A New Fast Motion Search Algorithm for Block Based Video Encoders

by

Simon Peter William Booth

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2003

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research

Signature

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

Borrower’s Page

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Acknowledgements

I would like to extend my gratitude to all those who helped and supported me while preparing this thesis. Firstly, I would like to thank my supervisor Prof. D. Clausi for his support and guidance – both technical and otherwise over the last three years. I would like to thank my co-supervisor Prof. M. E. Jernigan for his guidance and for first introducing me to the area of Image Processing.

I would like to thank Dr. M. Gallant, Dr. G. Coté and especially Dr. L. Winger at VideoLocus Inc. for providing me with the resources for this research, and for sharing with me their considerable expertise in the field of video compression.

Finally, I would like to thank my fiancée, Nazreen Perris, whose love and support have been invaluable.

Abstract

Block-based motion compensated transform coding is a ubiquitous paradigm for video compression systems. In such video encoders, block-based motion estimation is a significant tool for the reduction of redundant information. In a typical video encoder, motion estimation is the most computationally expensive process, accounting for 60-80% of computational resources. Many fast block-matching techniques have been proposed, however, few are well suited to implementation in a VLSI system. In this thesis, a novel motion estimation algorithm is proposed that exhibits some properties that facilitate efficient VLSI implementation, while providing an average compression performance increase of up to 15%, over the traditional full search block-matching algorithm.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

A vast array of applications involves the transmission or storage of video sequences. Recent advances in digital technologies have created heightened interest in digital formats of video. The advent of the Internet has enabled digital video applications such as video-conferencing, and, with the ubiquity of personal computers, has created a demand for consumer multimedia applications. Television broadcasting is also moving towards the use of digital broadcasting technologies with the introduction of satellite, digital subscriber layer (DSL) and other digital service providers. Emerging technologies, such as digital cinema, continue to create further demand for digital video.

Raw digital video is a sequence of digital frames. In a typical television application, each frame would be comprised of 720 rows and 480 columns of pixels (picture elements), and the video sequence would have 30 frames per second. With 24 bits of information per pixel, transmission of such video would result in a data rate of over 248 Mbps. Emerging high-definition television standards would require over 1.5 Gbps. With the limited bandwidth of today’s networks and storage media, there is a clear need for effective video compression technologies. For example, a DSL network connection has a typical data bandwidth of 6 mbps, far below the required data rate for uncompressed video.

1.1 Video Compression

Digital video signals contain redundant and irrelevant information [1]. Redundancy in video can be of many forms: spatial redundancy, due to the correlation of neighbouring pixels; temporal redundancy, due to the correlation between video frames [2]; and coding redundancy, due to other statistical redundancies that occur throughout the encoding process. Digital video signals can also include data that are imperceptible to the human visual system - this data can be considered irrelevant in most applications. Because of these properties, it is possible to compress video signals to manageable data rates without introducing significant quality degradation.

The video compression application can be viewed as a traditional digital signal transmission system, the basic components of which are shown in Figure 1-1. For video compression applications, the transmitter would include the functionality of a video encoder; the channel could take the form of a digital broadcast or storage medium; the receiver would include the functionality of a video decoder corresponding to the encoder. The efficiency of a video compression system is defined by two measurements: the data rate through the channel and the error or distortion between the source video signal and the decoded signal.

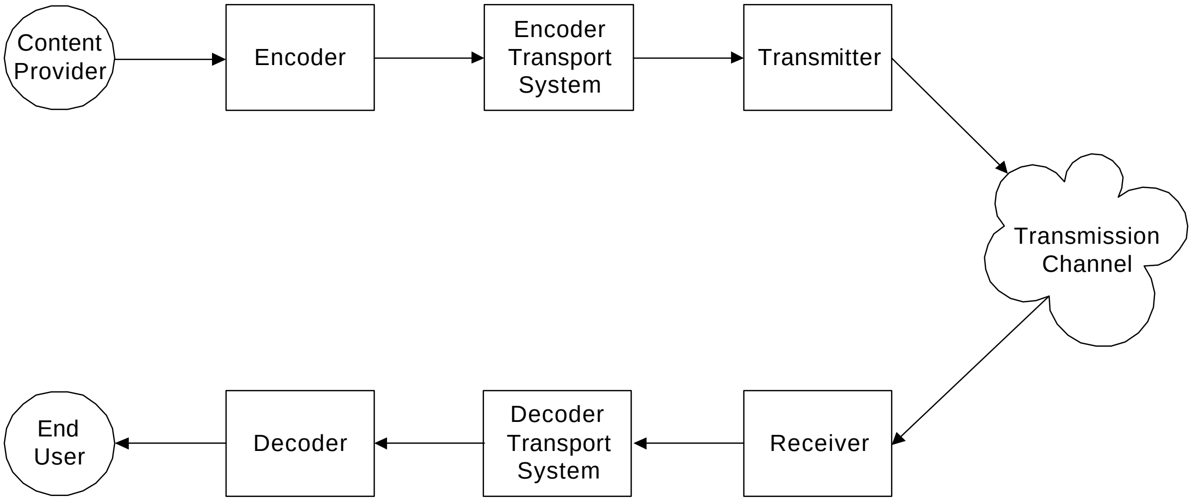


Figure 1-1 - Digital signal transmission system

There are two fundamental approaches to video compression systems. Lossless, or reversible compression [3], requires that there be zero error between the source video and the decoded video signals. The efficiency of the system is then only defined by the data rate. Compression systems of this kind are typically able to achieve compression ratios of between 2 and 5 times [4,5]. This form is usually found in specialised application areas such as remote sensing or medical imaging [6]. The majority of video compression applications can tolerate some degradation between source and decoded video signals. Such lossy compression systems are able to achieve much higher compression ratios depending on the tolerance for signal degradation.

1.2 Exploiting Pixel Redundancy in Video Coding

An observation fundamental to video compression is that successive frames of a video sequence are often very similar. As well, pixel values within a region of the same frame are also quite similar. The high correlation of pixel values between frames and within the same frame implies significant redundant information for encoding. Modern video compression systems have several mechanisms for exploiting this redundant information to achieve data rate reductions. Among them, one of the most significant such mechanisms is called motion compensation [3].

The purpose of motion compensation is to decorrelate pixel values. This is done by creating a predicted value for each pixel in a frame with respect to pixels in past, current, or future frames. The aggregation of the predicted pixel values for a video frame is called the predicted frame. The predicted frame is subtracted from the original frame to be encoded, creating a prediction residual frame. With a perfect motion compensation system, the pixels in this residual frame would be completely uncorrelated.

Ultimately, the goal of motion compensation is to reduce the required data transmission rate for the compressed video sequence. Data rate reduction is possible through motion compensation in video coding systems through the syntax defined for the compressed stream, which is known to the encoder and the decoder. The syntax defines a set of parameters that is used to generate predicted pixel values for the current frame based on previously encoded pixel values. In general, it is possible to find appropriate prediction parameters, such that the data rate for encoding the prediction parameters and the residual pixel information is significantly less than the data rate that would be required to encode the original pixel values.

A very simple form of motion compensation would be to use a previous frame as the predicted frame – the predicted value for each pixel is the co-located pixel value in a previous frame. In this case, the prediction parameters would simply signal the previous frame to be used for prediction, and the pixel information to be coded would be the difference between the two frames. As is true for most video sequences, when successive frames are very similar, this approach would be effective at decorrelating the pixel value, and reducing the encoded data rate.

1.3 Block Based Motion Compensation and Estimation

Various video coding systems allow different prediction models. These can vary widely in complexity from the simple frame difference model, discussed in the previous section, to far more complex models where the prediction model is better able to accurately predict the pixel values. In the latter case, the proportion of the data rate allocated to the prediction parameters is greater, but the portion allocated to the pixel data is far less – resulting in overall compression gains.

The process by which the best prediction parameters are identified for each video frame is called motion estimation. As the complexity of the prediction model increases, so too does the required complexity of the motion estimation process, and therefore the computational resources required for implementation. There is therefore a trade-off in the determination of the complexity of the prediction model used for motion compensation. There is an obvious need to achieve significant compression of video sequences. However, this must be balanced against the implied complexity and feasibility of implementation.

In practice, the most common prediction model used for video coding is block translation [3]. In this model, video frames are divided into rectangular blocks of pixels. For each block of pixels, a similar block of pixels is identified in a previously encoded video frame. The pixels in this block of the previous frame are then used as the predicted values for the pixels in the current frame. The difference in location of the pixel blocks in the previous and current frame is called motion. This motion is coded as the motion vector that defines the location of the prediction block in the previous frame, with respect to the block in the current frame. In most cases, for each block in a video frame, it is possible to find a block in a previous frame where the data rate to encode the motion information and the pixel residual information for the block is less than the data rate that would be required to encode the original pixel information.

1.4 Contribution

In modern incarnations of video compression systems that use block based motion compensation, the motion estimation task is often the most computationally expensive component of the encoding process. Recent complexity advances, such as variable block size compensation and sub-pixel motion vector resolution, have further highlighted the need for efficient block based motion estimation.

In recent years, much research has been done to improve the efficiency of motion estimation algorithms [7-16]. Many of the approaches use complex procedural strategies that are far more appropriate to software implementations than to efficient hardware very large scale integration (VLSI) implementation. This thesis proposes a novel method for efficient motion estimation within the context of a block translation prediction model. This method is aimed at addressing some of the issues associated with a VLSI implementation of a video encoder.

The next chapter will describe the theory and structure that is common to most modern video compression systems. Particular reference will be made to recent standardisation efforts, specifically International Telecommunication Union Telecommunication Standardization Sector (ITU-T) recommendation H.264. The following chapters will discuss the motion estimation problem, and a variety of motion estimation techniques. Several fast motion estimation algorithms proposed in the literature will be described and their suitability for hardware implementation will be considered. Chapter four will then describe the proposed motion estimation algorithm, and its performance will be evaluated. Chapter five concludes the thesis and provides recommendations for future enhancements to the proposed algorithm.

Chapter 2

Block-Based Motion Compensated Transform Coding

2.1 Video Compression Standardisation

Wide interest in video compression throughout various industries and academia has led to the standardisation of video compression. Several successful video compression standards have emerged over recent years – largely falling within the Motion Pictures Experts Groups MPEG, or ITU-T H.26x groups of standards[17-20]. The most common of these standards is MPEG-2[18], which has become ubiquitous within the television broadcasting and DVD industries. The most recent emerging standard is the ITU-T recommendation H.264 [21]. While H.264 offers much advancement over MPEG-2, both video standards are examples of Block Based Motion Compensated Transform Coding. This chapter will provide an overview of the basic structure and each of the compression tools of the H.264 compression standard. While many of the details are specific to H.264, much of the general theoretical discussion will be equally applicable to any of the aforementioned standards.

2.2 Encoder-Decoder System Overview

Standard video compression techniques[17-21] are often referred to as hybrid techniques because they make use of several compression tools simultaneously. Each such tool can be used independently, or in conjunction with the other methods. The methods common to all standards video encoding systems are colour sub-sampling, motion compensation, frequency transform, quantisation, and lossless or entropy encoding. Figure 2-1 shows the structure and interconnections of the hybrid coding scheme used by H.264 and other video compression standards.

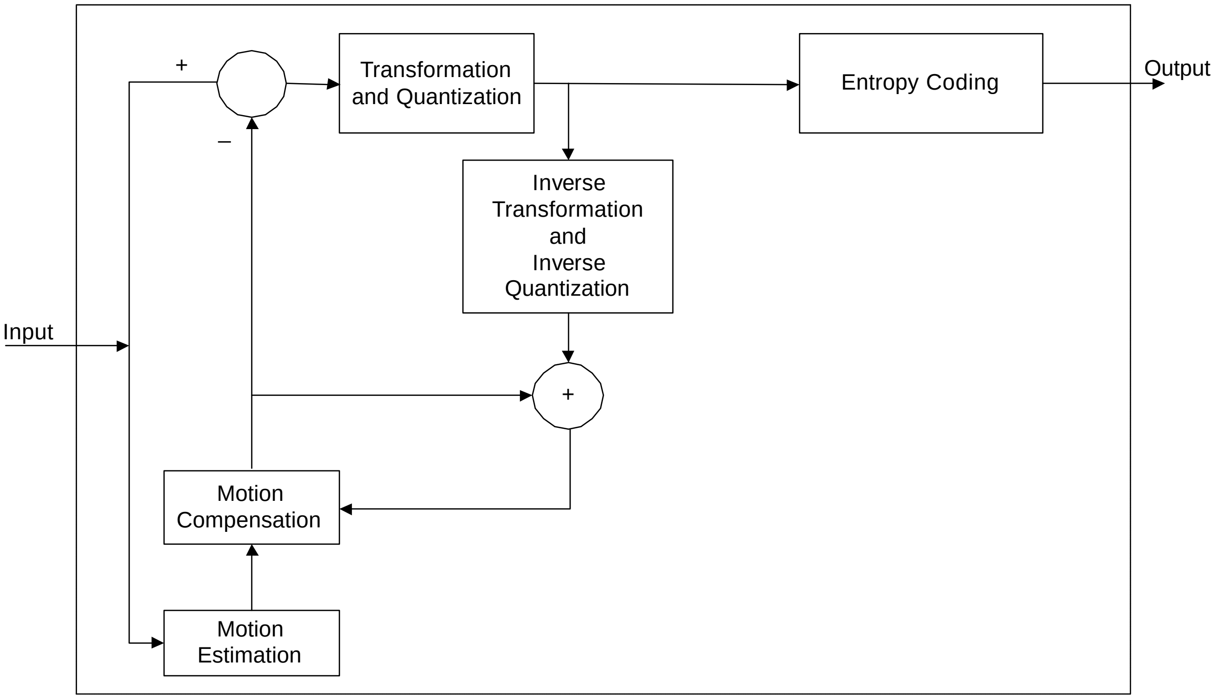

+
–
Transformation and Quantization
Entropy Coding
Output
Inverse Transformation and Inverse Quantization
Input
+
Motion Compensation
Motion Estimation

Figure 2-1 - Block diagram of the hybrid video encoding scheme

2.3 Colour Image Representation

A digital video sequence can be viewed as a series of 2-D colour images or frames. Frames are represented by a luminance signal (luma Y) and two chrominance signals (chroma Cr and Cb). The human psycho-visual perception system exhibits much greater sensitivity to high frequency variations in the brightness or luminance of an image that in the chrominance components [22]. The MPEG video compression standards[18,20,21] exploit this aspect of the psycho-visual system by reducing the resolution of the chrominance components of a video signal. Specifically, in H.264, the two chrominance signals are represented with half the vertical and horizontal resolution of the luminance signal (4:2:0 sub-sampling). This generates a 2:1 compression ratio with minimal visual quality degradation [23].

2.4 Motion Compensation

The most important feature of video compression systems is the ability to exploit the spatial and temporal redundancies inherent in all video sequences. Largely, this is accomplished through predictive coding. In this scheme, a predicted value is estimated for each pixel of a video frame, and the difference between the predicted and actual values of each pixel is the only pixel information required to be encoded (Figure 2-2):

$$P[x,y] = \hat{P}[x,y] + R[x,y], \tag{1}$$

where P is the original pixel value, $\hat{P}$ is the predicted value of the pixel, and R is the prediction error or residual, at position *(x,y)*. If the predicted pixel value is derived from other pixels from the same video frame, the effect of this technique is to reduce the spatial redundancy of the video. If the prediction is derived from other video frames of the sequence, the temporal redundancy is reduced.

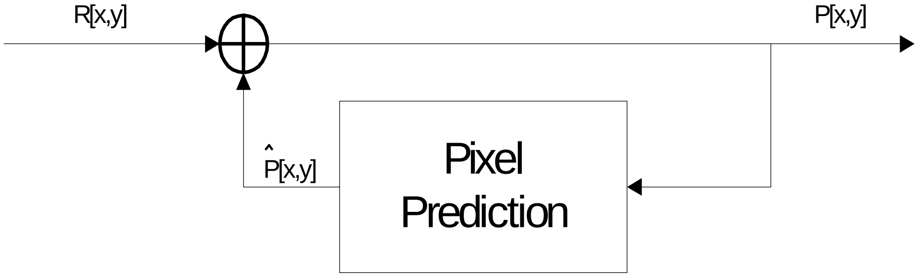


Figure 2-2 - Predictive Coding

The MPEG/ITU video compression standards referred to in section 2.1 allow many prediction models to be used to estimate pixel values. As a result, the encoded stream must also contain a set of prediction parameters that define the prediction model, and therefore, estimate the pixel using previously decoded pixels. When considering the data rate cost of encoding, it is important to consider both the amount of data required for the pixel information and for the prediction parameters. The MPEG/ITU video compression standards[17-21] divide each frame into blocks of pixels and define the estimates of the entire block through one set of prediction parameters. These blocks are called macroblocks and for H.264 [21] they have a size corresponding to 16x16 luma pixels and two fields of 8x8 chroma pixels. For each macroblock, there are two types of prediction models used by the video compression standards: Intraframe estimation, and Interframe estimation. These are discussed in the next sections.

2.4.1 Intraframe prediction

In each frame of a video sequence there is a high spatial correlation. This implies that it is possible to calculate reasonable estimates of the values of a block of pixels based on a set of neighbouring pixels. H.264 defines intra prediction modes that generate a 16x16 predictive block of pixels based on the neighbouring pixels above and to the left of the macroblock. Also, H.264 allows for intra prediction on a 4x4 pixel block level using a similar set neighbouring pixels. There are four directional intra prediction modes for 16x16 intra prediction and nine directional 4x4 intra prediction modes. These are described in full detail in [21].

The estimation strategies used for intraframe prediction are typically an exhaustive evaluation of the available prediction modes for each block [15]. The resource requirements for this are far less than for interframe estimation. As a result, this thesis focuses on interframe estimation.

2.4.2 Interframe prediction

The most significant video compression tool is interframe prediction. Also called motion compensation, this form of prediction is used to remove temporal redundancies in video data. Motion compensation attempts to model the motion of objects within a video sequence over time. This corresponds to a mapping of pixels in one frame to pixels in a previously coded frame.

Referring to Eq. (1), the predicted value of a pixel in the current frame $\hat{P}_k[x, y]$ is estimated from the previously coded frame P_{k-1}, using prediction model *PRED()*, and a set of motion parameters:

$$\hat{P}_k[x, y] = PRED(\{motion\_parameters\}, P_{k-1}) \qquad (2)$$

The aggregation of this mapping will be a predictive frame, $\hat{P}_k$, as indicated in (1). The residual frame, *R*$_k$, is produced by the subtraction of the predictive frame from the original frame:

$$R_k = P_k - \hat{P}_k \qquad (3)$$

The residual frame will contain much less information than the original frame therefore reducing the data rate required for the pixel information. When the data rate required to encode the motion parameters that define the motion-based prediction and the residual frame is less than the data rate that would required to encode the original frame directly, motion compensation provides compression gain.

Motion in a video sequence can be of several different types – an object can undergo any combination of rotation, zoom and translational motion in any direction. Many research papers in the literature have proposed motion compensation models to precisely describe the motion of groups of pixels. In [24] and [25], for example, arbitrarily shaped object models are defined to describe the shape of moving objects in video. In [26], a method for describing macroblock block motion as a generalised spatial transformation is presented. Many of these motion models, and others that exist like them, pose too great a challenge in estimation of the best parameters to be practical for most video encoding applications. As a result, simpler motion models are more appropriate for most video compression applications.

The motion compensation used in most MPEG/ITU video compression standards is limited to translational motion of blocks[17-21]. For each block of pixels, a similar block of pixels is identified in a previously encoded frame, and is used for prediction. The difference between the position of the predictive block of pixels in the previous frame and the position of the original pixel block can be represented by a motion vector with horizontal and vertical components: X_{MV} and Y_{MV}. For translational motion model, the prediction for each pixel within the block is formed with the following equation:

$$\hat{P}_k[X_0, Y_0] = P_{k-1}[X_0 + X_{MV}, Y_0 + Y_{MV}] \quad (4)$$

where (X_0,Y_0) is the position of the original pixel. The residual between the original block and the predictive block is encoded, as well as the motion vector that defines the difference between the positions of blocks in their respective respective frames. Figure 2-3 illustrates this motion compensation.

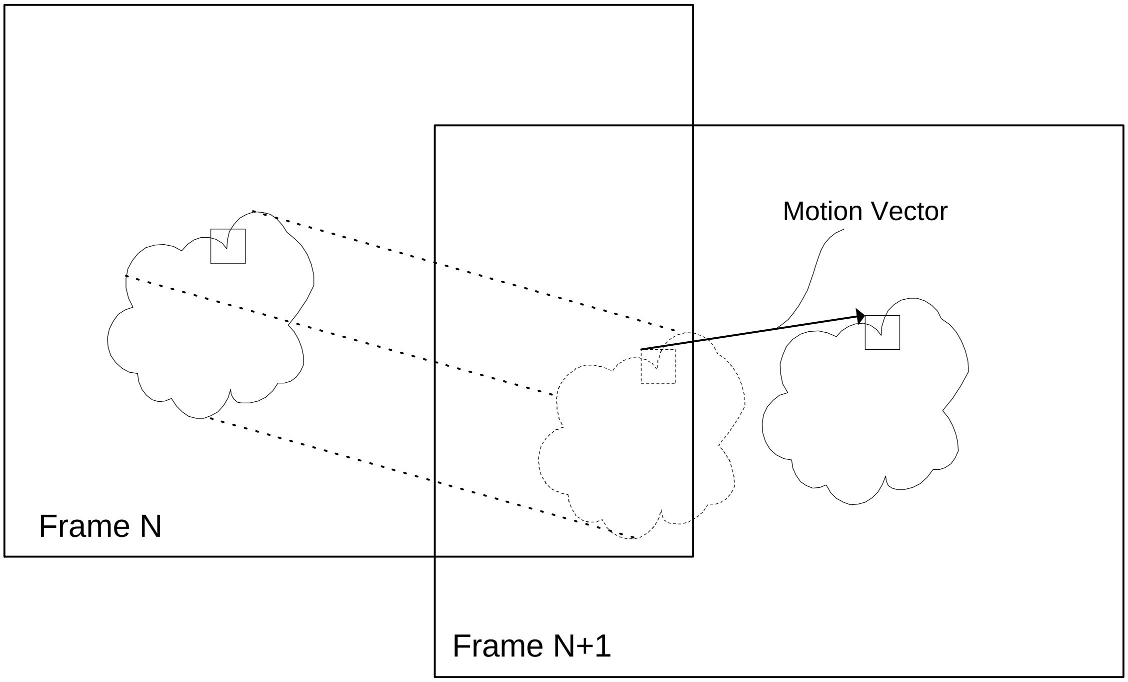


Figure 2-3 - Block Motion Compensation

Motion vectors used for motion compensation show a high degree of correlation, with motion vectors of neighbouring blocks [23]. For this reason, it is reasonable to use predictive coding for the motion vectors as well as the pixel information. In MPEG-2, a motion vector was predicted from the motion vector of the macroblock immediately to the left – the differential motion vector between two successive motion vectors was encoded. In H.264, additional neighbouring blocks are used for prediction. Referring to Figure 2-4, the predicted motion vector for block of pixels X is calculated from the vectors that have been applied to blocks A,B,C and D.

Early MPEG compression standards [18,20] used a fixed block-size for motion compensation. It has been shown that compression gains can be achieved with adaptive block size motion compensation [27]. Allowing variable block sizes allows better prediction in areas where image detail does not align with macroblock boundaries. H.264 supports seven block sizes for motion compensation: 4x4, 4x8, 8x4, 8x8, 8x16, 16x8 and 16x16. The resulting possible macroblock configurations are shown in Figure 2-5.

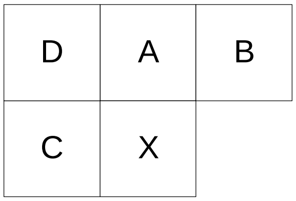


Figure 2-4 - Neighbouring blocks (A-D) used for motion vector prediction of X

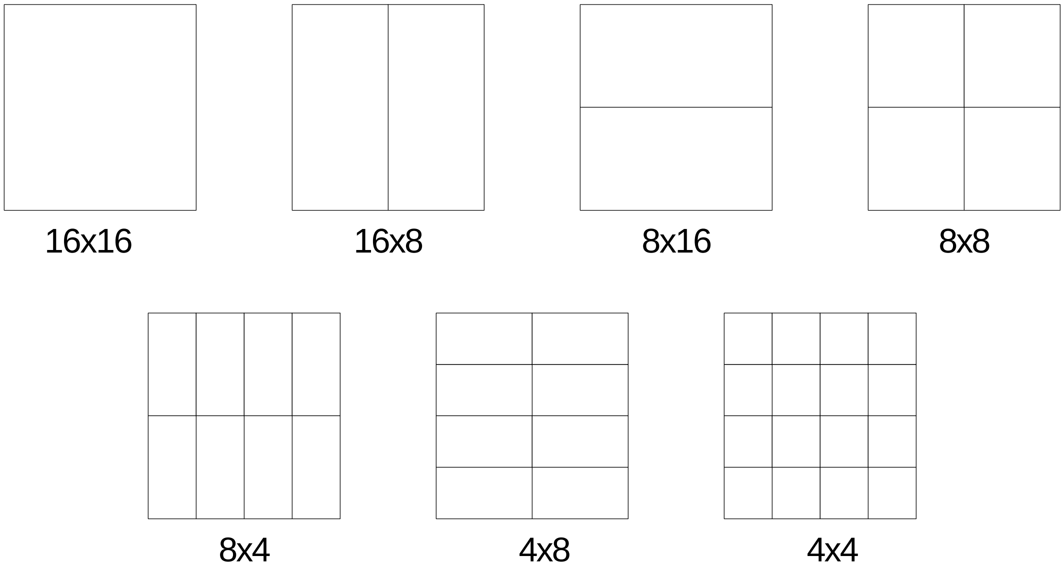


Figure 2-5 – Macroblock Partitioning for Motion Compensation in H.264

2.4.3 Video Sequence Structure

In the MPEG/ITU video compression standards[17-21], video frames are typically encoded as a series of groups of pictures (or GOPs). Each frame within a GOP is encoded as one of three types, according to the type of prediction used in encoding. Figure 2-6 shows a typical arrangement of video frames within a GOP. The frames are shown in the order that they would be displayed. The order that the frames are encoded in is determined by the inter estimation dependencies.

Intra Frames

The first frame of a GOP is an Intraframe (or I-frame). I-frames are encoded using only intra methods pixel prediction. Since all predictive pixels are from the same frame, they are coded independently of all other frames.

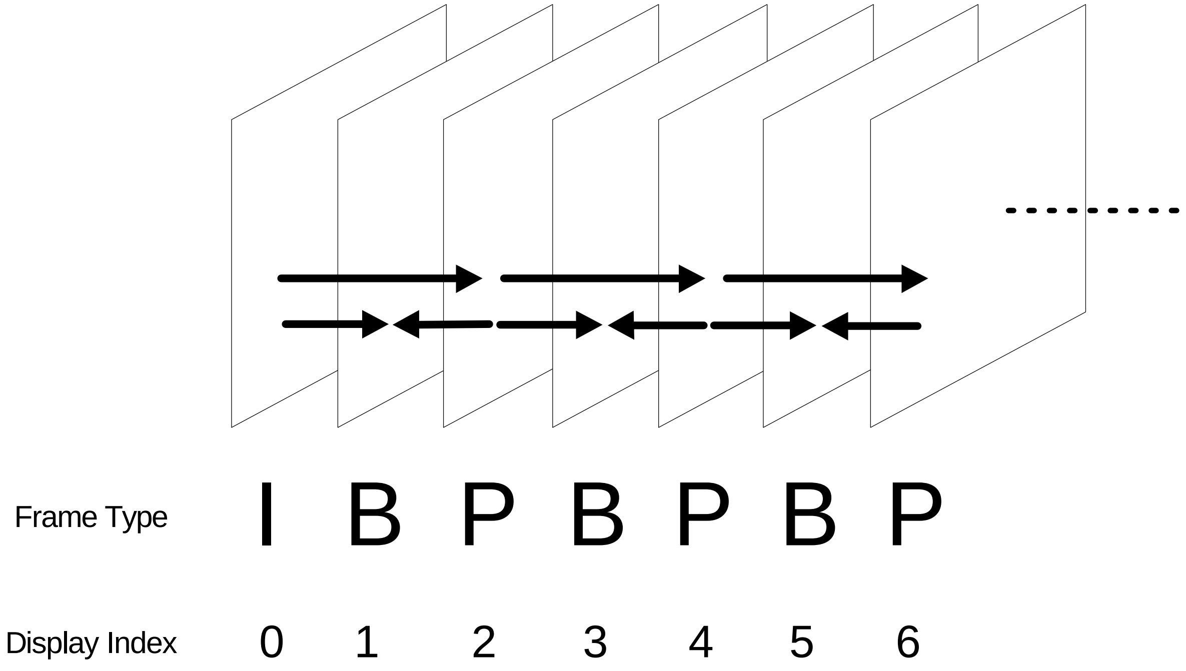


Figure 2-6 - Typical frame ordering of a group of pictures

Predictive Frames

Predictive frames (or P-frames) use interframe prediction methods as well as intraframe methods. For P-frame motion compensation, only forward prediction is supported – frames used for prediction must temporally precede the encoded frame. In H.264, multiple reference frame prediction is permitted, i.e. P-frames pixel blocks may be predicted from any preceding I-frame or P-frame. This feature is useful for encoding transitionally covered background and periodic non-translational motion [2].

