PHD

Controllable compression of motion descriptions

Tredwell, Simon

Award date:
2004

Awarding institution:
University of Bath

Link to publication

CONTROLLABLE COMPRESSION OF MOTION DESCRIPTIONS

Submitted by **Simon Tredwell**
for the degree of Doctor of Philosophy
of the University of Bath
2004

Abstract

In video coding, temporal redundancy can be reduced by predicting subsequent frames from a reference frame. In many practical video codecs this is carried out using block matching techniques to generate a motion description (usually a motion vector field) from a pair of frames which is used to form a prediction of the current frame. As the predicted frame is only an approximation of the true current frame and to improve its similarity a residual or error image is generated and encoded. While the number of bits used for the residual image is made scalable by using a lossy compression scheme, the bit allocation for the motion description is fixed. As a consequence, the optimal bit allocation between the motion description and the residual image cannot be achieved. This thesis investigates methods for controlling the bandwidth allocated to the motion description in order to more closely achieve the optimal allocation of bits between the description and the residual. Several new techniques are introduced in order to attempt to control the bandwidth. The Extended Block Algorithm produces a smoother field with the higher correlation leading to greater compression. The Embedded Quad-tree Motion Estimation technique describes the motion field using an embedded stream that can be truncated depending on the desired quality/cost criteria. Building on the observation that motion vector fields offer better compression when there are fewer unique vectors in the field, a number of strategies for controlling the number of unique vectors are proposed and evaluated. The List/Mapping Motion Description (LMMD) is an alternative representation of the vectors generated by these techniques. Combinations of the vector selection and encoding methods are then evaluated within an H.263 based codec and compared to the standard encoder. At a higher bit-rate (128 kbits/s) the vector selection strategies show an improvement in PSNR when compared with the standard codec. At the lower bit-rates (32 kbits/s and 64 kbits/s) an improvement is shown when the motion description generated by the vector selection strategies is encoded using the LMMD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Digital Video Encoding

One of the key components of digital video coding is that of motion compensation. This allows modern codecs to take advantage of the temporal redundancy which exists between neighbouring frames in a video sequence by producing a description of the motion which occurs between frames. The bandwidth used to encode a frame of motion compensated video is distributed between the motion compensation data and a compressed residual image. The more bandwidth used by the motion compensation the less is available to the residual and vice versa.

One problem associated with motion compensation is the complexity involved. Much research has been done into techniques which reduce this complexity whilst retaining acceptable performance. However, there is another issue to be addressed which is concerned with the distribution of bandwidth between the the motion compensation data and the residual.

The greater the bandwidth which is allocated to the motion description the more accurate the motion compensated image will be. This increase in bandwidth will be at the expense of the quality of the residual image which will be less able to correct errors in the compensated image, or to add image data due to revealed occlusions. The converse also applies, reducing the bandwidth available to the motion description allows gives a less accurate compensated image but a higher quality residual.

Traditional video codecs have only allowed the bandwidth of the residual to be controlled; they would be allocated the bandwidth *not* used by the motion description. This work proposed method which give greater control over the bandwidth used by the motion description. By varying both the motion description and the residual it should be possible to find a more optimal distribution of bandwidth and therefore improve the overall quality of the encoded sequence.

It is also worth noting that the method described in this work are *not* designed to be low complexity. However the complexity of the various techniques are compared and discussed.

1.1 Digital Image Representation

In order to capture an image on a digital device such as a computer, the infinitely complex information which comprises the real image must be broken down into discrete pieces of data which the computer is able to process.

A computer can describe an image as a two-dimensional array of points, also known as *pels* (picture elements) or *pixels*. Although pixels can be rectangles of arbitrary dimensions they are usually considered to be square. Each of these pixels contains the information needed to describe the colour of that particular part of the image. This information varies depending upon the type of image to be stored.

Binary Image The pixel is either on or off. Each pixel only requires one bit of storage to represent its binary state.

Indexed Image The pixel contains an index which references a lookup table containing the actual pixel colour information. In this kind of image the total number of colours in an image is limited by the number of indices (which is usually $\leqslant 256$).

Greyscale Image The pixel contains the brightness. The pixel is usually stored using one byte (8 bits) which allows 256 different levels of grey to be expressed. In some cases, such as medical imaging [2] greater discrimination between grey levels may be required. Such applications may use 12 bits per pixel.

Colour Image The pixel contains three components of a given colourspace. These are usually expressed as a triplet of bytes, one byte for each component.

1.2 Colourspaces and Image Planes

Due to the tri-stimulus theory of the Human Visual System (HVS) a particular perceived colour requires three values to describe it. In a colour image these three values are usually represented as three distinct image planes, which are superimposed to produce the complete image. The exact contents of each image plane depends upon the particular *colourspace* which is used to describe the colour of each pixel.

The RGB colourspace is one of the most familiar. Its name is derived from the Red, Green and Blue components into which the colour is split. This colourspace is analogous to that used by the human eye and is used in modern Cathode Ray Tubes (CRTs) which use phosphors to give off red, green and blue light[3]. In digital systems each of the three components is usually specified as an integer between 0 and 255, where RGB(0,0,0) represents black, RGB(255,255,255) is white and RGB(x,x,x) are the intermediate grey levels. Figure 1.1 shows an image which has been split up into red, green and blue planes.

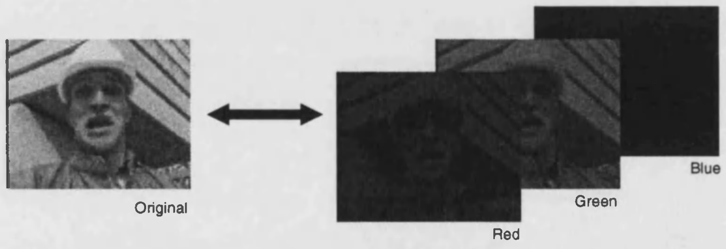


Figure 1.1: Colour images are made up of different 'planes' depending upon the colourspace used. In this example the image has been split into the three planes of the RGB colourspace.

In the RGB colourspace the luminance information is spread across all three planes. In many image processing applications it is useful to have direct access to the luminance data. Suppose we have a luminance component which we denote as Y and that is is some function of the three primaries of the RGB colourspace, $Y = f(R, G, B)$. We can now subtract this component from the RGB primaries to give $(R - Y)$, $(G - Y)$ and $(B - Y)$. Due to the fact that Y is a function of R, G and B only two of these differences are needed. Suppose we choose $(B - Y)$ and $(R - Y)$ which we call C_b and C_r respectively[4, 5]. We now have the three values which form the basis of the YCbCr colourspace, Y, C_b and C_r. The Y component can be represented as an integer between 0 and 255, with 0 being black and 255 being white. C_r and C_b are also represented as integers between 0 and 255 with 128 indicating no colour difference.

A greyscale image only contains luminance data which is stored in a single image plane. This plane corresponds to the Y plane in the YCbCr colourspace.

1.2.1 Colourspace conversion

Video data often has to be converted from one colourspace to another, for example, after decoding an H.263 data stream[1], the images are presented in YCbCr colourspace. In order to display such images on an RGB CRT a colourspace conversion needs to be performed. A suitable RGB to YCbCr conversion is given equation 1.1 with the inverse given in equation 1.2[4].

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \begin{bmatrix} 65.738 & 129.057 & 25.064 \\ -37.945 & -74.494 & 112.439 \\ 112.439 & -94.154 & -18.285 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1.1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 298.082 & 0.000 & 408.583 \\ 298.082 & -100.291 & -208.120 \\ 298.082 & 516.411 & 0.000 \end{bmatrix} \left(\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right) \tag{1.2}$$

Although this pair of transforms are mathematically reversible, the limited precision of floating point arithmetic used in computers can cause losses which in some applications are unsuitable. In these cases the following integer only approximations can be used,

$$\begin{aligned} C_b &= B - G \\ C_r &= R - G \\ Y &= G + \left\lfloor \frac{C_b + C_r}{4} \right\rfloor \end{aligned}$$

$$\begin{aligned} G &= Y - \left\lfloor \frac{C_b + C_r}{4} \right\rfloor \\ B &= C_b + G \end{aligned}$$

$$R = C_r + G$$

In this thesis $\lfloor x \rfloor$ will indicate the *floor* of x, that is, x rounded down to the next integer.

For a more detailed discussion of colour and colourspaces in digital video see [6, 7, 4].

