3) = 20. In the worst case, the total number of checking points with a maximum displacement, d, is $(8 + 8\lceil \log_2 (d+1) \rceil)$. This algorithm has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of search points to be evaluated.
- Obtains high quality at low processing power for a centre-biased motion scene.

Disadvantages:

- Candidate vector generation is more complex. Loses the regularity and simplicity of the TSS algorithm.
- Complexity increases with the increased search area.
- High probability of falling into local minima if the sequence is not centre-biased.
- Lack of a control parameter for performance scalability.
- Fixed complexity for a particular video sequence.

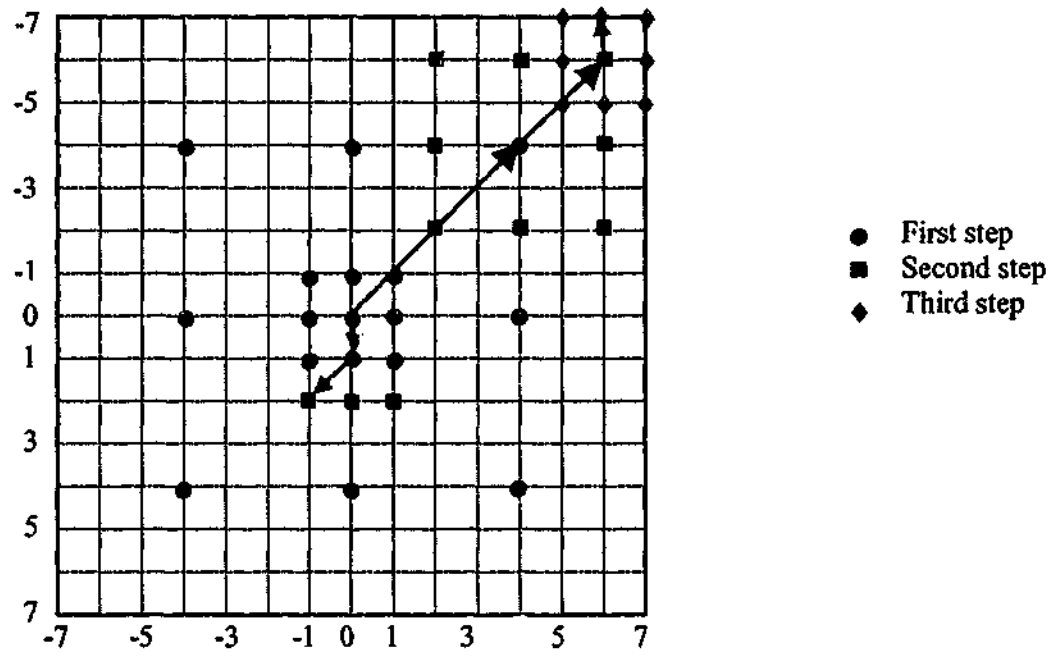


Fig. 2.8: New three step search path.

2.4.6 Block-based Gradient Descent Search

The *block-based gradient descent search* algorithm (BBGDS) proposed by Liu [52] uses a centre-biased search pattern of nine checking points in each step, with a step size of one. It does not restrict the number of searching steps but it stops when the minimum BDM point of the current step is the centre of that step, or it reaches the search window boundary. There are also overlapped checking points between adjacent steps. Two low motion search paths examples are shown in Fig. 2.9. The average number of checking points is $\left[9+\sum_{i}^{n}\beta_i\right]$ where $\beta_i \in \{3,5\}$, and $i = 2,3,...,n$ where n represents the number of search steps. This algorithm has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of search points to be evaluated.
- Good regularity in terms of motion vector generation.

Disadvantages:

- Optimised for only low motion video sequences (i.e. video conferences), as it is highly centre-biased.
- Unable to capture high motion vectors unless very large number of steps are used.
- High probability of falling into local minima if the sequence is not centre-biased.
- Lack of a control parameter to provide flexibility for performance scalability.

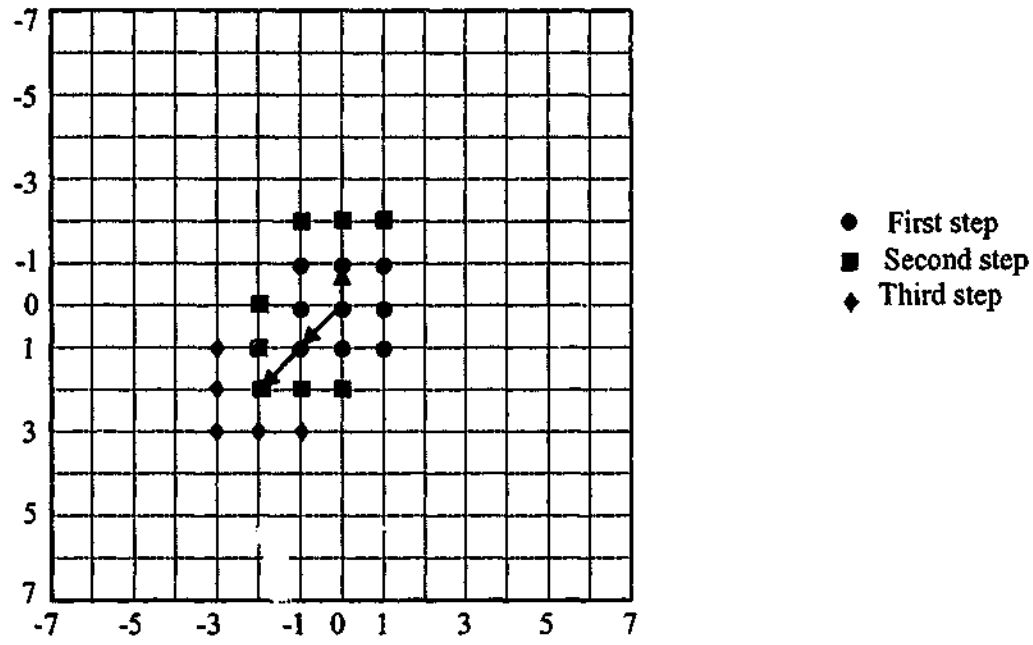


Fig. 2.9: Block-based gradient descent search path.

2.4.7 Four Step Search

The *four step search algorithm* (FSS) proposed by Po and Ma [46] also exploits the centre-biased characteristics of real world video sequences by using a smaller initial step size compared with TSS. The initial step size is a quarter of the maximum motion displacement d (i.e. $\lceil d/4 \rceil$). Due to the smaller initial step size, the FSS algorithm needs four searching steps to reach the boundary of a search window with $d = \pm 7$. Fig. 2.10 shows two search paths in an FSS algorithm for searching high motion. The lower-left path requires (9+5+3+8) = 25 checking points while the upper-right path requires (9+5+5+8) = 27 checking points, which is the worse case for this algorithm for $d = \pm 7$. This algorithm follows the *halfway-stop* technique used by the NTSS algorithm in its second and third search steps for low motion video sequences. Moreover, if the minimum BDM point is found at the centre of that search step, the step size is reduced by half and the process jumps to the fourth step. For the general case, the algorithm can be extended as follows. If the step size of the fourth step is greater than one, then another four step search is performed with the first step equal to the last step of the previous search. The number of checking points required for the worse case is $\left(18\left\lceil \log_2 \frac{d+1}{4} \right\rceil + 9\right)$. This algorithm has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of search points to be evaluated.
- Faster convergence than BBGDS if the motion vector is far from the centre.

- Suitable for centre-biased motion scenes.

Disadvantages:

- Complexity increases with the increased search area.
- High probability of falling into local minima if the sequence is not central-biased.
- Lack of a control parameter for different applications
- Fixed complexity for a particular video sequence.

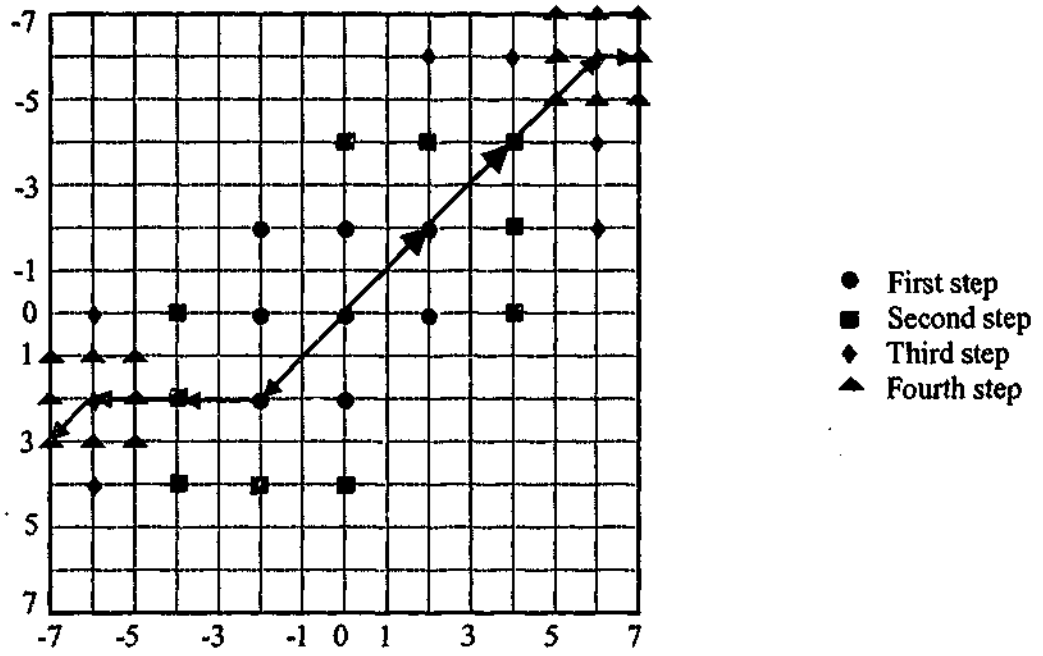


Fig. 2.10: Two different search paths in a four step search.

2.4.8 Diamond Search

The *diamond search* (DS) algorithm proposed by [53, 83] is another efficient search algorithm for central-biased motion distributed video sequences. The search always starts from the centre of the search window, by examining nine checking points as shown in Fig. 2.11(a). If the minimum is found at the centre, then the four additional checking points, shown in Fig. 2.11(b), are matched and the search stops. Otherwise, depending on the position of the minimum, for example Fig. 2.11(c) for a corner point, additional points are examined and the centre of the search is now considered to be the new minimum. This procedure continues until the minimum is found to be in the centre. The number of search points depends on the video sequence and the search window. This approach has the following advantages and disadvantages: -

Advantages:

- Low complexity in terms of search points to be evaluated.
- High performance with centre-biased low motion video sequences.

Disadvantages: -

- High probability of falling into local minima if the sequence is not centre-biased.
- Complexity is high with high motion video sequences.
- Fixed complexity for a particular video sequence.
- Lack of a control parameter for performance scalability.

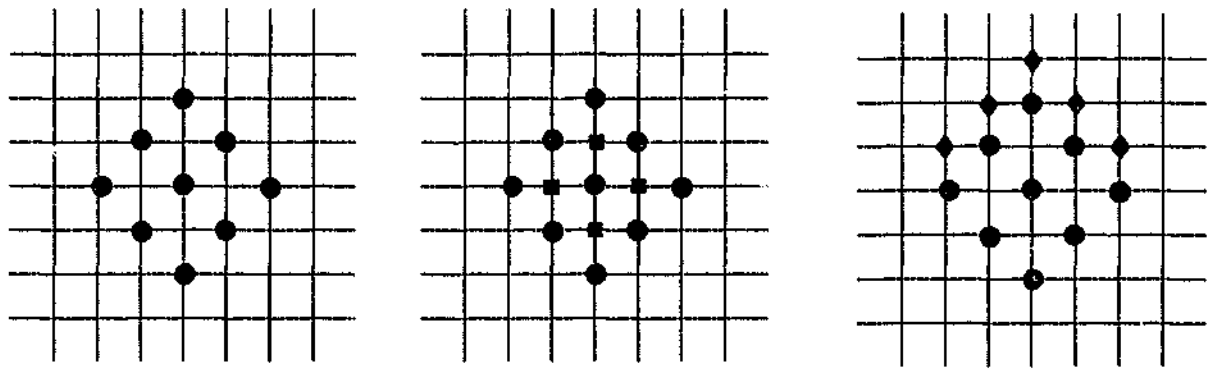


Fig. 2.11: Diamond search.

2.4.9 Concluding Comments

In reviewing the general attributes of the fast search algorithms discussed in the section above, the following conclusions can be drawn: -

Advantages:

- All existing fast BMAs reduce computational complexity in terms of the number of search points the FS algorithm needs to estimate motion vectors.
- Most have good regularity in terms of motion vector generation.

There are, however, a number of limitations associated with these fast BMAs: -

- Lower complexity is achieved only by sacrificing quality.
- All are directional search techniques whose performance depends on the *unimodal error surface assumption* (USEA). There is a probability of falling into local minima if this assumption does not always hold true.
- The complexity factor increases with the search area. The number of steps increases linearly with *d,* so increasing the probability of falling into local minima.
- All are application-dependent. For example, TSS performs reasonably well on uniformly distributed motion video sequences but has very poor performance with centre-biased low motion video sequences. On the other hand, the NTSS performs better with centre-biased motion distributed sequences (low motion) only.
- There is no single mechanism or parameter to provide any flexibility in controlling the quality as well as the complexity for motion estimation in different applications.

None of the fast BMAs can satisfy any prescribed level of QoS in terms of prediction image quality or processing speed.

- They are not suitable for real-time software-only or low power video coding, which requires a more flexible approach in the trade-off between quality and complexity.

The above discussion leads to the conclusion that although there are a number of fast algorithms available for motion estimation, there still remain many challenges. Amongst the key issues are directionality, application dependency, performance scalability for QoS, and flexibility for real-time software or low power video coding applications. This thesis seeks to address all of these in the development of a new motion estimation system.

The various examples that have been included in this section used integer-pel accuracy motion estimation. While fast algorithms reduce computational complexity in terms of the number of search points by sacrificing image quality, the prediction quality of these search algorithms can be improved by considering sub-pel accuracy motion estimation, as discussed below.

2.4.10 Half-pel Accurate Motion Estimation

Subpixel motion estimation has become a main ingredient in many modern video compression standards [84]. This is because sub-pel accuracy (half-pel/quarter-pel) motion estimation improves the performance of BMA by finding a better matching block in the search window.

Although digital video is represented by pixels, the moving object is not necessarily limited to moving by an integer number of pixels between successive video frames. So, the true frame-to-frame displacements are unrelated to the integer-pel sampling grids. Representing fractional motion vectors gives sub-pel accuracy to motion compensation. Here, only half-pel accuracy is considered.

Searching using half-pel accuracy obviously requires more computational complexity than integer-pel. In order to limit the increase in complexity, it is common practice to first find the motion vector with integer-pel accuracy using any BMA, and then to carry out a search using the eight neighbouring half-pel positions blocks. These half-pel position blocks are calculated using bilinear interpolation as shown in Fig. 2.12, where A, B, C, and D represent the integer pixel values and a, b, c, and d represent the pixel values at the half-pel level. The example in Fig. 2.13 highlights that the minimum BDM will be at the centre of the search window, with a motion vector of (u,v) = (0,0) when using integer-pel accuracy, but after checking half-pel positions, a smaller BDM is found at the (0,0.5) position. Despite the increased complexity, half-pel motion estimation and compensation can significantly improve motion prediction accuracy since it reduces noise by averaging and interpolating the pixels. For this reason, many

video coding standards such as MPEG-1/2 [9, 10] and H.261/263 [11, 12] permit motion vectors to be specified to a half-pel accuracy.

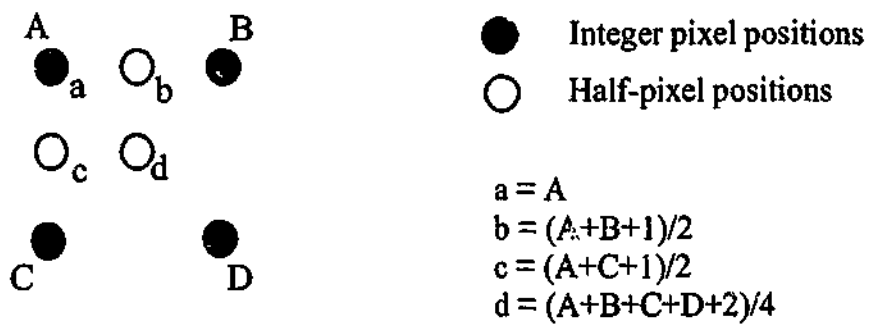


Fig. 2.12: Half-pel prediction by bilinear interpolation.

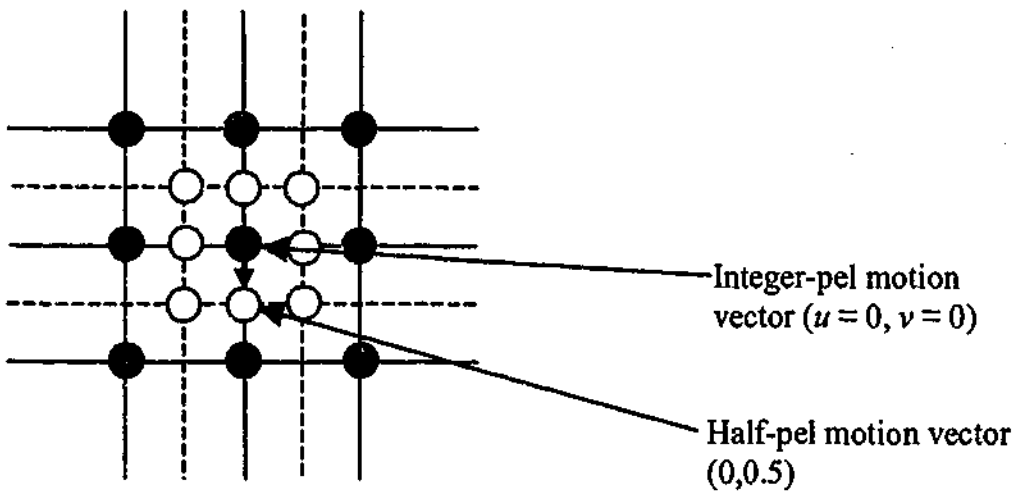


Fig. 2.13: Half-pel accurate motion vector estimation.

2.4.11 Variants of Block-Matching Algorithms

The fast search algorithms, based on UESA, described in previous sections, are designed to reduce computation in the process of finding the best matching block in the search window. There are also some other approaches which can, in general, be integrated into the BMAs mentioned in the previous sections so as to further improve the search efficiency. These algorithms are based on using the concept of *inter-block motion correlation* [49, 66-68, 85-90], the consideration of a subset of pels (pixel subsampling) inside the image blocks when computing the matching function [54, 64, 65], and a multiresolution approach [91-93]. There also exist some different fast-matching motion estimation techniques [94-97] which can be integrated into the FS algorithm to improve its search efficiency by reducing the computational cost in the number of operations.

In the following section these different types of BMAs are described based on the above options to improve the search efficiency for computational cost minimisation with, or without, low quality degradation.

2.4.11.1 Inter-block (Spatio-Temporal) Correlation

The motion estimation search strategies previously described imply a fixed initial starting point that can be centered on the origin of the search window. However, the spatio-temporal correlation of the motion vector fields are often high and can be used to predict a better starting point, other than the centre of the search window, which reflects the trend of the current block's motion, and therefore, may lead to obtaining motion vectors with less BDM using fewer search points [66].

Indeed, usually the objects span over several macroblocks and move mostly uniformly from frame to frame. A motion tracking algorithm proposed in [86] used the previous frames' motion vectors in the neighborhood of the current block to form an initial estimate of the current block motion vector. Spatial as well as temporal motion vector correlation as an offset vector to track the motion vector of the current block is used in [49, 67, 68, 87-90]. In [68], Luo *et al.* proposed an algorithm utilising the linear weighting of the motion vectors of the three adjacent blocks to obtain a prediction motion vector, namely the initial search point. Xu *et al.* in [49] and Cheung *et al.* in [85] used the spatial relation to predict the initial search centre and then used the centre-biased block matching algorithm [67, 68] to refine the final motion vectors. All these approaches have the following advantages and disadvantages: -

Advantages: -

- Incorporating these schemes in any fast search algorithms reduces the computational cost and motion vector overhead.
- It also increases the possibility of finding the global minima.

Disadvantages: -

- In case of acceleration or moving object boundaries, this technique may become trapped in a local minimum due to inaccurate initial estimates.
- Temporal correlation requires a large memory buffer to keep the previous frame motion vectors in the decoder.

As the above limitations only apply to temporal, and not to spatial correlation, this technique will be exploited further as discussed in Chapter 4.

2.4.11.2 Pixel Subsampling

Since block-matching is based on the assumption that all pixels move in the same way, a good estimate can be obtained by using only a fraction of the pixels in the block to be matched. As this approach only considers a subset of the pixels in the matching macroblock, it reduces overall computational complexity in terms of the number of operations required for motion vector estimation.

A straightforward approach to pixel suabsampling is to adopt a fixed chess-board like pattern with subsampling factors ranging from two to eight with an equivalent saving in complexity. An example of this class of fast algorithm is the simple 4:1 pixels subsampling technique [54, 64, 65] shown in Fig. 2.14. However, since only a uniform fraction of the pixels are used in the matching computation, the use of these standard subsampling techniques can seriously affect the accuracy of motion vector detection, and the computational cost is only reduced by four compared to the FS algorithm.

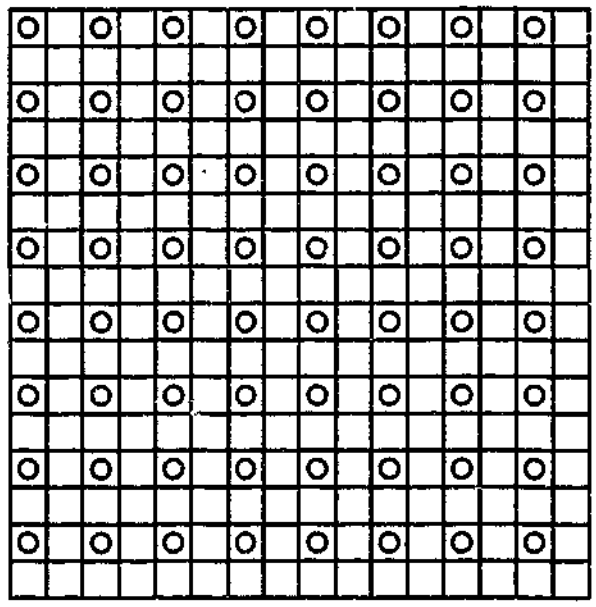


Fig. 2.14: 4:1 pixel subsampling.

Liu and Zaccarin in [54] proposed a popular subsampling algorithm referred to as alternating pixel subsampling. This corresponding 4:1 pixel subsampling pattern consists of alternating over the locations searched so that all pixels of a block contribute to the computation of the motion vector. Fig. 2.15 shows a block of 8×8 pixels with each pixel labeled as a, b, c, or d in a regular pattern. This method considers all four subsampling patterns, but only one at each location of the search area, and in a specific alternating manner. The one that has a minimum BDM among the four is selected as the motion vector for the block. Alternating between these patterns allows the use of all the pixels of the current block and all the pixels of the search area. Nevertheless, though the performance is better than the standard 4:1 subsampling method, the

computational complexity of the FS algorithm is also reduced only by a factor of four. To rectify this problem, in [98], Chan *et al.* proposed an adaptive pixel subsampling technique instead of the regular pixel pattern in [54], where a lesser number of pixels are considered with uniform intensity blocks, and more pixels are considered with high active blocks for the BDM calculation. The above mentioned subsampling algorithms have the following advantages and disadvantages: -

Advantages:

- All the macroblock positions in the search area are covered.
- Regular data flow and easy generation of candidate motion vectors is provided, while pixel extraction complexity is dependent on the algorithm used.

Disadvantages:

- Only a small complexity reduction factor is achieved (typically four to eight) which is often inadequate for real-time applications.
- High probability of falling into a local minimum if a scene contains significant spatial detail.
- The special structure of alternating pixel subsampling makes it difficult to embed within algorithms such as TSS and NTSS.

As a result of these disadvantages, this technique for motion estimation will not be considered any further in this research.

a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	d	b	a	b	a	b
c	d	c	d	c	d	c	d

Fig. 2.15: Alternating patterns of pixels used for computing the matching criterion with 4:1 subsampling.

2.4.11.3 Hierarchical Block-Matching Algorithm

A different approach, the *hierarchical block-matching* proposed in [91-93], can improve the prediction performance of the BMA. The basic principles are similar and can be summarised as follows. A large block size is chosen at the beginning to obtain a rough estimate of the motion vector. By considering a large block, the ambiguity problem - blocks of similar content – can

often be eliminated, although motion vectors estimated from large blocks are not accurate. It is then possible to refine the estimated motion vectors by decreasing the block size and the search region. A new search with a smaller block size starts from an initial motion vector that is the best matched motion vector in the previous stage. As the pels in a small block are more likely to share the same motion vector, the reduction of block size typically increases motion vector accuracy. In hierarchical block matching (multiresolution), the basic idea is to perform motion estimation at each level successively, starting with the lowest resolution level as shown in Fig. 2.16. The estimate of the motion vector at a lower resolution level is then passed onto the next higher resolution level as an initial estimate. The motion estimation at the higher levels refines the motion vector at the lower one. This technique has the following advantages and disadvantages: -

Advantages: -

- All the blocks in the searching area are likely to be covered.

Disadvantages: -

- Memory requirement is increased because of subsampling and pre-stage filtering, and the need to store images at several resolutions.
- Lower accuracy results because of the high probability of local minima when a scene contains significant spatial detail. (This depends on subsampling factors).

As a result of these disadvantages, this technique for motion estimation will not be considered any further in this research.

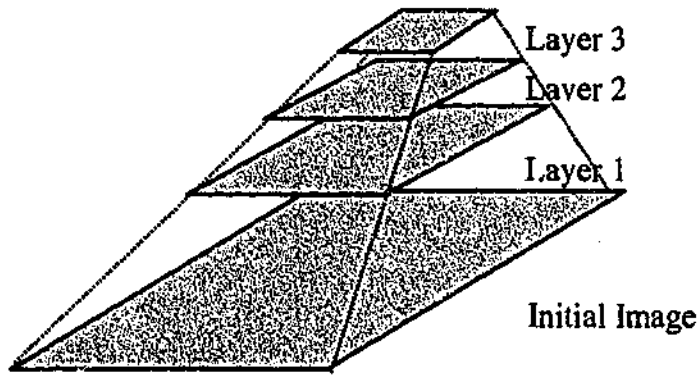


Fig. 2.16: Hierarchical matching pyramid.

2.5 Summary

In this chapter, a comprehensive review of different block-matching motion estimation techniques has been provided. It is clear that a number of key issues in block motion estimation remain to be resolved. In particular, none of the techniques can be seen as a complete solution for all types of motion video sequences. All the fast algorithms are directional and based on the

unimodal error surface assumption, which does not always hold true in real world video sequences. Moreover, these fast algorithms are not designed to provide any flexibility for performance management in motion estimation for QoS in terms of prediction image quality or computational complexity (processing speed).

This, therefore, has motivated the development of a general system, which can be used under all conditions; is non-directional; is scalable for any level of quality of services; and provides flexibility in performance management for real-time video coding applications, especially software-only and low power video coding, where available resources are restricted.

The next chapter details one of the key constituent algorithms, which forms the basis for such a solution.

Chapter 3

Distance-dependent Thresholding Search Algorithm

3.1 Introduction

In this chapter, a non-directional *Distance-dependent Thresholding Search* (DTS) block motion estimation algorithm (**Block 1** in Fig. 1.6) is proposed that employs the novel concept of distance dependent thresholds. As revealed in Chapter 1, this algorithm is based on one key finding from real world video sequences—this is that the distortion of an object in any video frame increases with its velocity, as well as camera zoom and pan factors. To accommodate this key finding, the DTS algorithm uses a *parametric* thresholding function to terminate the search even at relatively high BDM values, especially when the length of the motion vector tends to increase. The DTS algorithm encompasses both the FS and very fast searching modes. Different threshold settings can provide different QoS levels and therefore, the DTS algorithm provides a general solution for all types of video sequences and coding demands. This unique feature provides good flexibility in controlling performance, especially the computational complexity required for motion estimation in real-time video coding applications. Moreover, the non-directional nature of the DTS algorithm means it does not suffer from potential difficulties due to the *unimodal error surface assumption*.

Although the DTS algorithm is developed specifically for video coding applications, it also exhibits significantly improved performance in capturing *true* object motion, a feature which can be used in object motion-based video analysis applications such as video indexing, retrieval, and segmentation, and object detection and tracking for surveillance applications.

This chapter is organized as follows. Error surface analysis of some typical video sequences is presented in Section 3.2. Section 3.3 presents a detailed statistical analysis of video sequences, which provides the key premise for the DTS algorithm. Section 3.4 explains the proposed DTS algorithm with a comparative discussion of the influence of linear and exponential thresholding functions. Experimental results and evaluation of the performance of the DTS algorithm compared with other fast search algorithms are presented in Section 3.5 for video coding applications, together with an analysis of the computational complexity of the

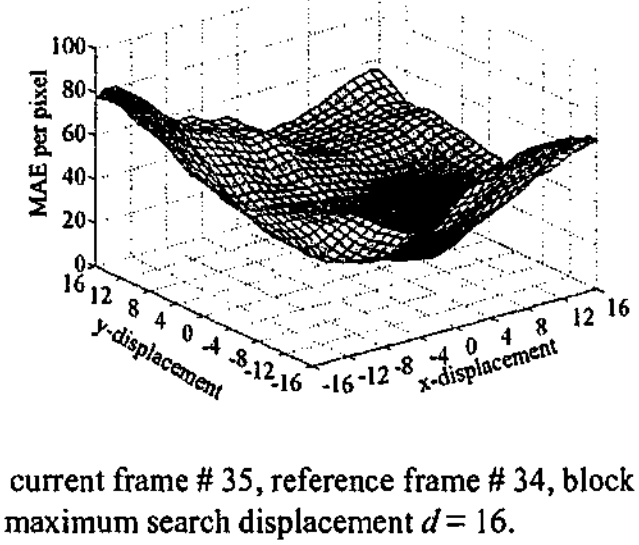
DTS algorithm. Preliminary results for *true* object motion estimation using the DTS algorithm are presented in Section 3.6, while Section 3.7 summarises the chapter.

3.2 The Error Surface and its Characteristics

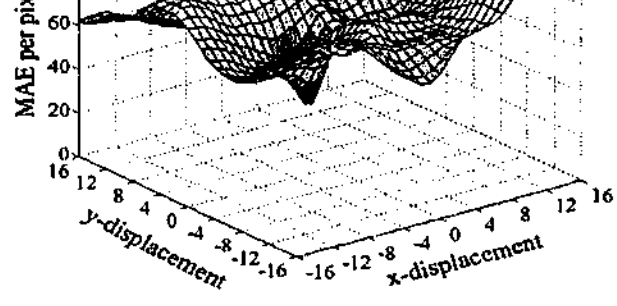
In Chapter 2, it was shown that many contributions in the domain of block-matching algorithms are based on the principle of reducing the checking points in a search window. All of these search algorithms are based on one assumption, that the error surface is *unimodal*. If a matching function, such as MAE in (2.1), is monotonic along any direction away from the optimal point, a well-designed fast algorithm can then be guaranteed to converge to the global optimal point. According to Chow and Liou [99], however, this assumption does not hold true for real world video sequences. We have also tested this on a number of standard video sequences (Appendix B) and observed that this assumption does not always hold true. The MAE error surface has, in fact, a significant impact on the performance of these fast algorithms for block-matching motion estimation. Fig. 3.1 shows some typical examples of MAE surfaces for a search window of ±16 pixels for the *Football* and *Table Tennis* sequences. For the surface shown in Fig. 3.1(a), the MAE error decreases monotonically as the search location moves towards the global minimum value. It implies that most existing fast algorithms, such as TSS and NTSS will perform well for this type of error distributed block.

The MAE surfaces in Figs. 3.1(b) and (c) by contrast, have many local minima due to the non-stationary characteristics of the video signal. As a consequence, it is unlikely that conventional fast search algorithms, which use few directional candidates, would ever converge to the global minima. In other words, the search could easily be trapped in a local minimum instead of the global minima and generate a higher prediction error. A non-directional search, such as FS algorithm, on the other hand, can guarantee reaching the global minima on any kind of error surface at the expense of a large number of search points.

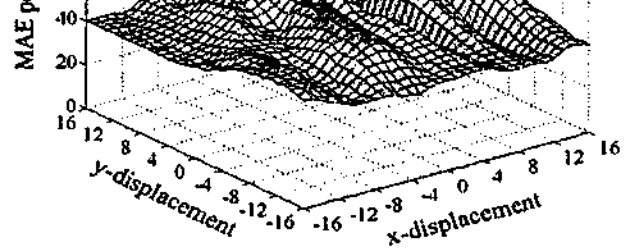
Non-directional search techniques naturally guarantee improved prediction image quality. The key question however, is whether it is possible to develop a non-directional search algorithm fast enough to be comparable to other fast directional search algorithms while retaining its supremacy in achieving improved image quality. This chapter proposes a solution that answers this question affirmatively for low motion video sequences by the development of a *Distance-dependent Thresholding Search* (DTS) algorithm. The next section introduces the basic principles behind the DTS algorithm.



(a) *Football* sequence, current frame # 35, reference frame # 34, block coordinate (5,8), block size 16×16 pixels, maximum search displacement $d = 16$.



(b) *Football* sequence, current frame # 35, reference frame # 34, block coordinate (10,9), block size 16×16 pixels, maximum search displacement $d = 16$.



(c) *Table Tennis* sequence, current frame # 70, reference frame # 69, block coordinate (6,12), block size 16×16 pixels, maximum search displacement $d = 16$.

Fig. 3.1: Three example MAE error surfaces for blocks from the *Football* and *Table Tennis* sequences.

3.3 Rationale in Distance-dependent Thresholding

Before introducing the full DTS algorithm, a series of key definitions need to be provided.

Definition 3.1: Consider two points with Cartesian coordinates (x_1,y_1) and (x_2,y_2). The Euclidean distance, City-block distance, and City-block-max distance between these two points are calculated as $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$, $|x_1-x_2|+|y_1-y_2|$, and $\max(|x_1-x_2|,|y_1-y_2|)$ respectively.

It is interesting to note that the trajectory of a point maintaining constant Euclidean distance, City-block-max distance, or City-block distance from a fixed point traces a circle, a square, and a diamond (45° rotated square) shape, respectively.

Definition 3.2 (Search Squares SS_τ): The search space with maximum displacement $\pm d$, centred at pixel $p_{cx,cy}$, can be divided into d+1 mutually exclusive concentric search squares SS_τ such that a checking point at pixel $p_{x,y}$, representing motion vector $(x-cx, y-cy)$, is in SS_τ if, and only if, the city-block-max distance (Definition 3.1) of the motion vector is $\max(|x-cx|,|y-cy|)=\tau$, for all $-d+cx \le x \le d+cx$, $-d+cy \le y \le d+cy$, and search square index $\tau = 0,1,...,d$.

It can be readily verified that the number of checking points in search square SS_τ,

$$|SS_\tau| = \begin{cases} 1, & \tau = 0 \\ 8\tau, & \tau = 1,2,\ldots,d \end{cases} \tag{3.1}$$

and SS_τ represents the motion vectors of city-block-max distance τ that translates to a conventional Euclidean distance (Definition 3.1) in the range of $[\tau, \tau\sqrt{2}]$ for all $0 \le \tau \le d$. The checking points used in the first three search squares, SS_0, SS_1, and SS_2 are shown in Fig. 3.2.

Throughout this thesis, the length of a motion vector is expressed as Euclidean distance, unless stated otherwise.

Definition 3.3 (Cumulative Probability): Consider a continuous probability function $f(t)$ in a range of fully ordered events $0 \le t \le T$ such that $\int_0^T f(t)dt = 1$. The cumulative probability function $F(t)$ of this probability function $f(t)$ is defined as:

$$F(t) = \int_0^t f(x)dx \tag{3.2}$$

for all $0 \le t \le T$.

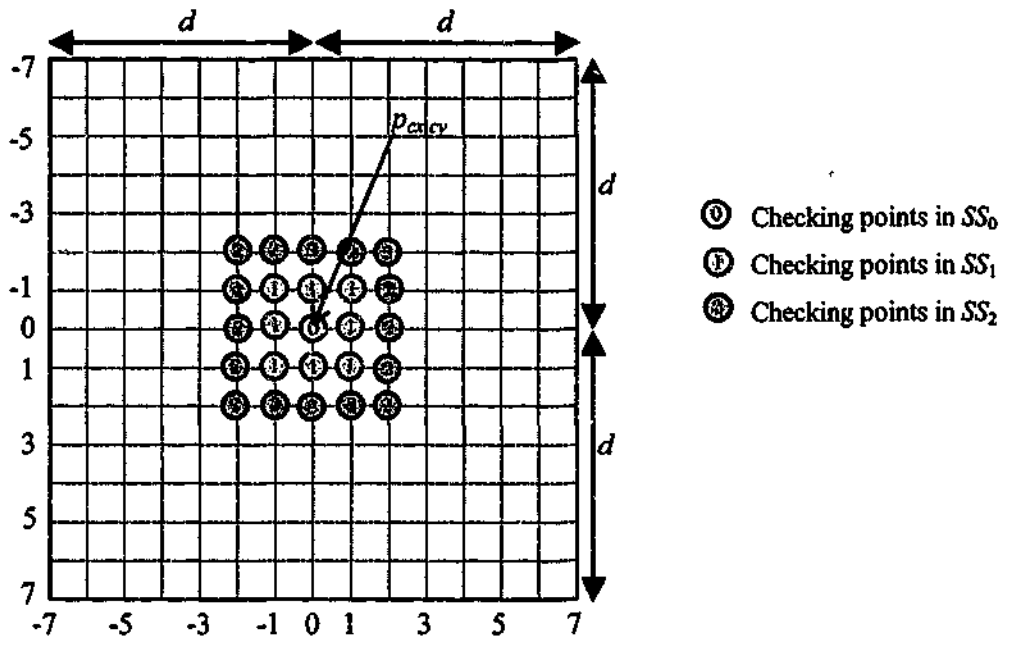


Fig. 3.2: DTS search squares SS_0, SS_1, and SS_2.

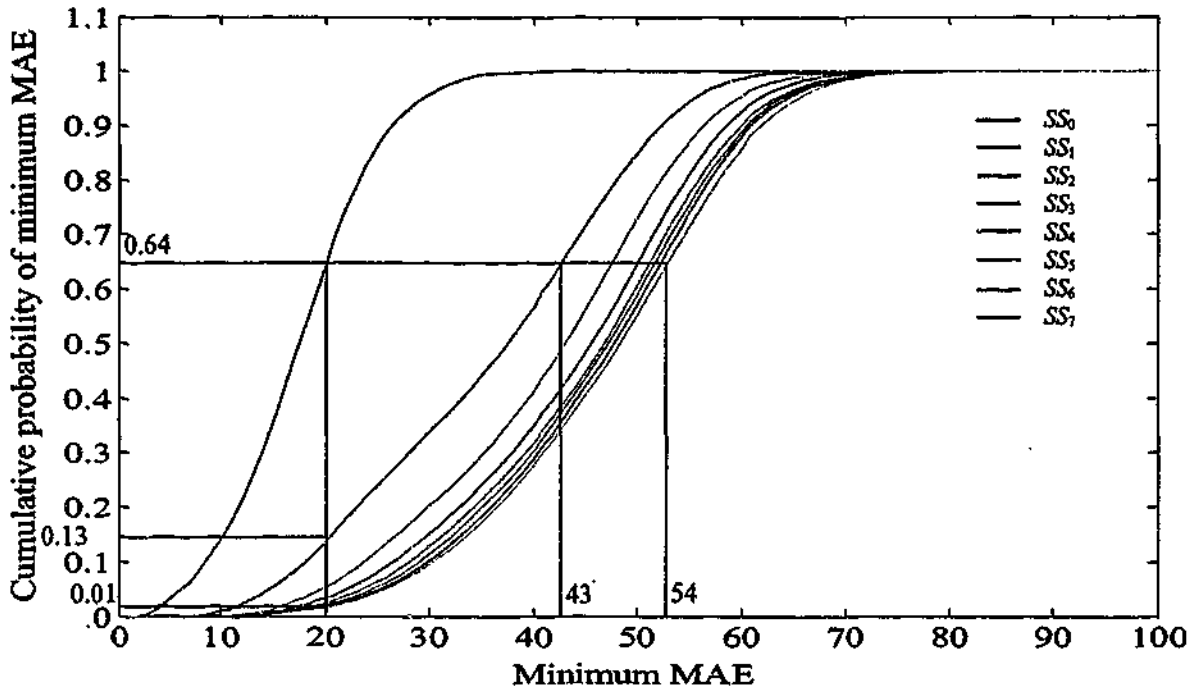
The cumulative probability of an event represents the probability of all possible events up to and including that event, and it can be verified that $F(t_1) \leq F(t_2)$ if and only if $t_1 \leq t_2$ and $F(T) = 1$.

Now consider the average MAE per pixel of a macroblock used as the BDM in the FS algorithm. For each macroblock, the FS algorithm looks for the minimum MAE per pixel value in the range of $[0,2^b\text{-}1]$ for a b-bit gray scale image. Throughout the thesis, it is assumed that $b = 8$.

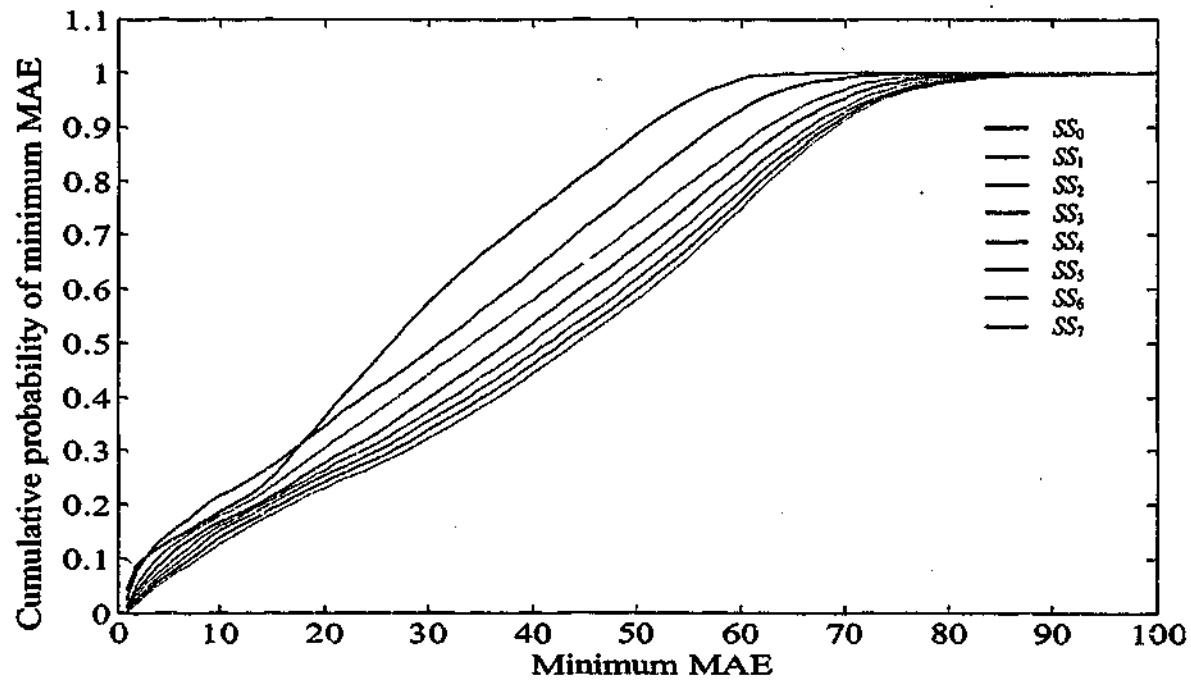
In Feng *et al.* [73] and Lim and Ho [74], it was stated that the magnitude of a motion vector is proportional to the magnitude of the BDM. This observation has been explored further on a number of standard and non-standard video sequences covering a wide range of object and camera motions. In Fig. 3.3, the cumulative probabilities of minimum MAE are plotted for different search squares on four different sequences. For each sequence, the minimum MAE is calculated for each search square of macroblocks in the first 80 frames. In each search square, the probability for each distinct minimum MAE is calculated based on frequency, and cumulative probability using (3.2).

To interpret these graphs, consider the cumulative probability of finding a minimum MAE of 20 or less in individual search square of the *Football* video sequence in Fig. 3.3(a). It follows from this that the cumulative probability gradually decreases from 0.64 to 0.13 and then to 0.01 as the city-block-max distance of a motion vector increases from 0 to 1 and then to 7. This means that as the motion vector length increases, so does the probability of terminating the FS

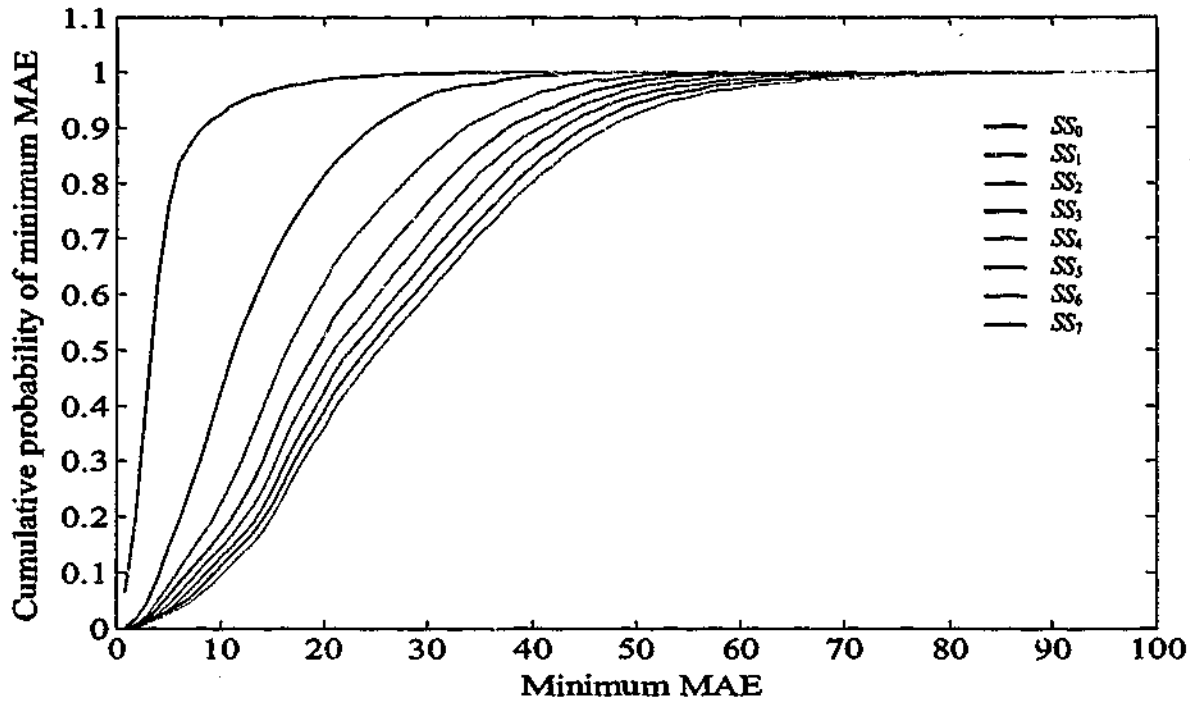
algorithm at a higher MAE value. This observation is further enhanced by the horizontal extension shown in Fig 3.3(a) at a cumulative probability of 0.64, which reveals that the minimum MAE increases from 20 to 43 and 54 as the city-block-max distance of a motion vector increases from 0 to 1 and then to 7. A similar trend is witnessed in the other three video sequences shown in Figs. 3.3(b), (c), and (d).



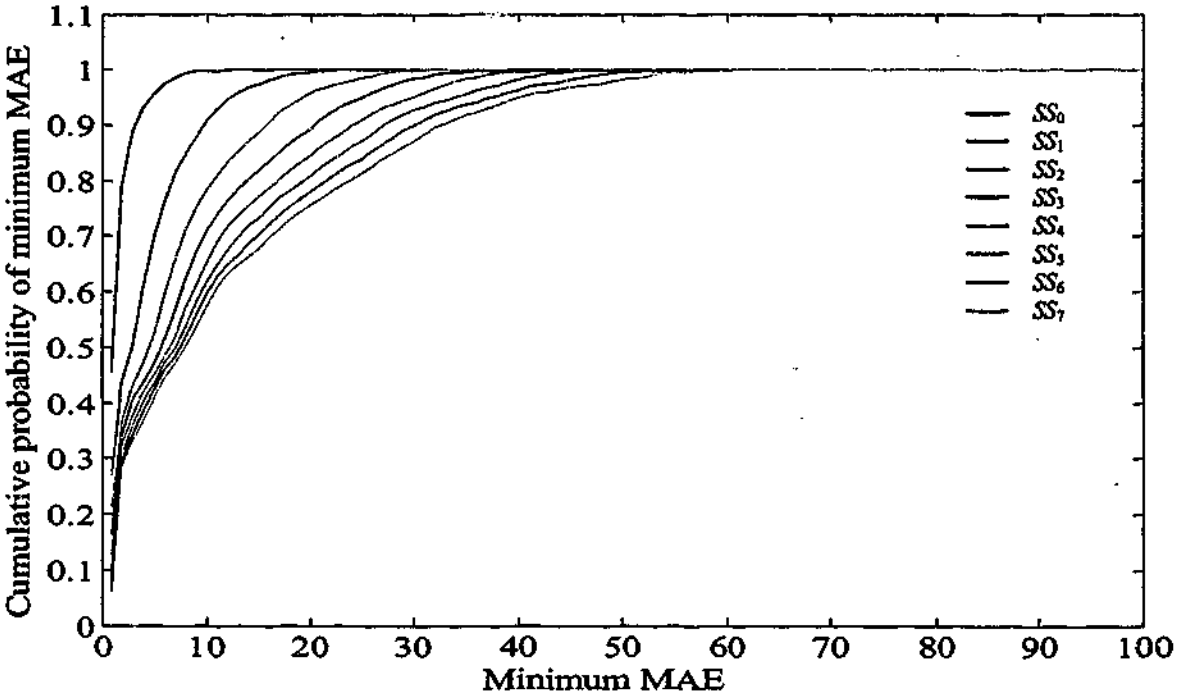
(a) *Football* sequence.



(b) *Flower Garden* sequence.



(c) *Salesman* sequence.



(d) *Miss America* sequence.

Fig. 3.3: The cumulative probability of minimum MAE for different search squares on the first 80 frames of four standard video sequences.

A comparison of the general trends of the cumulative probability curves for the same search square, across all four video sequences in Fig. 3.3, indicates that the distortion level is higher in the *Flower Garden* and *Football* sequences, which exhibit relatively high motion compared

with the *Salesman* and *Miss America* sequences, where instead, low motion is involved. As *Miss America* has almost no motion, the cumulative probability of a minimum MAE = 0 is very high, whereas the cumulative probability of a minimum MAE = 0 in *Football* sequence is almost zero.

Fig. 3.3 therefore, reveals the following : -

- The cumulative probability of having a particular minimum MAE decrease as the motion vector length increases.
- The minimum MAE, in which the cumulative probability first reaches the value 1, increases as the motion vector length increases.

Both these findings reveal that the probability of terminating the FS algorithm at a higher MAE value increases with the length of the motion vector.

Based on these observations, the key finding is that the distortion of an object in a video frame increases with its velocity, as well as with the zoom and pan factors of the camera. As the length of the motion vector grows, so does the distortion error. It can be therefore concluded that locating a block with a minimum prediction error but with a motion vector of high length, is not only ineffectual in the prevailing distorted search space, but may lead to *false* motion vectors being erroneously selected.

This leads, elegantly, to a potential solution for the challenge raised in Section 3.2 of developing a non-directional search algorithm which is comparable in terms of speed to the other fast directional search algorithms, but also provides improved image quality. A non-directional search can be effectively made faster if the search is not directed by the sole desire of reaching the global minima unconditionally. As well, terminating a relatively high BDM once the current minimum BDM exceeds a threshold value should also be considered, where this threshold also increase as the search moves away from the centre.

By making the thresholding function distance-dependent, the search can be controlled by a user-defined parameter, so that the new search algorithm can be transformed from the qualitatively best, but slow, FS algorithm to extremely fast searches which trade-off quality for search speed. This search technique thus provides an effective control mechanism for performance scalability and QoS by trading between predicted image quality and processing time in motion estimation.

Being non-directional and incorporating a relationship between distortion, object velocity and camera factors, this new *Distance-dependent Thresholding Search* (DTS) algorithm has the potential of capturing more *true* object motion vectors compared with other directional fast search algorithms and non-directional FS algorithm. This issue will be explored further in Section 3.6.

The DTS algorithm is now presented in detail, and includes an analysis of the influence of using linear and non-linear thresholding functions.

3.4 The Distance-dependent Thresholding Search (DTS) Algorithm

Definition 3.4: $MAE_{(cx,cy)}(u,v)$ denotes the Mean Absolute Error per pixel of the macroblock centred at pixel $p_{cx,cy}$ in the current frame with respect to the block centred at pixel $p_{cx+u,cy+v}$ in the reference frame.

3.4.1 The Formal DTS Algorithm

Like all block-base motion estimation search techniques, the DTS algorithm starts at the centre of the search space. The search then progresses outwards by using search squares, SS_τ in order while monitoring the current minimum MAE. A parametric thresholding function, $Threshold(\tau,C)$, is used to determine the various thresholds to be used in the search involving each SS_τ where the parameter, C, is set at the start of each search and acts as a control parameter, as alluded to at the end of Section 3.3. After searching each SS_τ the current minimum MAE is compared against the threshold value of that specific search square and the search is terminated if this MAE value is not higher than the threshold value. The DTS algorithm is formally presented in Fig. 3.4.

- **Parameter** C
- **Precondition:** Pixel $p_{cx,cy}$ is the centre of the search space with maximum displacement d.
- **Initialisation:**
  $MAE_{\min} = MAE_{(cx,cy)}(0,0)$ (Definition 3.4)
  $MV = (0,0)$
- **Main Algorithm:**
  If $MAE_{\min} > 0$ Then
    For $\tau = 1,2,...,d$
      For each checking point $p_{x,y}$ in SS_τ
        $\varepsilon = MAE_{(cx,cy)}(x-cx, y-cy)$
        If $\varepsilon < MAE_{\min}$ Then
          $MAE_{\min} = \varepsilon$
          $MV = (x-cx, y-cy)$
      If $MAE_{\min} \le Threshold(\tau,C)$ Then STOP
- **Postcondition:** MV contains the motion vector and $MAE_{\min}$ contains the distortion error of the respective block.

Fig. 3.4: The DTS algorithm.

3.4.2 Characteristics of the Thresholding Function

To make sure that the DTS algorithm can be transformed to an exhaustive FS algorithm, the threshold value for SS_0 is always assumed to be 0. As the maximum MAE value using a b-bit gray level intensity is 2^b-1, threshold values for all other search squares can, at most, be 2^b-1. However, to ensure the algorithm includes the entire search space, all but the outermost threshold value must be less than 2^b-1. Moreover, to make the thresholding function distance-dependent, the function must monotonically increase.

The DTS algorithm, therefore, assumes the following general properties of the thresholding function:

$$Threshold(0,C) = 0 \tag{3.3}$$

$$Threshold(1,C) \le Threshold(2,C) \le ... \le Threshold(d,C) \tag{3.4}$$

$$Threshold(\tau,C) < 2^b - 1, \text{ for all } \tau = 1,2,...,d-1 \tag{3.5}$$

$$Threshold(d,C) \le 2^b - 1 \tag{3.6}$$

Parameter C plays a significant role in the DTS algorithm by allowing users to define different sets of monotonically increasing threshold values based on specific values of C. Obviously, a set of larger threshold values terminates a search earlier than a set of smaller values. C, therefore, provides a control mechanism to allow trading-off between the computational complexity in terms of search points and prediction image quality.

The monotonic increasing function requirement means the DTS algorithm could use a linear, exponential, or any other complex analytic function to control C. In the next two sections, linear and exponential thresholding functions within the DTS algorithm will be explored.

3.4.2.1 Linear Thresholding (LT) Function

A linear thresholding function can be defined as follows:

$$Threshold(\tau, C_L) = C_L \times \tau, \text{ for all } \tau = 0,1,...,d \tag{3.7}$$

The above notation uses subscript L for the parameter, C, to specify linear thresholding, while in the next section, subscript E is used to indicate exponential thresholding.

It can be verified that the above definition satisfies conditions (3.3) and (3.4) if $C_L \ge 0$. In order to satisfy the remaining two properties in (3.5) and (3.6), $C_L \times d \le 2^b - 1$, i.e., $C_L \le \frac{2^b - 1}{d}$. So, the range of values for parameter C_L is:

$$0 \le C_L \le \frac{2^b - 1}{d} \tag{3.8}$$

Sets of threshold values corresponding to C_L = 0, 10, 20, and 36 are presented in Fig. 3.5, to illustrate the different values of threshold for each search square, where 8-bit gray level intensity with a maximum search displacement $d = \pm 7$ pixels is assumed.

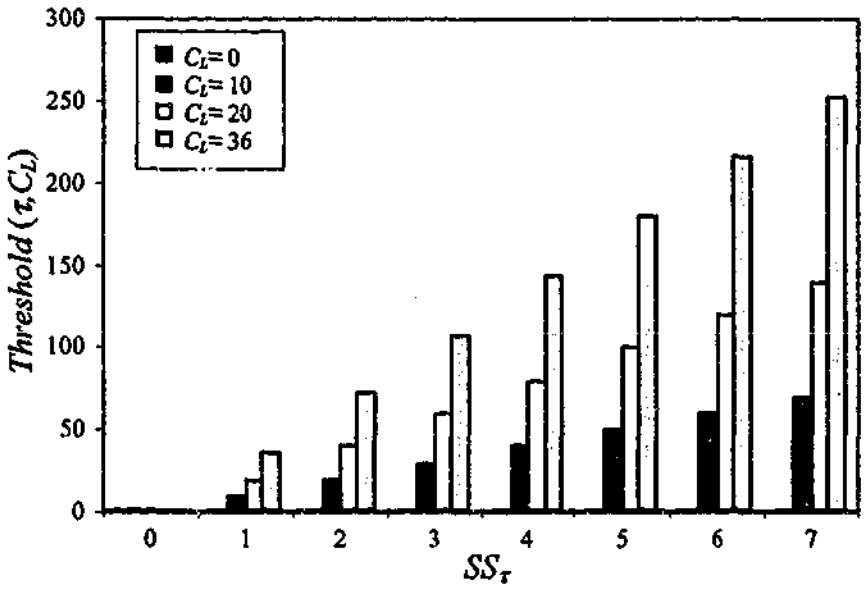


Fig. 3.5: Sets of threshold values for different values of parameter C_L.

3.4.2.2 Exponential Thresholding (ET) Function

An exponential thresholding function can be defined as follows:

$$Threshold(\tau, C_E) = \begin{cases} 0 & \text{if } \tau = 0 \\ 2^{\tau / C_E} & \text{otherwise} \end{cases} \tag{3.9}$$

The above definition satisfies conditions (3.3) and (3.4) if $C_E > 0$. In order to satisfy the remaining two properties in (3.5) and (3.6), $2^{d/C_E} \le 2^b - 1$, i.e., $C_E \ge \frac{2^b - 1}{\log_2(2^b - 1)}$. So, the range of values for parameter C_E is:

$$\frac{d}{\log_2(2^b - 1)} \le C_E < \infty \tag{3.10}$$

Sets of threshold values corresponding to C_E = 0.88, 1, 3, and 7 are presented in Fig. 3.6, to illustrate the different values of the threshold function for each search square, where 8-bit gray level intensity with a maximum search displacement ±7 pixels is assumed.

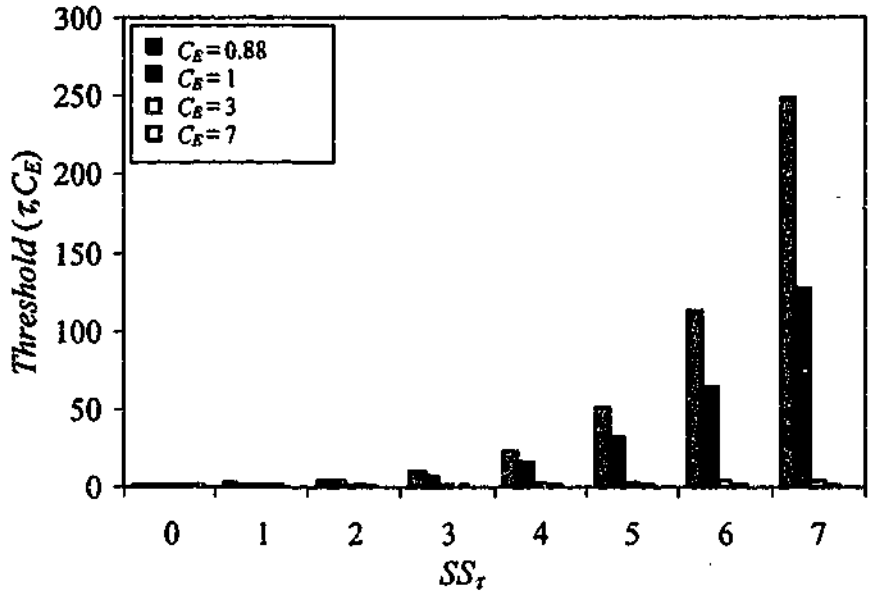


Fig. 3.6: Sets of threshold values for different values of parameter C_E.

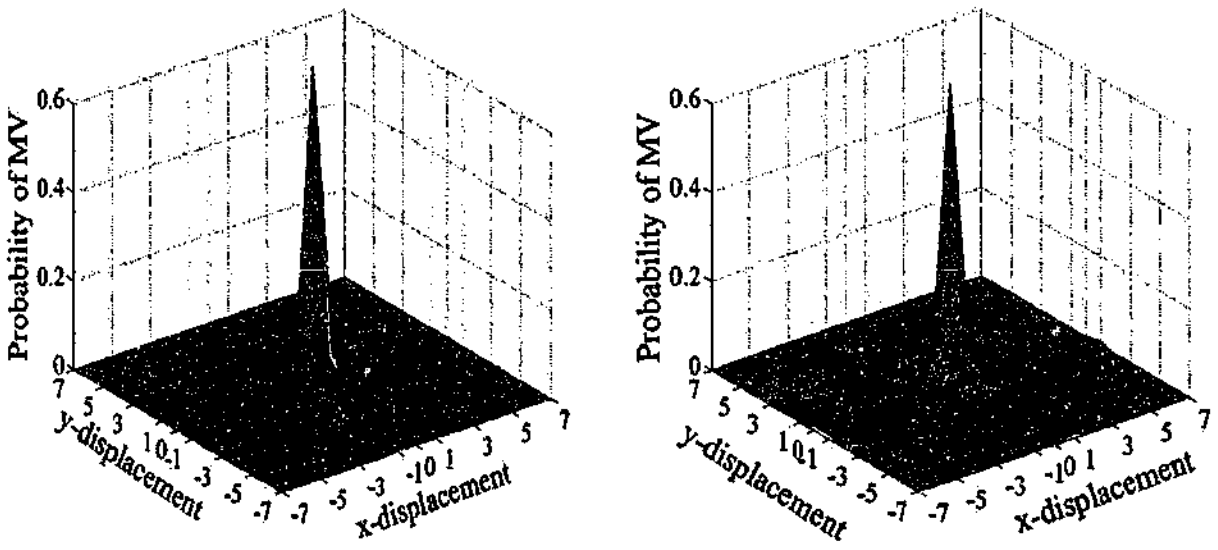
3.4.2.3 Selecting the Thresholding Control Parameter

The values of C_L and C_E have a significant influence on the level of computation, the quality of the motion vector, and the prediction error. In the previous section, the upper and lower limits on the values of C_L and C_E were defined. To clarify the nomenclature, the DTS algorithm using the linear thresholding parameter C_L and exponential thresholding parameter C_E are denoted as LT(C_L) and ET(C_E) respectively.

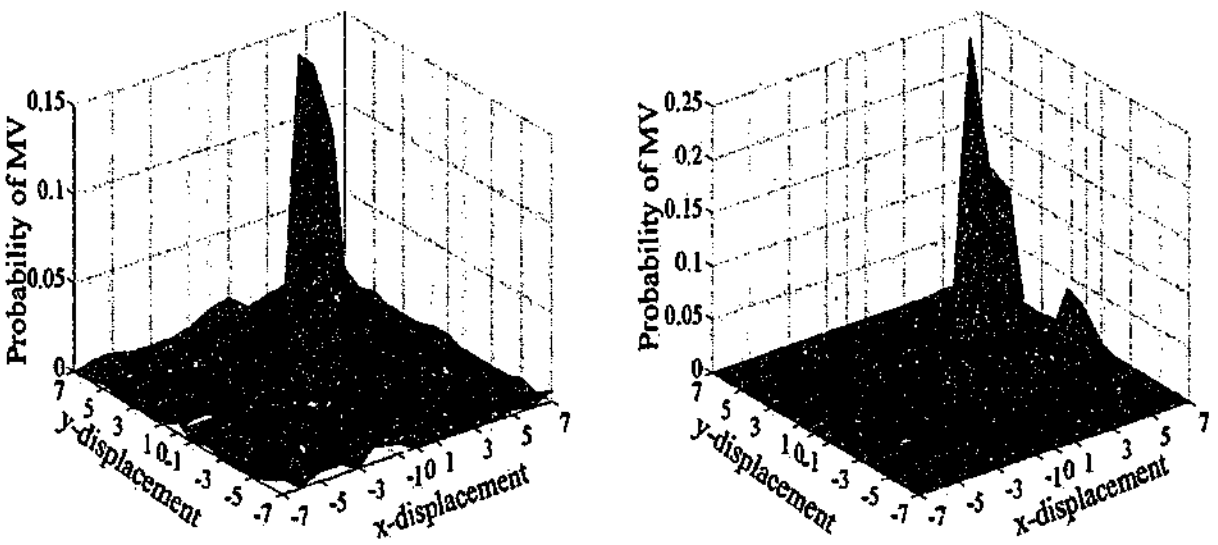
The choice of C_L involves a trade-off between the quality of the motion estimation and the computational complexity. When LT(0) in (3.7), the search terminating threshold value of any search square (*SS*) is zero as shown in Fig. 3.5. In this case, the DTS algorithm translates into the exhaustive FS algorithm as there is no threshold to terminate the search until all possible locations in the search space have been visited. Conversely, when C_L is very high in (3.7), say for example LT(36), which is close to the maximum limit of C_L (3.8) for an 8-bit gray level image and $d = 7$, the threshold value of each search square is shown in Fig. 3.5. In this case, the DTS algorithm will be as fast as the probability of getting the minimum BDM within the search terminating threshold limit is high, especially around the search centre. In case of low motion video sequences, such as *Salesman* and *Miss America,* where MV distribution is centre-biased, as shown in Fig. 3.7(a) and (b) respectively, a high C_L performs well with low computation. Conversely, for high motion video sequences, such as *Football* and *Flower Garden*, where the motion vector distribution is not centre-biased, as shown in Fig. 3.7(c) and (d), a high C_L may stop the search with an inaccurate motion vector and high prediction error.

The effect of the exponential control parameter will now be considered. When ET(0.88) with a maximum displacement, $d = 7$, the search terminating threshold value for different search

squares gradually increases (3.9), as shown in Fig. 3.6. In this case, the DTS algorithm will be faster than the FS algorithm. If the value of C_E is higher, for example ET(7), the threshold value for all search squares, except the centre, becomes a maximum of 2 as shown in Fig. 3.6. In this case, the likelihood of a BDM being within this small range is low for most types of video sequence, especially if it is non-stationary. Therefore, a high value of C_E means the DTS algorithm moves towards the FS algorithm.



(a) *Salesman.* (b) *Miss America.*



(c) *Football.* (d) *Flower Garden.*

Fig. 3.7: Motion vector distributions of some high and low motion video sequences where MVs have been calculated by the FS algorithm with a maximum displacement, $d = 7$.

From the above discussion, it can be seen that the flexibility in controlling the search speed is very limited, although different values of C_E, for example, in the range 0.88 to 7, enable the search speed and predicted image quality to be controlled. The experimental results in Table 3.1

make clear that while the search speed of the DTS algorithm using the ET function is comparable to the NTSS or TSS algorithm for the low motion *Miss America* video sequence where ET(1), the search speed is not comparable for other values of C_E. Conversely, for the high motion *Flower Garden* sequence, the maximum search speed of the DTS algorithm with the ET function, where ET(1), is on average, 70.16 search points per motion vector, whereas the corresponding numbers for NTSS and TSS are 21.63 and 23.22 respectively. This clearly indicates that the search speed of the DTS algorithm with the ET function does not provide comparable performance to other fast algorithms, which leads to the conclusion that the ET function does not afford sufficient flexibility in controlling the computational complexity in real-time video coding applications.