Bi-Predictive Frames

Bi-predictive frames (or B-frames) use an expanded set of inter-prediction methods compared to P-frames. Specifically, B-frames support forward and backward prediction for motion compensation – reference frames may occur before or after the encoded frame in the display order of the video sequence. In addition, H.264 B-frames support bi-predictive block compensation[21]. In this method, the predictive blocks may be calculated as a combination of blocks that occur in different frames. In early video compression standards, B-frames were predicted from previously encoded I-frames or P-frames as shown in Figure 2-6. The H.264 standard allows motion compensating prediction from any previously encoded frame in the video sequence, including prior B-frames.

2.5 Transform Coding

Block based transform coding is ubiquitous in the area of image and video coding. The purpose of transform coding is to express pixel information in a way in which it can be more efficiently encoded. This is done by selecting a transform that will decorrelate the elements of the block and compress as much of the energy of the block into as few coefficients as possible. The optimum transform for spatial decorrelation is the Karhunen-Loeve transform (KLT), however, it is unsuitable for practical encoder implementations, because the basis functions of the transform are image dependent [3]. The Discrete Cosine Transform (DCT) is the most commonly used transform for video coding because its performance is close to the KLT, and there are efficient hardware and software implementations. In the H.264 standard, a 4x4 integer “pseudo-DCT” transform [2] is used for transform coding of the prediction residuals.

The main disadvantage of the block-based transform coding approach is that it can result in perceptually significant artefacts at block boundaries [28]. To correct for this phenomenon in H.264 a deblocking filter has been defined for the frame reconstruction and decoding paths.

2.6 Quantisation

To this stage, none of the compression tools discussed so far have been inherently lossy. Information loss is introduced to the encoded video through quantisation. Quantisation is the process through which data values are expressed with a lower degree of precision. In this case, the pixel block coefficient values are to be quantised. The extent of information loss is determined by the quantisation step-size – a larger step-size implies fewer unique values that a coefficient can take, and therefore greater information loss.

In H.264, the transform representation of a pixel block will contain one DC and fifteen AC coefficients. Very little of the energy of the block will be contained in the high frequency coefficients- these coefficients will normally be near zero. As a result, these coefficients can be quantised heavily with little impact on the quality of the encoded video sequences. In practice, many AC coefficients are quantised to zero, and therefore this frequency information is discarded.

In early video compression standards, the quantisation step-size was adaptive over frequency components. The purpose of this adaptivity was to exploit a reduced sensitivity of the human visual systems of higher frequency components – information loss at high frequencies is less perceptually significant than at lower frequencies. However, due to the smaller transform size and better pixel prediction, a flat quantisation matrix is used for H.264.

2.7 Entropy Coding

The final video compression component used in the hybrid encoding scheme is entropy coding [28]. The purpose of this compression tool is to exploit the statistical redundancies that exist in the sequence of syntax elements (i.e. coefficients, motion vectors, etc) to be encoded to the bitstream. An entropy encoder provides a mapping between input symbols (i.e. syntax elements) and codewords to be written to the coded bitstream. The fundamental idea is to use shorter codewords for more frequent symbols and longer codewords for less frequent symbols. The entropy decoder is able to perform the inverse mapping, and recreate the original sequence of symbols. As no information is lost through this process, it is often called lossless encoding.

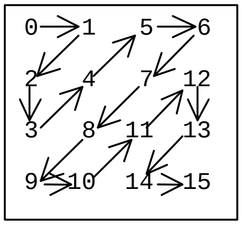


Figure 2-7 - Zig-Zag Scan Order for Coefficients

The efficiency of the entropy coding is closely related to the transform and quantisation steps. As described above, many of the quantised AC coefficients of the residual pixel data will be zero. Each video compression standard defines a scan pattern that defines the order that the coefficients are encoded into the bitstream. The scan patterns are roughly designed to increase monotonically in frequency, so there will be a series of consecutive zeroes. Such symbol sequences can be very efficiently coded by an entropy coder. Figure 2-7 shows the zig-zag coefficient scan pattern used in H.264 [21]. The goal of motion compensation is then to reduce the magnitude of the coefficients that result from the prediction residual, particularly in the high frequency components, since these are more expensive to encode

The H.264 standard supports two versions of entropy coding: Universal Variable Length Coding (UVLC) and Context-Based Adaptive Binary Arithmetic Coding (CABAC). UVLC

[29] uses a fixed codebook that is based on prior probability models for each symbol. CABAC [30] adapts the probability model for each symbol according to the context of the symbol and the frequency of occurrence in the previously encoded bitstream. CABAC entropy coding requires a more complex implementation than UVLC, but it has been shown to provide a coding efficiency gain of 9-27% [30].

Chapter 3
Motion Estimation

Video compression standards define a syntax that allows a motion model based on translational block motion to be used for pixel prediction. Effective use of the motion compensation tools provided for by the standards definitions requires that the encoding process identify appropriate prediction parameters. Specifically this requires that the encoding process identify the best, or at least good, prediction modes and motion vectors for each block in each inter-predicted frame. The process of determining the best motion parameter is called motion estimation. The accuracy of motion estimation has a significant impact on the effectiveness of motion compensation block prediction, and ultimately the compression efficiency of video encoder. In addition, as the motion estimation process is not defined by the syntax of a video standard, its effectiveness is the main distinguishing feature in assessing the quality of a standards-based video encoder.

The goal of motion estimation is to identify the motion parameters that will result in an encoded stream with the lowest possible data rate and the best visual quality. Several approaches to motion estimation are explored in the literature, including image feature-based estimation techniques (e.g. [14]), and optimisation based on mean field theory (e.g. [31]). By far the most common approach to motion estimation for video compression is block-matching. This chapter will describe first block-matching motion estimation and discuss the two features of a block-matching algorithm (BMA) – the matching cost function and the search strategy. The well-known full search strategy as well as some fast BMAs will be discussed in terms of coding performance and implementation considerations.

3.1 Block-matching

The purpose of block-matching is to match the current block of pixels with a similar block in a reference frame. Each block of pixels in a reference frame is identified by the vector that defines its position in the frame. For each candidate reference block there is a corresponding candidate motion vector ($\overrightarrow{CMV}$). Figure 3-1 shows the setup for a general block-matching algorithm. A cost function is defined that evaluates the match between the original block and the candidate block: $J\left(\overrightarrow{CMV}\right)$. The block-matching algorithm then tries to find the motion vector that gives the minimum cost value:

$$\overrightarrow{MV} = (dx, dy) = \arg\min_{(dx,dy)} J(dx, dy), \tag{5}$$

where *dx* and *dy* are the horizontal and vertical components of the motion vector.

The brute force approach to block-matching would be to perform the cost calculation for every possible block position in every possible reference frame. This approach is far too computationally expensive, and is unnecessary to achieve good compression. Block-matching algorithms instead perform the cost calculation for a subset of all possible motion candidates. The defining characteristics of a block-matching algorithm are the selection strategy for determining which positions to test, and the cost function that measures the quality of a match.

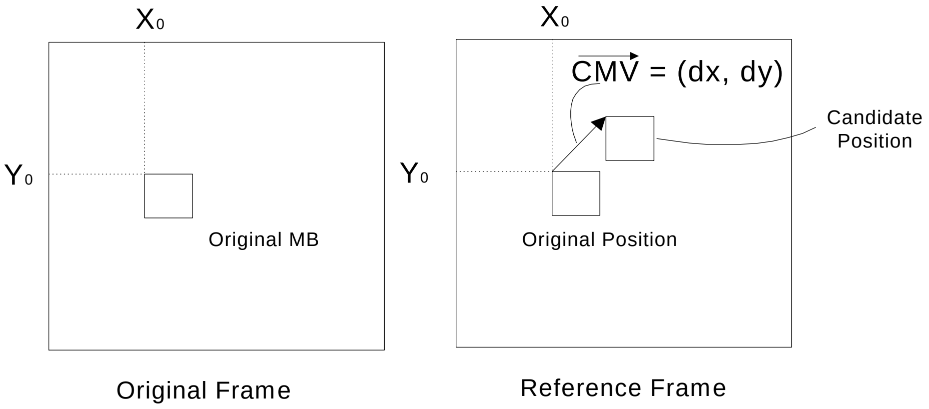


Figure 3-1 - Block-matching

3.1.1 Matching cost functions

Block-matching cost functions measure the difference between the candidate block and the original block. This cost is to be minimised so that the prediction residual has less energy and therefore the cost of encoding pixel coefficients is minimised. A good overview of block matching distortion measures is provided in [32]. By far the most commonly used distortion measure is the Sum of Absolute Differences (SAD). For an NxM block of pixels, the SAD cost function for the candidate block is defined as:

$$J_{SAD}(dx,dy)=\sum_{j}^{N}\sum_{i}^{M}\left|O(i,j)-R_{(dx,dy)}(i,j)\right| \tag{6}$$

where O is the original block, and $R_{(dx,dy)}$ is the candidate block in the reference frame corresponding to motion vector (*dx,dy*). This distortion measure is sometimes expressed as Mean Absolute Difference (MAD), which is an equivalent measure where the SAD is normalized by the number of pixels in the block.

An alternative to SAD is the Mean Squared Error (MSE):

$$J_{MSE}(dx,dy)=\frac{1}{N\cdot M}\sum_{j}^{N}\sum_{i}^{M}\left[O(i,j)-R_{(dx,dy)}(i,j)\right]^{2} \tag{7}$$

MSE, or Euclidean distance, provides better coding performance than SAD, because it is a closer measure to perceptual quality of the human visual systems. However, due to the need for one multiplication per pixel, it has a significantly higher computationally complexity. As a result, SAD is more commonly used.

Many distortion measures have been developed to reduce the computational complexity of the SAD distortion measure. One approach to reduce complexity is the truncation of pixel values to reduce the bit depth of the SAD measure [9,16,33,34]. SAD truncation does provide computational efficiencies, especially for VLSI implementation, however, it does result in non-trivial coding loss. Other approaches to complexity reduction include pixel sub-sampling of the original and reference blocks [35,36], and integral projection matching [8,37].

Block-matching cost functions have been proposed that provide compression improvements at the expense of additional computational complexity. For example, [38] presents the use of the Walsh-Hadamard transform (WHT) in the block-matching cost calculation. The WHT is a frequency transform domain similar to the integer transform used in H.264, and has been used in transform coding image compression applications [39]. The bases of the WHT use only 1/-1 values, so application of this transform requires only addition and subtraction operations. When the WHT is applied to a block-matching residual, the result approximates the frequency coefficients that would be produced by the integer DCT. The quantisation operation can be approximated by removing near-zero coefficients. Measuring the magnitude of the resultant coefficients provides an accurate indication of the data rate cost associated each candidate position. This block-matching is used by the reference encoding software for H.264 provided by Joint Video Team (JVT), and yields substantial compression gains.

The distortion measures above measure the prediction error of block-based motion compensation. A second factor to consider in block-matching cost evaluation is the amount of data required to encode the prediction parameters – i.e. motion vectors. Rate-Distortion optimal methods have been proposed to incorporate estimates of the bit-rate and distortion [40]. This approach is especially beneficial at low bit-rates and for variable block-size encoders, where the bit allocation for the prediction parameters becomes a significant part of the overall bit-rate.

3.1.2 Full Search Block-matching

It is not feasible to evaluate the cost function for every possible set of prediction parameters for any block of pixels. A subset of positions must chosen for this evaluation. The most common approach in VLSI implementations is the well known full search block-matching algorithm [41], where a rectangular window is defined in the reference frame, and block-matching is performed at every position within that window (Figure 3-2). This algorithm is used frequently because it is conceptually intuitive, and provides accurate motion estimation results. The search window is typically centred on the co-located position in the reference frame of the original block, and is defined by its dimensions. In most applications, search ranges from 8 to 64 pixels in each dimension are used.

This algorithm is used almost exclusively in VLSI applications because it provides many implemental benefits [42,43]. Specifically, the regularity of operation and memory accesses inherent in this algorithm translate into architectural efficiencies. In addition, this approach is well suited to variable block size motion estimation. Since the data flow for each block of all block sizes is the same, it is possible to compute the distortion for all block sizes in parallel with minimal additional implementation cost [44,45].

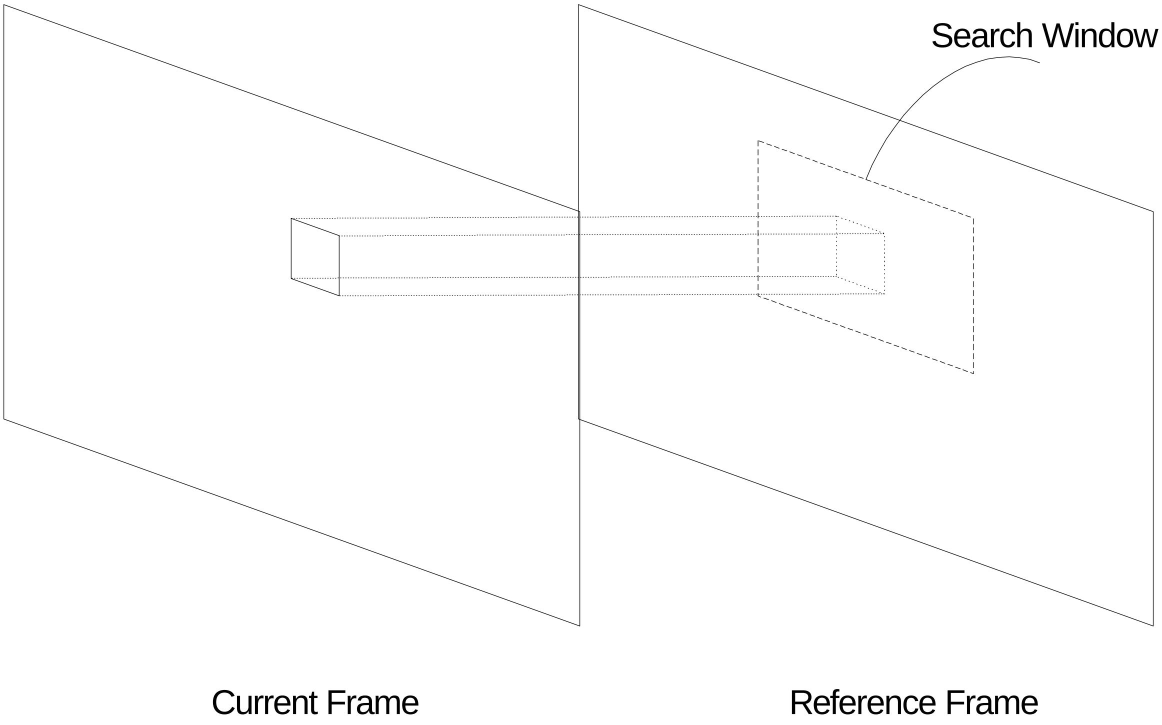


Figure 3-2 - Full search block-matching search window

Despite these implementation efficiencies for VLSI, full search block-matching is computationally very expensive. Each pixel SAD requires three arithmetic operations: subtraction, absolute value and addition. The total number of arithmetic operations per second is then:

$$numOps = 3 \cdot F_X \cdot F_Y \cdot W_X \cdot W_Y \cdot f \qquad (8)$$

where F_X and F_Y are the dimensions of each video frame, W_X and W_Y are the dimensions of the full search window, and *f* is the frame rate. For a full D1 720x480 video sequence at 30 fps, using a search window with 32x32 search positions, the total number of arithmetic operations per second is:

$$numOps = 3.19 \times 10^{10}$$

In typical video encoders, the motion estimation is responsible for 60-80% of the computational load. When considering the increasingly popular high-definition standards (HDTV), this proportion is likely to be more. For this reason much work has been done towards developing fast block-matching motion estimation techniques.

3.2 Fast Block-Matching Algorithms

Fast block-matching algorithms follow one of two distinct approaches. The first type are guaranteed to produce the same result as the exhaustive full search algorithms but are designed to find the optimal position with fewer calculations. The methods proposed in [13,15] follow such an approach. In both cases, a “best-so-far” threshold is developed in each case and the distortion calculation for each search position is stopped once the threshold has been reached. This type of algorithm could be useful for statistical power savings within a VLSI implementation, but it does not help the “worst-case” requirements for implementation.

The second approach is to reduce the number of search positions in such a way that, on average, the compression performance is not severely impacted. These techniques require some level of adaptive control over the search path. Methods have been proposed that alter the parameters of the search window on a block adaptive basis, for example, the method in [7] proposes that the size of the search window be altered for each block. The JVT H.264 reference encoder motion estimation [46]provides block adaptive control over the placement of the search window in the reference window to exploit the spatial correlation of motion vectors. In this algorithm, the search window is centred on the position corresponding to the predicted motion vector for the block (recall from section 2.4.2. that the motion vector predictor is calculated from neighbouring blocks). Due to the high spatial correlation of motion vectors, using this approach, a smaller search window is needed to achieve equivalent encoding performance to the non-adaptive full search algorithm.

For a VLSI implementation, this algorithm has some disadvantages. The efficiencies of implementation described in [42] are dependent on the consistent overlapping of the search window of two successive macroblocks. This was useful for management of the search window buffer, reducing the memory bandwidth required for loading reference pixel data. For block-adaptive search window the search window buffer must be able to be reloaded for each block – significantly increasing the memory bandwidth requirement of the encoder. These properties will be discussed further in section 4.2.

Many fast motion search strategies have been proposed that reduce the number of search positions. These motion estimation methods require more low-level adaptive search control, adapting the search path at each search position. Such block-matching algorithms include the Logarithmic Search [10], the Three Step Search [47], the Four Step Search [48], the Diamond Search [49], as well as others [11,12,50]. These block-matching algorithms follow the same approach to block-matching that is characterised by the following steps:

1. Evaluate the block-matching cost function at a few positions (typically four) surrounding the centre of the search window.
2. Compare the resultant costs and re-centre the search in the direction of the best position (of the few).
3. Steps 1 and 2 are repeated either a fixed number of iterations, or until the algorithm converges on a local minimum.

These algorithms assume a unimodal cost surface, and therefore that the local minimum is actually the global minimum. Since this unimodal assumption is sometimes not valid, these algorithms are susceptible to local minima, and as a result, do not achieve the same rate-distortion performance as the full search. These algorithms drastically reduce the number of search position over the full search strategy. For software implementations, this results in a substantial reduction in the computational load and so the implemental benefit is worth the loss in compression efficiency for many applications. Kuhn [32] compares the computational complexity and coding efficiency of the full search algorithms with the three-step search. The three–step search resulted in an average bit-rate increase of 3.9%, while reducing overall computational complexity by approximately 12 times. However, in sequences with complex motion, the three-step search showed a coding efficiency degradation of up to 26%.

The direct relationship between the number of search positions and the cost of implementation that exists in a software implementation is not present in a VLSI implementation. Due to the dependencies in execution from one stage of the motion search to the next, and between macroblocks [12], these algorithms do not lend themselves to parallel architectures. This problem is exacerbated with variable block-size motion compensation. Since the search paths of each block of all sizes do not coincide, there is no opportunity for parallel accumulation of the costs of different sized blocks. While efficient architectures have been proposed for fast motion search algorithms, such as [41,51], none address variable block-size estimation.

3.3 Summary

The most computationally intensive element of a video encoder is the motion estimation module, requiring 60%-80% of the computational resources of typical implementations. The most common motion estimation algorithm is full search block-matching. This algorithm is exhaustive and is guaranteed to be optimal within a rectangular search window, but has a high computational load. Several fast block-matching strategies have been proposed, which are based on adaptive search paths. These methods do reduce the required computational load, particularly for software implementations. However, due to its regular structure, simple control overhead, and improved compression efficiency, full search block-matching is widely used for VLSI encoder implementations.

In the next chapter, a block-matching method is proposed that offers all of the implementation benefits of the full search algorithm, while yielding comparable coding performance with far fewer search positions.

Chapter 4
Constant Search Offset Motion Estimation

In the previous chapter, the motion estimation problem was explained. The basic full search block-matching algorithm was discussed, as well as several fast block-matching algorithms. It was shown that while the fast block-matching algorithms do require fewer search positions to achieve reasonable coding performance, they are generally ill suited to VLSI implementation. Moreover, the full search block-matching algorithm is computationally intensive, but it does lend itself well to efficient VLSI implementation. In the next section, the full search block-matching algorithm will be formulated in detail. In section 4.2, the suitability of the full search block-matching algorithm will be explained in relation to three significant implemental considerations to the design of VLSI systems. For comparison, the suitability of the block-adaptive full search algorithm will be discussed, and shown to be far more expensive than the traditional full search algorithm. In section 4.3, the motion characteristics of video will be examined. Sections 4.4 through 4.6 will present a proposed motion estimation algorithm that provides many of the implemental advantages of the traditional full search, while enabling a significant reduction in the implemental expense required for good coding performance. Finally, experimental results will be presented in section 4.7 and 4.8.

4.1 Full Search Block-Matching Motion Estimation

As described in section 3.1.2, the full search block-matching algorithm entails the matching of the original block of pixels to blocks of pixels at every possible location with a rectangular area within a reference frame. The rectangular area of the reference frame can be defined by the location of the top-left and bottom-right pixel locations:

$$(X_{TL}, Y_{TL}) = (X_C, Y_C) - (S_X, S_Y) \tag{9}$$

$$(X_{BR}, Y_{BR}) = (X_C, Y_C) + (S_X, S_Y) + (N_X - 1, N_Y - 1) \tag{10}$$

where *(X_{TL}, Y_{TL})* and *(X_{BR}, Y_{BR})* are the positions of the top-left and bottom-right pixels of the reference area, S_X and S_Y are the search ranges in the horizontal and vertical direction respectively, N_X and N_Y are the horizontal and vertical dimensions of the original block of pixels and *(X_C, Y_C)* is the centre position of the motion search.

The number of pixels in the rectangular reference area is:

$$Area(S_X, S_Y, N_X, N_Y) = (2 \cdot S_X + N_X) \cdot (2 \cdot S_Y + N_Y) \tag{11}$$

For the traditional full search, the centre position is co-located to the position of the original block of pixels in the original frame. That is,

$$(X_C, Y_C) = (X_O, Y_O)$$

where *(X_O, Y_O)* is the position of the top-left pixel of the original block of pixels.

For variable block-size motion estimation, the traditional full search algorithm has the property that the search area for each sub-block of a macroblock is contained within the search area of the 16x16 block. In the block-adaptive full search algorithm, the search area of each sub-block of the original macroblock is centred on a different position *(X_C, Y_C)*. Therefore each sub-block requires an independent search area.

4.2 VLSI Implemental Considerations

The three most significant properties to compare VLSI architectures or algorithms are time, power consumption and required area [32,43]. In this case, time refers to the number of clock cycles required to perform the motion estimation for each macroblock. The time required for motion estimation of each macroblock is affected by the scope of the motion search (i.e. the number of search positions) and amount of parallelisation that is possible in the VLSI design due to the regularity of the execution flow [32]. Power consumption is difficult to estimate prior to design of the VLSI architecture, but it has been shown that I/O bandwidth is an important criterion to the power consumption of a VLSI implementation.

The manufacturing cost of a chip is directly determined by its area [43]. While chip area is difficult to determine precisely before a chip is fully designed [32], it is possible to consider some of the factors. Specifically, the reference memory area typically represents a significant proportion of the total area of a motion estimation module, and can be accurately estimated according to the algorithm, prior to a detailed design of the VLSI architecture[32]. In addition, chip area is also impacted by the I/O bandwidth through the complexity of the memory control system, and the area of the data buses[32].

A specific video compression application will dictate the appropriate trade-off between the execution time, memory size and I/O bandwidth. It is important to consider these implemental properties when selecting a motion estimation algorithm for implementation within a VLSI architecture. In the rest of this section, each of these properties will be discussed for the both the traditional full search motion estimation algorithm and the block-adaptive full search motion estimation algorithm.

4.2.1 Number of Search Positions

To estimate the time required to perform block matching, we consider the total number of consecutive search positions. For the traditional full search algorithm number of search positions is given by:

$$NumSearchPositions = (2 \cdot S_X + 1) \cdot (2 \cdot S_Y + 1) \qquad (12)$$

Since the search paths of the sub-blocks is identical to the search path of the 16x16 block, the SADs for all block-types can be calculated simultaneously. As a result, these sub-blocks do not contribute to the time requirements of the motion estimation.

In the case of the block-adaptive full-search algorithm, the search paths or the sub-blocks are generally not identical to that of the 16x16 block, so these block must be considered separately. Furthermore, since the search path for each block is dependent on the estimation results of the other blocks, the motion estimation must the done consecutively. The time required for each macroblock is related to the total number of search position for all sub-blocks, of which there are 41 [21]:

$$NumSearchPositions = 41 \cdot (2 \cdot S_X + 1) \cdot (2 \cdot S_Y + 1)$$

4.2.2 Local Memory Requirement

The local memory requirement must include storage for the original macroblock, for which the motion estimation is being performed, and storage for the reference data. The storage for the original macroblock is constant regardless of the motion estimation algorithms – 256 pixels are required. The actual reference memory requirement is dependent on the specific VLSI design of a motion estimation module. However, the minimum requirement is the memory needed for the estimation of the largest block of pixels, which is 16x16. The size of the reference area is given by (11). Since N_X and N_Y, are both equal to 16, the reference memory size is only influenced by the search range.

4.2.3 I/O Bandwidth

The I/O bandwidth of a motion estimation module is determined mainly by the pixel access rate for loading the search window. The number of pixels required for full search block-matching for a block of pixels is given by (11). In the case where the centre position of the search window (X_C ,Y_C) is block-adaptive, this many pixels are required to be loaded for each block in the original image. In H.264, each macroblock has 41 blocks, as described in 2.4.2. The total memory bandwidth requirement for motion estimation is:

$$\begin{aligned} MemBandwidth / MB = {} & 16 \cdot Area(S_X, S_Y, 4,4) + 8 \cdot Area(S_X, S_Y, 8,4) + 8 \cdot Area(S_X, S_Y, 4,8) \\ & + 4 \cdot Area(S_X, S_Y, 8,8) + 2 \cdot Area(S_X, S_Y, 16,8) + 2 \cdot Area(S_X, S_Y, 8,16) \\ & + Area(S_X, S_Y, 16,16) \end{aligned} \tag{13}$$

For example, a search with *(S_X, S_Y)* = (24,12) has a bandwidth requirement of:

$$MemBandwidth/MB = 67456\ \text{pixel/MB}$$

For the traditional full search, where the search window is centred on the collocated macroblock, the required memory bandwidth is reduced for two reasons. Firstly, the search area for each sub-block within a macroblock is contained within the search area of the 16x16 block. Secondly, there is a predictable overlap of the search windows of subsequent macroblocks. Exploiting this means that only a stripe of new reference pixels must be loaded for each macroblock, for which the reference window of the adjacent macroblock is loaded into local memory. Assuming raster order macroblock processing, this benefit is applicable to all macroblocks of a macroblock row except the left most. For a search with search range *(S_X, S_Y)*, the number of new pixels in each reference window is $16 \cdot (2 \cdot S_Y + 16)$. The total memory bandwidth requirement for each macroblock row of a frame MB_{col} columns wide is:

$$MemBandwidth/row = Area(S_X, S_Y, 16, 16) + (MB_{col} - 1) \cdot 16 \cdot (2 \cdot S_Y + 16) \qquad (14)$$

A standard definition frame is 45 macroblocks wide [52]. The average bandwidth with a search where *(S_X, S_Y)* = (24,12) is:

$$MemBandwidth/MB = \frac{[Area(S_X, S_Y, 16, 16) + (MB_{col} - 1) \cdot 16 \cdot (2 \cdot S_Y + 16)]}{MB_{col}} \qquad (15)$$

$$MemBandwidth/MB = 682.7\ \text{pixel/MB}$$

The traditional full search algorithm obviously has a significantly lower bandwidth requirement as compared to the block-adaptive full search and is therefore better suited to VLSI implementation.

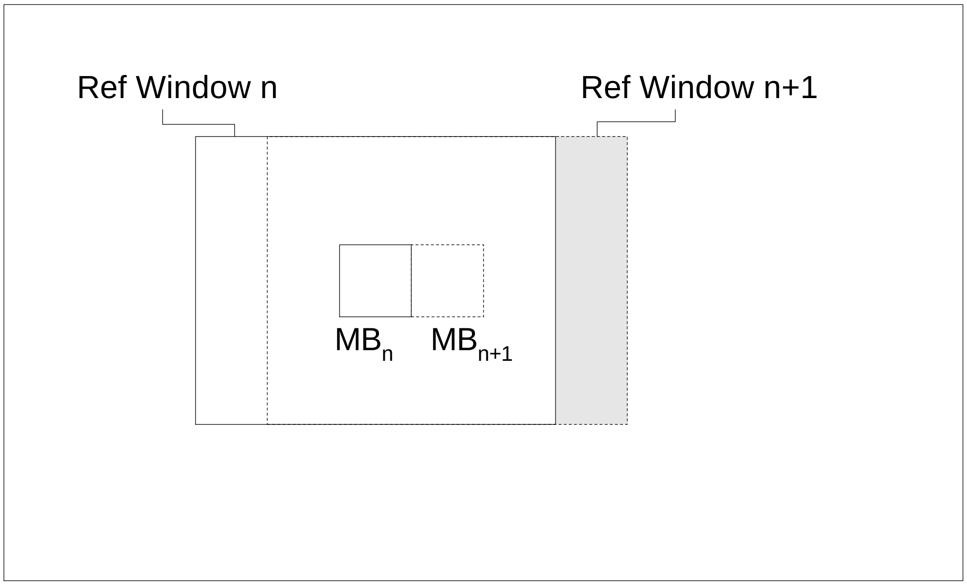


Figure 4-1 - Full Search Reference Window Overlap

4.2.4 Regularity of Execution flow

The efficiency of VLSI implementations is heavily reliant on concurrency of operations, which is brought about through highly pipelined and parallelised architectures [42]. Pipelined and parallel architecture are suited to algorithms that minimise the interdependence of computations. The traditional full search block-matching algorithm has no interdependence between steps in the motion estimation task. Specifically, there is neither feedback between the motion estimation of successive macroblocks, nor feedback within the block-matching of each macroblock. In [53], it was shown that the regularity of the traditional full search algorithm allowed it to be mapped to an efficient array architecture. In contrast, when the block-adaptive full search algorithm is used, the centre of each search window can be chosen according to the motion estimation results of neighbouring blocks[46]. This implies that the estimation of one block must be complete before the motion estimation of the next block can begin. This in turn prevents an efficient parallel or pipelined implementation, increasing the number of cycles required for the motion estimation of each macroblock.