1.3 Image Representation

The size of a digital image is usually measured in pixels. In this thesis we will take the origin of the image to be at the top left hand corner of the image, which is traditional in image processing. The width of an image (I_{width}) is defined as the number of pixels in the horizontal dimension and the height (I_{height}) as the number of pixels in the vertical dimension. Values of pixels are specified using horizontal and vertical offsets into the image, with the top left pixel having the offset $(0,0)$ (see figure 1.2). Thus the range of values (x,y) for referencing a general pixel inside the image is

$$0 \leqslant x < I_{\text{width}}, 0 \leqslant y < I_{\text{height}}.$$

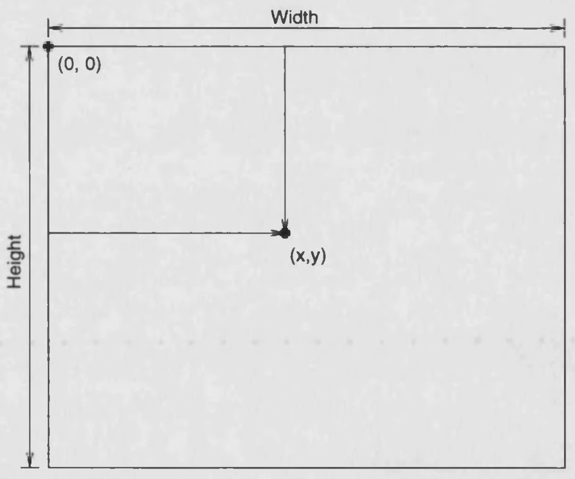


Figure 1.2: The origin of an image is taken to be in the top-left corner. Image pixels are referenced as offsets from this origin.

There are certain image sizes which are frequently used in digital video processing[8]. These have been given names and are summarised in table 1.1.

Name	Width in pixels	Height in pixels
sub-QCIF	128	196
QCIF	176	144
CIF	352	288
4CIF	704	576
16CIF	1408	1152

Table 1.1: Different image sizes

1.4 Video Sequence Representation

A digital video sequence is simply a temporal sequence of digital images, which are usually referred to as frames. These frames are displayed at a regular frequency which varies depending upon the application, but for smooth, high quality video, is usually 25Hz or 30Hz. These values are inherited from analogue video standards, which vary across the globe. Unlike analogue video, digital video is not usually interlaced which is why the frequencies are half of those used in traditional analogue broadcasting, (usually 50Hz or 60Hz).

When stored in a raw format it is straightforward to determine the storage required for uncompressed digital video. For example 30 minutes of CIF size (roughly VHS resolution) video running at 25 Hz contains,

$$(30 \times 60) \text{ seconds} \times 25 \text{ frames/second} = 45,000 \text{ frames.} \tag{1.3}$$

If each frame is stored as three separate red, green and blue colour planes then each frame requires,

$$(352 \times 288) \text{ pixels/frame} \times 3 \text{ bytes/pixel} = 304,128 \text{ byes/frame.} \tag{1.4}$$

So the total storage required would be,

$$45,000 \text{ frames} \times 304,128 \text{ bytes/frame} = 13.7 \times 10^9 \text{ bytes} \quad (13.7\text{GB}). \tag{1.5}$$

Even with the current increased availability of large, cheap computer storage, this is still unacceptably large. As well as the storage requirements it is also worth considering the time taken to transmit this data between computers on a network.

Video compression techniques have been developed in order to facilitate both the transmission and storage of digital video.

1.5 Sources of Redundancy in Digital Video

Any form of compression relies on the fact that the data to be compressed contains redundant information [9]. This implies that the data is in some way correlated.

1.5.1 Spatial Redundancy

In natural images there is often substantial correlation between pixels in a local neighbourhood. For example, if a particular pixel is part of a blue sky, neighbouring pixels are also likely to be a similar shade of blue. This *spatial redundancy* is used by *intra-frame coding* techniques to compress individual frames of a sequence. Figure 1.3 shows an enlarged 16×16 block from a natural image. This clearly shows that there is correlation between the values of neighbouring pixels.

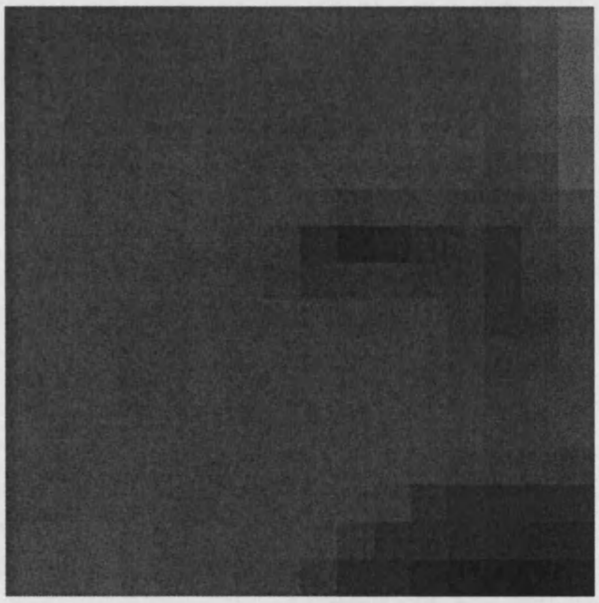


Figure 1.3: This is a 16×16 block taken from a natural image. The shade of each pixel is frequently similar to those around it.

1.5.2 Temporal Redundancy

There is also a high level of *temporal redundancy* within a video sequence. That is to say that one frame is usually very similar to the next. In order to take advantage of such similarity *inter-frame coding* techniques are used. These methods often try to describe a frame in a sequence in terms of a previously transmitted frame or frames. Figure 1.4 shows two frames from the *Carphone* sequence. It can be seen that there is a large amount of redundant information between the two frames.

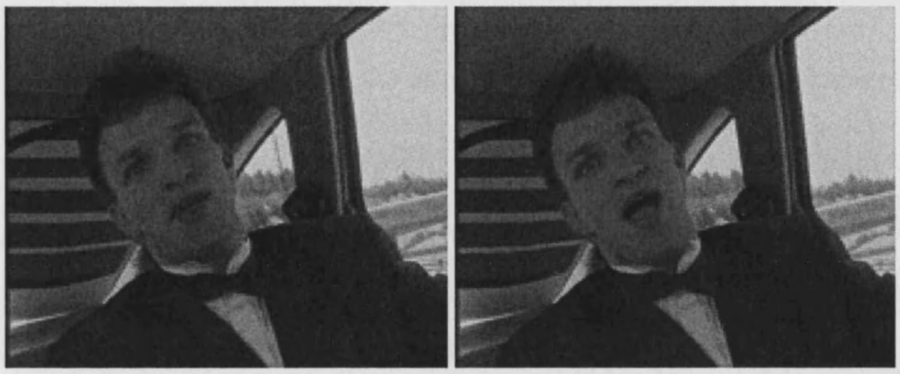


Figure 1.4: Two frames from Carphone sequence (frames 100 and 104). It is clear that there is a great deal of temporal redundancy between the frames.

1.5.3 Psychovisual Redundancy

The nature of the Human Visual System (HVS) leads to a further type of redundancy known as psychovisual redundancy. For example, the human eye is less sensitive to spatial variation in chrominance than it is in luminance. As a result the chrominance planes of an image can be transmitted at a lower resolution than that of the luminance.

1.5.4 Coding Redundancy

The video compression process can be thought of as transforming the input video sequence into a stream of symbols. Often, some of these symbols are statistically more likely to appear than others. It therefore makes sense to use fewer bit to encode the more frequent symbols, even if it at the expense of using more bits for infrequent symbols. This *coding redundancy* is usually exploited using *entropy encoding* techniques.

1.6 Video Coding Techniques

In general compression techniques fall into two categories, *lossless* compression [10] and *lossy* compression. As its name implies, in lossless compression, the compressed representation of the data contains *all* the information contained in the original, so when decompressed the original data can be recovered exactly. This is vital for many types of data such as binary executable files, text files and many medical images. However, because all the information must be retained, the amount of compression which can be obtained is limited. In certain cases it is acceptable for the compression process to cause the original data to be subject to a certain amount of degradation. In this case the recovered data is similar, although not identical to the original. This is known as lossy compression and lends itself well to data which is analogue in origin and is intended for 'consumption' by humans. Digital images, video and audio data is ideal for this type of compression. As some information is discarded during the compression process higher rates of compression can be achieved when compared with lossless techniques. There is a subset of lossy coding which is known as visually-lossless coding, or more generally perceivably lossless coding. In this case the decompressed signal has still been degraded with respect to the original, but the difference is imperceivable to an observer (or listener).

One further feature of lossy coding is that the amount of compression can often be controlled. Compression is increased by throwing away more of the information present in the original data, however, the greater the compression the greater the degradation of the output data.

1.7 Lossy Image Compression

The Transform Coding Model (TCM) is popular in the field of still image compression as it is able to take advantage of much of the spatial and psychovisual redundancy found in natural images. The addition of an entropy encoder also exploits any coding redundancy in the system. The underlying principle of transform coding is to remove the correlation in the input data by transforming it into a different domain. The general form of the transform coding model is shown in figure 1.5.