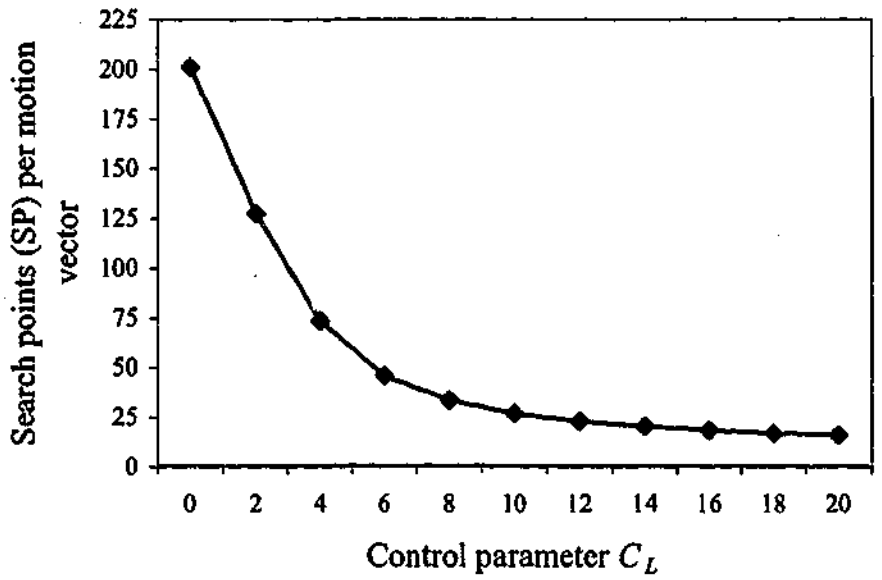


Fig. 3.8: Flexible search speed in the DTS algorithm with an LT function using different values of C_L for the *Flower Garden* sequence.

Table 3.1 also includes the search speed of the DTS algorithm using the LT function, and shows that it is comparable to the FS, NTSS, and TSS algorithms for both the *Miss America* and *Flower Garden* video sequences, using various threshold settings. The search speeds in terms of average number of search points (SP) per motion vector obtained for different values of C_L in Table 3.1 have been plotted in Fig. 3.8, demonstrating that the DTS algorithm with LT function provides the flexibility to control the search speed from the FS algorithm across the range of different speeds, and is in principle, even faster than existing fast algorithms. This leads to the conclusion that the LT function provides full flexibility in controlling the computational complexity for real-time video coding applications, and since complexity management is crucial for such applications, only linear thresholding will be considered in the subsequent chapters.

Finally, while linear thresholding means that different levels of QoS can be achieved by trading off between predicted image quality and computational complexity, in terms of search

points, selecting the best value of C_L still remains a challenging problem. C_L has to be manually set and the DTS algorithm is unable to automatically adjust C_L as the motion content in the video sequence changes over time.

In Chapter 5, the DTS algorithm will be modified to provide a mechanism for fully adapting appropriate C_L parameters automatically as the motion content changes so that a QoS target, either in terms of search points, or predicted image quality, can be maintained.

3.5 Performance Analysis in Video Coding Applications

All experiments were performed on a Pentium III 600 MHz computer running the Windows 2000 operating system and using MATLAB 6. The FS, TSS, NTSS, and DTS algorithms were implemented to compute the block-based inter-frame motion vectors from the luminance (Y-component) signal of a number of standard and non-standard test video sequences (Appendix B).

All sequences were uniformly quantised to an 8-bit gray level intensity. The block size dimensions $[N,N]$ and maximum displacement, d, were considered as [16,16] and ±7 respectively throughout the experiments. A maximum of $(2d+1)^2 = 225$ checking points were used, and the MAE distortion measure was used as the matching criterion. In all cases, the centre of the search window was examined first, and if the MAE = 0, then the search was immediately terminated without checking any further points.

The test results for the low motion video conferencing sequence, *Miss America*, in QCIF format (176×144 pixels) and the high motion *Flower Garden* sequence in SIF format (352×240 pixels) are included, and the results for the high motion sequences, *Football* and *Table Tennis* and low motion video conferencing sequence, *Salesman*, are included in the supplementary results section in Appendix C. All test results are shown for integer-pel accuracy whereas in Section 3.5.4, the performance of the DTS algorithm using half-pel accuracy motion estimation is included for comparative purposes.

3.5.1 Quantitative Evaluation

To quantitatively evaluate the performance of the DTS algorithm in video coding applications, the following three measures were used: -

i. The average MSE (2.2) between the predicted and corresponding original frames.
ii. The average *peak signal-to-noise ratio* (PSNR) between the predicted and corresponding original frames.
iii. The average number of *search points* (SP) per motion vector as the measure of computational complexity.

Average MSE Performance

The performance of the DTS algorithm using both the LT and ET functions is compared with the FS, TSS, and NTSS, in terms of the average MSE between the predicted and original frames in Table 3.1 for the *Miss America* and *Flower Garden* sequences. A wide range of different values for C_L and C_E based on (3.8) and (3.10), were tested to analyse the performance of the DTS algorithm, while only a subset of the results are presented in Table 3.1.

Block-matching algorithms	*Miss America* sequence (1-80 frames)			*Flower Garden* sequence (1-80 frames)		
	MSE	PSNR [dB]	SP	MSE	PSNR [dB]	SP
FS/LT(0)	5.386	40.818	168.21	266.86	23.90	201.24
TSS	5.511	40.719	19.67	322.88	23.12	23.22
NTSS	5.398	40.808	15.14	279.37	23.76	21.63
LT(2)	5.395	40.811	14.60	270.97	23.89	127.43
LT(4)	5.399	40.805	9.01	275.80	23.82	73.48
LT(6)	5.408	40.800	7.90	283.98	23.69	45.93
LT(8)	5.408	40.800	7.50	293.61	23.55	33.59
LT(10)	5.408	40.800	7.30	305.63	23.37	27.00
LT(12)	5.408	40.800	7.25	318.74	23.19	23.06
LT(14)	5.408	40.800	7.23	334.18	22.97	20.47
LT(16)	5.408	40.800	7.22	353.03	22.72	18.61
LT(18)	5.408	40.800	7.21	373.68	22.48	17.21
LT(20)	5.408	40.800	7.21	395.23	22.23	16.11
ET(1)	5.400	40.807	12.81	275.96	23.82	70.16
ET(2)	5.398	40.808	26.61	270.49	23.90	155.02
ET(4)	5.397	40.809	48.20	270.47	23.90	175.65
ET(8)	5.397	40.809	64.31	270.46	23.90	179.18

Table 3.1: Average MSE and PSNR per pixel, and SP per motion vector comparison for the *Miss America* and *Flower Garden* sequences (1-80 frames).

It can be observed that for the *Miss America* sequence, with LT(4), the speed improvement factor was almost 20 times faster whereas the average MSE was very similar (within 0.24%) to the optimal average MSE of the FS algorithm. The prediction error performance of the DTS algorithm was aslo better than TSS, and comparable with NTSS when the search speed was either similar, or higher. For example, for the DTS algorithm with LT(2), the average MSE was

slightly better than that of NTSS, and 2% better than the TSS algorithm, while the search speed was 26% and 3% faster than the TSS and NTSS algorithms respectively.

For the *Flower Garden* sequence, the speed-up factor of the DTS algorithm with LT(6) was almost 5 times that of the FS algorithm, whereas the error performance was within 5% of the optimal average MSE of the FS algorithm. Table 3.1 also illustrates that the performance of the DTS algorithm was slightly better than that of the TSS algorithm when considering LT(12), but was not so satisfactory when compared with NTSS algorithm. The reasons for these results will be discussed in Section 3.5.2.

In Figs. 3.9 and 3.10, the MSE performance of the DTS algorithm against the FS, TSS, and NTSS algorithms for each frame of both video sequences is plotted. For the sake of clarity, only those threshold control values for the DTS algorithm that used search points comparable to the TSS and NTSS algorithms are shown. Fig. 3.9 shows that the error performance for each algorithm except TSS was very similar for the whole (*Miss America*) video sequence. Fig. 3.10 illustrates that the error performance of the DTS algorithm for most of the frames was not so satisfactory for the high motion video sequence, *Flower Garden*. This limitation will be more fully discussed in Section 3.5.2.

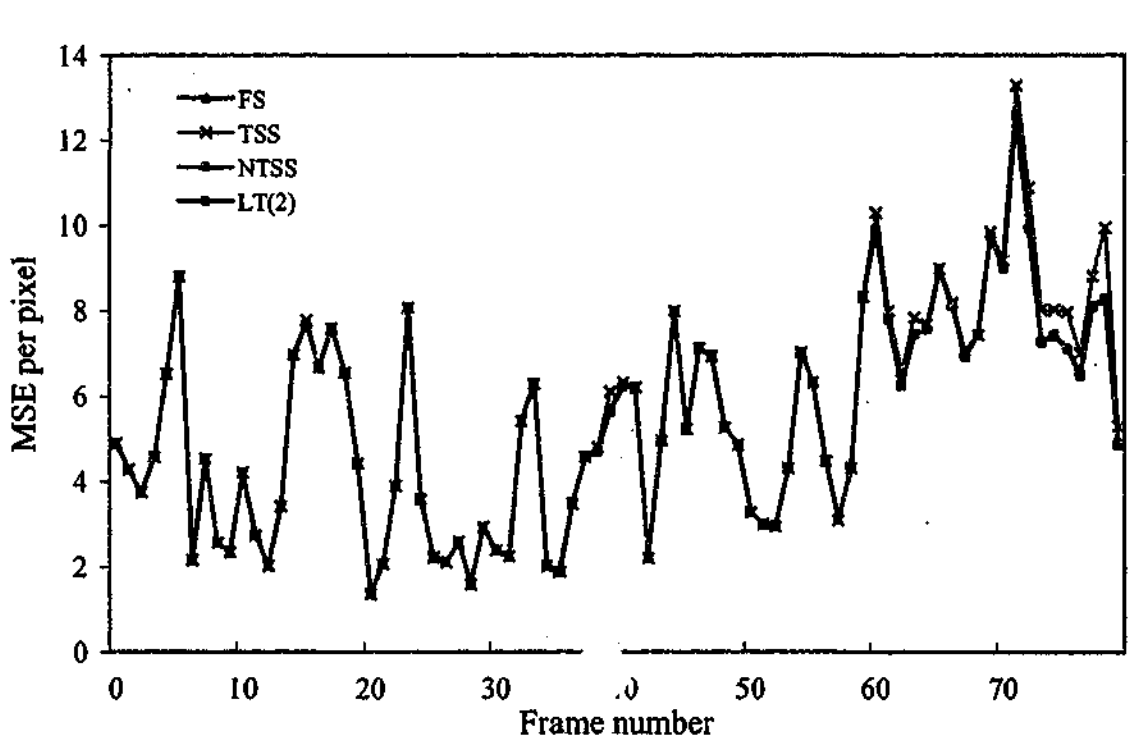


Fig. 3.9: Average MSE comparison for different BMAs with the *Miss America* video sequence.

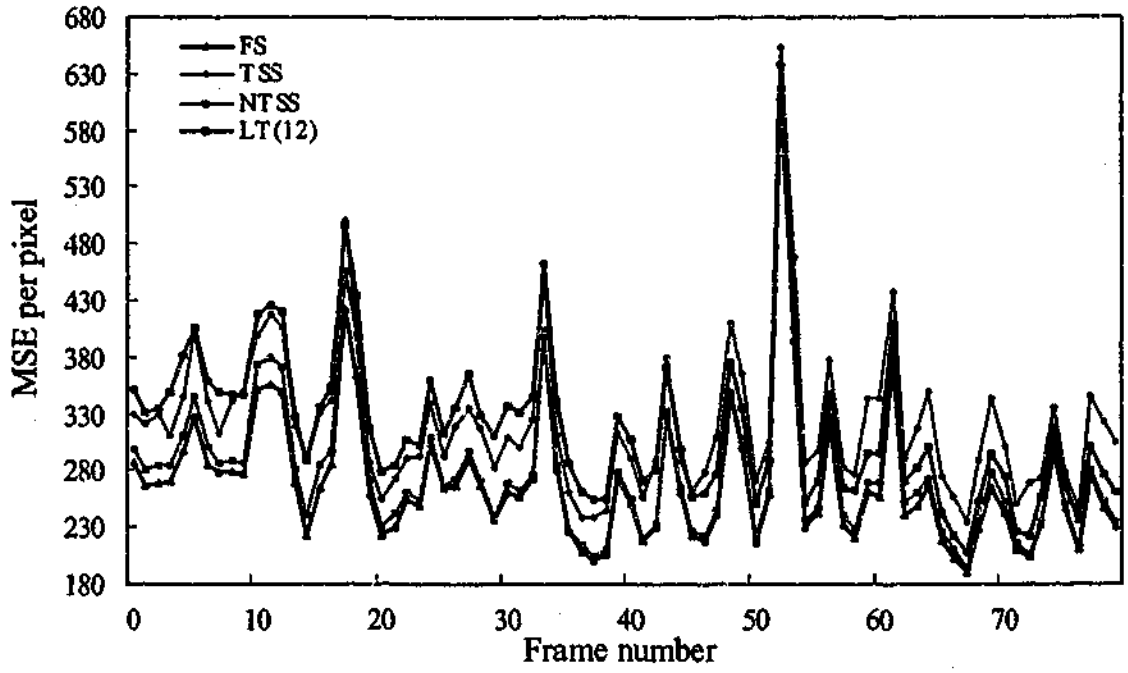


Fig. 3.10: Average MSE comparison for different BMAs with the *Flower Garden* sequence.

Peak Signal-to-Noise Ratio (PSNR) Performance

All PSNR values were evaluated from: -

$$PSNR = 10\log_{10}\frac{(2^b-1)^2}{MSE} \tag{3.11}$$

The performance comparison of the FS, TSS, NTSS, and the DTS algorithms using the LT and ET functions in terms of the average PSNR between the predicted and the original image, is given in Table 3.1. The improvement in the PSNR value for the DTS algorithm with LT(2) was 0.088dB compared to the TSS algorithm for the *Miss America* sequence, while the corresponding search speed was 26%. For the *Flower Garden* sequence, the DTS algorithm with LT(12) gained 0.07dB PSNR improvement compared to the TSS algorithm, with the search speed being similar. The DTS algorithm with LT(2) gained a negligible 0.007 dB PSNR for the *Miss America* sequence while the search speed was 3% faster than that of the NTSS algorithm. Table 3.1 also shows that the PSNR performance of the DTS algorithm compared to the NTSS was not as satisfactory for the *Flower Garden* sequence using a similar search speed.

Finally, the plot in Figs. 3.11 and 3.12 illustrate the overall PSNR performance of the DTS algorithm against the FS, TSS, and NTSS algorithms. Again, only that LT function for the DTS algorithm that has similar search speeds compared to the TSS and NTSS algorithms, have been included.

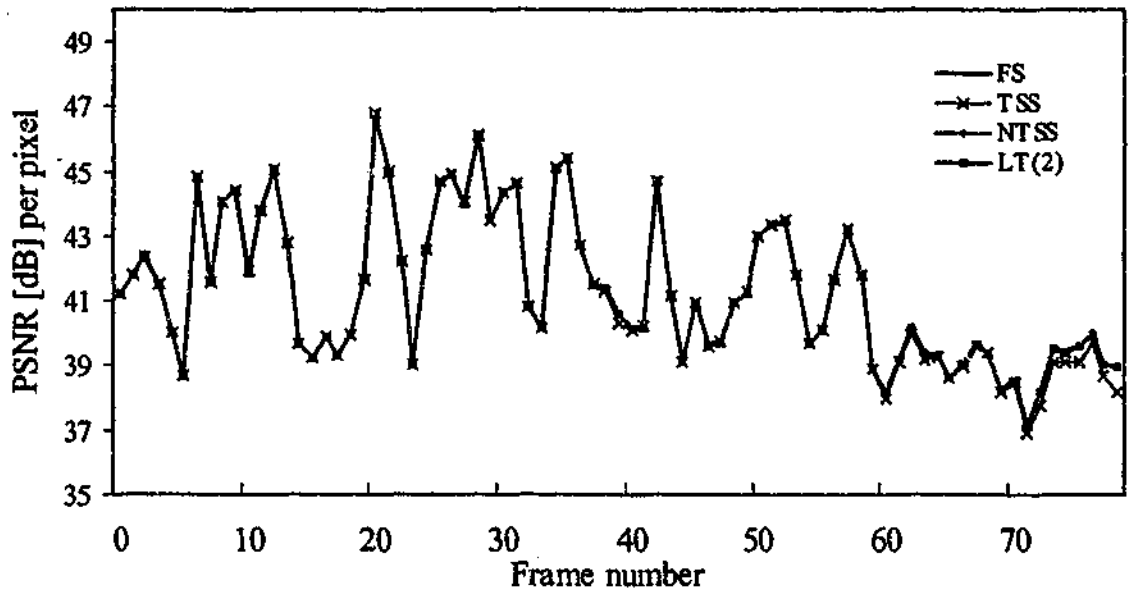


Fig. 3.11: Average PSNR comparison for different BMAs with the *Miss America* video sequence.

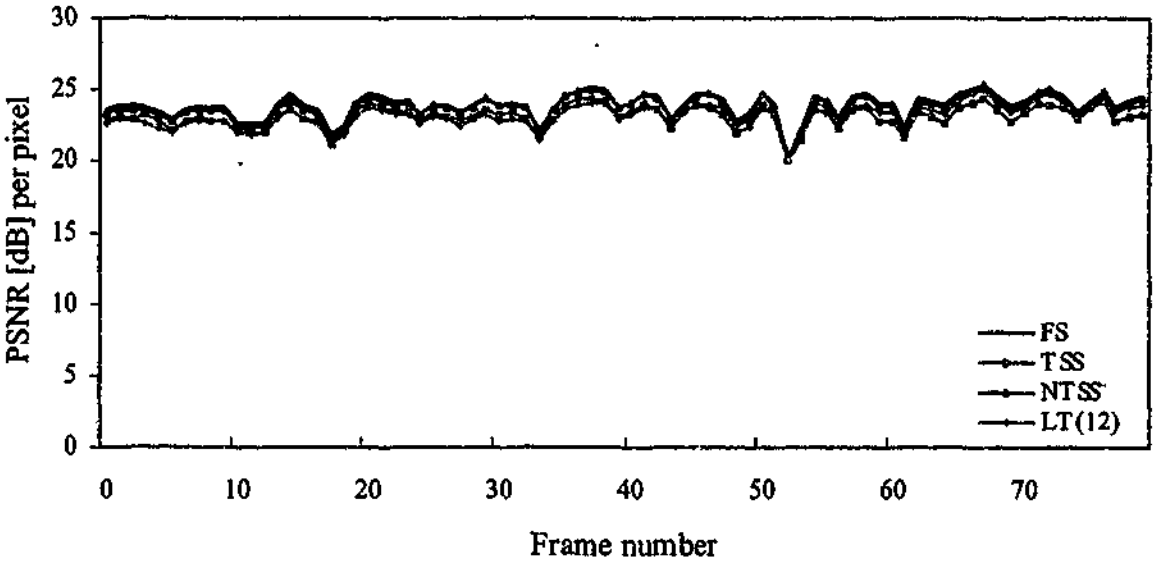


Fig. 3.12: Average PSNR comparison for different BMAs with the *Flower Garden* video sequence.

Search Speed Performance

For all values beyond LT(6) in Table 3.1, for the *Miss America* sequence, while the search speed increased, the quality performance in terms of average MSE or PSNR remained constant, indicating that although the speed-up factor was high compared to TSS, it still provided better prediction quality. This also indicates that the search speed of the DTS algorithm with LT(4) was on average 40% faster than that of the NTSS algorithm, with both providing similar error or PSNR performance. This clearly demonstrates that for low motion video sequences, the DTS algorithm, with LT function, outperforms both the TSS and NTSS algorithms. The results also

confirm that by choosing a suitable value for the control parameter for the selected threshold function, the average number of search points required by the DTS algorithm will be considerably less, while concomitantly, having a significantly higher average PSNR or MSE. For the *Flower Garden* sequence however, the search speed of the DTS algorithm, for example in the case of LT(12), was similar to TSS but worse than NTSS for a similar error performance. In Figs. 3.13 and 3.14, the search point performance of the DTS algorithm compared with the FS, TSS, and NTSS algorithms is shown. Again, only the LT function that generated MSE values similar to those of the TSS and NTSS algorithms, has been plotted. Fig. 3.13 shows that the number of average search points per motion vector is always less compared to TSS and NTSS for any frame of the *Miss America* sequence, while Fig. 3.14 illustrates that the search speed of the DTS algorithm was not as good for the *Flower Garden* sequence. This recurring limitation, which has been identified in the DTS algorithm, will now be analysed more fully.

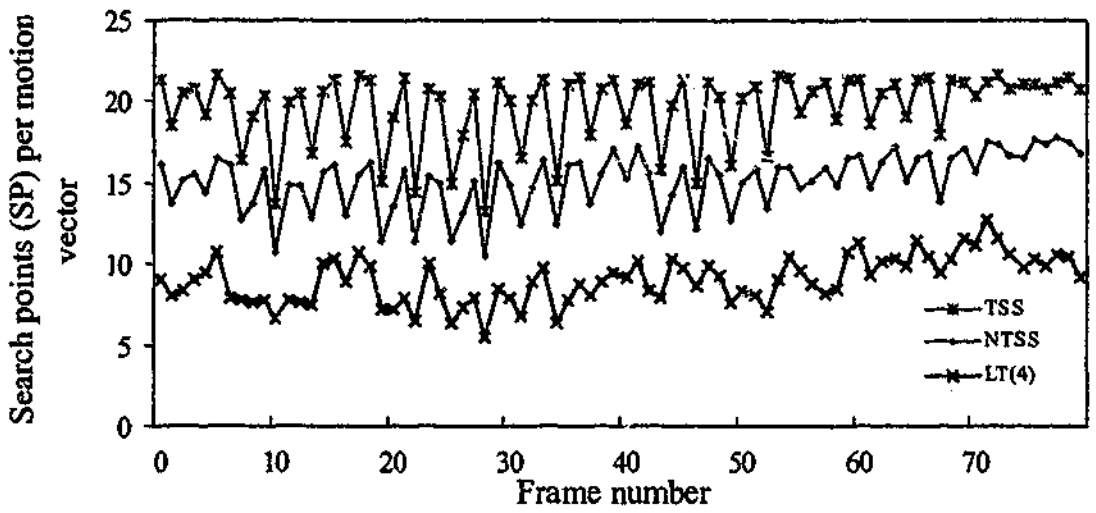


Fig. 3.13: Average search points (SP) comparison for different BMAs with the *Miss America* video sequence.

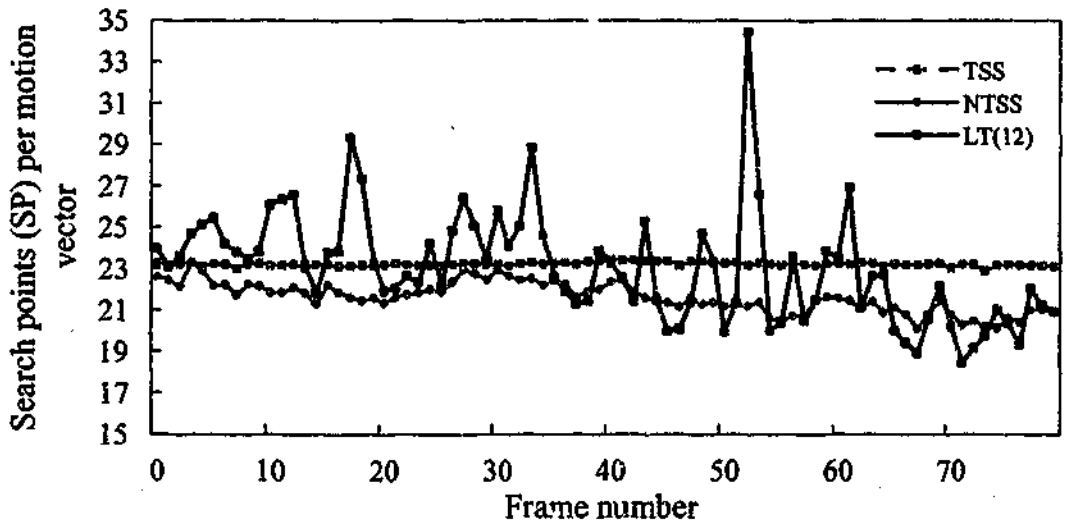


Fig. 3.14: Average search points (SP) comparison for different BMAs with the *Flower Garden* video sequence.

3.5.2 Key DTS Performance Issues

The overall conclusion from the results in Table 3.1 is that for low motion video sequences such as *Miss America* (and the *Salesman* sequence presented in the supplementary results in Appendix C), the DTS algorithm consistently performs better than either the TSS or NTSS algorithm. This is because, as was noted in Section 2.4.2, the TSS algorithm always searches 25 points irrespective of the video content. For the low motion case, the NTSS algorithm searches at least 17 points (Section 2.4.5). Li *et al.* [45], Zhu and Ma [53], Luo *et al.* [68], and Cheung *et al.* [85] show that in central-biased low motion sequences, more than 80% of blocks are stationary or quasi-stationary, and most of the motion vectors are within a 3×3 or 5×5 area around the search centre, as shown in Figs. 3.7(a) and (b). As the probability of getting a small prediction error is higher nearer the centre (Figs. 3.3(c) and (d)), the DTS algorithm can stop searching after completing the 8 neighboring search points, even when a small threshold value is involved. For this reason, the DTS algorithm always outperforms the TSS algorithm and provides similar or better performance than the NTSS algorithm for low motion video sequences.

The reverse is true however, for high motion video sequences where the number of search points for TSS is again 25, and for NTSS, between 17 to 33 based on the level of motion involved in the current block. In this case, the DTS algorithm may search up to 225 points if the distortion error is higher compared to the threshold in any search square. Figs. 3.7(c) and (d) illustrate the motion vector distributions of high motion video sequences, indicating that to capture real motion vectors, any search algorithm has to search at least a few pixels' distance from the search centre. If for example, a block moves 3 pixels from the centre, and if a threshold value in the DTS algorithm is such that it stops the search before reaching SS_3 (3.1), it may produce a higher prediction error with a faster search. Conversely to obtain the real vector, a threshold value should be selected that allows the DTS algorithm to search up to SS_3, where the total number of checking points required is 49. This produces better prediction quality in terms of MSE or PSNR with higher search points. For this reason, the DTS algorithm does not perform satisfactorily with the high motion video sequence, *Flower Garden,* compared to the NTSS algorithm. In this case, if the initial search centre of the search space could be predicted, near to the minimum BDM position for the current block, by exploiting relevant information from neighbouring blocks, the search efficiency of the DTS algorithm for high motion sequences would be improved. This issue will be addressed more fully in Chapter 4.

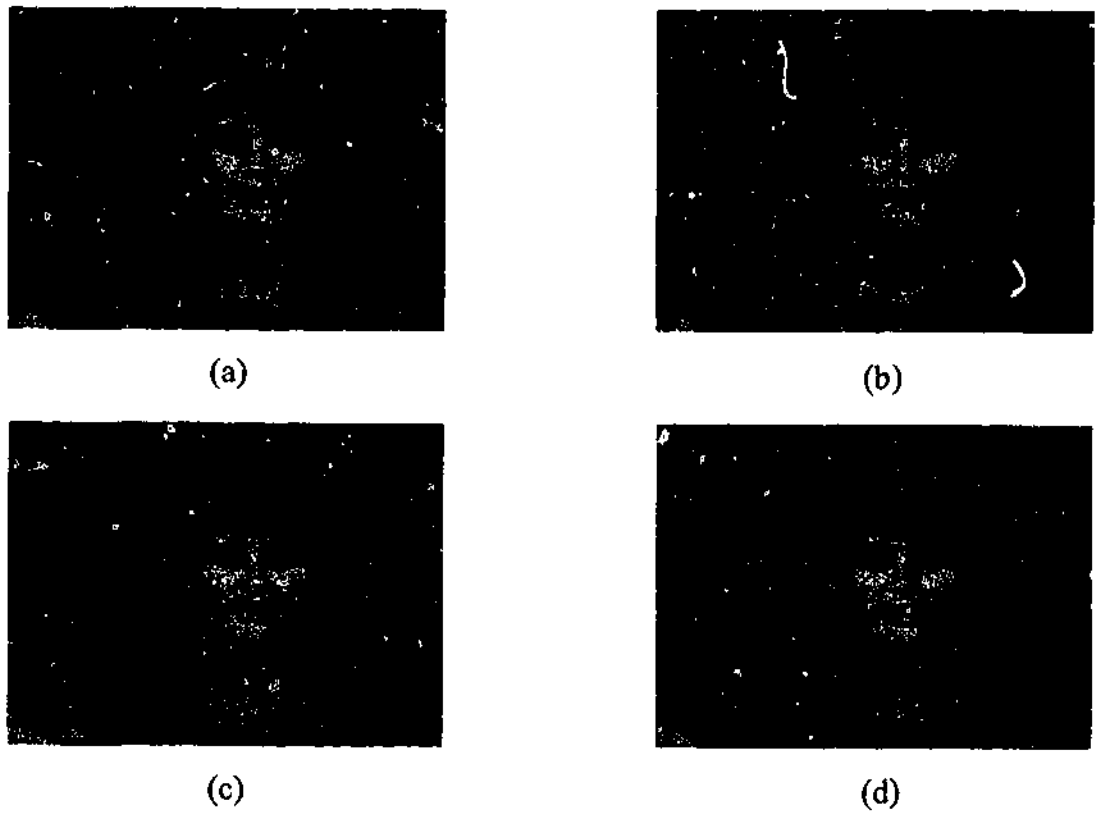
The most noticeable feature of Table 3.1 is that the different values of C_L and C_E, especially C_L, provide different levels of image prediction quality in terms of average MSE or PSNR with different levels of computational complexity. This indicates that the DTS algorithm provides

flexibility in controlling the predicted image quality, as well as computational complexity by varying the control parameter. Conversely, Table 3.1 illustrates that the existing FS, as well as the fast TSS and NTSS algorithms, have a fixed performance for both video sequences. This proves that these algorithms do not provide any flexibility in controlling quality or computational complexity based on a user's demands. These results validate one of the objectives defined in Chapter 1 that is, of developing a system which can provide QoS in terms of prediction image quality and computational complexity. This flexibility has considerable potential for exploitation in a range of applications ranging from low-bit rate video conferencing through to adaptive high-quality video coding. It is especially important for low power video coding (mobile or handheld computing platforms) and software-only video coding, which demands a more flexible approach to trade-off between predicted image quality and computational complexity. In this chapter, different values of control parameters have been set up manually. To gain the full potential of this algorithm, this control parameter in any system has to be automatically adaptable based on user requirements in terms of predicted image quality or computational complexity. Such a system will be presented in Chapter 5.

While the error performance in using the ET function is similar to that of the FS algorithm, the complexity is higher. This means the ET function for any value of C_E is not comparable with the TSS or NTSS algorithm, especially for the high motion video sequences. For this reason therefore, in the next section, only the LT function DTS algorithm will be considered for qualitative performance comparison.

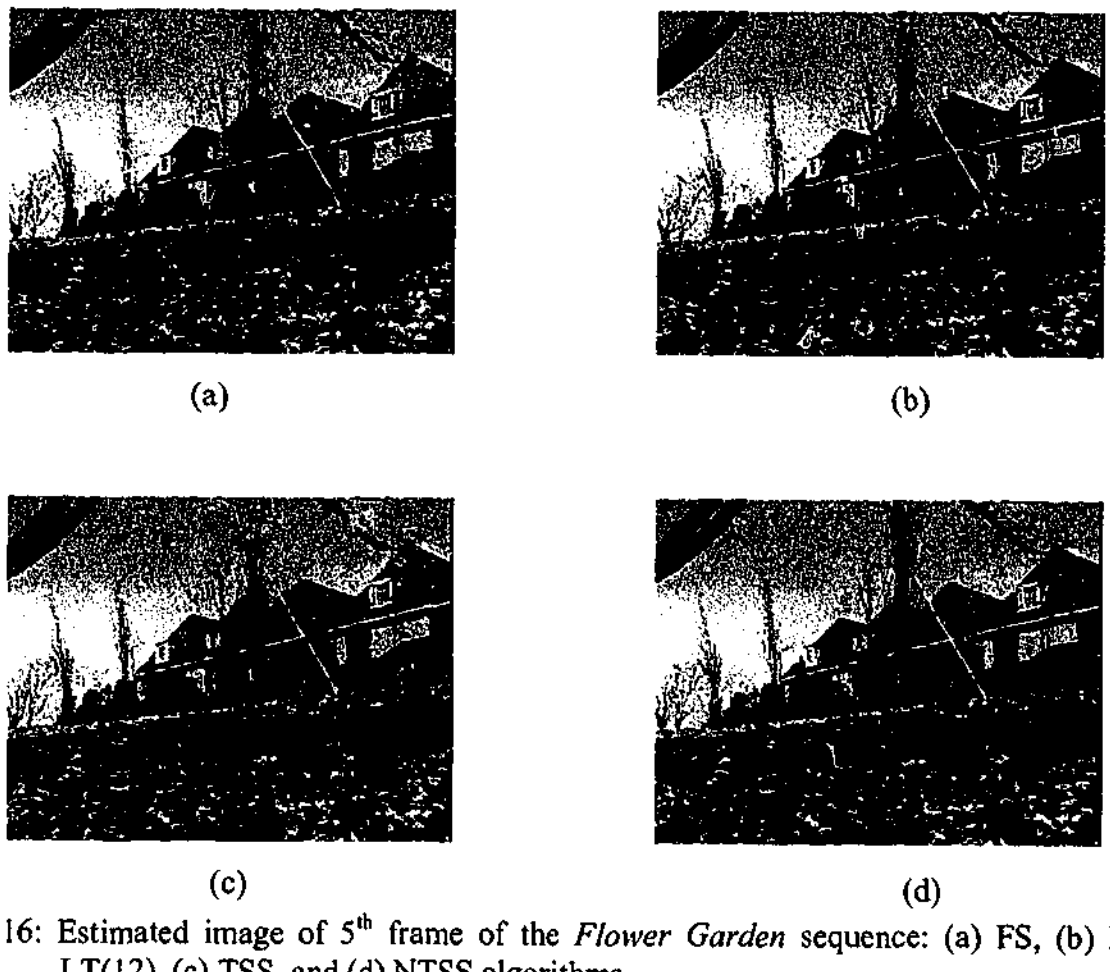
3.5.3 Qualitative Evaluation

The perceptual performance of the LT function DTS algorithm has been evaluated based on the predicted image quality for motion estimation. Figs. 3.15 and 3.16 show the estimated 76th and 5th frames respectively of the *Miss America* and *Flower Garden* sequences, for the FS, DTS, TSS, and NTSS algorithms. Fig. 3.15 shows that although the predicted image quality of the DTS algorithm is very similar to that of FS, TSS, and NTSS for the *Miss America* video sequence, the computational cost is approximately 55% and 40% less than TSS and NTSS respectively. This indicates that the DTS algorithm can predict the same image quality with a faster search speed than existing fast algorithms for low motion video sequences. Fig. 3.16 shows that the predicted image quality of the DTS algorithm is very similar to that of TSS and NTSS algorithms for the *Flower Garden* sequence with comparable computational complexity (Table 3.1).



(a) (b) (c) (d)

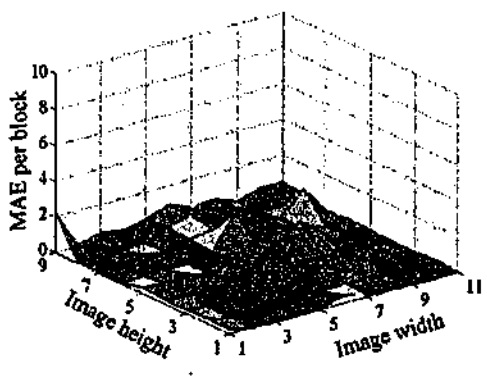
Fig. 3.15: Estimated image of 76th frame of the *Miss America* sequence: (a) FS, (b) DTS: LT(4), (c) TSS, and (d) NTSS algorithms.



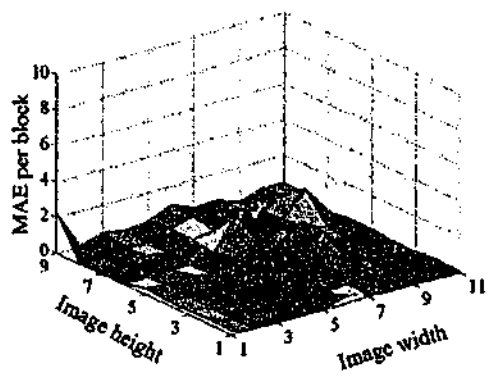
(a) (b) (c) (d)

Fig. 3.16: Estimated image of 5th frame of the *Flower Garden* sequence: (a) FS, (b) DTS: LT(12), (c) TSS, and (d) NTSS algorithms.

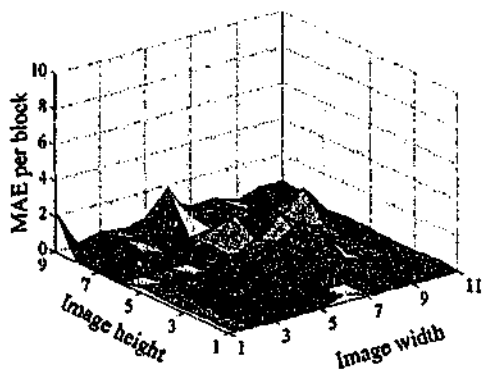
The prediction error distribution corresponding to Figs. 3.15 and 3.16 in terms of MAE per block between each original and its predicted frame, is shown in Figs. 3.17 and 3.18 respectively. As the FS is optimum in terms of error performance, Figs. 3.19 and 3.20 show the prediction error distribution of DTS, TSS, and NTSS with respect to the FS algorithm for the *Miss America* and *Flower Garden* video sequences. These figures indicate that the error performance of the DTS algorithm with LT(4) is very similar to that achieved with the FS algorithm for the *Miss America* video sequence, whereas error performance of the DTS algorithm with LT(12) is comparable to NTSS, and better than the TSS algorithm.



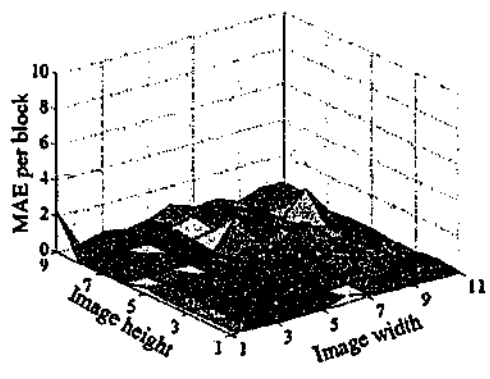
(a)



(b)



(c)



(d)

Fig. 3.17: Prediction error distribution of the 76th frame of the *Miss America* sequence: (a) FS, (b) DTS: LT (4), (c) TSS, and (d) NTSS algorithms.

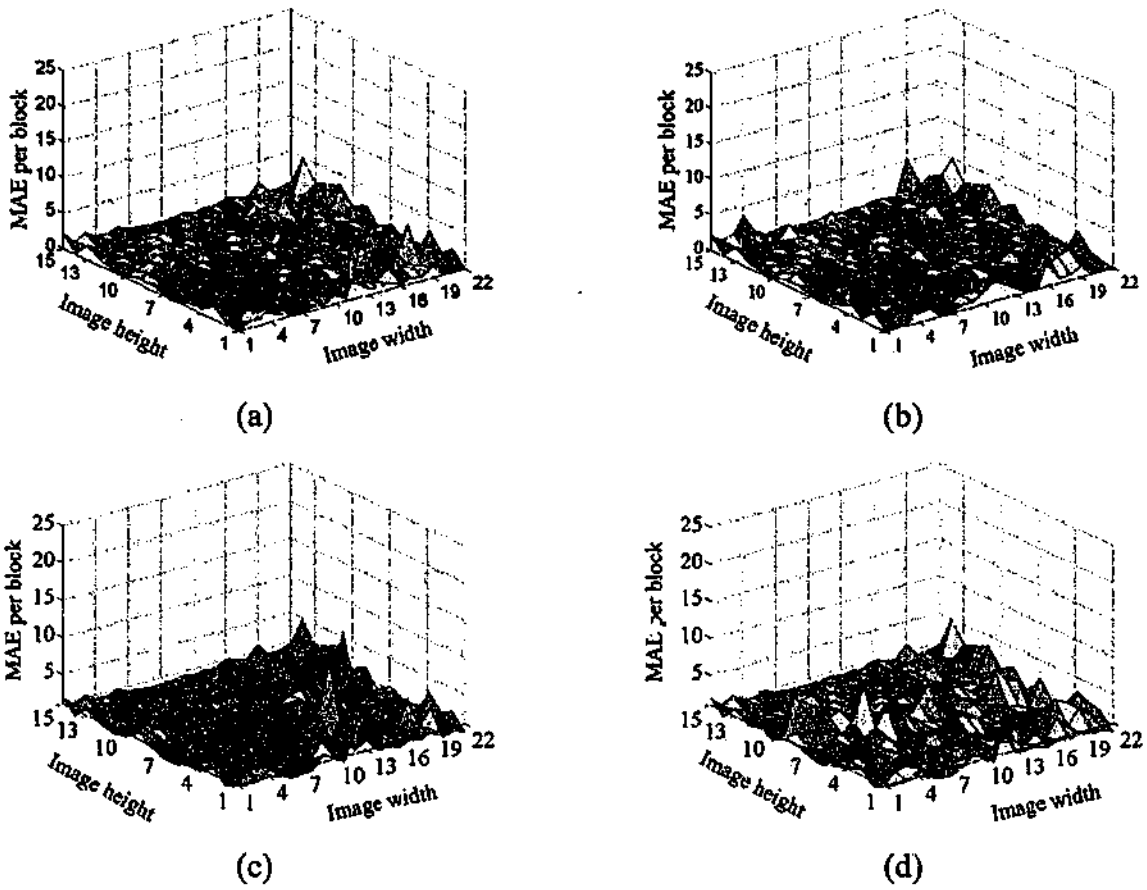


Fig. 3.18: Prediction error distribution of the 5th frame of the *Flower Garden* sequence: (a) FS, (b) DTS: LT(12), (c) TSS, and (d) NTSS algorithms.

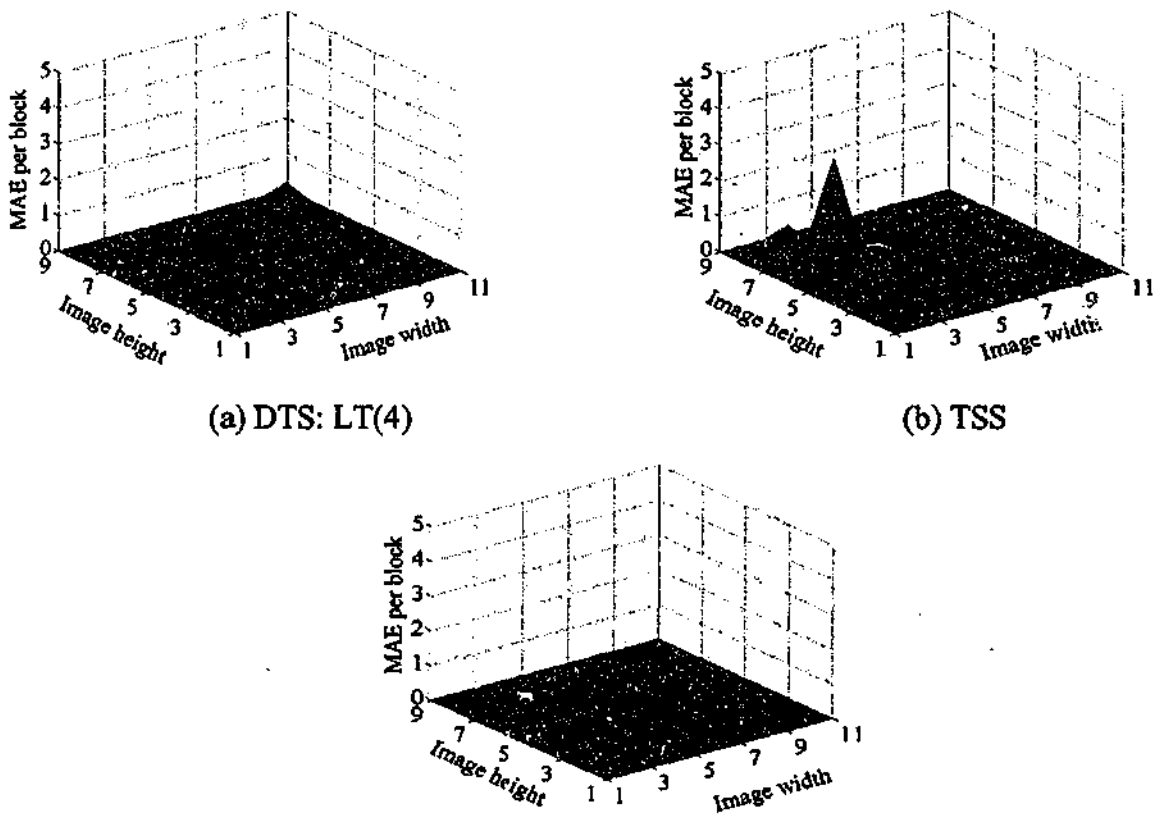
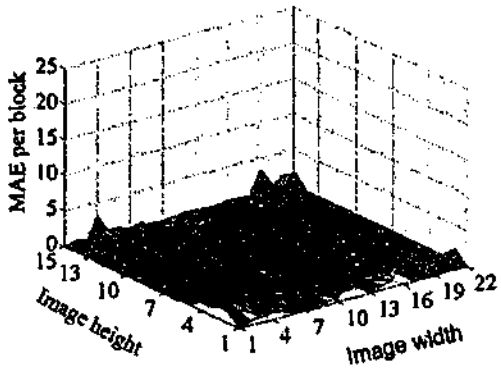
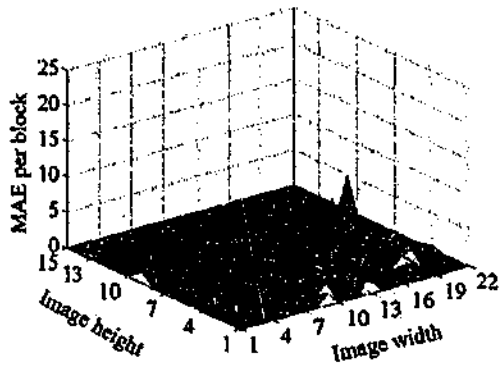


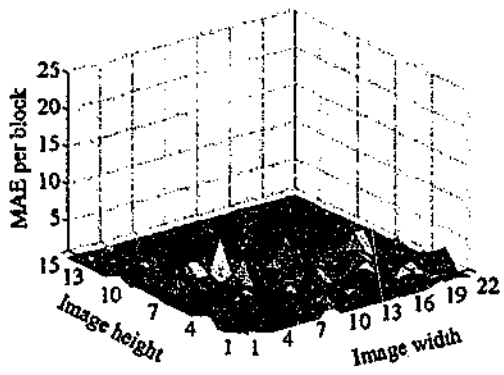
Fig. 3.19: Prediction error distribution of the 76th frame of the *Miss America* sequence with respect to that of the FS algorithm.



(a) DTS: LT(12)



(b) TSS



(c) NTSS

Fig. 3.20: Prediction error distribution of the 5th frame of the *Flower Garden* sequence with respect to that of the FS algorithm.

3.5.4 Performance Comparison using Half-pel Accuracy

The performance of the DTS, FS, TSS, and NTSS algorithms was also tested in terms of half-pel accuracy, and the results compared to integer-pel accuracy motion estimation. The process of half-pel accuracy was detailed in Section 2.4.10. Table 3.2 shows the average MSE and PSNR performance of the four algorithms for the *Miss America* and *Flower Garden* video sequences using half-pel accuracy. Comparing Table 3.1 with Table 3.2 shows that half-pel accuracy improved PSNR on average by 2.36 dB for *Miss America*, and 1.18 dB for the *Flower Garden* sequences. Half-pixel accuracy improves, in fact, the motion prediction accuracy since it also reduces noise by averaging and interpolation of pixels. This is balanced by, on average, the need to check an extra 8 search points for each motion vector. Although the overhead cost for half-pel accuracy motion estimation is higher, it has been recommended by many video coding standards such as the MPEG-1/2 and H.261/263 for significant improvement in image quality.

Block-matching algorithms	*Miss America* sequence (1-80 frames)		*Flower Garden* sequence (1-80 frames)	
	MSE	PSNR [dB]	MSE	PSNR [dB]
FS/ LT(0)	3.148	43.150	207.98	24.951
TSS	3.171	43.118	244.75	24.244
NTSS	3.158	43.137	214.39	24.819
LT(2)	3.157	43.138	208.12	24.943
LT(4)	3.158	43.137	210.10	24.907
LT(6)	3.158	43.137	214.58	24.815
LT(8)	3.159	43.135	219.52	24.716
LT(10)	3.159	43.135	226.06	24.589
LT(12)	3.159	43.135	231.78	24.480
LT(14)	3.159	43.135	237.25	24.379
LT(16)	3.159	43.135	243.63	24.263
LT(18)	3.159	43.135	251.26	24.130
LT(20)	3.159	43.135	259.83	23.984
ET(1)	3.158	43.137	210.21	24.904
ET(2)	3.158	43.137	207.99	24.950
ET(4)	3.158	43.137	207.98	24.951
ET(8)	3.157	43.138	207.98	24.951

Table 3.2: Average MSE and PSNR per pixel, and search points (SP) per motion vector for the *Miss America* and *Flower Garden* sequences (1-80 frames) with half-pel accuracy motion estimation.

3.5.5 Computational Complexity of the DTS Algorithm

The computational complexity of a motion estimation algorithm is usually expressed in terms of either the number of search points or operations that the algorithm requires to calculate the motion vectors. The complexity of the DTS algorithm is analysed in terms of the average number of search points considered in calculating the best matching block for each candidate block. However, since the DTS process calculates the motion vectors by considering all pixels of the current and candidate blocks, the number of operations is directly proportional to the number of search points. In Sections 3.4.2.3 and 3.5.2, it was stated that only the LT function would be considered for the DTS algorithm. Therefore, the complexity of the LT-based DTS algorithm is discussed as follows.

Consider a motion estimation system with the following parameters: frame size = $[N_h, N_v]$; macroblock size = $N \times N$; maximum motion vector displacement = $\pm d$; and temporal frequency =

f frames per second. If there are Ω number of operations required for the BDM calculation of one search point, then the FS algorithm requires a maximum $\psi = \Omega(2d+1)^2 \varsigma$ operations per second using integer-pel accuracy, where ς is the total number of macroblocks per second, $\varsigma = \left(\frac{N_h N_v}{N^2}\right) f$.

The DTS algorithm requires an extra d operations to compare the current minimum BDM with the predefined threshold of each search square, for each macrboblock, while searching the entire search space. The total number of extra operations required per second is therefore $d\varsigma$.

Hence, for the LT function DTS algorithm with control parameter $C_L = 0$ (the FS case), the number of operations required per second is:

$$\psi + d\varsigma \tag{3.12}$$

which is the upper computational bound, and thus the worst case scenario for computational complexity.

Conversely, by using a very high threshold value, when only the corresponding centre of the search space is checked, only one operation is required to compare the BDM found at the search centre with a predefined threshold for each macroblock. So, the number of extra operations required per second is ς, and the total number of operations required per second is:

$$\varsigma(\Omega+1) \tag{3.13}$$

which forms the lower bound. From (3.12) and (3.13), the computational complexity based on user-defined levels is always bounded between $\varsigma(\Omega+1)$ and $\psi + d\varsigma$ operations per second for the DTS algorithm using an LT function.

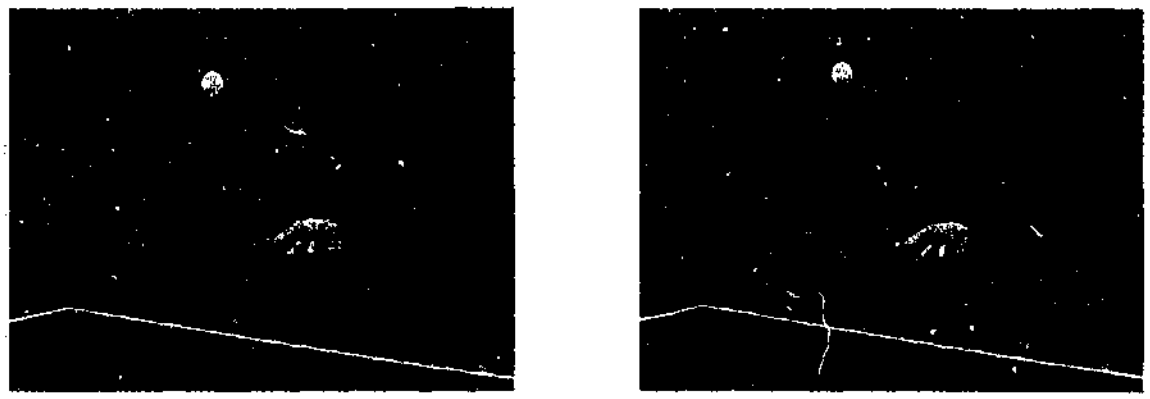
When half-pel accuracy is used for motion estimation, eight neighbouring half-pel positions (Section 2.4.10) around the current minimum point, obtained with integer-pel accuracy, are checked. In this case, the upper and lower bounds of computational complexity of the DTS algorithm increases by a further $8\varsigma\,\Omega$ operations per second.

3.6 Performance of the DTS Algorithm in *True* Object Motion Estimation

In order to represent a video object using the object motion vector, it is important to extract the *true* object motion. As alluded to in Chapter 1 and in Section 3.3, the DTS algorithm has, by virtue of its non-directional nature, the potential to capture more *true* object motion vectors than the FS or any other directional fast search algorithms. To observe this potential, the performance of the LT function in the DTS algorithm was evaluated in terms of how effectively it was able to capture *true* object motion. As the *Miss America* sequence contained very low

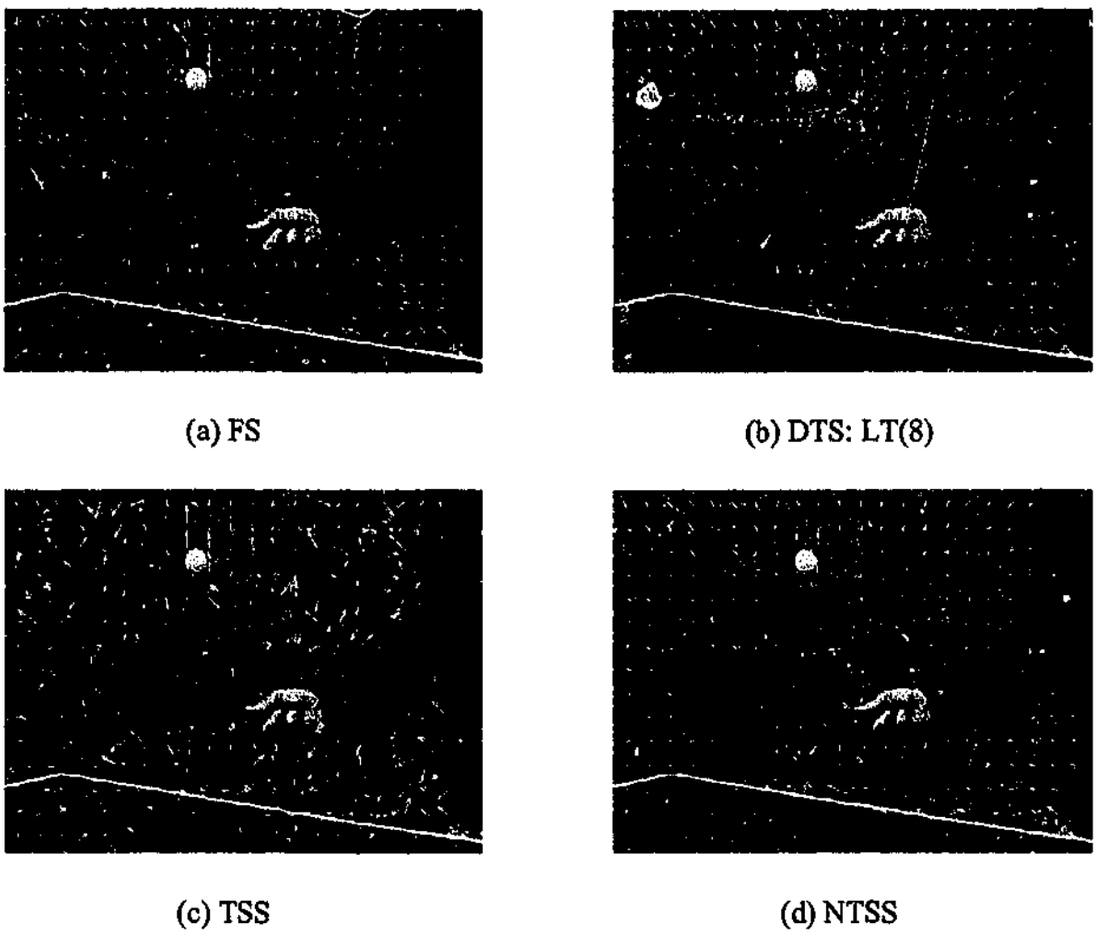
object motion and the *Flower Garden* sequence had no object motion, they were not suited to analysing the performance of the DTS algorithm for *true* object motion estimation. The *Table Tennis* sequence has, therefore, been considered instead because this sequence contains different object motions as well as camera motion, as previously indicated in Figs. 1.2 and 1.3.

Fig. 3.22 shows the motion vector needle diagrams for the captured motion vectors using FS, TSS, NTSS, and DTS with LT functions for the pair of frames #32 and #33 from the *Table Tennis* sequence shown in Fig. 3.21. To clearly show the exact direction of *true* object motion, the next frame, instead of the previous frame, has been considered as the reference frame. Apart from camera motion due to zooming, the only moving objects in the frame pair are a ball, bat, and a portion of a hand holding the bat. From Fig. 3.22, it can be subjectively observed that though the FS algorithm is optimum for video coding when minimum prediction error is the optimum criterion, using this algorithm also captures a large number of *false* motion vectors as shown in Fig. 3.22(a), particularly in the area of the table where there are no moving objects. In Fig. 3.22(c), it can be observed that TSS algorithm captured *false* motion vectors across most of the frame even where only camera motion was involved. Figs. 3.22(b) and (d) contrast the performance of the NTSS algorithm in capturing the *true* object motion vectors with that of the DTS algorithm for this sequence. It shows similar results, though there are even more *false* motion vectors visible around the side of the table. From this elementary analysis, an initial conclusion can be drawn that the DTS algorithm is capable of outperforming existing fast, as well as the FS algorithm, in capturing more *true* object motion vectors. This discussion indicates that motion estimation searching algorithms, driven principally by the need to locate the minimum BDM to optimise video coding, do not necessarily lead to accurate *true* object motion in many instances. The DTS algorithm in contrast, does provide prima facie evidence of an improved *true* object motion performance.



(a) (b)

Fig. 3.21: (a) Current frame #32 and (b) reference frame #33 of the *Table Tennis* video sequence.



(a) FS (b) DTS: LT(8)

(c) TSS (d) NTSS

Fig. 3.22: The motion vectors obtained from all four search algorithms applied to the frame pair #32 and #33 of the *Table Tennis* sequence.

Clearly, comparing the motion vector needle diagrams above provides only a visual qualitative judgment as to the potential superiority of the DTS algorithm in capturing an improved number of *true* object motion vectors. However, to prove this observation quantitavely, motion vectors must first be compensated for their *global* motion components (if any), and then all *false* object motion vectors must be removed. The number of *true* object motio vectors retained after this process can then be compared to make an evaluation. This process is elaborated in detail in Chapter 6 through the introduction of both modified *global* motion esti: ation and a compensation strategy, and a *false* motion vector elimination system using a novel filtering technique.

3.7 Summary

In this chapter, a new *Distance-dependent Thresholding Search* (DTS) algorithm has been presented for block-based motion estimation in video coding and *true* object motion estimation for object motion based video analysis. The important feature of the DTS algorithm is that the FS as well as fast searching modes are encompassed, with different threshold settings, providing

various quality-of-service levels. A unique characteristic of the DTS algorithm is that it has the flexibility of being able to trade predicted picture quality (MSE or PSNR) for search speed, across the full range of LT for different video sequences. This flexibility has considerable potential for exploitation in a wide range of applications ranging from low-bit rate video conferencing, through to adaptive high-quality video coding. Though the values of the control parameter were manually defined in this chapter, the DTS algorithm will be extended to a fully adaptive distance-dependent thresholding search (FADTS) algorithm in Chapter 5, to achieve guaranteed levels of QoS based on user demands in performance management motion estimation for video coding applications.

The performance of the DTS algorithm has also been compared to other popular fast algorithms such as TSS and NTSS, and it has been proven that the LT function DTS provided superior speed performance. While it retained a distortion error similar to the minimum value produced by the optimal FS algorithm, for stationary or quasi-stationery video sequences such as *Miss America*, its searching efficiency was not as high for high motion sequences such as *Flower Garden* and *Football*. This drawback will be addressed in the next chapter where the DTS algorithm will be modified into a fast DTS algorithm by considering additional motion related information.

Finally, the DTS algorithm was shown to afford superior performance in terms of capturing *true* object motion vectors compared to FS, NTSS, and TSS algorithms from a qualitative perspective. A comprehensive evaluation of this aspect of the DTS algorithm will be presented in Chapter 6.

Chapter 4

Adaptive-Centre Diamond Search DTS Algorithm

4.1 Introduction

In the previous chapter, it has been shown that the DTS algorithm (**Block 1** in Fig. 1.6) is capable of responding to any QoS requirements in terms of predicted image quality and computational complexity using different threshold settings provided by a control parameter. It was also demonstrated that with low motion video sequences such as *Miss America*, the error performance of the DTS algorithm was very similar to optimum FS performance while the computational cost was almost 55% and 40% less than existing fast algorithms, TSS and NTSS, respectively. However, it was also shown that the error performance of the DTS algorithm was not as satisfactory for complex motion video sequences such as *Football* and *Flower Garden*, when the processing speed was comparable to fast algorithms such as the above. As explained in Chapter 3, the DTS algorithm fails to improve search points for video sequences with complex motion, as it transforms to nearly exhaustive FS mode for a large number of macroblocks with high motion vectors. To improve the performance of the DTS algorithm on high motion video sequences, two well-established enhancement measures have been implemented, which are explored in this chapter.

The review in Chapter 2 revealed that a number of enhancement algorithms exist which improve the search efficiency of any BMA by exploiting some interframe and/or intraframe correlation properties of a video sequence. Among these properties, spatio-temporal correlation among neighbouring blocks [49, 68, 86-90] is the most researched one, and this will be discussed in detail in the next section. Because of temporal and spatial correlations, the motion vector of a block in the current frame is highly correlated to the motion vector of the block of the same co-ordinates in the previous frame, and the adjacent blocks in the same frame respectively. If sufficient useful information can be obtained from the motion vectors of the previous blocks, the total number of search points needed to find the motion vector of the current block can be significantly reduced.

So far, the centre of the search window has been considered as the search starting point in the DTS algorithm. In this chapter, the DTS algorithm with a *Linear Thresholding* (LT)

function is refined by integrating motion correlation, referred to as the *Adaptive-Centre* DTS (ACDTS) algorithm (**Block 2** in Fig. 1.6), to automatically predict the best search starting point in each search window.

Again from Chapter 2, it can be seen that the search pattern's shape and size influence both search speed and error performance. The distribution of MV in image sequences with a gentle and smooth motion is highly biased towards the central region. It is shown by certain authors [45, 46, 53, 68, 85] that nearly 80%-98% of the MVs of different video sequences such as *Football*, *Tennis*, *Miss America*, and *Salesman* are enclosed in the central 5×5 pixel region, and around 80% are enclosed in the 3×3 pixel area around the initial search centre prediction point of each block. With this centre-biased characteristic, it is reasonable to place more search points in the central region of the search window to get more samples, as implemented in [85, 100, 101]. In Zhu and Ma [53], it has also been shown that about 52.76% to 98.70% of the motion vectors of different video sequences are enclosed in a circular area with a radius of 2 pixels, centred on the position of zero motion. This indicates that the search points within the circle of a 2 pixels' radius are the most appropriate ones to be chosen in composing the search pattern. It is also mentioned by Zhu and Ma in [53] that the block displacement of real-world video sequences are mainly in the horizontal and vertical directions. Based on these two observations, recent research, [53, 102-104] has proved that the diamond search pattern is more efficient than the square or any other rectangular shaped search pattern in terms of using fewer search points for comparable prediction quality. This is because such as pattern (i) tries to behave in an ideal circle shaped manner in order to cover all possible directions of an investigating motion vector; (ii) can find large motion blocks with fewer search points; and (iii) has reduced susceptibility to falling into a local optimum due to its relatively large step size in horizontal and vertical directions. The diamond search pattern is not only efficient but also quite regular and very simple to implement.

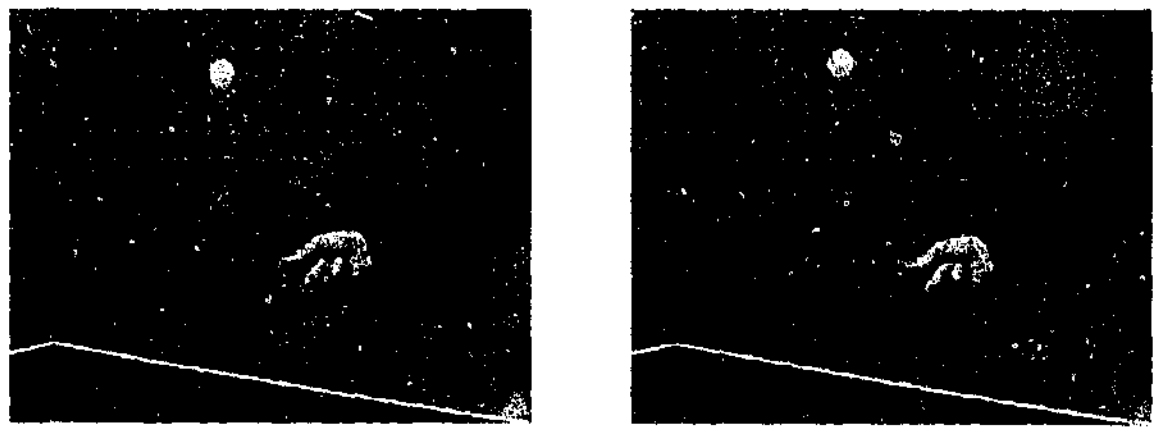
The performance of the ACDTS algorithm can be further enhanced by considering the diamond search pattern instead of the usual square shape for trading off quality and processing speed. Changing the pattern of searching from a square shape to a diamond shape leads to the *Adaptive-Centre Diamond Search* ACDTS (ACDSDTS) algorithm (**Block 2** in Fig. 1.6).

This chapter is organized as follows. Section 4.2 discusses the inter-block motion correlation among the neighbouring blocks in spatial and temporal domains. The proposed ACDTS and ACDSDTS algorithms are presented in Section 4.3. The overhead complexity incurred in ACDSDTS is also analysed in this section. Section 4.4 includes experimental results and performance analysis of the ACDTS and ACDSDTS algorithms with comparison against

fast algorithms such as TSS, NTSS, and *adaptive-Centre* NTSS (ACNTSS). Section 4.5 summarises the chapter.

4.2 Inter-block Motion Correlation

In general moving scenes, moving objects often cover many macroblocks such that the motion vector of spatial neighbouring blocks may be very similar. In addition, due to the continuity of motion in the temporal direction, the motion fields of the temporal neighbour blocks may be highly correlated. In other words, the motion vector of the current block can be predicted from the neighbouring blocks' motion vectors in the temporal (from the previous frame) or spatial (neighbouring blocks' motion vectors in the same frame) direction. These spatial and temporal correlations are clearly evident in Fig. 4.1 where the motion vectors for the *Table Tennis* sequence at frames #3 and #4 are obtained using the FS algorithm with maximum displacement $d = \pm 7$. From Figs. 4.1(a) or (b), it can observed that the motion vectors of a few of the neighbouring macroblocks of any moving object, such as a moving ball, show a similarity in magnitude and direction. This represents the spatial correlation among the neighbouring block's motion vectors in the same frame. From Figs. 4.1(a) and (b), it can be observed that the motion vectors of the corresponding macroblocks in both the frames are also very similar in magnitude and direction. This represents the temporal correlation between the motion vector of the current block and the motion vector of the corresponding block in the temporal direction. Using this information, a better search starting point can be predicted which can eventually reduce the computational burden associated with motion estimation of the current block.