4.3 Motion Characteristics of Video

The traditional full search block-matching algorithm involves a single rectangular search window centred on the collocated macroblock position of the original macroblock. However, all of the implementation advantages discussed in this section would be equally true of a full search block-matching algorithm, where the search windows are not centred on the collocated position, but at a constant offset from that position for all macroblocks in the frame. The next section provides an investigation into the characteristics of the motion parameters for typical video sequences. From this investigation, a modified block-matching algorithm is proposed.

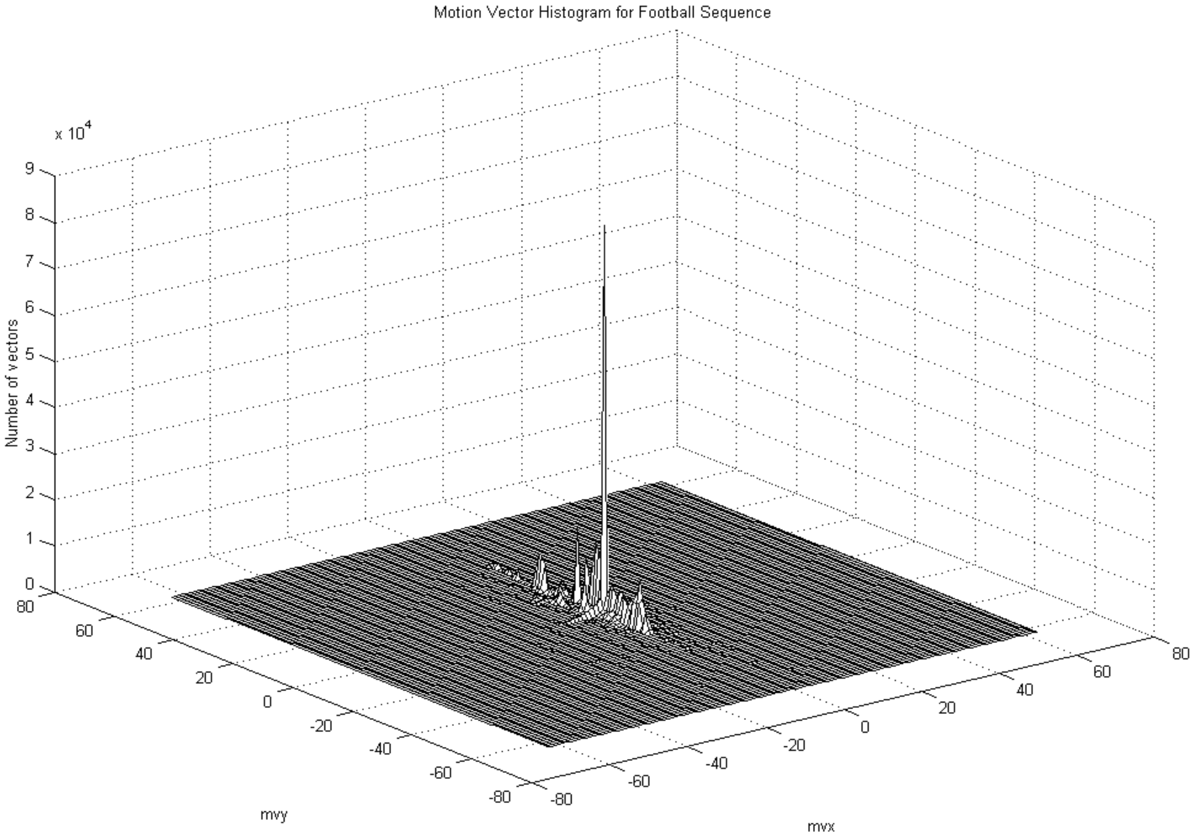


Figure 4-2 - Histogram of Motion Vectors for Full Football Sequence

Many of the fast search algorithms are based on the centre-biased nature of motion vectors. For example, in [50], the authors observe that for many video sequences, including those with high motion, the vast majority of motion vectors selected for motion compensation are near zero. In the standard football sequence, for example, they observed that 80% of the motion vectors used for motion compensation were enclosed in a 5x5 area centred on the zero vectors. Figure 4-2 shows a histogram of the motion vectors for ‘football’ using the JVT reference encoder. It is clear that over the whole sequences, there is a heavy bias towards the zero motion vector. This lends credibility to centre biased motion estimation approaches, including the traditional full search.

Figure 4-3 - Figure 4-8 show a series of motion vector histograms for individual frames of the ‘football’ sequence. In these, it is seen that the centre-bias is a motion characteristic that is true, on average, over the whole sequence, is not necessarily true for each frame. Instead, it is observed that the motion vectors do vary widely over the course of the sequence. There does appear, however, to be a clustering of motion vectors within each frame. In frames 55 to 58 the motion vectors are closely clustered around the vector (22,0) - this corresponds to the pan at that instant of the sequence. If a traditional full search block-matching algorithm were used for motion estimation, a search window with a horizontal search range of at least +/- 24 would be required to identify these motion vectors. However, if a full search block-matching algorithm were used, but centred on the dominant motion cluster, it would be possible to perform equally effective motion estimation with a much smaller search range than with the larger traditional full search. The challenge is then to identify the dominant motion characteristic of the frame and using this information to perform motion estimation.

Nam et. al. [54] observe that the direction of motion of a block is highly related to the direction of motion of the collocated block in the previous frame. We can also observe this temporal correlation of motion vectors in the series of motion histograms for ‘football’. For example, in Figure 4-3 - Figure 4-6 it is seen that the location of the motion cluster in each frame does not change significantly for several frames. When the pan stops from frame 58-60 (Figure 4-6 - Figure 4-8), the cluster moves slowly back towards the centre.

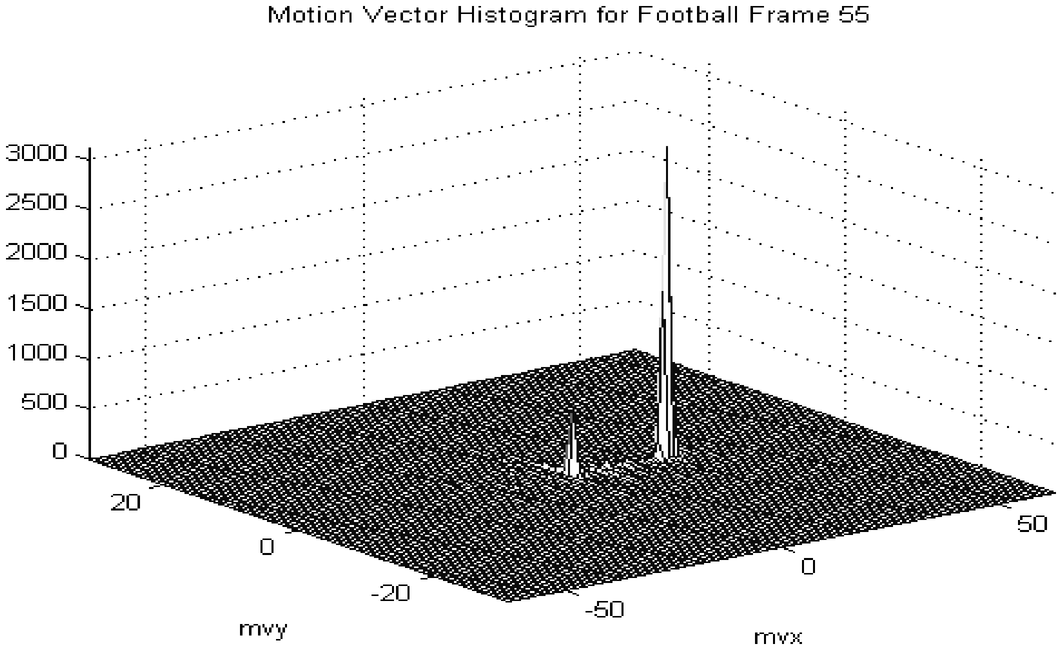


Figure 4-3 - Motion Vector Histogram for Football Frame 55

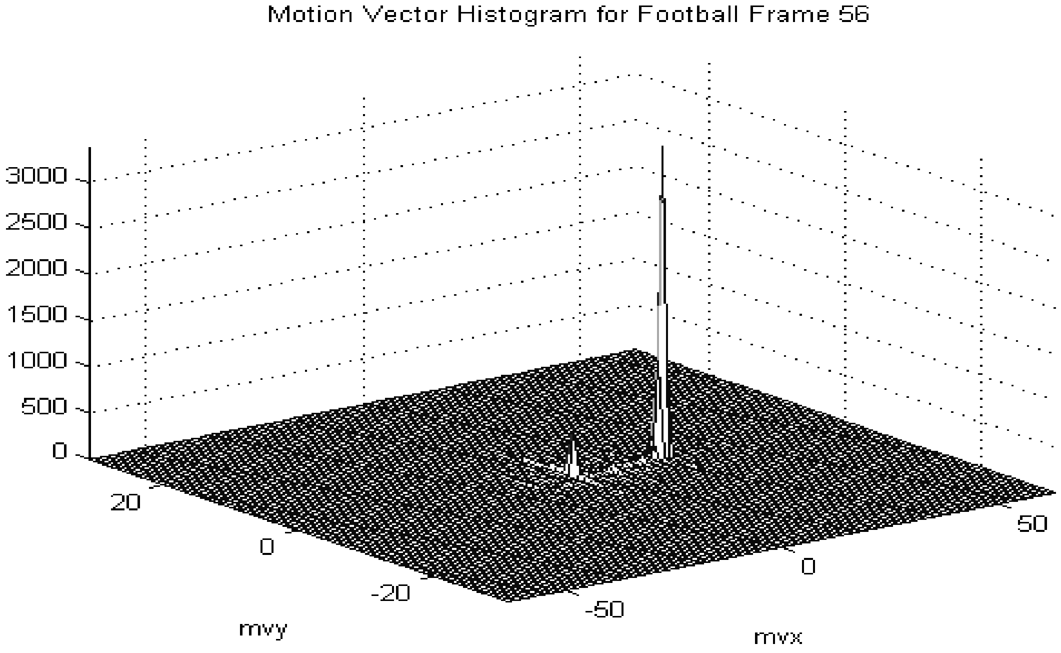


Figure 4-4 - Motion Vector Histogram for Football Frame 56

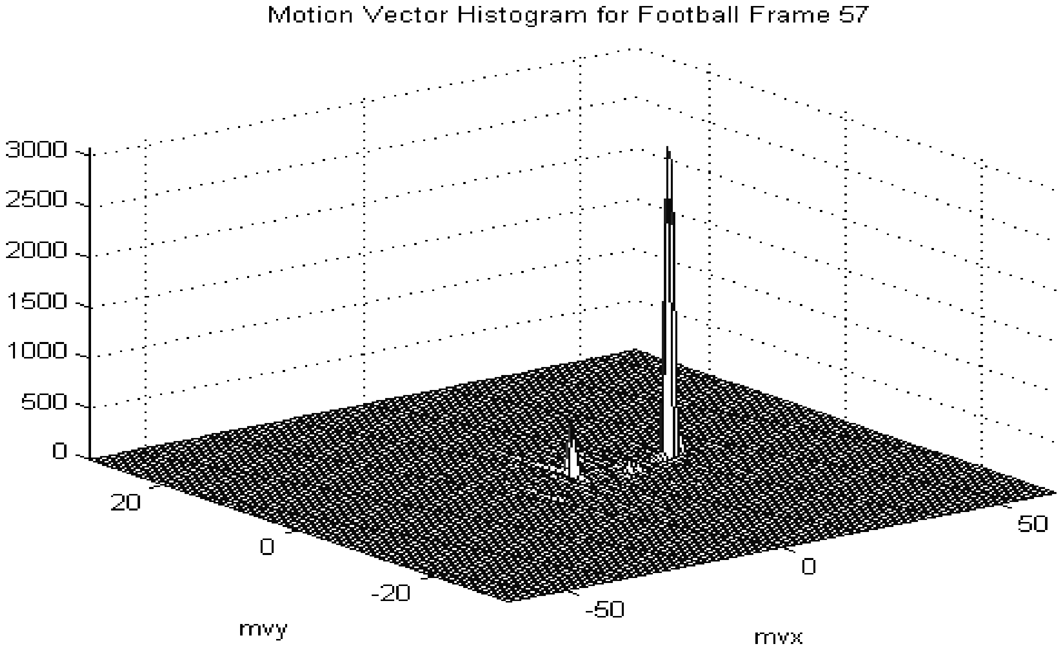


Figure 4-5 - Motion Vector Histogram for Football Frame 57

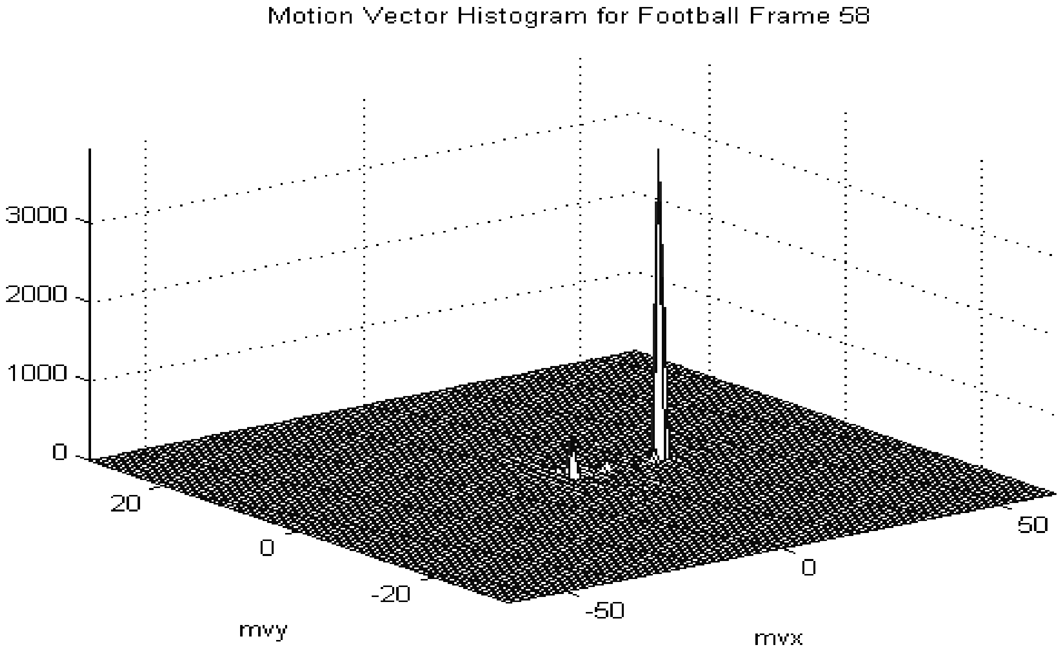


Figure 4-6 - Motion Vector Histogram for Football Frame 58

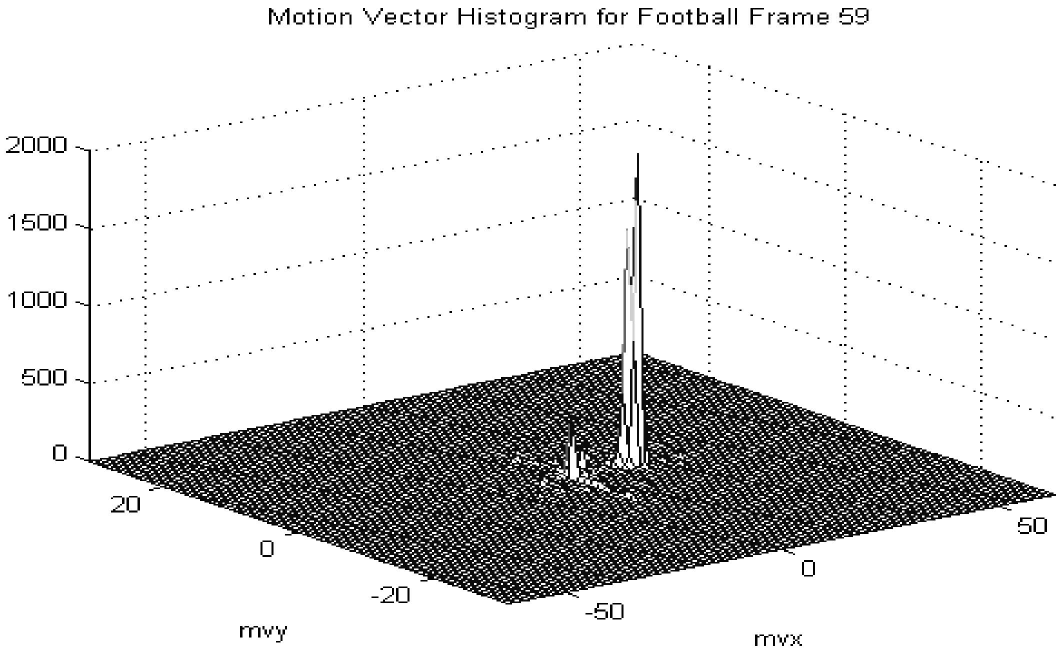


Figure 4-7 - Motion Vector Histogram for Football Frame 59

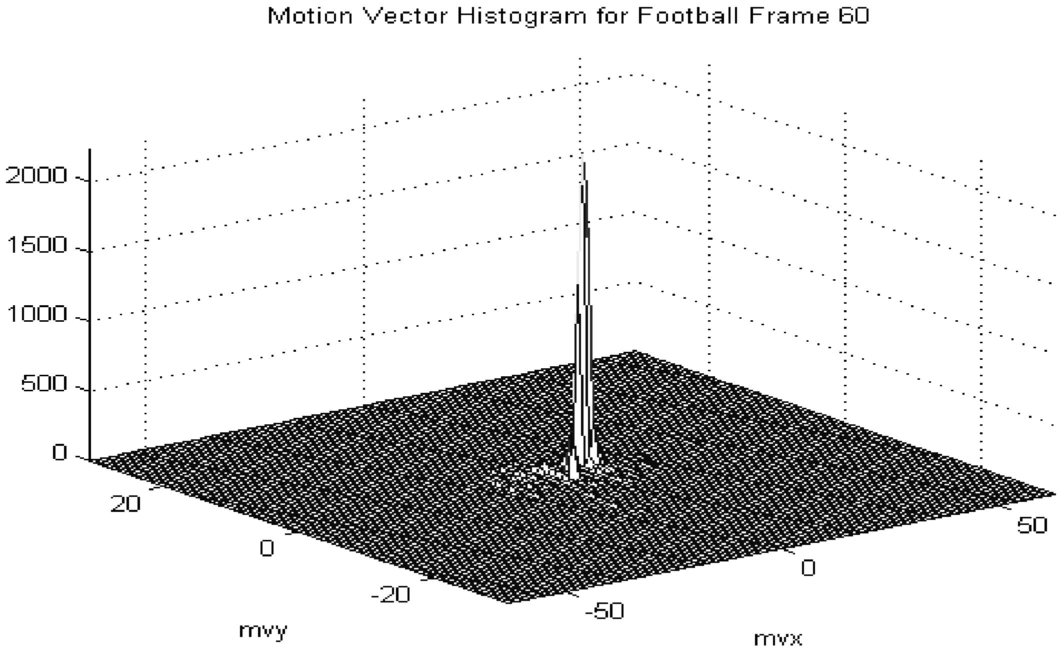


Figure 4-8 - Motion Vector Histogram for Football Frame 60

In other sequences, the dominant motion characteristics of each frame are too complicated to be well represented by a single cluster of motion vectors. Figure 4-9 to Figure 4-14 show the motion vector histograms for several frames of ‘bus’. In each of these frames, there are two distinct clusters of motion vectors. The cluster centred on (0,0) corresponds to the bus in the foreground of the video, while the cluster centred at (25,0) corresponds to the pan of the scenery in the background of each frame. In order to identify these motion vectors using a traditional full search with a horizontal search range of at least +/- 48 would be necessary. However, two smaller windows centred on each cluster could provide equally effective motion estimation, while requiring significantly fewer search positions.