Six of the eight elements in the model form pairs of complementary processes, in which one half of the pair is the inverse of the other. The entropy coding pair must form a completely reversible transform, (the entropy codec must not

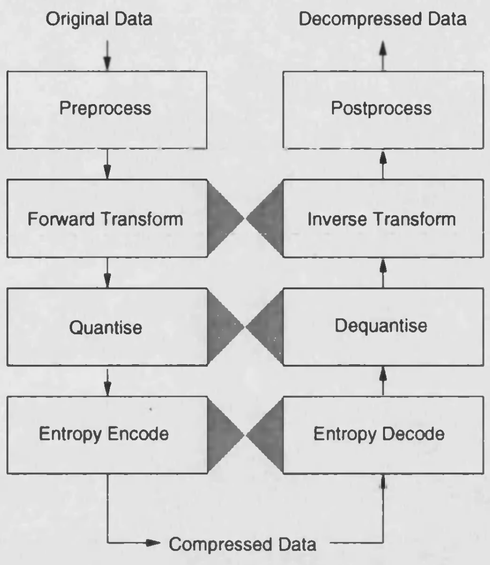


Figure 1.5: The Transform Coding Model.

cause data to be lost). Ideally the same is true for the transform pair, although in practice many transforms work with floating point numbers, so small inaccuracies due to lack of arithmetic precision are often introduced. The final pair, quantisation/dequantisation is not at all reversible. It is at this stage that information is irrevocably discarded, allowing greater compression of the information. As a result of this the model represents a lossy compression technique.

1.7.1 Preprocessing/Postprocessing

Many systems which acquire digital signals are susceptible to introducing noise into the signal. Such noise is unwanted and due to the fact that it often contains high frequency components, can be very costly to encode. Video compression systems often contain a preprocessing step which aims to reduce the amount of noise in an image without affecting the quality of the image itself. There are many types of noise reduction methods which use a variety of techniques [11, 12].

The preprocessing stage may also be used to convert the data into a more convenient form for encoding. For example, many video compression systems require the images to be represented using the YCbCr colourspace, so RGB images would have to be converted before they could be processed.

Sometimes the video compression process can produce ‘artifacts’ in the output sequence. These artifact are unsightly aberrations which become more pronounced as the amount of compression is increased. Common artifacts are *blocking* where the image appears to be divided into blocks and *ringing* where phantom edges appear to echo away from a true edge. Figure 1.6(a) shows part of an image which has been heavily compressed with the Joint Photographic Expert Group (JPEG) [13] algorithm; the artifacts are clear. Applying postprocessing techniques (smoothing and sharpening) on this image produces the image seen in figure 1.6(b) in which the artifacts have been reduced.

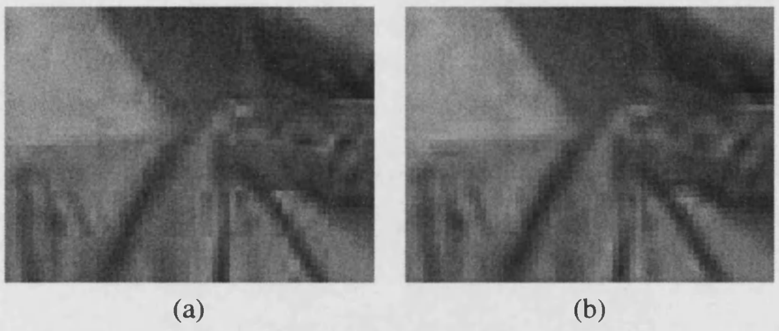


Figure 1.6: In (a) we seen part of an image which has been heavily compressed using the JPEG algorithm. Applying smoothing and sharpening filters results in (b) in which the artifacts have been reduced.

Pre and post processing techniques often rely on the properties of the HVS and aim to exploit psychovisual redundancy.

1.7.2 Transforms

In images it is the edges which convey much of the information to the observer. Conversely, large flat regions contain very little information. It would therefore be desirable for a video encoding system to distinguish between such areas, and more accurately describe those which are high in information content.

The transform stage of the Transform Coding Model (TCM) is designed to achieve this. It exploits the spatial redundancy within an image and attempts to order the data so as to identify those parts which contain the most useful visual information.

There are two main transforms used in image processing which are the Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). We shall briefly examine these to see how they are used in the context of image processing.

1.7.2.1 The Discrete Cosine Transform

The Discrete Cosine Transform (DCT) [14] is related to the Fourier Transform and Discrete Fourier Transform and aims to decorrelate data by transforming it into the frequency domain. In image processing the DCT is performed in a two dimensional fashion, by first applying the transform in the horizontal direction, then the vertical.

Suppose we have an $N \times N$ block of data, $f(x,y)$ we can transform to $F(u,v)$ using,

$$F(u,v) = C_u C_v \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}, \qquad (1.6)$$

where $C_a = \sqrt{1/N}$ if $a = 0$, $\sqrt{2/N}$ otherwise.

The inverse transform is similar and is given by,

$$f(x,y) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_u C_v F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}. \qquad (1.7)$$

Due to the computational cost it is impractical to carry out the DCT over the whole image. Many types of compression such as JPEG and H.263 take $N = 8$ and spilt the image the image up into 8×8 blocks. The DCT is performed of each of these blocks independently. The 2D DCT basis functions for $N = 8$ are shown in figure 1.7.

1.7.2.2 The Discrete Wavelet Transform

Wavelets represent a multiresolutional approach to image processing. The foundations of wavelet theory were proposed by Gabor in 1946 [15], but it is the last decade which has seen an explosion in research in the subject [16, 17, 18, 19]. In the same way that Fourier transforms can lead to the DCT the DWT can be derived from wavelet theory [20, 21]. The replacement for the current JPEG standard JPEG2000 [22] uses wavelet transforms to replace the DCT of the original. As with the DCT the job of the DWT is to transform the image data into a domain where there is less correlation. This leads to the exploitation of the spatial redundancy in the data. One important difference between Fourier transforms and wavelet transforms is that the basis functions for wavelet transforms are finite in their ex-

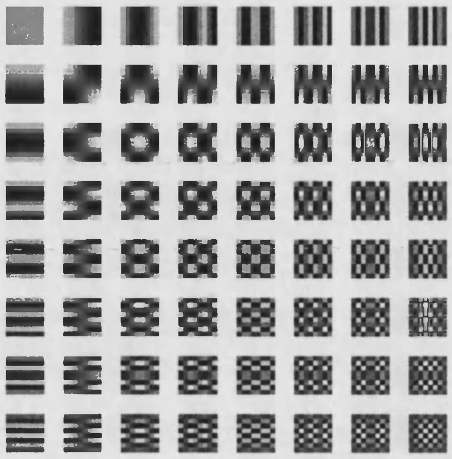


Figure 1.7: The 64 basis functions for the two dimensional DCT for a block size of 8×8.

tent, (unlike the infinite sinusoids used by Fourier transforms). There are an infinite variety of wavelets with different wavelets being useful in different applications.

Like the DCT the two dimensional wavelet transform is composed of two separable one dimensional transforms, one horizontal, one vertical. Each application of the transform divides the data into two components, one which is an approximation of the original signal and the other the detail which is evident at the given resolution. The transform is reversible, so these two components can be recombined to retrieve the original data. The transform is usually recursively applied to the approximated data to extract details at different resolutions.

Figure 1.8 shows an image which has undergone one level of wavelet decomposition. (In this case the wavelet used was the Haar wavelet.) The top left corner contains an approximation of the original image but at half the resolution. The other three quadrants contain the detail which has been removed from the approximation. It is clear that most of the detail lies along the edges of the original image. The top right quadrant contains information about the vertical edges, where as the horizontal edge detail is mainly contained in the lower left quadrant.

Further recursive decompositions are shown in figure 1.9.

In a similar way to the DCT the low frequency information has been moved towards the top left of the image, leaving the higher frequency information towards the bottom right.

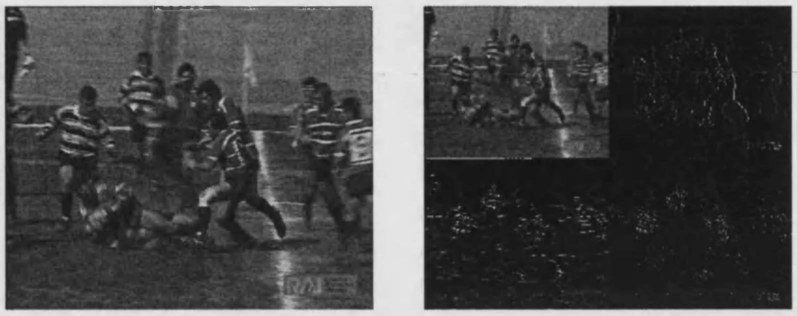


Figure 1.8: On the left is the original image, and on the right, the same image having undergone one level of wavelet decomposition.

Figure 1.9: Further wavelet decomposition of the image in figure 1.8.

1.7.3 Quantisation

During quantisation information is discarded from the original data; it is this stage which makes the compression lossy. Although there are more complicated quantisation schemes[23, 24], quantisation is usually achieved using integer division. Suppose we have a value x which we wish to quantise. Dividing x by a quantisation parameter Q and discarding the remainder would yield the quantised value y, (equation 1.8).

$$y = \mathrm{int}(\frac{x}{Q}) \tag{1.8}$$

In order to obtain an estimate of the original value we can simply multiply y by Q to obtain x_q, (equation 1.9).

$$x_q = y.Q \tag{1.9}$$

However x_q will always be a lower bound of the range of values in which x originally lay, so it is usual to employ the process of *half adjusting* to move this estimate to the middle of the range. This is done by adding half of Q to x_q,

$$x_h = x_q + \mathrm{int}(\frac{Q}{2}). \tag{1.10}$$

Embedded coding using a difference approach to quantisation. In an embedded stream *all* the original data is encoded, so that in its entirety the stream contains a losslessly compressed version of the original data. However, the stream is constructed in such a way that the most significant data is placed towards the beginning of the stream. As a result the more of the stream which is transmitted/stored the greater the fidelity of the reconstructed data. In this case quantisation is achieved by terminating the stream at a particular place. The earlier it is terminated the higher the compression, but the harsher quantisation results in poorer quality output. Shapiro's Zerotree [25, 26] is an excellent example of an embedded stream.