(a) (b)

Fig. 4.1: The motion vector diagram for the *Table Tennis* sequence at frames #3 and #4 with respect to the previous frame.

As stated in the review in Chapter 2, a number of researchers have used this inter-block correlation to predict the starting point for a search window in order to reduce the computational

cost involved in motion estimation. Although in most cases, there is a high temporal correlation for a high temporal sampling rate, when the motion of objects changes direction abruptly or the speed of motion is not steady, tracking the motion from the previous frame's motion fields in the neighbourhood of the current block proves ineffective. Moreover, to keep the previous frame motion vectors in the decoder requires a large memory buffer [49], which will also complicate the system. For these reasons, only the spatial correlation has been integrated with the DTS algorithm to predict the search starting point and reduce the computational complexity in motion estimation.

The four spatial neighbouring blocks of the current block, *MV*0, in the current frame are shown in Fig. 4.2, where *MV*1 represents the previous block in the horizontal direction, and *MV*2, *MV*3 and *MV*4 are those in the vertical directions. Though other four neighbouring blocks could be available around the current block, the assumption only considering the four blocks is that motion vectors are calculated in row major order starting from the topmost row.

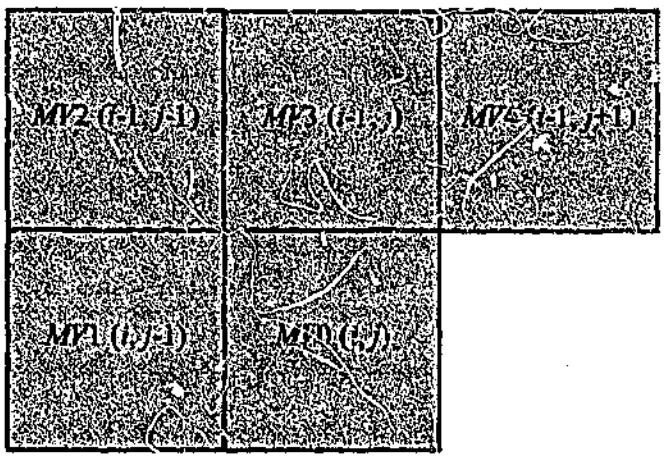


Fig. 4.2: Four neighbouring blocks around the current block, *MV*0.

For neighbouring blocks, some authors [68, 88] used the motion vectors of three adjacent blocks in the current frame, *MV*1, *MV*3 and *MV*4. Their argument in ignoring the motion vector of block *MV*2 is that the motion vector of this block is highly correlated with the horizontal or vertical neighbour block, and subsequently can be appropriately replaced by one, or both, the blocks. On the other hand, Xu *et al.* [49] and Cheung *et al.* [85] considered all the four neighbouring blocks for predicting the motion vector of the current block, without making the above assumption. Despite this, all the four neighbouring blocks as shown in Fig. 4.2, are used in this research.

The inter-block motion correlation is defined by the displacement between the current block's motion vector and the mean motion vector of the four neighbouring blocks' motion vectors, which is formulated as [49]:

$$\vec{D}_{mv} = \vec{V}_{MV0} - \vec{V}_m \tag{4.1}$$

where $\vec{V}_m = \frac{1}{4}\sum_{i=1}^{4}\vec{V}_{MVi}$ represents the mean motion vector, $\vec{V}_{MVi}$ is the motion vector of block MVi, $i = 0,1,...,4$, and $\vec{D}_{mv}$ is the displacement. When the magnitude of $\vec{D}_{mv}$ is small, the current block's motion vector can be considered highly correlated to those of its neighbours and this information can be used to predict the starting search centre of the current block.

4.3 The ACDTS and ACDSDTS Algorithms

4.3.1 Adaptive-Centre DTS (ACDTS) Algorithm

The purpose of centre prediction is to refine the DTS by integrating motion correlation within the DTS process to automatically predict the best search starting point in each window. The major advantage is that it can increase the chance of finding the real motion vector and reduce the computational requirement of the DTS algorithm, especially with high motion video sequences. This is because it reduces the distance between the starting search point and the global optimum point. The adaptive search centre for a block is predicted using four causal neighbouring motion vectors as shown in Fig. 4.2. This predictive centre is considered as the initial search centre and continues the DTS process for estimating the final motion vector. The search centre is predicted as follows:

Let $\vec{V}_{init}$ be the initial search window's centre from the origin of the current block. To predict the starting search point of any current block, it must first be determined whether the current block and its neighbours, as shown in Fig. 4.2, contain the same object. If the difference between the mean vector, $\vec{V}_m$, and each of all the four neighbouring blocks' motion vectors is less than a predefined threshold, it can be fairly assumed that these all blocks are covering the same moving object, or are in the background region. In such a case, the four neighbouring motion vectors can be used to predict an initial search centre, $\vec{V}_p$. Otherwise, no correlation can be established and no prediction will be made. The above process is formulated in [49]:

$$\vec{V}_{init} = \begin{cases} \vec{V}_p, & \text{if } \max_{i=1,...,4}\left\|\vec{V}_{MVi} - \vec{V}_m\right\| < T \\ (0,0), & \text{otherwise} \end{cases} \tag{4.2}$$

where T is a predefined threshold and $\|.\|$ represents the norm of the corresponding vector.

In [49], Xu *et al.* proposed three $\vec{V}_p$ motion prediction methods as follows:

centre-biased prediction: $\vec{V}_p = \arg\min_{\vec{V}_{MVi}}\left\|\vec{V}_{MVi}\right\|, i = 1,...,4$ (4.3)

mean prediction: $\vec{V}_p = round(\vec{V}_m)$ (4.4)

mean-biased prediction: $\vec{V}_p = \arg\min_{\vec{V}_{MVi}} \left\| \vec{V}_{MVi} - \vec{V}_m \right\|, i = 1,...,4$ (4.5)

where the arg function indicates that the argument, here the motion vectors, $\vec{V}_{MVi}$, of all four neighbouring blocks, for which the function value is minimum will be assigned to $\vec{V}_p$ instead of the minimum function value itself, and round (.) is the rounding of all elements of the vectors.

Among these three search centre prediction methods, the first one is suitable for those types of video sequences where the motion vectors of different blocks are usually gentle, smooth and only slowly vary with time. However, this method is not suitable for fast motion video sequences [49]. The *mean prediction* gives an accurate estimation if the assumption that the blocks cover the same moving object is true [49]. Xu *et al.* also mentions that sometimes, the four neighbouring blocks cover too large an area preventing tracking of any small motion, which may lead to produce larger prediction error when the mean prediction technique fails to track the real motion. Conversely, in *mean-biased prediction*, the motion vector which is the closest to the mean vector, $\vec{V}_m$, is selected to represent the object's movement. If all four blocks are within the same object, the predicted initial search point will be close to the real motion vector. Otherwise, these blocks probably belong to different motion objects. In this way, the selection of a minimum displacement can preserve the centre-biased distribution property of the motion vector [49] and maintain a better balance in both cases. Xu *et al.* [49] proves that the *mean-biased prediction* provides the best results compared to the other two methods (4.3 and 4.4). Based on this, the *mean-biased prediction* technique will be considered in this research to predict the initial search starting point.

4.3.2 The Formal ACDTS Algorithm

The ACDTS algorithm can be outlined as follows. If the current block is in the first row, or the first column, or the last column of a frame, the search starts using the origin of the search space as the initial search centre without any centre prediction. For other blocks, the ACDTS algorithm first predicts the initial starting search centre as in Section 4.3.1, and then the search progresses outwards by using search squares SS_r (Definition 3.2) in order, while keeping track of the current minimum MAE. A parametric thresholding function, defined in Section 3.4.2.1, is used to determine the various thresholds to be used with the search involving each search square, with the parameter is initialised at the onset of the search. After the searching of each search square is completed, the current minimum MAE is compared against the threshold value

of that specific search square and the search is terminated as soon as the current minimum MAE is found to be no higher than the threshold value. Fig. 4.3 summarises the proposed ACDTS algorithm.

- **Parameter** C_L
- **Precondition**: Pixel $p_{cx,cy}$ is the centre of search window with maximum displacement d.
- **Prediction**:
  If the current block is in the first row or the first column or the last column of a frame Then
  $MV_{predicted} = (0,0)$
  Else
  $MV_{predicted} = V_p$ in (4.5)
  $p_{cx,cy} = p_{cx,cy} + MV_{predicted}$
- **Initialisation:**
  $MAE_{\min} = MAE_{(cx,cy)}(0,0)$
  $MV = MV_{predicted}$
- **Main algorithm:**
  If $MAE_{\min} > 0$ Then
  For $\tau = 1,2,...,d$
  For each checking point $p_{x,y}$ in SS_τ such that $p_{cx,cy}$ is in the current block
  $\varepsilon = MAE_{(cx,cy)}(x-cx, y-cy)$
  If $\varepsilon < MAE_{\min}$ Then
  $MAE_{\min} = \varepsilon$
  $MV = (x-cx, y-cy) + MV_{predicted}$
  If $MAE_{\min} \le Threshold(\tau, C_L)$ Then STOP
- **Postcondition:** MV contains the motion vector and $MAE_{\min}$ the distortion error of the respective block.

Fig. 4.3: The ACDTS algorithm.

4.3.3 Adaptive-Centre Diamond Search DTS (ACDSDTS) Algorithm

Before defining the ACDSDTS algorithm, it is important to define the search diamond (SD_τ) pattern in the search space.

Definition 4.1 (Search Diamond SD_τ): The search space with maximum displacement, $\pm d$, centred at pixel, $p_{cx,cy}$, can be divided into d+1 mutually exclusive concentric search diamond, SD_τ such that a checking point at pixel, $p_{x,y}$, representing motion vector $(x-cx, y-cy)$, is in SD_τ if, and only if, the city-block distance (Definition 3.1) of the motion vector, $|x-cx|+|y-cy| = \tau$, for all $-d+cx \le x \le d+cx$, $-d+cy \le y \le d+cy$, and $\tau = 0,1,...,d$.

It can be readily verified that the number of checking points in search diamond SD_τ,

$$|SD_\tau| = \begin{cases} 1, & \tau = 0 \\ 4\tau, & \tau = 1,2,\ldots,d \end{cases} \tag{4.6}$$

and SD_τ represents the motion vectors of city-block distance, τ, that translates to a conventional Euclidean distance in the range of $\left[\frac{\tau}{\sqrt{2}}, \tau\right]$ for all $0 \le \tau \le d$. The checking points used in the first four search diamonds, SD_0, SD_1, SD_2, and SD_3, are shown in Fig. 4.4. It is interesting to note that $|SD_\tau| = \frac{1}{2}|SS_\tau|$ for $\tau = 1,2,\ldots,d$.

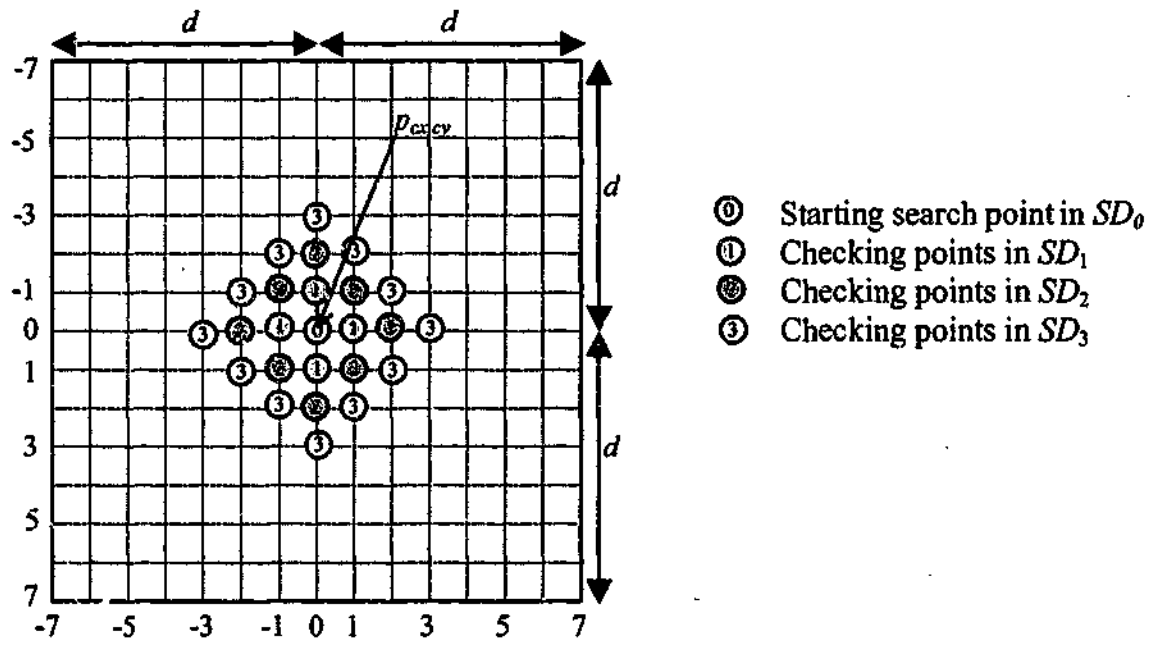


Fig. 4.4: Search diamond SD_0, SD_1, SD_2, and SD_3.

After predicting the initial search starting point as is done in Section 4.3.1, the final motion vector should be very close to the initial search window's centre. Now, if the above-mentioned diamond search pattern is used in the DTS algorithm, it will obtain the final motion vector by searching a lower number of search points, with a trade-off between quality and processing speed.

By integrating the search centre prediction method discussed in section 4.3.1 and the above-mentioned search diamond pattern, instead of the search square in DTS algorithm, an *Adaptive-Centre Diamond Search* DTS (ACDSDTS) algorithm is developed. This is presented in the next section.

4.3.4 The Formal ACDSDTS Algorithm

The ACDSDTS algorithm can be outlined as follows. As with the ACDTS algorithm, the search starts using the origin of the search space as the initial search centre without any centre prediction for a block in the first row, or the first column, or the last column of a frame. For other blocks, the ACDSTDS algorithm first predicts the initial starting search centre as is done in the ACDTS algorithm and then the search progresses outwards by using the search diamonds, SD_τ (Definition 4.1) in order, while keeping track of the current minimum MAE. A parametric thresholding function, defined in Section 3.4.2.1, is used to determine the various thresholds to be used with the search involving each search diamond, with the parameter being set at the onset of the search. After completion of searching each search diamond, the current minimum MAE is compared against the threshold value of that specific search diamond, and the search is terminated as soon as the current minimum MAE is found to be no higher than the threshold value. Fig 4.5 summarizes the ACDSDTS algorithm.

- **Parameter** C_L
- **Precondition**: Pixel $p_{cx,cy}$ is the centre of search window with maximum displacement d.
- **Prediction**:
  If the current block is in the first row or the first column or the last column of a frame Then
  $MV_{predicted} = (0,0)$
  Else
  $MV_{predicted} = V_p$ in (4.5)
  $p_{cx,cy} = p_{cx,cy} + MV_{predicted}$
- **Initialisation:**
  $MAE_{\min} = MAE_{(cx,cy)}(0,0)$
  $MV = MV_{predicted}$
- **Main algorithm:**
  If $MAE_{\min} > 0$ Then
  For $\tau = 1,2,\ldots, d$
  For each checking point $p_{x,y}$ in SD_τ such that $p_{cx,cy}$ is in the current block
  $\varepsilon = MAE_{(cx,cy)}(x-cx, y-cy)$
  If $\varepsilon < MAE_{\min}$ Then
  $MAE_{\min} = \varepsilon$
  $MV = (x-cx, y-cy) + MV_{predicted}$
  If $MAE_{\min} \le Threshold(\tau, C_L)$ Then STOP
- **Postcondition:** MV contains the motion vector and $MAE_{\min}$ the distortion error of the respective block.

Fig. 4.5: The ACDSDTS algorithm.

4.3.5 Computational Complexity Analysis

In Chapter 3, it was defined that the computational complexity associated with the DTS algorithm is bounded between $\varsigma(\Omega+1)$ and $\psi+d\varsigma$ operations per second (3.12 and 3.13) for integer-pel motion accuracy. For half-pel accuracy, these limits will be increased by $8\varsigma\,\Omega$ operations per second (Section 3.5.5). As the *mean-biased prediction* (4.5) adaptive–centre algorithm is integrated with the DTS algorithm to predict the starting search point, it will increase the computational cost of the whole process. If the number of operations involved in predicting the search centre for each block is R, it requires a total of $R\varsigma$ operations per second; that is the only extra overhead cost incurred in the whole motion estimation process in the ACDSDTS algorithm. According to Xu *et al.* [49], the value of R is only 23 arithmetic operations, which is much less than that required in the matching function evaluation for motion estimation in the DTS algorithm mentioned above. Experimental results also show that this overhead cost can be justified by reducing the number of checking points with better prediction error.

4.4 Experimental Results

The purpose of this section is to present the results of experiments performed to compare the performance of the FS, ACDTS, ACDSDTS, DTS, TSS, NTSS, and *adaptive-centre* NTSS (ACNTSS) algorithms for motion estimation in video coding applications. The MAE function was used as the criterion for locating the best motion vector for each block, and all the results shown in this section are formulated with half-pel accuracy motion estimation.

Test Video / T	*Table Tennis*		*Football*		*Flower Garden*	
	MSE	SP	MSE	SP	MSE	SP
3.0	90.80	186.98	366.27	196.66	275.26	190.14
3.5	90.82	186.19	366.63	195.31	275.28	188.71
4.0	90.84	185.66	366.67	194.17	275.29	188.02
4.5	90.89	184.97	366.73	192.61	275.29	187.55
5.0	90.91	184.40	367.35	190.93	275.28	187.17
5.5	90.93	183.82	367.98	189.14	275.29	186.83
6.0	90.98	183.34	368.81	187.65	275.29	186.54
6.5	91.38	182.83	369.73	186.14	275.29	186.31
7.0	91.50	182.51	370.33	185.13	275.29	186.26

Table 4.1: Performance comparison of different values of the predefined threshold, T, for the *Table Tennis*, *Football*, and *Flower Garden* sequences (1-50 frames).

The predefined displacement threshold, T, in (4.2) has also been tested for a range of different values. Table 4.1 shows the performance of T with a range of values from 3 to 7 for

the *Football*, *Table Tennis*, and *Flower Garden* video sequences. To ensure the optimum results for T, the FS algorithm is considered for calculating the motion vector. Experimental results show that the value of threshold, T, is not very sensitive to performance, especially for prediction error. For the T range from 3 to 7, the average MSE and the average number of search points (SP), of the first 50 frames of the *Table Tennis*, *Football*, and *Flower Garden* sequences, varies less than 1% and 5% respectively. For average performance, the predefined threshold value of T is considered as 5 in this research.

Performance Analysis

To compare the performance of the ACDSDTS, ACDTS, DTS, TSS, NTSS, and ACNTSS algorithms, a number of tests were performed using standard video sequences—*Table Tennis*, *Football*, *Flower Garden*, *Salesman*, and *Miss America* (Appendix B). The block size and maximum displacement were [16,16] and ±7 respectively. As indicated in Chapter 3, although the performance of the DTS algorithm was better than that of the TSS or NTSS algorithm on low motion video sequences such as the *Miss America*, it was not so on high motion video sequences such as the *Flower Garden* and *Football*. The motivation of this chapter has been primarily to improve the performance of the DTS algorithm on high motion video sequences. For this reason, the test results for only the *Football* and *Flower Garden* sequences are included in this chapter, and the results for the *Table Tennis*, *Salesman*, and *Miss America* sequences are included in Appendix D. The *Flower Garden* sequence comprises mainly camera panning, while the *Football* sequence consists of complex motions ranging from slow motion to very fast motion.

To quantitatively evaluate the performance of the ACDSDTS, ACDTS, DTS, FS, TSS, NTSS, and ACNTSS algorithms, the following three measures were considered: -

- The average MSE (error performance) per pixel between the estimated and the corresponding original frames.
- The average PSNR (predicted image quality) per pixel between the estimated and the corresponding original frames.
- The average number of search points (SP) per motion vector as computational complexity.

The experimental results are presented in the following two-sub sections.

4.4.1 Performance Analysis of the ACDTS Algorithm

This section analysises the performance of the ACDTS algorithm against that of the DTS algorithm and some other fast BMAs for motion estimation.

Average MSE Performance

The statistical performance comparison of the FS, ACDTS, DTS, TSS NTSS, and ACNTSS algorithms in terms of the average MSE per pixel are shown in Tables 4.2 and 4.3 for the *Flower Garden* and *Football* video sequences respectively. The ACDTS algorithm achieved better performance than that of the DTS algorithm for both test video sequences, using different values of the threshold control parameter, C_L. It can be observed from these tables that the gain in terms of MSE gradually increased as the values of C_L were increased. This is because the higher the value of the threshold considered, the higher the probability of search deviation from the global minimum. This indicates that predicting the search centre through motion tracking increases the chance of finding better motion vectors.

Block-matching algorithms		*Flower Garden* sequence							
		MSE	PSNR [dB]	SP	Block-matching algorithms		MSE	PSNR [dB]	SP
DTS	$C_L=2$	208.12	24.95	135.43	ACDTS	$C_L=2$	208.01	24.95	133.40
	$C_L=4$	210.10	24.91	81.48		$C_L=4$	208.56	24.94	78.59
	$C_L=6$	214.58	24.82	53.93		$C_L=6$	209.82	24.91	50.92
	$C_L=8$	219.52	24.72	41.59		$C_L=8$	211.72	24.87	38.51
	$C_L=10$	226.06	24.59	35.00		$C_L=10$	214.22	24.82	31.74
	$C_L=11$	229.01	24.53	37.89		$C_L=11$	215.50	24.80	29.38
	$C_L=12$	231.78	24.48	31.06		$C_L=12$	216.82	24.77	27.58
	$C_L=14$	237.25	24.38	28.47		$C_L=14$	220.26	24.70	24.96
	$C_L=18$	251.26	24.13	25.21		$C_L=18$	228.94	24.53	21.72
	$C_L=20$	259.83	23.98	24.11		$C_L=20$	233.45	24.45	20.59
	$C_L=24$	277.23	23.70	22.63		$C_L=24$	242.51	24.28	19.07
FS		207.98	24.95	209.77	ACNTSS		213.41	24.60	27.37
TSS		244.75	24.24	31.22					
NTSS		214.39	24.82	29.63					

Table 4.2: Average MSE and PSNR per pixel, and search points (SP) per motion vector for the *Flower Garden* sequence (1-80 frames) with different BMAs.

It is also shown in Table 4.2 that for the *Flower Garden* sequence, the performance of the ACDTS was better than TSS, and very similar to NTSS algorithms. For example, ACDTS with $C_L = 10$ & $C_L = 11$, achieved almost 15% better error performance compared to TSS, and was similar to the NTSS and ACNTSS algorithms, with the search speed being very similar in all cases. On the other hand, for the *Football* sequence, Table 4.3 shows that though the performance of the ACDTS was not as satisfactory compared to TSS or NTSS algorithm, search efficiency was better than that for the DTS algorithm. For example, the DTS algorithm with $C_L = 8$ had an average 308.27 MSE, with a search point average of 49.75, whereas the ACDTS

algorithm with C_L = 9 had an average of 303.90 MSE, with a search point average of 42.75. In this case, the ACDTS algorithm achieved 2% better error performance when the search speed was 17% faster than the DTS algorithm.

Block-matching algorithms		*Football* sequence						
		MSE	PSNR [dB]	SP	Block-matching algorithms	MSE	PSNR [dB]	SP
DTS	C_L = 2	278.87	23.68	131.71	ACDTS C_L = 2	275.84	23.72	131.48
	C_L = 4	281.55	23.64	90.57	C_L = 4	277.71	23.69	88.92
	C_L = 6	291.67	23.48	65.10	C_L = 6	284.58	23.59	62.99
	C_L = 8	308.27	23.24	49.75	C_L = 8	296.76	23.41	47.77
	C_L = 9	318.08	23.11	44.52	C_L = 9	303.90	23.30	42.75
	C_L = 10	327.18	22.98	40.55	C_L = 10	310.50	23.21	38.80
	C_L = 12	345.45	22.75	34.88	C_L = 12	324.08	23.02	33.39
	C_L = 14	361.30	22.55	31.18	C_L = 14	336.27	22.86	29.77
	C_L = 16	376.46	22.37	28.51	C_L = 16	346.63	22.73	27.19
	C_L = 18	390.73	22.21	26.50	C_L = 18	357.47	22.60	25.33
	C_L = 20	403.08	22.08	24.96	C_L = 20	367.77	22.48	23.92
FS		275.76	23.73	210.05	ACNTSS	299.51	23.37	28.32
TSS		309.11	23.22	31.11				
NTSS		303.35	23.31	28.79				

Table 4.3: Average MSE and PSNR per pixel, and search points (SP) per motion vector comparison for the *Football* sequence (1-80 frames) with different BMAs.

Peak Signal-to-Noise Ratio (PSNR) Performance

The performance comparison of the FS, ACDTS, DTS, TSS NTSS, and ACNTSS algorithms in terms of the average PSNR per pixel are shown in Tables 4.2 and 4.3 for the *Flower Garden* and *Football* video sequences respectively. The PSNR value was calculated by using (3.11). Table 4.2 shows that the PSNR performance of ACDTS was better than that of TSS, and very similar to the NTSS algorithm for the *Flower Garden* sequence. For example, ACDTS with C_L = 10 and C_L = 11, achieved almost average 0.6 dB gain in PSNR compared to TSS, and an almost similar PSNR compared to NTSS algorithm when the search speed was very similar to TSS, and NTSS, respectively. Conversely, for the *Football* sequence, Table 4.3 shows that though the performance of the ACDTS was not as satisfactory compared to TSS or NTSS algorithm, the search efficiency was better than the DTS algorithm. For example, the DTS algorithm with C_L = 8 had an average 23.24 dB PSNR with a search point average of 49.75, whereas the ACDTS algorithm with C_L = 9 had an average 23.30 dB PSNR with a search point average of 42.75. In this case, the ACDTS algorithm gained 0.06 dB PSNR when the search speed was 17% faster than the DTS algorithm.

Search Speed Comparison

The performance of the FS, TSS, NTSS, ACNTSS, DTS, and ACDTS algorithms in terms of the average search points per block when estimating motion vectors is also presented in Tables 4.2 and 4.3 for the *Flower Garden* and *Football* sequences respectively. It is clear from these tables that the average number of search points (SP) needed by ACDTS algorithm was always less than that of the DTS algorithm for both sequences with any value of C_L. Table 4.2 also demonstrates that the search speed of ACDTS was better than that of TSS algorithm, and very similar to NTSS algorithm for the *Flower Garden* sequence, while achieving the same MSE or PSNR values. For example, ACDTS with C_L = 24 and C_L = 10, achieved the similar MSE or PSNR performance, while the search speed was almost 40% faster than that of TSS, and very similar to that of the NTSS algorithm. On the other hand, Table 4.3 shows that though the searching efficiency of the ACDTS algorithm with the same average MSE or PSNR was not as good compared to that of the NTSS or TSS or ACNTSS algorithm for the *Football* video sequence, it was better than that of the DTS algorithm.

It is interesting to note in Tables 4.2 and 4.3 that the performance of the ACNTSS algorithm did not improve significantly compared to the original NTSS algorithm. The reason can be explained as follows. If the motion vector distribution is within only a 3×3 pixels region after predicting the search centre, the probability of reaching the global minimum point is very high for the ACNTSS algorithm. However, the motion vector distribution around the predicted search centre will not always be within a 3×3 pixels region. For high motion video sequences, the motion vector distribution around the predicted search centre will be within a 5×5 region for many macroblocks. In such cases, the ACNTSS algorithm will fail to improve, or may degrade, compared to the NTSS algorithm due to its large directional step size. It can be concluded that the performance of directional algorithms may not always improve even after initial search centre prediction.

4.4.2 Performance Analysis of the ACDSDTS Algorithm

This section presents comparative results while considering both adaptive centre-prediction and the diamond search pattern technique in the DTS algorithm described in Section 4.3, in terms of fast motion estimation trade-offs between quality and complexity.

Average MSE Performance

The performance of the ACDSDTS algorithm, in terms of the average MSE per pixel between the estimated and original frames, is shown in Tables 4.4 and 4.5 for both the *Flower Garden* and *Football* sequences. From Tables 4.2, 4.3, 4.4, and 4.5, it can be observed that the

ACDSDTS algorithm showed better performance than that of the ACDTS algorithm for both test video sequences with trade-offs in quality and processing speed when using different values of C_L.

Block-matching algorithm		*Flower Garden* sequence		
		MSE	PSNR [dB]	SP
ACDSDTS	$C_L = 2$	212.55	24.86	72.10
	$C_L = 4$	212.70	24.85	44.96
	$C_L = 6$	213.39	24.84	30.80
	$C_L = 7$	213.60	24.83	26.97
	$C_L = 9$	214.66	24.81	22.26
	$C_L = 10$	215.42	24.80	20.71
	$C_L = 12$	216.74	24.77	18.57
	$C_L = 14$	218.15	24.74	17.20
	$C_L = 16$	219.43	24.72	16.27
	$C_L = 18$	221.70	24.67	15.59
	$C_L = 24$	228.18	24.55	14.30

Table 4.4: Average MSE and PSNR per pixel, and search points (SP) per motion vector of the ACDSDTS algorithm for the *Flower Garden* sequence (1-80 frames).

Block-matching algorithm		*Football* sequence		
		MSE	PSNR [dB]	SP
ACDSDTS	$C_L = 2$	290.23	23.50	65.85
	$C_L = 4$	296.74	23.41	46.69
	$C_L = 6$	301.19	23.34	35.09
	$C_L = 8$	305.59	23.28	28.02
	$C_L = 9$	307.72	23.25	25.71
	$C_L = 12$	317.64	23.11	21.36
	$C_L = 14$	328.88	22.96	19.66
	$C_L = 16$	339.70	22.82	18.40
	$C_L = 18$	349.59	22.70	17.50
	$C_L = 20$	359.98	22.57	16.83

Table 4.5: Average MSE, PSNR per pixel, and search points (SP) per motion vector of the ACDSDTS algorithm for the *Football* sequence (1-80 frames).

From Tables 4.2 and 4.4, it can also be observed that the ACDSDTS algorithm performed not only better than TSS but also better than NTSS, or ACNTSS, for the *Flower Garden* video sequence. For example, ACDSDTS with $C_L = 6$ and $C_L = 7$, achieved almost 15% and 1% better error performance compared to the TSS and NTSS algorithms, while its search speed was 13% and 8% faster than both. For this sequence, its performance was very similar to the ACNTSS

algorithm. Tables 4.3 and 4.5 prove that ACDSDTS outperformed TSS, and was very similar to NTSS or ACNTSS even for the *Football* video sequence. For example, ACDSDTS with $C_L = 9$, achieved almost 3% better error performance compared to the TSS algorithm despite its search speed being almost 20% faster than the TSS algorithm.

Figs 4.6 and 4.7 plotted the average MSE performance of the ACDSDTS, ACDTS, FS, TSS, NTSS, and ACNTSS algorithms for the first 80 frames of the *Flower Garden* and *Football* sequences. For the sake of clarity in plotting, we have only considered the values of C_L for the ACDTS and ACDSDTS algorithms that used a search speed similar to the TSS, NTSS, and ACNTSS algorithms. These figures clearly show the improvement of the ACDSDTS algorithm over the ACDTS algorithm in terms of prediction quality when considering the similar computational complexity.

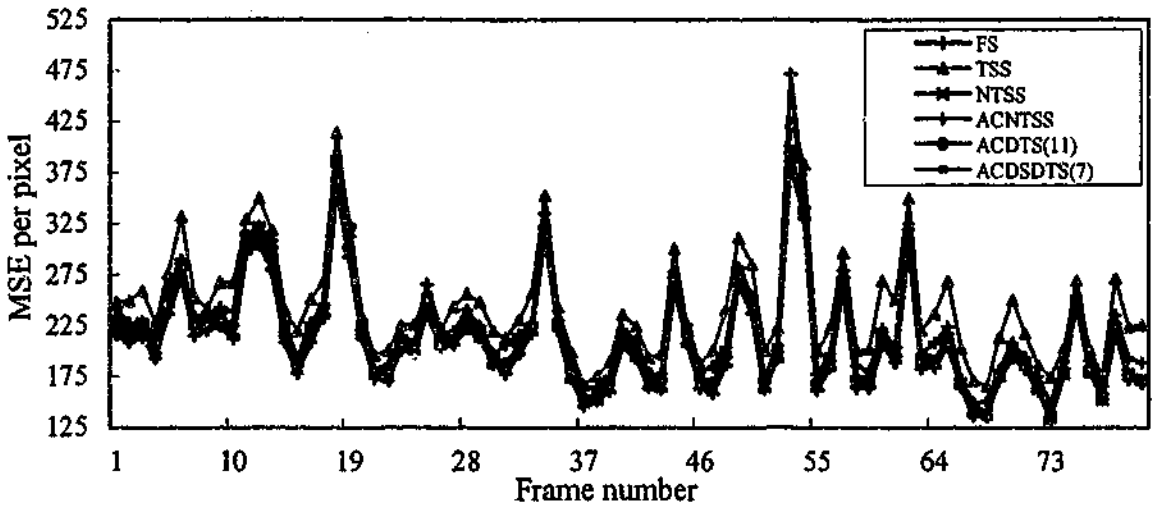


Fig. 4.6: Average MSE performance comparison of FS, TSS, NTSS, ACNTSS, ACDTS, and ACDSDTS for the *Flower Garden* video sequence.

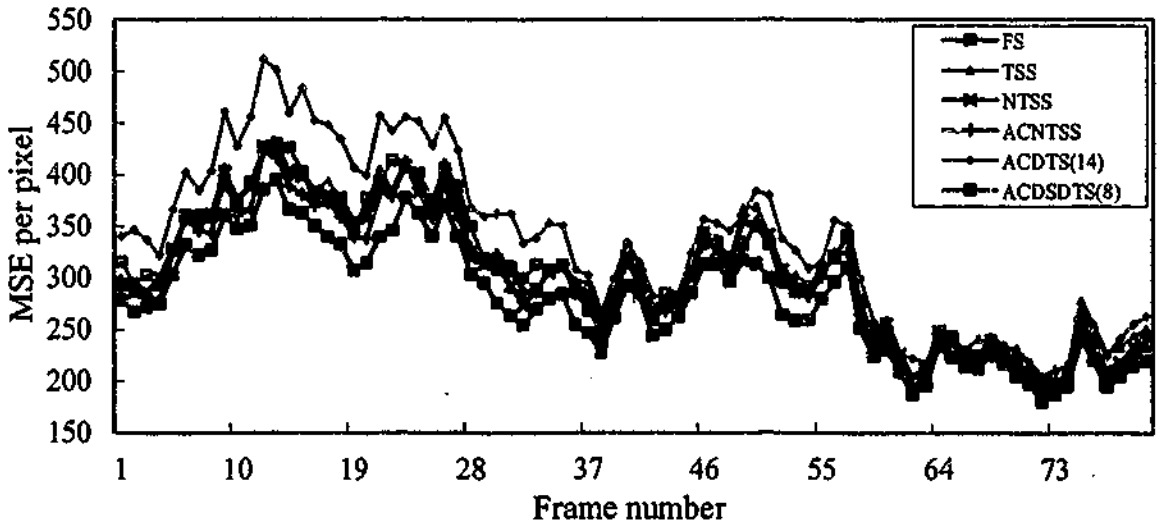


Fig. 4.7: Average MSE per pixel comparison of FS, TSS, NTSS, ACNTSS, ACDTS, and ACDSDTS for the *Football* video sequence.

Peak Signal-to-Noise Ratio (PSNR) Performance

The performance of the ACDSDTS algorithm, in terms of the average PSNR per pixel between the estimated and original frames, is shown in Tables 4.4 and 4.5 for the *Flower Garden* and *Football* sequences. From Tables 4.2, 4.3, 4.4, and 4.5, it can be observed that the ACDSDTS algorithm showed better performance than that of the ACDTS algorithm for both test video sequences with trade-offs between quality and processing speed when using different values of C_L. From Tables 4.2 and 4.4, it can also be observed that the ACDSDTS algorithm performed not only better than the TSS but also better than NTSS, or ACNTSS algorithm for the *Flower Garden* video sequence. For example, ACDSDTS algorithm with $C_L = 6$ and $C_L = 7$, achieved almost 0.6, 0.02, and 0.3 dB gain in PSNR compared to TSS, NTSS, and ACNTSS algorithm respectively, while search speeds were similar in all cases. On the other hand, Tables 4.3 and 4.5 prove that ACDSDTS outperformed TSS, and was similar to NTSS or ACNTSS algorithm even with the *Football* video sequence. For example, ACDSDTS with $C_L = 9$, achieved almost 0.03 dB gain in PSNR compared to TSS, when its search speed was almost 20% faster than that of the TSS algorithm.

Search Speed Comparison

The performance of the ACDSDTS algorithm in terms of the actual average number of search points per block in estimating motion vectors is also presented in Tables 4.4 and 4.5 for the Flower Garden and Football sequences. It is clear from Tables 4.2, 4.3, 4.4, and 4.5 that the average number of search points (SP) needed with the ACDSDTS algorithm was always less than that of the ACDTS algorithm for both video sequences, while providing same prediction quality. Tables 4.2 and 4.4 also prove that the search speed of the ACDSDTS algorithm, for example, in the case where $C_L = 24$, was more than 55% faster than the TSS algorithm, with more than 0.4 dB gain in PSNR for the Flower Garden sequence. Again, with $C_L = 7$ and $C_L = 18$, the ACDSDTS algorithm achieved almost 10% and 45% faster search speeds compared to the NTSS and ACNTSS algorithms respectively with similar MSE or PSNR performance. On the other hand, Tables 4.3 and 4.5 show that for the *Football* sequence, the search speed of ACDSDTS, in the case of $C_L = 9$ and $C_L = 8$, was almost 20% higher than that of TSS and similar to NTSS or ACNTSS, with the same PSNR or MSE performance. These results clearly indicate that the ACDSDTS algorithm not only outperformed for low motion sequences but also for any complex motion video sequences.

Figs. 4.8 and 4.9 plot the average SP performances of the ACDSDTS, ACDTS, TSS, NTSS, and ACNTSS algorithms for the first 80 frames of the *Flower Garden* and *Football* sequences. For the sake of clarity in plotting, the values of C_L for the ACDTS and ACDSDTS algorithms

that achieved the PSNR or MSE performance comparable to the TSS, NTSS, and ACNTSS algorithms have been considered. From the figures, it can also be observed that the performance of ACDSDTS algorithm over the ACDTS, TSS, NTSS, and ACNTSS algorithms in terms of processing speed in terms of SP is remarkably better for the *Flower Garden* sequence, and very similar for the *Football* sequence.

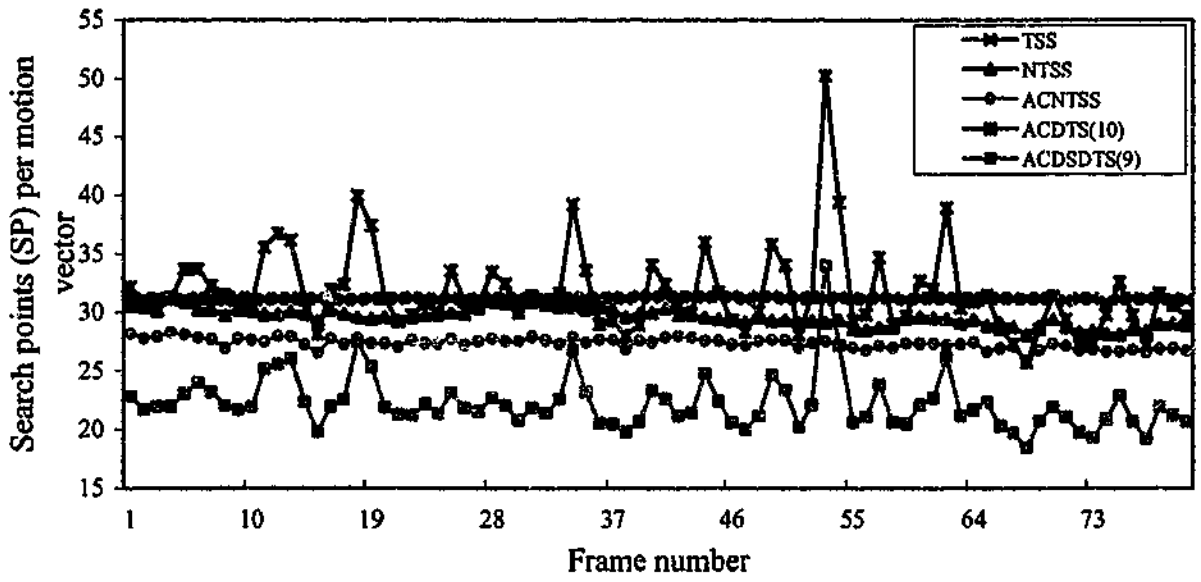


Fig. 4.8: Average search point (SP) comparison of FS, TSS, NTSS, ACNTSS, ACDTS, and ACDSDTS for the *Flower Garden* sequence.

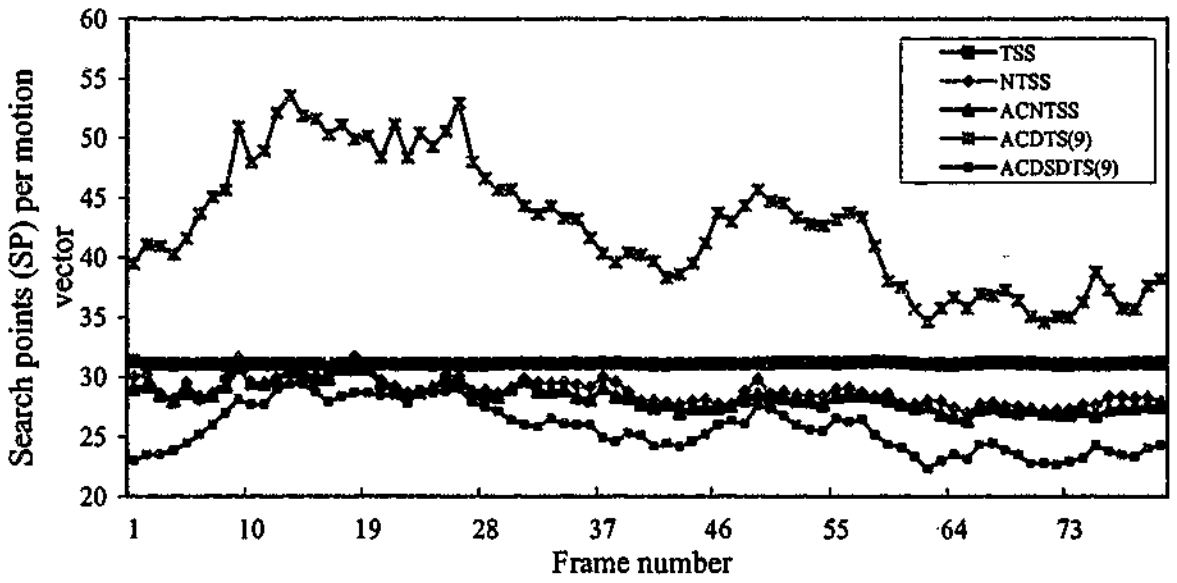
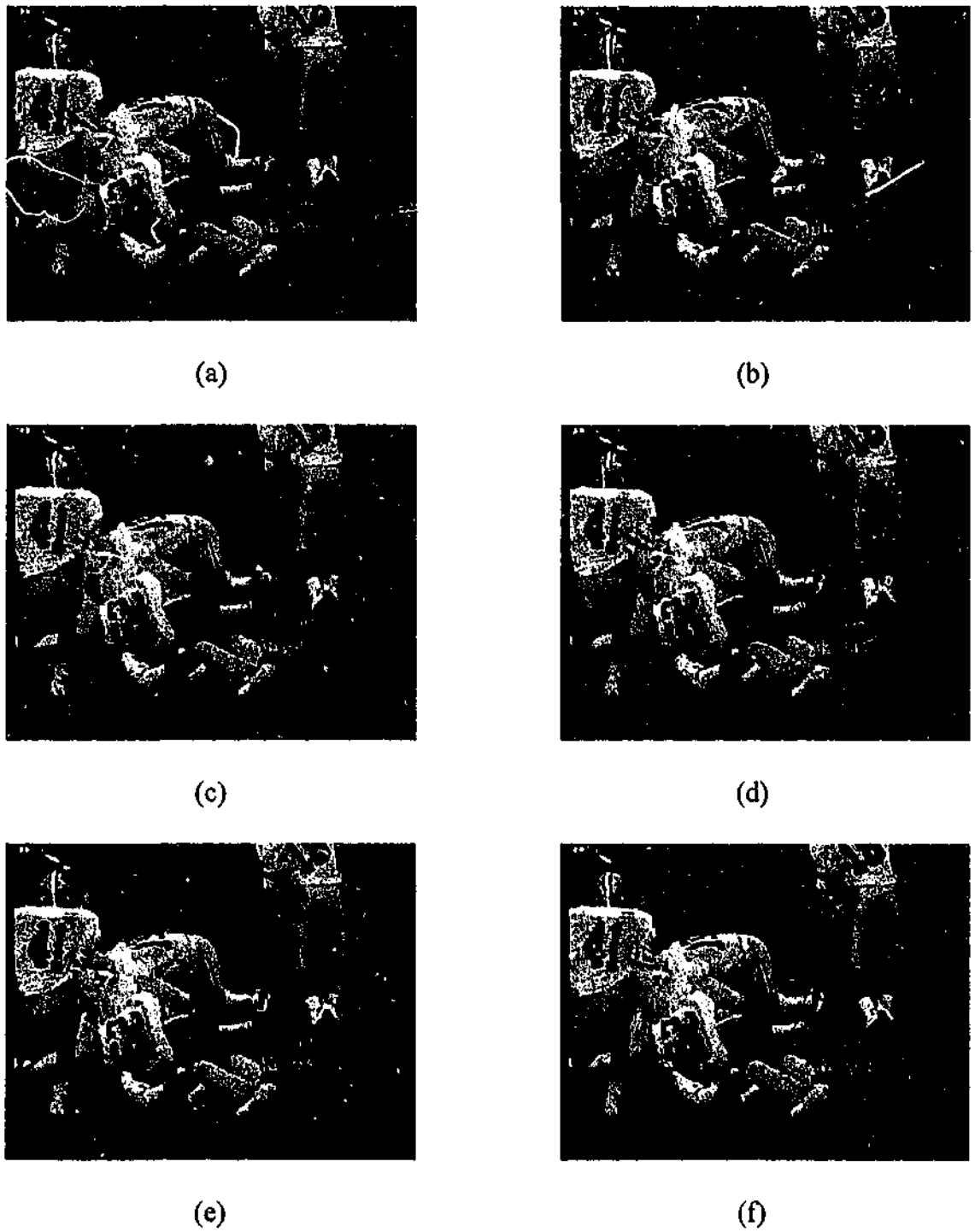


Fig. 4.9: Comparisons of average search points (SP) per motion vector of the FS, TSS, NTSS, ACNTSS, ACDTS, and ACDSDTS for the *Football* video sequence.

4.4.3 Qualitative Evaluation

The performance of the TSS, NTSS, ACNTSS, ACDTS, and ACDSDTS algorithms compared to the FS algorithm for the *Football* video sequence was also evaluated based on the perceptual

predicted image quality. Fig. 4.10 shows the estimated 40th frame of the *Football* sequence with different BMAs.

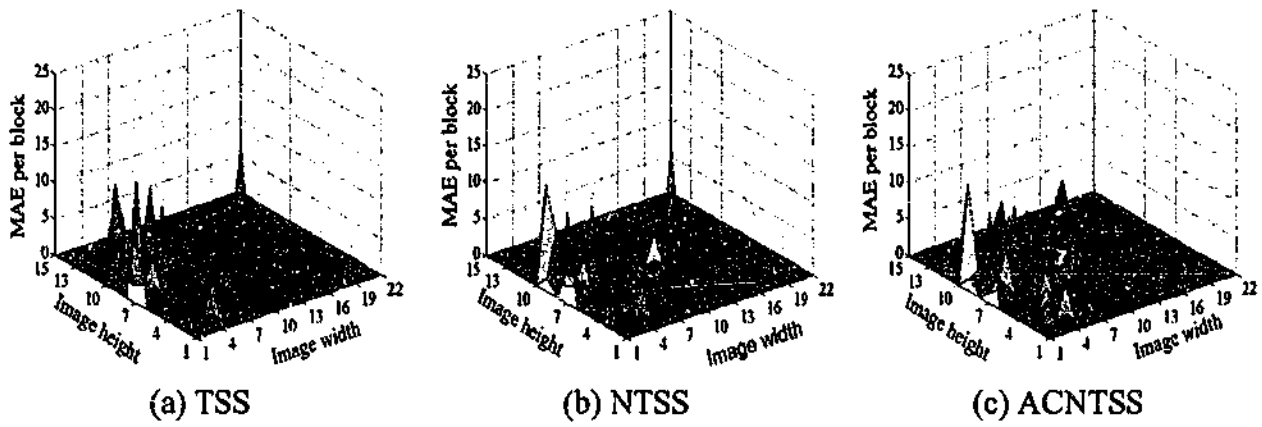


(a) (b) (c) (d) (e) (f)

Fig. 4.10: Estimated image of the 40th frame of the *Football* sequence: (a) FS, (b) TSS, (c) NTSS, (d) ACNTSS, (e) ACDTS: C_L = 14, and (f) ACDSDTS: C_L = 8 algorithms.

As the FS is the optimum in terms of error performance, Fig. 4.11 shows the MAE per block distribution for all other BMAs with respect to the FS algorithm. In terms of subjective image quality, the performance of ACDSDTS was very similar to the TSS, NTSS, and

ACNTSS algorithms given this complex motion video sequence, where search speed was almost 15% higher than TSS, and similar to NTSS or ACNTSS algorithm.



(a) TSS (b) NTSS (c) ACNTSS

(d) ACDTS: $C_L = 14$ (e) ACDSDTS: $C_L = 8$

Fig. 4.11: Prediction error distribution of the 40th frame of the *Football* sequence with respect to that of the FS algorithm.

4.5 Summary

In this chapter, an *Adaptive-Centre* DTS (ACDTS) algorithm has been developed by integrating spatial inter-block motion correlation within the DTS algorithm to automatically predict the best search starting point close to the global minimum. Experimental results have shown that the ACDTS algorithm improved the performance of the DTS algorithm for all test video sequences. The search efficiency of the ACDTS algorithm has further been improved by considering the diamond search pattern instead of the traditional rectangular search pattern, through trade-offs between quality and computational complexity.

Experimental results have shown that the ACDSDTS algorithm effectively improved its performance in terms of MSE or PSNR, with lower average searching points. It has also been shown that the ACDSDTS algorithm outperformed two very well-known fast BMAs, TSS and NTSS, for the *Flower Garden* video sequence, and demonstrates a very similar performance for the *Football* video sequence. This indicates the effectiveness of the ACDSDTS algorithm for use with any motion video sequences.

The overhead complexity incurred in ACDTS or ACDSDTS algorithm in relation to motion estimation was also analysed, and it was shown that the number of operations required for determining the initial search centre using *mean-biased prediction* is very low compared to the whole motion estimation process. This was the only overhead cost incurred in the operation of the ACDSDTS algorithm. Thus, the proposed algorithm incurs negligible computational overhead.

It has also been shown that like the DTS algorithm, the ACDSDTS algorithm has achieved different levels of performance in terms of quality as well as processing speed in terms of SP per motion vector, using different values of the threshold control parameter, where it was selected manually. The most important and challenging part of this process is how this control parameter can be adapted automatically to a target level of quality or processing speed. In the next chapter, this issue will be addressed by presenting a *Fully Adaptive Distance-dependent Thresholding Search* (FADTS) algorithm for motion estimation.

Chapter 5

Fully Adaptive Distance-dependent Thresholding Search Algorithm

5.1 Introduction

The *Adaptive-Centre Diamond Search Distance-dependent Thresholding Search* (ACDSDTS) algorithm (**Block 2** in Fig. 1.6) signifies an important improvement to the *Distance-dependent Thresholding Search* (DTS) algorithm (**Block 1** in Fig. 1.6), by increasing the search efficiency of the DTS process, by trading-off between predicted image quality and complexity. However, both the DTS and ACDSDTS algorithms appear to be highly dependent on predefined linear threshold values that are controlled by the control parameter, C_L, for performance scalability in motion estimation. In the ACDSDTS algorithm, the value of C_L is manually set at the onset of the search. As indicated in Chapters 3, the control parameter, C_L, allows users some flexibility to control the DTS search (or, its enhancements, as developed in Chapter 4) in order to achieve specific target prediction error, or search points (if achievable) by the trial and error method. However, setting the optimum C_L value by trial and error severely limits the flexibility of these searching algorithms, especially when motion estimation must be carried out in real-time. On the other hand, it is quite impractical to use the same C_L value for all the frames of a video sequence, especially when the motion content varies significantly throughout the video sequence. Therefore, to derive the full potential of the DTS algorithm compared with algorithms such as TSS and NTSS, or any other non-flexible BMA, the value of C_L must be adjusted automatically based on the content of the video as well as user QoS demands.

This chapter presents a new *Fully Adaptive Distance-dependent Thresholding Search* (FADTS) algorithm (**Block 3** in Fig. 1.6), which can dynamically adjust the value of C_L to achieve QoS requirement in terms of either predicted image quality or processing speed as the target. This adaptive algorithm is especially important for complexity management in software-only video coding or low power coding (mobile or handheld computing platforms), as it require more a flexible approach in trading-off between predicted image quality and computational complexity.

The remainder of this chapter is organized as follows. In Section 5.2, an adaptive system for the FADTS algorithm, for performance management block-based motion estimation in real-time video coding applications, is proposed by incorporating a novel adaptive model based on the *Normalized Block Least Mean Square* (NBLMS) technique. Some fundamental concepts in adaptive systems are briefly discussed in the light of adapting the control parameter, C_L, in the ACDSDTS algorithm. In Section 5.3, an integrated shot detection technique using *Artificial Neural Network* (ANN) and BDM thresholding are presented with a brief review of existing techniques. As the initial value of the threshold control parameter impacts significantly on the adaptation process, automatically initialisation process of this parameter is discussed in Section 5.4. The impact of other different parameters, and their possible operating ranges related to the proposed adaptive system are then analysed in Section 5.5, while the computational complexity of the proposed searching algorithm is analysed in Section 5.6. Both experimental results and the performance analysis of the proposed algorithms are included in Section 5.7. Section 5.8 summarises this chapter.

5.2 Adaptive Algorithms

According to Widrow and Stearns [105], "*An adaptive automation is a system whose structure is alterable or adjustable in such a way that its behaviour or performance (according to some desired criterion) improves through contact with its environment.*" These types of systems usually have the following characteristics: -

- They can automatically adapt (self-optimise) in the face of changing (non-stationary) environments and changing system requirements.
- They can usually be described as nonlinear systems with time-varying parameters.

Consequently, an adaptive algorithm is a procedure that changes its parameters as it gains more knowledge of its possibly changing environment. Preferably, the algorithm will change its parameters in a fashion that optimizes some criteria such as the mean squared difference between two given signals. An adaptation process can be classified as [105]:

1. Open-loop adaptation
2. Closed-loop adaptation

Open-Loop Adaptation

The open-loop adaptive process involves making measurements of input or environment characteristics, applying this information to a formula or to a computational algorithm, and using the results to set the adjustment of an adaptive system. The principle of open-looped adaptation is shown in Fig. 5.1. In this configuration, a computer or signal processor performs

the adjustments based on an adaptation algorithm carried on a set of input signals and the environment data.

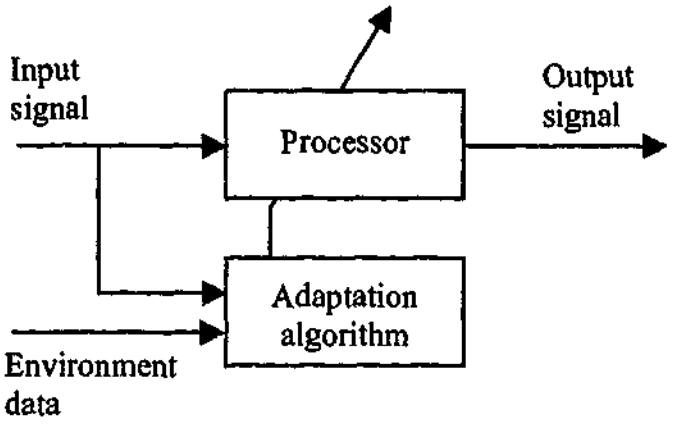


Fig. 5.1: The open-loop adaptation process.

Closed-Loop Adaptation

Closed-loop adaptation involves automatic experimentation with the adjustments of the adaptive system and knowledge of their outcomes in order to optimise a measured system performance. This process may be called adaptation by *performance feedback*. The principle of closed-loop adaptation is shown in Fig. 5.2. In this case, the performance criterion is the function of input signal, output signal and the target output.

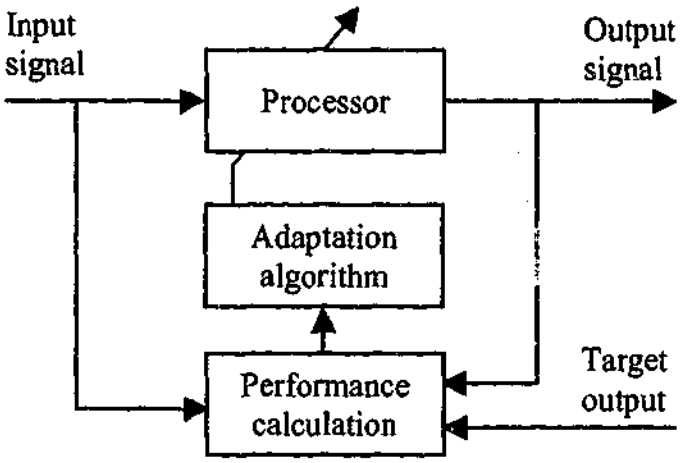


Fig. 5.2: The closed-loop adaptation process.

The main advantages of a closed-loop system over an open-loop system is that it is workable in many applications where no analytic synthesis procedure either exists or is known; for example, where error criteria other than the mean-square are used, where systems are nonlinear or time variable, or where signals are non-stationary. On the other hand, the closed-loop adaptation process may suffer from instability by diverging rather than converging. In spite of this possibility, closed-loop adaptation through performance feedback is regarded as a powerful technique for implementing real-time adaptation.

5.2.1 The FADTS Closed-Loop Adaptation Model

The ACDSDTS algorithm works sequentially on frames of an input video sequence. Although consecutive frames are considered to be highly correlated, the input video signal can be considered time variable or non-stationary from the adaptation point of view. Therefore, a closed-loop adaptation model is presented for the FADTS algorithm, as shown in Fig. 5.3.

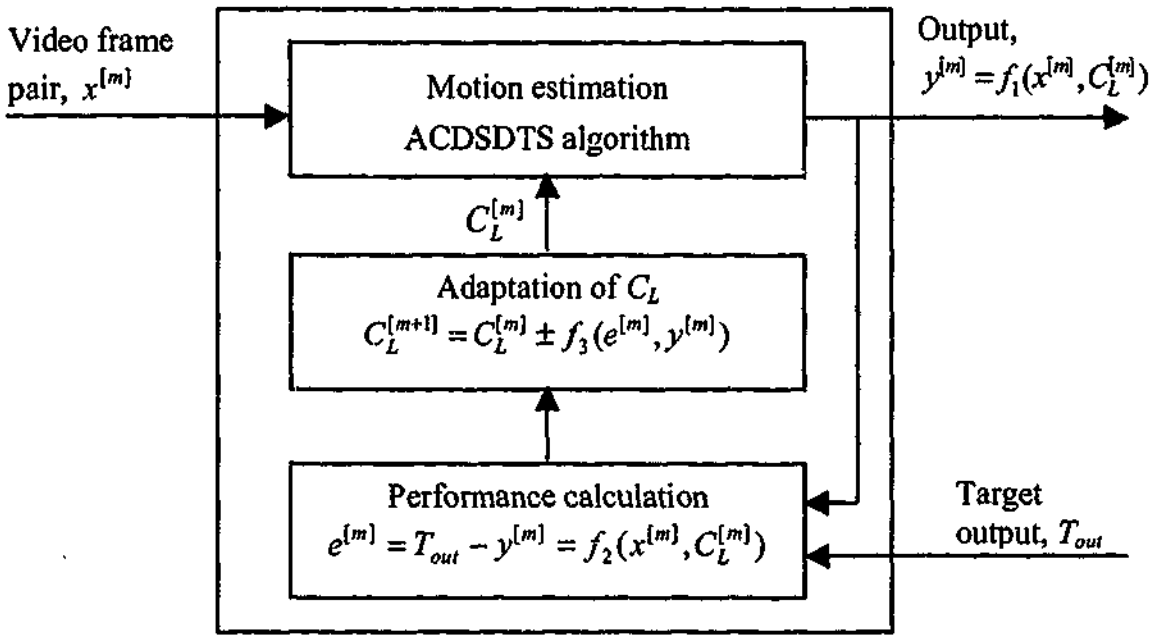


Fig. 5.3: The closed-loop adaptation process for the FADTS algorithm.

The model has the following three modules:-

- Motion estimation—this module calculates motion vectors using the ACDSDTS algorithm. The input of the module at iteration, m, are the video frame pair, $x^{[m]}$, and the control parameter, $C_L^{[m]}$. The output of the model can be either average MSE or average speed in terms of number of search points as selected by the user. The output at iteration m can be expressed as:

$$y^{[m]} = f_1(x^{[m]}, C_L^{[m]}) \tag{5.1}$$

  where f_1 is a monotonically increasing or decreasing function of C_L (under stationary x), if the output is MSE, or a number of search points, respectively.

- Performance calculation—this module calculates the performance of the adaptive system by calculating the error signal as:

$$e^{[m]} = T_{out} - y^{[m]} = f_2(x^{[m]}, C_L^{[m]}) \tag{5.2}$$

  at each iteration m, where:

$$f_2(x^{[m]}, C_L^{[m]}) = T_{out} - f_1(x^{[m]}, C_L^{[m]}). \tag{5.3}$$

The value of e must be minimised as the adaptation process progresses.

- Adaptation of C_L—this module updates the value of C_L for the next iteration, m+1, as:

$$C_L^{[m+1]} = C_L^{[m]} + f_3(e^{[m]}, y^{[m]}) \tag{5.4a}$$

if the output is MSE or as:

$$C_L^{[m+1]} = C_L^{[m]} - f_3(e^{[m]}, y^{[m]}) \tag{5.4b}$$

if the output is a number of search points, where f_3 can be any linear or non-linear function.

The performance of an adaptive system largely depends on how the function $f_3(e, y)$ is defined. A few gradient search algorithms exist which can adapt a system in searching for the optimal parameter to minimise error signal in (5.2). Among them, *Newton's method*, the *Steepest Descent method*, the *Least Mean Square* (LMS) algorithm, and the *Recursive Least Square* (RLS) algorithm are the most well-known. The suitability of these algorithms for updating the control parameter, C_L, in Fig 5.3 is discussed below.

Newton's Method

In this method [106], $f_3(e^{[m]}, y^{[m]})$ in (5.4a) and (5.4b) is defined as:

$$f_3(e^{[m]}, y^{[m]}) = -\left.\frac{f_2(x, C_L)}{\frac{d}{dC_L} f_2(x, C_L)}\right|_{C_L = C_L^{[m]}} \tag{5.5}$$

to find the zeros of function $f_2(x, C_L)$. Obviously, Newton's method is not applicable to the FADTS algorithm as $f_2(x, C_L)$ is unknown.

The Steepest Descent Method

The principle of the steepest descent method [105] is to adjust system parameters in the direction of the gradient at each step, thereby minimizing the function for error surface. In this method, $f_3(e^{[m]}, y^{[m]})$ in (5.4a) and (5.4b) is defined as:

$$f_3(e^{[m]}, y^{[m]}) = -\mu \left.\frac{d}{dC_L} f_2(x, C_L)\right|_{C_L = C_L^{[m]}} \tag{5.6}$$

where μ is a constant that regulates the step size. Like Newton's method, the steepest descent method is also not applicable to the FADTS algorithm as $f_2(x, C_L)$ is unknown.

The LMS Algorithm

As indicated earlier, gradient estimation by derivation of the performance measurement function is not possible when the function is unknown. The most well-known method, the *Least Mean Square* (LMS) adaptive algorithm [105], overcomes such a situation by approximating the gradient based on a single input and output example *taken in isolation*. In this case, for each new input-output, an independent estimate of gradient is performed. In the LMS algorithm, $f_3(e^{[m]}, y^{[m]})$ in (5.4a) and (5.4b) is defined as:

$$f_3(e^{[m]}, y^{[m]}) = \lambda e^{[m]} y^{[m]} \tag{5.7}$$

where λ represents the correction factor or gain factor or step size.

The LMS algorithm is the most popular method for its computational simplicity, robustness, and relatively easy implementation for on-line estimation of time-varying system parameters. A number of variants on the LMS theme have been conceived in order to ratify potential problems of the original LMS algorithm such as the need to guess the best value of λ, slow convergence, and numerical instability. Some of these variants are discussed as follows:

The Block LMS Algorithm

One popular LMS variant is called the *Block* LMS (BLMS) algorithm [107], also known as the *Fast* LMS (FLMS) algorithm, which reduces computational cost by not performing the actual correction for every input. Instead, an averaged estimate of the gradient is computed. For a block of length, K, in which the input signal can be considered stable, the standard LMS algorithm will perform C_L correction in all the K iterations while the BLMS algorithm performs C_L correction only in the first of these. The value of C_L is, therefore, updated for iteration $m+K$ as:

$$C_L^{[m+K]} = C_L^{[m]} + \lambda e^{[m]} \frac{1}{K} \sum_{i=0}^{K-1} y^{[m+i]} \tag{5.8a}$$

if the output is MSE or as:

$$C_L^{[m+K]} = C_L^{[m]} - \lambda e^{[m]} \frac{1}{K} \sum_{i=0}^{K-1} y^{[m+i]} \tag{5.8b}$$

if the output is the number of search points.

The Normalized LMS Algorithm

The *Normalized* LMS (NLMS) algorithm [107-109] replaces the step size, λ, in the original LMS algorithm with μ/E_y where μ is the normalized step size and E_y is the output signal's energy or power.

The NLMS algorithm has two distinct advantages over the original LMS algorithm:-

- Has a potentially faster convergence speed [107, 109];
- Always converges when $0 < \mu < 2$ [107, 107].

The Normalized BLMS Algorithm

The benefits of both the BLMS and NLMS variants can be combined in the *Normalized* BLMS (NBLMS) algorithm [110] where the threshold control parameter is updated as:

$$C_L^{[m+K]} = C_L^{[m]} + \mu e^{[m]} \frac{\frac{1}{K}\sum_{i=0}^{K-1} y^{[m+i]}}{E_y} \tag{5.9a}$$

if the output is MSE or as:

$$C_L^{[m+K]} = C_L^{[m]} - \mu e^{[m]} \frac{\frac{1}{K}\sum_{i=0}^{K-1} y^{[m+i]}}{E_y} \tag{5.9b}$$

if the output is the number of search points, where $E_y = \sum_{i=0}^{K-1} \left(y^{[m+i]}\right)^2$.

The RLS Algorithm

All the adaptive algorithms discussed so far are non-recursive in nature. The *Recursive Least Square* (RLS) algorithm [107, 111] is a recursive implementation of the minimisation of the least square error theme. In this algorithm, output y is a function of not only the current input and the adaptive control parameter, C_L, but also of some of the previous outputs such as the following:

$$y^{[m]} = f\left(x^{[m]}, C_L^{[m]}, y^{[m-1]}, y^{[m-2]}, \ldots\right) \tag{5.10}$$

Though the RLS algorithm converges faster than the LMS algorithm, each iteration of this method is more complex than the previous one. Therefore, this technique is not practical for real-time adaptation applications [107].

Based on the above discussion on the various adaptation techniques, the NBLMS algorithm can be considered as the best option for automatically adjusting the control parameter, C_L, in order to achieve a target average MSE or average number of search points while coding a video sequence, where this sequence can be considered as a time varying non-stationary input to the adaptation system.