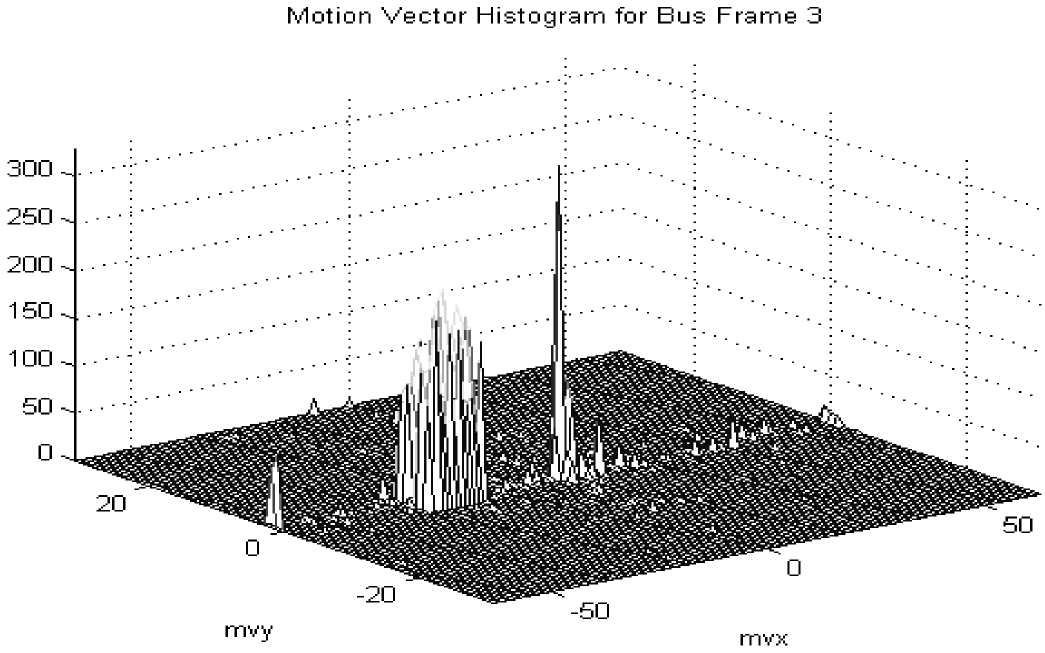


Figure 4-9 - Motion Vector Histogram for Bus Frame 3

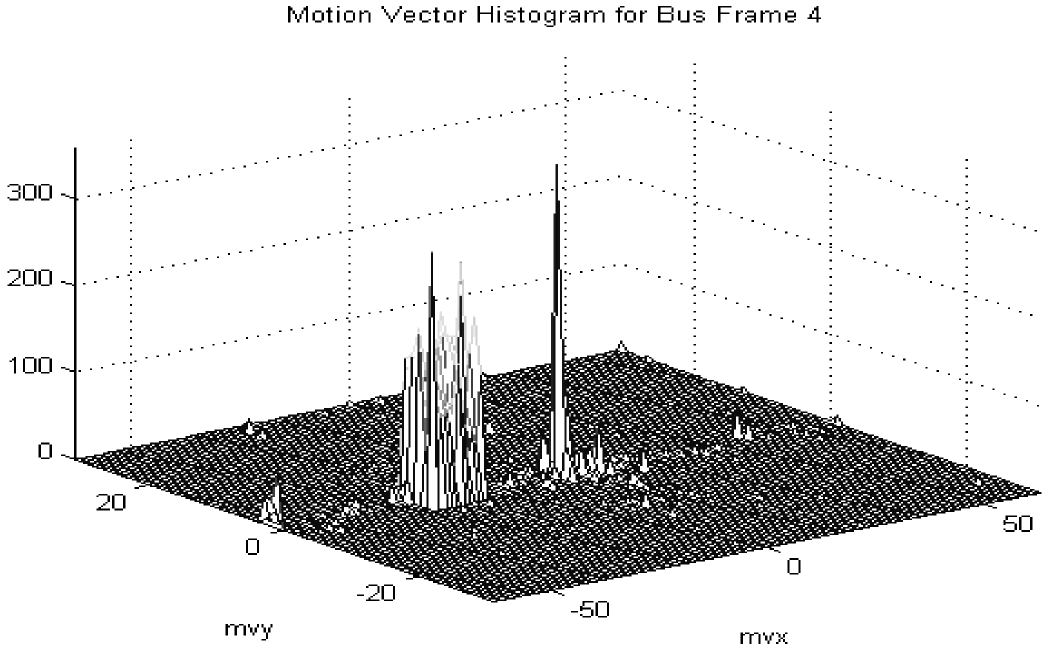


Figure 4-10 - Motion Vector Histogram for Bus Frame 4

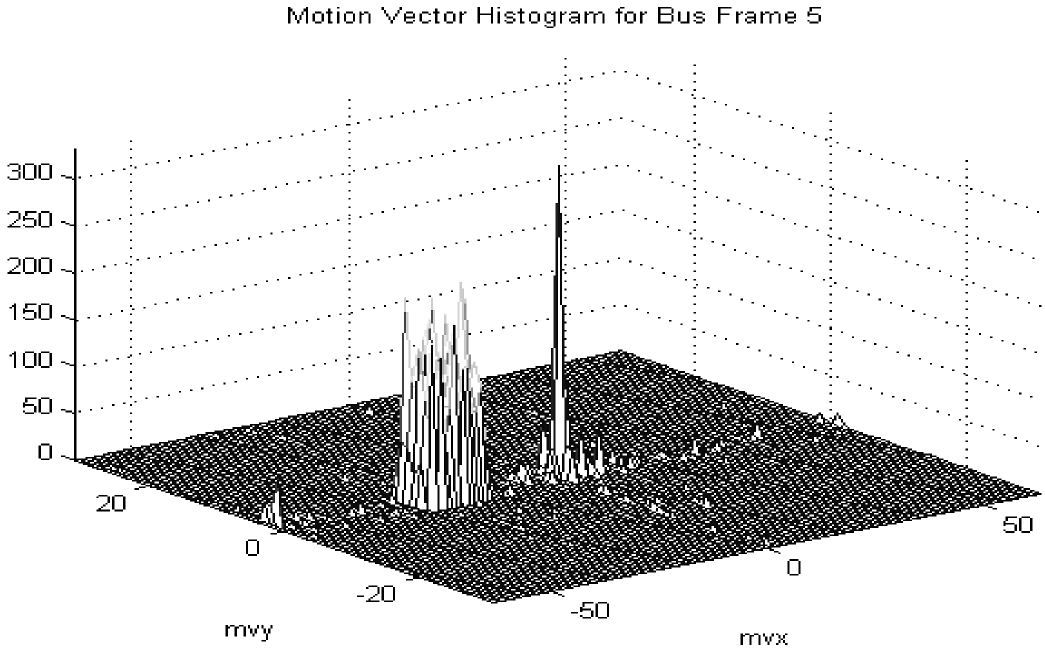


Figure 4-11 - Motion Vector Histogram for Bus Frame 5

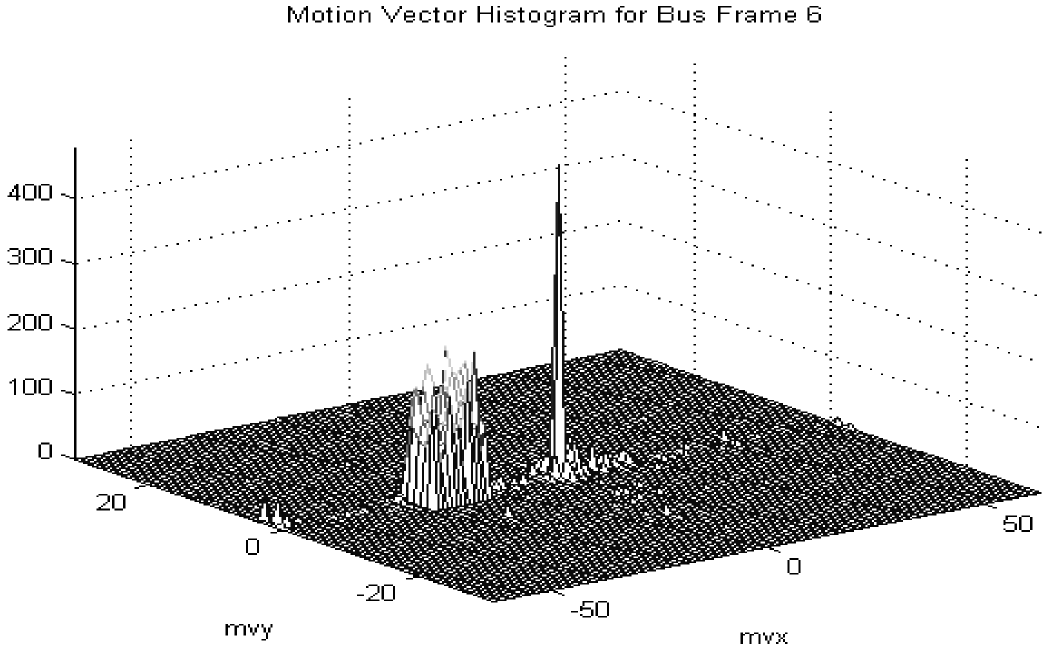


Figure 4-12 - Motion Vector Histogram for Bus Frame 6

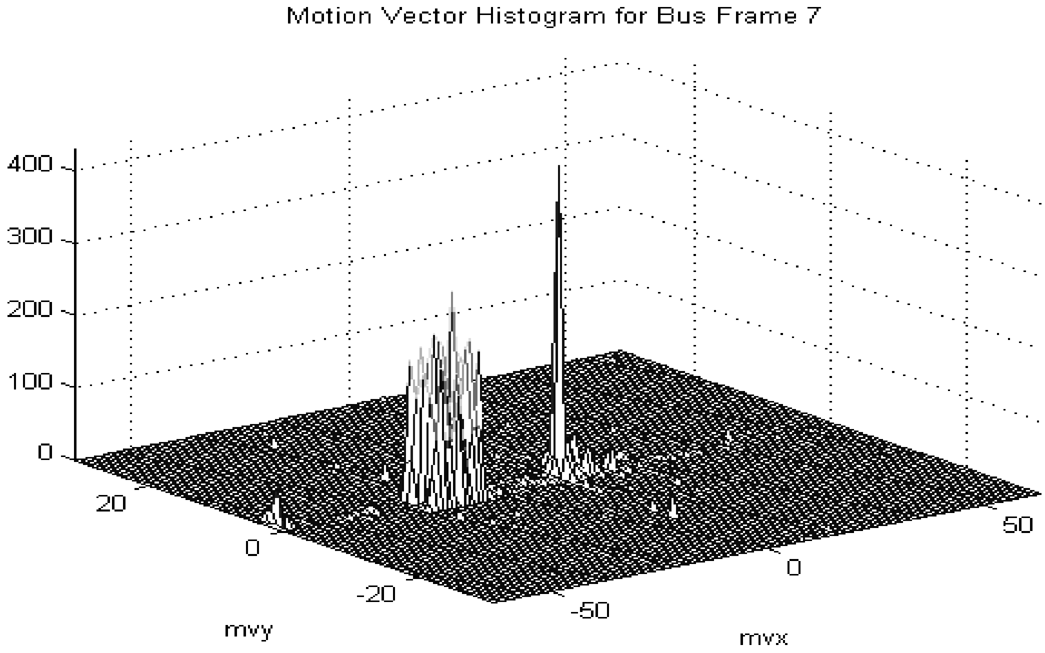


Figure 4-13 - Motion Vector Histogram for Bus Frame 7

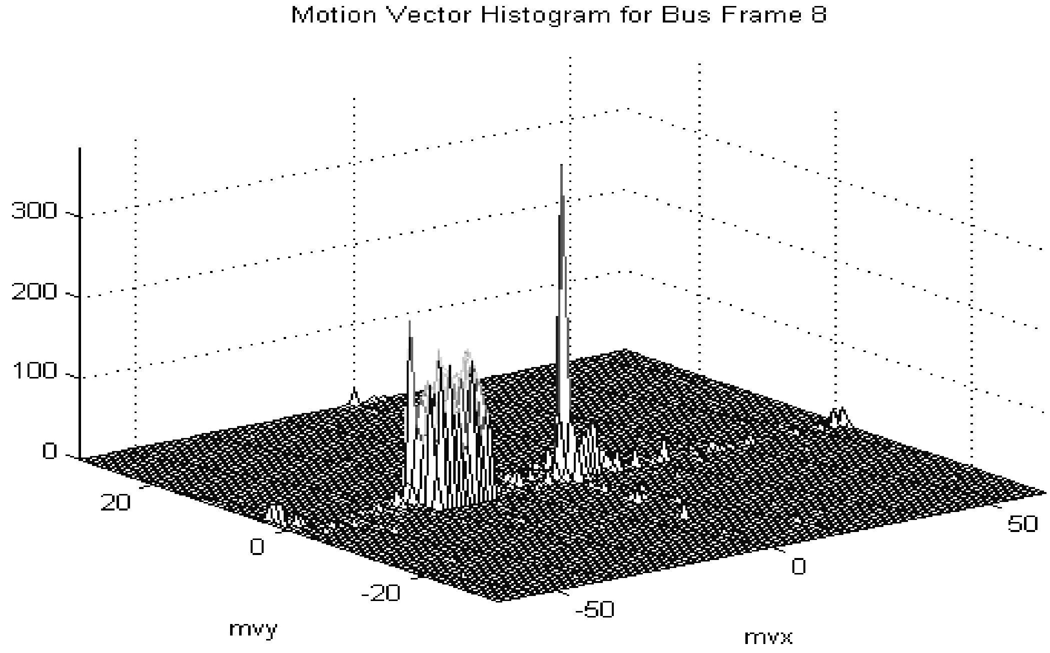


Figure 4-14 - Motion Vector Histogram for Bus Frame 8

4.4 Proposed Motion Estimation

The fundamental approach of the proposed motion search algorithm is to perform the full search algorithm on one or more independent search windows in each frame of the video sequence. The centre of each window is determined by an offset that is constant over each frame. The constant search offsets are determined from analysis of the motion vectors from the previous frame. Figure 4-15 shows the motion estimation windows for the two search window algorithm, for example. In this case, the positions of the search windows are defined by two global motion vectors (GMV_0 and GMV_1). The algorithm can be generalised to support any number of search windows, the centre of each being defined by a constant global motion vector.

In a VLSI implementation, this algorithm would be implemented using independent full search modules, each using a different offset. Each such module has all the implementation properties of the full search discussed in section 4.2. The regular data flow that allows efficient variable block-size motion estimation is present, and from Figure 4-15, it can be seen that each window has the predictable overlap between consecutive macroblocks, giving the memory bandwidth benefit. Since the size of each search window can be much smaller, the total number of search positions, and therefore the overall memory bandwidth, is reduced. The implemental benefits will be discussed in greater detail in the next section.

The proposed Constant Search Offset Motion Estimation algorithm (CSOME) is described by the following steps:

1. Initialise the search modules with the global motion components (GMV_i).
2. For each macroblock in the frame:
    a. Perform the full search algorithm for each global offset, where the centre of each motion search:

    $$\left(X_C, Y_C\right) = \left(X_O + X_{GMVi}, Y_O + Y_{GMVi}\right) \tag{16}$$

    where (X_O,Y_O) is the position original block of pixels and X_{GMVi} *and* Y_{GMVi} are the horizontal and vertical components of the i^{th} global motion vector.

    b. Compare the results of all the search modules, and choose the best motion vectors for each block within the macroblock.
3. Store each best motion vector for use for motion compensation.
4. Analyse the set of chosen motion vectors to determine the global offsets for the following frame. As motion estimation is performed on a pixel resolution reference frame, the global offsets are integer resolution.
5. Repeat steps 1-4 for each inter-predicted frame in the video sequence.

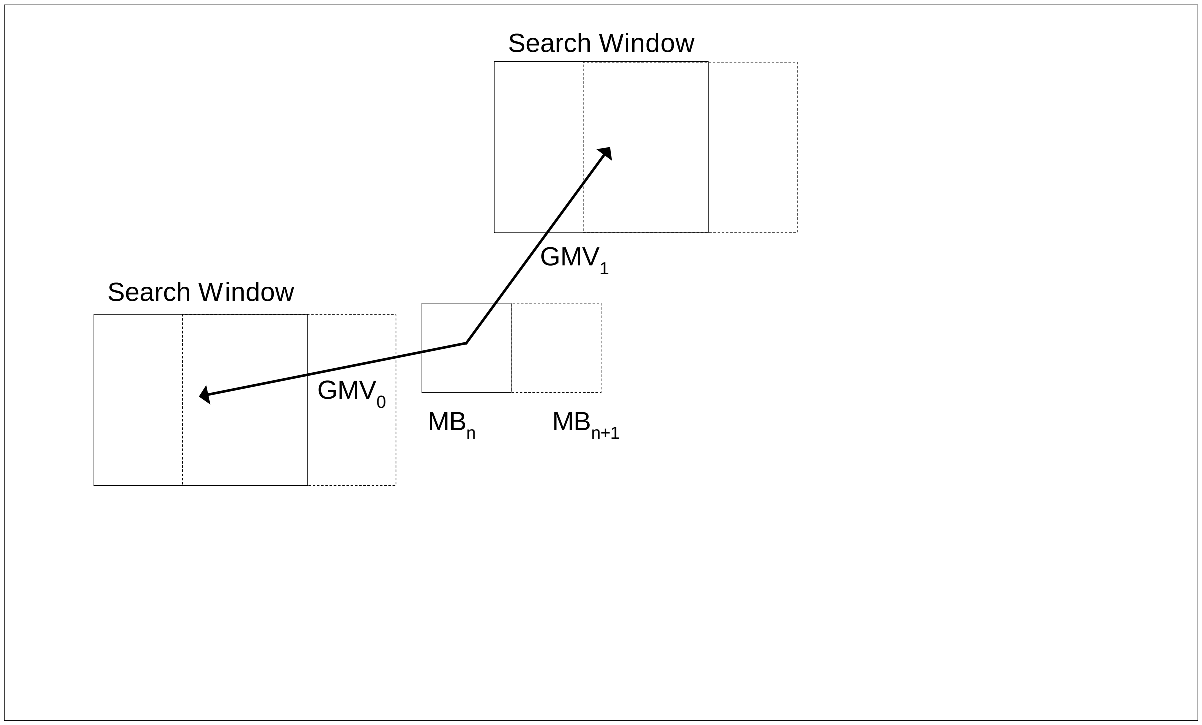


Figure 4-15 - Multiple Constant Offset Full Search Windows

4.4.1 Implemental Properties of Proposed Method

In this section, the proposed method will be examined with respect to the four VLSI implemental properties discussed in section 4.2.

Number of Search Positions

This consideration is important for estimation of the execution time of the motion estimation algorithm in the VLSI implementation. The proposed motion estimation algorithm requires parallel execution of multiple full search estimation modules. The execution time is thus dictated by the number of search positions tested in each module. The number of search positions is given by (12):

$$NumSearchPositions = (2 \cdot S_X + 1) \cdot (2 \cdot S_Y + 1) \qquad (12)$$

The analysis of the motion vectors between frames to determine the global motion components requires far less computation than the combined motion estimation for each macroblock. Furthermore, this analysis would likely be executed on a CPU rather than within the VLSI implementation. Accordingly, this computational expense will not be considered in the discussion of execution time requirements.

Local Memory Requirement

The reference pixel memory requirement directly affects the size of the VLSI implementation, and therefore its cost. The memory requirement for the overall algorithm is the sum of the memory required for each parallel full search module, given by (12):

$$ReferenceMemory = Q \cdot (2 \cdot S_X + 16) \cdot (2 \cdot S_Y + 16) \qquad (17)$$

where S_X and S_Y are the dimensions of each search window in the horizontal and vertical directions respectively and Q is the number of search windows

I/O Bandwidth

The overall reference pixel bandwidth of the proposed motion estimation algorithm is the sum of the required bandwidth of each full search module:

$$MemBandwidth / MB = Q \cdot \frac{[Area(S_X, S_Y, 16, 16) + (MB_{col} - 1) \cdot 16 \cdot (2 \cdot S_Y + 16)]}{MB_{col}} \qquad (18)$$

Regularity of Execution Flow

The proposed motion estimation exhibits the same properties as the traditional full search algorithm in terms of regularity of execution flow. As with the traditional full search algorithm, there is interdependence of execution path neither within the motion estimation of each macroblock, nor between the motion estimation of successive macroblocks within the same frame. This property facilitates an efficient VLSI implementation, requiring a short execution time.

In the next section, the analysis methods for determining the global offsets from the motion vectors are described (step 4). In the following section, the details of the full search algorithms will be discussed (step 2).

4.5 Motion Vector Analysis

The full search block-matching modules are initialised according to the dominant motion components of the previous frame. These dominant motion components are identified by identifying motion vector clusters in the set of motion vectors for a frame. The prototypes of the clusters are then used as the dominant motion components. For the proposed algorithm, two methods for identifying the motion vector clusters were used: the well-known k-means clustering algorithm [55] and a histogram peak detection algorithm.

4.5.1 K-Means Clustering

The first clustering algorithm used is the k-means clustering algorithm [55]. In this iterative method, each element is classified into one of k clusters. On convergence of the algorithms, each cluster is represented by a prototype that is the mean of the cluster. These prototypes are then used as the global motion offsets to initialise the motion search modules. The algorithm is as follows:

1. Initialise the k cluster prototypes to the global motion vector from the previous frame.
2. For each 4x4 (inter-coded) block of the previous frame:
    a. Classify its motion vector into one of the k clusters according to the smallest distance to the cluster prototypes. The distance measure is discussed below.

   b. Re-calculate the mean of the cluster members to update the cluster prototype to reflect the cluster prototype.

3. Repeat step 2 until all cluster prototypes converge. Here convergence occurs when the distance between the prototypes of the same cluster for consecutive iterations has a value of less than 0.5 pixels. Since the global offsets used for initialising the motion estimation are integer, further iterations to achieve more precise cluster prototypes would be redundant.

One defining characteristic of the k-means algorithm is the distance measure used for classification of each motion vector. A common approach is the Euclidean distance:

$$dist_{euc} = \sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2}\ , \qquad (19)$$

where the cluster prototype is (x_o,y_o) and the candidate motion vector is (x_i,y_i). The distance measure defines the shape of the equal probability contours of the clusters. The Euclidean distance, for example, results in circular clusters. The clusters of motion vectors are not expected to have a simple circular shape. In general, motion characteristics of video show more horizontal motion than vertical. This is reflected in the fact that motion search windows are often rectangular, with a greater horizontal dimension than vertical dimension.

In order to account for this asymmetry, two other distance measures were used to better capture the nature of the motion vector clusters. Firstly, the Euclidean metric was generalised to reflect the different dimension of the search windows:

$$dist_{ged} = \sqrt{\left(\frac{x_0 - x_i}{S_X}\right)^2 + \left(\frac{y_0 - y_i}{S_Y}\right)^2}\ , \qquad (20)$$

where S_X and S_Y are the dimensions of the search windows in the horizontal and vertical directions respectively. This distance measure assumes elliptical clusters that are aligned with the horizontal and vertical axes. The Euclidian distance measure can be generalised further to allow clusters to be unaligned with the horizontal and vertical axes. In this application, that would be inappropriate since it would be impractical to implement a search window that is not aligned with these axes.

The second distance measure for the clustering algorithm was designed to exactly reflect the shape of the search windows. Specifically, this distance measure has equal probability contours that are rectangular and similar to each search window. This measure is:

$$dist_{rect} = \sqrt{\max\left(\left|\frac{x_0 - x_i}{S_X}\right|, \left|\frac{y_0 - y_i}{S_Y}\right|\right)} \tag{21}$$

It was found that the effectiveness of the proposed motion estimation was not greatly affected by the distance measure, but of the three, the generalised Euclidean distance measure (20) consistently resulted in a bitrate reduction of about 0.5% as compared to the other two measures. The improvement over the simple Euclidean distance was expected since it does not account for the expected shape of the motion vector clusters. The rectangular distance measure was designed to reflect the shape of the search window. While the relative dimensions of the search windows do reflect the expected shape of the motion vector clusters, that the search windows are rectangular is a result of implementation practicalities rather than the expectation of rectangular motion vector clusters. As a result, it is reasonable that clustering based on the rectangular distance measure was outperformed by the elliptical distance measure, since the latter better reflects the actual shape of the clusters.

4.5.2 Single Iteration K-Means Clustering

The above k-means algorithm is an iterative approach, which requires processing each motion vector from the previous frame more than once to determine the final cluster prototypes. There are two main disadvantages to this approach. Firstly, since it continues until convergence of the prototypes, the amount of computation required for the clustering is dependent on the video data. Secondly, it requires memory to store the motion vectors for the entire frame.

A simplification to this algorithm that is worth evaluating is to only perform one iteration of clustering. In this case, each motion vector is only referred to once, and can therefore be discarded once clustered. This means that the motion vectors do not have to be stored in memory for the purpose of the global offset estimation. In addition, this implies a more regular processing module with a constant latency, which is better suited to integration with a VLSI implementation. It is expected that this clustering will result in less effective motion estimation, but if the loss in encoding performance is sufficiently small, the implementation benefit may be of greater importance.

4.5.3 Histogram Peak Detection

While the clustering methods described above do, somewhat, account for the nature of motion vector clusters, they do not account for the details of the full search motion estimation algorithm. The results of the above clustering could result in global motion vectors that may accurately reflect the motion vector clusters, but are sub-optimal in terms of the motion estimation for the next frame. One such problem may occur in a frame where there is really a single predominant cluster of motion vectors - the resultant global motion vectors are likely to be close to each other. The result of this is that the search windows for the frame will overlap significantly, thereby performing redundant block-matching calculations.

A second potential inadequacy of the k-means clustering is that while it accounts for the shape of the search windows, is does not account for their size. This means that motion vectors that are classified as belonging to a cluster, may not be within the search window placed on the cluster centre. To provide a possible alternative motion vector analysis algorithm to k-means clustering, a histogram peak detection algorithm was developed for the motion vector analysis.

The method is defined by the following steps:

1. A two dimensional histogram of the previous frame motion vectors is generated with a bin resolution of integer motion vector values. I.e. $Hist[X, Y]$ is the number of occurrences of motion vectors *(X_{MV}, Y_{MV})* where

$$X - 0.5 \leq X_{MV} \leq X + 0.25 \text{ and}$$

$$Y - 0.5 \leq Y_{MV} \leq Y + 0.25$$

2. The histogram is smoothed with an $(S_X + 1)\times(S_Y + 1)$ averaging mask, where S_X and S_Y are the dimensions of the search windows:

$$Hist_{SM}[X,Y] = \sum_{i=-\frac{S_X}{2}}^{\frac{S_X}{2}} \sum_{j=-\frac{S_Y}{2}}^{\frac{S_Y}{2}} Hist[X+i, Y+j] \tag{22}$$

3. The bin with the maximum value in smoothed histogram is identified, and the motion vector corresponding to that histogram bin is taken as a global motion component:

   Define (X_{PEAK}, Y_{PEAK}) such that $Hist_{SM}[X_{PEAK}, Y_{PEAK}] = \max(Hist_{SM}[X,Y])$. Then the global motion component is

$$GMV_i = (X_{GMVi}, Y_{GMVi}) = (X_{PEAK}, Y_{PEAK}) \tag{23}$$

4. All motion vectors that lie within a search window centred on the global motion component with search ranges S_X and S_Y are removed from the original histogram:

$$Hist[X,Y] = 0 \text{ for all (X,Y) such that}$$

$$X_{GMVi} - S_X \le X \le X_{GMVi} + S_X \text{ and}$$

$$Y_{GMVi} - S_Y \le Y \le Y_{GMVi} + S_Y$$

5. Steps 2-4 are repeated for each required global motion component.

There are several important features of this method that differentiate it from the k-means clustering methods discussed in the previous sections. The first conceptual difference is the way in which the quality of a cluster is measured. K-means clustering is a method that reduces the sum of the distances of each data point to the prototype of the cluster to which it belongs. The quality of a cluster is defined, therefore, by the average distance of the points within the cluster to the cluster prototype. This means that the more narrow the distribution of the cluster, the better. However, when using the results for placement of a block-matching search window, what is important is the number of best block-matches that will be found according to the window placement.

The purpose of histogram smoothing of step 2 above is to count the number of motion vectors that are within a certain area surrounding a cluster centre, without regard for the distribution of the data within that area. The size of the smoothing window must necessarily be smaller than the size of the search window to allow some tracking of the motion characteristics of the frame. Using the full search algorithm to search around each global motion component, all motion vectors found for a frame will be within the search window area centred on each motion component. If the smoothing window were equal in size to the search window, then the optimal solution according to the algorithm would be to not alter the search window placement from frame to frame. This is clearly not reasonable since the motion characteristics of a video sequence do change over time. The $(S_X+1)\times(S_Y+1)$ smoothing window size was chosen to balance the need to track changes in the motion characteristics of the video sequence with increasing the number of data points that are used to determine the global motion components.

The second differentiating feature of this histogram method to the k-means clustering is the explicit reduction of redundant block-matching calculations. If the global motion components used to initialise the full search block-matching modules are in close proximity, then the resulting search windows for each macroblock will overlap. Where there are overlapping search windows for a macroblock, some candidate motion vectors will be tested in the course of multiple full search modules, hence redundant block-matching calculations are performed. In step 4 of the histogram method, the motion vectors within a neighbourhood, equivalent to the size of a search window, of the identified global motion component are removed from the histogram. Therefore, all motion vectors that would be candidate motion vectors in the full search module initialised with one global motion component are removed from consideration when identifying remaining global motion components. The result will be reduced overlap between different search windows.

4.6 Full Search

Once the global motion components have been identified for each frame, the subsequent frame is encoded using multiple full search motion estimation modules for each block. As discussed in section 3.1, the full search block-matching algorithm is defined by the size and placement of the search window and by the block-matching cost function used to evaluated each candidate position. In the proposed method, the placement of the search windows is defined by the global motion components. The size of the search window will be determined with reference to the nature of the video compression application – for example, the larger the frame resolution of the video sequence, the more likely the search window will be larger.

The block-matching cost function measures the distortion between the original pixel block and the candidate predicted pixel block. Typically, the sum of absolute differences is used for this measure. The recommended motion estimation algorithm provided in the reference encoder uses the SAD measure, but also incorporates a penalty according to the bit-rate cost of the motion vectors. This is based on a rate-distortion optimisation approach to the motion estimation problem [56].

Sullivan and Wiegand [56] note that the goal of an encoder is to optimise the distortion, D, of an encoded video sequence, subject to a constraint on the bit-rate, R. That is:

$$\min\{D\}, \text{subject to } R \le R_c\,, \tag{24}$$

where R_c is the bit-rate constraint. This optimisation can by solved using the Langrangian formulation of the minimisation problem given by:

$$\min\{J\}, \text{ where } J = D + l \cdot R\,, \tag{25}$$

where J is minimised for a particular value of the Langrangian multiplier l. For a given value of l, the solution to (25) corresponds to an optimal solution to (24) for a particular value of R_c. This optimisation approach can be used for motion estimation by using a block-matching cost function of the form:

$$J_{MOTION} = D + l_{MOTION} \cdot R_{MOTION}\,, \tag{26}$$

where J_{MOTION} is the cost value associated with a candidate motion vector, D is the distortion measure between the original pixel block and the reference block, and R_{MOTION} is the bit-rate associated with encoding the candidate motion vector. Assuming the SAD distortion measure is used for the block-matching distortion measure, the appropriate value of l was developed as a function of the quantisation step-size (Q) [56]:

$$l_{MOTION} = \sqrt{0.85}Q \tag{27}$$

Using this block-matching cost results in greater compression efficiency over using only the SAD distortion measure. As a result, this method has been incorporated into the H.264 reference encoder.

As discussed in section 2.4.2, predictive coding is used for encoding the motion vector associated with each block. In order to estimate the R_{MOTION}, the differential motion vector is calculated for each candidate motion vector during the motion estimation process:

$$dMV = CMV - predMV\,, \tag{28}$$

where dMV is the differential motion vector, CMV is the candidate motion vector, and $predMV$ is the predicted motion vector. The bit-rate cost of the motion vectors is then estimated by:

$$R_{MOTION} = \log_2 \left| dMV_X \right| + \log_2 \left| dMV_Y \right|, \tag{29}$$

where dMV_x and dMV_y are the horizontal and vertical components of the differential motion vector. Since this motion estimation approach uses the predicted motion vector for each block, which is derived from the motion vectors of neighbouring blocks, it is necessary to have made all encoding decisions for one block before performing motion estimation on the next block. This approach is therefore ill-suited to an efficient VLSI implementation, as explained in section 3.2.

For the full search motion estimation in the proposed method, the block-matching cost function will still be in the form of (26). Since the true differential motion vector, cannot be calculated for each block during estimation, it will be estimated using the global motion vector corresponding to the search window:

$$dMV \approx CMV - GMV_i \tag{30}$$

where GMV$_i$ is the global motion vector corresponding to the search window.

4.7 Experimental Setup

The proposed motion estimation algorithm was designed for use in a block-based transform video encoder using variable block-size motion compensation. The proposed algorithm was tested with the emerging H.264 video compression standard[21]. Specifically, the multiple constant offset motion estimation algorithm was implemented in the JVT H.264 reference encoder [46][1]. As this research was completed while the H.264 standard was under development, an intermediate version of the standard is used – this is defined in [21]. However, the proposed motion estimation process will be equally applicable to the final version of the standard as to the version used for testing.

4.7.1 Encoding Parameters

Several parameters define the encoding process used in the H.264 encoder. These control the compression tools that are used for encoding and the encoding decisions that are made to trade-off quality with compression efficiency.

GOP Structure. For these experiments, an infinite IPPP GOP structure is used for encoding. This means that the first frame in each sequence is coded as an Intra-frame, and each successive frame is coded as an inter-predicted frame. Bi-predictive frames are not used in these experiments, but the proposed method is easily adapted to the bi-directional motion searches.

Quality. The experiments were done using variable bit-rate (VBR) – constant quality. Specifically, the quantisation parameter (QP) is constant throughout each encoded sequence. To assess the performance of the motion search across the range of quality supported by the H.264 standard, each sequence was encoded using four QP values (20, 25, 30, 35).

[1] H.264 reference software available at standards.pictel.com

Supported Motion Compensation Block-Sizes. All motion compensation block-sizes supported by the H.264 standard were used for encoding each sequence.

4.7.2 Test Video Sequences

A set of six video sequences is used for evaluating the performance of the proposed motion estimation algorithm (Table 4-1). These video sequences are all commonly used for evaluation of video compression tools and are publicly available through Video Quality Experts Group (VQEG).

4.7.3 Motion Estimation Algorithms and Parameters

The benchmark motion estimation algorithm used for these experiments is the block adaptive full search, where the search window for each block is centred according to the motion vector predictor for that block. As discussed above, this is inefficient for VLSI implementation, but provides a suitable reference, against which to compare all other motion estimation algorithms. A very large search window (+/-128x64) is used for each search to ensure near optimal motion estimation.

A non-adaptive full search algorithm will also be used. For this algorithm, the placement of the search window will be centred on the co-located block in the reference frame to the block whose motion is being estimated. This is done to provide a comparison to a likely VLSI implementation.

Several implementations of the CSOME were evaluated in the experiments. Specifically, CSOME using one, two and four search offsets was performed. In addition, each of the three motion vector clustering algorithms, described in 4.5, was used. Table 4-2 lists the motion estimation algorithms used and the total number of positions used for each search. In order to provide a fair comparison between motion estimation algorithms, the dimensions of the search windows were chosen so that each method would use a similar total number of search positions.

Name	Resolution	Number of Frames
Bus	720 x 240	75
Canoa	720 x 288	110
Ferris	720 x 240	60
Football	720 x 240	130
Mobile and Calendar	352 x 240	130
Rugby	720 x 288	110

Table 4-1 - Video Sequences Used for Evaluation of Motion Estimation

Name	Number of Search Windows	Search Range	Total Number of Search Position	Search Window Centre
JVT_128	1	+/-128x64	33153	Block-adaptive
JVT_16	1	+/-16x8	561	Block-adaptive
COL_16	1	+/-16x8	561	Colocated block
CSOME_1_16	1	+/-16x8	561	Single global offset
CSOME_2_11	2	+/-11x5	506	Two global offsets
CSOME_4_8	4	+/-8x4	612	Four global offsets
COL_24	1	+/-24x12	1225	Colocated block
CSOME_1_24	1	+/-24x12	1225	Single global offset
CSOME_2_16	2	+/-16x8	1122	Two global offsets
CSOME_4_11	4	+/-11x5	1012	Four global offsets

Table 4-2 - Evaluated Motion Estimation Algorithms

Table 4-3 shows the implemental properties, discussed in section 4.2, of each of the motion estimation algorithms to be evaluated in the experiments. It is important to note that the implemental properties of the traditional full search algorithms (COL_16 and COL_24) are identical to the single search window CSOME algorithms (CSOME_1_16 and CSOME_1_24). The two search window CSOME algorithms (CSOME_2_11 and CSOME_2_16) require about 45% of the consecutive number of search positions compared to CSOME_1_16 and CSOME_1_24, which implies a faster execution. However, the corresponding memory and I/O bandwidth requirements are about 20% and 60% higher respectively. Similarly, the CSOME_4_8 and CSOME_4_11 further reduce the number of consecutive search positions, while increasing the memory and I/O bandwidth requirements.

Given a similar total number of search positions, CSOME seems to provide a reasonable overall trade-off between execution time and memory and I/O bandwidth requirements. Accordingly, the evaluated motion estimation algorithms have been grouped according the total number of search position. Most compression efficiency comparisons will be made within these groups. The correct trade-off between the implemental properties and coding efficiency will be determined according to the specific application for which these motion estimation algorithms are being considered.

Name	Number of Search Positions / FS Module	Reference Memory Requirement	Reference I/O Bandwidth / MB
JVT_128	33153	39168	1443584
JVT_16	561	1536	35072
COL_16	561	1536	534.8
CSOME_1_16	561	1536	534.8
CSOME_2_11	253	1976	857.4
CSOME_4_8	153	3072	1570.1
COL_24	1225	2560	682.7
CSOME_1_24	1225	2560	682.7
CSOME_2_16	561	3072	1069.5
CSOME_4_11	253	3952	1714.8

Table 4-3 - Implemental Properties of Evaluated Motion Estimation Algorithms

4.7.4 Measures of Performance

The performance of a video compression tool is typically evaluated through the use of Rate-Distortion (R-D) curves. In these curves, the bitrate of the video sequence is plotted against the compression distortion, typically measured in PSNR of the luminance channel. The relative performance of two encoders is represented by the distance between the two curves. In [57], Bjontegaard proposes a method for analysing the R-D curves to provide a single value that represents the average performance difference of two encoders over a range of QP values. By estimating the integral of the difference between the R-D plots, a single value representing the average bit-rate or distortion improvement over the QP range can be calculated. These values are called the Bjontegaard Delta values. In the analysis of the compression performance of the CSOME motion estimation algorithms, both the R-D curves and the Bjontegard Delta bit-rate values will be considered.