1.7.4 Entropy Encoding

In [27, 28] Shannon states that for an alphabet of N symbols in which each symbol appears with probability p_i, $i = 0, 1, \ldots, N-1$, the cost in bits per symbol H is given by,

$$H = -\sum_{i=0}^{N-1} p_i \log_2(p_i). \tag{1.11}$$

This value is also known as the entropy of the data and represents the statistical lower limit to the cost in bits required to encode the data.

In video encoding we often have data which is represented as a set of symbols from an alphabet. Entropy encoding techniques allow us to losslessly compress such data in such a way as we can approach the limit presented by equation 1.11.

Suppose we have an alphabet composed of the 8 symbols, $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}, \mathcal{G}$ and $\mathcal{H}$. These it is known that these symbols occur with the probabilities given in table 1.2.

Symbol	**Probability**
$\mathcal{A}$	1/32
$\mathcal{B}$	4/32
$\mathcal{C}$	11/32
$\mathcal{D}$	3/32
$\mathcal{E}$	3/32
$\mathcal{F}$	7/32
$\mathcal{G}$	2/32
$\mathcal{H}$	1/32

Table 1.2: The symbols in our alphabet and their corresponding probabilities.

Since $8 = 2^3$ each of these symbols can be encoded with a 3 bit code, so a message of length N would require $3N$ bits to encode. Let us assume we have a message which is 32 symbols long and contains each symbol in proportion to their probability. Using our naïve, 3 bit, coding system we would need $32 \times 3 = 96$ bits to encode our message. However, if we use the data from table 1.2 in equation 1.11 we find that the expected cost per symbol is in fact 2.59 bits. Using this we would expect our 32 symbol message to cost only 82.88 bits. Entropy encoders allow us to encode data in such a way as we can approach this theoretical cost.

1.7.4.1 Huffman Coding

Huffman Coding [29] provides a method for determining code words for symbols based on the frequency in which they appear in a message. It produces the most optimal encoding for situations where each symbol is encoded independently of the others.

Producing Huffman codes is a straightforward process and is best described by means of an example. Using our alphabet from table 1.2 we list our symbols in order of increasing probability.

Symbol	$\mathcal{A}$	$\mathcal{H}$	$\mathcal{G}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{B}$	$\mathcal{F}$	$\mathcal{C}$
Prob.	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{2}{32}$	$\frac{3}{32}$	$\frac{3}{32}$	$\frac{4}{32}$	$\frac{7}{32}$	$\frac{11}{32}$

We then combine the two symbols with the smallest probabilities, in this case $\mathcal{A}$ and $\mathcal{H}$, to produce an aggregate symbol $(\mathcal{AH})$ which is assigned the sum of the probabilities. The list is then resorted to ensure the new set of symbols is still in order of increasing probability. After this first step we have,

Symbol	$(\mathcal{AH})$	$\mathcal{G}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{B}$	$\mathcal{F}$	$\mathcal{C}$
Prob.	$\frac{2}{32}$	$\frac{2}{32}$	$\frac{3}{32}$	$\frac{3}{32}$	$\frac{4}{32}$	$\frac{7}{32}$	$\frac{11}{32}$

The process is then repeated, each time combining the two symbols with the lowest probabilities and resorting the list.

Symbol	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{B}$	$((\mathcal{AH})\mathcal{G})$	$\mathcal{F}$	$\mathcal{C}$
Prob.	$\frac{3}{32}$	$\frac{3}{32}$	$\frac{4}{32}$	$\frac{4}{32}$	$\frac{7}{32}$	$\frac{11}{32}$

Symbol	$\mathcal{B}$	$((\mathcal{AH})\mathcal{G})$	$(\mathcal{DE})$	$\mathcal{F}$	$\mathcal{C}$
Prob.	$\frac{4}{32}$	$\frac{4}{32}$	$\frac{6}{32}$	$\frac{7}{32}$	$\frac{11}{32}$

Symbol	$(\mathcal{DE})$	$\mathcal{F}$	$(\mathcal{B}((\mathcal{AH})\mathcal{G}))$	$\mathcal{C}$
Prob.	$\frac{6}{32}$	$\frac{7}{32}$	$\frac{8}{32}$	$\frac{11}{32}$

Symbol	$(\mathcal{B}((\mathcal{AH})\mathcal{G}))$	$\mathcal{C}$	$((\mathcal{DE})\mathcal{F})$
Prob.	$\frac{8}{32}$	$\frac{11}{32}$	$\frac{13}{32}$

Symbol	$((\mathcal{DE})\mathcal{F})$	$((\mathcal{B}((\mathcal{AH})\mathcal{G}))\mathcal{C})$
Prob.	$\frac{13}{32}$	$\frac{19}{32}$

Symbol	$(((\mathcal{DE})\mathcal{F})((\mathcal{B}((\mathcal{AH})\mathcal{G}))\mathcal{C}))$
Prob.	$\frac{32}{32}$

The final aggregate symbol $(((\mathcal{DE})\mathcal{F})((\mathcal{B}((\mathcal{AH})\mathcal{G}))\mathcal{C}))$ can be represented as a binary tree with the original symbols as the leaf nodes, and each aggregation resulting in an internal node, (figure 1.10). From this tree we can create variable length codes for each symbol by starting at the route of the tree and tracing a path

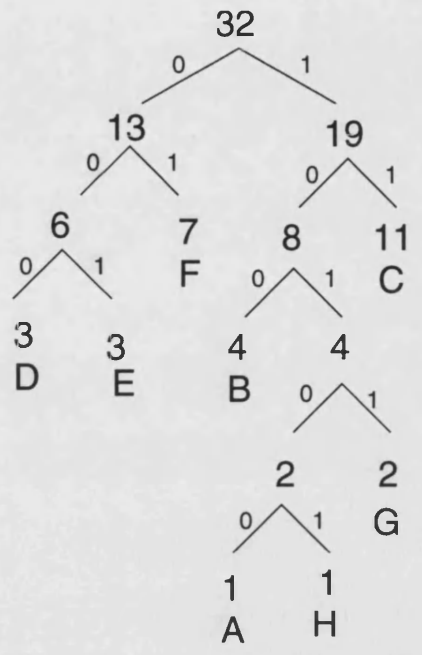


Figure 1.10: The Huffman tree for the aggregate $(((\mathcal{D}\mathcal{E})\mathcal{F})((\mathcal{B}((\mathcal{A}\mathcal{H})\mathcal{G}))\mathcal{C}))$.

Symbol	**Huffman Code**
$\mathcal{A}$	10100
$\mathcal{B}$	100
$\mathcal{C}$	11
$\mathcal{D}$	000
$\mathcal{E}$	001
$\mathcal{F}$	01
$\mathcal{G}$	1011
$\mathcal{H}$	10101

Table 1.3: The Huffman codes for the example alphabet given in table 1.2.

to the leaves. For each left hand branch we traverse we add a 0 bit to the code, and each right hand branch adds a 1 bit.

If we now use these codes to encode our 32 symbol message we find we only need 84 bits, which is much closer to theoretical entropy limit of 82.88 bits. Another interesting thing to notice about Huffman codes is that they have the *unique prefix* property. This means that no code is the prefix of a longer code. This is a great aid in decoding the data.

The problem with Huffman coding is that each symbol requires at least a 1 bit code. If we have an alphabet with two symbols we would still require N bits to encode a message of length N even if one of the symbols was much more frequent that the other. One of the solutions to this problem is to use arithmetic encoding.

1.7.4.2 Arithmetic Encoding

Arithmetic encoding [30, 31] maps messages of symbols from a given alphabet into a single floating point number which lies in the range $[0,1)$. Longer, more complicated messages require more accuracy in the floating point number, and therefore require more bits. To see how the encoding process works let us consider an alphabet of two symbols, $\mathcal{X}$ and $\mathcal{Y}$ which occur with probabilities 0.2 and 0.8 respectively. We can express these probabilities as intervals over the range $[0,1)$ as shown in figure 1.11(a).

Suppose we wish to encode $\mathcal{YXYY}$. The first symbol in this message is $\mathcal{Y}$ so we restrict our range to $[0.2, 1.0)$. We then subdivide this range using the same proportions as for our original range (figure 1.11(b)). Our second symbol is $\mathcal{X}$ so the range $[0.20, 0.36)$, is chosen to be subdivided further. Selecting $\mathcal{Y}$ twice more (figures 1.11(c) and (d)) leaves us with the range $[0.2576, 0.3600)$. Any number in this range will describe our intended message. It should be clear that each symbol added to the message causes the range to narrow. The narrower the range the more accuracy is required to describe a number within the range and therefore more bits are needed to encode the message.