5.2.2 The Formal FADTS Algorithm

The FADTS algorithm utilising the NBLMS algorithm for adapting the control parameter, C_L, in order to achieve a target output, i.e., predicted image quality, in terms of average MSE is now outlined as follows. The algorithm applies the ACDSDTS algorithm (Section 4.3.3) on a block of K frames of the video sequence using the same C_L, value for motion estimation. The C_L is initialised to 0 for the first block of K frames and the value of C_L is then updated for the next block of K frames by (5.9a) using the average output MSE and the total energy of all output MSE of the motion estimation carried out so far on the current block of K frames. Fig. 5.4 presents the complete FADTS algorithm.

- **Precondition:**
  Input video sequence of N frames, target output T_{out}, block length K, normalised step size μ.
- **Initialization:**
  $C_L^{[1]} = C_L^{[2]} = \cdots = C_L^{[K]} = 0$
- **Body:**
  For $m = 1, 1+K, \cdots, 1+\left(\left\lfloor \frac{N-1}{K} \right\rfloor\right)K$
  - ***K*-Block Motion estimations:**
    $S = V = 0$
    For $i = 0, 1, \cdots, \min(K-1, N-m-1)$
    Calculate MV between frame pair $x^{[m+i]}$, consisting of frames $m+i$ and $m+i+1$, using ACDSDTS($C_L^{[m+i]}$) algorithm and
    let $y^{[m+i]}$ be the average MSE of the MV estimation.
    $S = S + y^{[m+i]}$
    $V = V + \left(y^{[m+i]}\right)^2$
  - **Performance measurement:**
    $e^{[m]} = T_{out} - \frac{S}{K}$
  - **Adaptation of C_L:**
    $C_L^{[m+K]} = C_L^{[m+K+1]} = \cdots = C_L^{[m+K+\min(K-1, N-m-1)]} = C_L^{[m]} + \mu e^{[m]} \frac{S}{KV}$
- **Postcondition:**
  Motion vector with target average MSE.

Fig. 5.4: The FADTS algorithm.

The flexibility of the FADTS algorithm is illustrated by the fact that it is capable of adapting the motion estimation in order to achieve a target prediction image quality in terms of the average MSE output by trading off search speed in terms of average number of search points. However, the same algorithm can easily be transformed for adapting the motion

estimation with a target search speed in terms of average number of search points, while trading off prediction image quality by incorporating the following minimal changes:

- The number of average search points is used as the target T_{out},
- $y^{[t]}$ is the average number of search points for motion estimation between frames t and $t+1$,
- The updating factor in the adaptation of C_L is negative (5.9b) instead of positive (5.9a),
- Postcondition: Motion vector with target average search points.

5.3 Shot Detection

A shot is a sequence of frames generated during a continuous operation and it represents a continuous action in time and space [112]. As different shots contain different visual content and motions, the motion calculated between two successive frames in each different shot can produce a quite unrealistic prediction error and motion vector. Whenever a shot change occurs, the first frame is always considered as the reference frame and is always intracoded (without motion compensation). It has also been shown in the previous chapters that the different values of C_L give different performance for different types of motion sequence. For these reasons, estimating motion with the appropriate C_L, requires the incorporation of shot detection, especially camera breaks in the FADTS algorithm, so that C_L can be reinitialised as the shot changes.

Shots can be joined together in either an abrupt transition mode, in which two shots are simply concatenated, or through gradual transitions, in which additional frames may be introduced using editing operations such as dissolve, fade-in, fade-out, and wipe. A number of algorithms for shot detection in both the uncompressed and the compressed domains have been reported in the literature. In general, automatic shot boundary detection techniques are classified into the following categories: pixel based, statistics based, transformed based, histogram based, and motion vectors based [113, 114]. In pixel-based methods, pixel-wise intensity difference is considered as the indicator for shot boundary detection. Boreczky and Rowe [113], Zhang *et al.* [115], Otsuji and Tonomura [116], and Hampapur *et al.* [117] compute the absolute sum of pixel-by-pixel inter-frame difference and later compare it to a selected threshold. If the difference is more than the threshold value, a shot boundary is declared. It is a very simple method, but the drawback associated with it is that it is very sensitive to noise, and camera and object motion. It is also difficult to adjust the threshold value manually. Shahraray [118], Kasturi and Jain [119], and Zhang *et al.* [120] propose different shot boundary detection methods based on content statistics such as mean, standard deviation, and the likelihood ratio.

These methods are reasonably tolerant of noise, though they are slow due to the complexity of the statistical formulas used and they generate many false positives (wrong boundaries detected as correct ones). In order to effectively protect against camera operation and object motion, an option is to select a motion-independent metric, like overall intensity histogram difference. Histograms are the most common method used to detect shot boundaries. In the simplest histogram method, the gray level or color histogram is computed and compared bin-wise difference with a threshold. If this bin-wise difference is above a threshold, a shot boundary is assumed. Ueda *et al.* [121] and Nagasaka and Tanaka [122] use the color histogram change rate to find shot boundaries. This is the most common method and more robust to noise and object motion. According to Boreczky and Rowe [113], the histogram methods were a good trade-off between accuracy and speed. An alternative to all these algorithms is to work with derived parameters directly extracted from the compressed sequence. Arman *et al.* [123, 124] and Liu and Zick [125] use differences in DCT coefficients of JPEG [126] compressed frames to detect shot boundaries as their measure of frame similarity, thus avoiding the need to decompress frames. However, this DCT based technique generates false positives where it increases the speed. Zhang *et al.* [115], Ueda *et al.* [121], and Deng and Manjunath [127] use motion vectors in MPEG video to detect whether or not a shot change had occurred. Motion discontinuity will occur if there is any sudden change between two consecutive frames. This results in a significant drop of forward motion prediction coded macro blocks and can be easily detected by setting a threshold.

From the above discussion, it can be clearly seen that different shot detection methods work best in different situations. A histogram comparison should be less sensitive to object motion than the DCT difference comparison algorithm, since it ignores the spatial changes in a frame. But there maybe cases in which two images have similar histograms but completely different content. Again, a histogram comparison may not be robust against lighting change. Therefore, if the different features are combined appropriately, a more desirable result can be expected. As a method of combining features, *Artificial Neural Network* (ANN) has been widely used and has been successful in various applications. Based on ANN, an integrated technique for abrupt shot detection will be presented in the next section.

Neural networks are computer algorithms inspired by the way information is processed in the nervous system [128]. An important difference between neural networks and other Artificial Intelligence techniques is their ability to learn. The network *learns* by adjusting the interconnections between layers. When the network is adequately trained, it is able to generalize relevant output for a set of input data. A valuable property of neural networks is that of generalization, whereby a trained neural network is able to provide correct matching in the form

of output data for a set of previously unseen input data. Learning typically occurs by example through training, where the training algorithm iteratively adjusts the connection weights (synapses). Backpropagation for example, is one of the most famous training algorithms for multilayer perceptrons. A comprehensive description of this technique can be found in Zurada [128].

5.3.1 Proposed Integrated Shot Detection Technique

In regard to simplicity of implementation, an integrated method combining the different features discussed above, with ANN used for camera break detection, is proposed for non-real-time FADTS implementation. An intensity histogram, DCT, and motion vector differences are considered as the input of the proposed algorithm. For histogram difference, a 256 level gray scale histogram over the entire frame is calculated and then the sum of the absolute bin-wise histogram difference is normalized. The DCT coefficient difference method closely resembles the algorithm described by Arman *et al.* in [124]. As the DC coefficient represents the average intensity of the block, only the DC component of each block (8×8 pixels) is considered in reducing the computational cost. The absolute sum of the difference of the DC values of each block is normalized by the total number of blocks of a frame, and these are concatenated to produce a vector. For motion vector difference, the magnitude of each block-motion (16×16 pixels), obtained using the DTS algorithm (Chapter 3), is calculated and then normalized for each pair of frame.

A structure of the adopted neural network is shown in Fig. 5.5. The feed forward neural network has an input layer of three neurons that correspond to the features of histogram difference, DC coefficient difference, and motion vector difference, two hidden layers (selected empirically), and an output layer of two neurons that correspond to the shot boundary and continuous frame respectively.

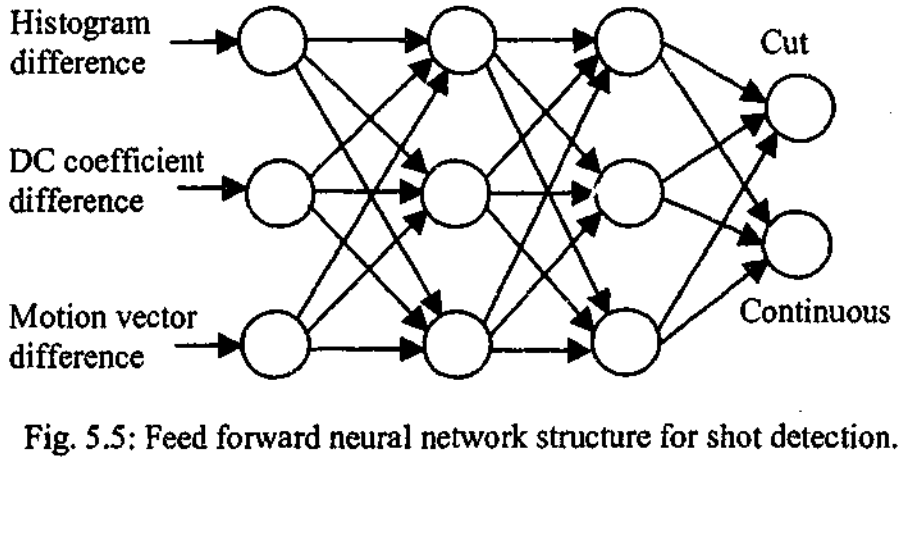


Fig. 5.5: Feed forward neural network structure for shot detection.

Although this integrated technique improves both the *recall* and *precision* of detecting shot changes compared to any of the underlying three individual methods, as is evident in Table 5.2, the computational complexity of calculating three different input features to the neural network is not suitable for applying this combined method in motion estimation for real-time video coding applications. Therefore, the FADTS algorithm can only apply this ANN technique in detecting shot changes for non-real-time motion estimation.

In order to support real-time motion estimation for video coding, the FADTS algorithm employs a simpler, yet elegant technique to detect shot changes by utilising the abrupt change in the error energy, BDM, over a range of threshold values, as an approximated cue to possible shot changes. This technique has been applied to a number of video sequences with intermediate shot changes and no unsatisfactory adaptation of C_L has been encountered. As an example, in Fig. E.1, the shot change between Frames #89 and #90 of the *Table Tennis* sequence is effectively detected by this simple BDM thresholding method.

5.4 Initialisation of C_L

Generally, adaptive algorithms start by setting the initial weight vector (in this case, the value of C_L) to zero. Although in the case of a large number of iteration cycles its impact may be negligible, the performance of adaptive algorithms with relatively fewer iteration cycles depends heavily on the initial value of its weight. Once shot detection is incorporated in the FDATS algorithm, the number of iterations based on an initialisation of C_L depends on the number of frames in each shot, which again depends on the visual content and editorial decisions. However, after studying a large number of standard and non-standard video sequences, it can be fairly concluded that the average number of frames in a shot is not large enough to consider it as nullifying the impact of initialising C_L to zero. Thus, the choice of the initial value of C_L impacts significantly on the performance of the FADTS algorithm for motion estimation. Based on empirical data, the initial value of C_L, i.e., $C_{L_{ini}}$, has been determined for quality and speed in the following two sections.

5.4.1 C_L Initialisation for Quality Adaptation

Fig. 5.6 shows the average search speed and prediction error characteristics of different video sequences with a range of values for the threshold control parameter, C_L. From (3.8), the upper bound for C_L with maximum displacement d is $< \frac{2^b}{d}$. For $d = 7$, this bound is ≈ 36 for an 8 bit gray level image, as indicated in Fig 5.6. The experimental results revealed that above a certain limit, $C_L > 25$, the speed variation was insignificant, so that the upper limit of the threshold

control parameter, $C_{L_{max}}$, can be defined as 25 instead of 36. Similarly, though the minimum value of $C_L = 0$ (FS case) in the ACDSDTS algorithm, experimental results also showed in Chapters 3 and 4 that $C_L \leq 2$ provided almost the same prediction quality as the FS algorithm. Therefore, the lower limit of the threshold control parameter $C_{L_{min}} = 2$ is defined.

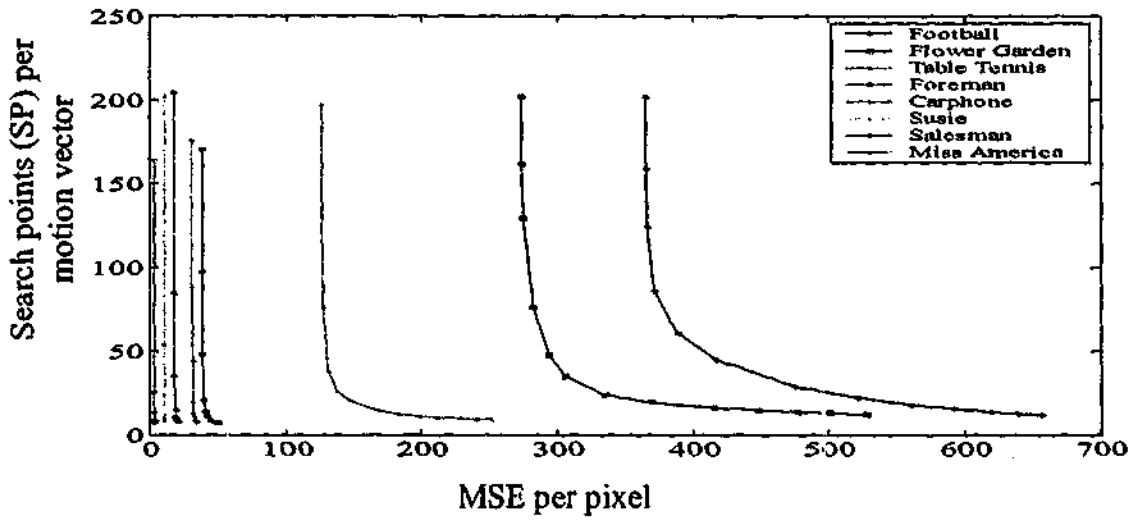


Fig. 5 6: Error-speed characteristics of different video sequences with different values of C_L (1-36).

Fig. 5.7 indicates that the prediction error (quality) variation in terms of the average MSE per pixel using different values of C_L is significant for all motion sequences such as *Flower Garden, Football*, and *Table Tennis*. It is also shown that although prediction error variation with different values of C_L is not exactly linear, it can be approximated as so. Based on this premise, the initial $C_{L_{ini}}$ for a particular sequence is automatically computed from information in the first few frames of the sequence as follows:

- Compute the minimum prediction error $\left(MSE_{C_{L_{min}}}\right)$ between the frame pair #1 and #2 using the $C_{L_{min}}$.
- Compute the maximum prediction error $\left(MSE_{C_{L_{max}}}\right)$ between the frame pair #2 and #3 using the $C_{L_{max}}$.
- Compute the initial value of $C_{L_{ini}}$ for a particular scene in a video sequence as: -

$$C_{L_{ini}} = \left(\frac{T_{out(MSE)} - MSE_{C_{L_{min}}}}{MSE_{C_{L_{max}}} - MSE_{C_{L_{min}}}} \times \left(C_{L_{max}} - C_{L_{min}}\right) + C_{L_{min}} \right) \tag{5.11}$$

where $T_{out(MSE)}$ is the target prediction quality (in this instance, the average MSE). This value of C_L is used for the first K frames of an input video sequence starting from Frame 4.

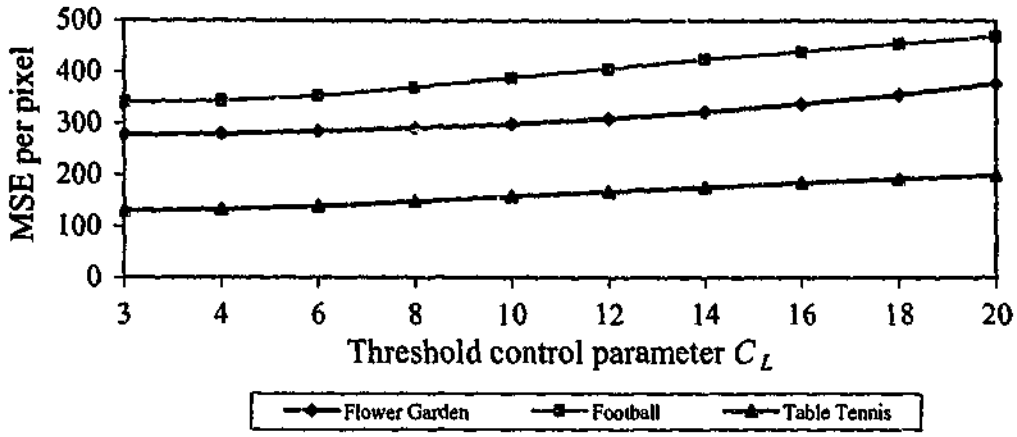


Fig. 5.7: Average MSE characteristics of different video sequences with different values of C_L (3-20).

5.4.2 C_L Initialisation for Search Point Adaptation

Fig. 5.8 shows the computational cost in terms of the average number of search points per motion vectors for some standard high motion and low motion video sequences with values of C_L from 1 to 20.

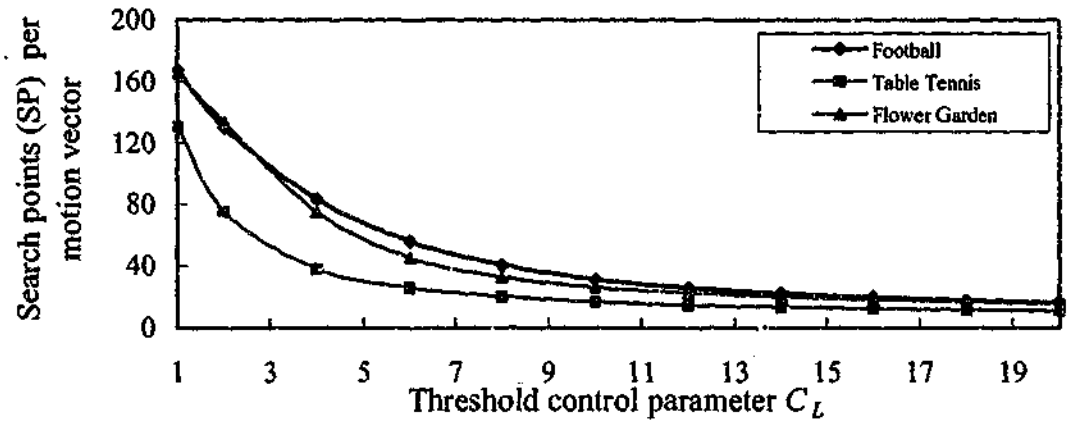


Fig. 5.8: Average speed characteristics of some standard video sequences with different values of C_L.

If a logarithmic scale is used instead of a linear scale, the characteristic curve can be converted into a linear approximation as shown in Fig. 5.9. Using the same procedure described in Section 5.4.1, the initial value of the threshold control parameter, C_L, can be calculated as:

$$C_{L_{ini}} = \left(\frac{SP_{C_{L_{min}}} - T_{out(SP)}}{SP_{C_{L_{min}}} - SP_{C_{L_{max}}}} \times \left(C_{L_{max}} - C_{L_{min}} \right) + C_{L_{min}} \right) \tag{5.12}$$

where $SP_{C_{L_{max}}}$ and $SP_{C_{L_{min}}}$ are the maximum and minimum speed obtained for $C_{L_{max}}$ and $C_{L_{min}}$ (defined in the previous section) respectively, and $T_{out(SP)}$ is the target speed (average number of search points per motion vector).

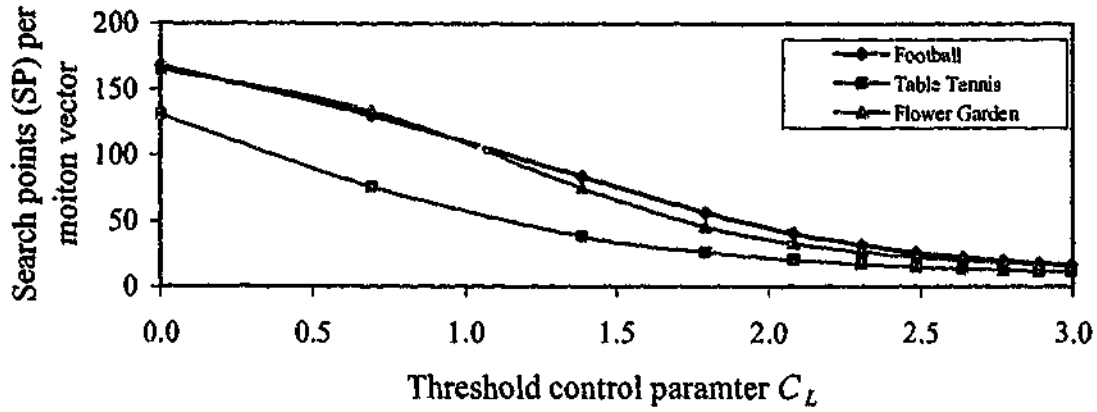


Fig. 5.9: Speed characteristics of three standard video sequences with different values of C_L.

5.5 Study of Different Parameters of the FADTS Model

From (5.9), it is shown that the performance of the proposed adaptive model in Fig. 5.3 also depends on the values of the block length K and step size μ. Each of these is now explored.

Block Length K

Figs. 5.10 and 5.11 examine the impact of the value of K on adapting the C_L parameter in achieving target level quality and search speed for the *Football*, *Flower Garden*, and *Salesman* video sequences. Others parameters such as $C_{L_{int}}$ and μ, are considered as constant. For example, to analyse the effect of block length of K on the predicted image quality and processing speed performance of the FADTS algorithm, average 370, 300, and 15 MSE and 32, 27, and 8 search points are considered as the target error and speed for the *Football*, *Flower Garden* and *Salesman* video sequences respectively. Figs 5.10 and 5.11 show that calculated average MSE and search points using different values (1, 2, 4, 6, 8, 10, 12, and 14) of K to follow the above-mentioned targets for 100 frames of each sequence. It is shown that the influence of different values of K in satisfying the target MSE and search points (SP) is insignificant. The reason for this is that the picture content does not change too frequently.

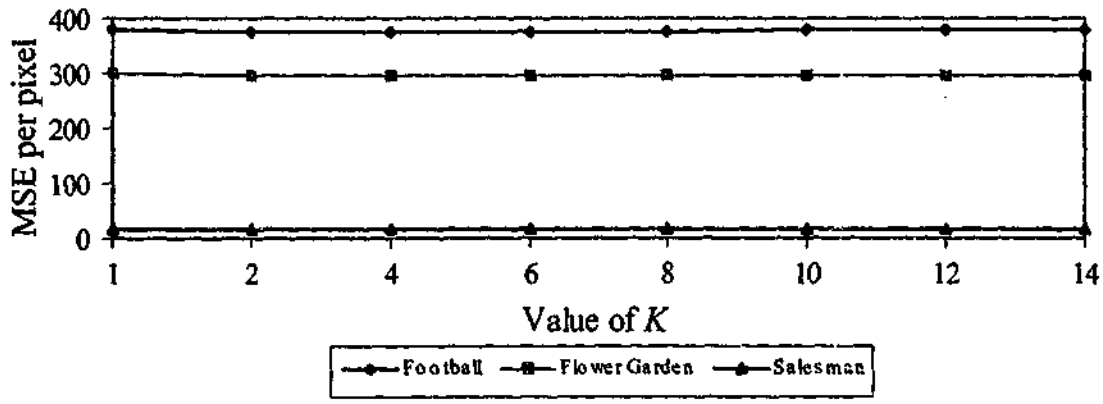


Fig. 5.10: MSE characteristics of different video sequences with different values of K.

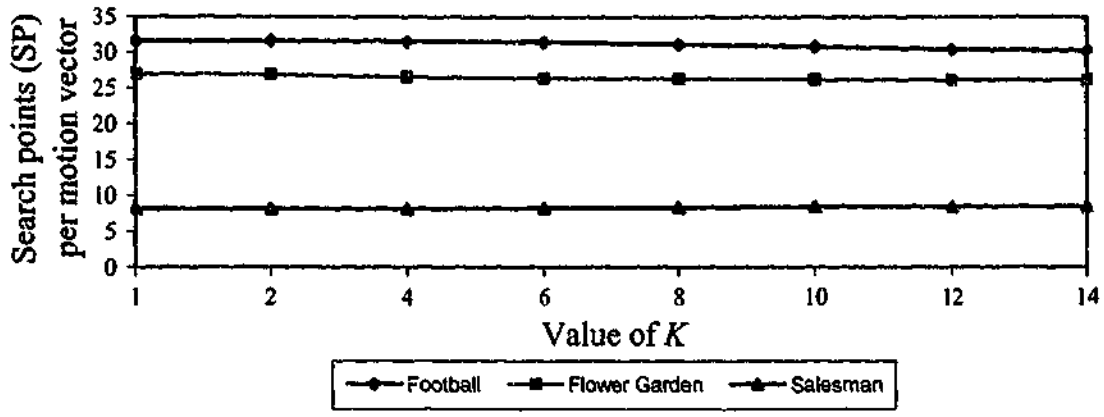


Fig. 5.11: Speed characteristics of different video sequences with different values of K.

Table 5.1 also shows the performance of the FADTS algorithm with different values of K with average 390, 280, and 16 MSE considered as the target error for the *Football*, *Flower Garden* and *Salesman* video sequences respectively. It can also be seen that the FADTS algorithm obtained an output average MSE closer to the target MSE with a comparatively fewer number of search points, when the block length $K = 4$ for all cases. Although a lower value of K also performed almost similar performance in satisfying the targets, according to (5.9), it increases the overhead computational cost for the adaptation process. Conversely, a higher value of K can be considered in order to reduce the overhead cost. As stated in Section 5.2, the block length of K in the BNLMS algorithm cannot be too high if it is assumed that the content of a video sequence may be unstable. Based on this assumption, and the experimental results, the value of $K = 4$ is defined for this thesis for all experiments.

Values of K	*Football*		*Flower Garden*		*Salesman*	
	MSE	SP	MSE	SP	MSE	SP
2	392.0	37.5	280.99	46.71	15.42	9.37
4	391.0	37.8	281.11	44.32	15.42	9.29
6	392.8	39.1	282.86	43.87	15.42	9.32
8	391.1	40.6	283.51	43.50	15.42	9.31
10	392.2	41.7	283.89	44.32	15.42	9.31

Table 5.1: Performance comparison of FADTS algorithm with different values of K.

Step Size μ

With respect to this parameter, Meghriche [129] highlights that there is no universal solution in finding the optimal value of μ. Section 5.2 indicates that the NLMS algorithm considers a step size range of $(0 < \mu < 2)$ for signal processing applications. The lower the value of μ, the slower the convergence rate, while a high step size can lead to system instability. The

application of μ in this thesis is not constant, as the variable step size depends on the error signal e. If the error becomes large, then a greater step size is considered for the next iteration to speedily move towards the target level. If error is low, the step size will be smaller in order to follow the target line. A high value is always chosen for the step size for the first few video frames to enable a quick adaptation towards the target level. Choosing $\mu \geq 2$ can lead to instability; however, in this system, the application of the ceiling $C_{L_{max}} \leq 25$, enforces a dampening effect which avoids such instability. For all the video sequences tested for the FADTS algorithm, no instability was encountered.

5.6 Computational Complexity Analysis of the FADTS Algorithm

The computational complexity of a motion estimation algorithm is usually expressed in terms of either the number of search points or operations that the algorithm requires to calculate the MV. Since the main focus in this thesis is upon the computational cost incurred for the adaptive processing, the latter is used as the complexity measure.

In Chapters 3 and 4 it has been shown that the range of computational complexity based on user-defined levels is bounded between $\varsigma(9\Omega+1)$ and $\psi+\varsigma(d+8\Omega)$ operations per second with half-pel accuracy motion estimation, and the overhead cost for centre adaptation is $R\varsigma$ per second.

Lemma 1 *Computational overhead of the FADTS algorithm, compared to the ACDSDTS algorithm, is negligible.*

Proof: Assume the block distortion is measured using MAE, which requires 3 basic operations per pixel. If the frame rate $f = 30$ fps, $[N_h, N_v] = [352,240]$, $d = 7$ and $N = 16$, the number of integer arithmetic operations required for the upper and lower complexity bounds are 1.77 billion and 68.4 million per second, respectively, with half-pel accuracy (Section 3.5.5). This contrasts with the total number of operations for (5.9) of only $(3K+5)\times f$ per second. Since, in experiments, $K = 4$, this means a total of only 510 additional operations per second, which is negligible. ■

In summary therefore, the FADTS algorithm consumes minimal additional computational overhead compared to the BDM calculation in ACDSDTS algorithm, while providing significant performance benefits including user-definability of key parameters by employing an adaptive thresholding process.

5.7 Experimental Results

The purpose of this section is to analyse the experimental performance, first, of the integrated

shot detection technique for temporal shot detection and second, the FADTS algorithm for prediction error quality and processing speed adaptation.

5.7.1 Performance Analysis of Proposed Shot Detection Technique

To evaluate the efficiency of the proposed integrated method, an objective measure, *Recall* and *Precision* as in (5.13) are used. *Recall* is the relevant detection rate from all the relevant items in the image database and *precision* represents the correct detection rate.

$$Recall = \frac{C_d}{C_d + M},\ Precision = \frac{C_d}{C_d + F_p} \tag{5.13}$$

Where C_d is the number of correct detections, M is the number of missed items and F_p is the number of *false positives*. So a large *recall* value means that the correct shot boundaries are not missed very much, and a large *precision* value means that relatively few wrong boundaries are declared as a shot boundaries.

These two (*recall* and *precision*) are interdependent and closely related to threshold values. The threshold must be assigned so that it can tolerate variations in individual frames while still ensuring a desired level of performance. In order to achieve high accuracy in video partitioning, an appropriate threshold must be found. As, in general, it is a really difficult process to find an appropriate threshold value manually, threshold selection remains a significant problem for traditional methods. Thus, for automatic selection of the threshold, some researchers [115] have used the following relation, $Threshold = \delta + \alpha\beta$, where δ and β are the mean and the standard deviation of the frame-to-frame differences, respectively, and α is constant. This is, however, very much application- dependent, thus in this research, the threshold values have been selected according to experimental observation.

For the experiment, different video clips such as movies, animation, and sports containing approximately 5000 frames in total were considered. For training the neurocomputing model, we used 80% datasets and the remaining 20% datasets were used for testing purpose. After a clinical analysis, it was found that the neural network was giving good generalization performance when 2 hidden layers, with 30 neurons each, were considered. Table 5.2 shows the comparative results of histogram distance, DCT coefficient distance, motion vector distance, and proposed integrated method for shot detection. From the table, it is shown that the *recall* and *precision* percentage with integrated method is 11%, 4%, and 4% and 9%, 3%, and 2% higher compared to that of the histogram, DCT, and motion distance method, respectively. It demonstrates that the integrated technique outperformed all other three existing techniques for camera break detection.

Different methods	*Recall*	*Precision*
Histograms distance method	86%	74%
DCT coefficient distance method	93%	90%
Motion vector difference method	93%	91%
Proposed integrated method	97%	93%

Table 5.2: Performance comparison of histogram, DCT coefficient, and motion difference methods with the proposed integrated method.

5.7.2 Performance Analysis of the FADTS Algorithm

The performance of the FADTS algorithm was evaluated using the luminance (Y-component) signal of a number of test video sequences such as *Football*, *Flower Garden*, *Table Tennis*, and *Salesman* (Appendix B). The test results for *Football* and *Flower Garden* are included in this chapter. Some supplementary results for the *Table Tennis* and *Salesman* video sequences are included in Appendix E.

In the experiments, all sequences were uniformly quantised to an 8-bit gray level intensity. The block size dimensions were $N = 16$ and $d = \pm 7$. The MAE measure (2.1) was used as the criterion for locating the best motion vector for each block. The value of $K = 4$ was chosen for the experiments. All the results are shown with half-pel accuracy motion estimation.

To compare the searching efficiency of the FADTS algorithm, the test results of the FS, TSS, and NTSS algorithms have been shown in Table 5.3.

Block-matching algorithms	*Football* sequence (345 frames)			*Flower Garden* sequence (150 frames)		
	MSE	PSNR [dB]	SP	MSE	PSNR [dB]	SP
FS	218.88	24.73	160.05	208.91	24.93	209.73
TSS	240.79	24.31	25.63	242.97	24.28	31.20
NTSS	239.15	24.34	26.9	213.28	24.84	28.98

Table 5.3: Average MSE per pixel and SP per motion vector of the FS, TSS, and NTSS algorithms for the *Football* and *Flower Garden* video sequences.

The performance of the FADTS algorithm was tested and evaluated for quality and speed adaptation as follows:

Quality Adaptation

The performance of the FADTS algorithm for quality adaptation is presented in Tables 5.4 and 5.5 for a number of different target values for the *Football* and *Flower Garden* sequences. From

these tables, it can be seen that the FADTS algorithm achieved all target demands in terms of predicted image quality and processing speed. For example, targets were set to estimate motion with an average 230 MSE or 24.51 dB PSNR image quality for the *Football* sequence and 215 MSE or 24.81dB PSNR image quality for the *Flower Garden* sequence. It is shown that the FADTS algorithm satisfied these demands providing MSE or PSNR very close to targets such as 232.04 MSE or 24.48 dB PSNR for the *Football* sequence, and 214.41 MSE or 24.82 dB PSNR for the *Flower Garden* sequence, with average search points 49.18 and 24.54, respectively. The flexibility of the FADTS algorithm for QoS demand was investigated by setting different targets, for example, a target of average 250 MSE or 24.15 PSNR for the *Football* sequence, and 225 MSE or 24.61 PSNR for the *Flower Garden* sequence. Tables 5.4 and 5.5 show that the FADTS algorithm, again, satisfied these demands by calculating motion with 252.13 MSE or 24.11PSNR for the *Football* sequence, and 222.79 MSE or 24.65 PSNR for the *Flower Garden* sequence, while reducing computational cost almost 3 and 1.6 times respectively compared to previous demand. These settings reveal that the FADTS algorithm is able to reach any bounded target level of quality, with the implicit assumption that the minimum target error obtained by FS is the lower bound. Note that if a target is set so high that the resultant C_L exceeds $C_{L_{max}}$ to achieve the target, the FADTS algorithm will fail. However, defining such a high target is very unlikely, as it will produce an extremely poor quality output.

Target quality		Calculated Quality		Search Point (SP)
MSE	PSNR [dB]	MSE	PSNR [dB]	
230	24.51	232.04	24.48	49.18
235	24.42	234.70	24.43	32.25
240	24.32	241.00	24.31	19.87
250	24.15	252.13	24.11	16.56

Table 5.4: Prediction error adaptation for the *Football* video sequence (345 frames).

Target quality		A Quality		Search Points (SP)
MSE	PSNR [dB]	MSE	PSNR [dB]	
210	24.91	212.80	24.85	34.95
215	24.81	214.41	24.82	24.58
220	24.71	218.62	24.73	16.65
225	24.61	222.79	24.65	15.21

Table 5.5: Prediction error adaptation for the *Flower Garden* video sequence (150 frames).

The corresponding adaptive values of control parameter, C_L, for different frames are plotted in Figs. 5.12 and 5.13, where the adaptive nature of the FADTS algorithm is shown for varying content between different frames. It also indicates that the FADTS algorithm automatically computed a different starting value for C_L based on both the content of the video sequence, and the desired target.

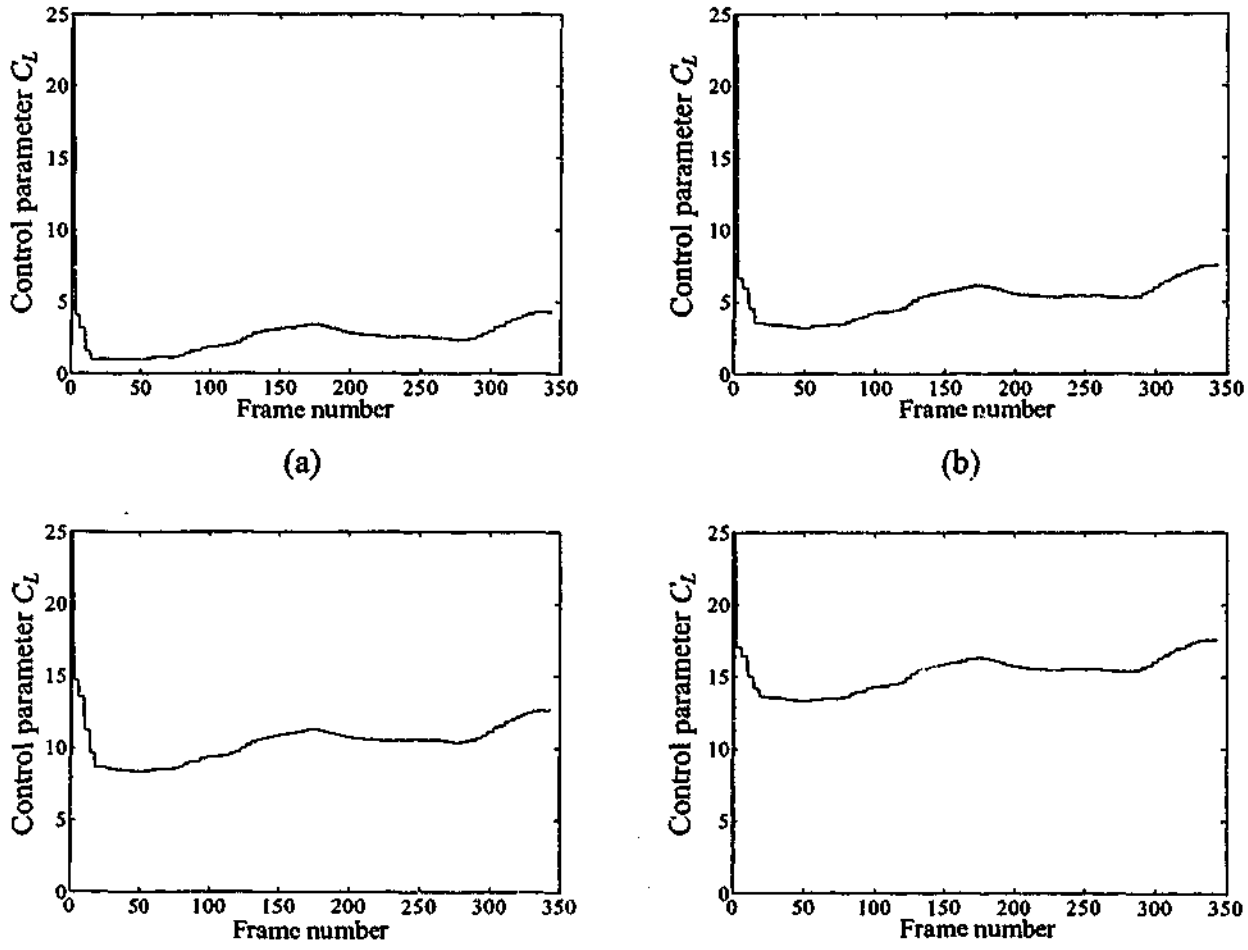


Fig. 5.12: Threshold control parameter adaptation for the *Football* sequence with average (a) 230, (b) 235, (c) 240, and (d) 250 MSE per pixel prediction quality.

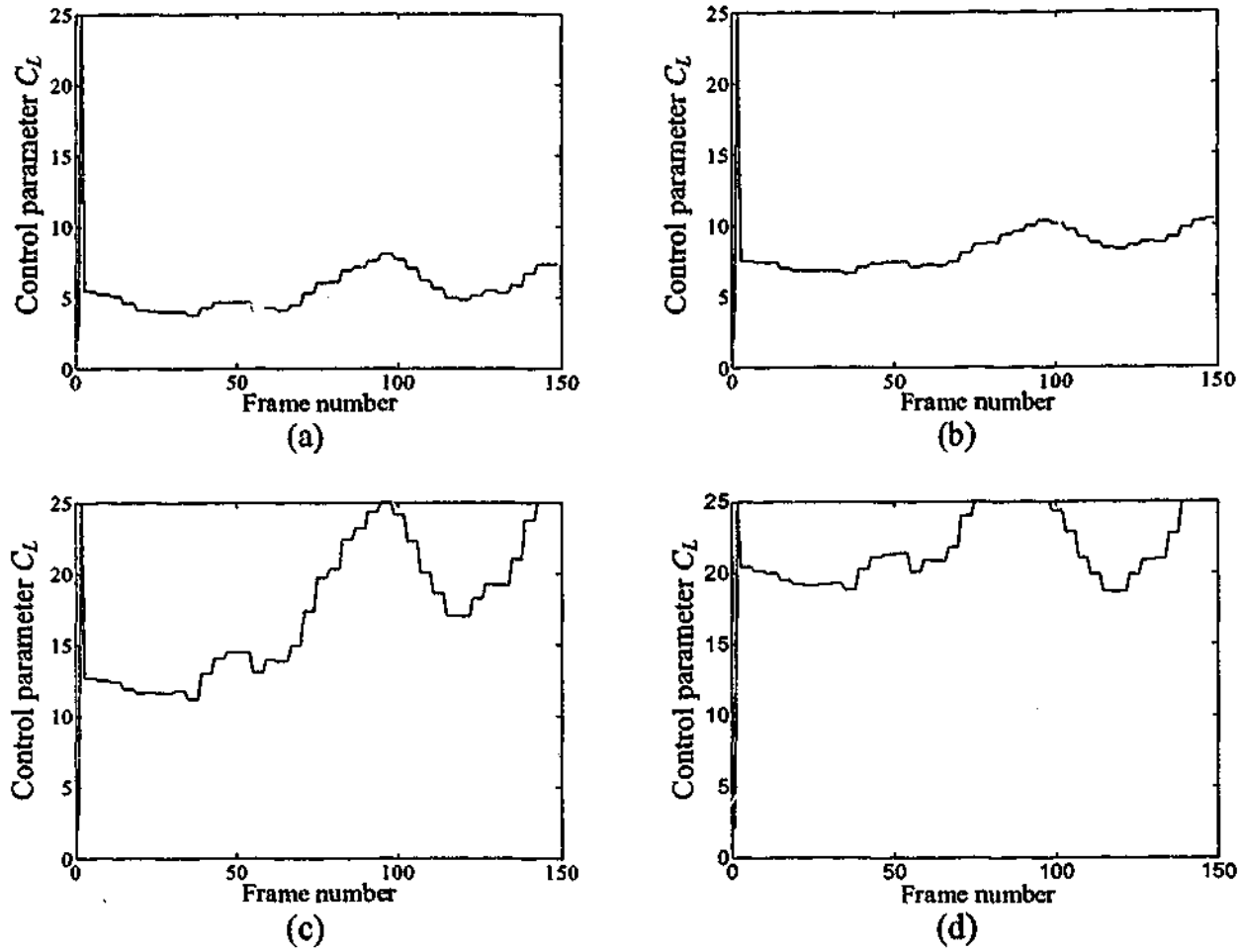


Fig. 5.13: Threshold control parameter adaptation for the *Flower Garden* sequence with average (a) 210, (b) 215, (c) 220, and (d) 225 MSE per pixel prediction quality.

Search Point Adaptation

The performance of the FADTS algorithm for computational scalability in terms of the average number of search points (SP) per motion vector was tested with a number of targets, i.e., average search points considered. Table 5.6 shows some of these targets and the actual values obtained by the FADTS algorithm for the *Football* and *Flower Garden* video sequences.

Football sequence				*Flower Garden* sequence			
Target SP	Actual SP	Actual Error		Target SP	Actual SP	Actual MSE	
		MSE	PSNR [dB]			MSE	PSNR [dB]
20	20.10	243.00	24.27	15	15.52	220.54	24.70
25	24.99	237.82	24.37	20	19.68	216.27	24.78
30	29.89	235.32	24.41	25	24.69	214.85	24.81
40	39.08	230.22	24.51	30	29.68	214.19	24.82

Table 5.6: Speed adaptation for the *Football* and *Flower Garden* video sequences (345, and 150 frames respectively).

Table 5.6 indicates that the FADTS algorithm can satisfy all user demand for different computational complexity in terms of the average number of search points. For example, consider the target search points of average 20 SP, for the *Football* sequence, and 15 SP for the *Flower Garden* sequence, per motion vector. From Table 5.6, it can be observed that the FADTS algorithm satisfied these demands by estimating motion vector with an average 20.10 and 15.52 SP, where the prediction image quality in terms of PSNR was average 24.27 and 24.70 dB for the *Football* and *Flower Garden* sequences, respectively. Another example, with the targets of average 30 SP for the *Football* sequence, and 25 SP for the *Flower Garden* sequence is in Table 5.6, which demonstrates that the FADTS algorithm satisfied the demands of the target estimating motion vector with an average 29.89 and 24.69 SP, with 24.41 and 24.81 dB PSNR, respectively.

The corresponding adaptive values of control parameter, C_L, for different frames are shown in Figs. 5.14 and 5.15 which show the adaptive power of the FADTS algorithm with content variation in different frames. It is also shown that with higher speed i.e., a lower SP, as target, the FADTS algorithm automatically started with a higher initial value of C_L based on the content of the video sequence and the expected target.

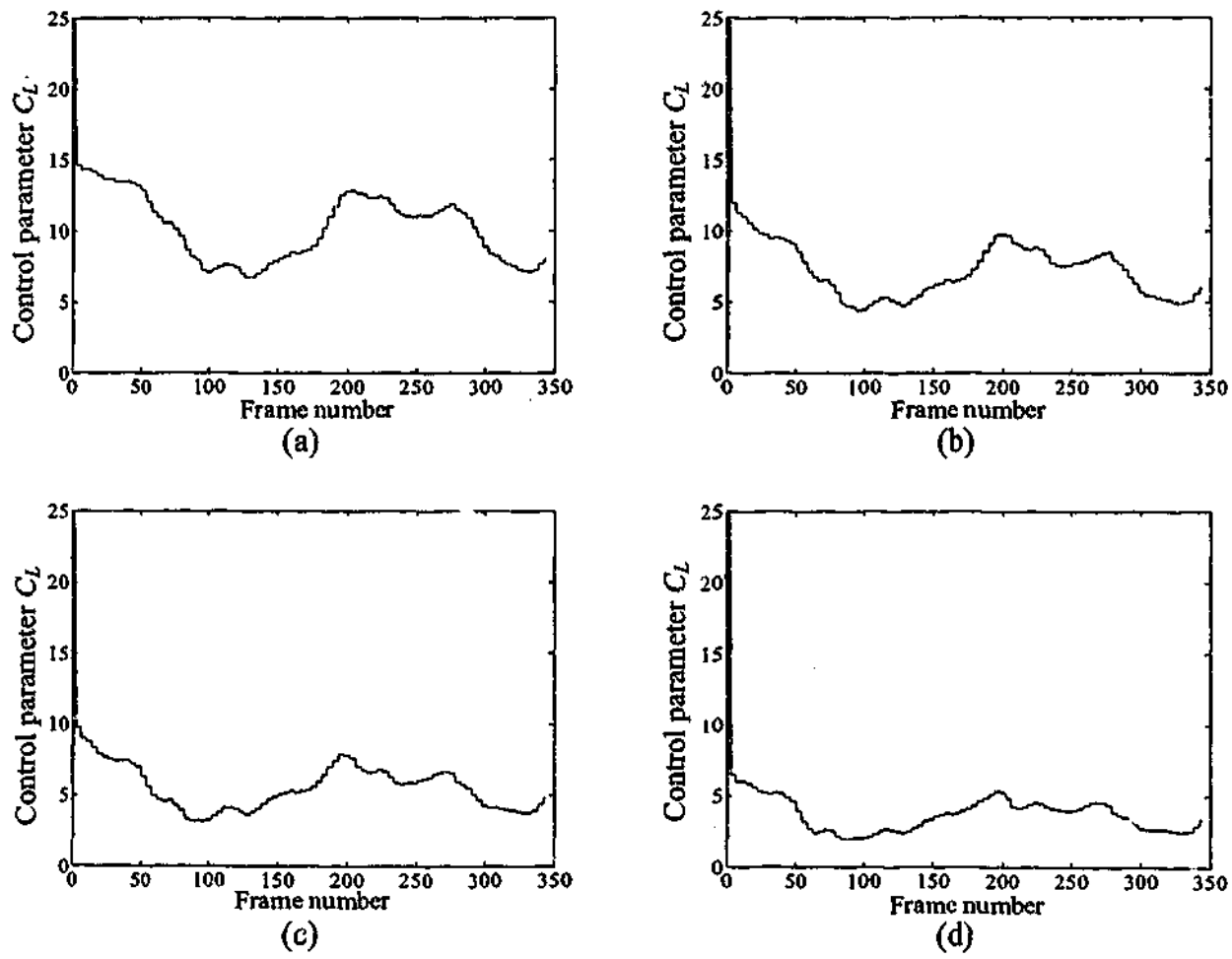


Fig. 5.14: Threshold control parameter adaptation for the *Football* sequence with average (a) 20, (b) 25, (c) 30, and (d) 40 search points per motion vector.

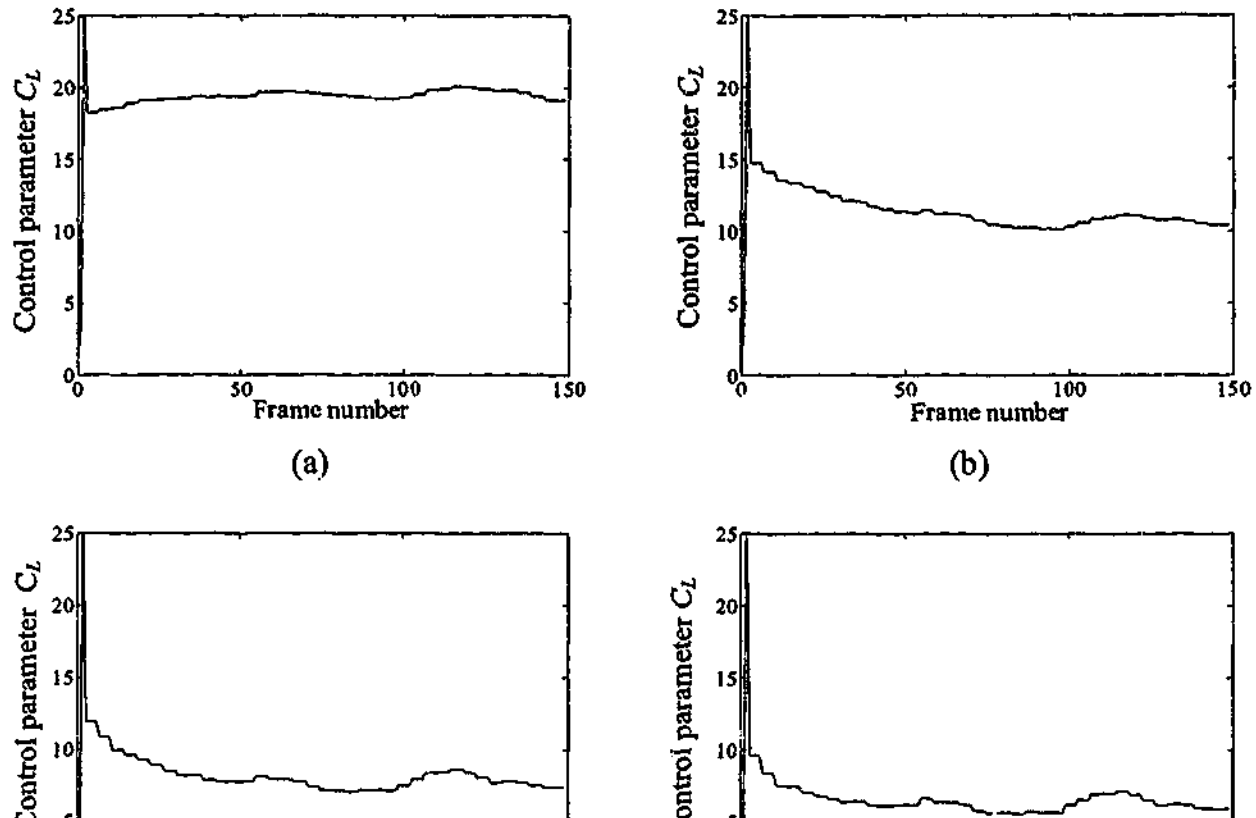


Fig. 5.15: Threshold control parameter adaptation for the *Flower Garden* sequence with average (a) 15, (b) 20, (c) 25, and (d) 30 search points (SP) per motion vector.

From tables 5.3, 5.4, 5.5, and 5.6, it is clear that the FADTS algorithm not only satisfied any user defined targets, but also showed better error performance with computational complexity, similar to the TSS or NTSS algorithm. For the *Football* sequence shown in Table 5.3, the TSS and NSS algorithms have computational complexity in terms of average number of search points, 25.63 and 26.9 respectively, and predicted image quality in terms of PSNR of 24.31 and 24.34 dB, respectively. As shown in Table 5.6 that the FADTS algorithm achieved even better PSNR performance compared to that of the TSS and NTSS algorithms while considering a smaller number of search points (average 25 SP). Tables 5.3 and 5.6 also show that the performance of the FADTS algorithm is very similar to the NTSS and better than that of the TSS algorithm for the *Flower Garden* sequence. This is because the FADTS algorithm adaptively selects the threshold control parameter to limit the search for different frames with different content, whereas a directional fast algorithm, such as TSS, always searches for 25 points irrespective of the content variation. It is shown by certain authors [45, 46, 53, 68, 85], most of the macroblocks in a video sequence are stationary or quasi-stationary in nature. In this

case, the FADTS algorithm stopped searching after using a smaller number of search points with similar error performance.

Although the search efficiency of the FADTS algorithm was found to be similar to the fast TSS and NTSS algorithms for video sequences incorporating all kind of motions, the main strength of the FADTS algorithm lies in its unique performance scalability as shown in Tables 5.4, 5.5, and 5.6. No other existing fast directional algorithm provides such a level of flexibility in trading off predicted image quality and computational complexity, whereas the FADTS algorithm demonstrates considerable flexibility in providing target-driven services, especially in terms of computational complexity.

5.8 Summary

In this chapter a *fully adaptive distance-dependent thresholding search* (FADTS) algorithm has been developed for performance management block-based motion estimation in real-time video coding applications. A key feature of this approach is the progressive adjustment of the required threshold control value via an adaptive process which uses the information from previous frames to achieve specified user demands i.e., prediction quality or processing speed. The performance of the FADTS algorithm has been examined, and proof that it affords a unique feature in being able to trade off between two key model parameters, namely prediction quality and search speed, for the entire range of values of the threshold control parameter, C_L. Experimental results have shown that this novel FADTS algorithm has achieved guaranteed QoS demands. Moreover, the performance scalability, especially complexity scalability, found in this algorithm, represents an effective solution to the overall problem of performance scalability for real-time software-only or low power video coding applications.

The search efficiency of the FADTS algorithm has been compared to the most popular fast algorithms, TSS and NTSS. Experimental results have proved that the FADTS algorithm is not only able to provide QoS but also demonstrates similar, or faster search speed, with similar error performance. Therefore, the FADTS algorithm solves the problem of existing fast directional algorithms in providing different levels of quality of service.

The initial value of the threshold control parameter is an important factor in the adaptation process. In this regard, some adaptation techniques have been formalized based on the contents of the first few frames of each video shot. Consequently, the FADTS algorithm adaptively estimates the initial value of threshold control parameter for each shot or scene. To detect a shot change, an integrated shot detection technique using *Artificial Neural Network* (ANN) and BDM thresholding technique have been presented for non-real-time and real-time applications.

In this research, all the popular existing adaptive algorithms have been studied in order to select the most appropriate one i.e. *Normalized Block Least Mean Square* (NBLMS) adaptive algorithm, for implementing in the FADTS system.

The overhead computational complexity of the proposed FADTS algorithm for motion estimation has also been analysed. It has been shown that although the FADTS algorithm has some overhead complexity in the process of threshold control parameter adaptation, this overhead is negligible compared with BDM calculation in motion estimation.

Chapter 6

Block-based *True* Object Motion Estimation

6.1 Introduction

In Chapter 1, it was identified that although block-based object motion has been used in many different applications, especially video indexing by exploiting object motion, there are limitations in estimating block-based *true* object motion using existing BMAs. To address this issue, the *Distance-dependent Thresholding Search* (DTS) block-based motion estimation algorithm (**Block 1** in Fig. 1.6), in Section 3.6 was subjectively examined, and perceptually exhibited superior performance compared to existing BMAs for *true* object motion capture. As the block-based technique captures both object and camera motion, and also introduces some *false* motion vectors as noise, to re-affirm the superior performance of the DTS algorithm for *true* object motion estimation, it is necessary to remove the *global* motion component and eliminate the *false* motion vectors. In this chapter, a novel filter, called the *Mean Accumulated Thresholded* (MAT) filter (**Block 4** in Fig. 1.6) which eliminates the *false* motion vectors in order to extract the *true* object motion vector for video object representation, is introduced. The experimental results described in this chapter establish that the DTS algorithm (**Block 1** in Fig. 1.6), when combined with the MAT filter, can be a very useful tool for block-based *true* object motion estimation.

To remove camera motion when capturing *true* object motion vectors, a *Modified Iterative-Least-Square Estimation* (MILSE) (**Block 4** in Fig. 1.6) technique is presented, and is used to estimate the *global* motion parameter. The MILSE technique significantly reduces the computational overhead in calculating this parameter compared with the original *Iterative-Least-Square Estimation* (ILSE) technique described by Rath and Makur in [130].

The chapter is organized as follows. Section 6.2 presents some different applications where block-based object motion has recently been used. A review of existing parametric *global* motion estimation techniques is presented in Section 6.3. The well-established pan-zoom *global* motion modeling technique is reviewed and a MILSE technique is proposed for camera parameter estimation. An analysis of the complexity of the MILSE technique is provided.

Section 6.4 details *global* motion cancellation and noise (*false* motion vector) elimination techniques. The novel *Mean Accumulated Thresholded* (MAT) filter for *false* motion elimination is then presented in detail. Experimental results on the performance of the MILSE and ILSE techniques are described in Section 6.5, as well as the performance of the DTS and MAT filter combination in capturing *true* object motion vectors compared to other BMAs. This section also analyses the computational complexity of the MAT filter in detail. Section 6.6 summarises the chapter.

6.2 Importance of Block-based Object Motion

With the rapid expansion of digital broadcasting systems, the Internet, and digital library services, digital video data has become pervasive. As a result, among the many video analysis applications extant, one of the most important applications has been content-based video indexing in order to access video material from the tremendous pool of video information available.

A typical approach to video indexing for browsing and retrieval is the shot-based approach [131, 132] where a raw video stream is first segmented into a sequence of shots. After segmentation, features within each shot such as content, length, and camera operations are used for video indexing purposes. Two approaches for video indexing are distinguished [133]: still image feature (spatial) based techniques, and temporal feature based techniques. In the former, a small set of representative (key frame) frames are selected to represent the visual content of each shot to be stored in the database. The information from each key frame is then represented by low level still-image features such as colour, texture and shape. The major drawback of these still-image feature indexing techniques is that video sequences are treated as still images, so the semantics contained in a sequence are lost [133]. Motion, especially *true* object motion (as a temporal feature) allows the user to specify queries that involve the exact position and trajectories of the objects in a shot, and so can be considered as key feature in video indexing for the sake of searching, browsing, or retrieval.

In previous chapters, it has been shown that though block-based motion estimation techniques are primarily designed for video coding applications, they are increasingly being used in other video analysis applications due to their simplicity and ease of implementation. With MPEG being the worldwide standard for video data compression, and videos being available in MPEG-compressed form, it would be desirable to directly process the compressed video to compute relevant motion features thereby avoiding time-consuming computation of optical flow. As a result, current research in some video analysis applications, especially

content-based video retrieval, seeks to exploit this MPEG coded motion information. Some recent applications are now briefly reviewed.

An object motion descriptor for content-based indexing of MPEG video has been proposed by Kim and Ro in [18]. This technique considers the motion vectors, which are available in the MPEG coded bitstream, as spatially connected four nearest macroblock motion. By clustering the object motion, this technique identifies moving objects in a shot, and the motion for each object is then used for video indexing. In [19], Aghbari *et al.* use MPEG coded block motion vectors to calculate the motion vector features such as motion velocity and angle for a video indexing system for MPEG video retrieval. The object, as well as the camera motion, are calculated from these block motion vectors and a motion index vector is then generated using this information. Another video indexing method based on MPEG coded block motion vectors has been proposed by Heuer in [24], where the motion features (magnitude or/and direction) and motion-based frame segmented features are used for querying the video. In [31], Sahouria and Zakor propose a system to analyse and index surveillance videos based on the block motion of an object which are available in MPEG-1 coded bitstream. Using this information, the trajectories of the moving object are extracted for video indexing and classification. AbouGhazaleh [134] proposes a video indexing system based on the object's motion from MPEG coded bitstream. Based on motion vector similarity in the adjacent block in terms of magnitude and angle, this technique clusters the blocks for a single object. It calculates the absolute motion trajectory of a particular object by detecting and removing the background motion as a camera motion. In [20], Yoneyama *et al.* propose a technique to detect the moving objects by macroblock information such as motion vectors and *Discrete Cosine Transform* (DCT) coefficients. After determining the moving region, the macroblocks regarded as moving regions are grouped using spatial motion similarities with the same moving regions. If the angular difference of motion vectors between the target macroblock and one of the spatially neighbouring eight macroblocks is smaller than the pre-determined threshold value, then these two are regarded as the same object. Zen *et al.* [23] have also proposed an object detection and tracking method using MPEG coded motion vectors and DCT coefficients. This technique identifies the different moving objects by merging the different macroblocks in which the motion vectors are similar in magnitude and direction. A target is selected from the objects and then tracking is carried out by considering the similarity of the average motion vector of each target object between frames. Besides video indexing, Ji and Park [62] propose a video object segmentation technique based on the block motion vector and DCT coefficients.

6.3 Global Motion Estimation

Different camera operations such as fixed, horizontal rotation (panning), vertical rotation (tilting), change of focal length (zooming), rotation around the optical axis (rolling), and horizontal transverse movement (tracking) induce different *global* motions in video sequences. The following two steps are generally involved in estimating the different *global* motions: -

1. *Global* Motion Parameter Modeling
2. Parameter Estimation.

6.3.1 *Global* Motion Parameter Modeling

Various *global* motion modeling schemes have been proposed in the literature, with the most well-known being the three-parameter model corresponding to pan and zoom [135-137], the four-parameter model corresponding to pan, zoom and rotation [130, 138], the six-parameter affine model [139, 140], and the eight-parameter quadratic and perspective model [140].

The different parametric *global* motion models estimate camera motions with varying degrees of complexity. In estimating *global* motion parameters, the pan-zoom model is computationally efficient and gives sufficient accuracy in motion description to represent the *global* motion of a video sequence, especially when the *global* motion is primarily used for compensating for camera motion. Although there are more complex models, the associated benefits are small and computational complexity high. Their use also leads to greater difficulty in parameter estimation, thus incurring higher additional overhead cost. For these reasons, the following pan zoom model for *global* motion representation is considered.

Assume that luminance changes between successive frames are due only to camera motion. If there are I rows and J columns of pixels in a frame, the coordinates of any pixel will be $(i, j), i = 0,1,...,I-1, j = 0,1,....,J-1$ which will be presented by $\mathbf{s}_{ij} = (s_i, s_j)$ with respect to the centre of the frame. The displacement of the pixel (i,j) is represented by $\mathbf{v}_{ij}$. It is assumed that the camera works on the central projection model [141] in which the camera coordinate system lies at the lens of the camera, and the image coordinate system sits at the focal plane. Using these assumptions, the following two models can be established.

6.3.1.1 Camera Pan

Pan is caused by the camera's rotation about either the x-axis (vertical) or the y-axis (horizontal) of the camera coordinate system. It affects both the camera and the image-space coordinates. The pan parameter is normally represented as a two-dimensional vector in which the scalar components refer to the rotation angles θ_x and θ_y about x-axis and y-axis, respectively.

If they are sufficiently small (the camera has low motion between frames), the displacement of the pixel is given as [141]:

$$\mathbf{v}_{ij} = F\begin{bmatrix}\theta_x \\ \theta_y\end{bmatrix} \tag{6.1}$$

where F is the focal length of the camera. In general, small pan causes the entire frame to be uniformly displaced by one vector i.e. $\mathbf{v}_{ij} = \mathbf{p}$, $i = 0,1,...,I-1$, $j = 0,1,...,J-1$,

where $\mathbf{p} \equiv \begin{bmatrix}p_x & p_y\end{bmatrix}^t$ is a constant vector and p_x and p_y are called the pan parameters. Hence, this model includes not only the slow pan of the camera, but also the camera translation along a plane parallel to the image plane. If the motion is due to the slow pan of the camera, then:

$$\begin{aligned} p_x &= F\theta_y \\ p_y &= F\theta_x \end{aligned} \tag{6.2}$$

Note that if θ_x and θ_y are not small enough, then the resulting motion is not constant [141], and so cannot be modeled using the pan model.

6.3.1.2 Camera Zoom

Zoom is caused by a change in the camera's focal length. It changes only the image-space coordinates while the camera coordinates remain unchanged because there is no camera movement. The zoom parameter is normally expressed as a scalar since it is the ratio of the camera's focal lengths. Zoom causes linear motion along both the x-axis and y-axis of the image plane i.e. the scalar components of the motion vector of a pixel are directly proportional to the corresponding scalar components of its displacement from the centre of the frame. The proportionality constants along the x-axis and the y-axis, which are functions of the zoom parameter, are equal. It has been shown by Tse and Baker in [136] that when zooming, the motion vector of the pixel (i,j) is:

$$\mathbf{v}_{ij} = \left(\frac{F_a}{F_b} - 1\right)\begin{bmatrix}s_i \\ s_j\end{bmatrix} \tag{6.3}$$

where F_b and F_a are the camera's focal lengths before, and after, the zoom, and F_a/F_b is the zoom parameter.

A similar motion vector field is created when the camera is translated along the direction of view. Let z_b and z_a be the z-coordinates (i.e. along the direction of view) of the object point, which corresponds to the pixel on the image plane, before, and after, the camera translation, respectively. The displacement of the pixel is given as:

$$\mathbf{v}_{ij} = \left(\frac{z_b}{z_a} - 1\right)\begin{bmatrix} s_i \\ s_j \end{bmatrix} \tag{6.4}$$

The above two concepts are generalized by Rath and Makur in [130] as follows. *Global* motion is referred to as zoom when the scalar components of the motion vector of a pixel are directly proportional to the corresponding scalar components of its displacement from the centre of the frame i.e.

$$\mathbf{v}_{ij} = \begin{bmatrix} z_x s_i \\ z_y s_j \end{bmatrix} \tag{6.5}$$

where z_x and z_y are called zoom parameters. When linear motion is due to the zooming of the camera,

$$z_x = z_y = \left(\frac{F_a}{F_b} - 1\right) \tag{6.6}$$

while when it is due to the translation of camera along the direction of view,

$$z_x = z_y = \left(\frac{z_b}{z_a} - 1\right) \tag{6.7}$$

The reason for using two zoom parameters is that in the majority of cases, *global* motion is usually accompanied by *local* motion. The values of the estimated parameters are affected differently along the *x*-axis and the *y*-axis depending on the nature of the *local* motion. One of the estimated parameters may be a better estimate than the other. Representing both parameters by a single parameter will not produce a better estimate.

So far the zoom and pan motions have been discussed separately. Although less frequent, there is still the possibility of the simultaneous occurrence of both pan and zoom motions. In such cases, the *global* motion field will be a combination of both pan a d zoom. Mathematically, an effective *global* motion vector has to be decomposed into two *global* motion vectors, each corresponding to one of the above mentioned models. The order of these two motion parameters is important, since the process is non-commutative ($a.b \neq b.a$), with different orders giving rise to different models for the resultant motion vector. Zoom followed by pan gives [130]:

$$\mathbf{v}_{ij} = \begin{bmatrix} z_x s_i \\ z_y s_j \end{bmatrix} + \mathbf{p} \tag{6.8}$$

whereas pan followed by zoom gives:

$$\mathbf{v}_{ij} = \begin{bmatrix} z_x(s_i + p_x) \\ z_y(s_j + p_y) \end{bmatrix} + \mathbf{p}$$

or:

$$\mathbf{v}_{ij} = \begin{bmatrix} z_x s_i \\ z_y s_j \end{bmatrix} + \begin{bmatrix} (1+z_x)p_x \\ (1+z_y)p_y \end{bmatrix} \tag{6.9}$$

Rath and Makur [130] generalise these two models as a single model:

$$\mathbf{v}_{ij} = \begin{bmatrix} a_1 s_i \\ a_3 s_j \end{bmatrix} + \begin{bmatrix} a_2 \\ a_4 \end{bmatrix} \tag{6.10}$$

where

$$a_1 = z_x \text{ and } a_2 = f_1(p_x, z_x) \tag{6.11}$$

$$a_3 = z_y \text{ and } a_4 = f_2(p_y, z_y) \tag{6.12}$$

In the above definition, z_x and z_y are the zoom factors along the x-axis and y-axis respectively, and (p_x, p_y) is the pan vector. Accordingly, the pan and zoom parameters are represented by a_1, a_2, a_3, and a_4.