4.8 Experimental Results

Throughout this section, summary results and specific results will be presented as the discussion warrants. Full tabulated results and R-D curves are provided in Appendix B.

4.8.1 Performance of Collocated Search

Figure 4-16 shows the performance of the collocated search full search algorithms, as compared to the benchmarks motion estimator. On average, the COL_16 search yields a compression performance loss equivalent to 20.2% compared to the JVT_128 benchmark motion estimator. The larger search range of the COL_16 search yields a smear average loss of 12.0%. Predictably, the greatest loss was observed for the Bus sequence, as it has consistent high motion from frame to frame. The smallest loss was observed in the Mobile and Calendar sequence, which has interframe motion that is small in magnitude.

Figure 4-17 shows the R-D curves of the JVT_128 motion estimation encoder and the two collocated motion estimation encoders for the Canoa sequence. It is observed that the vertical distance between the JVT_128 R-D curve and the COL_16 and COL_24 R-D curves is smaller at higher bit-rates. This is because at higher bit-rates, lower QP values are used for quantisation. The smaller resulting quantisation step-size means that even with a small prediction residual, more coefficients are non-zero and must be encoded in the compressed bitstream. As a result, accurate motion estimation has a greater impact on video compression performance at lower bit-rate, where less prediction residual information is encoded. This observation was consistent for all tested video sequences.

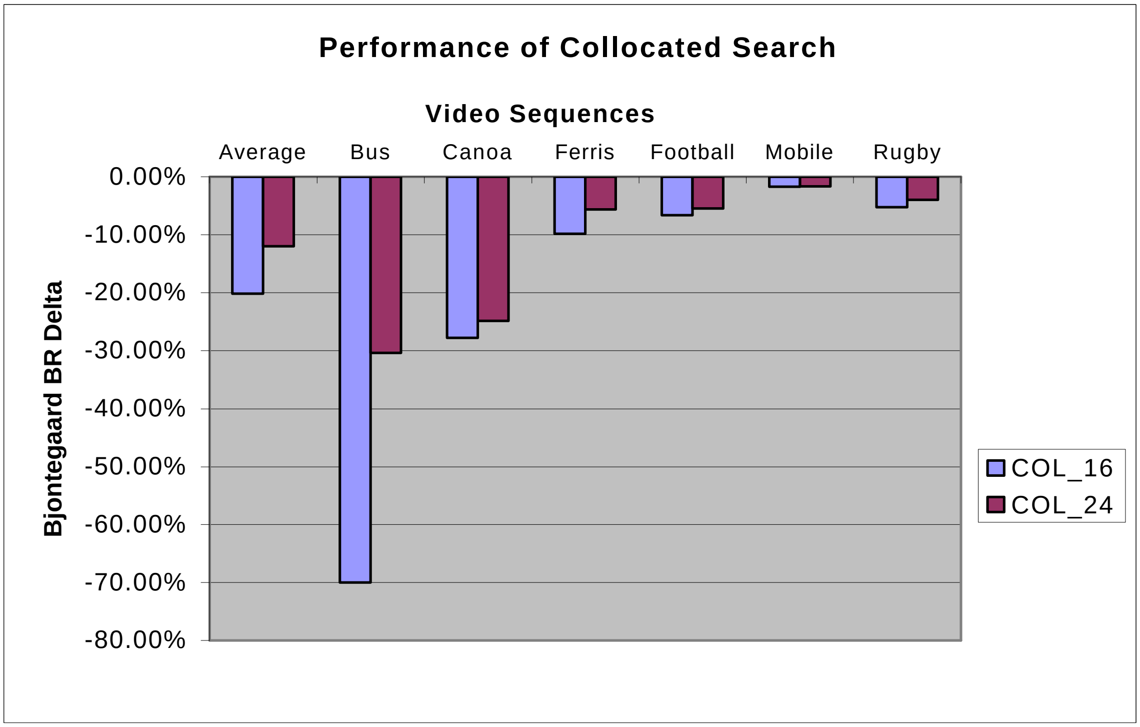


Figure 4-16 - Bjontegaard BR Delta performance of collocated searchs

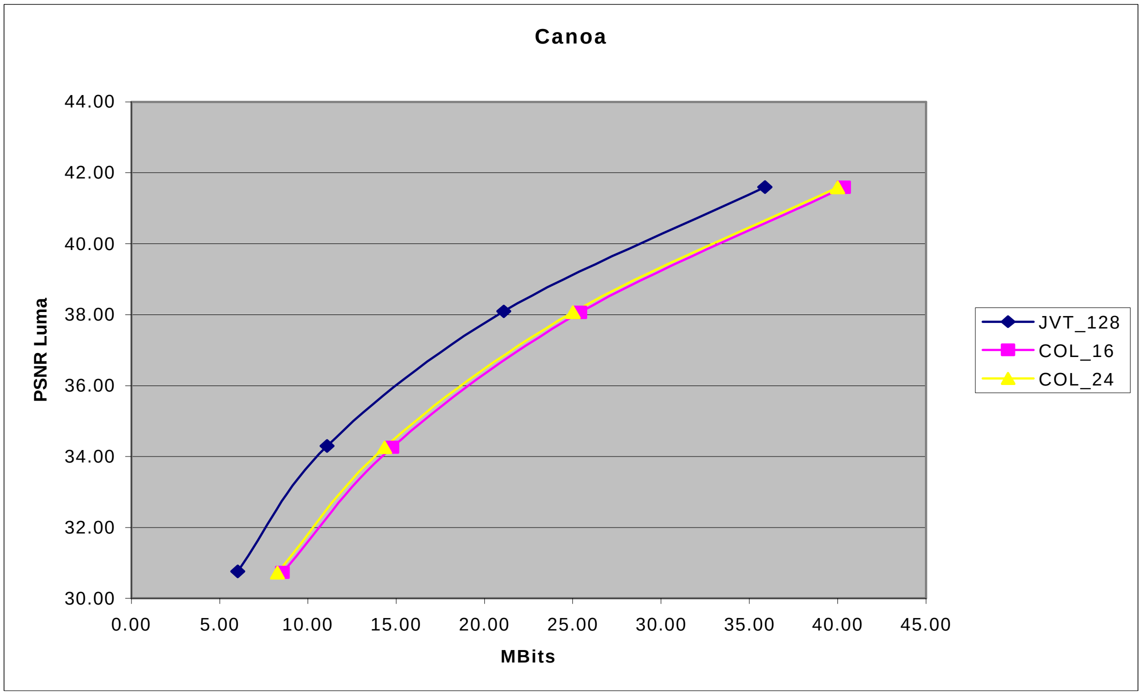


Figure 4-17 - R-D Perfomance of Collocated Motion Search

4.8.2 Motion Vector Clustering Methods

Figure 4-18 shows the relative performance of the three motion vector clustering methods for the CSOME algorithm for one, two and four search windows. For the single window implementation, the performance of the single iteration k-means algorithms is identical to that of the full k-means algorithm, since both reduce to a simple mean of all motion vectors. For the other cases, there is no significant difference in the performance from the full k-means clustering to the single iteration k-means. Since the single iteration method clearly requires less computation, it is the better practical method. The remainder of the discussion of the experimental results will not distinguish between these two methods.

On average, the k-means approach resulted in marginally better compression performance (less than 1% on overall) than the histogram peak detection method. The difference in performance between the clustering methods was most evident for the Mobile and Calendar sequence. The overall Bjontegaard BR delta between these two methods for this sequence is 2.1%. The Mobile and Calendar sequence has several objects each moving in a different direction, but the magnitude of the inter-frame motion for each object is small. The k-means algorithm may have accurately identified each of the motion components corresponding to an object (or the pan) and centred a search window accordingly. The histogram peak detection algorithm prevents significantly overlapping clusters, so could not have centred the search windows with the accuracy of the k-means clustering. Since all of the correct motion vectors for each frame are small, they all would be covered by the first the search window. Any additional search windows would be at best irrelevant, and may result in a few sub-optimal motion vectors to be chosen. Figure 4-19 shows the R-D curves for the k-means clustering and the histogram peak detection method, when using the CSOME_2_16 motion estimation

It is again observable that the compression performance impact of the motion estimation is greater at lower bit-rates. At low bit-rates, the proportion of the bit-rate that is due to coding the motion vector is greater, so any sub-optimal motion vectors that are chosen in motion estimation have a greater relative impact on the compression performance.

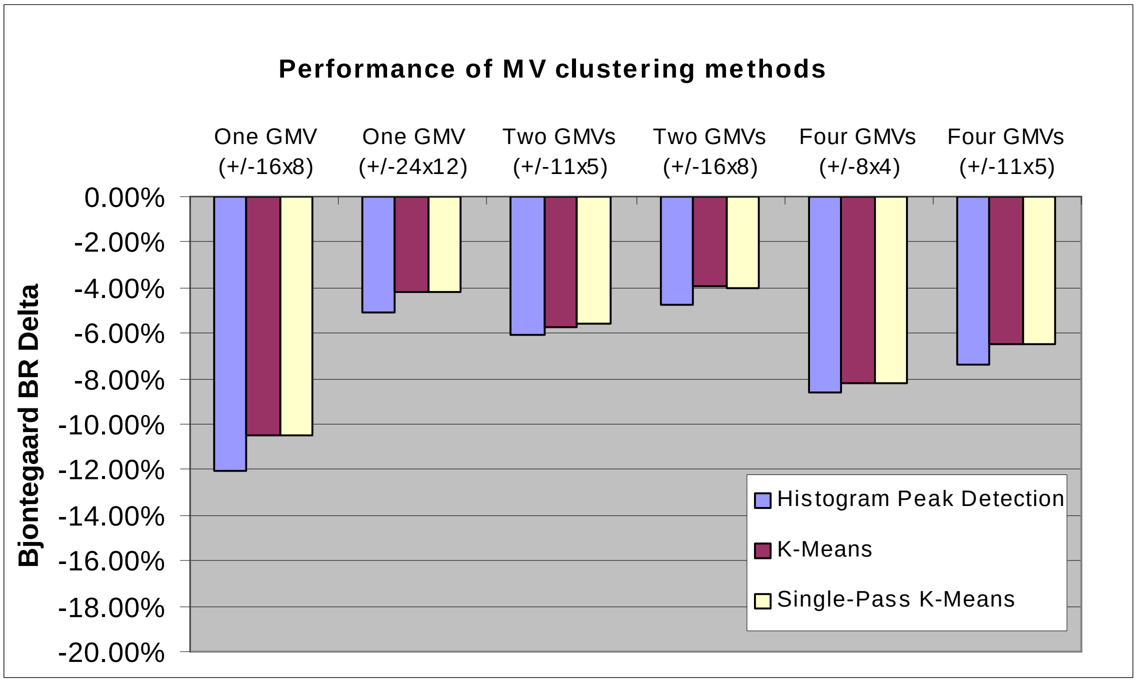


Figure 4-18 - Performance of MV Clustering Methods

4.8.3 CSOME vs Collocated Search

The experiments are divided into two groups. All motion estimation mechanisms within each group have approximately equal numbers of search positions, and are therefore fairly compared. Figure 4-20 shows the relative performance of the motion estimation algorithms that are comparable in size to a +/-16x8 search window. The performance is indicated by the Bjontegaard BR Delta value as compared to the JVT_128 motion estimator.

Overall, the JVT_16 method is the best motion estimation algorithm, in terms of the R-D performance. However, this method is ill-suited to VLSI implementation. The CSOME_2 algorithm is 1.1 % less efficient than this algorithm, and is only 4.0% worse than the benchmark JVT_128 motion estimator. Referring to Table 4-3, the local reference memory requirement of the JVT_16 method is about 25% less than that of the CSOME_2 algorithm. However, the JVT_16 method requires more than twice the number of consecutive search positions and more than 40 times the reference memory bandwidth per macroblock than the CSOME_2 algorithm.

For the Bus sequence, the CSOME_2 algorithm is particularly effective, giving superior R-D performance to the JVT_16 algorithm. In this sequence, there are two motion components: the near-zero inter-frame motion of the bus, and the fast pan of the background. The CSOME_2 was able to accurately identify these motion characteristics and place the search windows accordingly. CSOME_1 was able to provide a significant improvement over COL_16 with the same implemental properties, but the spread of the motion throughout each frame was too great to be fully captured by a single search window. The performance of COL_16 was 70% worse than the benchmark JVT_128 algorithm. The performance of CSOME_2 was only 7.7% worse than JVT_128, using fewer total search positions than COL_16. In comparison to CSOME_2, COL_16 requires 23% less local memory and 38% less I/O bandwidth, but more than double the number of consecutive search positions.

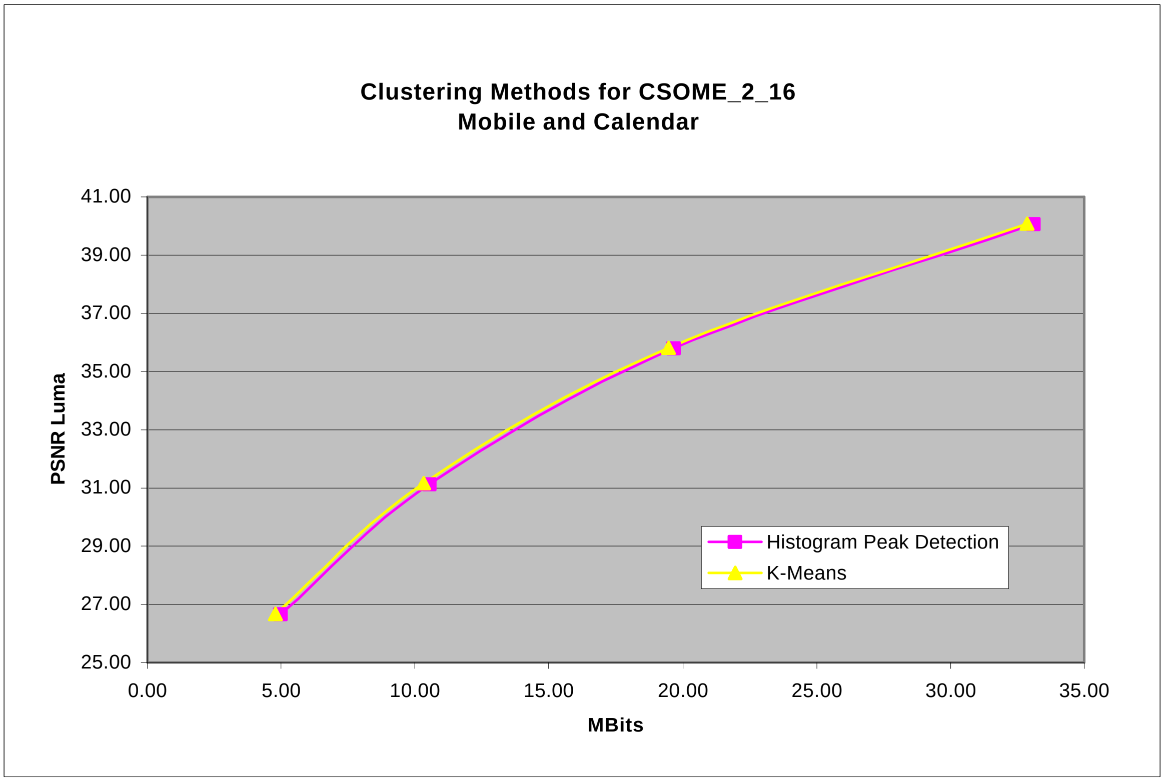


Figure 4-19 - Clustering Methods for Mobile and Calendar

CSOME_4 was slightly less effective (~1%) than CSOME_2 for the Bus sequence and overall. This is because for the sequences used and the search dimensions chosen, two search windows is adequate to capture most of the dominant characteristics of inter-frame motion. The exception is Mobile, where CSOME_4 outperformed CSOME_2 by 0.5%. There are several moving objects in Mobile, and these motion characteristics are accurately represented by four identified GMVs. However, while there are several moving objects in this sequence, the magnitude of the inter-frame motion is small. As a result, all tested algorithms are able to accurately capture the motion; none having a performance loss of more than 3.3% compared to JVT_128.

Figure 4-21 shows the relative performance of the group of motion estimation algorithms with the larger search areas. As for the first group, the best performing VLSI-suitable motion estimator is CSOME_2 with the k-means clustering algorithm. For the larger search areas, CSOME_1 gives only a 0.2% loss from CSOME_2. This is because the single large search window is able to capture as many of the optimal motion vectors as the two search windows. For sequences such as Bus and Canoa, where there are two distinct motion clusters, this implies that many motion vectors that are never chosen are also being tested for each block, which is wasted computation. Overall, the performance of CSOME_2_16 is only 1.6% better than the performance of CSOME_2_11, with more that twice the number of search positions. The CSOME_2_11 with the k-means clustering algorithm appears to be the best motion estimation algorithm, considering the compression performance and the computational expense of block-matching.

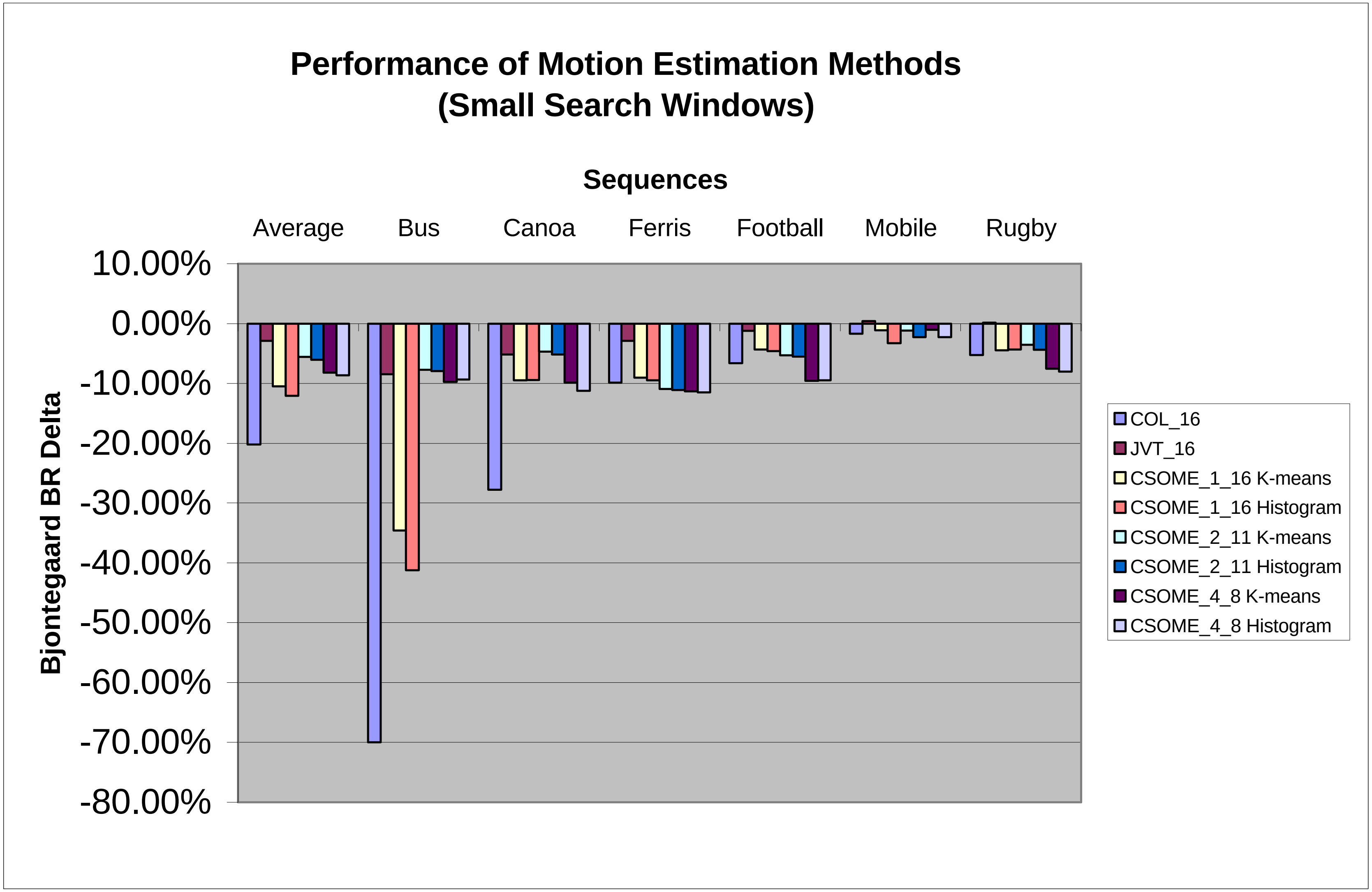


Figure 4-20 - Performance Comparison of Motion Estimation (Small Search Areas)

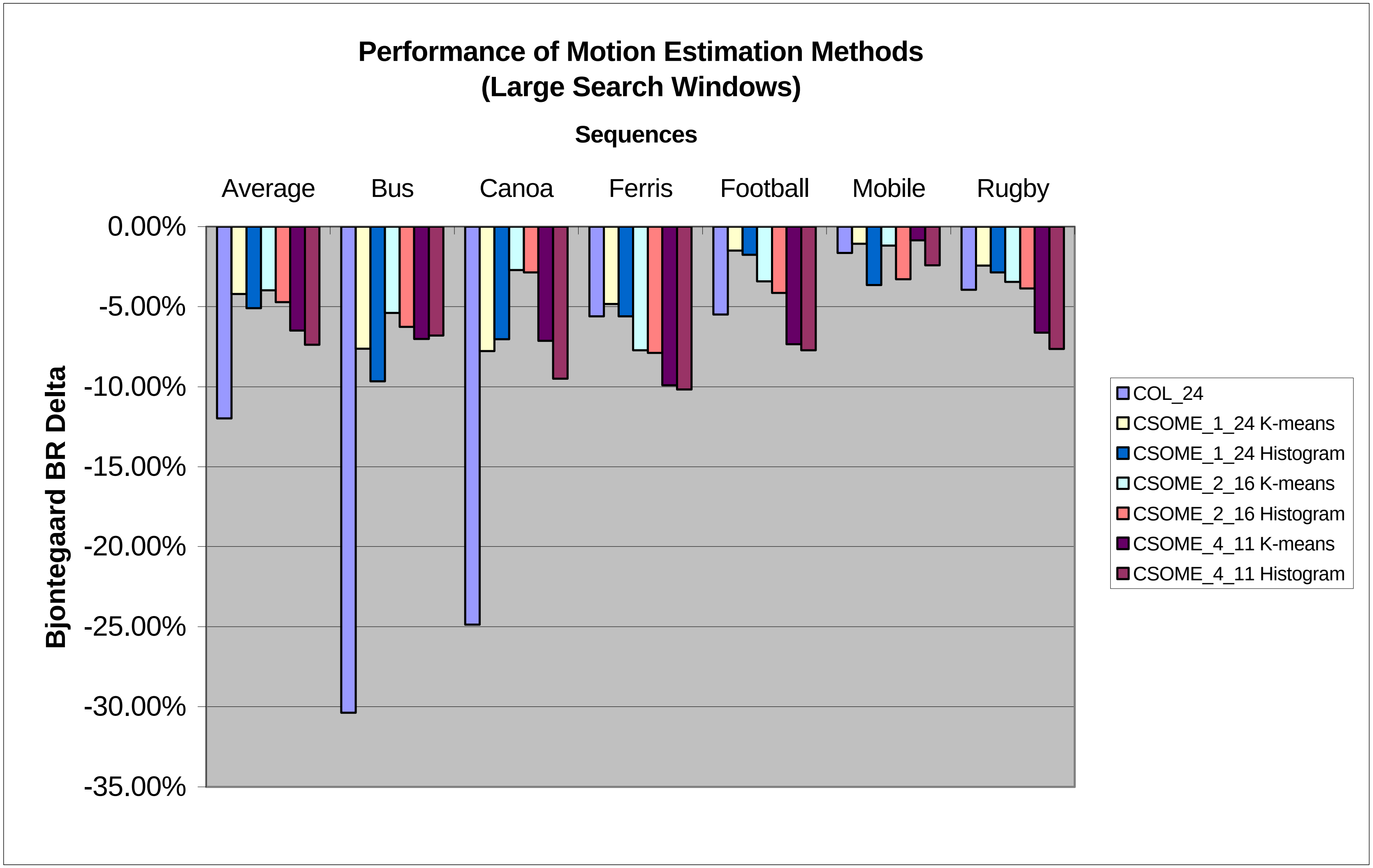


Figure 4-21 - Performance Comparison of Motion Estimation (Large Search Areas)

Chapter 5
Conclusion

Modern practical video encoders employ block-based motion compensated transform coding to achieve the necessary compression ratios needed for many applications. This approach uses several compression tools to exploit the various forms of information redundancy that exists in typical video sequences. The most significant tool is motion compensation, which reduces the temporal redundancy of video through a differential coding scheme using previously coded pixels as predictors for later frames. Estimation of the best motion parameters for block based motion estimation has been shown to require between 60 - 80% of the total computation resources in order to achieve adequate compression performance. Recent standardisation efforts have incorporated variable block-size motion compensation, which has further complicated the motion estimation process. Since motion estimation is such a computationally demanding process, for many applications, it is necessary to have a VLSI implementation achieve real-time video compression.

The efficiency of motion estimation is an important differentiating feature in commercial implementations of video encoders. As a result, much research has been undertaken into fast motion estimation algorithms. Many fast motion estimation algorithms have been proposed that are able to achieve adequate compression performance with few search block-matching positions. Many of these algorithms, however, require irregular memory accesses and execution paths, and so are ill-suited for implementation in a VLSI system.

The traditional full search block-matching algorithm lends itself well to efficient VLSI implementation, due to the independence of successive execution stages and the regularity of memory access. Additionally, since the block-matching search path is not data dependent, estimation for different block sizes is easily performed in parallel, further contributing to implementation efficiency. However, in order to achieve adequate motion estimation, a large search window is required. This large search window results in increased size, power consumption, and ultimately increased cost of encoder implementation.

Motion in video exhibits spatial correlation. Often, motion vectors for a frame are tightly clustered around a global motion component for the frame. In cases of more complex motion, several distinct clusters of motion vectors are evident. The motion estimation algorithm proposed in this thesis attempts to take advantage of this clustering of motion vectors. The proposed algorithm employs multiple full search estimation modules, placing each search window at a constant offset for each frame. The offsets chosen are based on estimates of the global motion components for the frame.

The motion characteristics of video are also temporally well correlated – the motion characteristics for consecutive frames are similar. This temporal correlation is exploited in the estimation of the global motion components for a frame. Specifically, the motion vectors for one frame are clustered into a fixed number of clusters. The prototype motion vector of each cluster is then used as a constant search offset for the next frame. Two clustering methods were used in the evaluation of this motion estimation algorithm: K-means clustering, and a histogram peak detection algorithm.

The constant search offset motion estimation (CSOME) algorithm has similar properties to the traditional full search algorithm that facilitates more efficient VLSI implementation than many other motion estimation algorithms. The implemental properties that affect the quality and cost of a VLSI implementation include execution time, memory size requirements and I/O bandwidth. It was shown that better coding performance was achieved using CSOME than the traditional full search algorithm, with the same time, memory and I/O bandwidth requirements. On average, using a single search window, a compression gain of more than 5% was achieved using a frame adaptive search offset over centring a full search on the collocated position. CSOME with a single search window provides more effective motion estimation with the same implemental quality as the traditional full search block matching.

For different specific video coding applications, different trade-offs between the implemental properties of the motion estimation are appropriate. For the proposed CSOME, increasing the number of search windows reduces the execution time of the motion estimation, while increasing the memory and bandwidth requirements. In terms of coding performance, using two search offsets resulted in improved results compared to the single search window CSOME with a similar total number of search position. CSOME with two search windows provided a 15% compression improvement over the traditional full search algorithms when approximately 600 total search positions were used. Expanding the algorithm to four search offsets resulted in an overall deterioration of performance, as compared to the single or two search offset CSOME. This implies that, for the video sequences evaluated, two global motion components are often sufficient to characterise the motion parameters of a video frame.

The selection of clustering method for estimation of the global motion component did not have a significant impact on the overall coding efficiency resulting from the motion estimation; however, the K-means clustering did consistently provide a slight (~1%) improvement over the histogram method. A simplification of the K-means clustering resulted in a negligible impact on compression performance, while reducing the complexity of the global motion component estimation.

5.1 Future Enhancements

There are several possible enhancements to the CSOME algorithm that may be worth investigating in the future. As it has been proposed, the number of search windows used for block-matching must be known before estimating global motion components. For some sequences, a CSOME using a single search window achieved greater compression performance over CSOME using two search windows. For other sequences, using two search windows provided gains over using a single search window. A possible enhancement to the CSOME method would be adaptive selection of the number of CSOME search windows to be used for block matching.