Practical implementations of arithmetic encoding [30, 32] can be written which use integer arithmetic and produce output as a binary fraction. Such methods can approach the statistical bit limit even more closely than Huffman coding.

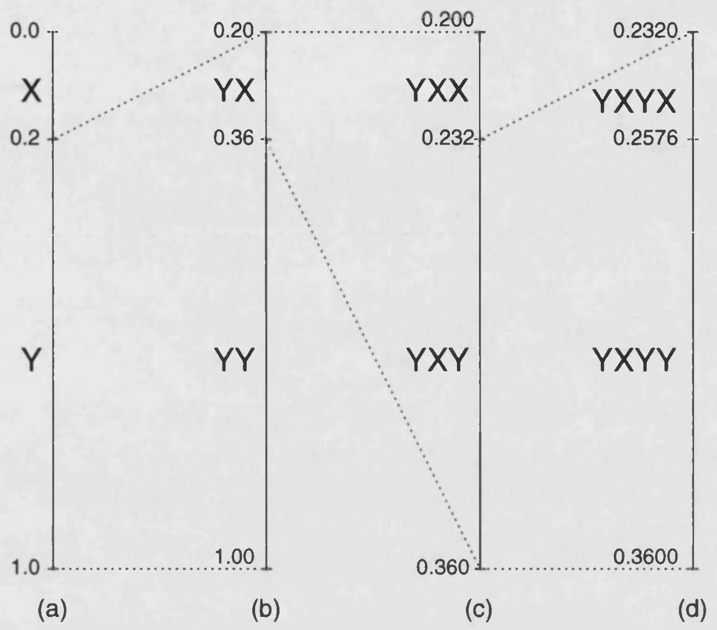


Figure 1.11: An example of arithmetic encoding.

1.8 Measuring Image Quality

In image processing we often want to numerically compare two images to see how similar (or different) they are. The comparison is normally performed on the luminance of the pixel values as this is usually the most dominant aspect of the perceived image.

Although there are many types of image quality measures[33] this work will concentrate on several commonly used ones. These give a numerical indication of the quality of the image using statistical means.

In the following equations $f(x,y)$ and $g(x,y)$ represent the value of the pixel at coordinates (x,y) in the images f and g respectively. The area to be compared has dimensions $M \times N$ pixels and is located at an offset of (i_f, j_f) into image f and (i_g, j_g) into image g.

The Mean Square Error (MSE) between two frames is defined as

$$\text{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x+i_f, y+j_f) - g(x+i_g, y+j_g)]^2. \qquad (1.12)$$

Another commonly used metric is the Root Mean Square Error (RMSE). This is easily defined in terms of the Mean Square Error (MSE).

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} \tag{1.13}$$

The Peak Signal-to-Noise Ratio (PSNR) of a signal with peak signal power $S_{\max}^2$ is

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{S_{\max}^2}{\mathrm{MSE}}\right). \tag{1.14}$$

In video coding, image planes are often 8 bits deep, giving a signal range from 0–255. As a result $S_{\max} = 255$ is often used in this context, giving

$$\begin{aligned}\mathrm{PSNR}_{255} &= 10\log_{10}\left(\frac{255^2}{\mathrm{MSE}}\right)\\ &= 20\log_{10}\left(\frac{255}{\mathrm{RMSE}}\right).\end{aligned} \tag{1.15}$$

In many video encoding applications comparisons between images, or parts of images, can account for a significant percentage of the total computation time. For this reason even small increases in efficiency can yield considerable performance increases. The Sum of Absolute Difference (SAD) metric only contains the computationally cheap *addition*, *subtraction* and *absolute value* operations and removes the need for the more expensive *multiply*. It is also an integer only metric which can lead to faster computation.

$$\mathrm{SAD} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathrm{abs}(f(x+i_f, y+j_f) - g(x+i_g, y+j_g)). \tag{1.16}$$

While the SAD is very useful in the computationally intensive core of compression algorithms its dependence on the block area makes it unsuitable as a general comparison metric. However, dividing by this area yields the Mean of Absolute Differences (MAD) which is a 'per-pixel' metric and therefore useful for more general comparisons.

$$\mathrm{MAD} = \frac{1}{MN}\mathrm{SAD}. \tag{1.17}$$

The Peak Signal-to-Noise Ratio (PSNR) is unique in the above metrics as it is the only one in which a ***higher*** value indicated a closer match. All the others are measures of error so lower values correspond to closer matches.

It is sometimes useful to have a visual representation of the differences between two images. We can create a difference image D by simply subtracting corresponding pixels from each image plane of our image pair f and g.

$$D(i,j) = f(i,j) - g(i,j) \forall (i,j) \in \Re, 0 \leqslant i < M, 0 \leqslant j < N. \tag{1.18}$$

Suppose we perform this action on the luminance planes of two images. Since the pixel values in each of these planes lie between 0 and 255, the values in the resulting difference image will lie between -255 and 255. In order to visualise this image we need to overcome two problems.

1. The dynamic range requires more than the 8-bits provided by standard grey-level image formats.

2. Some of the pixels have negative values.

If we take the absolute value of all the differences we solve both of these problems,

$$D_{\text{abs}}(i,j) = \text{abs}(D(i,j)). \tag{1.19}$$

This is convenient if we are just interested in the magnitude of the difference. However, this method fails to preserve the *direction* of the difference. We no longer know how image f changed to produce image g. We can use an alternative transformation to preserve this information.

$$D_{\text{mag}}(i,j) = \text{int}\left(\frac{D(i,j)+256}{2}\right) \tag{1.20}$$

Here we first make sure all values are positive by adding 256 and then reduce the dynamic range by dividing by two (discarding any remainder). The disadvantage of this approach is that we have lost some of our ability to discriminate between different levels of difference. It is also interesting to note that if using equation 1.19 zero difference is represented by 0 (black) whereas in equation 1.20 is becomes 128 (mid-grey).

Figure 1.12 shows two images from the Carphone sequence. Subtracting the two images using equation 1.18 and applying both the above visualisation approaches gives us the difference images shown in figure 1.13.

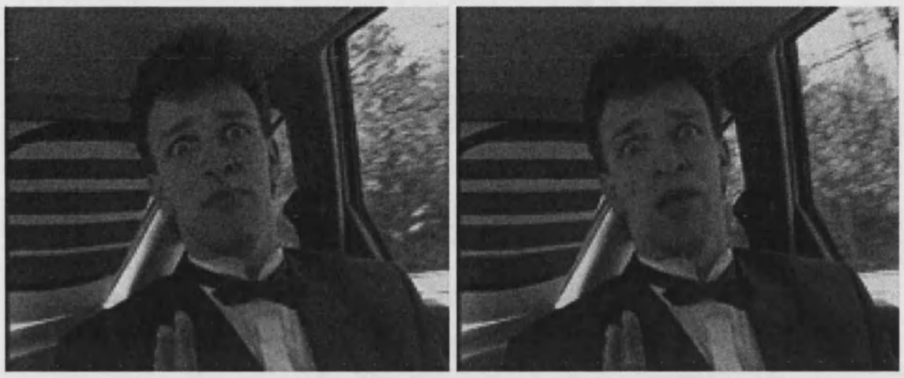


Figure 1.12: Two frames from Carphone sequence (frames 220 and 224). The interior of the car is stationary and the actor moves very slightly. The scenery passing the window accounts for most of the motion between these two images.

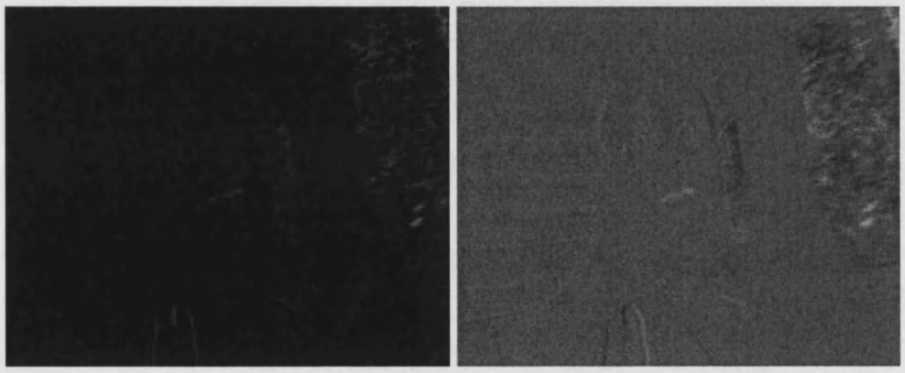


Figure 1.13: Visualisation of the absolute difference on the left, and with the magnitude preserved on the right. These images represent a PSNR of 25.41. The differences caused by the moving scenery are clearly visible in each case.

1.9 Temporal Redundancy in Digital Image Sequences

At its most basic level encoding video sequences can involve no more than simply encoding a series of separate images using a still image codec such as JPEG. In video coding a frame which is encoded without any reference to other frames is known as an *INTRA* frame, or I-frame.