6.3.2 Parameter Estimation

As the *global motion estimation* (GME) procedure depends on parametric models of camera motion and the way the model parameters are estimated, different GME techniques have been reported in the literature based on the diverse motion models discussed in the previous section. In [142], Dufaux and Konrad classify *global* motion parameter estimation methods into three categories: (i) direct minimisation of the prediction by a differential technique [141]; (ii) direct minimisation of the prediction error by a matching technique [135, 139, 143]; and (iii) a two-step method consisting of *local* motion estimation followed by estimation of the *global* motion parameters [130, 136, 137, 140, 144-148].

The first two categories represent *Least-Square* (LS) minimisations of motion parameter estimation. As the minimisation is carried out on the video sequence without corresponding establishment, both can be considered as direct methods. Conversely, the third category represents methods which carry out the estimation process in two stages. In the first stage, an overall motion field is computed generally by block-matching algorithms. The *global* motion parameters are subsequently computed by regression on this motion field. The technique can be seen as indirect as it does not compute the motion parameters from the luminance signal. Most indirect methods use a LS estimation method to minimize the prediction error function using a matching technique to estimate the *global* motion parameters.

As the purpose of *global* motion estimation is to compensate for the *global* motion already calculated by the block-based DTS algorithm detailed in Chapter 3, the indirect parameter estimation (two-stage) technique is exploited in this research. Whether direct or indirect implementation is employed, the main difficulty in estimating *global* motion parameters resides in the existence of independently moving objects which introduce a bias into the estimated parameters. To reduce the impact of localised object motion and detection errors [149] on the determination of *global* motion parameters, different techniques have been proposed for parameter estimation. A non-iterative histogram based *global* motion parameter estimation technique has been used to reduce the disturbance of local objects [137, 149, 150]. In [149], Kamikura and Watanabe calculate the parameters associated with pairs of blocks symmetrically located with respect to the centre of the image, and then a histogram of these parameters is used to calculate the parameters (pan and zoom) of the *global* motion. Meng and Chang [150] propose a technique to calculate the *global* motion parameters based on a histogram of the motion vector angles with respect to the origin of the frames. As the LS method is the most popular *global* motion estimation method, to reduce the disturbance of moving objects, an *iterative-least-square estimation* (ILSE) algorithm is used to remove the motion vectors of moving objects from the LS approximation by thresholding. As well, some robust direct methods exist, which use statistics and maximum-likelihood-theory, such as M-estimators [151] and Least-medium-of-square method [152] for parameter estimation. However, they introduce more computational complexity into the parameter estimation process.

For computational efficiency, sufficient accuracy and simplicity in application, the iterative LS method has been considered in this thesis.

6.3.3 *Modified Iterative Least-Square Estimation* (MILSE)

To estimate the parameters using *Iterative Least-Square Estimation* (ILSE), the procedure described by Rath and Makur in [130], is used with a modification, and is referred to as the *Modified Iterative Least-Square Estimation* (MILSE). In work by Rath and Makur [130], all the rows and columns of blocks in a frame of a sequence are considered for the first iteration, so that the parameter calculation depends on the whole frame. *Global* motion generally spreads over the frame uniformly as shown in Fig. 6.1. If only pan is involved, the value is constant for the entire frame, but in the case of zoom, the value is proportional to the distance from the centre points. The convergence centre is generally at the centre of a frame provided there is no panning. Fig. 6.1(a) shows the *global* motion characteristics of video frames where only zoom and a little pan is involved. Consequently, the convergence centre has been shifted from the centre of the frame.

(a) *Global* motion (Zoom) in the *Table Tennis* sequence.

(b) *Global* motion (Pan) in the *Flower Garden* sequence.

Fig. 6.1: *Global* motion needle diagrams for the *Table Tennis* and *Flower Garden* sequences.

Another assumption regarding *global* motion is that in most video sequences, only a few blocks are occluded by the moving objects and these objects are mostly in, or around, the middle of a frame, but rarely at the edge of the frame [19]. Based on this assumption, Aghbari *et al.* in [19] estimate the different types of camera motion using the macroblock motion information at the edge of the frame; thus, for panning motion, all motion vectors at the outer edges will be in the same direction, whereas for zoom-like motion, vectors on opposite sides will be in the opposite directions. Therefore, instead of using the motion vectors of all macroblocks, a few macroblocks, especially at the edge of the frame, are sufficient to enable calculation of the global parameters. To implement this strategy, the ILSE technique [130] has been modified, and is referred to as the MILSE technique. In the latter, instead of considering the rows and columns as indices, the number of blocks for parameter estimation is considered as follows.

Let there be N blocks in a video frame, and assume that the motion vector of a block is the motion vector of the central pixel of that block. Let $(v_x(k), v_y(k))$ be the measured motion vector, according to the original DTS algorithm (**Block 1** in Fig 1.6), of the block k, $k = 0, 1, \ldots, N\text{-}1$, whose central pixel's coordinates are $(s_x(k), s_y(k))$ with respect to the centre of the frame. In this regard, the *global* motion estimation model represented in (6.10) can be rewritten for camera zoom and pan as:

$$\begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix} = \begin{bmatrix} a_1 s_x(k) \\ a_3 s_y(k) \end{bmatrix} + \begin{bmatrix} a_2 \\ a_4 \end{bmatrix} \tag{6.13}$$

Now consider the ILSE algorithm [130], the optimal values for camera parameters (a_1, a_2, a_3, and a_4) are obtained by using the following criteria:

$$\min_{a_1,a_2}\sum_{k=0}^{N-1}\left(v_x(k)-a_1 s_x(k)-a_2\right)^2 \tag{6.14}$$

$$\min_{a_3,a_4}\sum_{k=0}^{N-1}\left(v_y(k)-a_3 s_y(k)-a_4\right)^2 \tag{6.15}$$

By differentiating with respect to the parameters, and setting the derivatives to zero, the following solution is obtained as:

$$a_1 = \frac{N\sum_{k=0}^{N-1} v_x(k)s_x(k) - \left(\sum_{k=0}^{N-1} v_x(k)\right)\left(\sum_{k=0}^{N-1} s_x(k)\right)}{N\sum_{k=0}^{N-1} s_x^2(k) - \left(\sum_{k=0}^{N-1} s_x(k)\right)^2} \tag{6.16}$$

$$a_2 = \frac{\left(\sum_{k=0}^{N-1} v_x(k)\right)\left(\sum_{k=0}^{N-1} s_x^2(k)\right) - \left(\sum_{k=0}^{N-1} v_x(k)s_x(k)\right)\left(\sum_{k=0}^{N-1} s_x(k)\right)}{N\sum_{k=0}^{N-1} s_x^2(k) - \left(\sum_{k=0}^{N-1} s_x(k)\right)^2} \tag{6.17}$$

$$a_3 = \frac{N\sum_{k=0}^{N-1} v_y(k)s_y(k) - \left(\sum_{k=0}^{N-1} v_y(k)\right)\left(\sum_{k=0}^{N-1} s_y(k)\right)}{N\sum_{k=0}^{N-1} s_y^2(k) - \left(\sum_{k=0}^{N-1} s_y(k)\right)^2} \tag{6.18}$$

$$a_4 = \frac{\left(\sum_{k=0}^{N-1} v_y(k)\right)\left(\sum_{k=0}^{N-1} s_y^2(k)\right) - \left(\sum_{k=0}^{N-1} v_y(k)s_y(k)\right)\left(\sum_{k=0}^{N-1} s_y(k)\right)}{N\sum_{k=0}^{N-1} s_y^2(k) - \left(\sum_{k=0}^{N-1} s_y(k)\right)^2} \tag{6.19}$$

As shown by Rath and Makur [130], to eliminate the influence of the presence of *local* motion, the above procedure is evaluated iteratively, and each iteration eliminates blocks whose motion vectors (estimated by any BMA) do not match with the current *global* motion fields. Matching means that a motion vector lies within a threshold, called the *motion vector matching threshold*, from the corresponding *global* motion vector.

6.3.4 Computational Complexity Analysis of MILSE

The computational complexity incurred in *global* motion estimation by the ILSE method depends on two factors: the number of blocks considered in each iteration and the number of iterations required for the convergence achieved. Rath and Makur [130] mention that the convergence usually occurs in less than 5 iterations. Therefore, the computational complexity of

the ILSE and MILSE techniques is analysed in terms of the number of blocks used in the first iteration.

If the frame size, for example, is $[N_h, N_v]$ pixels and the block size is N^2 pixels, the total number of blocks in a frame is $\frac{N_h \times N_v}{N^2}$. In a 2-dimentional form this can be represented as $[B_h, B_v]$ where B_h and B_v represent the horizontal and vertical dimension of the blocks respectively. Suppose the total number of operations required for calculating the camera parameters for each block is ξ, then the total number of operations required for the whole frame is $\xi \times (B_h \times B_v)$ for each iteration, which is the computational cost involved in ILSE technique.

Clearly, if a subset of blocks is considered instead of all the blocks of a frame, the number of operations will be fewer, which is the rationale behind the MILSE technique.

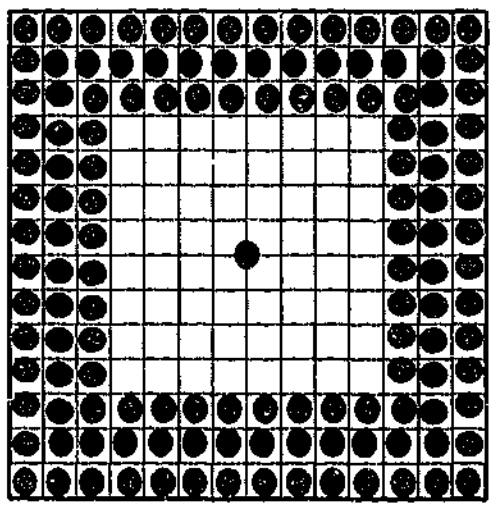


Fig. 6.2: An example of all the macroblocks in the three outermost grids of a frame.

If the frame size is $[N_h, N_v]=[352,240]$ and block size is $[N, N]=[16,16]$, the total number of blocks in this frame is $(B_h \times B_v)=330$. The total number of operations required is $\xi \times 330$ when all the blocks of a frame are considered for one iteration in the ILSE technique. Conversely, if only those blocks in the outermost first and second grids, shown as G_0 and G_1 in Fig. 6.2, are considered, the total number of operations involved in the first iteration is $\xi \times 132$. If the second and third outermost grids G_1 and G_2 are considered, the total number of operations required is $\xi \times 116$. For these two cases, the computational cost is reduced by 60% and 65%, respectively, compared to what is required when all the blocks are considered for parameter estimation in the first iteration. Experimental results

This research confirms that using the outer grid block motion vectors to calculate *global* motion parameter exhibits better performance than using the inner grids which are located

around the centre of the frame, since the local object generally exists towards the centre of the frame.

6.4 *True* Object Motion Estimation

If there is no *global* motion involved in a video sequence, the *true* object motion can be directly obtained by eliminating the *false* motion components from the block motion captured by the DTS algorithm discussed in Chapter 3. However, if there is *global* motion involved in a sequence, it will be necessary to cancel the *global* motion components before filtering the *false* motion vector, in order to retain only the *true* object motion vectors from the block motion captured by the DTS algorithm. The *global* motion cancellation technique is described below.

6.4.1 *Global* Motion Cancellation

For *global* motion cancellation, generally known as *global motion compensation*, firstly *global* motion parameters are calculated. In Section 6.3, a *global* motion parameter estimation technique has been described for parameters (pan-zoom) estimation where the pan and zoom were represented by four variables (a_1, a_2, a_3, and a_4). After calculating these four parameters, the *global* motion vectors for each block in a frame can be calculated as (6.13).

If the *true* object motion vector is represented by $(o_x(k), o_y(k))$ of the k^{th} block of a frame where $k = 0, 1, \ldots, N\text{-}1$, it can be calculated as:

$$\begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} = \begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix} - \begin{bmatrix} a_1 s_x(k) \\ a_3 s_y(k) \end{bmatrix} - \begin{bmatrix} a_2 \\ a_4 \end{bmatrix} \tag{6.20}$$

where $(v_x(k), v_y(k))$ represents the block motion vector calculated by the DTS algorithm.

Once *global* motion has been compensated from the estimated block motion, *true* object motion vectors are clustered in the blocks containing one or more objects. As block motion estimation cannot be performed with full accuracy due to the limitations of block-based estimation techniques discussed previously, *false* motion vectors appear as noise, together with the *true* object motion vectors. To retain only the *true* object motion vectors, these *false* motion vectors need to be removed.

6.4.2 Filtering the *False* Motion Vector

Among the various existing linear and non-linear filters, the most popular are the mean and median filters [153-157]. In the following two sections, the concepts behind these two type of filters will be discussed briefly.

6.4.2.1 The Mean Filter

The idea of mean filtering is simply to replace each value with the mean (average) value of its neighbours, including itself. This has the effect of smoothing values that are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter [158]. Like other convolutions, it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, although a larger kernel (e.g. 5×5 square) can be used for heavier smoothing. For example, in Fig. 6.3(b) the results are shown for the situation where a 3×3 mean filter is applied to a 3×3 image as shown in Fig. 6.3(a) where each value may represent the intensity of each pixel. Two major characteristics of the mean filter are:

- A single, very unrepresentative value can significantly affect the mean value of its neighbourhood.
- When the filter neighbourhood straddles an edge, the filter will interpolate new values.

1	3	2
2	12	1
3	3	3

(a) Unfiltered values.

4.5	3.5	3.7
4.0	3.3	3.3
4.0	3.2	3.1

(b) Filtered values.

Fig. 6.3: 3×3 kernel mean filter.

6.4.2.2 The Median Filter

Like the mean filter, the median filter considers each value, in turn, and looks at its nearby neighbours to decide whether or not it is representative of its surroundings. Instead of simply replacing a value with the mean of neighbouring values, it replaces the value with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighbourhood into numerical order and then replacing the pixel under consideration with the middle pixel value. If the neighbourhood under consideration contains an even number of pixels, the average of the two middle pixel values is used. Fig. 6.4(b) shows the results of a 3×3 kernel median filter applied to the 3×3 image in Fig. 6.4(a). Two major characteristics of the median filter are:

- The median is a more robust average than the mean, and so a single, very unrepresentative, value in a neighbourhood will not affect the median value significantly.

- Since the median value must actually be one of the values in the neighbourhood, the median filter does not create new, unrealistic, values when the filter straddles an edge.

1	3	2
2	12	1
3	3	3

(a) Unfiltered values.

2.5	2.0	2.0
3.0	3.0	2.0
2.5	2.0	2.0

(b) Filtered values.

Fig. 6.4: 3×3 kernel median filter.

While these filter types are effective for impulse noise suppression, the median filter and its variants have also been used in many applications to reduce noise from block motion vectors [17, 21, 31, 72] for vector field smoothing. However, the main requirement for the filter in this application is not noise reduction in image enhancement or motion vector field smoothing, but to explicitly eliminate the *false* motion vectors and retain the *true* object motion vectors. For this reason, a new filter design is proposed.

6.4.2.3 The *Mean Accumulated Thresholded* (MAT) Filter

In real world video sequences, most moving objects generally occupy more than one neighbouring macroblock (Section 4.2). Based on this characteristic, it can be assumed that *true* object motion vectors should always occur in a clustered form whereas *false* motion vectors will tend to appear as impulsive noise. In order to illustrate this, the following simple example of a motion vector field is provided in Fig. 6.5. The *true* motion vector is represented by m and *false* motion vector by m'. Real world video will involve a much more complex motion vector field; however, as results will show (Section 6.5.2.2. and Appendix F) the fundamental principle holds.

0	0	0	0	0	0	0	0
0	0	0	0	0	m	m	0
0	0	0	0	0	m	m	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	m'	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 6.5: An example of a motion vector field.

The objective is to eliminate m' and retain m in the vector field. The simplest strategy is to define a noise tolerance threshold where the decision is based on that threshold value [18]. This approach is flawed however, since the threshold only performs well if the *true* and *false* motion vectors have different lengths. In the case where the length of both m and m' are equal, this technique does not separate *true* motion vectors from *false* ones.

For filtering impulse noise, the most well-known technique is the median filtering technique which has been used in [17, 21, 31, 72] for noise reduction. Though the median filter performs well for noise reduction in a motion vector field, such a filter is not suitable for this application. For example, if a 3×3 median filter is implemented on the motion vector field shown in Fig. 6.5, the filtered values will be as shown in Fig. 6.6. These clearly demonstrate that though the *false* vector has been removed, all the *true* motion vectors have also been eliminated.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 6.6: Filtered values for the motion vector represented in Fig. 6.5.

The performance of a mean filter is now examined in this application. After implementing a 3×3 mean filter on the motion vector field in Fig. 6.5, the filtered values are as shown in Fig. 6.7, where a number of new *false* motion vectors have been introduced. To remove these new *false* vectors around the *true* vectors shown in the shaded blocks in Fig. 6.7, a predefined threshold, called the *false motion vector elimination threshold* T_f, must be used such that $\frac{2m}{9} < T_f < \frac{4m}{9}$. If the length of both vectors (*true* and *false*) is equal, then $\frac{m'}{9} = \frac{4m}{9} or\ m' = 4m$. In this case, any value of T_f, within the defined range, will be unable to remove the *false* vectors and retain the *true* ones.

From this simple example, it can be seen that the length of the *false* motion vector is greater than that of the *true* motion vector and the length ratio is 4:1. One strategy for removing these *false* vectors is to consider gradually increasing the length of the *true* motion vector with a

higher ratio compared to the increasing ratio of the length of the *false* motion vector, so that the ratio of the two different vector lengths is reduced gradually. When, finally, the *true* motion vector length becomes greater than that of the *false* one, a predefined threshold can separate the *true* one from the *false* one. To achieve this objective, accumulation of the mean vector length, in addition to the original vector length, is utilised. The reasoning behind this is that *true* motion vectors frequently occur in a clustered form, whereas *false* vectors tend to occur as isolated impulsive noise.

0	0	0	0	$m/9$	$2m/9$	$2m/9$	$m/9$
0	0	0	0	$2m/9$	$4m/9$	$4m/9$	$2m/9$
0	0	0	0	$2m/9$	$4m/9$	$4m/9$	$2m/9$
0	0	0	0	$m/9$	$2m/9$	$2m/9$	$m/9$
$m'/9$	$m'/9$	$m'/9$	0	0	0	0	0
$m'/9$	$m'/9$	$m'/9$	0	0	0	0	0
$m'/9$	$m'/9$	$m'/9$	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 6.7: Filtered values for motion vector represented in Fig. 6.5.

For the example, in Fig. 6.7, the length ratio of the *false* and *true* motion vector is 4:1. If the above mentioned accumulation procedure is applied to these motion vectors, the maximum length of the *false* motion vector after one iteration becomes:

$$4m+\frac{4m}{9}=\frac{40m}{9} \tag{6.21}$$

and the length of the *true* vector is:

$$m+\frac{4m}{9}=\frac{13m}{9} \tag{6.22}$$

From (6.21) and (6.22), it can be shown that the ratio of the length of the *false* and *true* motion vectors is now only 3.08:1 after one iteration.

Now, if the second iteration is considered, the length of the *false* motion vector becomes

$$4m+\frac{40m}{81}+\frac{8\times 4m}{81}=5.4m, \tag{6.23}$$

and the length of the *true* motion vector is:

$$\frac{13m}{9}+\frac{52m}{81}+\frac{10m}{81}=2.4m \tag{6.24}$$

so the ratio of the length of the *false* and *true* motion vectors is now 2.25 which, again, indicates the reduction of length ratio between the *false* and *true* motion vectors. In the same manner, if this iterative mean accumulated process continues, eventually after a number of iterations, the length of the *false* motion vectors (noise) will be smaller than the length of the *true* motion vectors. In that case, by applying a suitable value of T_f, it is possible to remove the entire set of *false* motion vectors in order to retain only the *true* one.

While a simple example has been used to explain the rationale behind the idea, the *false* motion vector elimination process has been formulated in the *Mean Accumulated Thresholded* (MAT) filter, which is designed explicitly for this application.

The MAT filter has two phases. The first phase is basically an iterative *in-place* application of the mean filter. However, in this case, a major difference arises in how the *in-place* values are updated. For each iteration, the mean value is added to, instead of replacing, the existing value as follows:

$$\begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} = \begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} + \begin{bmatrix} \text{mean}_x(k) \\ \text{mean}_y(k) \end{bmatrix} \tag{6.25}$$

where $\left(o_x(k), o_y(k)\right)$ represents the x and y components of the motion vector in the current block k, which are available after global motion cancellation, and where $\text{mean}_x(k)$ and $\text{mean}_y(k)$ are the mean values of the x and y components of the motion vectors, respectively, in the neighbourhood of any kernel considered for the current block, k.

The second phase of the MAT filter is to apply the false motion vector elimination threshold, T_f, so that the only motion vectors retained are those whose lengths are higher than T_f. This is mathematically formulated as:

$$\begin{cases} \text{If } \sqrt{\left(o_x(k)\right)^2 + \left(o_y(k)\right)^2} \le T_f, & \text{eliminate the vector in block } k; \\ \text{Otherwise,} & \text{retain the vector in block } k. \end{cases} \tag{6.26}$$

Before examining the performance of the filter, a few key points need to be highlighted in respect to applying the MAT filter.

1. The MAT filter has been explicitly designed to eliminate *false* motion vectors while retaining the *true* motion vectors. It is not designed for vector field smoothing purposes.
2. The MAT filter can be integrated with any existing BMA for *true* object motion vector capture.

3. The number of iterations in the MAT filter depends on the video content and the performance of the search algorithm. These must be selected empirically for optimising the performance of any BMA using the MAT filter.
4. The threshold T_f setting is empirically derived to maximise the number of retained *true* object motion vectors while minimising the number of *false* motion vectors.
5. The overall computational complexity of the MAT filter depends on the kernel size used and the number of iterations required in the whole process.
6. One assumption in the literature is that motion vectors tend to occur in a clustered form which defines moving objects. For the inherent limitation of the BMA search technique, the performance of MAT filter will probably not be so effective, and capture rates will deteriorate if there is large cluster of *false* motion vectors with no moving objects. For most real world objects however, this clustering effect is rare, and thus the MAT filter will improve the overall number of *false* motion vectors eliminated.

6.5 Performance Analysis

The purpose of this section is to analyse the experimental performance, firstly, of the MILSE technique for *global* motion estimation and secondly, the DTS algorithm using the MAT filter for *true* object motion estimation.

6.5.1 Performance Analysis of MILSE

In this section, simulation results for *global* motion parameters (zoom and pan) estimation, in terms of a_1, a_2, a_3, and a_4, are presented using the original ILSE method [130] and the new MILSE method. The simulation was carried out using different video sequences with different motion types as detailed in Appendix B.

In the simulation program, two predefined thresholds were used to compare motion vector magnitude and angle. If the difference in magnitude and angle between the original motion vector calculated by the DTS algorithm and the calculated current *global* motion using (6.13) is greater than these predefined thresholds, they are considered mismatched motion vectors and are removed during the next iteration. Tables 6.1 and 6.2 show the statistical comparison of camera pan and zoom factors represented by a_1, a_2, a_3, and a_4, calculated by considering the motion vectors for a range of different numbers of macroblocks in given frames.

Test sequences	a_1 (zoom)	a_2 (pan)	a_3 (zoom)	a_4(pan)	No. of blocks considered	
Table Tennis (Frames #32 and #33)	-0.02	-0.33	-0.02	0.24	All blocks	ILSE
	-0.02	-0.40	-0.02	-0.14	Blocks in G_0 and G_1	MILSE
	-0.02	-0.27	-0.02	0.14	Blocks in G_1 and G_2	
	-0.02	-0.47	-0.02	-0.16	Blocks in G_1, G_2, and G_3	
	-0.02	-0.43	-0.02	0.05	Blocks in G_1, G_2, G_3, and G_4	
	-0.02	-0.41	-0.01	0.04	Blocks in G_1, G_2, G_3, G_4, and G_5	
	-0.02	-0.33	-0.01	0.12	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Flower Garden (Frames #10 and #11)	0.00	-2.00	0.00	0.00	All blocks	ILSE
	0.00	-2.51	0.00	0.00	Blocks in G_0 and G_1	MILSE
	0.00	-2.00	0.00	0.00	Blocks in G_1 and G_2	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, and G_3	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, and G_4	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, and G_5	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Ballet (Frames #99 and #100)	0.00	4.04	0.00	0.02	All blocks	ILSE
	0.00	3.87	0.00	0.08	Blocks in G_0 and G_1	MILSE
	0.00	3.95	0.00	0.00	Blocks in G_1 and G_2	
	0.00	3.96	0.00	0.05	Blocks in G_1, G_2, and G_3	
	0.00	3.77	0.00	0.00	Blocks in G_1, G_2, G_3, and G_4	
	0.00	3.77	0.01	0.02	Blocks in G_1, G_2, G_3, G_4, and G_5	

Table 6.1: Statistical comparison of camera pan (a_2 and a_4) and zoom (a_1 and a_3) factors in relation to the different numbers of macroblocks considered.

Test sequences	a_1 (zoom)	a_2 (pan)	a_3 (zoom)	a_4(pan)	No. of blocks considered	
Table Tennis (Frames #33 and #34)	-0.02	-0.12	-0.02	0.22	All blocks	ILSE
	-0.02	-0.38	-0.02	0.00	Blocks in G_0 and G_1	MILSE
	-0.02	-0.10	-0.02	0.03	Blocks in G_1 and G_2	
	-0.02	-0.22	-0.02	-0.02	Blocks in G_1, G_2, and G_3	
	-0.02	-0.18	-0.01	0.12	Blocks in G_1, G_2, G_3, and G_4	
	-0.02	-0.18	-0.01	0.29	Blocks in G_1, G_2, G_3, G_4, and G_5	
	-0.02	-0.07	-0.01	0.31	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Flower Garden (Frames #20 and #21)	0.00	-2.00	0.00	0.00	All blocks	ILSE
	0.00	-2.00	0.00	0.00	Blocks in G_0 and G_1	MILSE
	0.00	-2.00	0.00	0.00	Blocks in G_1 and G_2	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, and G_3	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, and G_4	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, and G_5	
	0.00	-2.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Ballet (Frames #97 and #98)	0.00	2.48	0.00	0.04	All blocks	ILSE
	0.00	2.83	0.00	0.23	Blocks in G_0 and G_1	MILSE
	0.00	3.89	0.00	0.01	Blocks in G_1 and G_2	
	0.00	3.71	0.00	0.29	Blocks in G_1, G_2, and G_3	
	0.00	3.20	0.01	0.12	Blocks in G_1, G_2, G_3, and G_4	
	0.00	2.82	0.01	0.00	Blocks in G_1, G_2, G_3, G_4, and G_5	

Table 6.2: Statistical comparison of camera pan (a_2 and a_4) and zoom (a_1 and a_3) factors in relation to the different numbers of macroblocks considered.

The first test sequence reflected in both Tables 6.1 and 6.2 contains two pairs of images (frames #32 and #33, and frames #33 and #34) of the *Table Tennis* (352×240 pixels) sequence

in where the camera zooms out, whilst slightly panning, with the moving objects including ball, bat, and the hand of the player holding the bat. Tables 6.1 and 6.2 show that the values of the zoom parameters, a_1 and a_3, were very similar for all cases for both sets of frames, except for the blocks in the outermost grids, G_0 and G_1. This indicates that some noise has been introduced due to boundary artefacts in the outermost grid, introducing a small error. It is also shown that when the blocks of G_1 and G_2 were considered, the panning factors of a_2 and a_4 were smaller than for all other cases. As these images contain almost no panning, the low values of a_2 and a_4 are fully consistent with expectations.

The next test sequence contains two pairs of images (frames #10 and #11, and frames #20 and #21) of the *Flower Garden* (352×240 pixels) sequence where the camera is panning horizontally to the right, and there are no moving objects. Tables 6.1 and 6.2 show that only $a_2 \neq 0$, indicating that there was no zooming and vertical panning involved in this sequence. Table 6.1 also illustrates that the value of a_2 was different for the blocks, G_0 and G_1, compared to all others, due to the aforementioned boundary artefacts.

For the *Ballet* (360×240 pixels) sequence, two pairs of images (frames #97 and #98, and frames #99 and #100, respectively) were considered; these contained camera panning to the right, and no zooming. Tables 6.1 and 6.2 also show that the zoom parameters were zero for all cases, which is consistent with the actuality observed in this sequence.

So far, the performance of MILSE and ILSE has been analysed in which consecutive pairs of frames have been considered for *global* motion estimation. Generally, the pan and zoom factors in a video sequence changes proportionally to the distance between the current and the reference frames. To analyse the effectiveness of the MILSE technique, a number of experiments were also conducted based on non-consecutive (*skipping*) frames. Table 6.3 shows the simulation results when *skipping* one and two frames of the *Table Tennis* and *Flower Garden* video sequences. It is interesting to note that the zoom and pan factors gradually increase when the distance between the current and the reference frames increases. It can also be observed that the parameter values were similar in all cases except when blocks in G_0 and G_1 were considered, again indicating the effect of boundary artefacts.

From the above analysis, it can be observed that *global* motion parameter estimation does not require a consideration of all blocks of a frame. It is also shown that if only those blocks in the second and third outermost grids are considered, then this provides better results compared to others, as well as avoiding boundary artefacts. Consequently, the proposed MILSE technique can be shown to improve computational efficiency by 65% compared with the first iteration of the ILSE technique described by Rath ands Makur [130].

Test sequences	a_1 (zoom)	a_2 (pan)	a_3 (zoom)	a_4(pan)	No. of blocks considered	
Table Tennis (Frames #32 and #34)	-0.03	-0.32	-0.03	-0.27	All blocks	ILSE
	-0.03	-0.38	-0.03	-0.09	Blocks in G_0 and G_1	MILSE
	-0.03	-0.57	-0.03	-0.08	Blocks in G_1 and G_2	
	-0.03	-0.44	-0.03	-0.02	Blocks in G_1, G_2, and G_3	
	0.03	0.40	-0.03	0.17	Blocks in G_1, G_2, G_3, and G_4	
	-0.03	-0.31	-0.03	0.27	Blocks in G_1, G_2, G_3, G_4, and G_5	
	-0.03	-0.29	-0.03	0.31	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Table Tennis (Frames #32 and #35))	-0.04	-0.26	-0.05	0.51	All blocks	ILSE
	-0.03	-0.08	-0.44	-0.79	Blocks in G_0 and G_1	MILSE
	-0.04	-0.25	-0.05	0.33	Blocks in G_1 and G_2	
	-0.04	-0.41	-0.05	0.28	Blocks in G_1, G_2, and G_3	
	-0.06	-0.33	-0.05	0.3	Blocks in G_1, G_2, G_3, and G_4	
	-0.05	-0.24	-0.04	-0.77	Blocks in G_1, G_2, G_3, G_4, and G_5	
	-0.05	-0.20	-0.04	0.82	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Flower Garden (Frames #10 and #12)	0.00	2.90	0.00	0.00	All blocks	ILSE
	0.00	3.40	0.00	0.00	Blocks in G_0 and G_1	MILSE
	0.00	3.00	0.00	0.00	Blocks in G_1 and G_2	
	0.00	3.00	0.00	0.00	Blocks in G_1, G_2, and G_3	
	0.00	3.00	0.00	0.00	Blocks in G_1, G_2, G_3, and G_4	
	0.00	3.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, and G_5	
	0.00	3.00	0.00	0.00	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	
Flower Garden (Frames #10 and #13)	0.00	3.51	0.00	0.24	All blocks	ILSE
	0.00	2.92	0.00	0.45	Blocks in G_0 and G_1	MILSE
	0.00	4.00	0.00	0.52	Blocks in G_1 and G_2	
	0.00	4.00	0.00	1.00	Blocks in G_1, G_2, and G_3	
	0.00	4.00	0.00	0.45	Blocks in G_1, G_2, G_3, and G_4	
	0.00	4.00	0.00	0.33	Blocks in G_1, G_2, G_3, G_4, and G_5	
	0.00	4.00	0.00	0.22	Blocks in G_1, G_2, G_3, G_4, G_5, and G_6	

Table 6.3: Statistical comparison of camera pan (a_2 and a_4) and zoom (a_1 and a_3) factors in relation to the different numbers of macroblocks considered.

6.5.2 Performance Analysis of the DTS and MAT Filter

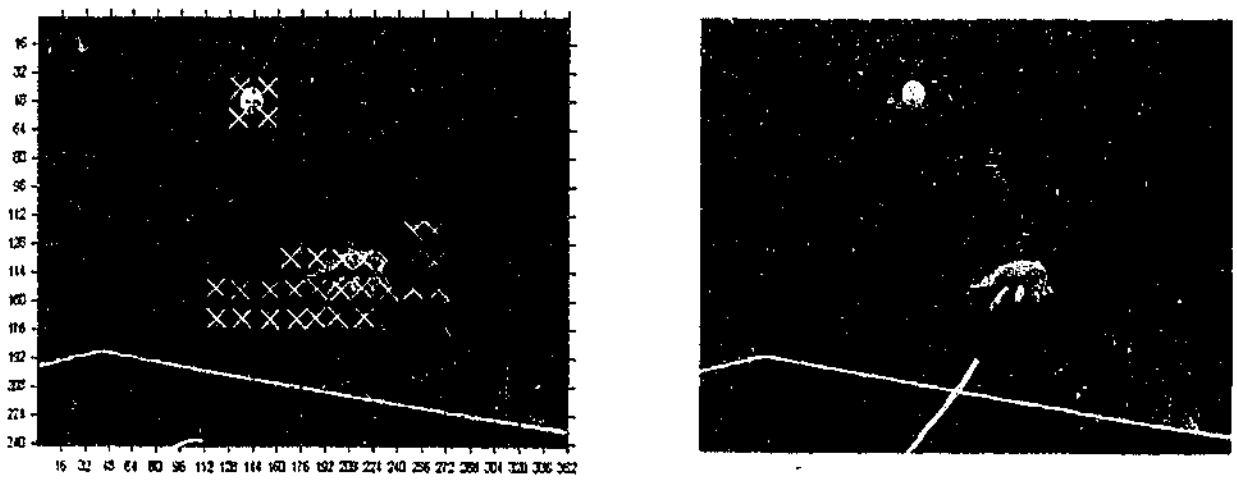
6.5.2.1 Analysis of the Kernel Effect in the MAT Filter

Kernel size, representing the size of the neighbourhood to be considered for calculating the mean value, is an important factor in the performance of the MAT filter. Before analysing the performance of the DTS algorithm with the MAT filter, it is important to analyse the kernel size effect. The standard *Table Tennis* (352×240 pixels) and *Foreman* (176×144 pixels) video sequences were considered in relation to this effect, with different kernel sizes being utilised. Different numbers of iterations were also considered as the performance of the MAT filter depends on this factor.

For the *Table Tennis* sequence, frames #32 and #33 shown in Fig. 6.8, were considered where the bat, ball, and the hand of the player holding the bat are the moving objects. Based on *a priori* knowledge about the moving objects in these frames, 30 moving macroblocks out of a total of 330, each having a size of 16×16 pixels, are identified manually and indicated with the

use of an × sign in Fig. 6.8(a). The motion vectors (MV) of these specific 30 macroblocks are the *true* vectors, whereas the motion vectors of the remaining blocks are the *false* one.

The block motion vector calculated by the DTS algorithm is shown in Fig. 6.9, which contains *true* object motion as well as *global* motion. To obtain the *true* object motion vector, these global motion components have been compensated according to the *global* motion estimation and compensation processes discussed in Sections 6.3 and 6.4. Fig. 6.10 shows the *global* motion compensated motion vector needle diagram, from which it is clear that while the only moving objects are the ball, bat and the hand holding the bat, spurious (*false*) motion vectors exist together with the *true* object motion vectors. These *false* vectors were introduced as a result of the inherent limitations of block-based motion estimation techniques. Some impulsive noise in the form of *false* motion vectors was also introduced due to imperfect *global* motion parameter modeling. To eliminate these spurious motion vectors, the MAT filter with different kernel sizes was applied.



(a) Current frame. (b) Reference frame.

Fig. 6.8: Frames #32 and #33 of the *Table Tennis* sequence.

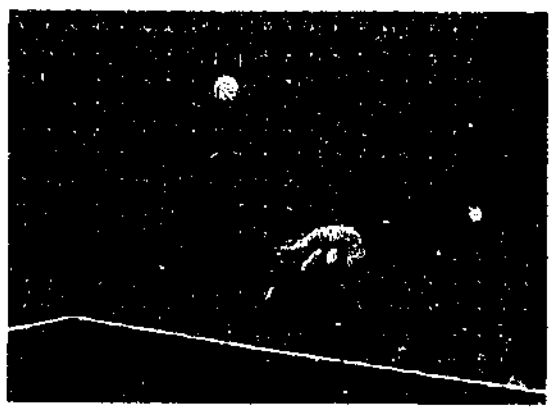


Fig. 6.9: Block motion vector captured by the DTS algorithm.

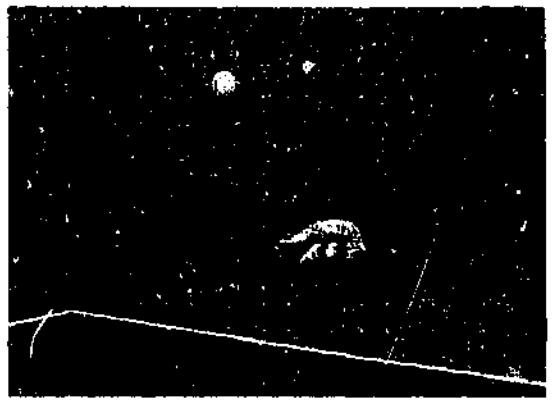


Fig. 6.10: Object motion vectors after *global* motion compensation.

Tables 6.4 and 6.5 show the experimental results for the *Table Tennis* and *Foreman* video sequences using different kernel sizes (3×3, 5×5, 7×7, and 9×9) and different numbers of iterations (2, 3, and 4). The objective in using the filter is to capture the maximum percentage of possible *true* motion vectors with a minimum percentage of *false* vectors. To remove the *false* motion vectors after each iteration, a range of values for T_f, starting from an empirically selected low value, is gradually increased until all the *false* motion vectors are removed. The length of a motion vector increases with the number of iterations, so when the length of a motion vector is high, the range for the threshold, T_f, in eliminating *false* vectors, is also correspondingly high. The step size for incrementing T_f was empirically selected. The percentage of *true* and *false* motion vectors after thresholding at different values of T_f was then calculated.

Kernel size / No. of iterations	3×3			5×5			7×7			9×9		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
2	56.7	0.3	8.0	70.0	1.0	5.5	53.3	0.3	5.5	50	0.7	5.0
	46.7	0.0	9.0	66.7	0.0	6.0	40.0	0.9	6.0	40	0.0	5.5
3	70.0	0.3	12.0	76.7	1.3	7.5	70.0	0.3	4.0	63.3	0.7	3.5
	63.3	0.0	13.0	70.0	0.0	8.0	53.3	0.0	4.5	43.3	0.0	4.0
4	80.0	0.7	18.0	73.3	0.3	16.0	36.7	0.0	12.0	56.7	0.7	9.0
	80.0	0.0	20.0	50.0	0.0	17.0	26.7	0.0	13.0	40.0	0.0	10.0

Table 6.4: Kernel effect on the performance of the MAT filter for the *Table Tennis* sequence.

Table 6.4 shows the two best pairs of results having a maximum percentage of *true* object motion vectors, and minimum percentage of *false* motion vectors (0%, or near 0%) for the *Table Tennis* sequence. For example, for a 3×3 kernel with 3 iterations where T_f = 12, the percentage of *true* object motion vectors was 70% whereas the percentage for *false* motion vectors was 0.3%. To eliminate the remaining *false* motion vectors, a higher value, T_f = 13, was

applied. Accordingly, the percentage of *false* motion vectors decreased to 0% while the percentage of *true* vectors also decreased to 63.3%. It is obvious from (6.26) that the higher the value of T_f, the higher the number of vectors will be eliminated. If a value of $T_f < 12$ was applied, the percentage of remaining *false* motion vectors would be ≥ 0.3%. Following the same procedure, the test results using different kernel sizes and iterations are shown in Table 6.4 for the *Table Tennis* sequence.

For the *Foreman* sequence, the pair of frames #8 and #9 shown in Fig. 6.11, were considered, as these contain both object motion and low camera panning. The corresponding experimental results are given in Table 6.5.

(a) Current Frame. (b) Reference frame.

Fig. 6.11: Frames #8 and #9 of the *Foreman* sequence.

Kernel size / No. of iteration	3×3			5×5			7×7			9×9		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
2	43.5	1.9	3.6	28.3	1.9	3.5	28.3	5.7	3.0	15.2	1.9	3.0
	39.1	0.0	3.8	23.9	0.0	4.0	13.0	0.0	3.5	10.9	0.0	3.5
3	71.7	1.9	4.8	76.1	5.7	4.0	73.9	3.8	3.5	41.3	1.9	3.5
	69.6	0.0	5.0	65.2	0.0	4.5	56.5	0.0	4.0	19.6	0.0	4.0
4	78.3	1.9	7.6	76.1	1.9	7.5	84.8	5.7	5.5	80.4	1.9	2.5
	76.1	0.0	7.8	67.4	0.0	8.0	73.9	0.0	6.0	63.0	0.0	3.0

Table 6.5: Kernel effect on the performance of the MAT filter for the *Foreman* sequence.

Tables 6.4 and 6.5 show that though the kernel effect was not significant, the number of *true* motion vectors captured with a kernel size of 7×7 or 9×9 was less than when compared to that for smaller kernel sizes (3×3 or 5×5). Generally, higher kernel sizes are used for heavier smoothing of images or motion vector fields; the purpose of this filter, however, was not for smoothing the motion vectors. Most objects in real world video sequences occur in small clustered forms, where each object contains a few neighbouring macroblocks but not the whole

frame. If a larger sized kernel is used, it includes a larger area of the frame, which will eventually reduce the lengths of the *true* vectors. In this case, in eliminating the *false* vectors, any value of threshold T_f will remove a higher number of *true* object motion vectors as well. For this reason, larger kernel sizes perform worse compared to smaller-sized kernels. Again from (6.25), it can be seen that the computational complexity for calculating the mean value is directly proportional to the kernel size utilised, so from a computational point of view, the smaller kernel incurs less computational cost compared to a larger kernel. For these reasons, the 3×3 sized kernel will be considered in the next section for analysis of the performance of the MAT filter and the DTS algorithm in terms of *true* object motion estimation.

6.5.2.2 Analysis of MAT filter and DTS for *True* Object Motion Estimation

To evaluate the combined performance of the MAT filter and DTS algorithm for *true* object motion estimation, a number of experiments were performed using the standard and non-standard video sequences, *Table Tennis*, *Foreman,* and *Rocket* (Appendix B), each of which exhibits different types of object motion. The value of linear threshold control parameter, C_L, in the DTS algorithm has been chosen such that the search speed remains similar, while comparing DTS with the fast algorithms, TSS and NTSS.

The experimental results for the pair of frames #32 and #33 in Fig. 6.8 of the *Table Tennis* sequence are given in Table 6.6, and were obtained utilising the same procedure explained in the previous section. Detailed supplementary results have also been included in Appendix F. To analyse the effect of the MAT filter, the experimental results from 0 to 5 iterations are shown in Table 6.6. It is important to clarify that the number of iterations used is not optimal, but based on the experimental results, it was found that the performance of all BMAs examined did not change significantly after four iterations. The increased percentage of *true* motion vectors captured for FS, NTSS, TSS, and DTS was 10%, 6.7%, 13.3%, and 17%, respectively, from iteration 3 to 4, whereas it was only 6.3%, 3.3%, 6.7%, and 0% from iteration 4 to 5. For this reason, only those results from 0 to 5 iterations have been included.

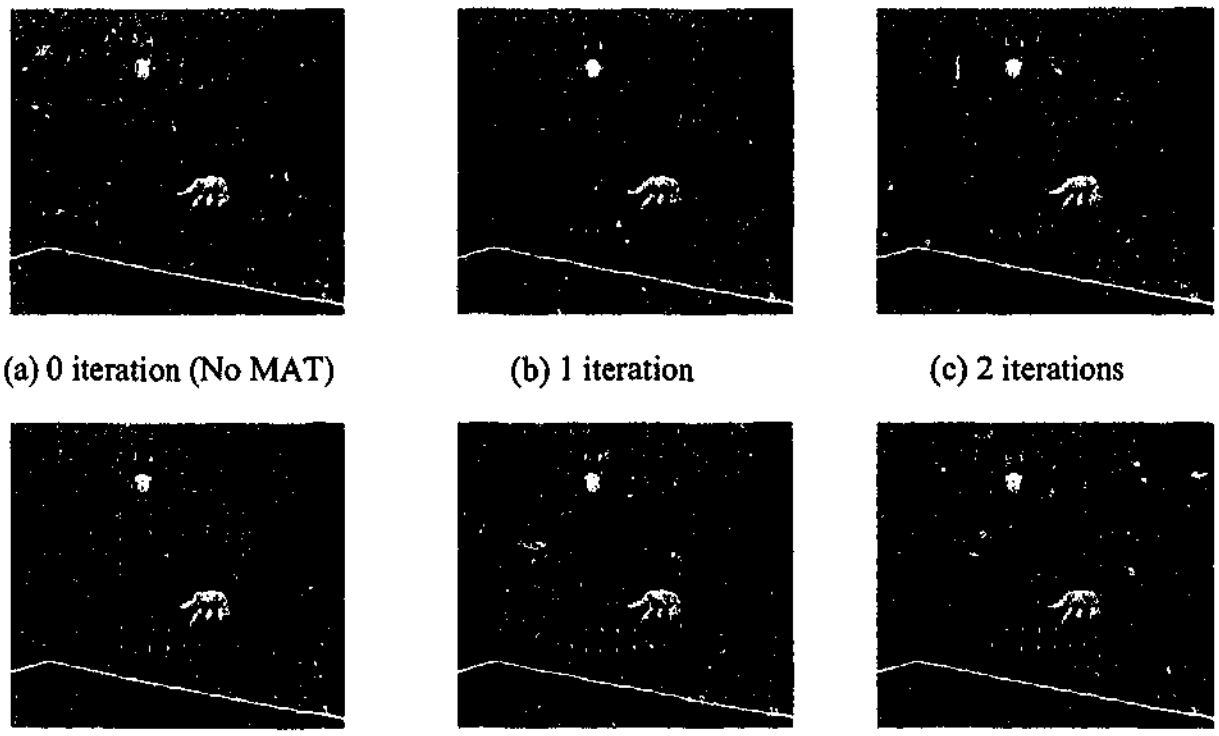
Table 6.6 shows that at iteration 0 (no MAT filter used), all the algorithms had 0% *false* motion vectors. However, NTSS and TSS had 0% of *true* motion vectors compared with the FS and DTS algorithms which had 13.3% of *true* motion vectors. This confirms that the DTS algorithm outperforms TSS and NTSS and performs as well as the FS algorithm in capturing *true* object motion vectors, without needing to use the MAT filter for this sequence.

It is clear that the percentage of *true* motion vectors captured by any BMA significantly increases with the number of iterations to the point at which the percentage of *false* motion vectors is zero. For example, the *true* motion vectors captured by the DTS algorithm increased

from 13.3% to 80% using the MAT filter over 5 iterations. This improvement can also be observed visually in Fig. 6.12. A similar trend was also found for the NTSS, TSS, and FTS algorithms. This demonstrates that when the MAT filter is used, the percentage of *true* object motion vectors captured significantly increases, while all *false* motion vectors are eliminated from the motion vector field.

BMAs / No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0	16.7	0.3	5.0	6.7	0.7	8.0	3.3	14.3	8.0	16.7	0.3	5.0
(No MAT)	13.3	0.0	6.0	0.0	0.0	9.0	0.0	0.0	9.0	13.3	0.0	6.0
1	13.3	0.3	8.0	16.7	0.3	9.0	10.0	5.7	9.0	46.7	1.7	5.0
	13.3	0.0	9.0	13.3	0.0	10.0	6.7	3.3	10.0	40.0	0.0	6.0
2	26.7	0.3	12.0	23.3	0.3	12.0	13.3	6.3	13.0	56.7	0.3	8.0
	16.7	0.0	13.0	16.7	0.0	13.0	10.0	3.7	14.0	46.7	0.0	9.0
3	26.7	0.3	19.0	56.7	0.7	15.0	16.7	4.7	21.0	70.0	0.3	12.0
	26.7	0.0	19.5	50.0	0.0	16.5	10.0	1.7	22.5	63.3	0.0	13.0
4	40.0	0.3	23.5	60.0	0.3	24.0	26.7	1.3	42.0	80.0	1.3	18.0
	36.7	0.0	24.0	56.7	0.0	26.0	23.3	0.7	44.0	80.0	0.0	20.0
5	43.3	0.3	60.0	66.7	0.3	42.0	36.7	1.0	64.0	80.0	0.7	37.5
	43.3	0.0	62.0	60.0	0.0	45.0	30.0	0.7	65.0	80.0	0.0	40.0

Table 6.6: Performance comparison of the DTS and MAT filters in capturing *true* object motion vectors for the *Table Tennis* sequence.



(a) 0 iteration (No MAT) (b) 1 iteration (c) 2 iterations

(d) 3 iterations (e) 4 iterations (f) 5 iterations

Fig. 6.12: The *true* object motion vectors captured by the DTS algorithm with, and without, the MAT filter.

The maximum percentage of *true* motion vectors captured by all BMAs using the MAT filter over 5 iterations is shown in Fig. 6.13. It can be concluded that the DTS algorithm in conjunction with the MAT filter significantly outperformed the FS, TSS, and NTSS algorithms by capturing 36.7%, 50%, and 20% more *true* motion vectors respectively, with all *false* motion vectors being eliminated. It can also be concluded that the MAT filter improved not only the performance of the DTS algorithm, but also the performance of the FS, TSS, and NTSS algorithms by capturing 30.3%, 30%, and 60% more *true* motion vectors, respectively.

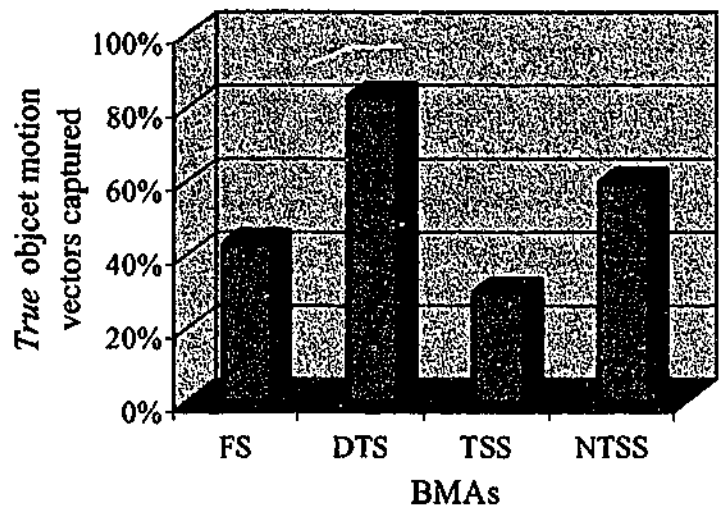


Fig. 6.13: Maximum percentage of *true* object motion captured by different BMAs in conjunction with the MAT filter for the *Table Tennis* sequence, over a maximum of 5 iterations.

The next experiment was conducted by considering the pair of frames, #8 and #9, of the *Foreman* sequence in Fig. 6.11. The experimental results are given in Table 6.7. Detailed supplementary results have also been included in Appendix F. To analyse the effect of the MAT filter, the experimental results, from 0 to 5 iterations, are given in Table 6.7. Again the improved percentage of *true* motion vectors captured for the FS, NTSS, TSS, and DTS algorithms was 41.3%, 19.6%, 6.5%, and 8.7%, respectively, from iteration 3 to 4, whereas it was only 3.1%, 2.2%, 34.8%, and 2.1% from iteration 4 to 5, so only results pertaining to 0 to 5 iterations are included.

Table 6.7 also shows that by using any threshold value without the MAT filter, no algorithm could eliminate the *false* motion vectors without also eliminating the *true* motion vectors.

The percentage of *true* motion vectors captured is significantly increased when the MAT filter is combined with a BMA. For example, the *true* motion vectors captured by the FS algorithm increased from 0% to 77% using the MAT filter over 5 iterations. A similar trend was found for the NTSS, TSS, and DTS algorithms. These results, again, endorse the effectiveness of the MAT filter in eliminating *false* vectors while retaining the *true* object motion vectors.

BMAs / No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0 (No MAT)	4.6	13.2	3.0	2.2	1.9	6.0	2.2	3.8	5.6	4.3	3.8	2.6
	0.0	11.3	3.2	0.0	1.9	6.2	0.0	3.8	5.8	0	1.9	2.8
1	4.4	11.3	4.0	2.2	1.9	7.0	2.2	3.8	5.6	13.0	1.9	3.0
	0.0	11.3	4.2	2.2	0.0	7.2	0.0	3.8	5.8	10.9	0.0	3.2
2	4.4	1.9	7.4	13.0	1.9	7.0	2.2	1.9	8.6	43.5	1.9	3.6
	0.0	1.9	7.6	10.9	0.0	7.2	0.0	1.9	8.8	39.1	0.0	3.8
3	37.0	1.9	10.0	58.7	1.9	8.5	2.2	1.9	15.0	71.7	1.9	4.8
	32.6	0.0	10.2	56.5	0.0	8.8	0.0	1.9	15.5	69.6	0.0	5.0
4	73.9	1.9	10.5	78.3	1.9	9.6	13.0	1.9	25.0	78.3	1.9	7.6
	73.9	0.0	11.0	76.1	0.0	9.9	6.5	0.0	26.0	78.3	0.0	7.8
5	76.1	1.9	17.5	80.4	1.89	16.0	43.5	1.9	33.0	80.4	1.9	13.0
	77.0	0.0	18.0	78.3	0.00	16.5	41.3	0.0	34.5	80.4	0.0	13.2

Table 6.7: Performance comparison of the DTS and MAT filter in capturing the *true* object motion vectors for the *Foreman* video sequence.

The maximum percentage of *true* motion vectors captured by the different BMAs in conjunction with the MAT filter over 5 iterations, is shown in Fig. 6.14. It demonstrates that the DTS algorithm with the MAT filter outperformed the FS, TSS, and NTSS algorithms by capturing 3.4%, 39.1%, and 2.1% more *true* motion vectors, respectively, with all *false* motion vectors being eliminated. Hence the MAT filter improved not only the performance of the DTS algorithm but also the performance of the FS, TSS, and NTSS algorithms in capturing an improved number of *true* motion vectors.

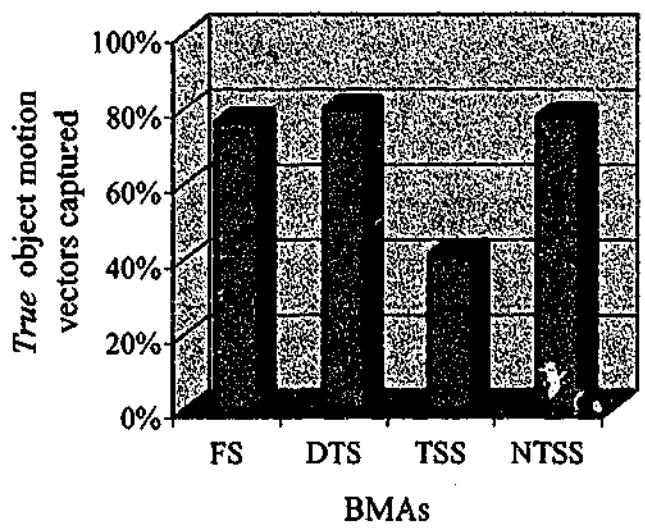


Fig. 6.14: Maximum percentage of *true* object motion captured by different BMAs in conjunction with the MAT filter for the *Foreman* sequence, over a maximum of 5 iterations.

In the above results, it is important to note that the use of any threshold value, without the MAT filter, could not eliminate all the *false* motion vectors even when all the *true* motion vectors were eliminated. The use of MAT filter, on the other hand, successfully removed all the *false* motion vectors, even when there were almost 80% *true* object motion vectors present. This result clearly supports the rationale behind the MAT filter design, being specifically, the removal of *false* motion vectors.

A third experiment was conducted involving the pair of frames #1 and #2 (Appendix F) from a non-standard video sequence, *Rocket*, to evaluate the performance of the DTS algorithm with the MAT filter for a high motion sequence. The experimental results are given in Table 6.8, this time however, for only 0 to 3 iterations. The number of iterations was based on experimental results where it was shown that the performance of most BMAs did not change significantly after two iterations. Detailed supplementary results have also been included in Appendix F.

Table 6.8 illustrates that the performance of the DTS algorithm in capturing *true* object motion vectors, without the use of the MAT filter (0 iteration), was almost 80% higher than the FS, NTSS, and TSS algorithms, indicating that the DTS algorithm outperformed other existing BMAs. It can also be observed that the capture of *true* motion vectors by FS, NTSS, DTS, and TSS was significantly increased when using the MAT filter to eliminate *false* motion vectors. Though the DTS algorithm captured 80% of *true* object motion vectors, without using the MAT filter, a further 8% improvement was obtained when the MAT filter was included for just 1 iteration.

BMAs / No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0 (No MAT)	3.9	4.6	9.0	3.85	2.3	9.0	3.9	2.3	9.0	84.6	9.1	3.5
	0.0	2.3	9.5	0.00	2.3	9.5	0.0	2.3	9.5	80.8	0.0	4.0
1	46.2	2.3	12.5	15.4	2.3	12.5	15.4	2.3	8.5	92.3	2.3	5.0
	30.8	0.0	13.0	3.9	0.0	13.0	7.7	0.0	9.0	88.5	0.0	5.5
2	57.7	2.3	20.5	34.6	2.3	18.0	42.3	2.3	15.0	73.1	2.3	10.5
	57.7	0.0	21.0	34.6	0.0	18.5	34.6	0.0	15.5	73.1	0.0	11.0
3	57.7	2.3	36.0	42.3	2.3	33.0	38.5	2.3	36.0	73.1	2.3	21.5
	57.7	0.0	37.0	34.6	0.0	34.0	34.6	0.0	37.0	73.1	0.0	22.0

Table 6.8: Performance comparison of the DTS and MAT filter in capturing *true* object motion vectors for the *Rocket* sequence.

The percentage of *true* motion vectors captured by the DTS algorithm decreased, however, during the second and third iterations. This is because each iteration of the MAT filter increases

the length of clustered *true* vectors at a higher rate than other vectors. Thus, each iteration improves the probability of its being able to separate the *true* from the *false* vectors when a suitable value of T_f is chosen. However, each iteration of the MAT filter also increases the length of *false* motion vectors in the vicinity of clustered *true* vectors at a comparable rate to the *true* vectors. This phenomenon leads to a decrease in the possibility of vector separation, and after a certain number of iterations, this decrease dominates the normal increase achieved in each iteration. Table 6.8 illustrates that the performance of any BMA was almost the same over 2 and 3 iterations, which implies that a higher number of iterations does not always improve the percentage of *true* motion vectors captured.

The highest percentage of *true* motion vectors captured by different search algorithms over 3 iterations is shown in Fig. 6.15. It demonstrates that the DTS algorithm in conjunction with the MAT filter significantly outperformed the FS, TSS, and NTSS algorithms by capturing 30%, 50%, and 50% more *true* motion vectors, respectively. It also indicates that the FS, TSS, and NTSS algorithms, with the MAT filter, subsequently improved their performance by capturing almost 60%, 23%, and 23% more *true* object motion vectors, respectively.

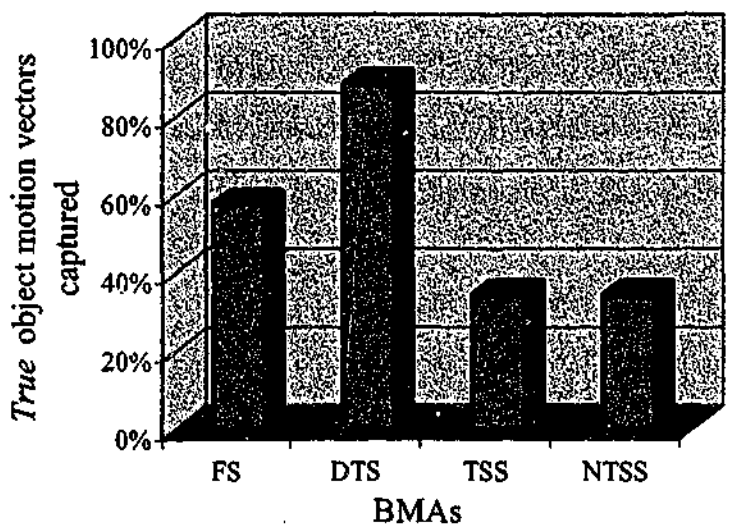


Fig. 6.15: Maximum percentage of *true* object motion captured by different BMAs in conjunction with the MAT filter for the *Rocket* sequence, over a maximum of 3 iterations.

Finally, the average maximum percentage of *true* motion vectors captured by the FS, DTS, NTSS, and TSS algorithms for *Table Tennis*, *Foreman*, and *Rocket* sequences has been plotted in Fig. 6.16. This clearly demonstrates that the performance of the DTS algorithm, used with the MAT filter, to capture *true* object motion vectors was considerably better than that of the FS, TSS, and NTSS algorithms. It is interesting to note that though the FS algorithm is the optimum algorithm for prediction image quality (Chapters 3, 4, and 5), its performance is not so satisfactory for capturing *true* object motion.

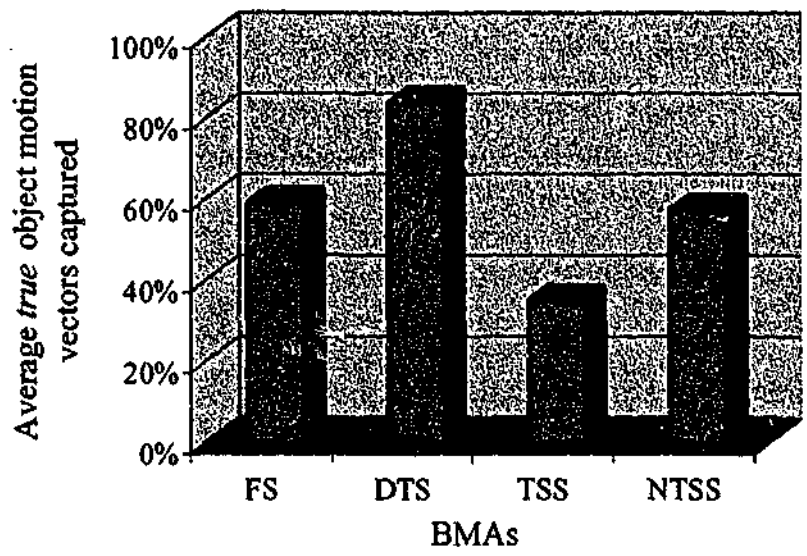


Fig. 6.16: Average percentage of *true* motion vectors captured by different BMAs for the *Table Tennis*, *Foreman*, and *Rocket* video sequences over a maximum of 5 iterations.

6.5.3 Computational Complexity Analysis

The computational complexity of the MAT filter depends on the kernel size used and the number of iterations involved. From (6.25), the total number of operations required for the MAT filter is $2\Re$ additions and two divisions for each iteration, where $\Re$ represents the kernel size. If the frame rate f = 30 fps, $\text{frame size} = [N_h, N_v]$, $\text{macroblock size} = [N, N]$, and the number of iterations is L, then the total number of operations per second required for the MAT filter is:

$$2(\Re+1)\times L\times\left(\frac{N_h\times N_v}{N^2}\right)\times f \tag{6.27}$$

Assume that the block distortion is measured using MAE, which requires three basic operations per pixel. If the frame rate is f = 30, frame size is $[N_h, N_v]$=[352,240], maximum displacement d = 7, and macroblock dimension N = 16, the number of integer arithmetic operations required for any BMA for motion estimation is bounded between 1.71 billion and 7.6 million per second using integer-pel accuracy (2.7).

Conversely, if a 3×3 sized kernel and, at most, 5 iterations are utilised, the total number of operations for the MAT filter is only 0.7 million per second. This overhead is not significant compared to the complexity involved in block motion estimation. These experimental results also prove that this overhead cost can be fully justified by the improvement in capturing a significant number of *true* motion vectors by elimination of *false* motion vectors from the motion vector field.

6.6 Summary

In this chapter, the superior performance of the DTS algorithm in capturing *true* object motion vectors compared to existing BMAs such as the exhaustive FS, and fast NTSS and TSS algorithms, has been proven, so confirming the argument made in Chapter 3 regarding the potential of the DTS algorithm.

A novel *Mean Accumulated Thresholded* (MAT) filter has been proposed and implemented for eliminating *false* motion vectors when capturing *true* object motion vectors obtained using any BMA for a video sequence. The effect of different kernel sizes and iterations of the MAT filter has also been analysed in detail, with smaller kernel sizes such as 3×3, or 5×5, not only exhibiting better performance than higher sized kernels, but also reducing the overhead cost for the filtering process. The experimental results have shown that for fewer than 5 iterations, it is possible to successfully eliminate all *false* vectors while retaining almost 80% of the *true* motion vectors captured by the DTS algorithm.

Experimental results clearly prove that although the DTS algorithm has outperformed other existing BMAs without the use of the MAT filter, the percentage of retained *true* object motion vectors while eliminating all *false* motion vectors, significantly increased with the use of the MAT filter. Nevertheless, although the performance of the DTS algorithm with the MAT filter has proven to be an effective and useful tool in *true* object motion estimation, some issues still need to be addressed in order to derive the full benefits of the MAT filter. This will constitute a potential future research direction.

This chapter also proposed a *Modified Iterative-Least-square Estimation* (MILSE) technique to calculate *global* motion parameters. The proposed technique is flexible enough for use with any number of blocks in a frame. Since in general, camera rotation is comparatively much less frequent than zooming or panning, it has not been considered in calculating *global* motion parameters. Experimental results show that the proposed MILSE technique has a similar performance compared to the traditional ILSE technique [130] while reducing computational cost involved in camera parameter estimation.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Block-based motion estimation represents one of the most powerful compression strategies in video coding. Among the different techniques, the best from a picture quality point of view is the *full search* (FS) algorithm, which guarantees optimal image quality, but at a very high computational cost. To reduce the computational complexity, a number of fast directionally-based BMAs have been developed. The fundamental drawback with existing BMAs such as TSS and NTSS, however, is that these algorithms do not provide any mechanism to support *Quality of Service* (QoS) in terms of either prediction quality or computational speed. This is a key issue in real-time video coding applications, especially for low bit-rate applications such as video over mobile and real-time software-only video encoding. Another drawback is that they have been designed solely for video coding where prediction error minimisation is the only criterion, irrespective of whether the motion vector indicates the direction of the *true* moving objects involved or not.

This thesis has directly addressed the above issues by presenting a flexible, generic, non-directional block-based motion estimation system that guarantees the achievement of a user-specified level of either prediction quality or computational speed for video coding applications. This system has also been proven to be very effective in capturing significantly more *true* object motion vectors than other approaches for object motion based video analysis applications. The system comprises the following constituent components: -

A new *Distance-dependent Thresholding Search* (DTS) block-based motion estimation algorithm, which has the advantage over other BMAs that it encompasses both the FS as well as fast searching modes. This unique feature provides a wide range of levels for performance scalability in motion estimation and QoS in terms of both predicted image quality and processing speed. This system also provides a general solution from high through very low bit-rate coding applications while the non-directional nature of the system means it does not suffer from any potential difficulties due to the *unimodal error surface assumption.*

The DTS algorithm, using a linear thresholding function, proved to be most effective in terms of flexibility and outperformed existing fast algorithms from a prediction image quality and search speed perspective for low motion video sequences.

Two limitations of the DTS algorithm however, were identified: its limited search efficiency for high motion video sequences and the need to manually define the threshold control parameter. To solve these issues, the *Adaptive-Centre* DTS (ACDTS) and *Adaptive-Centre Diamond Search* DTS (ACDSDTS) algorithms were developed. The ACDTS algorithm integrated spatial motion correlation of the neighbouring blocks' motion vectors with the DTS algorithm to improve the search efficiency by reducing the processing time while providing better prediction quality.