As proposed in this thesis, the estimates of the global motion components for a frame are the identified components of the previous frame. Estimates that are more accurate may be possible with consideration of a longer history of motion vectors – i.e. using the motion vectors identified in more previously encoded frames

Appendix A – Glossary of Terms

ITU-T: International Telecommunication Union Telecommunication Standardization Sector

MPEG: Motion Pictures Experts Group

JVT: Joint Video Team – The collaborative group from MPEG and ITU-T working on the H.264/MPEG-4 part 10 video compression standard

DVD: Digital Video Disc

MPEG-2: A widely used block-based motion compensated transform coding video compression standard developed by MPEG

4:2:0 Subsampling: Term for method of subsampling a three channel (Y,Cb,Cr) colour video signal. Each chrominance channel is subsampled by two in each spatial direction.

D1 Resolution: Refers to standard broadcast resolution of digital video signals. I.E. 720x480 at 30 frames per second or 720x576 at 25 frames per second

Appendix B – Full Experimental Results

Performance of COL_16 compared to JVT_128

			JVT_128					COL_16							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	31.31	40.50	41.73	43.32	466.11	-40.4%	-69.97%	-3.24
		25	11.70	36.97	38.75	40.96	10897.74	18.25	36.73	38.65	40.85	486.15	-56.0%		
		30	5.56	32.99	36.71	39.11	8901.60	9.53	32.74	36.48	38.89	450.61	-71.3%		
		35	2.67	29.40	35.40	37.39	7820.72	4.84	29.15	35.18	37.19	397.31	-81.3%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	40.35	41.59	43.32	43.71	865.96	-12.5%	-27.78%	-1.44
		25	21.09	38.09	40.91	41.34	22128.64	25.40	38.06	40.86	41.45	916.69	-20.4%		
		30	11.07	34.30	38.92	39.48	17839.20	14.76	34.26	38.87	39.57	791.44	-33.4%		
		35	6.02	30.77	36.97	37.45	15889.67	8.57	30.73	36.90	37.58	810.74	-42.3%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.91	16.46	42.23	42.16	42.68	268.40	-4.7%	-9.86%	-0.51
		25	9.08	38.86	38.50	39.14	6824.36	9.69	38.80	38.50	39.12	291.45	-6.6%		
		30	4.75	34.99	35.29	36.39	5574.78	5.22	34.91	35.26	36.32	235.91	-10.0%		
		35	2.43	31.31	32.77	34.61	4881.84	2.76	31.25	32.64	34.48	224.55	-13.5%		
Football	130	20	37.53	40.95	41.42	42.28	22921.70	36.91	41.00	41.53	42.37	831.85	1.7%	-6.63%	-0.29
		25	20.21	37.71	39.05	40.01	21491.59	20.83	37.79	39.20	40.13	846.84	-3.1%		
		30	10.06	34.27	37.04	38.26	17484.31	11.15	34.23	37.18	38.37	743.47	-10.9%		
		35	5.23	31.25	35.11	36.79	14986.00	5.95	31.13	35.31	36.86	665.81	-13.7%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.89	32.89	40.08	40.65	40.73	296.35	0.2%	-1.70%	-0.10
		25	19.42	35.83	36.61	36.67	11726.43	19.51	35.80	36.60	36.66	300.97	-0.4%		
		30	10.18	31.17	33.37	33.34	9982.05	10.36	31.14	33.37	33.32	281.38	-1.7%		
		35	4.64	26.69	30.76	30.58	8969.22	4.81	26.66	30.73	30.58	281.64	-3.7%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.16	48.63	40.99	42.51	43.36	846.34	1.7%	-5.22%	-0.27
		25	28.82	37.24	39.77	40.73	25192.61	29.32	37.28	39.86	40.83	854.78	-1.7%		
		30	14.60	33.41	37.69	38.87	20043.00	15.97	33.44	37.60	38.82	772.84	-9.4%		
		35	7.44	30.07	35.82	37.11	17910.82	8.59	30.07	35.67	37.06	720.38	-15.5%		

Performance of COL_24 compared to JVT_128

			JVT_128					COL_24							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	25.95	40.61	41.74	43.37	805.466	-16.4%	-30.37%	-1.53
		25	11.70	36.97	38.75	40.96	10897.736	14.43	36.85	38.72	40.96	864.456	-23.3%		
		30	5.56	32.99	36.71	39.11	8901.597	7.32	32.86	36.62	39.02	709.819	-31.5%		
		35	2.67	29.40	35.40	37.39	7820.718	3.69	29.22	35.29	37.34	644.424	-38.2%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	40.00	41.59	43.31	43.71	1564.572	-11.5%	-24.86%	-1.29
		25	21.09	38.09	40.91	41.34	22128.641	25.01	38.07	40.87	41.44	1567	-18.6%		
		30	11.07	34.30	38.92	39.48	17839.197	14.34	34.26	38.89	39.56	1350.892	-29.6%		
		35	6.02	30.77	36.97	37.45	15889.673	8.28	30.72	36.93	37.57	1225.442	-37.4%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.10	42.25	42.16	42.69	440.842	-2.3%	-5.61%	-0.29
		25	9.08	38.86	38.50	39.14	6824.357	9.41	38.82	38.49	39.11	456.921	-3.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.01	34.94	35.27	36.36	372.359	-5.6%		
		35	2.43	31.31	32.77	34.61	4881.838	2.63	31.27	32.70	34.53	349.385	-8.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.18	41.00	41.52	42.35	1453.629	0.9%	-5.50%	-0.24
		25	20.21	37.71	39.05	40.01	21491.589	20.80	37.78	39.18	40.11	1467.924	-2.9%		
		30	10.06	34.27	37.04	38.26	17484.307	10.94	34.24	37.17	38.36	1237.863	-8.8%		
		35	5.23	31.25	35.11	36.79	14986.002	5.81	31.15	35.29	36.88	1127.764	-10.9%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.92	40.07	40.65	40.72	527.953	0.1%	-1.66%	-0.10
		25	19.42	35.83	36.61	36.67	11726.425	19.52	35.81	36.60	36.66	586.436	-0.5%		
		30	10.18	31.17	33.37	33.34	9982.049	10.35	31.15	33.37	33.32	493.223	-1.6%		
		35	4.64	26.69	30.76	30.58	8969.222	4.81	26.66	30.73	30.59	486.78	-3.8%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.79	40.99	42.48	43.33	1488.853	1.4%	-3.95%	-0.21
		25	28.82	37.24	39.77	40.73	25192.607	29.21	37.28	39.84	40.84	1509.339	-1.3%		
		30	14.60	33.41	37.69	38.87	20042.996	15.65	33.43	37.63	38.85	1334.215	-7.2%		
		35	7.44	30.07	35.82	37.11	17910.821	8.33	30.06	35.70	37.09	1218.696	-11.9%		

Performance of JVT_16 compared to JVT_128

			JVT_128					JVT_16							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	23.03	40.68	41.76	43.41	363.85	-3.3%	-8.48%	-0.45
		25	11.70	36.97	38.75	40.96	10897.736	12.36	36.93	38.75	40.99	360.33	-5.7%		
		30	5.56	32.99	36.71	39.11	8901.597	6.07	32.95	36.68	39.05	334.55	-9.1%		
		35	2.67	29.40	35.40	37.39	7820.718	3.04	29.32	35.30	37.30	330.36	-13.9%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.74	41.62	43.37	43.75	743.66	0.4%	-5.19%	-0.26
		25	21.09	38.09	40.91	41.34	22128.641	21.47	38.09	40.95	41.50	730.35	-1.8%		
		30	11.07	34.30	38.92	39.48	17839.197	11.96	34.28	38.92	39.56	676.16	-8.1%		
		35	6.02	30.77	36.97	37.45	15889.673	6.67	30.75	36.92	37.56	642.14	-10.8%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	15.89	42.26	42.18	42.70	260.23	-1.0%	-2.89%	-0.15
		25	9.08	38.86	38.50	39.14	6824.357	9.23	38.83	38.51	39.12	254.82	-1.7%		
		30	4.75	34.99	35.29	36.39	5574.777	4.88	34.95	35.28	36.37	226.81	-2.7%		
		35	2.43	31.31	32.77	34.61	4881.838	2.53	31.28	32.72	34.56	216.82	-4.0%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.45	40.99	41.50	42.35	769.60	2.9%	-1.24%	-0.05
		25	20.21	37.71	39.05	40.01	21491.589	20.09	37.77	39.17	40.10	757.42	0.6%		
		30	10.06	34.27	37.04	38.26	17484.307	10.42	34.25	37.14	38.35	686.70	-3.6%		
		35	5.23	31.25	35.11	36.79	14986.002	5.50	31.16	35.28	36.89	639.76	-5.1%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.67	40.09	40.65	40.73	290.56	0.8%	0.39%	0.02
		25	19.42	35.83	36.61	36.67	11726.425	19.30	35.83	36.61	36.68	290.11	0.6%		
		30	10.18	31.17	33.37	33.34	9982.049	10.17	31.17	33.38	33.32	273.22	0.2%		
		35	4.64	26.69	30.76	30.58	8969.222	4.63	26.68	30.76	30.59	273.46	0.2%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	47.62	40.98	42.47	43.34	771.25	3.7%	0.16%	0.00
		25	28.82	37.24	39.77	40.73	25192.607	28.19	37.26	39.83	40.84	761.15	2.2%		
		30	14.60	33.41	37.69	38.87	20042.996	14.87	33.41	37.62	38.89	702.19	-1.9%		
		35	7.44	30.07	35.82	37.11	17910.821	7.84	30.05	35.69	37.09	672.05	-5.4%		

Performance of CSOME_1_16 with Histogram Peak Detection compared to JVT_128

			JVT_128					CSOME_1_16 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	28.16	40.59	41.75	43.35	506.161	-26.3%	-41.22%	-2.20
		25	11.70	36.97	38.75	40.96	10897.736	15.89	36.82	38.70	40.89	480.083	-35.8%		
		30	5.56	32.99	36.71	39.11	8901.597	7.60	32.83	36.59	39.00	446.141	-36.6%		
		35	2.67	29.40	35.40	37.39	7820.718	4.16	29.21	35.21	37.24	425.143	-55.9%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.75	41.59	43.39	43.78	934.144	0.4%	-9.43%	-0.50
		25	21.09	38.09	40.91	41.34	22128.641	21.84	38.06	40.95	41.50	905.25	-3.6%		
		30	11.07	34.30	38.92	39.48	17839.197	12.50	34.25	38.91	39.55	853.155	-12.9%		
		35	6.02	30.77	36.97	37.45	15889.673	7.29	30.74	36.89	37.52	816.323	-21.1%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.47	42.25	42.18	42.71	321.657	-4.7%	-9.48%	-0.48
		25	9.08	38.86	38.50	39.14	6824.357	9.69	38.82	38.52	39.16	304.716	-6.7%		
		30	4.75	34.99	35.29	36.39	5574.777	5.23	34.94	35.29	36.35	277.624	-10.2%		
		35	2.43	31.31	32.77	34.61	4881.838	2.76	31.28	32.68	34.49	266.745	-13.5%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.67	40.98	41.53	42.37	952.993	2.3%	-4.59%	-0.20
		25	20.21	37.71	39.05	40.01	21491.589	20.43	37.76	39.21	40.13	912.893	-1.1%		
		30	10.06	34.27	37.04	38.26	17484.307	10.84	34.23	37.18	38.36	836.092	-7.7%		
		35	5.23	31.25	35.11	36.79	14986.002	5.89	31.18	35.32	36.87	753.537	-12.5%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	33.08	40.06	40.65	40.72	418.397	-0.4%	-3.29%	-0.20
		25	19.42	35.83	36.61	36.67	11726.425	19.66	35.79	36.60	36.67	387.952	-1.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.51	31.12	33.35	33.31	392.798	-3.2%		
		35	4.64	26.69	30.76	30.58	8969.222	4.96	26.65	30.73	30.58	385.301	-7.0%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.33	40.99	42.52	43.36	935.906	2.3%	-4.32%	-0.23
		25	28.82	37.24	39.77	40.73	25192.607	29.08	37.29	39.85	40.85	905.093	-0.9%		
		30	14.60	33.41	37.69	38.87	20042.996	15.78	33.41	37.54	38.81	848.138	-8.1%		
		35	7.44	30.07	35.82	37.11	17910.821	8.44	30.05	35.58	37.03	798.268	-13.5%		

Performance of CSOME_1_16 with K-Means Clustering compared to JVT_128

			JVT_128					CSOME_1_16 with K-Means Clustering							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	26.94	40.60	41.75	43.38	433.514	-20.9%	-34.59%	-1.72
		25	11.70	36.97	38.75	40.96	10897.736	15.07	36.84	38.70	40.90	429.705	-28.8%		
		30	5.56	32.99	36.71	39.11	8901.597	7.46	32.84	36.61	38.99	401.136	-34.2%		
		35	2.67	29.40	35.40	37.39	7820.718	3.73	29.22	35.24	37.30	391.596	-39.8%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.70	41.59	43.40	43.79	844.189	0.5%	-9.47%	-0.50
		25	21.09	38.09	40.91	41.34	22128.641	21.82	38.06	40.96	41.50	839.284	-3.5%		
		30	11.07	34.30	38.92	39.48	17839.197	12.53	34.25	38.91	39.54	808.127	-13.1%		
		35	6.02	30.77	36.97	37.45	15889.673	7.29	30.73	36.83	37.51	777.756	-21.1%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.43	42.26	42.20	42.70	275.515	-4.5%	-9.01%	-0.46
		25	9.08	38.86	38.50	39.14	6824.357	9.67	38.83	38.51	39.14	269.765	-6.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.20	34.94	35.29	36.34	243.784	-9.6%		
		35	2.43	31.31	32.77	34.61	4881.838	2.75	31.28	32.67	34.51	232.45	-13.0%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.45	40.99	41.53	42.37	839.814	2.9%	-4.32%	-0.19
		25	20.21	37.71	39.05	40.01	21491.589	20.33	37.76	39.21	40.13	838.793	-0.6%		
		30	10.06	34.27	37.04	38.26	17484.307	10.84	34.23	37.17	38.38	762.91	-7.8%		
		35	5.23	31.25	35.11	36.79	14986.002	5.87	31.18	35.31	36.86	704.847	-12.2%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.82	40.08	40.65	40.73	291247	0.4%	-1.12%	-0.07
		25	19.42	35.83	36.61	36.67	11726.425	19.46	35.82	36.61	36.66	291240	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.31	31.15	33.35	33.32	275.046	-1.2%		
		35	4.64	26.69	30.76	30.58	8969.222	4.76	26.66	30.72	30.57	275347	-2.6%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.39	40.98	42.49	43.36	851.537	2.2%	-4.45%	-0.23
		25	28.82	37.24	39.77	40.73	25192.607	29.08	37.27	39.82	40.84	847.563	-0.9%		
		30	14.60	33.41	37.69	38.87	20042.996	15.80	33.42	37.55	38.82	796.349	-8.3%		
		35	7.44	30.07	35.82	37.11	17910.821	8.46	30.05	35.61	37.05	757.607	-13.7%		

Performance of CSOME_1_24 with Histogram Peak Detection compared to JVT_128

			JVT_128					CSOME_1_24 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.84	40.66	41.76	43.41	832.048	-2.5%	-9.67%	-0.51
		25	11.70	36.97	38.75	40.96	10897.736	12.33	36.91	38.75	40.98	806.873	-5.4%		
		30	5.56	32.99	36.71	39.11	8901.597	6.13	32.91	36.67	39.04	746.781	-10.2%		
		35	2.67	29.40	35.40	37.39	7820.718	3.09	29.26	35.31	37.30	715.252	-15.9%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.55	41.59	43.36	43.75	1659.482	0.9%	-7.03%	-0.37
		25	21.09	38.09	40.91	41.34	22128.641	21.52	38.06	40.94	41.50	1589.04	-2.0%		
		30	11.07	34.30	38.92	39.48	17839.197	12.15	34.25	38.88	39.56	1475.889	-9.7%		
		35	6.02	30.77	36.97	37.45	15889.673	7.01	30.72	36.86	37.51	1389.316	-16.4%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.11	42.26	42.19	42.71	531.673	-2.4%	-5.62%	-0.29
		25	9.08	38.86	38.50	39.14	6824.357	9.40	38.83	38.50	39.14	509.814	-3.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.03	34.95	35.30	36.37	458.858	-5.9%		
		35	2.43	31.31	32.77	34.61	4881.838	2.64	31.29	32.73	34.57	439.513	-8.6%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.25	40.96	41.51	42.36	1668.909	3.4%	-1.77%	-0.08
		25	20.21	37.71	39.05	40.01	21491.589	19.90	37.74	39.19	40.13	1611.809	1.5%		
		30	10.06	34.27	37.04	38.26	17484.307	10.46	34.22	37.15	38.36	1456.264	-4.0%		
		35	5.23	31.25	35.11	36.79	14986.002	5.70	31.20	35.29	36.84	1343.683	-8.9%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	33.18	40.07	40.65	40.72	736.702	-0.7%	-3.66%	-0.22
		25	19.42	35.83	36.61	36.67	11726.425	19.69	35.79	36.61	36.66	705.184	-1.4%		
		30	10.18	31.17	33.37	33.34	9982.049	10.55	31.12	33.36	33.32	697.081	-3.6%		
		35	4.64	26.69	30.76	30.58	8969.222	5.00	26.64	30.71	30.58	702.202	-7.7%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.42	40.97	42.45	43.32	1660.475	2.1%	-2.87%	-0.15
		25	28.82	37.24	39.77	40.73	25192.607	28.80	37.26	39.79	40.85	1618.151	0.1%		
		30	14.60	33.41	37.69	38.87	20042.996	15.40	33.41	37.56	38.85	1494.074	-5.5%		
		35	7.44	30.07	35.82	37.11	17910.821	8.17	30.03	35.61	37.06	1388.217	-9.8%		

Performance of CSOME_1_24 with K-Means Clustering compared to JVT_128

			JVT_128					CSOME_1_24 with K-Means Clustering							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.75	40.67	41.77	43.42	715.266	-2.0%	-7.64%	-0.40
		25	11.70	36.97	38.75	40.96	10897.736	12.17	36.93	38.76	41.00	706.781	-4.0%		
		30	5.56	32.99	36.71	39.11	8901.597	6.03	32.93	36.69	39.05	676.183	-8.5%		
		35	2.67	29.40	35.40	37.39	7820.718	3.00	29.28	35.34	37.34	627.783	-12.4%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.99	41.58	43.35	43.74	1471.656	-0.3%	-7.78%	-0.40
		25	21.09	38.09	40.91	41.34	22128.641	21.82	38.06	40.93	41.48	1459.033	-3.4%		
		30	11.07	34.30	38.92	39.48	17839.197	12.23	34.25	38.91	39.54	1349.484	-10.5%		
		35	6.02	30.77	36.97	37.45	15889.673	6.90	30.72	36.84	37.50	1287.335	-14.5%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.06	42.28	42.19	42.70	447.954	-2.1%	-4.84%	-0.25
		25	9.08	38.86	38.50	39.14	6824.357	9.38	38.84	38.51	39.14	438.111	-3.3%		
		30	4.75	34.99	35.29	36.39	5574.777	5.00	34.96	35.29	36.36	390.814	-5.3%		
		35	2.43	31.31	32.77	34.61	4881.838	2.62	31.29	32.73	34.57	368.358	-7.7%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.15	40.96	41.51	42.36	1488.443	3.7%	-1.52%	-0.07
		25	20.21	37.71	39.05	40.01	21491.589	19.83	37.73	39.19	40.13	1466.439	1.9%		
		30	10.06	34.27	37.04	38.26	17484.307	10.41	34.22	37.16	38.38	1331.482	-3.6%		
		35	5.23	31.25	35.11	36.79	14986.002	5.70	31.20	35.30	36.88	1188.208	-8.8%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.85	40.08	40.64	40.72	519.745	0.3%	-1.07%	-0.06
		25	19.42	35.83	36.61	36.67	11726.425	19.47	35.82	36.61	36.67	517.389	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.30	31.15	33.37	33.33	479.309	-1.2%		
		35	4.64	26.69	30.76	30.58	8969.222	4.75	26.67	30.75	30.59	469.832	-2.5%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.39	40.97	42.46	43.32	1511.314	2.2%	-2.43%	-0.13
		25	28.82	37.24	39.77	40.73	25192.607	28.74	37.25	39.80	40.83	1501.419	0.3%		
		30	14.60	33.41	37.69	38.87	20042.996	15.31	33.40	37.56	38.84	1384.153	-4.9%		
		35	7.44	30.07	35.82	37.11	17910.821	8.07	30.04	35.63	37.07	1300.49	-8.5%		

Performance of CSOME_2_11 with Histogram Peak Detection compared to JVT_128

			JVT_128					CSOME_2_11 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.70	40.68	41.75	43.41	393.08	-1.8%	-7.95%	-0.41
		25	11.70	36.97	38.75	40.96	10897.736	12.21	36.94	38.75	41.00	429.72	-4.4%		
		30	5.56	32.99	36.71	39.11	8901.597	6.05	32.92	36.69	39.07	407.80	-8.7%		
		35	2.67	29.40	35.40	37.39	7820.718	3.03	29.27	35.36	37.35	399.08	-13.4%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.45	41.60	43.38	43.76	805.78	1.2%	-5.20%	-0.28
		25	21.09	38.09	40.91	41.34	22128.641	21.56	38.07	40.94	41.48	851.33	-2.2%		
		30	11.07	34.30	38.92	39.48	17839.197	11.81	34.27	38.92	39.54	798.33	-6.7%		
		35	6.02	30.77	36.97	37.45	15889.673	6.78	30.74	36.92	37.51	774.16	-12.6%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.52	42.25	42.16	42.68	277.78	-5.0%	-11.11%	-0.57
		25	9.08	38.86	38.50	39.14	6824.357	9.82	38.83	38.51	39.14	298.01	-8.0%		
		30	4.75	34.99	35.29	36.39	5574.777	5.32	34.94	35.28	36.35	274.64	-12.0%		
		35	2.43	31.31	32.77	34.61	4881.838	2.84	31.29	32.67	34.49	263.26	-16.9%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.63	40.98	41.52	42.36	818.24	2.4%	-5.53%	-0.25
		25	20.21	37.71	39.05	40.01	21491.589	20.48	37.76	39.20	40.12	865.56	-1.3%		
		30	10.06	34.27	37.04	38.26	17484.307	10.92	34.21	37.15	38.35	806.72	-8.6%		
		35	5.23	31.25	35.11	36.79	14986.002	6.06	31.18	35.28	36.84	745.00	-15.7%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.88	40.08	40.65	40.73	389.59	0.2%	-2.29%	-0.14
		25	19.42	35.83	36.61	36.67	11726.425	19.56	35.81	36.62	36.66	335.78	-0.7%		
		30	10.18	31.17	33.37	33.34	9982.049	10.43	31.14	33.36	33.32	330.82	-2.5%		
		35	4.64	26.69	30.76	30.58	8969.222	4.89	26.65	30.73	30.57	340.17	-5.4%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.19	40.98	42.50	43.36	864.36	2.6%	-4.37%	-0.23
		25	28.82	37.24	39.77	40.73	25192.607	28.96	37.27	39.84	40.85	883.00	-0.5%		
		30	14.60	33.41	37.69	38.87	20042.996	15.87	33.42	37.60	38.84	834.58	-8.7%		
		35	7.44	30.07	35.82	37.11	17910.821	8.47	30.06	35.65	37.07	788.06	-13.8%		

Performance of CSOME_2_11 with K-Means Clustering compared to JVT_128

			JVT_128					CSOME_2_11 with K-Means Clustering							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.70	40.68	41.75	43.41	393.079	-1.8%	-7.65%	-0.40
		25	11.70	36.97	38.75	40.96	10897.736	12.18	36.94	38.75	41.00	392.643	-4.1%		
		30	5.56	32.99	36.71	39.11	8901.597	6.04	32.92	36.69	39.05	414.657	-8.6%		
		35	2.67	29.40	35.40	37.39	7820.718	2.99	29.29	35.34	37.35	372.172	-12.2%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.45	41.60	43.38	43.76	805.779	1.2%	-5.84%	-0.30
		25	21.09	38.09	40.91	41.34	22128.641	21.97	38.07	40.93	41.47	809.528	-4.2%		
		30	11.07	34.30	38.92	39.48	17839.197	11.83	34.27	38.93	39.54	775.324	-6.8%		
		35	6.02	30.77	36.97	37.45	15889.673	6.69	30.73	36.92	37.54	745.163	-11.1%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.52	42.25	42.16	42.68	277.782	-5.0%	-10.94%	-0.56
		25	9.08	38.86	38.50	39.14	6824.357	9.77	38.82	38.50	39.14	274.029	-7.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.29	34.91	35.27	36.35	281.453	-11.4%		
		35	2.43	31.31	32.77	34.61	4881.838	2.84	31.26	32.64	34.47	240.982	-16.5%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.63	40.98	41.52	42.36	818.242	2.4%	-5.36%	-0.24
		25	20.21	37.71	39.05	40.01	21491.589	20.49	37.76	39.19	40.11	857.564	-1.3%		
		30	10.06	34.27	37.04	38.26	17484.307	10.90	34.22	37.15	38.35	778.993	-8.4%		
		35	5.23	31.25	35.11	36.79	14986.002	6.01	31.16	35.29	36.83	696.02	-14.8%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.88	40.08	40.65	40.73	389.593	0.2%	-1.23%	-0.07
		25	19.42	35.83	36.61	36.67	11726.425	19.49	35.82	36.62	36.66	353.591	-0.4%		
		30	10.18	31.17	33.37	33.34	9982.049	10.32	31.15	33.37	33.33	274.867	-1.3%		
		35	4.64	26.69	30.76	30.58	8969.222	4.77	26.67	30.76	30.58	273.408	-2.9%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.19	40.98	42.50	43.36	864.358	2.6%	-3.46%	-0.19
		25	28.82	37.24	39.77	40.73	25192.607	28.87	37.28	39.85	40.86	865.374	-0.2%		
		30	14.60	33.41	37.69	38.87	20042.996	15.62	33.43	37.61	38.85	820.569	-7.0%		
		35	7.44	30.07	35.82	37.11	17910.821	8.43	30.06	35.67	37.08	764.971	-13.3%		

Performance of CSOME_2_11 with Single Iteration K-Means Clustering compared to JVT_128

			JVT_128					CSOME_2_11 with Single Iteration K-Means Clustering							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.74	40.68	41.76	43.41	407.268	-2.0%	-7.73%	-0.40
		25	11.70	36.97	38.75	40.96	10897.736	12.21	36.94	38.75	40.99	423.626	-4.4%		
		30	5.56	32.99	36.71	39.11	8901.597	6.04	32.93	36.69	39.05	412.627	-8.6%		
		35	2.67	29.40	35.40	37.39	7820.718	2.99	29.29	35.34	37.35	373.077	-12.2%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.43	41.60	43.38	43.76	804.67	1.3%	-4.68%	-0.24
		25	21.09	38.09	40.91	41.34	22128.641	21.39	38.08	40.95	41.49	829.251	-1.4%		
		30	11.07	34.30	38.92	39.48	17839.197	11.81	34.27	38.93	39.54	755.663	-6.7%		
		35	6.02	30.77	36.97	37.45	15889.673	6.69	30.73	36.92	37.54	746.308	-11.1%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.53	42.25	42.16	42.69	296.314	-5.1%	-10.91%	-0.56
		25	9.08	38.86	38.50	39.14	6824.357	9.77	38.82	38.51	39.13	293.766	-7.6%		
		30	4.75	34.99	35.29	36.39	5574.777	5.29	34.92	35.26	36.35	272.905	-11.4%		
		35	2.43	31.31	32.77	34.61	4881.838	2.84	31.27	32.67	34.49	255.034	-16.7%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.68	40.98	41.52	42.36	817.973	2.3%	-5.31%	-0.24
		25	20.21	37.71	39.05	40.01	21491.589	20.47	37.76	39.20	40.12	811.685	-1.3%		
		30	10.06	34.27	37.04	38.26	17484.307	10.89	34.22	37.15	38.36	765.765	-8.3%		
		35	5.23	31.25	35.11	36.79	14986.002	6.01	31.16	35.29	36.82	696.923	-14.8%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.88	40.09	40.64	40.72	380.905	0.2%	-1.19%	-0.07
		25	19.42	35.83	36.61	36.67	11726.425	19.49	35.82	36.61	36.67	353.656	-0.4%		
		30	10.18	31.17	33.37	33.34	9982.049	10.32	31.16	33.39	33.32	268.005	-1.3%		
		35	4.64	26.69	30.76	30.58	8969.222	4.77	26.66	30.74	30.58	274.645	-2.8%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.17	40.98	42.50	43.35	873.578	2.6%	-3.51%	-0.19
		25	28.82	37.24	39.77	40.73	25192.607	28.88	37.28	39.85	40.85	866.062	-0.2%		
		30	14.60	33.41	37.69	38.87	20042.996	15.62	33.43	37.62	38.85	795.999	-7.0%		
		35	7.44	30.07	35.82	37.11	17910.821	8.45	30.06	35.67	37.07	759.182	-13.5%		