Using only I-frames to encode a sequence fails to capitalise on the temporal redundancy which can yield huge gains in compression if properly exploited. One solution is to try and exploit this temporal redundancy by using a previously transmitted image to predict the image we are trying to encode. We can measure the quality of our prediction by examining the differences between the prediction and the frame it is supposed to predict.

Consider the two frames of the Salesman sequence shown in figure 1.14. They look remarkably similar; in fact there is no perceptible change in the background at all and even the salesman himself has hardly moved.

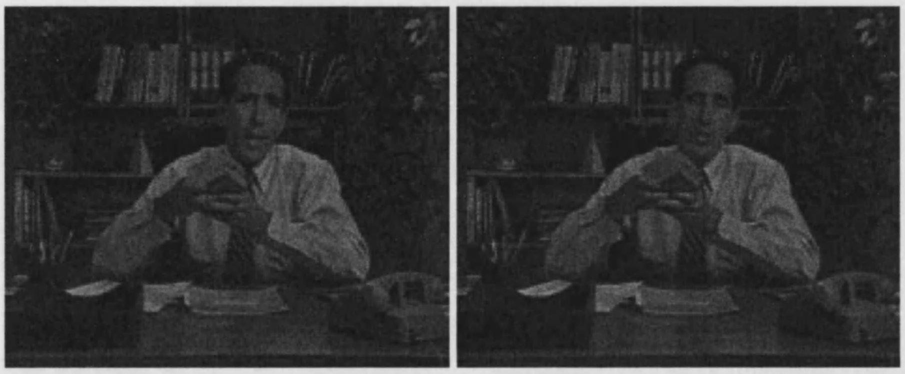
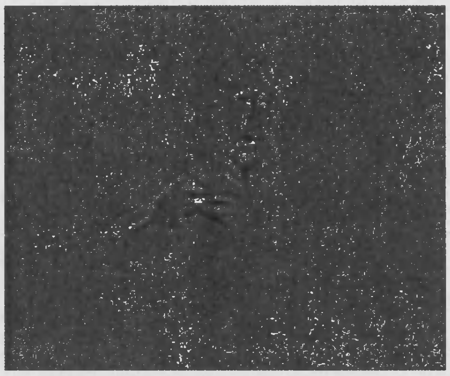


Figure 1.14: Two frames from Salesman sequence (frames 20 and 24). The background remains static and there is very little movement in the foreground.

Suppose we simply use the first image as the prediction for the second one. The error in our prediction is defined as the difference between our predicted image and the actual second image. This error is shown in figure 1.15, both visually and numerically. From this figure we can see that the prediction was very good indeed. The MAD of 2.97 implies that on average each predicted pixel is less than three greylevels away from its actual value.

What happens if we apply our simple prediction scheme to a a pair of frames containing more complicated motion, such as those in figure 1.16? In this case the results (figure 1.17) are a lot less satisfactory.

The problem with the Rugby sequence is that there is much more movement than in the Salesman sequence. Just using the first frame as the prediction has failed to capitalise on the redundancy between the two frames. We need a method which can form a better prediction which can take into account the movement which has occurred between the two frames.



MAD = 2.97
RMSE = 6.69
PSNR = 31.62

Figure 1.15: The difference between the two frames in figure 1.14.

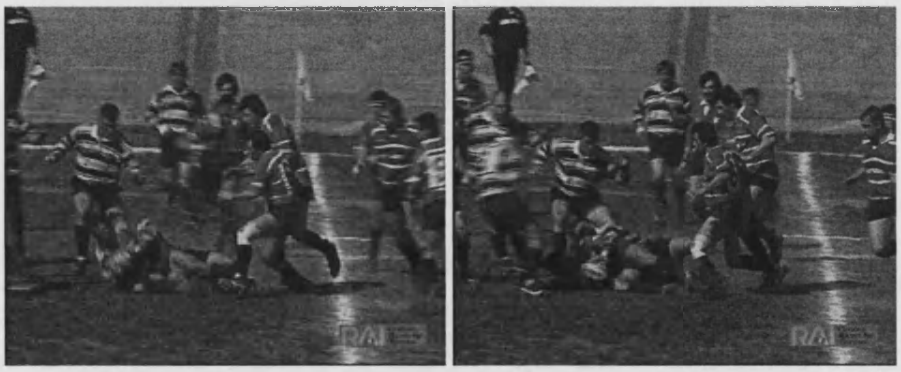
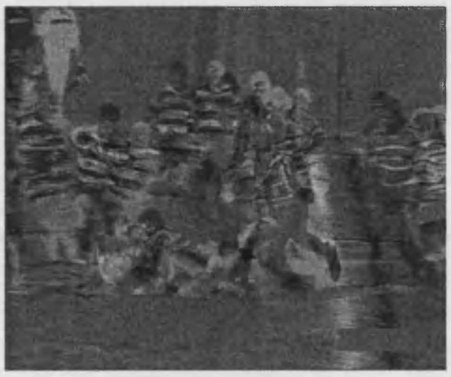


Figure 1.16: Two frames from Rugby sequence (frames 0 and 4). The shot has panned to the left a little, and of course, the players have moved.

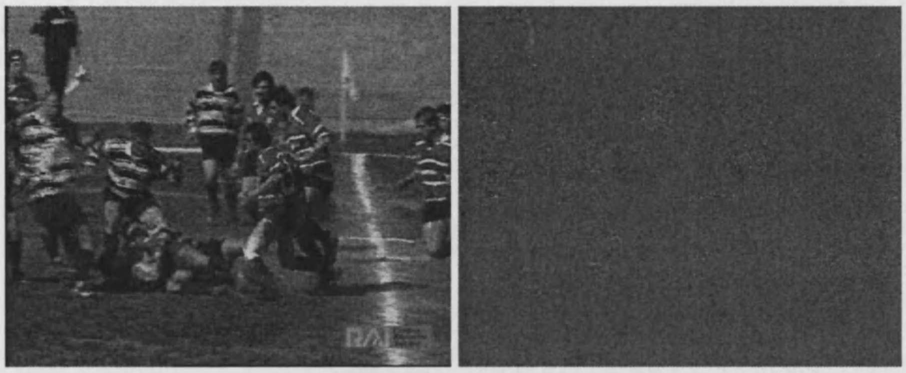


MAD = 30.66
RMSE = 43.92
PSNR = 15.28

Figure 1.17: The difference between the two frames in figure 1.16.

Suppose we devise a technique which when presented with two input frames, attempts to determine the motion which occurs between them. This *motion description* could then be applied to the first frame to produce a prediction of the second. This process is called *motion estimation* and is one of the key concepts in this thesis.

Figure 1.18 shows such a predicted image produced by a motion estimation technique which will be introduced later[1]. This visual representation indicates that prediction is much better than the one which produced figure 1.17. This is borne out by the numerical comparisons as well.



MAD = 4.94
RMSE = 9.54
PSNR = 28.54

Figure 1.18: On the left is the predicted image which was generated by applying a motion description to the first frame from figure 1.16. The error in this prediction is shown on the right and shows it performs much better than the non-motion compensated prediction technique.

Now that the concepts of image prediction have been explained we can review and formalise some of our definitions. The frame we are trying to predict is called the *input* or *current* image. The image from which the prediction is formed is known as the *reference* image. These two images act as the inputs to a *motion estimator* which produced a *motion description.* The motion description can then be applied to the reference image to produce the *predicted image*, this is known as *motion compensation.* Finally, the difference between the current image and the predicted image is known as the *residual* and is an indication of the accuracy of the

[1]For the curious, the method used was the ESA with a block size of 4×4 and a search range of 32×32.

prediction. This process and nomenclature is summarised in figure 1.19.

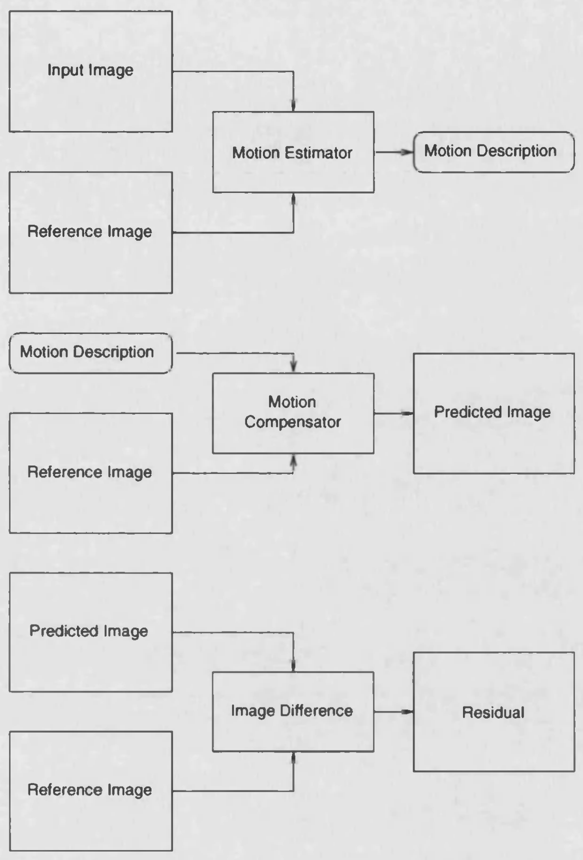


Figure 1.19: The image prediction process.