Implementing a diamond shaped instead of a general rectangular search pattern further enhanced the performance of the ACDTS algorithm. The subsequent *Adaptive-Centre Diamond Search DTS* (ACDSDTS) outperformed existing fast algorithms for all types of motion sequence, by being able to flexibly trade-off quality with computational complexity. The additional overhead cost incurred in the ACDSDTS algorithm was shown to be negligibly small compared to that required for the BDM calculation for motion estimation.

While the ACDSDTS algorithm enhanced the search efficiency of the DTS algorithm, to provide QoS, the threshold control parameter needed to be automatically set and adjusted depending on the video content and user demands. This was achieved in the *Fully Adaptive Distance-dependent Thresholding Search* (FADTS) block motion estimation algorithm which satisfied any level of user demand in terms of prediction image quality as well as processing speed. This system feature clearly provided flexibility in performance management in motion estimation, which is the crucial issue in real-time software-only or low power video coding applications. This system also had superior performance in terms of both prediction image quality and search speed compared to existing fast algorithms. The computational overhead cost associated with adapting process proved to be negligible compared to the computational cost involved in the BDM calculation. As shot changes in a video sequence are used as reference frames for the adaptation process, the choice of the initial value of the threshold control parameter impacted significantly on performance. To incorporate a shot change in the FADTS algorithm, an integrated shot detection (camera break) technique, based on an *artificial neural network* (ANN) has also been presented for non-real-time video coding, while for real-time processing, the FADTS algorithm employed an inexpensive strategy of detecting abrupt changes in the BDM to approximate possible shot changes.

Finally, the thesis explored extending the basic DTS algorithm into the burgeoning area of *true* object motion estimation. In the presence of camera motion, *global* motion compensation

was successfully applied to eliminate any effect of this motion from the *true* object's motion. A *Modified Iterative Least-Square Estimation* (MILSE) technique was implemented which reduced the computational cost for *global* motion estimation compared with the conventional *Iterative Least-Square Estimation* (ILSE) technique. The DTS algorithm, then in combination with a novel *Mean Accumulated Thresholded* (MAT) filter, provided a very powerful tool for block-based *true* object motion estimation, and significantly outperformed existing BMAs. The MAT filter design was also shown to be both very flexible and general enough that it could be combined with other BMAs to enhance their respective performances as well, in capturing more *true* object motion vectors.

In summary, this thesis has presented a generic distance-dependent thresholding block-based motion estimation system which has proved not only effective in providing performance scalability and QoS in motion estimation for video coding applications, but also for *true* object motion estimation in object motion based video representation.

7.2 Future Work

There are a number of potential areas where the research findings can be extended: -

1. Rate-complexity-distortion [59] of the encoding process is a challenging and relatively recent research direction for real-time video coding applications, especially for software-only or low power video encoding (mobile or handheld computing platforms). The performance of the video CODEC in these applications is often limited by available processing power and bandwidth. As motion estimation is the most costly part of an encoder, it requires variable complexity motion estimation. Therefore, an important extension of this work would be to implement the flexible FADTS algorithm to optimise the three parameters for coding applications.
2. The performance of the DTS algorithm depends on predefined threshold values, which have been controlled by a control parameter. For this threshold so far, the linear and exponential functions have been considered, though this threshold function can be any other complex form. Possible future research work could include investigating the impact of other complex functions on the performance of the DTS algorithm.
3. The effectiveness of the MAT filter in *false* motion vector elimination in improving the performance of the DTS algorithm in capturing *true* object motion has opened a potential new research direction in block-based true object motion estimation. Future research could extend to optimising the performance of the MAT filter by analysing the impact of video content on different design parameters. This area also includes the implementation of this filter for motion field smoothing purposes.

Bibliography

[1] G. Sorwar, M. Murshed, and L. Dooley, "Distance dependent thresholding search for fast motion estimation in real world video coding application," In the Proceedings of IEEE Asia-Pacific Conference on Circuits and System (APCCAS'02), Kartika Plaza Hotel, Bali, Indonesia, vol. 2, pp. 519-524, 2002.

[2] G. Sorwar, M. Murshed, and L. Dooley, "Modified full-search block-based motion estimation algorithm with distance dependent thresholds," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'02), Orlando, Florida, USA, vol. 4, pp. 4189-4189, 2002.

[3] G. Sorwar, M. Murshed, and L. Dooley, "Fast block-based true motion estimation using distance dependent thresholds (DTS)," In the Proceedings of the 6th International Conference on Signal Processing (ICSP'02), Beijing, China, vol. 2, pp. 937-940, 2002.

[4] G. Sorwar, M. Murshed, and L. Dooley, "Block-based true motion estimation using distance dependent thresholded search," In the Proceedings of the of ISCA 14th International Conference on Computer Applications in Industry and Engineering (CAINE'01), Las Vegas, USA, pp. 45-48, 2001.

[5] G. Sorwar, M. Murshed, and L. Dooley, "A fully adaptive performance-scalable distance-dependent thresholding search algorithm for video coding," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03), Hong Kong, 6-10 April 2003.

[6] G. Sorwar, L. Dooley, and M. Murshed, "Integrated technique with neurocomputing for temporal video segmentation," In the Proceedings of the First International Workshop on Hybrid Intelligent Systems (HIS'01), Adelaide, South Australia, pp. 159-167, 2001.

[7] G. Sorwar, M. Murshed, and L. Dooley, "A novel filter for block based object motion

estimation," In the Proceedings of the International Conference on Digital Image Computing Techniques and Applications (DICTA'02), Melbourne, Australia, pp. 201-206, 2002.

[8] D. H. Ballard and C. M. Brown, *Computer Vision*: Prentice-Hall, 1982.

[9] ISO/IEC JTC1/SC29/WG11, "Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit," ISO CD 11172-32, 1993.

[10] ISO/IEC JTC1/SC29/WG11, "Generic Coding of moving pictures and associated audio," ISO/IEC 13818-2, 1993.

[11] ITUT-T Recommendation H.261, "Video codec for audiovisual services at p´64 kbit/s," version 3, 1993.

[12] Draft ITU-T Recommendation H.263, "Video coding for low bit-rate communication," 1996.

[13] ISO/IEC JTC1/SC29/WG11 N4668, "Overview of the MPEG-4 Standard," International Organisation For Standardisation Organisation Internationale DeNormalisation ISO/IEC JTC1/SC29/WG11 Coding Of Moving Pictures And Audio, March 2002.

[14] F. Idris and S. Panchanathan, "Review of image and video indexing techniques," *Journal of Visual Communication & Image Representation*, vol. 8, pp. 146-166, 1997.

[15] E. Ardizzone and M. L. Cascia, "Video indexing using optical flow field," In the Proceedings of IEEE International Conference on Image Processing (ICIP'96), Lausanne, Switzerland, vol. 3, pp. 831-834, 1996.

[16] M. L. Cascia and E. Ardizzone, "JACOB: just a content-based query system for video databases," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP'96), Atlanta, GA, USA, vol. 2, pp. 1216-1219, 1996.

[17] J. -G. Kim, H. S. Chang, J. Kim, and H. -M. Kim, "Efficient camera motion characterization for MPEG video indexing," In the Proceedings of IEEE International Conference on Multimedia and Expo, New York, NY, USA, vol. 2, pp. 1171-1174, 2000.

[18] S. -Y. Kim and Y. M. Ro, "Fast content-based MPEG video indexing using object motion," In the Proceedings of IEEE Region 10 Conference, Cheju Island, South Korea, vol. 2, pp. 1506-1509, 1999.

[19] Z. Aghbari, K. Kaneko, and A. Makinouchi, "A motion-location based indexing method for retrieving MPEG videos," In the Proceedings of Ninth International Workshop on

Database and Expert Systems Applications, Vienna, Austria, pp. 102-107, 1998.

[20] A. Yoneyama, Y. Nakajima, H. Yanagihara, and M. Sugano, "Moving object detection and identification from MPEG coded data," In the Proceedings of IEEE International Conference on Image Processing (ICIP'99), Kobe, Japan, vol. 2, pp. 934-938, 1999.

[21] R. Milanese, F. Deguillaume, and A. Jacot-Descombes, "Efficient segmentation and camera motion indexing of compressed video," *Real-Time Imaging*, vol. 5, no. 4, pp. 231-241, 1999.

[22] E. Ardizzone, M. L. Cascia, A. Avanzato, and A. Bruna, "Video indexing using MPEG motion compensation vectors," In the Proceedings of IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, vol. 2, pp. 725-729, 1999.

[23] H. Zen, T. Hasegawa, and S. Ozawa, "Moving object detection from MPEG coded picture," In the Proceedings of International Conference on Image Processing (ICIP'99), Kobe, Japan, vol. 4, pp. 25-29, 1999.

[24] J. Heuer and A. Kaup, "Global motion estimation in image sequences using robust motion vector field segmentation," In the Proceedings of ACM Multimedia, Orlando, FL, USA, pp. 261-264, 1999.

[25] B. K. P. Horn, *Robot Vision*: MIT Press, Cambridge, MA, USA, 1986.

[26] D. Nair and J. K. Aggarwal, "Moving obstacle detection from a navigating robot," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 404-416, 1998.

[27] C. Collet, A. Quinquis, and J. M. Boucher, "Cloudy sky velocity estimation based on optical flow estimation leading with an entropy criterion," In the Proceedings of 11th IAPR International Conference on Pattern Recognition, Hague, Netherlands, pp. 160-163, 1992.

[28] J. L. Prince and E. R. McVeigh, "Motion estimation from tagged MR image sequences," *IEEE Transactions on Medical Imaging*, vol. 11, pp. 238-249, 1992.

[29] G. Funka-Lea and A. Gupta, "The use of hybrid models to recover cardiac wall motion in tagged MR images," In the Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Los Alamitos, CA, USA, pp. 625-630, 1996.

[30] J. U. Quistgaard, "Signal acquisition and processing in medical diagnostic ultrasound," *IEEE Signal Processing Magazine*, vol. 14, pp. 67-74, 1997.

[31] E. Sahouria and A. Zakhor, "Motion Indexing of video," In the Proceedings of

International Conference on Image Processing (ICIP'97), Santa Barbara, CA, USA, vol. 2, pp. 526-529, 1997.

[32] M. Irani and S. Peleg, "Motion analysis for image enhancement: resolution, occlusion, and transparency," *Journal of Visual Communication & Image Representation*, vol. 4, pp. 324-335, 1993.

[33] T. Vlachos and G. Thomas, "Motion estimation for the correction of twin-lens telecine flicker," In the Proceedings of International Conference on Image Processing (ICIP'96), Lausanne, Switzerland, vol. 1, pp. 109-112, 1996.

[34] J. K. Aggarwal and N. Nandhakumar, "On the computation of motion from sequences of images-A review," *Proceedings of the IEEE*, vol. 76, pp. 917-935, 1988.

[35] A. Mitiche and P. Bouthemy, "Computation and analysis of image motion: a synopsis of current problems and methods," *International Journal of Computer Vision*, vol. 19, pp. 29-55, 1996.

[36] J. R. Jain and A. K. Jain, "Displacement measurement and its application in inter frame image coding," *IEEE Transactions on Communications*, vol. 29, pp. 1799-1808, 1984.

[37] K. P. Horn and B. G. Schunck, "Determining Optical flow," *Artificial Intelligence*, vol. 17, pp. 185-203, 1981.

[38] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," In the Proceedings of DARPA Image Understanding Workshop, pp. 121-130, 1981.

[39] H. -H. Nagel, "Displacement vectors derived from second-order intensity variations in image sequences," *Computer Graphics and Image Processing*, vol. 21, pp. 85-117, 1983.

[40] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, pp. 43-77, 1994.

[41] D. R. Walker and K. R. Rao, "Improved pel-recursive motion compensation," *IEEE Transactions on Communications*, vol. COM-32, pp. 1128-1134, 1984.

[42] J. D. Robbins and A. N. Netravali, "Recursive motion compensation: a review," *Image Sequence Processing And Dynamic Sconce Analysis*, Springer-Verlag, pp. 76-103, 1983.

[43] L. Boroczky, "Pel-Recursive Motion Estimation for Image Coding," Edlft University of Technology, 1991.

[44] T. Koga, K. Linuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated inter frame coding for videoconferencing," In the Proceedings of IEEE National Telecommunication Conference, New Orleans, LA, USA, pp. G5.3.1-G.5.3.5, 1981.

[45] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 4, pp. 438-442, 1994.

[46] L.-M. Po and W. -C. Ma, "Novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 6, pp. 313-317, 1996.

[47] H. Nisar and T. -S. Choi, "An advanced centre biased search algorithm for motion estimation," In the Proceedings of International Conference on Image Processing (ICIP'00), Vancouver, BC, Canada, vol. 1, pp. 832-835, 2000.

[48] M. Ghanbari, "The cross-search algorithm for motion estimation (image coding)," *IEEE Transactions on Communications*, vol. 38, pp. 950-953, 1990.

[49] J. -B. Xu, L.-M. Po, and C. -K. Cheung, "Adaptive motion tracking block matching algorithms for video coding," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 9, pp. 1025-1029, 1999.

[50] A. Puri, H. M. Hang, and D. L. Schilling, "An efficient block matching algorithm for motion-compensated coding," International Conference on Acoustics, Speech, and Signal Processing (ICASSP'87), Dallas, TX, USA, vol. 2, pp. 1063-1066, 1987.

[51] J. Lu and M. L. Liou, "A simple and efficient search algorithm for block-matching motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 7, pp. 429-433, 1997.

[52] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 6, pp. 419-422, 1996.

[53] S. Zhu and K. -K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, pp. 287-290, 2000.

[54] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 3, pp. 148-157, 1993.

[55] J. K. Kearney, W. B. Thomson, and D. L. Boley, "Optical flow estimation: an error analysis of gradient-based methods with local optimisation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. PAMI-9, pp. 229-244, 1987.

[56] A. M. Tekalp, *Digital Video compression*: Prentice Hall PTR, 1995.

[57] F. Dufaux and F. Moscheni, "Motion estimation techniques for digital TV: a review and a new contribution," *Proceedings of the IEEE*, vol. 83, pp. 858-876, 1995.

[58] Y. Q. shi and H. Sun, *Image and video compression for multimedia engineering: fundamentals, algorithms, and standards*: Boca Raton, Fla.; London: CRC Press, 2000.

[59] I. E. G. Richardson, *Video Codec Design*: John Wiley & sons, ltd, 2002.

[60] W. D. Tancharoen, S. Jitapunkul, P. Kittipanya-ngam, and N. Siritaranukul, "Video segmentation based on adaptive combination of multiple features according to MPEG-4," In the Proceedings of International Conference of SPIE- the International Society for Optical Engineering, Perth, WA, Australia, vol. 4067, pt. 1-3, pp. 1099-1106, 2000.

[61] S. Ji and H. W. Park, "Region-based video segmentation using DCT coefficients," In the Proceedings of International Conference on Image Processing (ICIP'99), Kobe, Japan, vol. 2, pp. 150-154, 1999.

[62] --------, "Moving-object segmentation with adaptive sprite for DCT-based video coder," In the Proceedings of International Conference on Image Processing (ICIP'01), Thessaloniki, Greece, vol. 3, pp. 566-569, 2001.

[63] M. Kunt, A. IKonomopoulos, and M. Kocher, "Second generation Image coding techniques," *Proceedings IEEE*, vol. 73, pp. 549-575, 1985.

[64] N. Cao and B. W. Y. Wei, "A 4:1 checker-board algorithm for motion estimation," In the Proceedings of International Conference of SPIE- the International Society for Optical Engineering, Denver, CO, USA, vol. 2847, pp. 408-412, 1997.

[65] G. Kummerfeldt, F. May, and W. Wolf, "Coding television signals at 320 and 64 kbits/s," International Conference of 2nd International technology Symposium on Optical and Electro-Optical Applied Science and Engineering, Cannes, 1985.

[66] R. Armanito and R. Schafer, "Motion vector estimation using spatio-temporal prediction and its application to video coding," In the Proceedings of International Conference of SPIE- Digital Video Compression, San Jose, CA, vol. 2668, pp. 290-301, 1996.

[67] C. -H. Hsieh, P. Lu, J. -S. Shyn, and E. -H. Lu, "Motion estimation algorithm using

interblock correlation," *Electronics Letters*, vol. 26, pp. 276-277, 1990.

[68] L. Luo, C. Zou, X. Gao, and H. Zhenya, "A new prediction search algorithm for block motion estimation in video coding," *IEEE Transaction on Consumer Electronics*, vol. 43, pp. 56-61, 1997.

[69] S. Jeannin and A. Divakaran, "MPEG-7 Visual Motion Descriptors," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 11, pp. 720-724, 2001.

[70] G. Sudhir and J. C. M. Lee, "Video annotation by motion interpretation using optical flow streams," *Journal of Visual Communication & Image Representation*, vol. 7, pp. 354-368, 1996.

[71] P. Bouthemy, M. Gelgon, and F. Ganansia, "A Unified Approach to Shot Change Detection and Camera Motion Characterization," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 9, pp. 1030-1044, 1999.

[72] Y. S. Avrithis, N. D. Doulamis, A. D. Doulamis, and S. D. Kollias, "Efficient content representation in MPEG video databases," In the Proceedings of IEEE Workshop on Content-Based Access of Image and Video Libraries, Santa Barbara, CA, USA, pp. 91-95, 1998.

[73] J. Feng, K. -T. Lo, H. Mehrpour, and A. E. Karbowiak, "Adaptive block matching algorithm for video compression," *IEE Proceedings: Vision, Image & Signal Processing*, vol. 145, pp. 173-178, 1998.

[74] D. -K. Lim and Y. -S. Ho, "A fast block matching motion estimation algorithm based on statistical properties of object displacement," In the Proceedings of IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and control, Piscataway, NJ, USA, vol. 1, pp. 138-141, 1998.

[75] A. N. Netravali and J. O. Limb, "Picture coding: a review," *Proceedings of the IEEE*, vol. 68, pp. 366-406, 1980.

[76] A. K. Jain, "Image data compression: a review," *Proceedings of the IEEE*, vol. 69, pp. 349-389, 1981.

[77] V. Bhaskaran and K. Konstantinides, *Image and video compression standards- algorithms and architectures*: Kluwer Academic Publishers, 1995.

[78] A. Gersho and R. Gray, *Vector Quantisation and Signal Compression*, Kluwer Academic, 1992.

[79] T. S. Huang and A. N. Netravali, "Motion and structure from feature correspondences: a review," *Proceedings of the IEEE*, vol. 82, pp. 252-268, 1994.

[80] B. K. P. Horn and E. J. Weldon, "Direct methods for recovering motion," *International Journal of Computer Vision*, vol. 2, pp. 51-77, 1988.

[81] Y. Liu and T. S. Huang, "Motion estimation from corner correspondences," In the Proceedings of International Conference on Image Processing (ICIP'89), Singapore, pp. 785-790, 1989.

[82] F. Moschetti, "A statistical approach to motion estimation," Ecole polytechnique Federale De Lausanne, PhD. Thesis, 2001.

[83] J. Y. Tham, A. A. Kassim, M. Ranganath, and S. Ranganath, "A novel unrestricted centre-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 8, pp. 369-377, 1998.

[84] K. Panusopone and D. M. Baylon, "An analysis and efficient implementation of half-pel motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 724-729, 2002.

[85] P. Y. S. Cheung, H. Y. Chung, and N. H. C. Yung, "Adaptive search centre non-linear three step search," In the Proceedings of International Conference on Image Processing (ICIP'98), Chicago, IL, USA, pp. 191-194, 1998.

[86] A. Puri, H. M. Hang, and D. L. Schilling, "Motion compensated transform coding based on block motion tracking algorithm," In the Proceedings of IEEE International Conference on Communications, Seattle, Washington, vol. 1, pp.136-140, 1987.

[87] K. C. Lai and S. C. Wong, "A fast motion estimation using a three-dimensional reference motion vector," International Conference on Acoustics, Speech, and Signal Processing (ICASSP'02), Orlando, Florida, USA, vol. 4, pp. 3429-3432, 2002.

[88] J. -H. Lim and H. -W. Choi, "Adaptive motion estimation algorithm using spatial and temporal correlation," In the Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Piscataway, NJ, USA, vol. 2, pp. 473-476, 2001.

[89] J. Feng, T. Y. Liu, K. T. Lo, and X. D. Zhang, "Adaptive motion tracking for fast block motion estimation," In the Proceedings of IEEE International Symposium on Circuits and Systems, Sydney, NSW, Australia, vol. 5, pp. 219-222, 2001.

[90] J. -Y. Nam, J. -S. Seo, J. -S. Kwak, M. -H. Lee, and H. H. Yeong, "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," *IEEE Trans. on Consumer Electronics*, vol. 46, pp. 934-942, 2000.

[91] M. Bierling, "Displacement estimation by hierarchical block matching," In the Proceedings of SPIE VCIP 1988, Cambridge, MA, 1988.

[92] T. -H. Han and S. H. Hwang, "A novel hierarchical-search block matching algorithm and VLSI architecture considering the spatial complexity of the macroblock," *IEEE Transactions on Consumer Electronics*, vol. 44, pp. 337-342, 1998.

[93] K. Lim and J. B. Ra, "Improved hierarchical search block matching algorithm by using multiple motion vector candidates," *Electronics Letters*, vol. 33, pp. 1771-1772, 1997.

[94] H. -C. Huang and Y. -P. Hung, "Adaptive early jump-out technique for fast motion estimation in video coding," *Graphical Models & Image Processing*, vol. 59, pp. 388-94, 1997.

[95] V. G. Moshnyaga, "A new computationally adaptive formulation of block-matching motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 11, pp. 118-124, 2001.

[96] C. -K. Cheung and L.-M. Po, "Normalized partial distortion search algorithm for block motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 10, pp. 417-422, 2000.

[97] C. -H. Cheung and L.-M. Po, "A fast block motion estimation using progressive partial distortion search," In the Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong, pp. 406-409, 2001.

[98] Y. -L. Chan, W. -L. Hui, and W. -C. Siu, "A block motion vector estimation using pattern based pixel decimation," In the Proceedings of IEEE International Symposium on Circuits and Systems, Hong Kong, vol. 2, pp. 1153-1156, 1997.

[99] H. -K. Chow and M. L. Liou, "Genetic motion search algorithm for video compression," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 3, pp. 440-445, 1993.

[100] F. -H. Cheng and S. -N. Sun, "New fast and efficient two-step search algorithm for block motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 9, pp. 977-983, 1999.

[101] H. -H. Hsieh and Y. -K. Lai, "A novel fast motion estimation algorithm using fixed subsampling pattern and multiple local winners search," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 241-244, 2001.

[102] M. -G. Liu and C. -H. Hou, "A fast block-matching motion estimation algorithm based on spatial-temporal motion vector correlation," In the Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong, pp. 498-501, 2001.

[103] C. Zhu, L.-P. Chau, and X. Lin, "Hexagon-based search pattern for fast block motion estimation, "*IEEE Transactions on Circuits & Systems for Video Technology*, vol. 12, pp. 349-355, 2002.

[104] A. M. Tourapis, O. C. Au, and M. L. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 12, pp. 934-947, 2002.

[105] B. Widrow and S. D. Stearns, *Adaptive Signal Processing,* NJ: Prentice Hall, Englewood Cliffs, 1985.

[106] G. B. Thomas, *Calculus and Analytic Geometry*, vol. 4th edition: Addison-Wesley, 1968.

[107] J. Y. Stein, *Digital signal processing: a computer science perspective*: New York: J. Wiley, 2000.

[108] N. J. Bershad, "Analysis of the normalized LMS algorithm with Gaussian inputs," *IEEE Transactions on Acoustics, Speech, & Signal Processing*, vol. 34, pp. 793-806, 1986.

[109] S. C. Douglas, "A family of normalized LMS algorithms," *IEEE Signal Processing Letters*, vol. 1, pp. 49-51, 1994.

[110] Im Sungbin, "A normalized block LMS algorithm for frequency-domain Volterra filters," In the Proceedings of the IEEE Signal Processing Workshop on Higher-Order Statistics, IEEE Computer. Society, Banff, Alta., Canada, pp. 152-156, 1997.

[111] E. Eweda and O. Macchi, "Convergence of the RLS and LMS adaptive filters," *IEEE Transactions on Circuits and Systems*, vol. 34, pp. 799 -803, 1987.

[112] G. Davenport, T. A. Smith, and N. Pincever, "Cinematic primitives for multimedia," *IEEE Computer Graphics & Applications*, vol. 11, pp. 67-74, 1991.

[113] J. S. Boreczky and L. A. Rowe, "Comparison of Video Shot Boundary Detection Techniques," *Journal of Electronic Imaging,* vol. *5,* pp. 122-128, 1996.

[114] Y. Rui, T. S. Huang, and S. Mehrotra, "Constructing Table-of-Content for Videos," *ACM Multimedia Systems Journal, Special Issue Multimedia Systems on Video Libraries*, vol. 7, pp. 359-368, 1999.

[115] H. -J. Zhang, A. Kankanhalli, and S. W. Smoliar, "Automatic Partitioning of Full-Motion Video," *Multimedia Systems*, vol. 1, pp. 10-28, 1993.

[116] K. Otsuji and Y. Tonomura, "Projection detecting Filter for Video Cut detection," In the Proceedings of ACM Multimedia, Anaheim, New York, NY, USA, pp. 251-256, 1993.

[117] A. Hampapur, R. Jain, and T. Weymouth, "Digital Video segmentation," In the Proceedings of ACM Multimedia'94, San Francisco, CA, USA, pp. 357-364, 1994.

[118] B. Shahraray, "Scene change detection and content-based sampling of video sequence," In the Proceedings of International Conference of SPIE- digital Video Compression, vol. 2419, pp. 2-13, 1995.

[119] R. Kasturi and R. Jain, *Computer vision: advances and applications*, Los Alamitos, California, IEEE Computer Society Press, 1991.

[120] H. Zhang, J. Wu, D. Zhong, and S. W. Smoliar, "An integrated system for content-based video retrieval and browsing," *Pattern Recognition*, vol. 30, pp. 643-658, 1997.

[121] H. Ueda, T. Miyatake, and S. Yoshizawa, "Impact: An Interactive Natural-motion picture dedicated Multimedia Authoring systems," In the Proceedings of the CHI'91, pp. 343-450 1991.

[122] Nagasaka and Y. Tanaka, " Automatic Video indexing and Full-video Search for objects Appearances," In the Proceedings of IFIP 2nd Conference on Visual; database Systems, Budapest, pp. 113-127, 1991.

[123] F. Arman, A. Hsu, and M. -Y. Chiu, "Feature management for large video databases," In the Proceedings of International Conference of SPIE - storage and retrieval for image and video database, San Jose, CA, USA, vol. 1908, pp. 2-12, 1993.

[124] --------, "Image Processing on Encoded Video Sequences," *Multimedia Systems*, vol. 1, pp. 211-219, 1994.

[125] H. –C. H. Liu and G. L. Zick, "Scene decomposition of MPEG compressed video sequence." In the Proceedings of International Conference of SPIE- digital Video Compression: Algorithms Technology, vol. 2419, pp. 14-25, 1995.

[126] G. K. Wallace, "The JPEG still picture compression standard" Communications of *ACM*

Multimedia Systems Journal, Special Issue Multimedia Systems on Video Libraries, vol. 34, pp. 30-44, 1991.

[127] Y. Deng and B. S. Manjunath, "Content-based search of video using colour, texture, and motion," In the Proceedings of International Conference on Image Processing (ICIP'97), Santa Barbara, CA, USA, vol. 2, pp. 534-537, 1997.

[128] J. M. Zurada, *Introduction to artificial neural systems*: PWS Pub Co, 1992.

[129] K. Meghriche, S. Femmam, B. Derras, and M. Haddadi, "A non linear adaptive filter for digital data communication," In the Proceedings of the Sixth Int. Symposium on Signal Processing and its Applications, Piscataway, NJ, USA, vol. 1, pp. 304-306, 2001.

[130] G. B. Rath and A. Makur, "Iterative least squares and compression based estimations for a four-parameter linear global motion model and global motion compensation," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 9, pp. 1075-1099, 1999.

[131] E. Arman, R. Depommier, A. Hsu, and M. Y. Chiu, "Content-based browsing of video sequences" In the Proceedings of ACM Multimedia'94, New York, NY, USA, pp. 97-103, 1994.

[132] Y. Rui, T. S. Huang, and S. Mehrotra, "Exploring video structure beyond the shots," In the Proceedings of IEEE International Conference on Multimedia Computing and Systems, Los Alamitos, CA, USA, pp. 237-240, 1998.

[133] F. Idris and S. Panchanathan, "Review of image and video indexing techniques," *Journal of Visual Communication & Image Representation*, vol. 8, pp. 146-166, 1997.

[134] N. H. AbouGhazaleh, "Compressed video indexing based on object motion," In the Proceedings of International Conference of SPIE-International Society of Optical Engineering, Perth, WA, Australia, vol. 4067, pp. 986-993, 2000.

[135] D. Adolph and R. Buschmann, "1.15 Mbit/s coding of video signals including global motion compensation," *Signal Processing: Image Communication*, vol. 3, pp. 259-274, 1991.

[136] Y. T. Tse and R. L. Baker, "Global zoom/pan estimation and compensation for video compression," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'91), vol. 4, pp. 2725-2728, 1991.

[137] H. Jozawa, K. Kamikura, A. Sagata, H. Kotera, and H. Watanabe, "Two-stage motion compensation using adaptive global MC and local affine MC," *IEEE Transactions on*

Circuits & Systems for Video Technology, vol. 7, pp. 75-85, 1997.

[138] S. F. Wu and J. Kittler, "A differential method for simultaneous estimation of rotation, change of scale and translation," *Signal Processing: Image Communication*, vol. 2, pp. 69-80, 1990.

[139] F. Moscheni, F. Dufaux, and M. Kunt, "A new two-stage global/local motion estimation based on a background/foreground segmentation," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'95), New York, NY, USA, vol. 4, pp. 2261-2264, 1995.

[140] M. Etoh and T. Ankei, "Parametrized block correaltion2D parametric motion estimation for global motion compensation and video mosaicing," In the Proceedings of IEICE TR PRMU97, 1997.

[141] M. Hoetter, "Differential estimation of the global motion parameters Zoom and pan," *Signal Processing: Image Communications*, vol. 16, pp. 249-265, 1989.

[142] F. Dufaux and J. Konrad, "Efficient, robust, and fast global motion estimation for video coding," *IEEE Transactions on Image Processing*, vol. 9, pp. 497-501, 2000.

[143] A. Zakhor and F. Lari, "Edge-based 3-D camera motion estimation with application to video coding," *IEEE Transactions of Image Processing*, vol. 2, pp. 481-498, 1993.

[144] C. -T. Hsu and Y. -C. Tsan, "Mosaics of video sequences with moving objects," In the Proceedings of International Conference on Image Processing (ICIP'01), Thessaloniki, Greece, vol.2, pp. 387-390, 2001.

[145] Y. -P. Tan, S. R. Kulkarni, and P. J. Ramadge, "A new method for camera motion parameter estimation," In the Proceedings of International Conference on Image Processing (ICIP'95), Los Alamitos, CA, USA, vol. 1, pp. 406-409, 1995.

[146] R. Wang and T. Huang, "Fast camera motion analysis in MPEG domain," In the Proceedings of International Conference on Image Processing (ICIP'99), Kobe, Japan, vol. 3, pp. 691-694, 1999.

[147] T. Vlachos, "Simple method for estimation of global motion parameters using sparse translational motion vector fields," *Electronics Letters*, vol. 34, pp. 60-62, 1998.

[148] D. Wang and L. Wang, "Fast and robust algorithm for global motion estimation," In the Proceedings of International Conference of SPIE-International Society of Optical Engineering, San Jose, CA, USA, vol. 3024, pp. 1144-1151, 1997.

[149] K. Kamikura and H. Watanabe, "Global motion compensation in video coding," *Electronics & Communications in Japan*, vol. 78, pp. 91-102, 1995.

[150] J. Meng and S. -F. Chang, "Tools for compressed-domain video indexing and editing," In the Proceedings of International Conference of SPIE-International Society of Optical Engineering, San Jose, CA, USA, vol. 2670, pp. 180-191, 1996.

[151] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannary, *Numerical Recipes in C*: Cambridge University Press, 1992.

[152] Z. Zhang, "Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting," *Image and Vision Computing Journal*, vol. 15, pp. 59-76, 1997.

[153] D. R. K. Brownrigg, "The weighted median filter," *Communications of the ACM*, vol. 27, pp. 807-818, 1984.

[154] E. Davies, *Machine Vision: Theory, Algorithms and Practicalities*: Academic Press, 1990.

[155] D. Vernon, *Machine Vision*: Prentice-Hall, 1991.

[156] J. S. Kim and H. W. Park, "Adaptive 3D median filtering for restoration of an image sequence corrupted by impulse noise," *Signal Processing: Image Communications*, vol. 16, pp. 657-68, 2001.

[157] I. Pitas and A. N. Venetsanopoulos, *Nonlinear digital filters*, Kluwer Academic Publishers, 1990.

[158] R. Boyle and R. Thomas, *Computer Vision: A First Course*: Blackwell Scientific Publications, 1988.

Appendices

Key Publications

During the PhD research, the following selected papers have been published from the thesis.

Refereed International Conferences

[1] G. Sorwar, M. Murshed, and L. Dooley, "Distance dependent thresholding search for fast motion estimation in real world video coding application," In the Proceedings of IEEE Asia-Pacific Conference on Circuits and System (APCCAS'02), Kartika Plaza Hotel, Bali, Indonesia, vol. 2, pp. 519-524, 2002.

[2] G. Sorwar, M. Murshed, and L. Dooley, "Modified full-search block-based motion estimation algorithm with distance dependent thresholds," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'02), Orlando, Florida, USA, vol. 4, pp. 4189-4189, 2002.

[3] G. Sorwar, M. Murshed, and L. Dooley, "Fast block-based true motion estimation using distance dependent thresholds (DTS)," In the Proceedings of the 6th International Conference on Signal Processing (ICSP'02), Beijing, China, vol. 2, pp. 937-940, 2002.

[4] G. Sorwar, M. Murshed, and L. Dooley, "Block-based true motion estimation using distance dependent thresholded search," In the Proceedings of the of ISCA 14th International Conference on Computer Applications in Industry and Engineering (CAINE'01), Las Vegas, USA, pp. 45-48, 2001.

[5] G. Sorwar, M. Murshed, and L. Dooley, "A fully adaptive performance-scalable distance-dependent thresholding search algorithm for video coding," In the Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03), Hong Kong, 6-10 April, 2003.

[6] G. Sorwar, L. Dooley, and M. Murshed, "Integrated technique with neurocomputing for

temporal video segmentation," In the Proceedings of the First International Workshop on Hybrid Intelligent Systems (HIS'01), Adelaide, South Australia, pp. 159-167, 2001.

[7] G. Sorwar, M. Murshed, and L. Dooley, "A novel filter for block based object motion estimation," In the Proceedings of the International Conference on Digital Image Computing Techniques and Applications (DICTA'02), Melbourne, Australia, pp. 201-206, 2002.

DISTANCE DEPENDENT THRESHOLDING SEARCH FOR FAST MOTION ESTIMATION IN REAL WORLD VIDEO CODING APPLICATION

Golam Sorwar, Manzur Murshed and Laurence Dooley
Gippsland School of Computing and Information Technology
Monash University, Churchill Vic 3842, Australia
[redacted]

ABSTRACT

This paper presents a *distance dependent thresholding search* (DTS) block motion estimation algorithm that employs the novel concept of distance dependent thresholds. The key feature of this algorithm is its flexibility with trading-off quality and complexity with threshold variation. Where as the performance of the existing algorithms is fixed in terms of prediction quality as well as complexity, DTS can be used as *full search* (FS) where high quality video entertainments require motion estimation with small prediction error, as well as fast motion estimation such as *three-step search* (TSS), *new-three-step search* (NTSS) etc while real-time video applications, such as the speed-oriented video conferencing require fast motion estimation with sacrificing quality. Experimental results show that this *distance dependent thresholding search* (DTS) algorithm also achieves better *peak signal-to-noise ratio* (PSNR), as well as lowerse arch times in comparison to both the TSS and NTSS algorithms.

1. INTRODUCTION

Motion estimation in image sequences has been a key element in a wide range of applications from computer vision through to popular video compression standards such as MPEG (Motion Picture Expert Group).

There are many motion estimation algorithms available including pel-recursive [3], block matching [4], and the optical flow based method, [5]. Amongst these techniques, the *block-matching algorithm* (BMA) is the most popular and is widely used in video coding standards such as MPEG-1/2 [1][2] and H.261/263 [7][8] due to its simplicity and also the superior performance [6] it exhibits for large pixel block displacements.

The exhaustive BMA, known as the *full search* (FS) algorithm, searches each candidate block for the closest match within the entire search region to minimize the *block-distortion measure* (BDM). The BDM of image blocks may be measured using various criteria such as, the *mean absolute error* (MAE), the *mean square error* (MSE), and the *matching pel-count* (MPC) [18]. Since FS method uses an exhaustive search to locate the minimum BDM for each candidate block, it provides good performance, but at the expense of a very high computational overhead. Motion estimation indeed is the major bottleneck in real-time video coding applications; hence the need for faster algorithms is obviously felt.

A number of efficient fast block motion estimation algorithms have been proposed. In particular, three categories of algorithms have been identified, which are characterised by the strategy adopted in order to speed up the search process. These are (i) limiting the number of search candidates, (ii) subsampling in them otion vector field, and (iii) subsampling in the spatial domain. The first class of fast algorithm includes the 2-D logarithmic search (2DLOG) [4], the three-step search (TSS) [9], the new three-step search (NTSS) [10], the four-step search (FSS) [11], the cross-search [12], and the prediction search algorithm [13] etc. All the aforementioned fast algorithms have been based upon the assumption that the BDM increases as the checking points move away from the global minima. However, these assumptions are reasonable for certain applications e.g., in video-conferencing, where the motion is neither very fast nor complicated. However, they are generally invalid for many real video sequences because of the highly non-stationary characteristics of the video signal. Moreover, the search directions of these algorithms can be ambiguous, leading to the MV becoming entrapped in a local minimum, with a resulting degradation in predictive performance.

An example of the second class of fast algorithms is the 2:1 motion field subsampling technique [14]. This technique is rarely used in isolation and is normally integrated together with other methods because it only gives a small speed-up ratio. Another problem is that the approach does not perform well for blocks containing small objects moving in different directions to neighbouring objects.

The third class refers to subsampling in the spatial domain [14][15]. A drawback of this approach however is that the reduction in search complexity is often inadequate for real-time application and is therefore difficult to embed within algorithms such as the TSS and NTSS.

In reality, the distortion of an object in a video frame is proportional to its velocity and therefore, as the length of a

0-7803-7690-0/02/$17.00 ©2002 IEEE

motion vector grows so does the block difference error. In [17][18], the authors addressed this issue by modifying the FS algorithm to incorporate *variable* distance dependent thresholds for fast and robust true motion vector estimation for object-based video indexing applications. In this paper, the principles are further extended so they can be applied in real time video coding. Compared with the TSS and NTSS algorithms, the proposed technique is more robust, since it visits all candidates around the centre tracing out a concentric-square arrangement, and hence reducing significantly the probability of being trapped in some local minima. The main strength of this algorithm is its flexibility with trading-off quality and complexity with threshold variation. Where as the performance of the existing algorithms is fixed in terms of prediction quality as well as complexity, DTS can be used as *full search* (FS) where high quality video entertainments require motion estimation with small prediction error, as well as fast motion estimation such as *three-step search* (TSS), *new-three-step search* (NTSS) etc while real-time video applications, such as the speed-oriented video conferencing require fast motion estimation with sacrificing quality. Experimental results show that this *distance dependent thresholding search* (DTS) algorithm also achieves better *peak signal-to-noise ratio* (PSNR), as well as lowerse arch times in comparison to both the TSS and NTSS algorithms.

The paper is organized as follows. The DTS algorithm is detailed in Section 2. Section 3 includes experimental results and analysis of the performance of the DTS algorithm, compared with other algorithms. Section 4 concludes the paper.

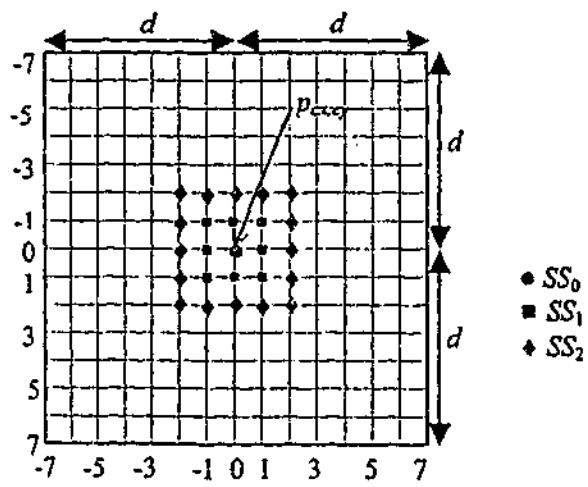


Fig. 1.DT S search squares.

2. THE DISTANCE DEPENDENT THRESHOLDING SEARCH (DTS) ALGORITHM

In the FS algorithm, the suitability of a matching block is measured based on the optimal (minimum)B DM. The FS algorithm works effectively when there is no distortion, but as alluded earlier, the level of distortion present in any video frame increases with the velocity of the moving objects and/or the zoom factor used for the camera. The exhaustive FS algorithm therefore becomes increasingly inefficient as the search trajectory, which is spiral in nature is traversed.

This paper proposes that the suitability measure of the FS algorithm is relaxed from the optimal criterion as the search trajectory moves from the centre, and becomes distance dependent. Locating a block with the minimum difference, but with a motion vector of high magnitude, is not only ineffectual in the prevailing distorted search space, but may lead to many "false" motion vectors being erroneously selected.

Definition 1: Search Squares *SS_i The search space with maximum displacement d, centred at pixel $p_{cx,cy}$, can be divided into d+1 mutually exclusive concentric search squares SS_i, for all $0 \le i \le d$, such that a checking point at pixel $p_{x,y}$ is $\in SS_k$ if and only if* $\max(|x-cx|,|y-cy|)=k$, *for all* $-d+cx \le x \le d+cx$ *and* $-d+cy \le y \le d+cy$.

The checking points used in the first three search squares SS_0, SS_1, and SS_2 are clearly shown in Fig. 1. From this figure it can be easily identified that

$$|SS_i| = \begin{cases} 1, & i=0 \\ 8i, & i=1,2,\ldots,d \end{cases} \quad (1)$$

2.1. The DTS Algorithm

- **Precondition:** Pixel $p_{cx,cy}$ is the centre of the search space with maximum displacement *d*.
- **Initialisation:**
  $MinMSE = MSE_{(cx,cy)}(0,0)$
  $MV = (0,0)$

Body:

```
If MinMSE > 0 Then
    For i = 1, 2, ..., d
        For each checking point p_{x,y} in SS_i
            e = MSE_{(cx,cy)}(x-cx, y-cy)
            If e < MinMSE Then
                MinMSE = e
                MV = (x-cx, y-cy)
        If sqrt(MinMSE) ≤ Threshold(i) Then Stop
```

- **Postcondition:** *MV* contains the motion vector and *MinMSE* contains the distortion error of the respective block.

Fig. 2. The DTS algorithm.

Threshold(*i*) in the DTS algorithm is a monotonically increasing function with respect to *i*, which can have a linear, exponential, or any other complex analytic form. In

[17][18] for example, a comparison was made on the performance of DTS algorithm using linear and exponential thresholding functions respectively. Experimental results confirmed that linear function consistently provided a better performance for a range of different types of video sequence. This was due to the fact that the distortion of an object in any frame tended to be directly proportional to its velocity, as well as the zoom factor of the camera. In the next section, we elaborate on the linear thresholding function.

2.2. Linear thresholding (LT) function

Let the centre of thesearc h region be at pixel $p_{cx,cy}$, which also defines the starting point of the search. In LT, the search will terminate at search square $SS_r$when:-

$$\sqrt{\mathrm{MSE}_{(cx,cy)}(x-cx, y-cy)} \le C_L \times r \qquad (2)$$

Assuming b-bit gray level intensity, the maximum valueo f theMS E is $(2^b\text{-}1)^2$, since thep ixel intensity is measured using 2^b levels. As SS_d is the outermost search square, an upper bound for the constant C_L can be set as:-

$$C_L < \frac{2^b}{d}. \qquad (3)$$

Note, that setting $C_L = 0$ in Eq. (2) transforms the DTS algorithm into the exhaustive FS algorithm. It is also clear that the search time required reduces as C_L increases. It is also interesting to note that if C_L is set higher than the above upper bound, the search will not explore the entire search area defined by the maximum displacement d.

3. EXPERIMENTAL RESULTS

The performance of the DTS algorithm for video coding was evaluated using the luminance (Y-component) signal of a number of standard test video sequences including "Tennis", "Flower Garden", "Football" "Salesman", "Foreman", "Carphone" and "Mis_America" (www-mugc.cc.monash.edu.au/~golam/). The results for the 80-frame "Tennis" sequence (SIF 352*240 pixel frame size) and "Salesman" (CIF 360*288 pixel frame size) are included in this paper. The "Tennis" sequence comprised various kinds of motions, including translation, zooming, and panning, while the "Salesman" sequence mainly consisted of low motion that was very similar to image sequences in low bit-rate video application such as videophone and videoconferencing.

In the experiments, all sequences were uniformly quantised to an 8-bit gray level intensity. The block size dimensions were $M = N = 16$ and $d = \pm 7$, i.e., each frame was divided into 16×16 pixel blocks and within each frame, a maximum of $(2d+1)^2 = 225$ checking points were used. The MSE measure was used as the criterion for locating the best motion vector for each block.

To quantitatively evaluate the video coding performance of the DTS algorithm, the following two measures were considered: -

i) The average peak signal-to-noise ratio (PSNR) after picture reconstruction.
ii) The average number of search points for computational complexity.

Table I: Avg. PSNR and avg. search points per motion vector for the "Tennis" sequences (1-80 Frames)

BMA	Tennis	
	PSNR $[dB]_{(avg)}$	Search $points_{(avg)}$
FS(±7)	27.94	196.98
LT(2)	27.63	75.68
LT(4)	27.48	38.14
LT(6)	27.27	26.03
LT(8)	26.98	20.35
LT(12)	26.42	14.65
LT(20)	25.68	11.14
TSS	26.14	23.01
NTSS	26.85	20.83

Table II: Avg. PSNR and avg.se arch points per motion vector for the "Salesman" sequences (1-80 Frames)

BMA	Tennis	
	PSNR $[dB]_{(avg)}$	Search $points_{(avg)}$
FS(±7)	35.52	204.17
LT(2)	35.51	34.65
LT(4)	35.47	14.46
LT(6)	35.43	10.37
LT(8)	35.38	9.24
LT(10)	35.34	8.84
LT(20)	35.13	8.26
TSS	35.23	23.21
NTSS	35.39	16.96

3.1. Peak Signal-to-Noise Ratio (PSNR) Results

The performance comparison of FS, TSS, NTSS and DTS using the LT function in terms of the average *Peak Signal-to-Noise Ratio* (PSNR), is given in Table I and Table II. It was observed that the PSNR value for the DTS algorithm improved by up to 0.85dB, when the number of search points was comparable with the TSS and NTSS algorithms, as for example in the "Tennis" LT(8) case. At higher search speeds, where the improvement factor was typically between 3 and 6, the average PSNR for the DTS

algorithm was still very close to the optimal average PSNR value of the FS algorithm. In contrast, the performance of the TSS and NTSS algorithms was significantly inferior, especially in respect of the fast motion segments involved in the "Tennis" video sequence, from frame 23 onwards. Being a directional search algorithm, TSS tended to converge to one of the local minima as explained in the introduction. The plot in Fig. 3 and 4 confirm the PSNR performance of the DTS algorithm against the FS, TSS and NTSS algorithms. Note for clarity, that only LT functions for the DTS algorithm that produced a comparable number of search points to the TSS and NTSS algorithms, have been included.

3.2. Search points results

The performance of the FS, TSS, NTSS and DTS algorithms in terms of the average number of search points to estimate the motion vectors is also presented in Table I and Table II. Again it is clear, that the DTS algorithm was faster by a factor of at least 3, and in the "Salesman" LT (6) case, more than 18 times, than the FS algorithm, while the PSNR remained comparable with the TSS and NTSS algorithms. The results also proved that by choosing a suitable constant for the selected threshold function, the average number of search points required by the DTS algorithm was considerably less, while concomitantly having a significantly higher average PSNR.

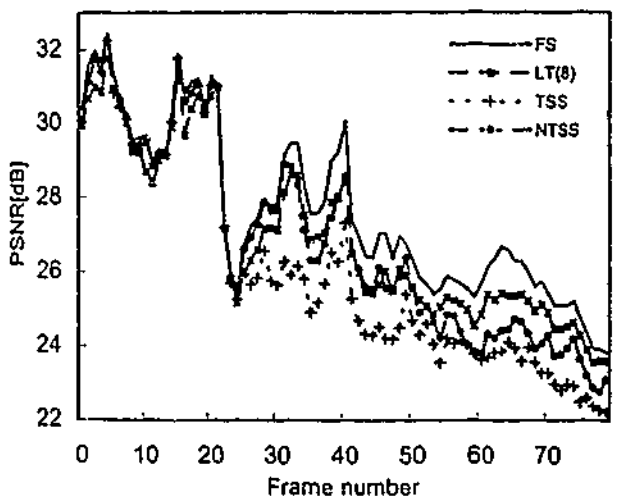


Fig. 3. PSNR Comparison of "Tennis" video sequence

It is interesting to note in Fig. 3, that during the first 23 frames of the "Tennis" sequence, which contain almost no camera motion, and only some object motion, there is considerable uniformity in the performance of all four search algorithms. In subsequent frames (24 to 80) however, where much faster object and camera motion is present, the DTS algorithm performs significantly better than the other two fast algorithms, while retaining the low number of search points.

Fig. 5 and 7 show the estimated 80th and 5th frame of "Tennis" and "Salesman" sequences with FS, DTS, TSS and NTSS algorithms respectively. As FS is optimum in terms of error performance, Fig. 6 and 8 show the prediction error distribution of DTS, TSS and NTSS with respect to FS algorithm. In terms of subjective image quality, the performance of DTS was very close to the FS algorithm and much better than both the TSS and NTSS.

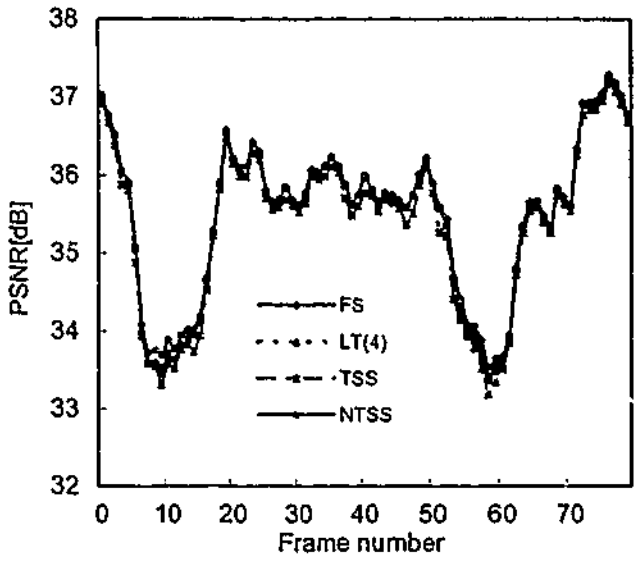


Fig. 4. PSNR Comparison of "Tennis" video sequence

Fig. 5 and 7 show the estimated 80th and 5th frame of "Tennis" and "Salesman" sequences with FS, DTS, TSS and NTSS algorithms respectively. As FS is optimum in terms of error performance, Fig. 6 and 8 show the prediction error distribution of DTS, TSS and NTSS with respect to FS algorithm. In terms of subjective image quality, the performance of DTS was very close to the FS algorithm and much better than both the TSS and NTSS.

The Table I and II clearly show the power of DTS in terms of its flexibility by trading of quality and complexity. It is also clear that the performance of FS, TSS and NTSS in terms of quality (PSNR) or computational times (avg. search point per motion vector) is fixed. On the other hand, DTS can be used as FS (with threshold constant 0) where high quality prediction required as well as faster even faster than TSS, NTSS (with higher threshold, say 10) with sacrificing quality.

4. CONCLUSIONS

This paper has presented a new distance dependent thresholding search (DTS) algorithm for block-based motion estimation in real-time video coding applications. The performance of DTS has been examined and proven that, in comparison to other popular fast algorithms such as, the three-step-search (TSS) and new three-step search (NTSS) algorithms, it provided comparable speed

performance, while retaining a distortion error similar to the minimum value produced by the optimal the full-search (FS) algorithm. In addition,th e DTS algorithm facilitated a flexibility that enabled a direct trade-off between PSNR and search speed for the entire range of threshold values. Automatically threshold adaptation for different level of performances is our on going work.

REFERENCES

[1] ISO/IEC JTC1/SC29/WG11, "Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit, ISO CD 11172-32.

[2] ISO/IEC JTC1/SC29/WG11, "Generic Coding of moving pictures and associated audio, ISO/IEC 13818-2, 1993.

[3] J.D. Robbins and A.N. Netravali, "Recursive motion compensation: a review," Image Sequence Processing And Dynamic Sconce Analysis, pp. 76-103, Springer-Verlag, 1983.

[4] J.R. Jain and A.K. Jain, "Displacement measurement and its application in inter frame image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799-1808, Dec. 1984.

[5] K.P. Horn and B.G. Schunck, "Determining Optical flow," *Artificial Intelligence*, vol. 17, pp. 185-203, 1981.

[6] F. Dufaux and F. Moscheni, "Motion estimation techniques for digital TV: A review and a new contribution," *Proc. IEEE*, vol. 83, pp. 858-876, June 1995.

[7] ITUT-T Recommendation H.261 version 3, "Video codec for audiovisual services at p×64 kbit/s", 1993

[8] Draft ITU-T Recommendation H.263, "Video coding for low bitrate communication", 1996.

[9] T. Koga, K. Iinuma, A. Hirano,Y. Iijima and T. Ishiguro, "Motion-compensated inter frame coding for videoconferencing," *in Proc. NTC81*, Nov. 1981, pp. G5. 3.1-G5.3.5.

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

[11] L. M. Po and W. C. Ma, "Novel four-step search algorithm for fast block motion estimation," *IEEE Trans. CSVT*, vol. 6, pp. 313-317, June, 1996.

[12] M. Ghanbari, "The cross-search algorithm for motion estimation (image coding)," *IEEE Trans. Commun.*, vol. 38, no. 7, pp. 950-953, July, 1990.

[13] J.-B. Xu, L.-M. Po, and C.-K. Cheung, "A new prediction model search algorithm for fast block motion estimation," *in Proc. ICIP*, 1997, vol.3, pp.610-13.

[14] B. Liu and Zaccarin, "A. New fast algorithms for the estimation of block motion vectors," *IEEE Trans. CSVT*, vol.3, no.2, pp.148-57, April 1993.

[15] N. Cao and W. BWY, "A 4:1 checkerboard algorithm for motion estimation," *SPIE-Int. Soc. Opt. Eng. Proc. of Spie - the Int. Society for Optical Eng.*, 1997, vol. 2847, pp. 408-12.

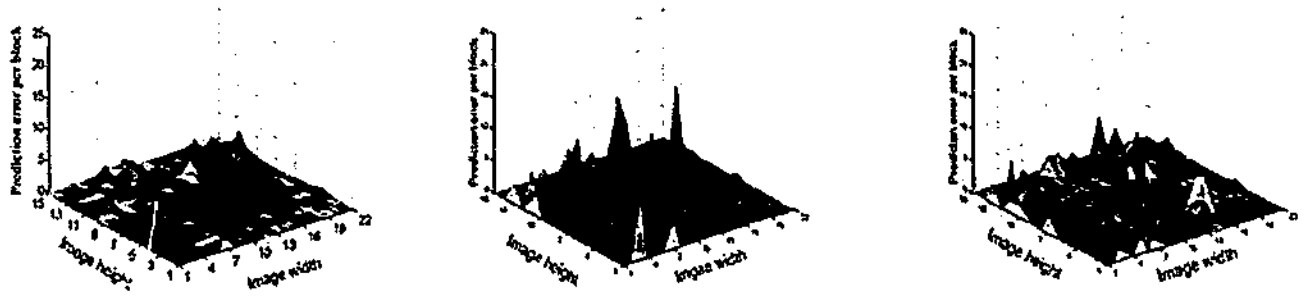
[16] S.C. Kwatra, C.-M. Lin, and W.A. Whyte, "An adaptive algorithm for motion compensated color image coding," *IEEE Trans. Commun.*, vol.COM-35, no. 7, pp. 747-54, July 1987.

[17] G. Sorwar, M. Murshed, and L. Dooley, "Modified Full-Search Block-Based Motion Estimation Algorithm With Distance Dependent Thresholds," submitted to ICASSP-2002, Florida, May 13-17, 2002 , USA.

[18] G. Sorwar, M. Murshed, and L. Dooley, "Block-based True Motion Estimation using Distance Dependent Thresholded search", in *Proc. ISCA Comp. Appl. in Indus. and Eng.*, pp. 45-48, 2001.

(a) (b) (c) (d)

Fig. 5. Estimated image of 80th frame of the "Tennis" sequence: (a) FS; (b) DTS: LT (8); (c) TSS; (d) NTSS algorithms.

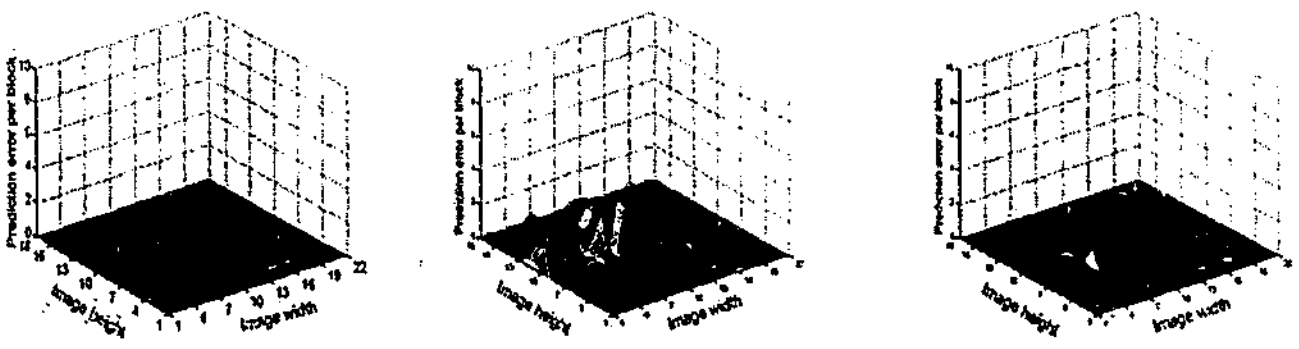


(a) LT(8) (b) TSS (c) NTSS

Fig. 6. Prediction error distribution of 80th frame of the "Tennis" sequence with respect to that of FS algorithm.

(a) (b) (c) (d)

Fig. 7. Estimated image of 5th frame of the "Salesman" sequence: (a) FS; (b) LT(6); (c) TSS; (d) NTSS algorithms.



(a) LT(6) (b) TSS (c) NTSS

Fig. 8. Prediction error distribution of 5th frame of the "Salesman" sequence with respect to that of FS algorithm.

BEAMLET CODER: A TREE-BASED, HIERARCHICAL CONTOUR REPRESENTATION AND CODING METHOD

Jihong Chen; Xiaoming Huo, Georgia Institute of Technology, United States

A quad-tree-based hierarchical contour representation and coding method is studied. This method is based on multiscale line segments---beamlets. Simulations are reported to evaluate the effectiveness of such an approach. This is a proof-of-concept study. The reported compression ratios are not the ``best". However, the idea of tree-based coding is novel; and this idea has good potential to realize a progressive contour coding, which is important in applications such as content-based video transmission.

MODIFIED FULL-SEARCH BLOCK-BASED MOTION ESTIMATION ALGORITHM WITH DISTANCE DEPENDENT THRESHOLDS

Golam Sorwar; Manzur Murshed; Laurence Dooley, Monash University, Australia

A modified full-search (MFS) algorithm is presented for block-based motion estimation applications, which introduces the novel concept of variable distance dependent thresholds. The performance of the MFS algorithm is analyzed and quantitatively compared with both the traditional and exhaustive full-search (FS) technique, and the computationally faster, non-exhaustive three-step-search (TSS) algorithm. Experimental results show that by applying an appropriate threshold function, the MFS algorithm not only matches the speed of the TSS algorithm, but both retains a block distortion error comparable to the global minimum produced by the FS algorithm, and avoids the problem of identifying large numbers of spurious motion vectors in the search process.

PERCEPTUAL CODING OF DIGITAL IMAGES

Damian Tan; Hong Ren Wu, Monash University, Australia; Zheng Yu, Motorola Research Centre, Australia

A novel perceptual image coder of grey level images is presented. This coder is an improved version of the coder by Tan et al with better optimised parameters featuring a local contrast sensitivity function, intra-frequency masking and inter-orientation masking functions for perceptual error modelling. The architecture of the proposed coder follows that of the state-of-the-art EBCOT by Taubman and adopted by the JPEG2000 standard as the core coding structure. The overall perceptual performance improvement of the proposed coder is noticeable compared with the EBCOT coder with the MSE and CVIS error measures.

0-7803-7402-9/02/$17.00 ©2002 IEEE

FAST BLOCK-BASED TRUE MOTION ESTIMATION USING DISTANCE DEPENDENT THRESHOLDS (DTS)

Golam Sorwar, Manzur Murshed, and Laurence Dooley
Gippsland School of Computing and Information TechnologyMonash University, Churchill Vic 3842, Australia
{Golam.Sorwar,Manzur.Murshed,Laurence.Dooley}@infotech.monash.edu.au

ABSTRACT

A new fast motion estimation algorithm, called *distance dependent thresholding search* (DTS), is presented for block-based true motion estimation applications, and introduces the novel concept of variable distance dependent thresholds. The performance of the DTS algorithm is analyzed and quantitatively compared with both the traditional and exhaustive *full-search* (FS) technique, and the computationally faster, non-exhaustive *three-step-search* (TSS) algorithm. Experimental results show that by applying an appropriate threshold function, the DTS algorithm not only matches the speed of the TSS algorithm, but both retains a block distortion error comparable to the global minimum produced by the FS algorithm, and avoids the problem of identifying a large number of spurious motion vectors in the search process.

1. INTRODUCTION

Motion estimation in image sequences has been a key element in a wide range of applications from computer vision through to popular video compression standards such as the MPEG (Motion Picture Expert Group) family.

Many different motion estimation algorithms have been proposed, including pel-recursive [1][13], block-matching [5], and the optical flow-based method [4][8]. The *block-matching algorithm* (BMA) has proved to be very popular because of its simplicity, robustness, and ease of implementation. It estimates motion on a block-by-block basis and has been widely exploited in many video coding standards including MPEG-1 and -2 as well as H.261 263. One particularly important feature [2] of the BMA is that it exhibits superior performance for larger pixel block displacements.

The exhaustive BMA, known as the *full search* (FS) algorithm, searches each candidate block for the closest match within the entire search region to minimize the *block-distortion measure* (BDM). The BDM of image blocks may be measured using various criteria such as, the *mean absolute error* (MAE), the *mean square error* (MSE), and the *matching pel-count* (MPC).

Since the FS algorithm exhaustively searches for a global minimum block-difference error for each candidate block, it generally provides the lowest possible distortion error of any BMA. The algorithm however, suffers two major drawbacks. Its exhaustive nature means it is computationally expensive and in addition, the algorithm tends to capture many "false" motion vectors even when there is no object motion within the search region. This is due to the fact that the distortion of an object in a video frame is directly proportional to its velocity as well as the zoom factor of the camera and therefore, as the length of a motion vector grows so does the block difference distortion error. Although this observation has very little impact when the algorithm is used for video coding, severe artifacts can arise when the algorithm is applied to estimate the true motion vectors, where both object and or camera motion is present.

A number of fast non-exhaustive block matching approaches have been proposed including the *three-step search algorithm* (TSS) [6], the *new three-step search algorithm* (NTSS) [7], the *2D-logarithmic search algorithm* (2DLOG) [5], the *four-step search algorithm* (4SS) [10], and the *cross-search algorithm* [3]. Of these the TSS has gained popularity because of its simplicity and effectiveness, and has been recommended by RM8 of H.261 and SM3 of MPEG [10].

All the aforementioned fast algorithms are based upon the assumption that the BDM increases as the checking points move away from the global minima. According to [1], however, this assumption does not hold true for real world video sequences. Any directional search algorithm can, therefore, be ambiguous and converge to one of the local minima. Moreover, none of the above fast algorithms address the key issue of avoiding the capture of significant numbers of spurious motion vectors in the search process [2].

This paper directly addresses these issues by introducing a new distance dependent thresholding search algorithm (DTS), which not only avoid picking a large number of "false" motion vectors, but also simultaneously exhibits the characteristics of a fast search and low BDM.

The paper is structured as follows. Section 2 describes the new distance dependent threshold search (DTS) algorithm using both linear and exponential thresholding functions. Experimental results to verify the performance of the DTS algorithm in terms of both its search speed and corresponding BDM error measure are presented in Section 3, which also discusses the selection of the threshold function and related parameters, as well as explaining how the DTS algorithm avoids a large number of spurious motion vectors in the search process. Conclusions are provided in Section 4.

2. THE DISTANCE DEPENEDNT THRESHOLDING SEARCH (DTS) ALGORITHM

In the FS algorithm, the suitability of a matching block is measured based on the optimal (minimum) BDM. The FS algorithm works well when there is no distortion, but as alluded in Section 1, the level of distortion present in any video frame increases with the velocity of the moving objects and or the zoom factor used for the camera. The exhaustive FS algorithm therefore, becomes increasingly inefficient as the spiral trajectory (search pattern) expands.

0-7803-7488-6/02/$17.00 © 2002 IEEE.

This paper proposes that the suitability measure of the FS algorithm is relaxed from the optimal criterion as the spiral search trajectory moves from the centre, and becomes distance dependent. Locating a block with the minimum difference, but with a motion vector of high magnitude, is both ineffectual in the prevailing distorted search space, and may also lead to many "false" motion vectors being erroneously selected. In estimating true motion, the suitability measure of the FS algorithm must be relaxed and for the new DTS algorithm, the following two variable distance dependent threshold functions are applied.

2.1. Linear Thresholding (LT)

Let the centre of the search region be at pixel $p_{cx,cy}$, which also defines the spiral search starting point. In LT, the spiral search will terminate at search point $p_{cx+u,cy+v}$ when:-

$$MAE_{(cx,cy)}(u,v) \le C_L \times magnitude(u,v) \qquad (1)$$

Assuming b-bit gray level intensity, the maximum value of the MAE will be 2^b-1, since the pixel intensity is measured using 2^b levels with values 0, 1,..., 2^b-1. As (d,d) is the longest possible motion vector within the search region, an upper bound may be set for constant C_L such that :-

$$C_L < \frac{2^b}{magnitude\,(d,d)}. \qquad (2)$$

2.2 Exponential thresholding (ET)

In ET, the spiral search terminates at search point $p_{cx+u,cy+v}$ when

$$MAE_{(cx,cy)}(u,v) \le 2^{C_E \times magnitude(u,v)}. \qquad (3)$$

Using a similar argument to that in section 2.1, an upper limit can be set for the constant C_E :-

$$C_E < \frac{b}{magnitude\,(d,d)}. \qquad (4)$$

It is interesting to note that setting either $C_L = 0$ or $C_E = 0$ in (1) or (3) respectively, it transforms the DTS algorithm into the original exhaustive FS algorithm. It is also clear that the search time for the DTS algorithm decreases as the value of the constant (C_L or C_E), used in the respective threshold function is increased.

3. EXPERIMENTAL RESULTS

All experiments were performed on a Pentium III 600 MHz computer under Windows NT and using MATLAB 6. The FS, TSS, and DTS algorithms were used to compute the block-based inter-frame motion vectors from the luminance (Y-component) signal of the first 80 frames of two standard test video sequences "Tennis" and "Flower Garden". The "Tennis" sequence comprised various kinds of motions, including translation, zooming, and panning, while the "Flower Garden" sequence mainly consisted of high portions of fast panning motion. Both sequences had the same frame size of 352×240 pixels, uniformly quantized to an 8-bit gray level intensity. In the experiments, the block size dimensions were $M * N = 16$ and $d = \pm 7$, i.e., each frame was divided into 16×16 pixel blocks and within frame, a maximum of $(2d+1)^2 = 225$ checking points were us

Both linear and exponential threshold functions were us assess the performance of the DTS algorithm. In the follo results, the linear threshold function with constant C_L is de as $LT(C_L)$ and the exponential threshold function with con C_E is denoted as $ET(C_E)$.