Performance of CSOME_2_16 with Histogram Peak Detection compared to JVT_128

			JVT_128					CSOME_2_16 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.53	40.67	41.77	43.42	759.52	-1.1%	-6.27%	-0.33
		25	11.70	36.97	38.75	40.96	10897.736	12.03	36.94	38.76	41.02	742.015	-2.8%		
		30	5.56	32.99	36.71	39.11	8901.597	5.96	32.93	36.70	39.09	706.76	-7.1%		
		35	2.67	29.40	35.40	37.39	7820.718	2.98	29.30	35.35	37.37	681.039	-11.5%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.13	41.60	43.37	43.74	1457.564	2.1%	-2.87%	-0.15
		25	21.09	38.09	40.91	41.34	22128.641	21.03	38.09	40.96	41.47	1425.22	0.3%		
		30	11.07	34.30	38.92	39.48	17839.197	11.59	34.27	38.94	39.54	1347.488	-4.6%		
		35	6.02	30.77	36.97	37.45	15889.673	6.56	30.70	36.93	37.53	1284.049	-8.9%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.33	42.26	42.19	42.70	506.646	-3.8%	-7.89%	-0.41
		25	9.08	38.86	38.50	39.14	6824.357	9.62	38.83	38.50	39.13	505.045	-5.9%		
		30	4.75	34.99	35.29	36.39	5574.777	5.14	34.96	35.29	36.37	443.835	-8.2%		
		35	2.43	31.31	32.77	34.61	4881.838	2.73	31.29	32.75	34.56	425.451	-12.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.99	40.98	41.51	42.35	1552.306	1.4%	-4.15%	-0.19
		25	20.21	37.71	39.05	40.01	21491.589	20.30	37.74	39.17	40.11	1510.574	-0.5%		
		30	10.06	34.27	37.04	38.26	17484.307	10.71	34.22	37.14	38.36	1375.032	-6.5%		
		35	5.23	31.25	35.11	36.79	14986.002	5.87	31.19	35.29	36.88	1273.266	-12.2%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	33.11	40.07	40.64	40.73	663.081	-0.5%	-3.30%	-0.20
		25	19.42	35.83	36.61	36.67	11726.425	19.65	35.79	36.61	36.66	629.528	-1.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.53	31.13	33.36	33.32	622.842	-3.4%		
		35	4.64	26.69	30.76	30.58	8969.222	4.97	26.65	30.72	30.57	631.97	-7.1%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.85	40.98	42.47	43.33	1546.43	1.2%	-3.87%	-0.20
		25	28.82	37.24	39.77	40.73	25192.607	29.12	37.26	39.83	40.84	1508.875	-1.0%		
		30	14.60	33.41	37.69	38.87	20042.996	15.55	33.42	37.61	38.86	1407.307	-6.5%		
		35	7.44	30.07	35.82	37.11	17910.821	8.35	30.05	35.66	37.09	1320.943	-12.2%		

Performance of CSOME_2_16 with K-Means Clustering compared to JVT_128

			JVT_128					CSOME_2_16 with K-Means Clustering							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.46	40.68	41.76	43.41	651.59	-0.7%	-5.32%	-0.28
		25	11.70	36.97	38.75	40.96	10897.736	11.97	36.94	38.75	41.01	648.984	-2.3%		
		30	5.56	32.99	36.71	39.11	8901.597	5.92	32.94	36.71	39.07	677.561	-6.5%		
		35	2.67	29.40	35.40	37.39	7820.718	2.91	29.30	35.36	37.39	594.517	-9.0%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.28	41.60	43.36	43.74	1341.125	1.7%	-2.74%	-0.14
		25	21.09	38.09	40.91	41.34	22128.641	21.11	38.08	40.96	41.47	1328.265	-0.1%		
		30	11.07	34.30	38.92	39.48	17839.197	11.53	34.27	38.93	39.55	1249.61	-4.1%		
		35	6.02	30.77	36.97	37.45	15889.673	6.50	30.72	36.95	37.52	1198.898	-8.0%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.29	42.26	42.18	42.69	507.578	-3.6%	-7.57%	-0.39
		25	9.08	38.86	38.50	39.14	6824.357	9.54	38.84	38.48	39.12	456.077	-5.0%		
		30	4.75	34.99	35.29	36.39	5574.777	5.14	34.93	35.27	36.35	409.722	-8.3%		
		35	2.43	31.31	32.77	34.61	4881.838	2.71	31.30	32.71	34.55	391.47	-11.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.58	40.96	41.50	42.35	1414.273	2.5%	-3.46%	-0.16
		25	20.21	37.71	39.05	40.01	21491.589	20.23	37.73	39.16	40.10	1506.77	-0.1%		
		30	10.06	34.27	37.04	38.26	17484.307	10.61	34.23	37.14	38.36	1256.424	-5.5%		
		35	5.23	31.25	35.11	36.79	14986.002	5.86	31.19	35.27	36.84	1243.587	-11.9%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.87	40.09	40.64	40.72	643.547	0.2%	-1.20%	-0.07
		25	19.42	35.83	36.61	36.67	11726.425	19.47	35.82	36.61	36.66	470.345	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.33	31.16	33.37	33.32	444.378	-1.4%		
		35	4.64	26.69	30.76	30.58	8969.222	4.79	26.67	30.76	30.58	465.582	-3.2%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.64	40.98	42.47	43.33	1472.025	1.7%	-3.52%	-0.19
		25	28.82	37.24	39.77	40.73	25192.607	29.06	37.27	39.83	40.83	1543.201	-0.8%		
		30	14.60	33.41	37.69	38.87	20042.996	15.50	33.42	37.62	38.86	1387.55	-6.2%		
		35	7.44	30.07	35.82	37.11	17910.821	8.30	30.04	35.69	37.09	1261.772	-11.6%		

Performance of CSOME_2_16 with Single Iteration K-Means Clustering compared to JVT_128

			JVT_128					CSOME_2_16 with Single Iteration K-Means Clustering							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.45	40.68	41.76	43.42	666.844	-0.7%	-5.40%	-0.28
		25	11.70	36.97	38.75	40.96	10897.736	11.99	36.95	38.76	41.03	680.937	-2.5%		
		30	5.56	32.99	36.71	39.11	8901.597	5.92	32.95	36.72	39.08	674.358	-6.5%		
		35	2.67	29.40	35.40	37.39	7820.718	2.92	29.29	35.36	37.37	599.53	-9.3%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.15	41.60	43.37	43.75	1371.666	2.0%	-2.72%	-0.14
		25	21.09	38.09	40.91	41.34	22128.641	21.12	38.08	40.96	41.47	1327.923	-0.1%		
		30	11.07	34.30	38.92	39.48	17839.197	11.54	34.28	38.93	39.55	1249.76	-4.2%		
		35	6.02	30.77	36.97	37.45	15889.673	6.50	30.73	36.93	37.53	1200.798	-8.0%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.28	42.26	42.19	42.69	471.81	-3.5%	-7.72%	-0.39
		25	9.08	38.86	38.50	39.14	6824.357	9.57	38.83	38.47	39.11	493.982	-5.4%		
		30	4.75	34.99	35.29	36.39	5574.777	5.14	34.94	35.28	36.35	411.184	-8.4%		
		35	2.43	31.31	32.77	34.61	4881.838	2.71	31.29	32.73	34.54	391.987	-11.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	36.59	40.96	41.50	42.35	1401.028	2.5%	-3.42%	-0.15
		25	20.21	37.71	39.05	40.01	21491.589	20.18	37.73	39.16	40.10	1450.557	0.2%		
		30	10.06	34.27	37.04	38.26	17484.307	10.62	34.23	37.14	38.35	1323.697	-5.5%		
		35	5.23	31.25	35.11	36.79	14986.002	5.86	31.20	35.29	36.86	1218.999	-12.0%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.83	40.09	40.65	40.73	478.92	0.3%	-1.19%	-0.07
		25	19.42	35.83	36.61	36.67	11726.425	19.47	35.82	36.61	36.67	471.809	-0.3%		
		30	10.18	31.17	33.37	33.34	9982.049	10.32	31.16	33.37	33.32	445.309	-1.3%		
		35	4.64	26.69	30.76	30.58	8969.222	4.80	26.68	30.73	30.58	571.169	-3.6%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	48.62	40.98	42.47	43.33	1470.627	1.7%	-3.46%	-0.18
		25	28.82	37.24	39.77	40.73	25192.607	29.01	37.27	39.83	40.83	1458.401	-0.6%		
		30	14.60	33.41	37.69	38.87	20042.996	15.53	33.42	37.62	38.86	1382.985	-6.4%		
		35	7.44	30.07	35.82	37.11	17910.821	8.31	30.06	35.70	37.11	1284.853	-11.6%		

Performance of CSOME_4_8 with Histogram Peak Detection compared to JVT_128

			JVT_128					CSOME_4_8 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.88	40.67	41.74	43.40	535.359	-2.6%	-9.34%	-0.49
		25	11.70	36.97	38.75	40.96	10897.736	12.31	36.93	38.75	41.01	523.185	-5.3%		
		30	5.56	32.99	36.71	39.11	8901.597	6.13	32.93	36.69	39.06	506.38	-10.2%		
		35	2.67	29.40	35.40	37.39	7820.718	3.09	29.27	35.37	37.44	501.879	-15.9%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	38.11	41.58	43.33	43.71	1074.066	-6.2%	-11.23%	-0.60
		25	21.09	38.09	40.91	41.34	22128.641	22.65	38.06	40.92	41.45	1033.597	-7.4%		
		30	11.07	34.30	38.92	39.48	17839.197	12.42	34.26	38.91	39.52	976.05	-12.2%		
		35	6.02	30.77	36.97	37.45	15889.673	7.19	30.70	36.90	37.48	978.35	-19.4%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.56	42.27	42.18	42.70	391.355	-5.2%	-11.52%	-0.60
		25	9.08	38.86	38.50	39.14	6824.357	9.86	38.84	38.50	39.13	376.968	-8.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.35	34.96	35.28	36.36	349.767	-12.6%		
		35	2.43	31.31	32.77	34.61	4881.838	2.89	31.30	32.72	34.57	354.112	-18.8%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.80	40.98	41.51	42.35	1120.556	-0.7%	-9.49%	-0.43
		25	20.21	37.71	39.05	40.01	21491.589	21.21	37.76	39.17	40.10	1072.806	-4.9%		
		30	10.06	34.27	37.04	38.26	17484.307	11.39	34.22	37.13	38.35	1011.831	-13.3%		
		35	5.23	31.25	35.11	36.79	14986.002	6.34	31.21	35.24	36.84	1108.07	-21.2%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.92	40.08	40.64	40.72	457.971	0.1%	-2.27%	-0.14
		25	19.42	35.83	36.61	36.67	11726.425	19.57	35.81	36.62	36.67	428.569	-0.8%		
		30	10.18	31.17	33.37	33.34	9982.049	10.45	31.15	33.38	33.32	420.829	-2.6%		
		35	4.64	26.69	30.76	30.58	8969.222	4.88	26.67	30.74	30.57	471.678	-5.3%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.33	40.97	42.47	43.32	1126.251	0.2%	-8.02%	-0.42
		25	28.82	37.24	39.77	40.73	25192.607	29.85	37.26	39.82	40.83	1084.899	-3.6%		
		30	14.60	33.41	37.69	38.87	20042.996	16.31	33.41	37.61	38.83	1041.811	-11.7%		
		35	7.44	30.07	35.82	37.11	17910.821	9.00	30.04	35.67	37.08	1076.539	-21.0%		

Performance of CSOME_4_8 with K-Means Clustering compared to JVT_128

			JVT_128					CSOME_4_8 with K-Means Clustering							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.98	40.6781	41.7404	43.4031	566.14	-3.1%	-9.61%	-0.50
		25	11.70	36.97	38.75	40.96	10897.736	12.36	36.9297	38.7475	40.9964	550.691	-5.7%		
		30	5.56	32.99	36.71	39.11	8901.597	6.16	32.9362	36.691	39.0342	495.619	-10.8%		
		35	2.67	29.40	35.40	37.39	7820.718	3.09	29.2865	35.3493	37.3646	498.463	-15.6%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	36.50	41.5864	43.3498	43.7307	988.462	-1.7%	-10.36%	-0.54
		25	21.09	38.09	40.91	41.34	22128.641	22.47	38.0567	40.9298	41.4472	1010.719	-6.5%		
		30	11.07	34.30	38.92	39.48	17839.197	12.48	34.2655	38.9127	39.5214	995.66	-12.7%		
		35	6.02	30.77	36.97	37.45	15889.673	7.12	30.7218	36.9554	37.5166	933.949	-18.3%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.56	42.2579	42.1493	42.6643	336.484	-5.3%	-11.53%	-0.59
		25	9.08	38.86	38.50	39.14	6824.357	9.82	38.83	38.4739	39.0979	381.765	-8.1%		
		30	4.75	34.99	35.29	36.39	5574.777	5.34	34.9324	35.2551	36.3432	347.459	-12.6%		
		35	2.43	31.31	32.77	34.61	4881.838	2.86	31.2685	32.6823	34.5113	292.191	-17.6%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.70	40.981	41.5141	42.3589	1023.607	-0.4%	-9.46%	-0.42
		25	20.21	37.71	39.05	40.01	21491.589	21.18	37.7633	39.171	40.101	1066.653	-4.8%		
		30	10.06	34.27	37.04	38.26	17484.307	11.37	34.2146	37.1322	38.3357	946.557	-13.0%		
		35	5.23	31.25	35.11	36.79	14986.002	6.33	31.1788	35.275	36.8123	861.575	-20.9%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.81	40.0938	40.6466	40.7288	431.306	0.4%	-1.06%	-0.06
		25	19.42	35.83	36.61	36.67	11726.425	19.49	35.8294	36.6244	36.6709	459.373	-0.4%		
		30	10.18	31.17	33.37	33.34	9982.049	10.33	31.1721	33.3794	33.3375	384.535	-1.5%		
		35	4.64	26.69	30.76	30.58	8969.222	4.78	26.6818	30.7478	30.5809	367.46	-3.0%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.31	40.9791	42.4762	43.3353	1066.125	0.3%	-6.98%	-0.36
		25	28.82	37.24	39.77	40.73	25192.607	29.76	37.2621	39.8298	40.8332	1121.139	-3.2%		
		30	14.60	33.41	37.69	38.87	20042.996	16.10	33.406	37.6223	38.8442	988.941	-10.3%		
		35	7.44	30.07	35.82	37.11	17910.821	8.79	30.0576	35.7031	37.0805	965.398	-18.1%		

Performance of CSOME_4_8 with Single Iteration K-Means Clustering compared to JVT_128

			JVT_128					CSOME_4_8 with Single Iteration K-Means Clustering							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	23.00	40.67	41.74	43.40	609.122	-3.2%	-9.76%	-0.51
		25	11.70	36.97	38.75	40.96	10897.736	12.37	36.94	38.74	40.98	478.454	-5.7%		
		30	5.56	32.99	36.71	39.11	8901.597	6.16	32.93	36.70	39.07	551.415	-10.7%		
		35	2.67	29.40	35.40	37.39	7820.718	3.12	29.29	35.36	37.35	487.528	-16.7%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	36.60	41.59	43.35	43.73	991.708	-2.0%	-9.85%	-0.52
		25	21.09	38.09	40.91	41.34	22128.641	22.40	38.07	40.93	41.46	1037.718	-6.2%		
		30	11.07	34.30	38.92	39.48	17839.197	12.38	34.27	38.92	39.53	948.902	-11.8%		
		35	6.02	30.77	36.97	37.45	15889.673	7.13	30.72	36.95	37.52	896.043	-18.4%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.58	42.26	42.15	42.67	349.033	-5.4%	-11.32%	-0.59
		25	9.08	38.86	38.50	39.14	6824.357	9.76	38.82	38.48	39.11	329.744	-7.4%		
		30	4.75	34.99	35.29	36.39	5574.777	5.33	34.93	35.27	36.35	347.547	-12.3%		
		35	2.43	31.31	32.77	34.61	4881.838	2.88	31.27	32.67	34.50	301.687	-18.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.81	40.99	41.52	42.36	1062.781	-0.7%	-9.59%	-0.44
		25	20.21	37.71	39.05	40.01	21491.589	21.22	37.76	39.18	40.09	1023.84	-5.0%		
		30	10.06	34.27	37.04	38.26	17484.307	11.36	34.22	37.15	38.35	946.672	-12.9%		
		35	5.23	31.25	35.11	36.79	14986.002	6.35	31.16	35.24	36.84	887.634	-21.4%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.75	40.09	40.66	40.73	349.713	0.6%	-1.01%	-0.06
		25	19.42	35.83	36.61	36.67	11726.425	19.45	35.83	36.62	36.66	378.966	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.35	31.17	33.36	33.33	394.624	-1.7%		
		35	4.64	26.69	30.76	30.58	8969.222	4.77	26.69	30.75	30.60	345.436	-2.8%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.54	40.98	42.47	43.33	1071.627	-0.2%	-7.56%	-0.39
		25	28.82	37.24	39.77	40.73	25192.607	29.70	37.27	39.82	40.83	1029.373	-3.0%		
		30	14.60	33.41	37.69	38.87	20042.996	16.30	33.41	37.61	38.83	1027.942	-11.6%		
		35	7.44	30.07	35.82	37.11	17910.821	8.89	30.05	35.71	37.06	949.366	-19.5%		

Performance of CSOME_4_11 with Histogram Peak Detection compared to JVT_128

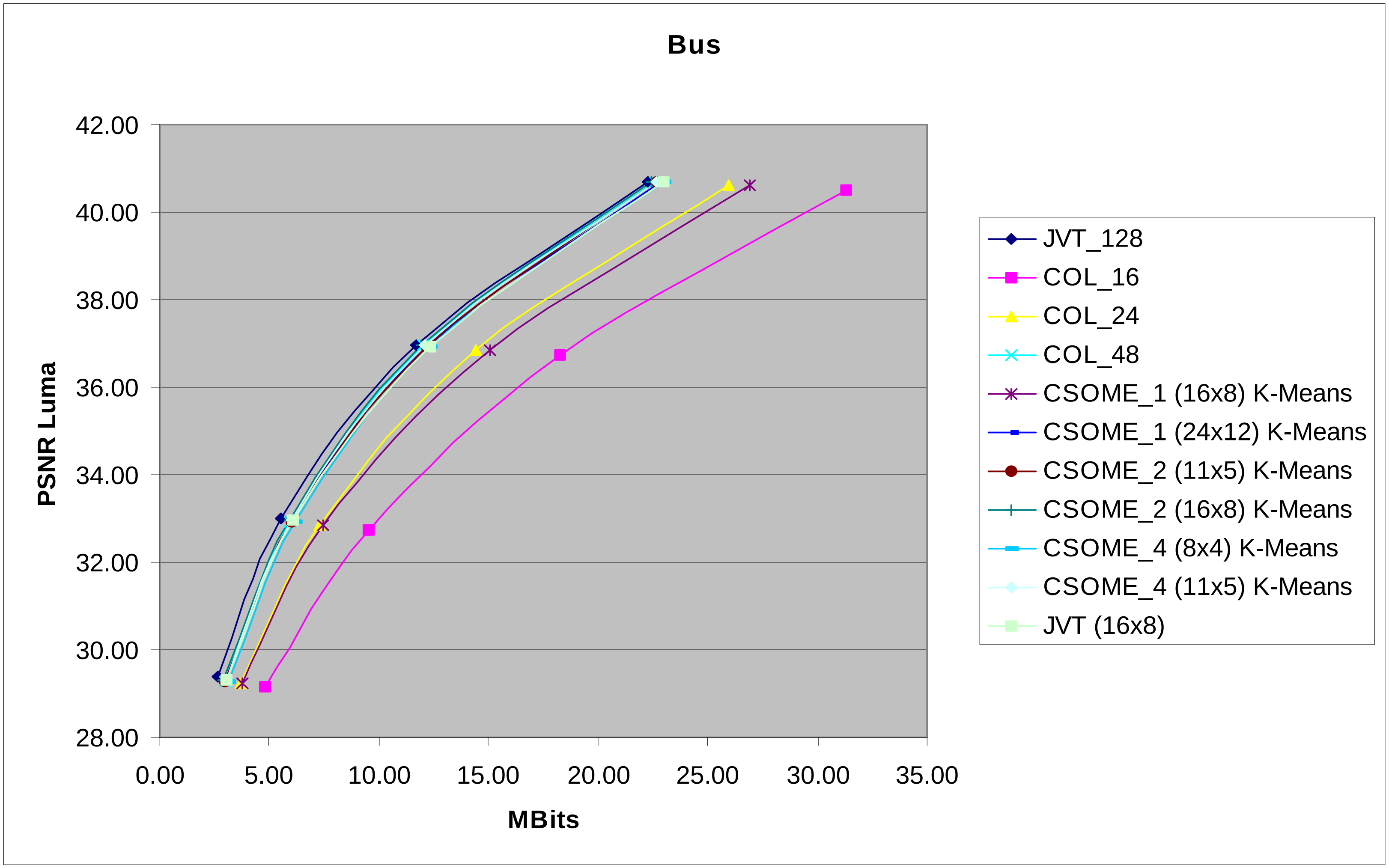
			JVT_128					CSOME_4_11 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.58	40.68	41.75	43.41	703.675	-1.3%	-6.82%	-0.36
		25	11.70	36.97	38.75	40.96	10897.736	12.11	36.95	38.73	41.00	685.841	-3.5%		
		30	5.56	32.99	36.71	39.11	8901.597	5.99	32.94	36.70	39.10	659.495	-7.7%		
		35	2.67	29.40	35.40	37.39	7820.718	3.02	29.31	35.38	37.41	674.799	-13.0%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	36.39	41.59	43.36	43.72	1400.032	-1.4%	-9.50%	-0.49
		25	21.09	38.09	40.91	41.34	22128.641	22.28	38.06	40.94	41.44	1393.638	-5.7%		
		30	11.07	34.30	38.92	39.48	17839.197	12.35	34.26	38.92	39.52	1293.436	-11.5%		
		35	6.02	30.77	36.97	37.45	15889.673	7.00	30.68	36.92	37.51	1252.857	-16.3%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.50	42.28	42.17	42.71	502.737	-4.9%	-10.17%	-0.53
		25	9.08	38.86	38.50	39.14	6824.357	9.77	38.84	38.49	39.12	484.547	-7.6%		
		30	4.75	34.99	35.29	36.39	5574.777	5.28	34.95	35.28	36.40	442.395	-11.2%		
		35	2.43	31.31	32.77	34.61	4881.838	2.83	31.30	32.75	34.56	449.546	-16.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.29	40.97	41.50	42.34	1479.976	0.6%	-7.73%	-0.35
		25	20.21	37.71	39.05	40.01	21491.589	20.89	37.73	39.14	40.09	1448.004	-3.3%		
		30	10.06	34.27	37.04	38.26	17484.307	11.16	34.22	37.13	38.34	1343.754	-10.9%		
		35	5.23	31.25	35.11	36.79	14986.002	6.22	31.20	35.21	36.84	1279.816	-18.8%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.96	40.08	40.64	40.73	596.422	-0.1%	-2.43%	-0.15
		25	19.42	35.83	36.61	36.67	11726.425	19.58	35.81	36.60	36.65	566.403	-0.8%		
		30	10.18	31.17	33.37	33.34	9982.049	10.45	31.15	33.36	33.32	555.703	-2.6%		
		35	4.64	26.69	30.76	30.58	8969.222	4.91	26.66	30.72	30.59	579.078	-5.8%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.60	40.97	42.45	43.31	1505.469	-0.3%	-7.66%	-0.39
		25	28.82	37.24	39.77	40.73	25192.607	29.95	37.25	39.80	40.81	1470.591	-3.9%		
		30	14.60	33.41	37.69	38.87	20042.996	16.22	33.41	37.62	38.84	1383.27	-11.1%		
		35	7.44	30.07	35.82	37.11	17910.821	8.73	30.04	35.68	37.08	1313.933	-17.3%		

Performance of CSOME_4_11 with K-Means Clustering compared to JVT_128

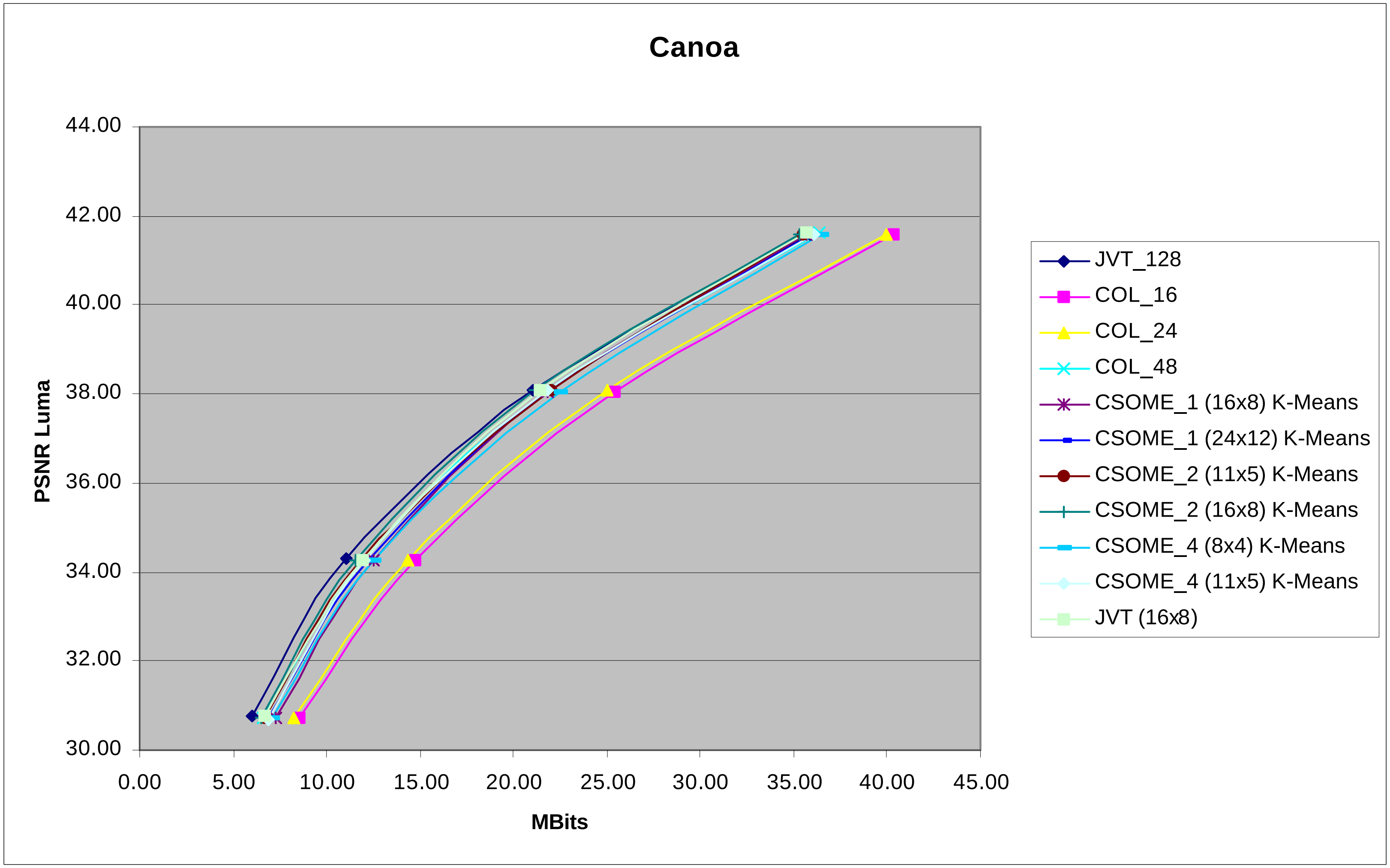
			JVT_128					CSOME_4_11 with K-Means Clustering							
Video Sequence	**Number of Frames**	**QP**	**Mbits**	**PSNRY (JVT_128)**	**PSNRU (JVT_128)**	**PSNRV (JVT_128)**	**Computation Time (sec)**	**Mbits**	**PSNRY**	**PSNRU**	**PSNRV**	**Computation Time (sec)**	**Total bit saving**	**Bjontegaard BR Delta saving**	**Bjontegaard PNSR Delta (dB)**
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.70	40.68	41.75	43.40	765.221	-1.8%	-6.92%	-0.37
		25	11.70	36.97	38.75	40.96	10897.736	12.09	36.95	38.76	41.01	651.232	-3.4%		
		30	5.56	32.99	36.71	39.11	8901.597	6.01	32.95	36.70	39.09	664.378	-7.9%		
		35	2.67	29.40	35.40	37.39	7820.718	3.04	29.32	35.37	37.34	647.186	-13.7%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	36.04	41.59	43.35	43.74	1405.072	-0.5%	-7.04%	-0.37
		25	21.09	38.09	40.91	41.34	22128.641	21.82	38.07	40.94	41.44	1420.241	-3.4%		
		30	11.07	34.30	38.92	39.48	17839.197	12.06	34.27	38.93	39.52	1310.263	-8.9%		
		35	6.02	30.77	36.97	37.45	15889.673	6.88	30.70	36.98	37.51	1304.722	-14.3%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.43	42.26	42.15	42.68	495.267	-4.5%	-10.18%	-0.52
		25	9.08	38.86	38.50	39.14	6824.357	9.67	38.84	38.48	39.11	490.795	-6.5%		
		30	4.75	34.99	35.29	36.39	5574.777	5.32	34.93	35.27	36.31	433.516	-12.0%		
		35	2.43	31.31	32.77	34.61	4881.838	2.80	31.29	32.73	34.55	398.95	-15.2%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.12	40.97	41.49	42.34	1385.52	1.1%	-7.46%	-0.34
		25	20.21	37.71	39.05	40.01	21491.589	20.85	37.73	39.15	40.09	1419.243	-3.2%		
		30	10.06	34.27	37.04	38.26	17484.307	11.08	34.22	37.11	38.33	1312.169	-10.1%		
		35	5.23	31.25	35.11	36.79	14986.002	6.20	31.19	35.23	36.84	1224.014	-18.4%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.78	40.09	40.65	40.73	524.436	0.5%	-0.81%	-0.05
		25	19.42	35.83	36.61	36.67	11726.425	19.46	35.83	36.62	36.67	465.607	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.30	31.17	33.39	33.33	446.154	-1.1%		
		35	4.64	26.69	30.76	30.58	8969.222	4.77	26.69	30.76	30.59	476.625	-2.8%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.48	40.98	42.45	43.31	1484.679	0.0%	-6.63%	-0.35
		25	28.82	37.24	39.77	40.73	25192.607	29.78	37.25	39.81	40.82	1492.972	-3.3%		
		30	14.60	33.41	37.69	38.87	20042.996	15.97	33.41	37.63	38.84	1369.02	-9.4%		
		35	7.44	30.07	35.82	37.11	17910.821	8.68	30.05	35.73	37.10	1320.338	-16.6%		