The combination of the motion description and the residual together make an *INTER* frame which is also known as a Predicted frame, (or P-frame). Due to the fact that they are able to take advantage of temporal redundancy P-frame are more efficient at coding digital video than I-frames.

A typical combination of I and P-frames is shown in figure 1.20. Here an I-frame is followed by three P-frames, each predicted from the temporally previous frame. The periodic insertion of I-frames ensures that any errors introduced by the transmission process are not allowed to propagate indefinitely through the sequence. It is also useful to insert I-frames when a scene change has occurred, as there will be no correlation between the reference frame and current frame, and the prediction process will not produce meaningful results.

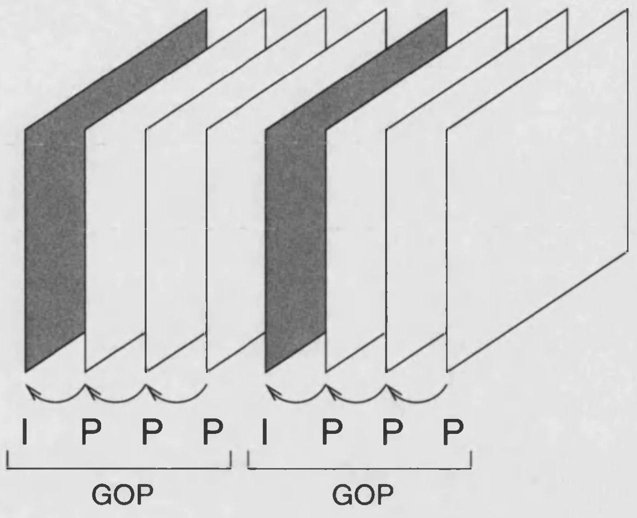


Figure 1.20: A typical ordering of I and P-frames in a compressed video sequence. The arrows indicate which frame was used as the reference frame for each of the P-frames.

The series of frames IPPP form a unit which can be decoded without reference to external data. Such a unit is often referred to as a Group Of Pictures, or GOP[8].

Some video codecs use a third type of frame called a Bidirectional frame, or B-frame. These are predicted from a pair of I or P-frames, one of which comes before the B-frame and one after, (see figure 1.21. Due to this B-frames are not transmitted in strict temporal order, as they must be sent after both frames upon which they depend. However this thesis will concentrate on image prediction using P-frames.

In some cases it is appropriate to use a single I-frame followed by P-frames until a scene change occurs. This produces better results when transmission errors are very unlikely to occur, but it does have the drawback that the sequence must always be decoded from the initial I-frame even if only a portion of the frames are of interest.

Figure 1.22 shows the process involved in transmitting a P-frame from the encoder to the decoder. One key thing to note from this diagram is that a P-frame is transmitted in two 'parts'; the motion description and the residual. As a result the bit cost for a P-frame is,

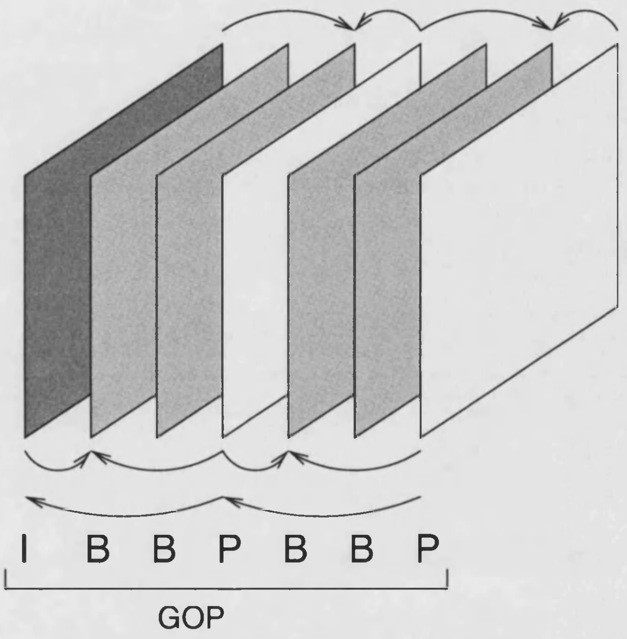


Figure 1.21: A typical ordering of I P, and B-frames in a compressed video sequence.

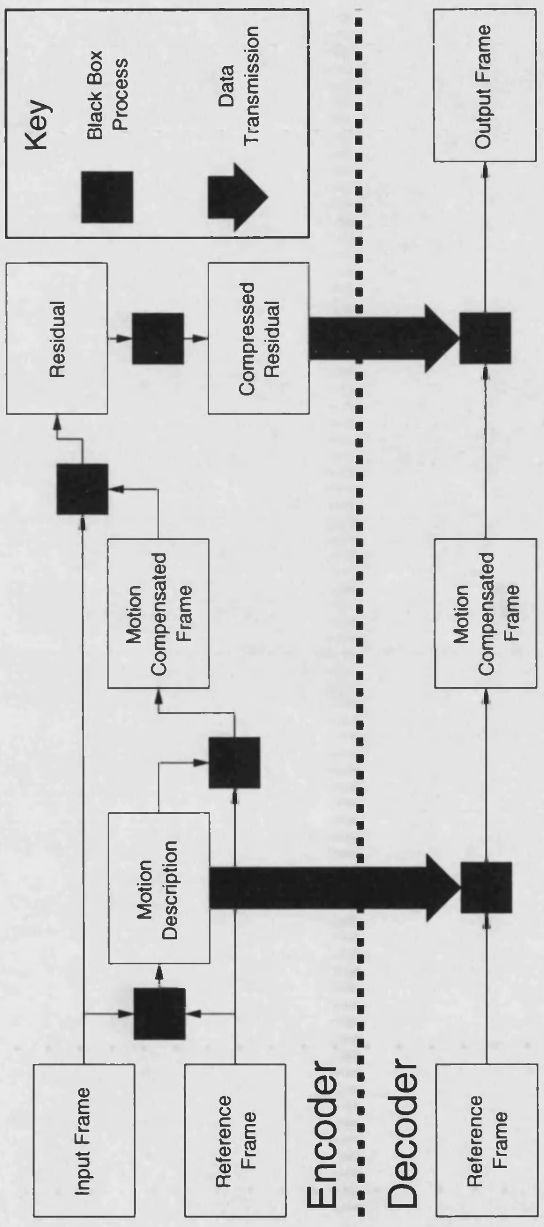


Figure 1.22: The process involved in encoding a P-frame.

$$B_{\text{total}} = B_{\text{md}} + B_{\text{res}}, \tag{1.21}$$

where B_{total} is the total number of bits in the P-frame, B_{md} is the number of bits required for the motion description and B_{res} the number of bits used to encode the residual. The motion description is compressed using lossless methods, so in general the total number of bits is altered by adjusting the quality of the residual which is compressed in a lossy manner.

In the case where B_{total} is fixed there is a trade off between B_{md} and B_{res}. The more bits allocated to one reduces the number of bits available to the other. For example, increasing the number of bits given to the motion description should improve its accuracy, leading to a less complicated residual to encode. This interaction between the motion description and residual is very interesting and will be explored in greater detail later in this thesis.

Chapter 2

Motion Estimation in Digital Video

In this chapter we will introduce the idea of optical flow and how it is used in digital video encoding. We shall explore some of the techniques used to produce motion descriptions (motion vector fields in particular) and investigate the advantages and problems of different approaches.

2.1 Techniques for Measuring Optical Flow

Measurement of optical flow (also known as image velocity) involves mapping the 3d velocities of objects in the image into the 2d image domain. According to [34] techniques for measuring optical flow can be divided into 4 broad categories;

Differential techniques These methods use derivatives of the image intensity to calculate the image velocity [35]. Such techniques tend to assume that the intensity of a point does not change as it moves through the scene. They also rely on the fact that the intensity field is differentiable. For this reason the image is often filtered (smoothed) before the intensity gradients are calculated.

Region based matching techniques This class of optical flow measurement divides the image into distinct regions. The velocity of a region is defined as the vector which yields the best fit (according to some metric) between different image regions at different times.

Energy based methods As an object travels through space, its motion can be described as spatio-temporal tilt [36]. Fourier techniques using velocity tuned filters can be used to measure the energy of these contours and thus derive the motion of the object.

Phase based techniques These methods find the optical flow by analysing the phase behaviour of band pass filter outputs. The general approach requires correlation surfaces to be found for a pair of temporally adjacent image regions, (usually the luminance plane of the image is used to generate the surface). The peaks in the correlation surface would indicate potential motion vectors for objects within the regions [37]. These vector could then be used as candidate vectors for region based matching algorithms and applied to various sub-regions (even down to the pixel level).

Interpolation of the correlation surface, along with using curve fitting techniques to establish the location of the peaks, allow this method to produce sub-pixel motion vectors.

2.2 Motion Estimation in Digital Video

Of the four methods mentioned above, region based matching techniques are by far the most prevalent for determining optical flow for use with digital video compression. In such cases the general regions are usually replaced by a more specific two-dimensional array of blocks; as a result the methods are also known as Block Matching Algorithms.