To quantitatively evaluate the performance of the algorithm for both the LT and ET functions, the following specific measures were identified:

- The average MSE between the reconstructed and th corresponding original frames.
- The average number of search points.
- The average percentage of true object motion vecto captured.

Table I: Average MSE and average search points pe motion vector for "Flower Garden" and "Tennis sequences (1-80 frames).

BMA	Flower Garden			Tennis	
	$MSE_{(avg)}$	Search $points_{(avg)}$		$MSE_{(avg)}$	Search $points_{(av}$
FS(±7)	270.46	199.76		126.34	196.98
LT(2)	270.97	155.02		127.25	75.68
LT(4)	275.80	73.48		131.64	38.14
LT(6)	283.98	45.93		138.26	26.03
LT(8)	293.61	33.59		147.59	20.35
LT(12)	318.73	23.05		166.71	14.65
LT(16)	353.03	18.60		183.65	12.36
ET(1)	275.96	70.16		135.22	49.22
ET(2)	270.49	155.02		126.5	120.80
ET(4)	270.47	175.65		126.35	171.16
ET(8)	270.47	180.65		126.35	182.36
TSS	322.88	23.22		190.81	22.75

3.1. MSE Results

The performance of the DTS algorithm using both the LT an functions in terms of the average MSE between the estimatec original frames is shown in Table I. It can be observed tha DST algorithm variants compared very favourably with th algorithm for both test video sequences, even when the nu of search points was comparable with the TSS algorithm, a example in the "Tennis" LT (8) case. At higher search spe where the improvement factor was typically between 3 and 5 average MSE for the DTS algorithm was still very close (w 1%) of the optimal average MSE produced by the FS algori In contrast, the performance of the TSS algorithm significantly inferior, especially in respect of the motion inve in the "Tennis" video sequence, since being a directional se algorithm, TSS tends to converge to one of the local minir explained in Section1. In Figures 1 and 2 respectively, the I performance of the DTS algorithm against the FS and algorithms is plotted. For the sake of clarity in plotting, only threshold function for the DTS algorithm that used a numbe search points comparable to the TSS algorithm was considere

3.2. Search Points Results

The performance of the FS, TSS and DTS algorithms in terms of the average search points to estimate motion vectors is presented in Table I. Again it is clear that the various versions of the DTS algorithm were faster by a factor of at least 3, and in the "Tennis" LT(16) case, it is more than 15 times faster than the FS algorithm while the MSE is comparable to the TSS algorithm (9 times faster). The results also proved that by choosing a suitable constant for the selected threshold function, the average search points required by the DTS algorithm could be considerably less, while concomitantly having a significantly lower average MSE.

Another finding from the results in Table I was that the LT function consistently provided a better performance in comparison with the ET function for a wide range of different video sequences. This was due to the fact that the distortion of an object in any frame was linearly proportional to its velocity as well as the zoom factor of the camera.

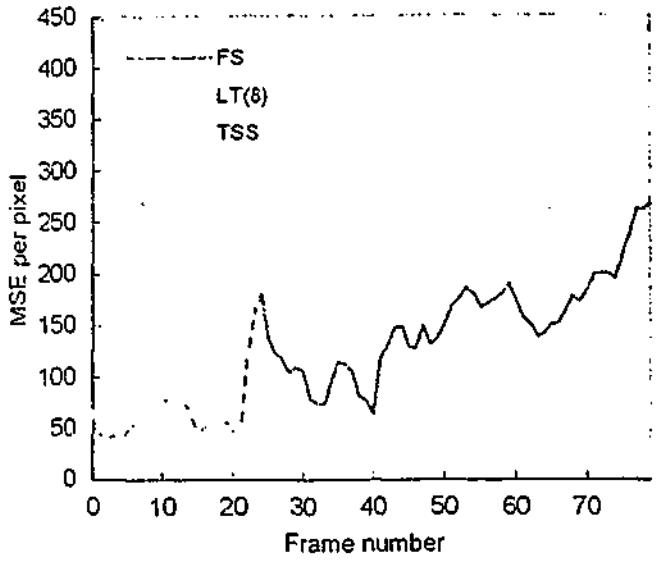


Figure 1. MSE Comparison of "Tennis" video sequence

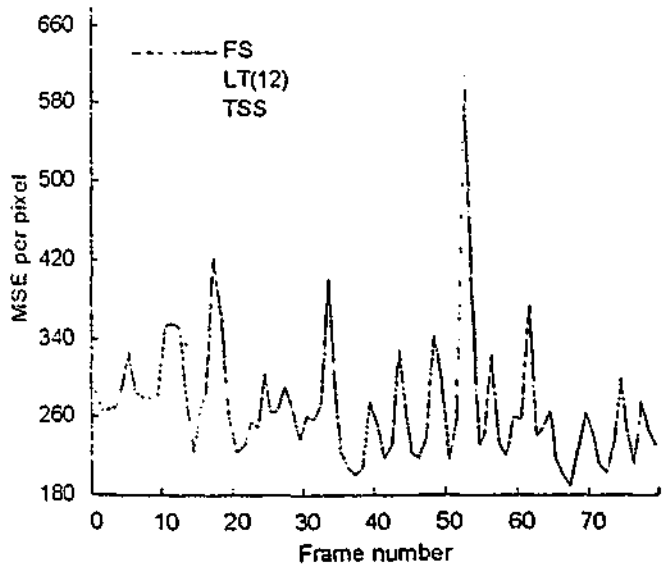


Figure 2. MSE Comparison "Flower Garden" video sequences.

3.3. True Motion Result

The performance of the DTS algorithm was also evaluated i terms of how effectively it could capture true object motion. Tru object motion vectors were calculated from the block motio vectors by compensating for camera motion and then filterin noise at various threshold levels. True motion vectors obtaine by the DTS algorithm were then compared with the result obtained from the newly proposed *Mean Accumulated Threshol* (MAT) filter [12] that captures true object motion wit significantly higher accuracy.

The performance of the MAT filter has been proven t significantly increase the length of "true" object motion vector compared with all other vectors, within a relatively small numbe of iterations (as low as 2). This has been achieved by a two-stag combination of mean filtering and thresholding.

Comparative results were made against the output from th MAT filter, by manually checking those blocks, which ha already been identified as containing moving objects in th frames.

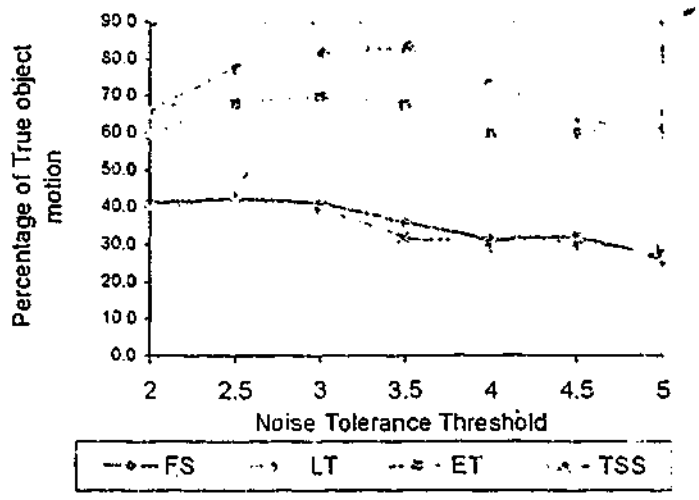


Figure 3. Comparison of LT, ET, FS and TSS in terms of th percentage of true motion vectors captured with different noi tolerance thresholds.

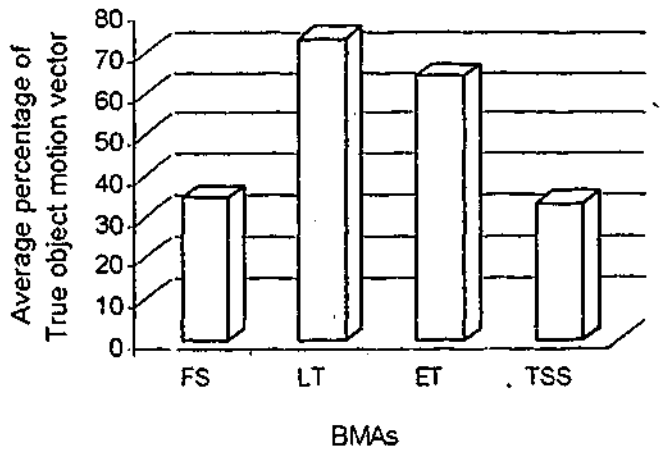
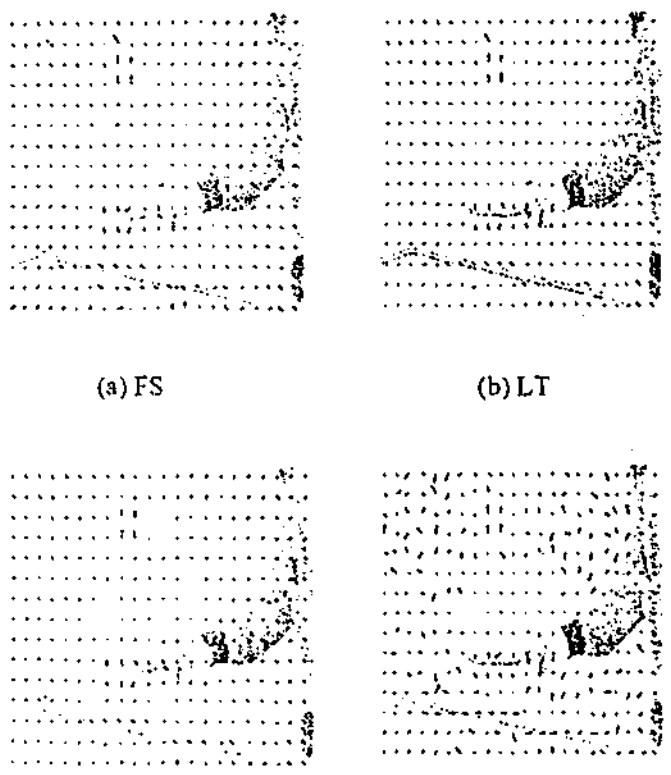


Figure 4. Average number of true object motion vector captured by different algorithms.

The above process was applied to six different standard video sequences and the average values are plotted in Figures 3 and 4. These clearly showed that the performance of the DTS algorithm, using both the LT and ET functions in capturing the true object motion vectors, was considerably better than that of both the FS and TSS algorithms for all noise tolerance threshold levels considered. The graphs also reaffirmed the earlier judgment that LT was a better thresholding function than ET for the DTS algorithm.

Figure 5 shows the motion vectors captured by all three algorithms for the contiguous pair of frames 32 and 33 from the "Tennis" sequence. Besides low camera motion (zoom out), the only moving objects appearing in these frames are the ball, the bat, and a portion of the hand holding the bat. The figure reveals that the FS and TSS algorithms perform far worse compared to the DTS algorithm, by capturing a large number of spurious true motion vectors. The DTS algorithm eliminates many of the false vectors so ensuring an overall superior performance.



(a) FS (b) LT

(c) ET (d) TSS

Figure 5. The motion vectors obtained from all algorithms applied to the frame pair 32 and 33 of the "Tennis" sequence.

4. CONCLUSIONS

This paper has presented a new fast true motion estimation algorithm, based on the concept of variable distance dependent thresholding. The performance of the NEW DTS algorithm was examined and shown, in comparison to both the full-search (FS) and the fast three-step-search (TSS) algorithms, that it provided comparable speed performance, while retaining a distortion error similar to the minimum value produced by the FS algorithm. Both linear and exponential thresholding functions were applied in the DTS algorithm, with the former consistently providing better performance. The variable thresholding feature of the DTS algorithm also avoided identifying large numbers of spurious motion vectors in the search process.

REFERENCES

[1] H.-K. Chow, M.L. Liou, "Genetic motion search algorithm for video compression," IEEE Trans. on Circuits and Systems for Video Tech., vol. 3, pp. 440–445, 1993.

[2] F. Dufaux, F. Moscheni, "Motion estimation techniques for digital TV: a review and a new contribution," Proc. of the IEEE, vol. 83, pp. 858-876, 1995.

[3] M. Ghanbari, "The cross-search algorithm for motion estimation (image coding)," IEEE Trans. on Comm. vol. 38, pp. 950-953, 1990.

[4] K.P. Horn, B.G. Schunck, "Determining Optical flow," Artificial Intell. vol 7, pp. 185-203, 1981.

[5] J.R. Jain, A.K. Jain, "Displacement measurement and its application in inter frame image coding," IEEE Trans. Comm., vol. 29, pp. 1799-1808, 1984.

[6] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, T. Ishiguro, "Motion-compensated inter frame coding for videoconferencing," IEEE Nat. Telecomm. Conf., vol. 4, pp. 1-5, 1981.

[7] R. Li, B. Zeng, M.L. Liou, "A new three-step search algorithm for block motion estimation," IEEE Trans. on Circuits & Systems for Video Tech., vol. 4, pp. 438-442, 1994.

[8] B.D. Lucas, T. Kanade, "An iterative image registration technique with an application to stereo vision," Proc. DARPA Image Understanding Workshop, pp. 121-130, 1981.

[9] J.-Y. Nam, J.-S. Seo, J.-S. Kwak, M.-H. Lee, H.H. Yeong, "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," IEEE Trans. on Consumer Electronics, vol. 46, pp. 934-942, 2000.

[10] L.-M. Po, W.-C. Ma, "Novel four-step search algorithm for fast block motion estimation," IEEE Trans. on Circuits & Systems for Video Tech., vol. 6, pp. 313-317, 1996.

[11] J.D. Robbins, A.N. Netravali, "Recursive motion compensation: a review," Image Sequence Processing And Dynamic Sconce Analysis, Springer-Verlag, pp. 76-103, 1983.

[12] G. Sorwar, M. Murshed, L. Dooley, "A Novel Filter for Block Based Motion Estimation," In Proc. of the Int. Conf. on Digital Image Comp. Tech. and Applications, Melbourne, Australia, January 21-22, 2002 (in publication).

[13] D.R. Walker, K.R. Rao, "Improved pel-recursive motior compensation," IEEE Trans. Comm. Vol. 32, pp. 1128-1134, 1984.

BLOCK-BASED TRUE MOTION ESTIMATION USING DISTANCE DEPENDENT THRESHOLDED SEARCH

Golam Sorwar, Manzur Murshed, and Laurence Dooley
Gippsland School of Computing and Information Technology
Monash University, Churchill Vic 3842, Australia
[redacted]

Abstract

The full-search (FS) block-matching algorithm for block-based motion estimation works best for video coding in terms of minimum block-distortion error. But in true motion estimation the FS algorithm tends to capture many "false" motion vectors even when no object motion is present in the search region. This is due to the fact that the distortion of an object in a video frame is proportional to its velocity and therefore, as the length of a motion vector grows so does the block- difference error. This paper introduces an improved version of the FS algorithm including distance dependent thresholds to avoid capturing "false" motion vectors and improve the efficiency of the search.

Keyword: Block-based motion estimation, Distance dependent thresholds.

1 INTRODUCTION

Motion estimation in image sequences has been a key element in a wide range of applications from computer vision through to popular video compression standards such as MPEG (Motion Picture Expert Group).

Motion is primarily due to the movement of a camera, movement of objects in the frame, or movement of both camera and objects. There are many types of estimation algorithms such as pel-recursive [12], block-matching [6], and optical flow based method [5][9]. In general, the block-matching algorithm (BMA) is popular due to its simplicity, robustness, and ease of implementation. This algorithm estimates motion block-by-block basis, which is already adopted by a large number of video coding standards (MPEG-1/2 and H.261/263 etc.).

The exhaustive block-matching algorithm, known as the full search (FS) algorithm, searches each candidate block for the closest match within the entire search region to minimize the block-distortion measure (BDM). The algorithms have been widely used in block motion estimation for video coding and indexing. Since the FS algorithm exhaustively searches for the minimum BDM for each candidate block, it generally provides reasonably good performance with the expense of high computational time.

Several fast algorithms have already been proposed to address the above issue. The three-step search algorithm (3SS) [7], the new three-step search (N3SS) [8], the four-step search algorithm (4SS) [11], and the cross-search algorithm [4] are based on the assumption that the block distortion measure increases as the checking points move away from the global minima. But this assumption dose not hold true in the real world video sequences [2]. Moreover, search directions of the above algorithms can be ambiguous and therefore, may converge to local minima. In order to solve the direction problem, a new method based on temporal and spatial correlation of motion vector is presented in [10].

In true motion estimation, where object and/or camera motions are estimated, the FS algorithm tends to capture many "false" motion vectors even when no object motion is present in the search region. This is due to the fact that the distortion of an object in a video frame is proportional to its velocity and therefore, as the length of a motion vector grows so does the BDM. This phenomenon has been observed in [1][3]. In spite of the above drawbacks, block-matching algorithms show better performances, especially for large displacement [3]. In this paper we address this issue by modifying the FS algorithm to incorporate distance dependent thresholds. This modification not only avoids picking a large number of "false" motion vectors but also makes the search quite faster.

The remainder of this paper is organized as follows. Section 2 describes different block-matching criteria used in various block-based motion estimation algorithms. The general block-based motion estimation technique, including the FS algorithm, is discussed in Section 3. In Section 4 we present two new algorithms developed using dynamic thresholds in the FS algorithm. Some experimental results are included in Section 5. Section 6 concludes the paper.

2 BLOCK MATCHING CRITERIA

Matching of image blocks can be measured using various criteria e.g., the mean absolute error (MAE), the mean square error (MSE), and the matching pel-count (MPC) etc. In block-based algorithms, each frame is divided into equal sized rectangular blocks, of size (say) $M \times N$, as shown in Figure 1. Throughout this paper, pixels of a frame are

numbered using the Cartesian coordinate system with the origin starting from the upper-left corner.

Let $F_n(k,l)$ denotes the intensity function of the $M \times N$ sized block containing all the pixels $p_{x,y}$ of frame number n, where $k \le x < k+N$ and $l \le y < l+M$. So, $F_n(k,l)(i,j)$ represents the intensity of the pixel $p_{k+i,l+j}$ of frame number n, for all $0 \le i < M$ and $0 \le j < N$.

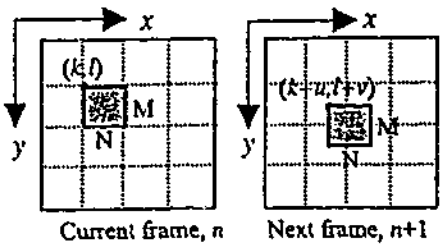


Figure 1: Frame-Block coordinate system.

(a) Mean Absolute Error (MAE)

$$MAE_{(k,l)}(u,v) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}|F_n(k+i,l+j) - F_{n+1}(k+u+i,l+v+j)| \quad (1)$$

(b) Mean Square Error (MSE)

$$MSE_{(k,l)}(u,v) = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[F_n(k+i,l+j) - F_{n+1}(k+u+i,l+v+j)]^2 \quad (2)$$

(c) Matching Pel-Count (MPC)

$$T_{(k,l)}(u,v)(i,j) = \begin{cases} 1, & \text{If } |F_n(k,l)(i,j) - F_{n+1}(k+u,l+v)(i,j)| \le Threshold \\ 0, & \text{Otherwise} \end{cases}$$

$$MPC_{(k,l)}(u,v) = \sum_{i=1}^{M-1}\sum_{j=1}^{N-1} T_{(k,l)}(u,v)(i,j) \quad (3)$$

Among the above matching criteria, MSE require multiplication and accumulation while the others require comparison and accumulation. Since multiplication is expensive compared to comparison, the MAE criterion is most widely used and is adopted in this paper.

3 BLOCK-BASED MOTION ESTIMATION

In a block-matching algorithm, the current frame is divided into small rectangular blocks as explained in Figure 1 and Figure 2. For each block of the current frame, a motion vector is obtained by finding a suitably matched block within the search window of the reference frame.

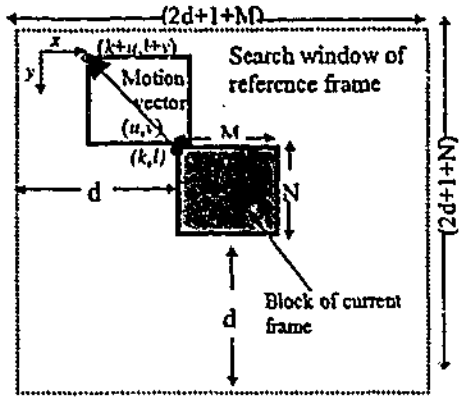


Figure 2: Search region of block-matching algorithms.

3.1 Full-Search Algorithm

The most straightforward block-matching algorithm is the full search (FS) algorithm. In selecting a suitably matched block, the FS algorithm searches the entire search region for a block such that the BDM is the minimum If the maximum displacement of a motion vector is $\pm d$ pixels shown in Figure 2, both in horizontal and vertical directions, the total number of search points used to find the motion vector for each block can be as high as $(2d+1)^2$. If more than one blocks produces the minimum BDM, the FS algorithm obviously prefers the block with motion vector of smaller magnitude. Therefore, the FS algorithm computes block-differences in a spiral trajectory starting from the center of the search region as shown in Figure 3.

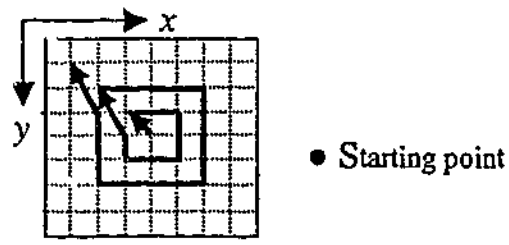


Figure 3. The spiral trajectory of searching.

4 OUR ALGORITHMS

In the FS search, the suitability of a matching block is measured based on optimal (minimum) block-difference error. The distortion of an object in a video frame is proportional to its velocity and therefore, as the length of a motion vector grows so does the block-difference error. Obviously, the exhaustive search in the FS algorithm is inefficient, especially when it fails to obtain a close match within a closer distance from the center of the search region.

We propose that the suitability measure of the FS algorithm be relaxed from the optimal criterion as the trajectory of the spiral searching moves away from the center. Looking for a block with the minimum difference, but with a motion vector of high magnitude, is not only useless in the prevailing distorted search space, but also erroneous as it may lead to many "false" motion vectors.

In estimating true motion, the suitability measure of the FS algorithm must be relaxed and we apply the following two threshold functions in relaxing the measure.

Let the center of the search region be at pixel $p_{cx,cy}$, i.e., the spiral search starts from search point $p_{cx,cy}$.

4.1 Linear Thresholding (LT)

The spiral search terminates in search point $p_{cx+u,cy+v}$, if

$$MAE_{(cx,cy)}(u,v) \le C_L \times magnitude(u,v). \quad (4)$$

The value of the MAE can be at most 255, considering that intensity of a pixel is measured using 256 levels. As (d,d) is the longest possible motion vector within the search region, we can set an upper limit to the constant C_L as

$$C_L < \frac{256}{magnitude(d,d)}. \quad (5)$$

4.2 Exponential Thresholding (ET)

The spiral search terminates in search point $p_{cx+u,cy+v}$, if

$$MAE_{(cx,cy)}(u,v) \le 2^{C_E \times magnitude(u,v)}. \quad (6)$$

Using the same argument as used in Section 4.1, we can set an upper limit to the constant C_E as

$$C_E < \frac{8}{magnitude(d,d)}. \quad (7)$$

It is interesting to observe that setting $C_L = 0$ or $C_E = 0$ would transform the above two algorithms into the original exhaustive FS algorithm.

5 EXPERIMENTAL RESULTS

We conducted a series of experiments on a Pentium III 600 MHz computer with Windows NT operating system. Inter-frame motion vectors were calculated based on the FS, the LT, and the ET algorithms implemented in MATLAB 6. Throughout the experiments, we used $M = N = d = 16$, i.e., each frame was divided into 16×16 pixel blocks and the size of the search region was 49×49 pixels, where at most 33^2 search points are used. All experiments were performed on the luminance (Y-component) of the frames.

In Table 1 we present comparative computational times of all the three algorithms applied on two candidate frame pair from each of the nine test video sequences. The last row in Table 1 reveals that the LT and the ET algorithms require no more than $1/8^{th}$ of time of the FS algorithm.

Table 1: Motion computational time comparison.

	Motion Computational Time (Seconds)		
	FS	LT	ET
Tennis	139.27	8.38	16.90
Flower	132.76	26.59	26.10
Us21	111.25	2.30	3.52
Interview	125.62	6.12	10.84
Ballet	126.49	4.90	8.39
Bicycle	136.66	44.09	37.69
Tesla	43.73	16.37	11.92
Seinfield	40.59	4.84	5.90
Foreman	38.02	5.04	6.00
Average Time	99.38	13.18	14.14

In Figure 4, the MAE per pixel for each of the three algorithms is plotted. In all the nine cases, the performance of our two algorithms appears to be the same as that of the FS algorithm. It may, therefore, be concluded that by relaxing the suitability measure of the FS algorithm, through the introduction of thresholding, the displaced block selection by both of our algorithms are at least as good as the selection by the FS algorithm.

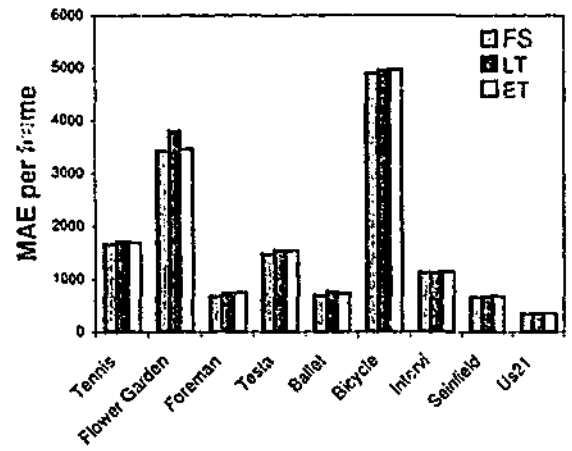


Figure 4: Comparative prediction errors.

Figure 5 shows the luminance of the frame number 32 and 33 of the test video sequence "Tennis" where besides low camera motion (zoom out), the only moving objects are the ball, the bat, and the hand holding the bat. The motion vectors obtained by all the three discussed algorithms, applied to these frame pair, are shown in Figure 6. Figure 6(a) clearly shows comparatively how badly the FS algorithm performs by capturing a large number of "false" motion vectors.

6 CONCLUSION

The full-search block-matching algorithm for block-based motion estimation works best for video coding. But in true motion estimation the FS algorithm tends to capture many "false" motion vectors even when no object motion is present in the search region. In this paper, we have modified

the FS algorithm by introducing distance dependent linear and exponential thresholds that have not only made the search faster but also avoided capturing a large number of "false" motion vectors.

REFERENECES

[1] Cheung C.-H. and Po L.-M., "A fast block motion estimation using progressive partial distortion search," International Symposium on Intelligent Multimedia, Video and Speech Processing, pp. (s): 506–509, 2001.

[2] Chow H.-K. and Liou M.L., "Genetic motion search algorithm for video compression," IEEE Trans. on Circuits and Systems for Video Technology, vol. 3, pp. (s): 440–445, 1993.

[3] Dufaux F. and Moscheni F., "Motion estimation techniques for digital TV: a review and a new contribution," Proc. of the IEEE, vol. 83, pp. 858-876, 1995.

[4] Ghanbari M, "The cross-search algorithm for motion estimation (image coding)," IEEE Trans. on Comm., vol. 38, pp. 950-953, 1990.

[5] Horn K.P. and Schunck B.G., "Determining Optical flow," Artificial Intelligence, Vol. 17, pp. 185-203, 1981.

[6] Jain J.R. and Jain A.K., "Displacement measurement and its application in inter frame image coding," IEEE Trans. Comm., vol. COM-29, pp. 1799-1808, 1984.

[7] Koga T., Iinuma K., Hirano A., Iijima Y. and Ishiguro T., "Motion-compensated inter frame coding for videoconferencing," IEEE National Telecommunications Conference, vol. 4, pp. G5. 1-5, 1981.

[8] Li R., Zeng B. and Liou M.L., "A new three-step search algorithm for block motion estimation," IEEE Trans. on Circuits & Systems for Video Technology, vol. 4, pp .438-442, 1994.

[9] Lucas B.D. and Kanade T., "An iterative image registration technique with an application to stereo vision," Proc. DARPA Image Understanding Workshop, pp. 121-130, 1981.

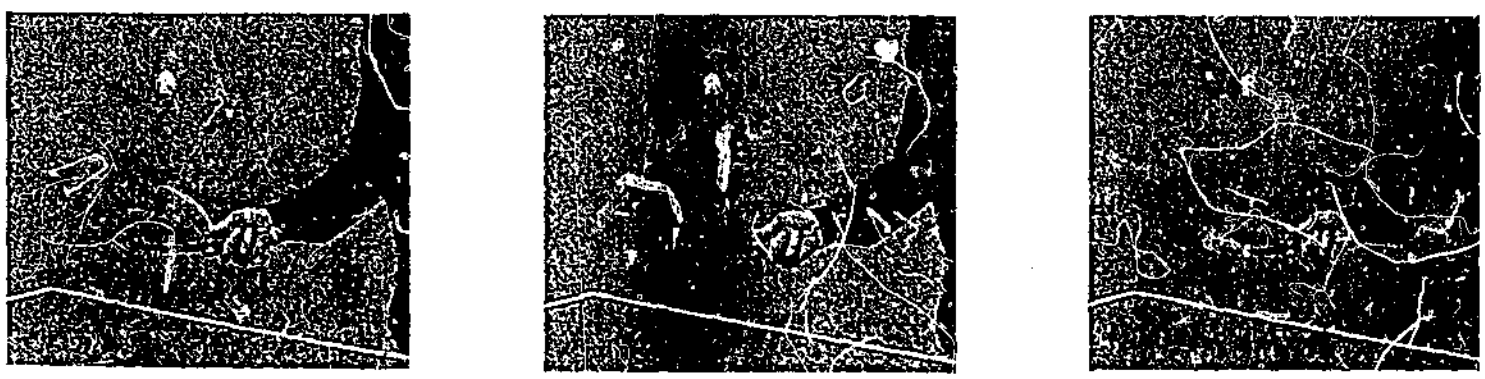
[10] Nam J.-Y., Seo J.-S., Kwak J.-S., Lee M.-H. and Yeong H.H., "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," IEEE Trans. on Consumer Electronics, vol. 46, pp. 934-942, 2000.

[11] Po L.-M. and Ma W.-C., "Novel four-step search algorithm for fast block motion estimation," IEEE Trans. on Circuits & Systems for Video Technology, vol. 6, pp. 313-317, 1996.

[12] Robbins J.D. and Netravali A.N., "Recursive motion compensation: a review," Image Sequence Processing And Dynamic Sconce Analysis, pp. 76-103, Spriger-Verlag, 1983.

(a) (b)

Figure 5: (a) Frame #32; (b) Frame #33; of the test video sequence "Tennis" sequence



FS LT ET

Figure 6: The motion vectors obtained by all the three algorithms applied to the frame pair of Figure 5.

A FULLY ADAPTIVE PERFORMANCE-SCALABLE DISTANCE-DEPENDENT THRESHOLDING SEARCH ALGORITHM FOR VIDEO CODING

Golam Sorwar, Manzur Murshed and Laurence Dooley

Gippsland School of Computing and Info. Tech., Monash University, Churchill Vic 3842, Australia
{Golam.Sorwar,Manzur.Murshed,Laurence.Dooley}@infotech.monash.edu.au

ABSTRACT

Trading-off computational complexity and quality is an important performance constraint for real time application of motion estimation algorithm. To address this issue, a *distance dependent thresholding search* (DTS) algorithm has been proposed for fast and robust true motion estimation in video coding/indexing applications. DTS encompassed both the *full search* (FS) as well as fast searching modes, with different threshold settings providing various quality-of-service levels. The main drawback of DTS was that the threshold value was manually defined. In this paper, the DTS algorithm has been extended to a *fully adaptive distance dependent thresholding search* (FADTS), a key feature of which is the automatic adaptation of the threshold using the desired target and the content from the actual video sequence, to achieve a guaranteed level of quality or processing complexity. Experimental results confirm the performance of the FADTS algorithm in achieving this objective with minimal additional computational cost.

1. INTRODUCTION

Motion estimation (ME) plays a vital role in video coding standards, such as MPEG-1/2 [1] [2] and H.261/3 [3][4], in exploiting latent temporal redundancy in video sequences. Most ME techniques use *block matching algorithms* (BMA) to compute motion vectors on a block-by-block basis. The most straightforward method, known as full search (FS), provides optimal performance by searching all possible locations within a given search area, but at the expense of very high computation. It is for this reason that FS is not used in real-time systems. Indeed ME is the major bottleneck in real-time video coding applications, hence the need for faster algorithms.

A number of fast block ME algorithms [5]-[10] have been proposed to lower the computation complexity by sacrificing quality. Among these, *three-step search* (TSS) [6] and *new three-step search* (NTSS) algorithms [7] become more mainly due to their simplicity. However, these motion estimation algorithms are not designed to provide flexible and predictable control of performance in terms of picture quality and computational cost (speed). There is no facility to trade system parameters depending upon a particular application or to preset a user-defined level of picture quality or computational complexity. Such a feature would be very advantageous in facilitating scalable performance management especially in the area of computational complexity management in real time video encoders.

It has been observed that the distortion of an object in a video frame is proportional to its velocity as well as the camera parameters (zoom and pan) and thus, as the length of a motion vector grows so does the block distortion error. Sorwar *et al.* [11]-[14] have addressed this issue by introducing the concept of a distance-dependent thresholding search (DTS) algorithm for fast and robust true motion estimation in object-based video indexing and coding applications. By varying the value of the threshold, the DTS algorithm provides both a FS capability for maximum quality as well as fast searching modes for ME (faster than most traditional algorithms [12]). The main drawbacks associated with DTS are that the threshold value has to be manually selected and cannot be adapted to the content of a particular video sequence.

This paper presents a new *fully automatic adaptive distance-dependent thresholding search* (FADTS) algorithm, which can dynamically adjust the threshold to achieve any level of service required in terms of both quality and processing speed. This means for example, that a higher (lower) error or speed can be achieved by automatically adapting the threshold to a correspondingly level, depending on video content so providing the potential for performance management real time video coding.

The paper is organized as follows. Section 2 briefly describes the basic distance dependent thresholding search (DTS) algorithm, while Section 3 details the new fully adaptive DTS (FADTS) algorithm. Section 4 includes both experimental results and analysis of the performance,

including a computational cost analysis of FADTS for various levels of quality and speed. Section 5 presents the conclusions.

2. DISTANCE-DEPENDENT THRESHOLDING SEARCH (DTS) ALGORITHM INTRODUCTION

A detailed description of DTS algorithm can be found in [11-14], where a technique is presented to estimate the motion vector by introducing the concept of distance-dependent threshold search for variable performance video encoder. This algorithm searches spirally starting from the center of the search window and the search terminates when the block distortion measure (BDM) becomes less than a predefined threshold.

Let the centre of the search region be at pixel $p_{cx,cy}$, which also defines the starting point of the spiral search starting point. In DTS, the spiral search terminates at search square SS_τ [12] when the mean absolute error (MAE), used as the BDM, is:-

$$MAE_{(cx,cy)}(x-cx, y-cy) \le C \times \tau \qquad (1)$$

where C is the threshold value and τ is the concentric square index. Assuming b-bit gray level intensity, the maximum value of the MAE is $(2^b\text{-}1)$, since the pixel intensity is measured using 2^b levels. As SS_d is the outermost search square where d is the maximum displacement, an upper bound for the constant C can be set as:-

$$C < \frac{2^b}{d} \qquad (2)$$

Note, that by setting $C = 0$ in (1), it transforms the DTS algorithm into the exhaustive FS algorithm. It is clear that the search time reduces as C increases and interesting to note that if C is set higher than the upper bound in (2), the search will not explore the entire search area defined by the maximum displacement d.

3. PROPOSED ADAPTIVE THRESHOLD MODEL

The approach adopted for embedding an adaptive threshold into the DTS algorithm is based upon the normalized least-mean-square (NLMS) algorithm [15]-[18]. The threshold is automatically adjusted between frames to achieve either a target level of prediction error (quality) or computation by considering specifically the number of search points per MV.

The block diagram of the proposed model is shown in Fig. 1, and has two modules: (i) motion estimation and (ii) threshold control. In the former, K is the sample vector length, which governs the number of consecutive frames that use the same threshold value, where sample means a pair of frames between which motion has to be calculated. Thus for K=1, a particular threshold value is used to calculate the motion between two consecutive frames, while K=L means the same threshold is used for L-1 consecutive frames (ME always being calculated between two successive frames).

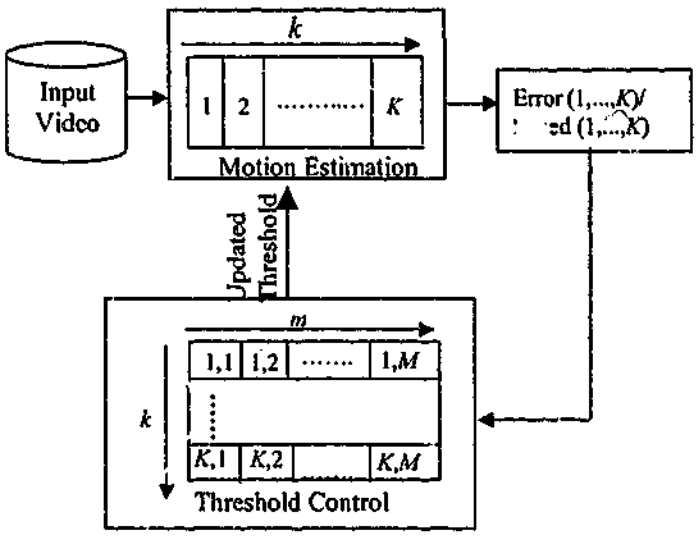


Fig. 1: The proposed DTS adaptive model.

The sample window size is M in the threshold control module, so the total memory requirement for this module is KM. Based on the NLMS method in [16][17] the following is used for threshold adaptation:-

$$C_{j+1} = C_j + \mu e_j(\overline{\mathbf{X}}_j) \frac{1}{\frac{\sum_{k=1}^{K}\sum_{m=1}^{M} \mathbf{X}^2_{k,m,j}}{KM}} \qquad (3)$$

where

$e_j = Desired_j - Actual_j$, $Actual_j = \frac{\sum_{k=1}^{K}\sum_{m=1}^{M} X_{k,m,j}}{KM}$, j is the number of iterations, μ is the step size, $\overline{\mathbf{X}}_j$ represents the average value of input vector (output of motion estimation module) $\mathbf{X}$, where the total number of elements of $\mathbf{X}$ is K.

The output of the motion estimation module is either prediction quality (*mean square error* (MSE)) per pixel or computational time (the number of *search points* (SP) per MV). This information is used to update the threshold for the following frames. The threshold control module selects whether the threshold for the next iteration is to be either increased or decreased depending on the average error or average number of search points so far calculated (*Actual*) and the target (*Desired*). As C decreases, the number of search points corresponding increase and the update factor

for speed adaptation is therefore negative.

The update term also depends on the value of M. The higher the value of M, the larger the update factor while other parameters remain constant. So the performance of the adaptive algorithms depends on the initial threshold constant selection, the step size and the values of K and M.

4. EXPERIMENTAL RESULTS

The performance of the FADTS algorithm was evaluated using the luminance (Y-component) signal of the following standard test video sequences:- "Football" (320×240 pixels), "Flower garden" (352×240 pixels), "Salesman" (360×288 pixels), "Miss America" (176×144 pixels), "Tennis" (352×240 pixels) and "Foreman" (176 ×144 pixels). In this paper only the results for the "Football" and "Flower Garden" are presented. The "Football" sequence contains various kinds of motion, including translation, zooming, and panning, while the "Flower Garden" sequence comprises high panning.

In the experiments, all sequences were uniformly quantised to an 8-bit gray level intensity. The block size dimensions were 16×16 and d = ±7, i.e., within each 16×16 block, a maximum of $(2d+1)^2 = 225$ checking points were used. The MSE measure was used to represent the prediction quality for the best motion vector for each block and the value of K and M are selected as 4 and 1 respectively based on the experiments. All results are shown using half-pel motion accuracy.

The performance of FS, TSS and NTSS algorithms are contrasted in Table I, for showing the comparative performance of FADTS algorithm.

Table I: Avg. MSE and SP of FS, TSS and NTSS algorithms for "Football" (344 frames) and "Flower garden" (150frames) video sequences.

BAM	Football		Flower garden	
	MSE	SP	MSE	SP
FS	218.88	160.05	208.91	209.73
TSS	240.79	25.63	242.97	31.20
NTSS	239.15	26.9	213.28	28.98

The performance of the FADTS algorithm was evaluated for both quality and speed adaptation as follows.

4.1. Quality Adaptation

The FADTS algorithm results in terms of quality adaptation are presented in Table II for a number of different target values for the high motion "Football" and "Flower Garden" sequences. This reveals the FADTS algorithm is able to reach any bounded target level of quality, with the implicit assumption that the minimum target error obtained by FS is the lower bound.

If the target is set so high that the resultant threshold constant will exceed the maximum threshold C_{max}, FADTS algorithm limits the upper bound to C_{max}. However, defining such a high target is unrealistic, because it will produce an extremely poor picture quality output.

Table II: Prediction error adaptation for "Football" a "Flower garden" Video sequences (344 and 150 Frames respectively) with K= 4 and M=1

Football			Flower garden		
Target quality	Actual	Search Point (SP)	Target quality	Actual	Search Points (SP)
MSE	MSE		MSE	MSE	
220	220.35	34.77	210	212.13	49.82
230	228.95	25.90	215	215.52	26.68
240	240.77	20.89	230	229.34	17.68
250	252.23	18.46	240	237.78	16.18

The corresponding adaptive threshold values for different frames are plotted in Fig. 2. This shows clearly the adaptive nature of the FADTS algorithm as content varies between different frames. It also confirms that FADTS automatically computes a different starting threshold value directly proportional to the target value. Thus initial thresholds are adaptive based on both the content of the video sequence and the desired target.

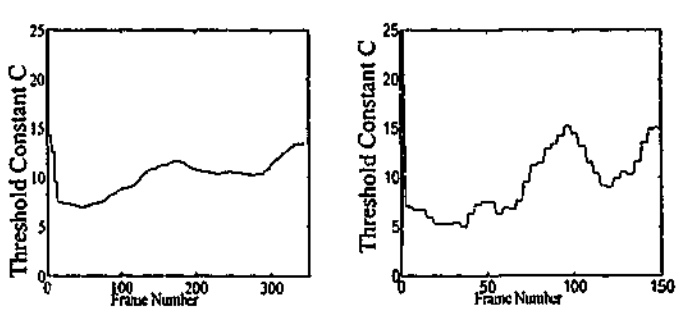


Fig. 2: Threshold constant adaptation for Football (left, 240 MSE) and Flower garden (right, 215 MSE) sequences.

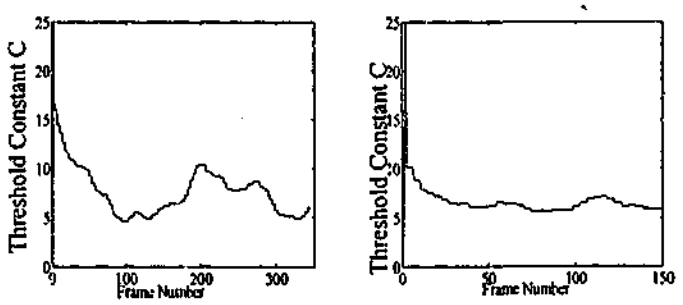


Fig. 3: Threshold constant adaptation for Football (left, 25 SP) and Flower garden (right, 30 SP) sequences.

4.2. Computational complexity adaptation

The computational performance of the FADTS algorithm for a number of different target speeds (average number of search points per MV) is shown in Table III. The table proves that FADTS can reach any average target level of

speed within the bounds (depends on d) by varying the threshold constant. Fig. 3 clearly shows both the adaptive nature of the algorithm as the content in the video sequence varies and also its ability to meet the user-defined target.

Table III: Processing speed adaptation For "Football" and "Flower garden" video sequences (149 Frames) with K= 4 and M=1

Football			Flower garden		
Target Speed (SP)	Actual	Actual Error MSE	Target Speed (SP)	Actual Speed (SP)	Actual Error MSE
SP	SP		SP	SP	
20	19.89	250.93	20	19.55	219.47
25	24.86	234.93	25	24.53	215.68
30	29.82	227.09	30	29.56	214.28
40	40.97	220.22	40	39.78	212.99

Table I, II and III also prove that FADTS does not only reach any user defined target, but also shows better error performance when complexity is comparable to TSS or NTSS. Another noteworthy point is that FADTS can achieve the same MSE performance as FS but with reduced complexity (SP) by a factor of 4 (approx.).

The total number of operations for (3) is only $(3KM+4)\frac{f}{K}$ per second where f is the number of frames per second. Since in the experiments, M=1 and K=4, this means a total of only 120 additional operations per second. This is negligible compare to the complexity involved in MAE distortion calculation for motion estimation, where one MAE calculation requires 511 additions, 256 absolute operations, and one comparison for a 16×16 block.

In summary therefore, the FADTS algorithm consumes minimal additional computational overhead, while providing significant performance benefits including user-definability of key parameters.

5. CONCLUSIONS

This paper has presented a fully adaptive distance dependent thresholding search (FADTS) algorithm for real-time block-based motion estimation in video coding. The performance of FADTS has been examined and proven that it affords a unique feature in being able to trade-off freely between the two key system parameters, namely prediction quality and search speed, for the entire range of threshold values. A key feature of this novel algorithm is its ability to progressively adjust the required threshold value based on the actual video content to achieve any user-specific level-of-service, in terms either prediction quality or processing speed. FADTS can therefore be used as an optimum algorithm for high quality prediction as well as a very fast algorithm. The algorithm proposed in this paper could also form part of a video encoder that can optimize performance in scenarios where computational resources are restricted. Further work is required to integrate this algorithm with other functions of the encoder such as the DCT and quantization scale to control the rate, complexity and distortion performance.

6. REFERENCES

[1] JTC1/SC29/WG11, I.I., Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit. 1993.

[2] JTC1/SC29/WG11, I.I., Generic Coding of moving pictures and associated audio. 1993, ISO/IEC.

[3] H.261, I.-T.R., Video codec for audiovisual services at p'64 kbit/s. 1993.

[4] H.263, D.I.-T.R., Video coding for low bitrate communication. 1996.

[5] J.R. Jain and A.K. Jain, "Displacement measurement and its application in inter frame image coding," IEEE Trans. Commun., vol. COM-29, pp. 1799-1808, Dec. 1984.

[6] T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, "Motion-compensated inter frame coding for videoconferencing," in Proc. NTC81, Nov. 1981, pp. G5. 3.1-G5.3.

[7] 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.

[8] H. Nisar and T.-S. Choi, "An advanced center biased search algorithm for motion estimation," in Proc. ICIP, 2000, vol.1, pp.832-5.

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

[10] M. Ghanbari, "The cross-search algorithm for motion estimation (image coding)," IEEE Trans. Commun., vol. 38, no. 7, pp. 950-953, July, 1990.

[11] G. Sorwar, M. Murshed, and L. Dooley, "Modified Full-Search Block-Based Motion Estimation Algorithm With Distance Dependent Thresholds," in proc. ICASSP. 2002.

[12] G. Sorwar, M. Murshed, and L. Dooley, "Distance Dependent Thresholding Search For Fast Motion Estimation in Video Coding," accepted in IEEE Asia-Pacific Conf. on Circuits and System (APCCAS'02). 2002. Kartika Plaza Hotel, Bali, Indonesia.

[13] G. Sorwar, M. Murshed, and L. Dooley, "Fast block-based true motion estimation using distance dependent thresholds (DTS)," accepted in the 6th International Conference on Signal Processing (ICSP'02). 2002. Beijing, China.

[14] G. Sorwar, M. Murshed, and L. Dooley, "Fast Motion Estimation in Video Coding Using Distance Dependent Thresholding," submitted in IEEE Trans. on circuit and systems for video tech., 2002.

[15] S. Haykin, Adaptive Filter Theory, Englewood Cliffs, NJ: Prectice-Hall, 1991:

[16] B. Widrow and J. M. E. Hoff, "Adaptive Switching Circuits," in IRE WESCON Conv. Rec. 1960.

[17] B. Widrow and S.D. Stearns, Adaptive Signal Processing, NJ: Prentice Hall,Englewood Cliffs, 1985.

[18] K. Meghriche, S. Femmam, B. Derras and M. Haddadi, "A non linear adaptive filter for digital data communication," Proc. of the Sixth Int. Symp. on Signal Proc. and its Appl., vol.1, pp.304-6, Piscataway, NJ, USA, 2001.

Integrated Technique with Neurocomputing for Temporal Video Segmentation

Golam Sorwar, Laurence Dooley and Manzur Murshed
Gippsland School of Computing and Information Technology
Monash University, Churchill Vic 3842, Australia

Abstract: Partitioning a video source into meaningful segments is an important step for video indexing. Many algorithms have been proposed for detecting video shot boundaries and classifying both shot and shot transition types. Different methods are suitable for different situations and most of the existing methods consider a threshold value determining the boundary between the two shots. However, selection of a generalized optimal threshold value is an extremely difficult task. In this paper, we propose an integrated method based on one of the popular soft computing techniques, namely neurocomputing, for temporal video segmentation that avoids problem with threshold calculations. We used a feedforward neural network trained using backpropagation algorithms. The soft computing model was trained using 80% of the frames data and the remaining 20% was used for testing and validation purposes. A performance comparison was made among the proposed soft computing method and traditional methods namely histogram difference, DCT difference, and Motion difference, for temporal shot detection.

1 Introduction

The use of digital video in many multimedia systems is becoming quite popular. Videos are playing an increasingly important role in both education and commerce. Besides the currently emerging services such as video-on demand and pay television, we see quite a number of new non-television like information services such as digital catalogues and interactive multimedia documents, including text, audio and video. Applications with digital video use time consuming fast forward or rewind to search, retrieval and get a quick overview of the content. In today's world, time is very expensive and time efficient media management is the key for the next generation. We need to devise new ways to index and access video content, which presents the visual information in compact forms such that the operator can quickly browse a video clip, retrieve content in different levels of detail and locate a segment of interest.

To enable time efficient and effective access, digital video has to be analyzed and processed to provide a representation that allows the user to locate any event in the video and browse it very quickly. Shot detection is the first step in this direction. A simplified structure of a content-based video database model is shown in Fig. 1.

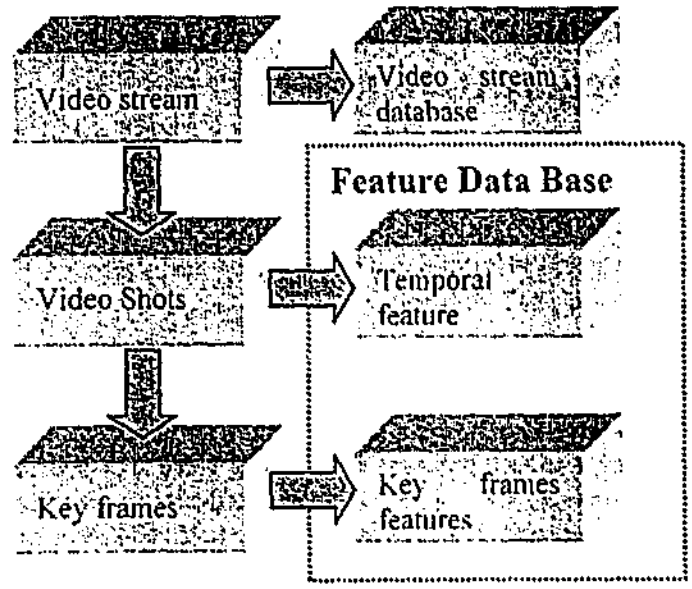


Fig. 1: A video date base model

As an enormous amount of information is available in each frame of video sequence, it is computationally expensive indexing based on each frame content. On the other hand if the video shots are properly detected, key frames (representatative frames) can be selected from each shot, it can represent the overall content of the whole sequence. So shot detection is the key part for extracting the representation frame, which can be used for video indexing.

The paper is structured as follows. Section 2 discusses the existing shot boundary detection techniques in detail. In Section 3, we present some basic theoretical aspects of neural networks while the proposed integrated method is described in Section 4. The results to verify the performance of the proposed method in terms of recall and precision values are presented in Section 5. The conclusions are provided in Section 6.

2 Related Work

The most important and fundamental processing step is to segment video into an appropriate set of units, which is known as shots. A shot is an uninterrupted video segment, that is, a sequence of consecutive frames generated as the result of a continuous single-camera operation. There are generally two types of shot transition-abrupt and progressive. Abrupt shot transition is the cut or camera break. It occurs in a single frame shown in Fig. 2(a). In this case, the prior 2 and the posterior 3 frames of the boundary show very different characteristics in terms of their content. But in case of progressive shot transition generated via the

application of mere elaborated edition effects that involve several frames such as fading, dissolves, wipes and many other types of gradual transition. An example of progressive shot transition (fade out) is shown in Fig. 2(b).

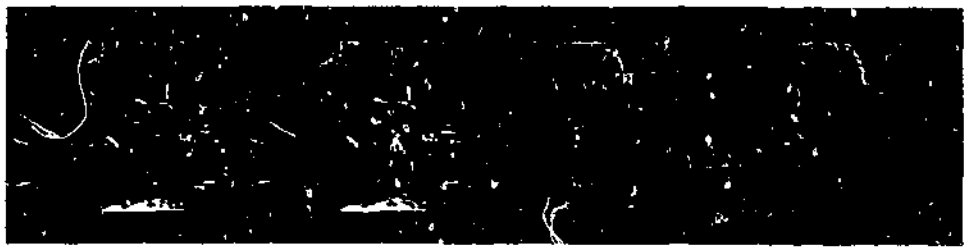


Fig. 2(a): Frames in camera break

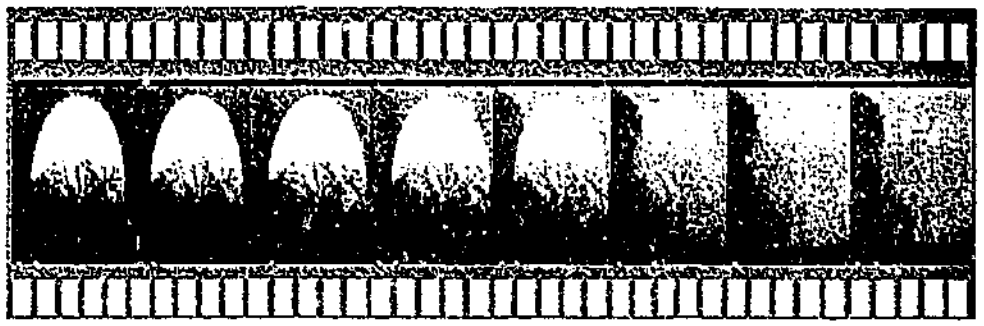


Fig. 2(b) Frames in fade

In general automatic shot boundary detection techniques are classified into the following categories: pixel based, statistics based, transformed based, histogram based and motion vectors based [12][18]. In pixel based methods, pixel-wise intensity difference is considered as the indicator for shot boundary detection. [12][13][10][9] computed the absolute sum of the pixel-by-pixel inter-frame difference and later compared it to a selected threshold. If the difference is more than the threshold value, a shot boundary is declared. It is very simple method, however the drawback associated with this method is that it is very sensitive to noise and camera and object motion. It is also difficult to adjust threshold value manually. In statistical difference based methods, large local changes lower most of the aforementioned detection algorithms' quality. Solution: compute the difference metric in image regions, instead of using the overall image, and later discard some of them for final sum up. Statistical methods expand on the idea of pixels differences by breaking the images into regions and comparing statistical measures of the pixels in those regions.[15][2][8] proposed some different shot boundary detection method based on content statistics such as mean, standard deviation, likelihood ratio. It is reasonably tolerant of noise, though its drawback is that it is slow due to complexity of the statistical formulas and it generates many false positive (wrong boundary detected as correct one). In order to effectively protect against camera operation and object motion, an option is to select a motion independent metric, like overall intensity histogram difference. Histograms are the

most common method used to detect shot boundary. In the simplest histogram method, the gray level or color histogram is computed and compared bin-wise difference with a threshold. If this bin-wise difference is above a threshold, a shot boundary is assumed. [6][14] used the color histogram change rate to find shot boundaries. This is very most common method and more robust to noise and object motion. According to [12][10], the histogram methods were a good trade-off between accuracy and speed. In order to properly detect gradual transitions such as wipes and dissolves, they used two thresholds. If the histogram difference fell between the thresholds, they tentatively marked it as the beginning of a gradual transition sequence, and succeeding frames ware compared against the first frame in the sequence. If the running difference exceeded the larger threshold, the sequence was marked as a gradual transition. An alternative to all these algorithms is to work with derived parameters directly extracted from the compressed sequence. [4][3] used differences in the size of JPEG [20] compressed frames to detect shot boundaries as a supplement to a manual indexing system. [16] found shot boundaries by comparing a small number of connected regions. They used differences in DCT coefficients of JPEG compressed frames as their measure of frame similarity, thus avoiding the need to decompress frames and increases the speed. But it generates too many false positive. [10][6] used motion vectors determined from block matching to detect whether or not a shot was a zoom or pan. [2] used the motion vectors extracted as part of the region-based pixel difference computation described above to decide if there is a large amount of camera or object motion in a shot. [17] detected the camera breaks using macroblock (16*16 pixels) information of P and B frames in MPEG [21] video sequences. Motion discontinuity will occur if there is any sudden change between two consecutive frames. This results in a significant drop of forward motion prediction coded macro blocks and can be easily detected by setting a threshold.

3 Artificial Neural Networks

Neural networks are computer algorithms inspired by the way information is processed in the nervous system [11]. An important difference between neural networks and other AI techniques is their ability to learn. The network "learns" by adjusting the interconnections between layers. When the network is adequately trained, it is able to generalize relevant output for a set of input data. A valuable property of neural networks is that of generalization, whereby a trained neural network is able to provide a correct matching in the form of output data for a set of previously unseen input data.

Learning typically occurs by example through training, where the training algorithm iteratively adjusts the connection weights (synapses). Backpropagation (BP) is one of the most famous training algorithms for multilayer perceptrons. BP is a gradient descent technique to minimize the error E for a particular training pattern. For adjusting the weight (w_{ij}) from the i-th input unit to the j-th output, in

the batched mode variant the descent is based on the gradient $\nabla E(\frac{\delta E}{\delta w_{ij}})$ for the total training set:

$$\Delta w_{ij}(n) = -\varepsilon^* \frac{\delta E}{\delta w_{ij}} + \alpha^* \Delta w_{ij}(n-1) \quad (1)$$

The gradient gives the direction of error E. The parameters ε and α are the learning rate and momentum respectively [1].

4 Proposed Integrated Algorithm

According to the discussion in section 2, it is very clear that the different methods are robust in different situation. The histogram comparison should be less sensitive to object motion than the DCT difference comparison algorithm, since it ignores the spatial change in a frame. But there may be the cases in which two images have similar histogram but completely different content. Again it is not robust against lighting change. So, if the different features are combined appropriately, a more desirable result can be expected. As a method of combining features, there can be a lot of alternative such as multi-level slicing, minimum distance method or maximum likelihood method. As another alternative, recently neural networks have been widely used for these purposes and have been successful in various applications. Considering these reported results and simplicity of implementation, a neural network of back error propagation model is adopted for the combination.

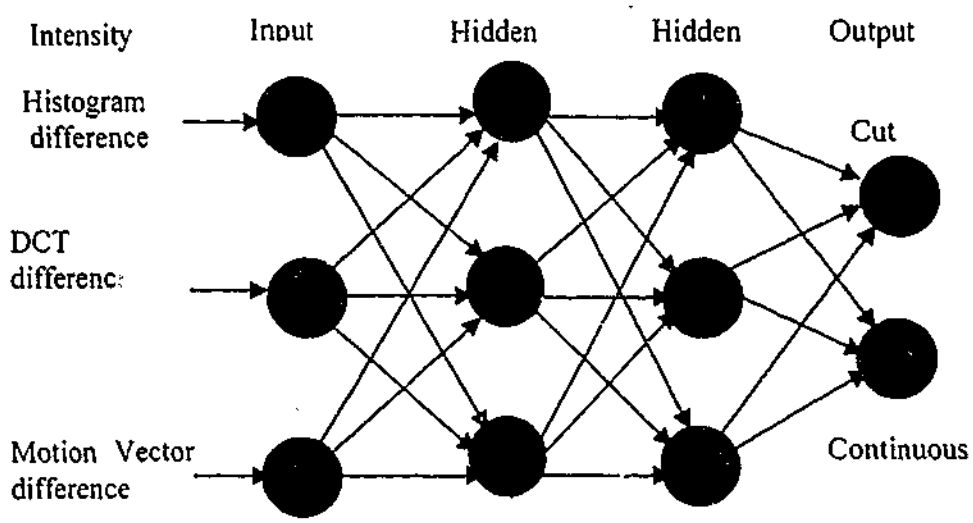


Fig. 3: Feed forward Neural network structure for shot detection
Histogram difference, DC component difference and Motion vector difference are considered as the input of the proposed algorithm

A structure of the adopted neural network is shown in Fig. 3. The feed forward neural network has an input layer of three neurons that correspond to the above three features, two hidden layers, and an output layer of two neurons that correspond to the shot boundary and continuous frame respectively.

5 Implementation and Evaluation

The shot detection algorithm proposed here was implemented and applied on a variety of video materials. For comparison the performance of this method we considered the histograms, DCT coefficient, motion vector difference method.

For histogram, we computed the 256 level gray scale histogram over the entire frame using the MATLAB version 6. The difference measure is the sum of the absolute bin-wise histogram difference that is as

$$H_{diff} = \frac{\sqrt{\sum_{m}^{M} (H_i(m) - H_{i+1}(m))^2}}{M} \tag{2}$$

Where M is the no of bins of each histogram, $H_i(m)$ is the current frame and $H_{i+1}(m)$ is the next frame.

The DCT coefficient difference method closely resembles the algorithm described by [3]. As the DC coefficient represents the average intensity of the block, we only considered the DC component of each block (8×8 pixels) for reducing the computational cost and concatenated them to produce a vector. The difference measure was computed by subtracting the inner product of the vectors of consecutive frames from one. If the difference exceeded the threshold, declare a possible shot boundary.

For motion vector difference method, we computed the magnitude of each block motion (16×16 pixels) extracted from mpeg encoded video sequences by [5] and produced a motion vector as in the DCT-based method. Then, comparing the difference with a threshold as DCT based to find the possible shot change.

To evaluate the efficiency of the proposed integrated method, we used the objective measure, *Recall* and *Precision* as in (3). *Recall* is the relevant detection rate from all the relevant items in the image database and *Precision* represents the correct detection rate.

$$Recall = \frac{C_d}{C_d + M}, \quad Precision = \frac{C_d}{C_d + F} \tag{3}$$

Where C_d is the number of correct detection, M is the number of missed item and F is the number of false positive. So a large recall value means that the correct shot boundaries are not missed very much and a large precision value means that relatively few wrong boundaries are declared as a boundary.

Of course these two are interdependent and closely related to threshold values. The threshold must be assigned so that it can tolerate variations in individual

frames while still ensuring a desired level of performance. A "tight" threshold makes it difficult for " imposters" to be falsely accepted by the systems, but at the risk of falsely rejecting true transitions. Inversely, a "loose" threshold enables transition to be accepted consistently, at the risk of falsely accepting "imposters". In order to achieve high accuracy in video partitioning, an appropriate threshold must be found. It is really difficult to find an appropriate threshold value manually in general. So threshold selection is another great problem for the traditional methods. For automatic selection of threshold, some researchers used the following relation,

Threshold = $\delta + \alpha\beta$, where δ and β are mean and the standard deviation of the frame-to-frame differences respectively and α is constant. This is however very much application dependent, so in our experiments, we evaluated the threshold according to experimental observation.

Table 1. Performance comparison of different methods with proposed method

	Recall	**Precision**
Histograms distance method	0.86	0.74
DCT coefficient distance method	0.93	0.90
Motion vector difference method	0.93	0.91
Proposed integrated method	0.97	0.93

For our experiment, we consider the different video clips such as movie; animation and sports contain approximately 5000 frames in total. For training the soft computing model, we used 80% datasets and remaining 20% datasets were used for testing purpose. After a trial and error approach we found that the neural network was giving good generalization performance when we had 2 hidden layers with 30 neurons each. According to Table 1, the proposed integrated algorithm shows better performance compare to other considered algorithms.

6 Conclusions

The extraction of the internal structure of the video contents is very important for the problem of searching and browsing of digital video. In this paper, we propose an integrated method for detecting abrupt shot boundary using artificial neural networks. On the contrary to the existing methods, the proposed method based on neurocomputing, avoids any threshold calculation, which is considered to be one of

the major problems in existing algorithms. Experimental results showed that the proposed integrated algorithm is more accurate, in detecting shot boundaries, than the other traditional approaches.

Incorporating progressive shot boundary detection and other soft computing models with extended database is on going work.

7 Acknowledgements

We gratefully acknowledge the valuable hints that came from the anonymous reviewers, which led to considerably to improving the quality of this paper.

References

[1] A. Abraham and B. Nath, Optimal design of neural nets using hybrid algorithms, In proceedings of 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), pp. 510-520, 2000.

[2] Shahraray, "Scene change detection and content-based sampling of video sequence," in Proc. of SPIE'95 digital Video Compression, vol. 2419, pp. 2-13, 1995.

[3] F. Arman, A. Hsu and M.-Y. Chiu, "Image Processing on Encoded Video Sequences," Multimedia Systems 1(5): 211-219, 1994.

[4] F. Arman, A. Hsu, and M.-Y. Chiu, "Feature management for large video databases," in Proc. SPIE Storage and retrieval for image and Video database, 1993.

[5] G. Sorwar, M. Murshed, and L. Dooley, "Fast Block-based True Motion Estimation using Variable Distance Dependent Thresholds in the Full-Search Algorithm, Submitted in Pattern Recognition Letters, 2001.

[6] H. Ueda et al., "Impact: An Interactive Natural-motion picture dedicated Multimedia Authoring systems," Proc. of the CHI'91, pp. 343-350, 1991.

[7] H. Yu, G. Bozdagi, and S. Harrington "Feature-based Hierarchical Video segmentation," in Proc. of ICIP'97, pp. 498-501, 1997.

[8] H. Zhang, J. Wu, D. Zhong, and S. W. Smoliar, " An integrated system for content-based video retrieval and browsing," Pattern Recognition, vol. 30, no. 4 , pp. 643-658, 1997.

[9] Hapmpapur, R. Jain, and T. Weymouth. "Digital Video segmentation," in proc. ACM conf. on Multimedia, 1994.

[10] H.-J. Zhang, A. Kankanhalli, and S.W. Smoliar. "Automatic Partitioning of Full-Motion Video," Multimedia Systems 1(1): 10-28 (1993).

[11] J.M. Zurada, Introduction to artificial neural systems, PWS Pub Co, 1992.

[12] John S. Boreczky and Lawrence A. Rowe: Comparison of Video Shot Boundary Detection Techniques. Storage and Retrieval for Image and Video Databases (SPIE), pp. 170-179, 1996.

[13] K. Otsuji, Y. Tonomura, "Projection detecting Filter for Video Cut detection," Proc. of the ACM Multimedia, pp. 251-256, Anaheim, August 1993.

[14] Nagasaka, and Y. Tanaka, "Automatic Video indexing and Full-video Search for objects Appearances," Proc. of IFIP 2nd Conference on Visual; database Systems, 113-127, 1991.

[15] R. Kasturi and R. Jain. Dynamic Vision. In Computer Vision: Principles, 1991.

[16] D. C. Little, Gulrukh Ahanger, R. J. Folz, J. F. Gibbon, F. W. Reeve, D. H. Schelleng and Dinesh Venkatesh, "A Digital On-Demand Video Service Supporting Content-Based Queries," Proc. of the ACM Multimedia'93, pp. 427-436, 1993.

[17] Y. Deng and B.S. Manjunath, "Content-based search of video using color, texture, and motion," Proc. of IEEE Intl. Conf. on Image Processing, vol. 2, pp. 534-37, 1997.

[18] Y. Rui, T.S. Huang, and S. Mehrotra, "Constructing Table-of-Content for Videos," ACM Multimedia Systems Journal, Special Issue Multimedia Systems on Video Libraries, vol.7, no.5, pp. 359-368, 1999.

[19] R. Zabih, J. Miller, and K. Mai, "A feature-based algorithm for detection and classifying scene brake," in Proc. ACM conf. On Multimedia, 1995.

[20] G.K. Wallace, "The JPEG still picture compression standard," Commun. ACM 34, 4, 1991.