Performance of CSOME_4_11 with Single Iteration K-Means Clustering compared to JVT_128

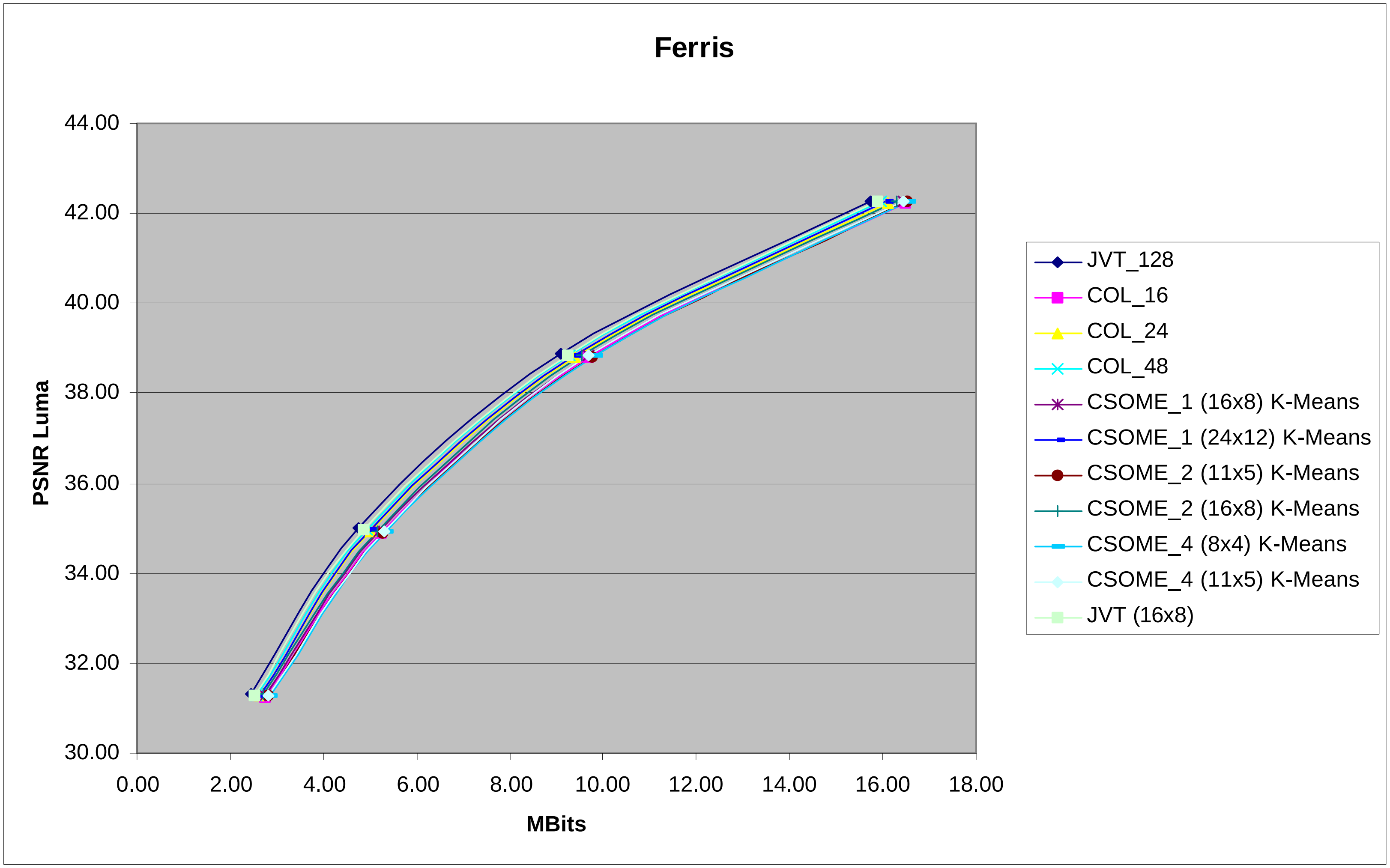
			JVT_128					CSOME_4_11 with Histogram Peak Detection							
Video Sequence	Number of Frames	QP	Mbits	PSNRY (JVT_128)	PSNRU (JVT_128)	PSNRV (JVT_128)	Computation Time (sec)	Mbits	PSNRY	PSNRU	PSNRV	Computation Time (sec)	Total bit saving	Bjontegaard BR Delta saving	Bjontegaard PNSR Delta (dB)
Bus	75	20	22.29	40.69	41.74	43.38	11586.032	22.65	40.68	41.74	43.40	733.004	-1.6%	-7.02%	-0.37
		25	11.70	36.97	38.75	40.96	10897.736	12.17	36.95	38.75	41.00	730.14	-4.1%		
		30	5.56	32.99	36.71	39.11	8901.597	5.99	32.94	36.71	39.09	651.784	-7.6%		
		35	2.67	29.40	35.40	37.39	7820.718	3.02	29.30	35.38	37.38	669.507	-13.3%		
Canoa	110	20	35.88	41.60	43.29	43.63	23232.25	35.99	41.59	43.35	43.73	1384.939	-0.3%	-7.14%	-0.38
		25	21.09	38.09	40.91	41.34	22128.641	21.80	38.07	40.94	41.45	1421.94	-3.3%		
		30	11.07	34.30	38.92	39.48	17839.197	12.09	34.27	38.94	39.51	1310.548	-9.2%		
		35	6.02	30.77	36.97	37.45	15889.673	6.90	30.70	36.97	37.52	1215.623	-14.6%		
Ferris	60	20	15.73	42.27	42.17	42.68	7170.907	16.35	42.25	42.16	42.67	468.721	-3.9%	-9.92%	-0.50
		25	9.08	38.86	38.50	39.14	6824.357	9.64	38.83	38.47	39.11	462.402	-6.2%		
		30	4.75	34.99	35.29	36.39	5574.777	5.31	34.94	35.30	36.36	422.061	-11.9%		
		35	2.43	31.31	32.77	34.61	4881.838	2.78	31.28	32.74	34.56	373.276	-14.3%		
Football	130	20	37.53	40.95	41.42	42.28	22921.7	37.14	40.97	41.49	42.34	1438.72	1.1%	-7.36%	-0.34
		25	20.21	37.71	39.05	40.01	21491.589	20.81	37.74	39.15	40.09	1384.755	-3.0%		
		30	10.06	34.27	37.04	38.26	17484.307	11.06	34.22	37.11	38.33	1296.607	-10.0%		
		35	5.23	31.25	35.11	36.79	14986.002	6.22	31.18	35.26	36.83	1223.935	-18.9%		
Mobile	130	20	32.94	40.09	40.63	40.73	11973.894	32.74	40.09	40.65	40.73	471.827	0.6%	-0.86%	-0.05
		25	19.42	35.83	36.61	36.67	11726.425	19.46	35.82	36.61	36.68	464.872	-0.2%		
		30	10.18	31.17	33.37	33.34	9982.049	10.30	31.17	33.38	33.33	458.157	-1.1%		
		35	4.64	26.69	30.76	30.58	8969.222	4.78	26.69	30.76	30.58	465.581	-3.1%		
Rugby	110	20	49.46	40.97	42.37	43.20	26423.156	49.56	40.98	42.46	43.31	1477.968	-0.2%	-6.64%	-0.35
		25	28.82	37.24	39.77	40.73	25192.607	29.76	37.26	39.81	40.81	1451.681	-3.3%		
		30	14.60	33.41	37.69	38.87	20042.996	15.95	33.40	37.63	38.85	1380.205	-9.3%		
		35	7.44	30.07	35.82	37.11	17910.821	8.71	30.05	35.72	37.10	1319.436	-17.0%		



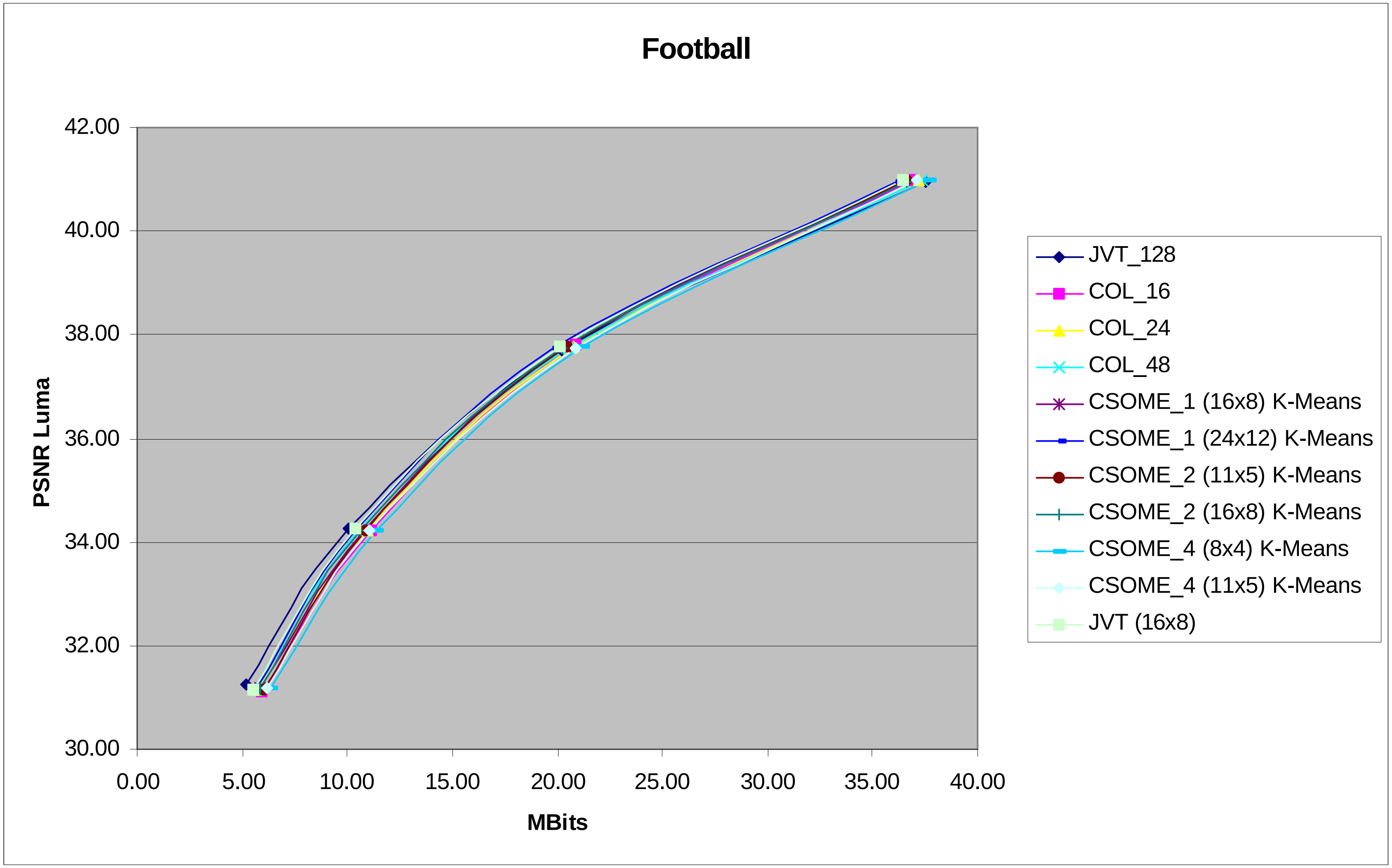
Rate-Distortion performance of evaluated motion estimation algorithms for Bus



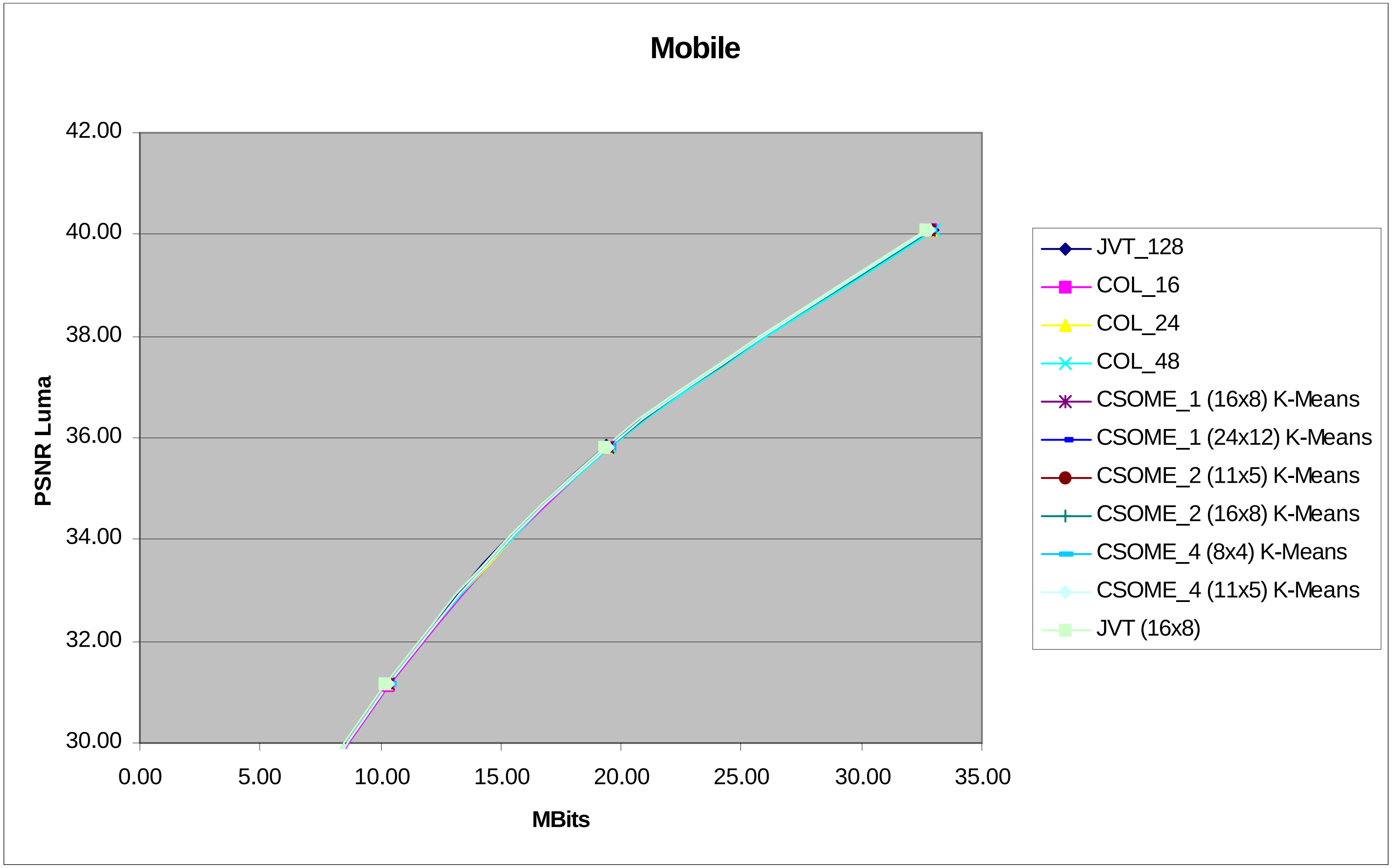
Rate-Distortion performance of evaluated motion estimation algorithms for Canoa



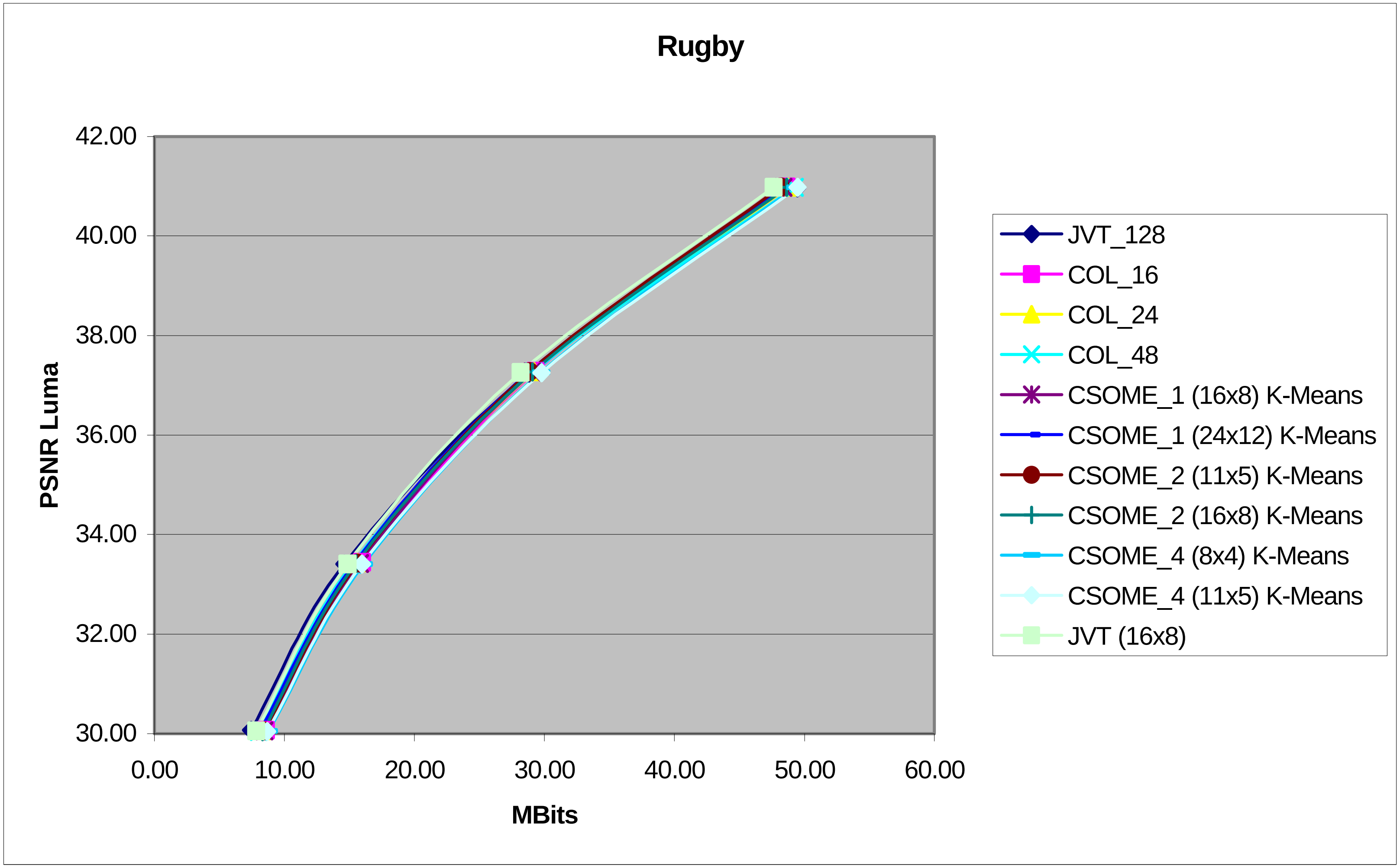
Rate-Distortion performance of evaluated motion estimation algorithms for Ferris



Rate-Distortion performance of evaluated motion estimation algorithms for Football



1 Rate-Distortion performance of evaluated motion estimation algorithms for Mobile



Rate-Distortion performance of evaluated motion estimation algorithms for Rugby

Bibliography

[1] Chen, J., U. Koc, and K. J. R. Liu. *Design of Digital Video Coding Systems - A Complete Compressed Domain Approach,* New York: Marcel Dekker Inc., 2002.

[2] Cote, G. and L. Winger, Recent Advances in Video Compression Standards *IEEE Canadian Review*, vol. pp. 21-24, 2002.

[3] Bhaskaran, V. and K. Konstantinides. *Image and Video Compression Standards - Algoritms and Architectures,* Boston: Kluwer Academic Publishers, 1997.

[4] Yang, K. H. and A.F. Faryar, "A context-based predictive coder for lossless and near-lossless compression of video," *Proceedings of International Conference on Image Processing,* pp. 144-147, 2000.

[5] Oami, R. and M. Ohta, "Efficient lossless video coding compatible with MPEG-2," *Preceedings of IEEE Confererence on Communications,* pp. 901-905, 1998.

[6] Ribas-Corbera, J. and D.L. Neuhoff, "Optimal bit allocations for lossless video coders: motion vectors vs. difference frames ," *Proceedings of International Conference on Image Processing,* pp. 180-183, 1995.

[7] Marlow, S., J. Ng, and C. McArdle, "Efficient motion estimation using multiple log searching and adaptive search windows," *Image Processing and Its Applications, Sixth International Conference on,* pp. 214-218, 1997.

[8] Lin, Y. and S. Tai., Fast full-search block-matching algorithm for motion-compensated video compression *IEEE Transactions on Communications,* vol. 45, pp. 527-531, 1997.

[9] Lee, S., J. Kim, and S. Chae, New Motion Estimation Algorithm Using Adaptively Quantized Low Bit-Resolution Image and Its VLSI Architecture for MPEG2 Video Encoding *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 8, pp. 734-744, 1998.

[10] Jain, J. and A. Jain, Displacement Measurement and its Application in Interframe Image Coding *IEEE Transactions on Communications,* vol. COM-29, pp. 1799-1808, 1981.

[11] Ghanbari, M., The cross-search algorithm for motion estimation *IEEE Transactions on Communications*, vol. 38, pp. 950-953, 1990.

[12] Gallant, M. , G. Cote, and F. Kossentini, A new center-biased search algorithm for block motion estimation *IEEE Transactions on Image Processing,* vol. 8, pp. 1816-1823, 1999.

[13] Chen, Y., Y. Hung, and C. Fuh, Fast Block Matching Algorithm Based on the Winner-Update Strategy *IEEE Transactions on Image Processing*, vol. 10, no. 8, pp. 1212-1222, 2001.

[14] Calvagno, G., F. Fantozzi, and R. Rinaldo, "Feature based global and local motion estimation for videoconference sequences," *Proceedings of International Conference on Image Processing,* pp. 102-105, 2001.

[15] Brunig, M. and W. Niehsen, Fast Full-Search Block Matching *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, pp. 241-247, 2001.

[16] Baek, Y. H. O. a. H. L., An efficient block-matching criterion for motion estimation and its VLSI implementation *IEEE Transactions on Consumer Electronics*, vol. 42, pp. 885-892, 1996.

[17] ITU-T. Video Codec for Audiovisual Services at px64 kbit/s. ITU-T Recomendation H.261 . 93.

[18] ITU-T and ISO/IEC JTC1. Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video. ITU-T Recommendation H.262 - ISO/IEC 13818-2 (MPEG-2) . 94.

[19] ITU-T. Video Coding for Low Bitrate Communication. ITU-T Recommendation H.263 . 98.

[20] ISO/IEC. Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5Mbit/s. ISO/IEC 11172 .

[21] VCEG, Joint Video Team of ISO IES MPEG and ITU-T. Joint Committee Draft (CD) of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC). Wiegand, T. 2002.

[22] Mullen, K. T., The contrast sensitivity of human colour vision to red-green and blue- yellow chromatic gratings *Journal of Physiology*, vol. pp. 381-400, 1985.

[23] Mitchell, J. L., W. B. Pennebaker, C. E. Fogg, and D. J. LeGall. *MPEG Video Compression Standard,* Boston: Kluwer Academic Publishers, 1996.

[24] Gerken, P., Object-based analysis-synthesis coding of image sequences at very low bit rates *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, pp. 228-235, 1994.

[25] Lee, S., H. D. Cho, Y. Cho, S. Son, E. S. Jang, J. Shin, and Y. S. Seo, Binary shape coding using baseline-based method *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 44-58, 1999.

[26] Nakaya, Y. and H. Harashima, Motion Compensation Based on Spatial Transformations *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, pp. 339-367, 1994.

[27] Sullivan, G. J. and R. L. Baker, "Rate-distortion optimized motion compensation for video compression using fixed or variable size blocks," *Proceedings of Global Telecommunications Conference*, pp. 85-90, 1991.

[28] Rao, K. R. and J. J. Hwang. *Techniques and Standards for Image, Video and Audio Coding*, London: Prentice Hall PTR, 1996.

[29] Itoh, Y. and N. Cheung, "Universal variable length code for DCT coding," *Proceedings of International Conference on Image Processing*, pp. 940-943, 2000.

[30] Marpe, D., H. Schwarz, G. Bldttermann, G. Heising, and T. Wiegand, "Context-based adaptive binary arithmetic coding in JVT/H.26L," *Proceedings of International Conference on Image Processing*, pp. 513-516, 2001.

[31] Wei, J. and Z. Li, An efficient two-pass MAP-MRF algorithm for motion estimation based on mean field theory *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 960-972, 1999.

[32] Kuhn, P. *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*, Boston: Kluwer Academic Publishers, 1999.

[33] Mizuki, M., U. Desai, I. Masaki, and A. Chandrakasan, "A binary block matching architecture with reduced power consumption and silicon area requirement," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3248-3251, 1996.

[34] He, Z., C. Tsui, K. Chan, and M. Liou, Low-power VLSI design for motion estimation using adaptive pixel truncation *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, pp. 669-678, 2000.

[35] Ogura, E., M. Takashima, D. Hiranaka, T. Ishikawa, Y. Yanagita, S. Suzuki, T. Fukuda, and T. Ishii, A 1.2-W Single-Chip MPEG2 MP@ML Video Encoder LSI Including Wide Search Range (H: +-288, V: +-96) Motion Estimation and 81-MOPS Controller *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1765-1771, 1998.

[36] Cheng, K. and S. Chan, "Fast block matching algorithms for motion estimation," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2311-2314, 1996.

[37] Sauer, K. and B. Schwartz, Efficient block motion estimation using integral projections *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 513-518, 1996.

[38] Qiu, X., W. Zhang, H. Chen, and R. Zhou, "Low entropy block matching algorithm for motion estimation," *Proceedings of 4th International Conference on ASIC*, pp. 405-408, 2001.

[39] Valova, I. and Y. Kosugi, Hadamard-based image decomposition and compression *IEEE Transactions on Information Technology in Biomedicine*, vol. 4, pp. 306-319, 2000.

[40] Hoang, D., P. Long, and J. Vitter, Efficient cost measures for motion estimation at low bit rates *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 488-500, 1998.

[41] Jong, H., L. Chen, and T. Chiueh, Parallel architectures for 3-step hierarchical search block-matching algorithm *IEEE Transactions on Circuits and Systems for Video Technology* , vol. 4, pp. 407-416, 1994.

[42] Pirsch, P. and H. Stolberg, VLSI Implementation of Image and Video Multimedia Processing Systems *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 878-891, 1988.

[43] Cheng, S. and H. Hang, A Comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementations *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, pp. 741-757, 1997.

[44] Shen, J., T. Wang, and L. Chen, A novel Low-Power Full-Search Block-Matching Motion-Estimation Design for H.263+ *IEEE Transactions on Circuits and Systems for Video Technology* , vol. 11, pp. 890-897, 2001.

[45] Kuhn, P., A. Weisgerber, R. Poppenwimmer, and W. Stechele , "A flexible VLSI architecture for variable block size segment matching with luminance correction," *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 479-488, 1997.

[46] Bjontegaard, G. H.26L Test Model Long Term Number 8 (TML-8) draft0. Video Coding Experts Group . 2001.

[47] Koga, T., K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," *Proceeding of NTC '81*, New Orleans, LA, pp. G5.3.1-G5.3.5, 1981.

[48] Po, L. and W. Ma, A novel four-step search algorithm for fast block motion estimation *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 313-317, 1996.

[49] Zhu, S. and K. Ma, A new diamond search algorithm for fast block-matching motion estimation *IEEE Transactions on Image Processing*, vol. 9, pp. 287-290, 2000.

[50] Po, L. and W. Ma, "A new center-biased search algorithm for block motion estimation," *Proceedings of International Conference on Image Processing*, pp. 410-412, 1995.

[51] Lakamsani, P., An architecture for enhanced three step search generalized for hierarchical motion estimation algorithms *IEEE Transactions on Consumer Electronics*, vol. 43, pp. 221-227, 1997.

[52] ANSI/SMPTE. Component Video Signal 4:2:2 - Bit-Parallel Digital Interface. ANSI/SMPTE 125M-1995 .

[53] Pirsh, P., N. Demassieux, and W. Gehrke, VLSI Architectures for Video Compression - A survey *Proceedings of the IEEE*, vol. 83, pp. 220-246, Feb, 1995.

[54] Nam, J., J. Seo, J. Kwak, M. Lee, and Y. H. Ha, New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 934-942, 2000.

[55] Duda, R. O. , P.E. Hart, and D.G Stork. *Pattern Classification*, Toronto: John Wiley & Sons, 2001.

[56] Sullivan, G. J. and T. Wiegand, Rate-distortion optimization for video compression *IEEE Signal Processing Magazine*, vol. 15, pp. 74-90, Nov, 1998.

[57] Bjontegaard, G., "Calculation of average PSNR differences between RD curves," *ITU-T VCEG Meeting*, Austin, TX, pp. VCEG-M33, 2001.