[21] ISO/IEC JTC1/SC29/WG11, "Generic Coding of moving pictures and associated audio, ISO/IEC 13818-2, 1993.

DICTA2002: Digital Image Computing Techniques and Applications, 21--22 January 2002, Melbourne, Australia.

A Novel Filter for Block-Based Object Motion Estimation

Golam Sorwar, Manzur Murshed, and Laurence Dooley
Gippsland School of Computing and Information Technology
Monash University, Churchill Vic 3842, Australia
[redacted]

Abstract

Noises, in the form of false motion vectors, cannot be avoided while capturing block motion vectors using block-based motion estimation techniques. Similar noises are further introduced when the technique of global motion compensation is applied to obtain "true" object motion from video sequences, where both the camera and object motions are present. We observe that the performance of the mean and the median filters in removing false motion vectors, for estimating "true" object motion, is not satisfactory, especially when the size of the object is significantly smaller than the scene. In this paper we introduce a novel filter, named as the Mean-Accumulated-Thresholded (MAT) filter, in order to capture "true" object motion vectors from video sequences with or without the camera motion (zoom and/or pan). Experimental results on representative standard video sequences are included to establish the superiority of our filter compared with the traditional median and mean filters.

1. Introduction

Extracting motion parameters from image sequences has been a central theme in the areas of computer vision and image coding. There are many types of motion estimation algorithm such as pel-recursive [22], block-matching [8], and optical flow based method [7]. In general, block-matching algorithm [8] attracted wider acceptance due to its simplicity, robustness, and lesser hardware complexity which is already adopted by a large number of video coding standards (MPEG-1/2 and H.261/262/263 etc.).

The exhaustive block-matching *full-search* (FS) [8], where each candidate block is searched for the closest match within the entire search region, it generally provides reasonably good performance with the expense of high computational time.

Several fast algorithms have already been proposed to address the above issue. The three-step search algorithm (3SS) [12], the new three-step search (N3SS) [13], the four-step search algorithm (4SS) [17], and the cross-search algorithm [6] are based on the assumption that the block distortion measure increases as the checking points move away from the global minima. But this assumption does not hold true in the real world video sequences [4]. Moreover, search directions of the above algorithms can be ambiguous and therefore, may converge to local minima.

In true motion estimation, where object and/or camera motions are estimated, the FS algorithm tends to pick many "false" motion vectors even when no object motion is present in the search region. This is due to the fact that the distortion of an object in a video frame is proportional to its velocity and therefore, as the length of a motion vector grows so does the block difference error. The FS algorithm is, therefore, modified in our paper [19] by introducing distance dependent *linear threshold* (LT) and *exponential threshold* (ET) named as the Modified Full Search (MFS) algorithm. In this paper we use this MFS algorithm for estimating true block motions.

Block motion is governed by the movement due to the camera (pan and/or zoom) referred as global motion, movement of the objects referred as object motion or "true" motion, or both. Many motion estimation techniques ignore this aspect and make no distinction between the global and the local motion. However, separating these two classes of motions is significant for "true" object motion. In case where both the local and the global motions are present in the video sequences, "true" object motions (i.e., the local motion), necessary for object-based video representation, segmentation, and retrieval, can only be obtained by canceling out the global motion component from the block motion, known as *global motion compensation*.

Once the global motion is compensated from the estimated block motion, "true" object motion vectors are clustered in the blocks containing one or more objects. As the block motion estimation cannot be done with complete accuracy due to the limitation of block-based estimation techniques, a number of impulse noises (false motion vectors) are also likely to be introduced after the above processing along with the "true" object motion vectors. To retain only the "true" object motion vectors, we must filter out these impulse noises from the scene.

Many types of filters have already been proposed and examined for filtering impulse noises. Among them the *median filter* and the *mean filter* are widely used. While applied to reduce noises in an image, the median filter performs better than the mean filter as the mean filter often blurs the edges [5][21]. The same is not true for filtering out noises from the motion vectors, especially when objects are quite small compare to the size of the scene. In such cases, the median filter tends to remove significant number of "true" object motion vectors along the edge of the objects whereas the mean filter reduces the length of all the motion vectors, including the "true" ones. To address this issue we develop a new filter, named as the *Mean-Accumulative-Thresholded* (MAT) filter, which is successfully applied to a number of representative standard video sequences to capture the "true" object motion vectors.

The remainder of this paper is organized as follows. Section 2 describes the block motion estimation technique used in this paper. The parametric global motion estimation techniques are introduced in Section 3. In Section 4 the general process of estimating local (object) motion, including our proposed MAT filter, is discussed. Some experimental results are included in Section 5. Section 6 concludes the paper.

2. Block Motion Estimation

In [19], we observed that in true motion estimation, the FS algorithm tends to pick many "false" motion vectors even when no object motion is present in the search region. To address this issue we modified the FS algorithm (names as the MFS algorithm) by introducing distance dependent thresholds. The MFS algorithm not only avoids capturing a large number of "false" motion vectors but also reduces the search time significantly. In this paper we use the MFS algorithm for estimating true block motions.

3. Global Motion Estimation

If there is no local motion in a scene and only the camera is moving, the dynamics of the resulting video sequences can be adequately described by only a few camera operation parameters.

3.1. Motion Model

Techniques for *global motion estimation* (GME) have been proposed in [9][18][20]. Most of the GME methods differ in the parametric model to represent the camera motion as well as in the technique to estimate the parameters of the chosen model. Although a complex model results in a better description of the motion, it also leads to a greater difficulty in parameter estimation and higher computational complexity. Conversely, a simple model is sufficient enough to represent the global motion of a small video sequence, especially when the global motion is primarily used for compensating the camera motion from the block motion to obtain "true" object motion.

The conventional block-matching algorithm assumes that all the pixels in a block have equal displacements, and thus estimates one motion vector for each block. Let there be N blocks in a video frame. Let us assume that the motion vector of a block is the motion vector of the center pixel of that block. Let $(v_x(k), v_y(k))$ be the measured motion vector, according to our MFS algorithms explained in Section 2, of the block k, $k = 0, 1, \ldots, N\text{-}1$, whose center pixel's coordinates are $(s_x(k), s_y(k))$ with respect to the center of the frame.

For global motion estimation, we consider the 4-parameter motion model depicted in [18] with some modification. The generalized 4-parameter motion model for camera zoom and pan is defined as

$$\begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix} = \begin{bmatrix} a_1 s_x(k) \\ a_3 s_y(k) \end{bmatrix} + \begin{bmatrix} a_2 \\ a_4 \end{bmatrix} \tag{1}$$

where

$$a_1 = z_x \text{ and } a_2 = f_1(p_x, p_y) \tag{2(a)}$$

$$a_3 = z_y \text{ and } a_4 = f_2(p_y, z_y) \tag{2(b)}$$

In the above definition, z_x and z_y are the zoom factors along the x-axis and y-axis respectively, (p_x, p_y) is the pan vector.

3.2. Motion Parameter Estimation

Now consider the *iterative least-square estimation* algorithm for obtaining the optimal values for camera parameters (a_1, a_2, a_3, a_4) by using the following criteria:

$$\min_{a_1, a_2} \sum_{k=0}^{N-1} \left(v_x(k) - a_1 s_x(k) - a_2 \right)^2 \tag{3}$$

$$\min_{a_3, a_4} \sum_{k=0}^{N-1} \left(v_y(k) - a_3 s_y(k) - a_4 \right)^2 \tag{4}$$

By differentiating with respect to the parameters, and setting the derivatives to zero, we obtain the following solution as shown in (5, 6, 7, 8).

$$a_1 = \frac{N \sum_{k=0}^{N-1} v_x(k) s_x(k) - \left(\sum_{k=0}^{N-1} v_x(k) \right) \left(\sum_{k=0}^{N-1} s_x(k) \right)}{N \sum_{k=0}^{N-1} s_x^2(k) - \left(\sum_{k=0}^{N-1} s_x(k) \right)^2} \tag{5}$$

$$a_2 = \frac{\left(\sum_{k=0}^{N-1} v_x(k)\right)\left(\sum_{k=0}^{N-1} s_x^2(k)\right) - \left(\sum_{k=0}^{N-1} v_x(k)s_x(k)\right)\left(\sum_{k=0}^{N-1} s_x(k)\right)}{N\sum_{k=0}^{N-1} s_x^2(k) - \left(\sum_{k=0}^{N-1} s_x(k)\right)^2} \quad (6)$$

$$a_3 = \frac{N\sum_{k=0}^{N-1} v_y(k)s_y(k) - \left(\sum_{k=0}^{N-1} v_y(k)\right)\left(\sum_{k=0}^{N-1} s_y(k)\right)}{N\sum_{k=0}^{N-1} s_y^2(k) - \left(\sum_{k=0}^{N-1} s_y(k)\right)^2} \quad (7)$$

$$a_4 = \frac{\left(\sum_{k=0}^{N-1} v_y(k)\right)\left(\sum_{k=0}^{N-1} s_y^2(k)\right) - \left(\sum_{k=0}^{N-1} v_y(k)s_y(k)\right)\left(\sum_{k=0}^{N-1} s_y(k)\right)}{N\sum_{k=0}^{N-1} s_y^2(k) - \left(\sum_{k=0}^{N-1} s_y(k)\right)^2} \quad (8)$$

Since all the blocks are taken into consideration, the above estimate will be affected by the presence of the local motion. To eliminate this influence we use the above procedure iteratively, each time eliminating the blocks whose motion vectors do not match with the so-far-estimated global motion fields. As observed in [18], the iteration converges very quickly in our experiments.

4. Object Motion Estimation

In case where both the local and the global motions are present in the video sequences, "true" object motions can only be obtained by canceling out the global motion component from the block motion, known as *global motion compensation*.

Once the global motion parameters for the scene is calculated according to section 3, the "true" object motion vector $(o_x(k), o_y(k))$ of the block k, $k = 0, 1, \ldots, N-1$, can be calculated as:

$$\begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} = \begin{bmatrix} v_x(k) \\ v_y(k) \end{bmatrix} - \begin{bmatrix} a_1 s_x(k) \\ a_3 s_y(k) \end{bmatrix} - \begin{bmatrix} a_2 \\ a_4 \end{bmatrix} \quad (9)$$

Once the global motion is compensated from the estimated block motion, "true" object motion vectors are clustered in the blocks containing one or more objects. As the block motion estimation cannot be done with complete accuracy due to the limitation of block-based estimation techniques, a number of impulse noises are also likely to be introduced after the above processing along with the "true" object motion vectors. To retain only the "true" object motion vectors, we must filter out these impulse noises from the scene.

Many types of filters have already been proposed and examined for filtering impulse noises. Among them the *median filter* and the *mean filter* are widely used [2][5][11][16][21]. The median filter and its variants have already been applied in many applications for noise rejection from block motion vectors [1][10][14][23].

4.1. The Mean Filter

The idea of mean filtering is simply to replace each value with the mean ('average') value of its neighbors, including itself. This has the effect of smoothing values that are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter [24]. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×3 square kernel is used. Two major characteristics of the mean filter are:

- A single very unrepresentative value can significantly affect the mean value of its neighborhood.
- When the filter neighborhood straddles an edge, the filter will interpolate new values.

4.2. The Median Filter

Like the mean filter, the median filter considers each value in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the value with the mean of neighboring values, it replaces it with the median of those values. Two major characteristics of the median filter are:

- The median is a more robust average than the mean and so a single very unrepresentative value in a neighborhood will not affect the median value significantly.
- Since the median value must actually be one of the values in the neighborhood, the median filter does not create new unrealistic values when the filter straddles an edge.

4.3. The Mean-Accumulated-Thresholded (MAT) Filter

While applied to reduce noises in an image, the median filter performs better than the mean filter as the mean filter often blurs the edges [5][21]. The same is not true for filtering out noises from the motion vectors, especially when objects are quite small compare to the size of the scene. In such cases, the median filter tends to remove significant number of "true" object motion vectors along the edge of the objects. If the length of the "true" object motion vector is of same order of the introduced impulsive noises after the global motion compensation, a single iteration of the mean filtering would fail to remove all the impulsive noises, introduced by the global motion

compensation, even after using a threshold value. To address this issue we introduce a new filter, named as the *Mean-Accumulated-Thresholded* (MAT) filter.

The MAT filter has two phases. The first phase of the MAT filter is basically an iterative "in-place" application of the mean filter. But the major difference lies in how the "in-place" values are updated. In each iteration, the mean value is added on top, instead of replacing, the existing value as follows:

$$\begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} = \begin{bmatrix} o_x(k) \\ o_y(k) \end{bmatrix} + \begin{bmatrix} \text{mean}_x(k) \\ \text{mean}_y(k) \end{bmatrix} \quad (10)$$

where, $\text{mean}_x(k)$ and $\text{mean}_y(k)$ are the mean values, along the x-axis and the y-axis respectively, in the 3×3 neighborhood kernel for all k, $k = 0, 1, \ldots, N-1$.

With the mean and the median filters, even after the iterative "in-place" application, the length of the updated motion vectors will never exceed the maximum length of the original vectors in the neighborhood. But the same is not true for the MAT filter. Just after a few iterations (as low as 2), length of the "true" object motion vectors will be increased significantly, compare to the other vectors, including the impulses introduced during the global motion compensation and/or due to the limitations of the block-based motion estimation.

It is, therefore, highly likely that only the "true" object motion will be retained if the vectors, with length higher than a preset threshold, are selected as the last phase of the MAT filter.

5. Experimental Results

This MAT filter has been successfully applied to a number of representative standard video sequences to capture the "true" object motions vectors. Throughout the experiments, we use $M = N = d = 16$, i.e., each frame is divided into 16×16 pixel blocks and the size of the search region is 49×49 pixels, where at most 33^2 search points are used. All experiments are performed on the luminance (Y-component) of the frames.

In Figures 1–3, we present (a) the current frame, (b) the next frame, (c) block motion vectors computed using the MFS algorithm [19], (d) object motion vectors using the median filter of 3×3 kernel, (e) object motion vectors using the mean filter of 3×3 kernel, and (f) object motion vectors using the proposed MAT filter. In all the above-mentioned figures, the MAT filter outperforms the popular median filter, while capturing "true" object motion.

6. Conclusions and Discussion

The median and the mean filters and their variants have been used widely to remove noises from images and to smooth global motion vectors of video sequences. We have observed that the performance of these filters in removing false motion vectors for estimating "true" object motion is not satisfactory, especially when the size of the object is significantly smaller than the scene. In this paper we have introduced a novel filter, named as the Mean-Accumulated-Thresholded (MAT) filter, in order to capture "true" object motion vectors from video sequences with or without the camera motion (zoom and/or pan). Experimental results on representative standard video sequences have been included to establish the superiority of our filter compared with the mean and the median filters.

It is worth mentioning that although the MAT filter increases the length of the original object motion vectors significantly, it should not cause any problem as long as these vectors are not used for video coding. In case we are interested in capturing object motion vectors of "normal" length, it can easily be achieved by normalizing the MAT filtered vectors.

Although in our definition, the MAT filter uses the mean filter of 3×3 kernel, any other kernel size can also be used without loosing any generality. No study is done on the optimal kernel size to be used with the MAT filter. In future, we also like to explore whether different optimal kernel sizes exist for different video sequences with objects of different velocity.

7. References

[1] Avrithis Y.S., Doulamis N.D., Doulamis A.D. and Kollias S.D., "Efficient content representation in MPEG video databases," Proceedings. IEEE Workshop on Content-Based Access of Image and Video Libraries, pp. 91-5, 1998.

[2] Brownrig D.R.K., "The weighted median filter", Comm. of the ACM, vol. 27, no. 8, pp. 807-18, 1984.

[3] Chieh-Min Fan and Namazi N.M., "Simultaneous motion estimation and filtering of image sequences," IEEE Trans. of Image Processing, vol. 8, no. 12, pp. 1788-95, 1999.

[4] Chow H.-K. and Liou M.L., "Genetic motion search algorithm for video compression," IEEE Trans. on Circuits and Systems for Video Technology, vol. 3, pp. (s): 440–445, 1993.

[5] Davies E., Machine Vision: Theory, Algorithms and Practicalities, Academic Press, 1990.

[6] Ghanbari M, "The cross-search algorithm for motion estimation (image coding)," IEEE Trans. on Comm., vol. 38, pp. 950-953, 1990.

[7] Horn K.P. and Schunck B.G., "Determining Optical flow," Artificial Intelligence, Vol. 17, pp. 185-203, 1981.

[8] Jain J.R. and Jain A.K., "Displacement measurement and its application in inter frame image coding," IEEE Trans. Comm., vol. COM-29, pp. 1799-1808, 1984.

[9] Jozawa H., Kamikura K., Sagata A., Kotera H. and Watanabe H., "Two-stage motion compensation using adaptive global MC and local affine MC," IEEE Trans.

on Circuits & Systems for Video Technology, vol. 7, no. 1, pp. 75-85, 1997.

[10] Kim J.-G., Chang H. S., Kim J., and Kim H.M. "Efficient camera motion characterization for MPEG video indexing," IEEE International Conference on Multimedia and Expo. ICME2000.

[11] Kim J.-S. and Park H.W., "Adaptive 3D median filtering for restoration of an image sequence corrupted by impulse noise," Signal Processing: Image Comm., vol. 16, no. 7, pp. 657-68, 2001.

[12] Koga T., Iinuma K., Hirano A., Iijima Y. and Ishiguro T., "Motion-compensated inter frame coding for videoconferencing," IEEE National Telecommunications Conference, vol. 4, pp. G5. 1-5, 1981.

[13] Li R., Zeng B. and Liou M.L., "A new three-step search algorithm for block motion estimation," IEEE Trans. on Circuits & Systems for Video Technology, vol. 4, pp .438-442, 1994.

[14] Milanese R., Deguillaume F. and Jacot-Descombes A., "Efficient segmentation and camera motion indexing of compressed video," Real-Time Imaging 1999.

[15] Nam J.-Y., Seo J.-S., Kwak J.-S., Lee M.-H. and Yeong H.H., "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," IEEE Trans. on Consumer Electronics, vol. 46, pp. 934-942, 2000.

[16] Pitas and Venetsanopoulos A.., Nonlinear digital filters, Kluwer, Dodrecht, 1990.

[17] Po L.-M. and Ma W.-C., "Novel four-step search algorithm for fast block motion estimation," IEEE Trans. on Circuits & Systems for Video Technology, vol. 6, pp. 313-317, 1996.

[18] Rath G.B. and Makur A., "Iterative least squares and compression based estimations for a four-parameter linear global motion model and global motion compensation," IEEE Trans. on Circuits & Systems for Video Technology, vol. 9, no. 7, pp. 1075-99, 1999.

[19] G. Sorwar, M. Murshed, and L. Dooley, "Fast Block-based True Motion Estimation using Adaptive Distance Dependent Thresholds in the Full-Search Algorithm, Submitted in Pattern recognition Letters, 2001.

[20] Tse Y.T. and Baker R.L., "Global zoom/pan estimation and compensation for video compression," International Conference on Acoustics, Speech and Signal Processing, vol. 4. pp. 2725-8, 1991.

[21] Vernon D., Machine Vision, Prentice-Hall, 1991.

[22] Walker D.R. and Rao K.R., "Improved pel-recursive motion compensation," IEEE Trans. Comm., vol. COM-32, pp. 1128-1134, 1984.

[23] Zhong D. and Shih-Fu, "Video Object Model and Segmentation for Content Based Video Indexing," ISCAS'97, vol. 2, pp. 1492-1495, 1997.

[24] Boyle R. and Thomas R., Computer Vision: A First Course, Blackwell Scientific Publications, pp. 32-34, 1988.

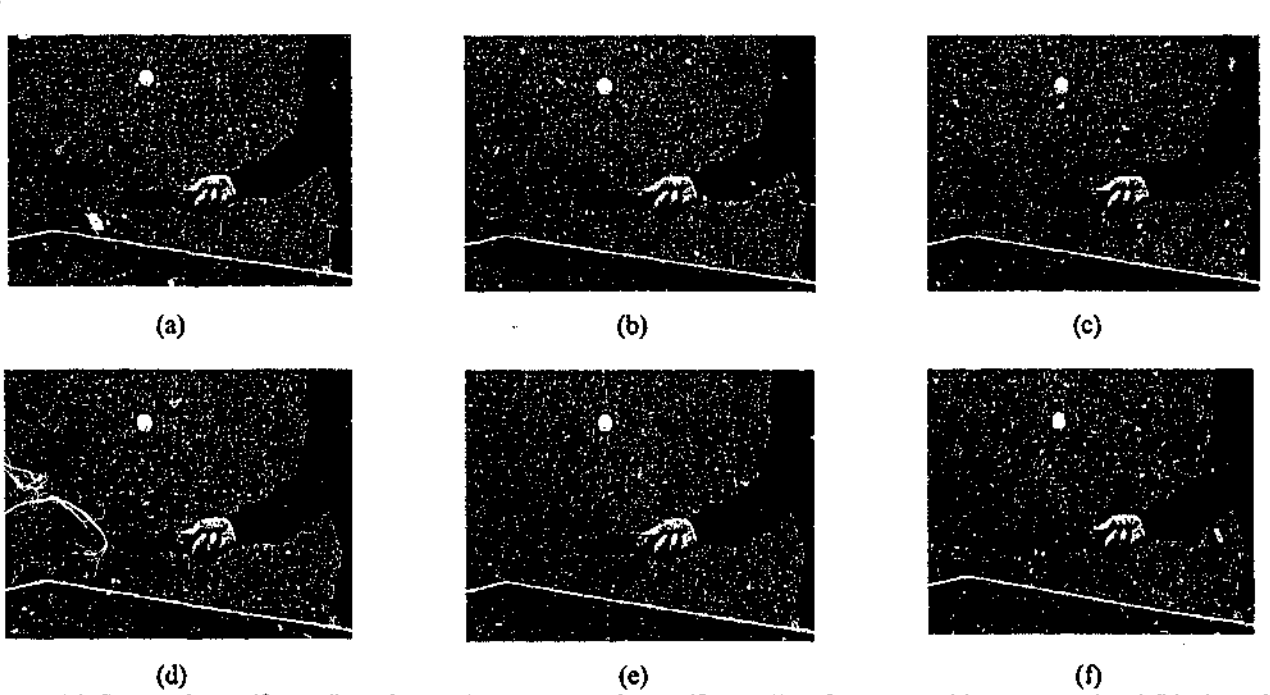
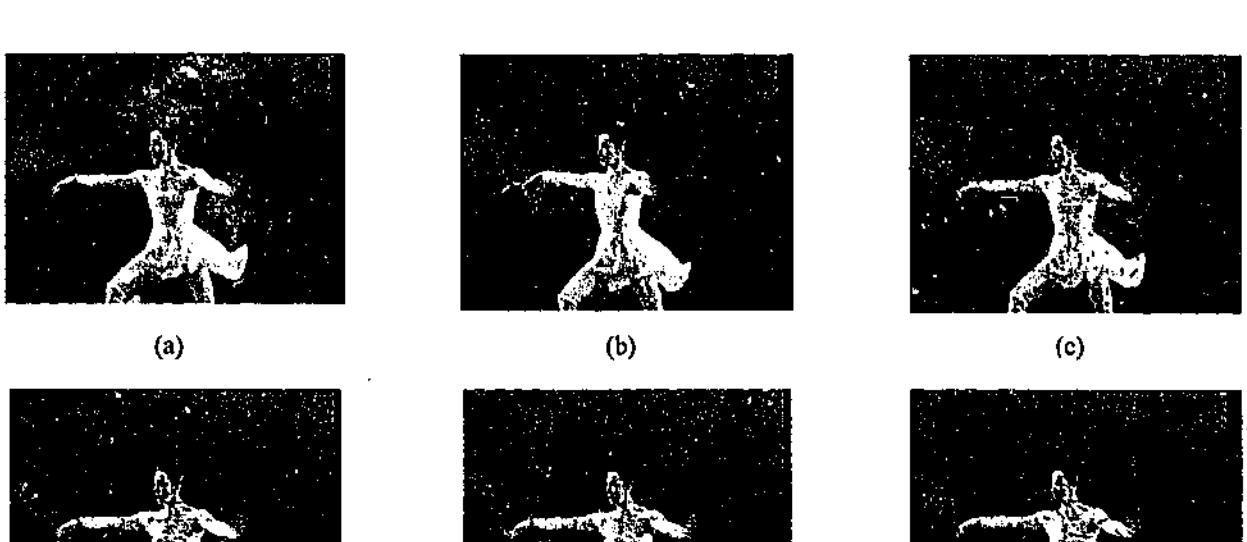


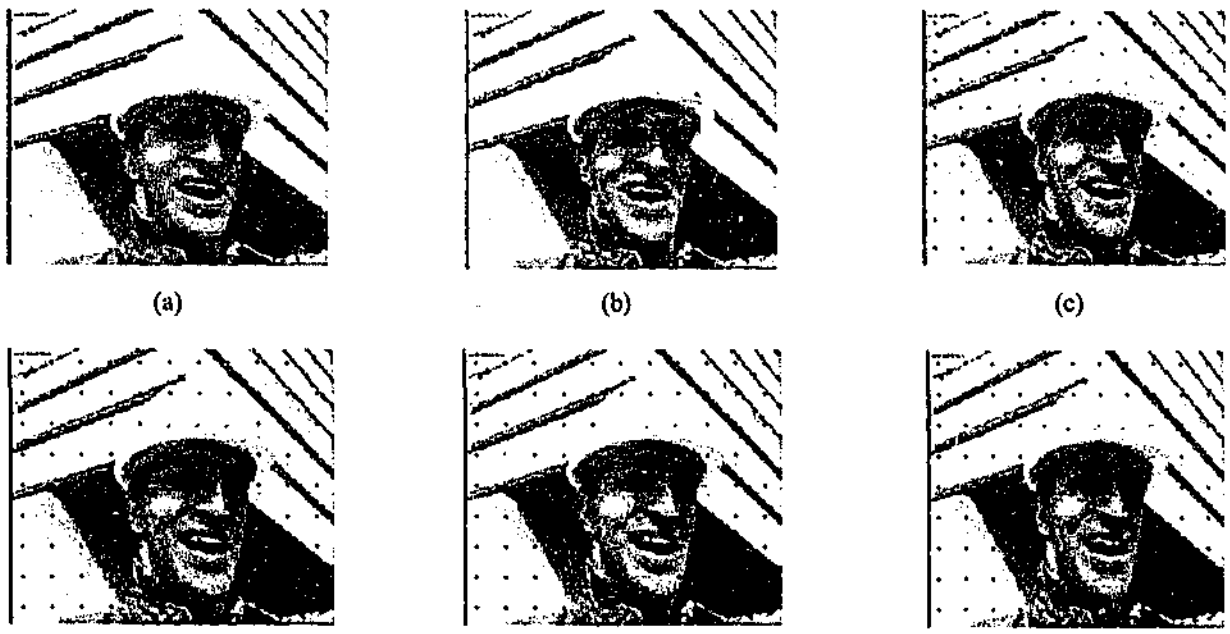
Figure 1: (a) Current frame (frame #32 of "Tennis"); (b) Next frame, (frame #33 of the same video sequence); (c) Block motion vectors computed using the LT algorithm [19]; (d) Object motion vectors using the median filter of 3×3 kernel; (e) Object motion vectors using the mean filter of 3×3 kernel; (f) Object motion vectors using the MAT filter.



(a) (b) (c)

(d) (e) (f)

Figure 2: (a) Current frame (frame #99 of "Ballet"); (b) Next frame, (frame #100 of the same video sequence); (c) Block motion vectors computed using the LT algorithm [19]; (d) Object motion vectors using the median filter of 3×3 kernel; (e) Object motion vectors using the mean filter of 3×3 kernel; (f) Object motion vectors using the MAT filter.



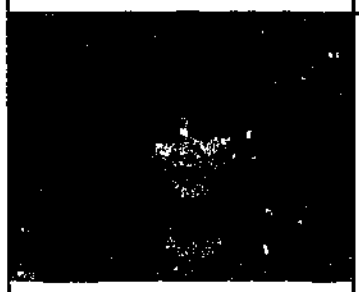
(a) (b) (c)

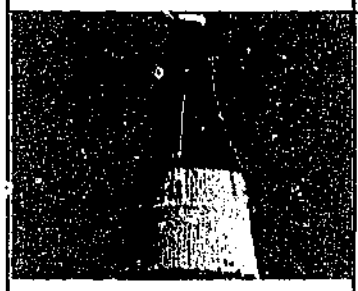
(d) (e) (f)

Figure 3: (a) Current frame (frame #15 of "Foreman"); (b) Next frame, (frame #16 of the same video sequence); (c) Block motion vectors computed using the LT algorithm [19]; (d) Object motion vectors using the median filter of 3×3 kernel; (e) Object motion vectors using the mean filter of 3×3 kernel; (f) Object motion vectors using the MAT filter.

Test Video Sequences

Sequences	Name	No. of Frames	Resolution	Motion Description
	*Table Tennis*	149	SIF (352×240)	Object translation, camera zooming and panning. Note: a shot change at Frame #90.
	*Football*	345	CIF (360×240)	General motion pictures with high motion activity.
	*Flower Garden*	150	SIF (352×240)	Fast panning with high motion activity.
	*Salesman*	300	CIF (360×288)	Head and shoulder type sequence with very low object translation with low motion activity.

Sequences	Name	No. of Frames	Resolution	Motion Description
	*Carphone*	382	QCIF (176×144)	Fast object translation and camera panning.
	*Miss America*	150	QCIF (176×144)	Head and shoulder type sequence with very low object translation with low motion activity.
	*Foreman*	298	QCIF (176×144)	Object translation and camera panning
	*Suise*	51	QCIF (176×144)	Head and shoulder type sequence with very low object translation with low motion activity.
	*Cycle*	413	SIF (352×240)	Object translation and camera panning

Sequences	Name	No. of Frames	Resolution	Motion Description
	*Rocket*	50	QCIF (176×144)	Very fast object translation.
	*Son*	174	CIF (352×288)	Object translation and camera panning.
	*Ballet*	100	SIF (352×240)	Object translation and camera panning

Supplementary Results for the DTS Algorithm Presented in Chapter 3

Block-matching algorithms	Integer-pel			Half-pel		
	MSE	PSNR [dB]	SP	MSE	PSNR [dB]	SP
FS/LT(0)	335.70	22.94	202.05	275.76	23.73	210.05
LT(2)	335.96	22.94	123.71	278.87	23.68	131.71
LT(4)	340.67	22.88	82.57	281.55	23.64	90.57
LT(6)	356.53	22.68	57.10	291.67	23.48	65.10
LT(8)	380.03	22.41	41.75	308.27	23.24	49.75
LT(10)	405.82	22.14	32.55	327.18	22.98	40.55
LT(12)	430.26	21.89	26.88	345.45	22.75	34.88
LT(14)	451.26	21.68	23.18	361.30	22.55	31.18
LT(16)	471.12	21.50	20.51	376.46	22.37	28.51
LT(18)	489.40	21.34	18.50	390.73	22.21	26.50
LT(20)	505.44	21.19	16.96	403.08	22.08	24.96
TSS	370.32	22.51	23.11	309.11	23.22	31.11
NTSS	363.81	22.59	20.79	303.35	23.31	30.79

Table C.1: Comparison of average MSE and PSNR per pixel, and search points (SP) per motion vector for the *Football* sequence (1-80 frames) with different BMAs.

Block-matching algorithms	Integer-pel			Half-pel		
	MSE	PSNR [dB]	SP	MSE	PSNR [dB]	SP
FS/LT(0)	126.34	27.94	197.14	102.72	28.01	205.14
LT(2)	127.25	27.63	75.68	103.14	28.00	83.68
LT(4)	131.64	27.48	38.14	105.34	27.91	46.14
LT(6)	138.26	27.27	26.03	109.07	27.75	34.03
LT(8)	147.59	26.98	20.35	114.64	27.54	28.35
LT(10)	157.34	26.69	16.88	120.91	27.32	24.88
LT(12)	166.71	26.42	14.65	127.04	27.09	22.65
LT(14)	175.16	26.21	13.30	132.08	26.92	21.30
LT(16)	183.65	26.00	12.36	137.22	26.76	20.36
LT(18)	191.71	25.82	11.66	142.57	26.59	19.66
LT(20)	198.84	25.68	11.14	146.61	26.47	19.14
TSS	190.81	25.32	23.01	159.18	26.11	31.01
NTSS	159.10	26.11	20.83	127.85	27.06	28.83

Table C.2: Comparison of average MSE and PSNR per pixel, and search points (SP) per motion vector for the *Table Tennis* sequence (1-80 frames) with different BMAs.

Block-matching algorithms	Integer-pel			Half-pel		
	MSE	PSNR [dB]	SP	MSE	PSNR [dB]	SP
FS/LT(0)	15.71	36.17	192.04	13.30	36.89	200.04
LT(2)	15.75	36.16	29.60	13.32	36.89	37.6
LT(4)	15.99	36.09	13.27	13.43	36.85	21.27
LT(6)	16.26	36.02	9.90	13.59	36.80	17.90
LT(8)	16.65	35.92	8.89	13.82	36.73	16.89
LT(10)	16.85	35.86	8.49	13.92	36.70	16.49
LT(12)	17.08	35.81	8.28	14.04	36.66	16.28
LT(14)	17.35	35.74	8.17	14.17	36.62	16.17
LT(16)	17.57	35.68	8.10	14.30	36.58	16.10
LT(18)	17.88	35.61	8.05	14.47	36.52	16.05
LT(20)	18.10	35.55	8.02	14.62	36.48	16.02
TSS	16.40	35.98	21.89	13.77	36.74	29.89
NTSS	16.01	36.09	15.93	13.44	36.85	23.93

Table C.3: Comparison of average MSE and PSNR per pixel, and search points (SP) per motion vector for the *Salesman* sequence (1-80 frames) with different BMAs.

Supplementary Results for the ACDTS and ACDSDTS Algorithms Presented in Chapter 4

Block-matching algorithms		MSE	PSNR [dB]	SP
ACDTS	C_L=2	103.23	27.99	82.8
	C_L=4	105.07	27.92	44.97
	C_L=6	108.60	27.77	32.42
	C_L=8	113.56	27.58	26.95
	C_L=10	119.44	27.36	23.81
	C_L=12	125.35	27.15	21.88
	C_L=14	130.03	26.99	20.59
	C_L=16	135.47	26.81	31.09
	C_L=18	140.35	26.66	31.03
	C_L=20	144.21	26.54	18.57
ACDSDTS	C_L=2	111.42	27.66	46.92
	C_L=4	112.64	27.61	27.70
	C_L=6	114.67	27.54	21.30
	C_L=8	118.89	27.38	18.50
	C_L=10	123.23	27.22	16.90
	C_L=12	128.93	27.03	15.86
	C_L=14	133.30	26.88	15.22
	C_L=16	137.81	26.74	14.77
	C_L=18	140.70	26.65	14.41
	C_L=20	145.56	26.50	14.14
FS		102.72	28.01	205.14
TSS		159.18	26.11	30.75
NTSS		127.85	27.06	28.83

Table D.1: Average MSE and PSNR per pixel, and average search points (SP) per motion vector comparison of different BMAs for the *Table Tennis* sequence (1-80 frames).

Block-matching algorithms		MSE	PSNR [dB]	SP
ACDTS	C_L=2	13.30	36.89	37.62
	C_L=4	13.39	36.86	21.25
	C_L=6	13.52	36.82	17.87
	C_L=8	13.67	36.77	16.86
	C_L=10	13.75	36.75	16.46
	C_L=12	13.85	36.72	16.25
	C_L=14	13.94	36.69	16.15
	C_L=16	14.06	36.65	16.09
	C_L=18	14.18	36.61	16.04
	C_L=20	14.26	36.59	16.02
ACDSDTS	C_L=2	13.42	36.85	23.67
	C_L=4	13.47	36.84	15.29
	C_L=6	13.56	36.81	13.56
	C_L=8	13.65	36.78	13.05
	C_L=10	13.78	36.74	12.84
	C_L=12	13.91	36.70	12.73
	C_L=14	13.97	36.68	12.68
	C_L=16	14.10	36.64	12.65
	C_L=18	14.27	36.59	12.62
	C_L=20	14.46	36.53	12.61
FS		13.30	36.89	200.04
TSS		13.77	36.74	29.89
NTSS		13.44	36.85	23.93

Table D.2: Average MSE and PSNR per pixel, and search points (SP) per motion vector comparison of different BMAs for the *Salesman* sequence (1-80 frames).

Block-matching algorithms		MSE	PSNR [dB]	SP
ACDTS	C_L=2	3.159	43.135	22.63
	C_L=4	3.158	43.137	17.04
	C_L=6	3.155	43.141	15.93
	C_L=8	3.155	43.141	15.54
	C_L=10	3.155	43.141	15.33
	C_L=12	3.155	43.141	15.28
	C_L=14	3.155	43.141	15.26
	C_L=16	3.155	43.141	15.25
	C_L=18	3.155	43.141	15.25
	C_L=20	3.155	43.141	15.25
ACDSDTS	C_L=2	3.163	43.130	16.03
	C_L=4	3.161	43.133	13.20
	C_L=6	3.162	43.131	12.63
	C_L=8	3.159	43.135	12.43
	C_L=10	3.160	43.134	12.33
	C_L=12	3.160	43.134	12.30
	C_L=14	3.160	43.134	12.29
	C_L=16	3.160	43.134	12.28
	C_L=18	3.160	43.134	12.28
	C_L=20	3.160	43.134	12.28
FS		3.150	43.152	176.21
TSS		3.170	43.123	27.67
NTSS		3.158	43.135	23.14

Table D.3: Average MSE and PSNR per pixel, and average search points (SP) per motion vector comparison of different BMAs for the *Miss America* sequence (1-80 frames).

Supplementary Results for the FADTS Algorithm Presented in Chapter 5

Table Tennis			*Salesman*		
Target Speed (SP)	Actual SP	Actual Error MSE	Target Speed (SP)	Actual SP	Actual MSE
16	15.54	95.05	16	15.61	12.96
20	19.42	85.04	20	19.75	12.95
25	24.94	82.56	25	23.74	12.95
30	29.97	81.59	30	30.22	12.92

Table E.1: Search speed adaptation for *Table Tennis* and *Salesman* video sequence (149 and 300 frames respectively).

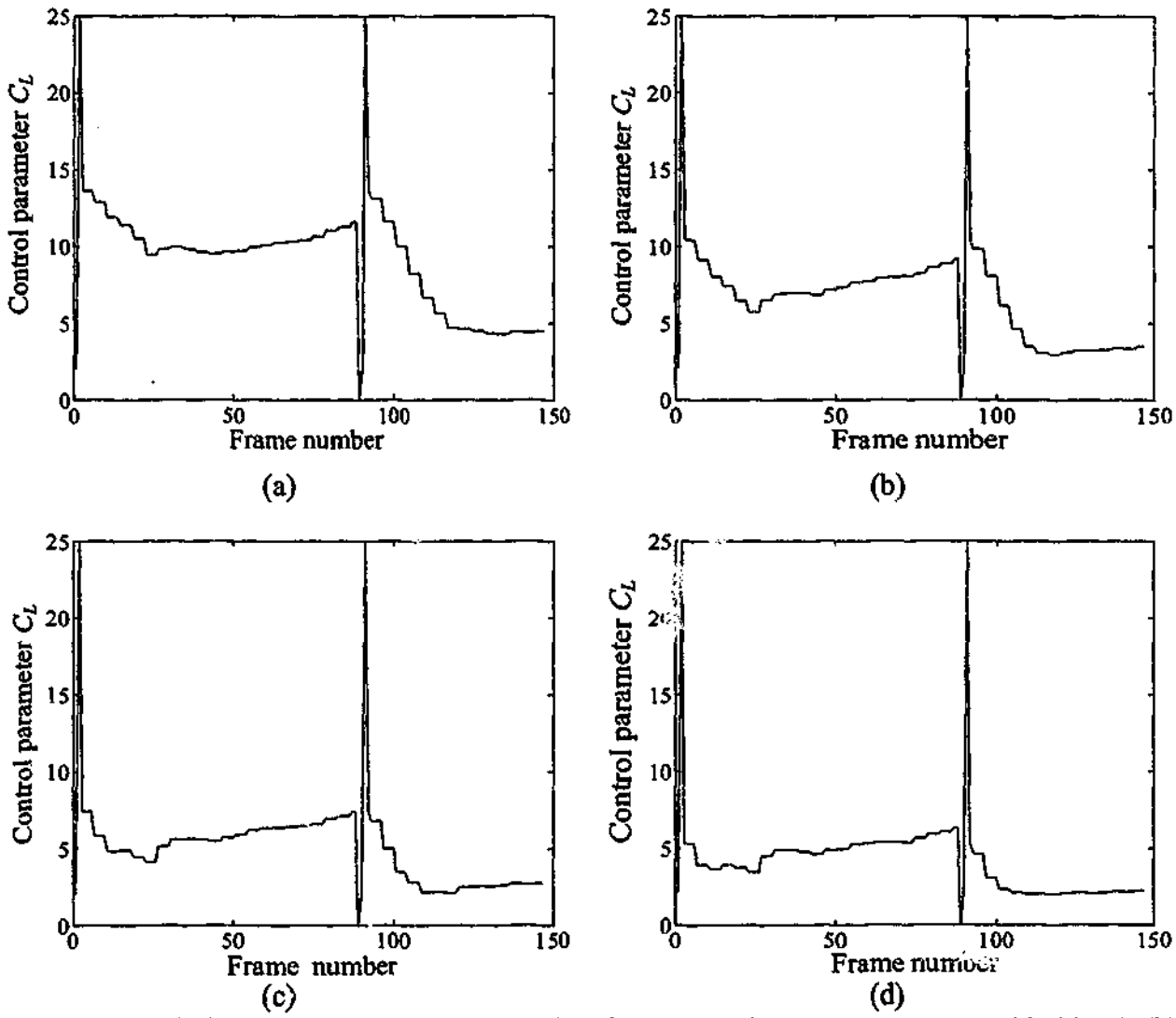


Fig. E.1: Threshold control parameter adaptation for the *Table Tennis* sequence with (a) 16, (b) 20, (c) 25, and (d) 30 average search points (SP) per motion vector.

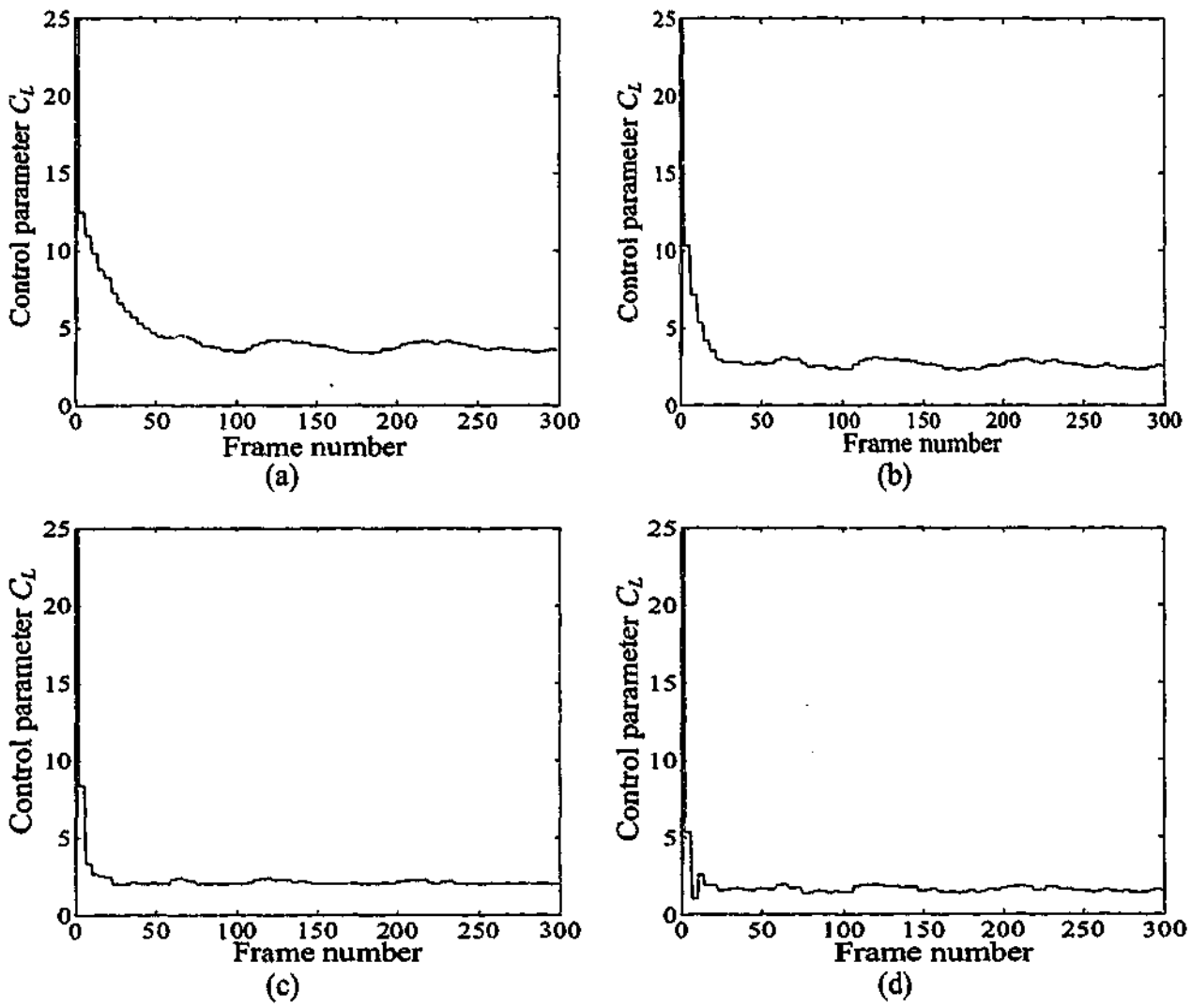
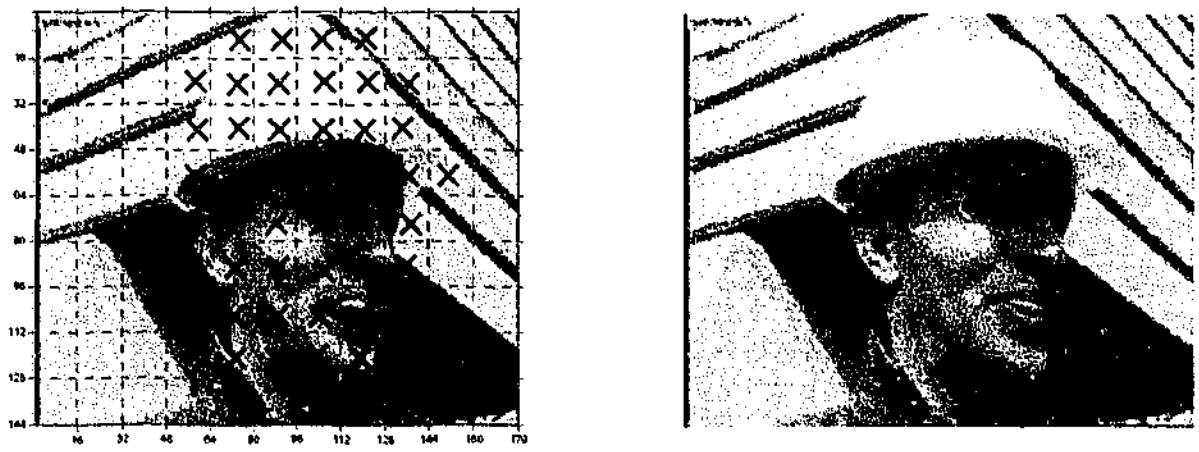


Fig. E.2: Threshold control parameter adaptation for the *Salesman* sequence with (a) 16, (b) 20, (c) 25, and (d) 30 average search points (SP) per motion vector.

Supplementary Results of the DTS and MAT Filter Presented in Chapter 6

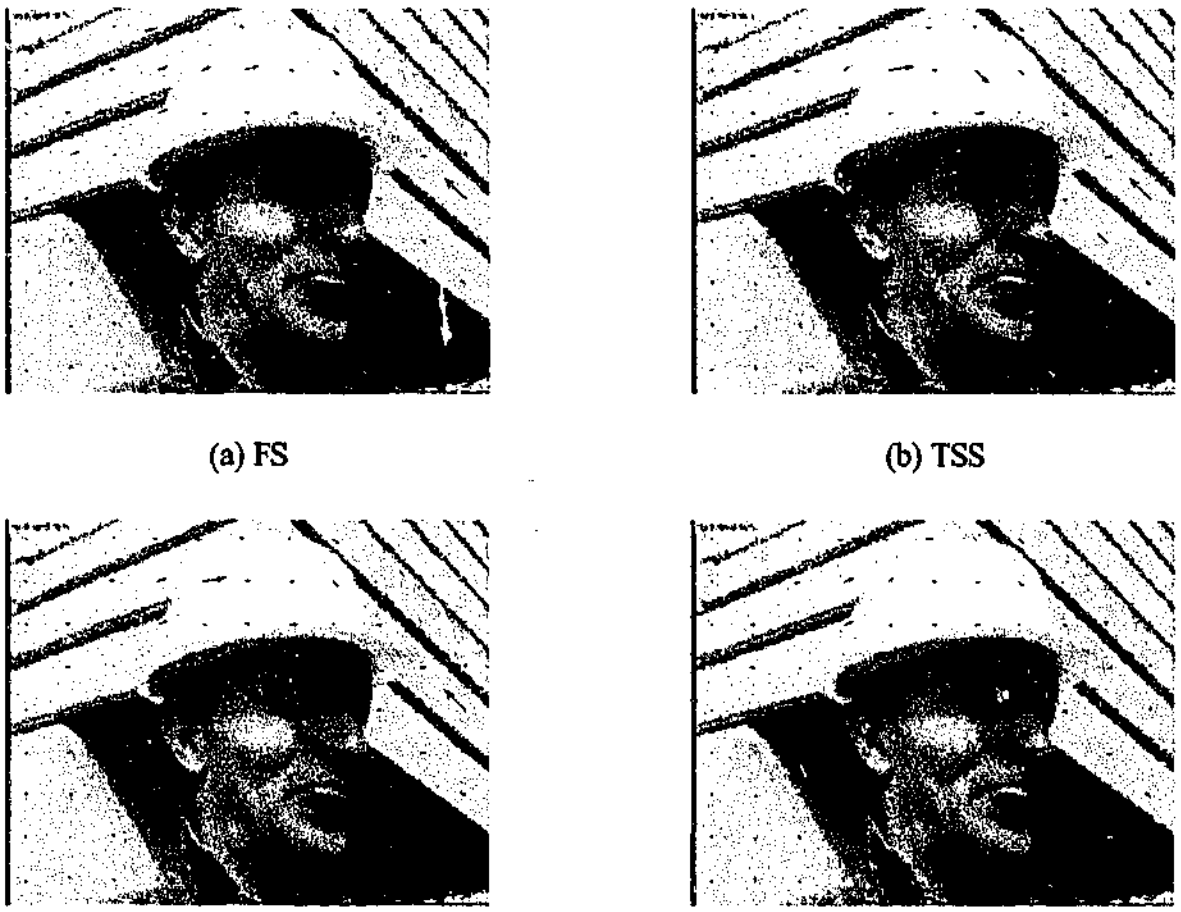
No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0	90.0	20.3	1.0	33.3	4.0	4.0	33.3	8.1	4.0	90.0	20.3	1.0
	73.3	7.0	2.0	20.0	2.7	5.0	20.0	6.3	5.0	73.3	7.0	2.0
	46.7	4.0	3.0	16.7	2.3	6.0	13.3	6.3	6.0	46.7	4.0	3.0
	30.0	1.3	4.0	13.3	1.3	7.0	6.7	7.1	7.0	30.0	1.7	4.0
	16.7	0.3	5.0	6.7	0.7	8.0	3.3	14.3	8.0	16.7	0.3	5.0
	13.3	0.0	6.0	0.0	0.0	9.0	0.0	0.0	9.0	13.3	0.0	6.0
1	46.7	3.7	6.0	46.7	3.0	5.0	43.3	34.3	5.0	100.0	37.0	1.0
	33.3	2.0	7.0	36.7	2.7	6.0	36.7	27.7	6.0	86.7	7.3	2.0
	23.3	1.0	6.0	26.7	2.3	7.0	26.7	19.3	7.0	76.7	5.3	3.0
	16.7	0.7	7.0	16.7	1.3	8.0	16.7	12.7	8.0	63.3	3.3	4.0
	13.3	0.3	8.0	16.7	0.3	9.0	10.0	5.7	9.0	46.7	1.7	5.0
	13.3	0.0	9.0	13.3	0.0	10.0	6.7	3.3	10.0	40.0	0.0	6.0
2	60.0	3.7	7.0	56.7	2.3	8.0	43.3	20.7	9.0	90.0	4.7	4.0
	50.0	3.0	8.0	46.7	2.0	9.0	36.7	15.7	10.0	76.7	3.0	5.0
	40.0	1.7	9.0	43.3	1.3	10.0	26.7	13.3	11.0	76.7	2.3	6.0
	26.7	1.0	10.0	26.7	0.7	11.0	20.0	10.7	12.0	63.3	1.0	7.0
	26.7	0.3	11.0	23.3	0.3	12.0	13.3	6.3	13.0	56.7	0.3	8.0
	16.7	0.3	12.0	16.7	0.0	13.0	10.0	3.7	14.0	46.7	0.0	9.0
3	63.3	3.0	15.0	83.3	8.3	9.0	50.0	17.0	15.0	86.7	5.3	8.0
	60.0	2.7	16.5	76.7	3.0	10.5	43.3	14.3	16.5	80.0	2.0	9.0
	46.7	2.0	17.0	66.7	2.0	12.0	30.0	11.0	18.0	80.0	1.0	10.0
	40.0	0.7	18.5	56.7	1.7	13.5	23.3	8.0	19.5	73.3	1.0	11.0
	26.7	0.3	19.0	56.7	0.7	15.0	16.7	4.7	21.0	70.0	0.3	12.0
	26.7	0.0	19.5	50.0	0.0	16.5	10.0	1.7	22.5	63.3	0.0	13.0
4	63.3	2.0	21.5	83.3	8.3	16.0	66.7	8.0	34.0	100.0	11.3	10.0
	53.3	1.7	22.0	80.0	5.0	18.0	60.0	6.3	36.0	93.3	8.7	12.0
	50.0	1.3	22.5	80.0	3.3	20.0	46.7	4.7	38.0	86.7	6.3	14.0
	43.3	0.3	23.0	66.7	1.0	22.0	40.0	2.7	40.0	86.7	4.7	16.0
	40.0	0.3	23.5	60.0	0.3	24.0	26.7	1.3	42.0	80.0	1.3	18.0
	36.7	0.0	24.0	56.7	0.0	26.0	23.3	0.7	44.0	80.0	0.0	20.0
5	63.3	2.0	52.0	86.7	10.0	27.0	70.0	6.0	60.0	86.7	6.0	27.5
	56.7	1.3	54.0	83.3	7.7	30.0	66.7	4.0	61.0	86.7	4.0	30.0
	50.0	1.0	56.0	80.0	4.7	33.0	53.3	3.3	62.0	80.0	2.7	32.5
	46.7	0.3	58.0	80.0	3.0	36.0	36.7	1.7	63.0	80.0	1.3	35.0
	43.3	0.3	60.0	73.3	1.7	39.0	36.7	1.0	64.0	80.0	0.7	37.5
	43.3	0.0	62.0	66.7	0.3	42.0	30.0	0.7	65.0	83.3	0.0	40.0

Table F.1: Performance comparison of the DTS algorithm with and/without the MAT filter in capturing *true* object motion vectors for the *Table Tennis* (frames #32 and #33) video sequence.



(a) (b)

Fig. F.1: (a) Current Frame #8 in which X indicates the moving macroblocks; and (b) Reference frame #9 of the *Foreman* sequence.



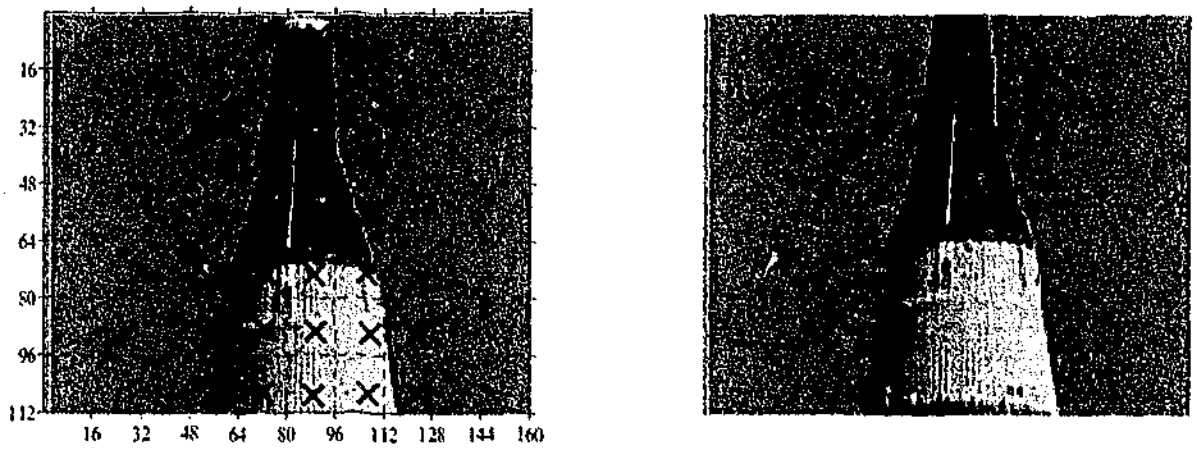
(a) FS (b) TSS

(c) NTSS (d) DTS

Fig. F.2: Motion vector obtained from all four search algorithms applied to the frame pair shown in Fig. F.1 for the *Foreman* sequence.

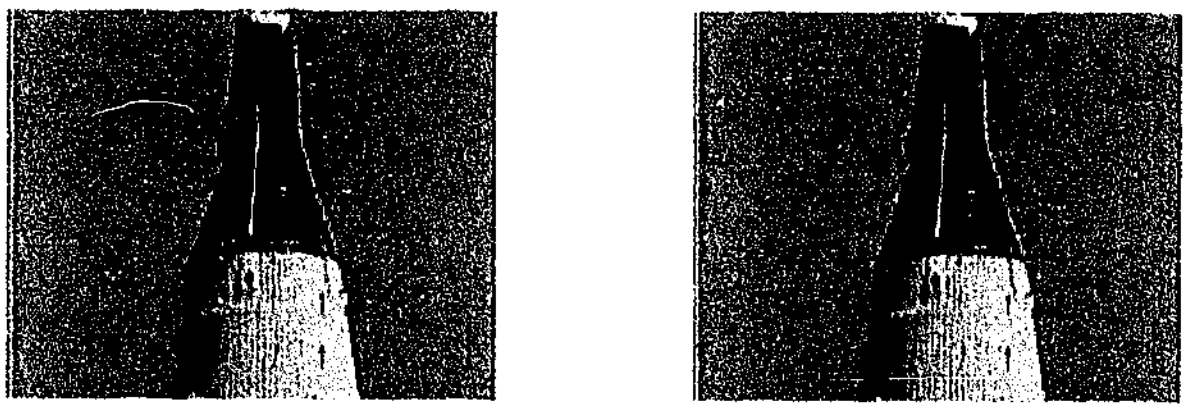
No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0	21.7	15.1	2.2	2.2	3.8	5.2	2.2	5.7	4.8	15.2	5.7	1.8
	4.3	13.2	2.4	2.2	3.8	5.4	2.2	5.7	5.0	15.2	5.7	2.0
	4.3	13.2	2.6	2.2	3.8	5.6	2.2	5.7	5.2	15.2	5.7	2.2
	4.3	13.2	2.8	2.2	1.9	5.8	2.2	3.8	5.4	8.7	3.8	2.4
	4.3	13.2	3.0	2.2	1.9	6.0	2.2	3.8	5.6	4.3	3.8	2.6
	0.0	11.3	3.2	0.0	1.9	6.2	0.0	3.8	5.8	0.0	1.9	2.8
1	23.9	11.3	3.2	2.2	1.9	6.2	2.2	3.8	4.8	34.8	5.7	2.2
	21.7	11.3	3.4	2.2	1.9	6.4	2.2	3.8	5.0	23.9	5.7	2.4
	13.0	11.3	3.6	2.2	1.9	6.6	2.2	3.8	5.2	21.7	5.7	2.6
	8.7	11.3	3.8	2.2	1.9	6.8	2.2	3.8	5.4	17.4	5.7	2.8
	4.3	11.3	4.0	2.2	1.9	7.0	2.2	3.8	5.6	13.0	1.9	3.0
	0.0	11.3	4.2	2.2	0.0	7.2	0.0	3.8	5.8	10.9	0.0	3.2
2	15.2	3.8	6.6	19.6	1.9	6.2	4.3	5.7	7.8	69.6	7.5	2.8
	8.7	3.8	6.8	17.4	1.9	6.4	4.3	3.8	8.0	63.0	3.8	3.0
	8.7	3.8	7.0	17.4	1.9	6.6	4.3	3.8	8.2	58.7	1.9	3.2
	6.5	1.9	7.2	15.2	1.9	6.8	4.3	1.9	8.4	50.0	1.9	3.4
	4.3	1.9	7.4	13.0	1.9	7.0	2.2	1.9	8.6	43.5	1.9	3.6
	0.0	1.9	7.6	10.9	0.0	7.2	0.0	1.9	8.8	39.1	0.0	3.8
3	37.0	1.9	9.2	71.7	1.9	7.3	13.0	1.9	13.0	80.4	9.4	4.0
	37.0	1.9	9.4	67.4	1.9	7.6	13.0	1.9	13.5	80.4	7.5	4.2
	37.0	1.9	9.6	65.2	1.9	7.9	4.3	1.9	14.0	78.3	5.7	4.4
	37.0	1.9	9.8	63.0	1.9	8.2	2.2	1.9	14.5	78.3	1.9	4.6
	37.0	1.9	10.0	58.7	1.9	8.5	2.2	1.9	15.0	71.7	1.9	4.8
	32.6	0.0	10.2	56.5	0.0	8.8	0.0	1.9	15.5	69.6	0.0	5.0
4	76.1	5.7	8.5	82.6	5.7	8.4	21.7	1.9	21.0	82.6	9.4	6.8
	76.1	3.8	9.0	82.6	5.7	8.7	19.6	1.9	22.0	80.4	7.5	7.0
	76.1	1.9	9.5	80.4	3.8	9.0	17.4	1.9	23.0	80.4	5.7	7.2
	76.1	1.9	10.0	80.4	3.8	9.3	15.2	1.9	24.0	80.4	3.8	7.4
	73.9	1.9	10.5	78.3	1.9	9.6	13.0	1.9	25.0	78.3	1.9	7.6
	73.9	0.0	11.0	76.1	0.0	9.9	6.5	0.0	26.0	78.3	0.0	7.8
5	78.3	1.9	15.5	82.6	5.7	14.0	56.5	1.9	27.0	80.4	3.8	12.2
	78.3	1.9	16.0	82.6	5.7	14.5	52.2	1.9	28.5	80.4	3.8	12.4
	78.3	1.9	16.5	82.6	5.7	15.0	50.0	1.9	30.0	80.4	3.8	12.6
	78.3	1.9	17.0	80.4	3.8	15.5	43.5	1.9	31.5	80.4	3.8	12.8
	76.1	1.9	17.5	80.4	1.9	16.0	43.5	1.9	33.0	80.4	1.9	13.0
	76.1	0.0	18.0	78.3	0.0	16.5	41.3	0.0	34.5	80.4	0.0	13.2

Table F.2: Performance comparison of the DTS algorithm with and without the MAT filter in capturing *true* object motion vectors for the *Foreman* (frames #8 and #9) video sequence.

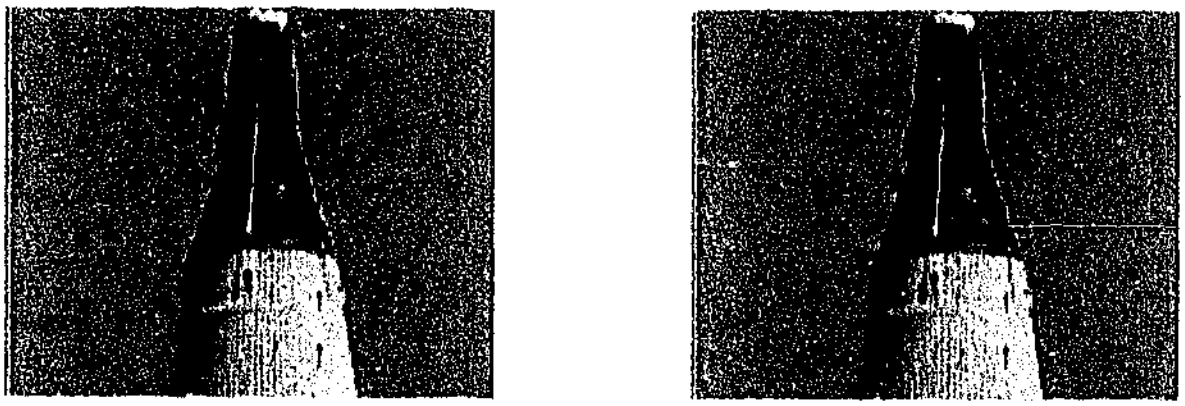


(a) (b)

Fig. F.3: (a) Current Frame #1 in which X indicates the moving macroblocks; and (b) Reference frame #2 of the *Rocket* sequence.



(a) FS (b) TSS



(c) NTSS (d) DTS

Fig. F.4: Motion vectors obtained from all four search algorithms applied to the frame pair shown in Fig. F.3 for the *Rocket* sequence.

No. of iterations	FS			NTSS			TSS			DTS		
	True MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f	*True* MV %	*False* MV %	T_f
0	69.2	34.1	7.0	46.2	27.3	7.0	53.8	29.5	7.0	100.0	61.4	1.0
	7.7	9.1	7.5	7.7	4.5	7.5	7.7	4.5	7.5	92.3	36.4	1.5
	3.8	4.5	8.0	3.8	2.3	8.0	3.8	2.3	8.0	92.3	36.4	2.0
	3.8	4.5	8.5	3.8	2.3	8.5	3.8	2.3	8.5	92.3	11.4	2.5
	3.8	4.5	9.0	3.8	2.3	9.0	3.8	2.3	9.0	84.6	9.1	3.0
	0.0	2.3	9.5	0.0	2.3	9.5	0.0	2.3	9.5	80.8	0.0	3.5
1	69.2	13.6	10.5	38.5	6.8	10.5	42.3	9.1	6.5	92.3	18.2	3.0
	65.4	6.8	11.0	30.8	2.3	11.0	42.3	2.3	7.0	92.3	4.5	3.5
	57.7	6.8	11.5	30.8	2.3	11.5	30.8	2.3	7.5	92.3	2.3	4.0
	53.8	2.3	12.0	19.2	2.3	12.0	23.1	2.3	8.0	92.3	2.3	4.5
	46.2	2.3	12.5	15.4	2.3	12.5	15.4	2.3	8.5	92.3	2.3	5.0
	30.8	0.0	13.0	3.8	0.0	13.0	7.7	0.0	9.0	88.5	0.0	5.5
2	65.4	6.8	18.5	50.0	9.1	16.0	53.8	11.4	13.0	73.1	2.3	8.5
	65.4	6.8	19.0	50.0	6.8	16.5	53.8	4.5	13.5	73.1	2.3	9.0
	61.5	4.5	19.5	50.0	6.8	17.0	50.0	4.5	14.0	73.1	2.3	9.5
	57.7	4.5	20.0	46.2	2.3	17.5	42.3	2.3	14.5	73.1	2.3	10.0
	57.7	2.3	20.5	34.6	2.3	18.0	42.3	2.3	15.0	73.1	2.3	10.5
	57.7	0.0	21.0	34.6	0.0	18.5	34.6	0.0	15.5	73.1	0.0	11.0
3	57.7	6.8	32.0	50.0	2.3	29.0	50.0	2.3	32.0	73.1	2.3	19.5
	57.7	6.8	33.0	50.0	2.3	30.0	50.0	2.3	33.0	73.1	2.3	20.0
	57.7	4.5	34.0	50.0	2.3	31.0	50.0	2.3	34.0	73.1	2.3	20.5
	57.7	2.3	35.0	46.2	2.3	32.0	38.5	2.3	35.0	73.1	2.3	21.0
	57.7	2.3	36.0	42.3	2.3	33.0	38.5	2.3	36.0	73.1	2.3	21.5
	57.7	0.0	37.0	34.6	0.0	34.0	34.6	0.0	37.0	73.1	0.0	22.0

Table F.3: Performance comparison of the DTS algorithm with and/without the MAT filter in capturing *true* object motion vectors for the *Rocket* (frames #1 and #2) video sequence.