The strengths of such methods lie in their compatibility with the way motion descriptions are used in digital video compression. In nearly all cases one of the main aims is to reduce the power in the residual image as much as possible. This is equivalent to using the reference image to produce the best prediction of the image to be transmitted. Block matching techniques produce a motion description based on this image difference and are therefore inherently coupled with the goals of video compression. Other methods may well produce more accurate description of motion within the image, but at the expense of increasing residual power, which is not beneficial to the compression process.

Phase based techniques are also used [38], but often as a precursor to a block matching algorithm, usually to identify a suitable set of candidate vectors.

Finally it is worth mentioning warping techniques[refs]. These treat the original image as a rubber sheet and distort or *warp* it by moving control points on the image.

2.3 Block Matching in Digital Video

A block is just a particular region of an image. In order to describe a block we need two pairs of values. The first of these are the block dimensions (M, N). In most current video encoding applications these are usually equal (the block is square) and often take the value of 2^N with 4, 8, 16 and 32 being the most common values. The second pair of values specify where in the image the block is. This offset is given as the displacement of the top left hand corner of the block from the image origin and is denoted (i, j), (figure 2.1).

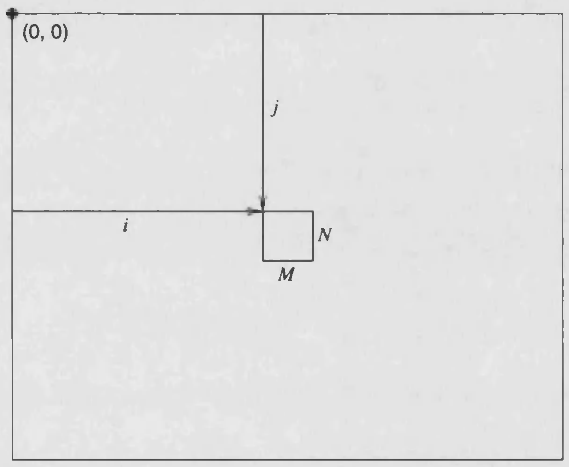


Figure 2.1: A block is defined by four values, the block dimensions (M, N) and its offset (i, j).

Suppose we start with an empty predicted image and split it into a regular array of blocks as shown in figure 2.2. We have to ensure that out block size divides exactly into the image dimensions so that we can cover the entire image in (non-overlapping blocks). We can then construct the predicted image by 'filling in' the empty blocks in our array with blocks from the reference frame. We would like to chose the blocks which result in the best predicted image. Choosing these blocks is the job of a *block matching algorithm.* The displacement between the block in the predicted image and the selected block in the reference frame is known as a *motion* *vector*. For example, if the offset of the block in the predicted frame is $(16, 32)$ and the block selected from the reference frame is at $(22, 24)$ then the motion vector (v) would be

$$\begin{aligned} v &= (22-16, 24-32) \\ &= (6, -8) \end{aligned}$$

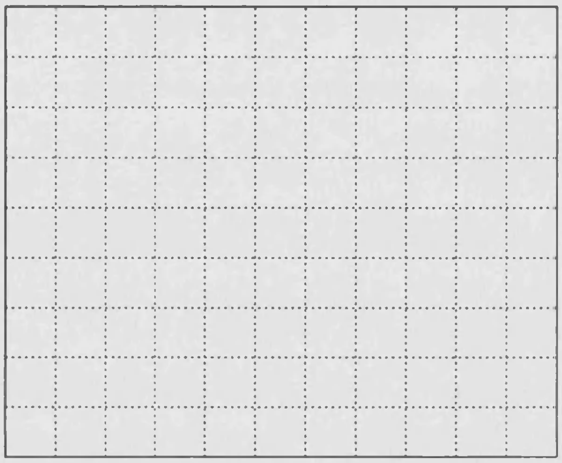


Figure 2.2: Block matching algorithms split the predicted image up into a regular array of blocks. These are then 'filled in' using blocks from the reference frame.

It follows that each block can have its own motion vector and it is this array of vectors which make up the *motion vector field*. A typical motion vector field is shown in figure 2.3.

For each block in the predicted image there is a motion vector (or perhaps vectors) which will minimise the prediction error for that block according to a specified metric. This is known as the optimal vector for that block. If every block in the predicted image has its optimal vector then the resulting motion vector field is also optimal[1].

One possible modification to the basic Block Matching Algorithm (BMA) is *Longterm* (or *Multi-reference*) block matching [39]. This enhances the technique described above by providing a number of different reference frames from which

[1]It is important to note that optimal fields generated using different error metrics are unlikely to be the same.

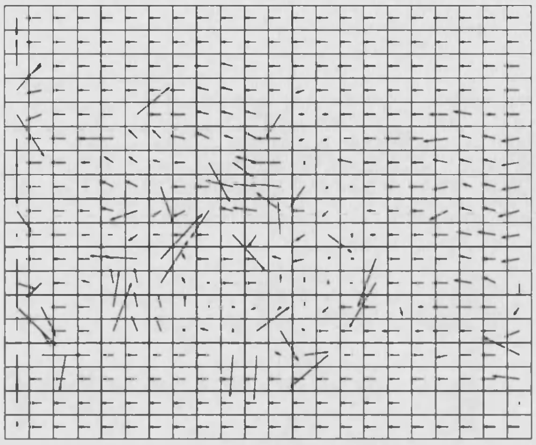


Figure 2.3: A typical motion vector field. Each image block is assigned its own motion vector.

to choose the most appropriate block. This has the great advantage of being able to more easily deal with occlusions which occur between frames.

Another issue with the basic BMA is that it can only produce motion vectors accurate to the nearest pixel. This can cause problems in sequences where the motion doesn't lie on pixel boundaries, (which is the case for most sequences). The solution to this is to perform block matching to *sub-pixel* accuracy. This involves interpolating the image before carrying out the BMA. It is worth noting that this leads to an increase in the computational cost of most algorithms. To work at half-pixel accuracy the number of pixels is increased fourfold with the equivalent increase in computation time.

One solution to this problem is to first perform a traditional full pixel motion description. Each motion vector in the description is then refined by considering only the sub-pixel positions surrounding this vector. In the case of half-pixel accuracy this would only involve an additional eight comparisons per motion block.

2.4 The ESA

The Exhaustive Search Algorithm (ESA)[40] will produce the optimal vector field for the current image from a given reference image, (for a specified search region). Each block in the search region is compared with the block from the current image and the best one is selected. As each candidate block needs to be tested this method is very computationally intensive.

The Exhaustive Search Algorithm (ESA) uses a brute force method to achieve an optimal block matching result. The first stage in the process is to split the input image into an array of rectangular blocks. For each of these blocks a set of candidate blocks is obtained from the reference frame. The larger this set, the greater the chance of finding a good match, but at the cost of increased computation. For this reason the candidate blocks are usually restricted to the locality of the input block, (see figure 2.4). This is a fair restriction to make when the motion between the input and reference frames is small.

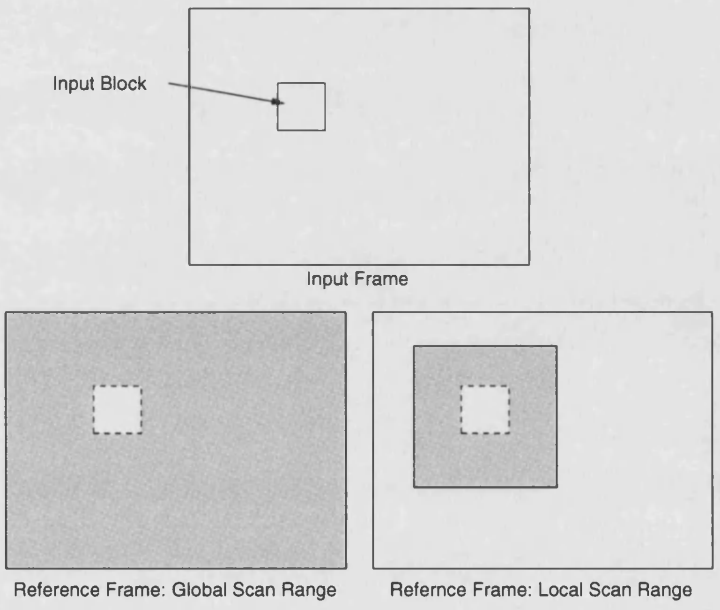

Input Block
Input Frame
Reference Frame: Global Scan Range
Refernce Frame: Local Scan Range

Figure 2.4: For a given input block it is unusual to use the entire reference frame for the candidate blocks, (this is due to the associated increase in computational load). More often the candidate blocks come from a region local to the input block.

The first candidate block is used as the initial guess for the best matching block, and the corresponding vector used as the best vector. The quality of this match is then established using a comparison metric such as RMSE, or SAD. Next, the input block is compared to each of the remaining candidate blocks in turn (using the same metric). The order in which the candidate blocks are processed is only important if two of them produce the same, most favourable metric. In most cases raster scan order is used. If the quality of the match is better than the current best match then the block becomes the new current best match. In the case of a better match the best vector is also updated.

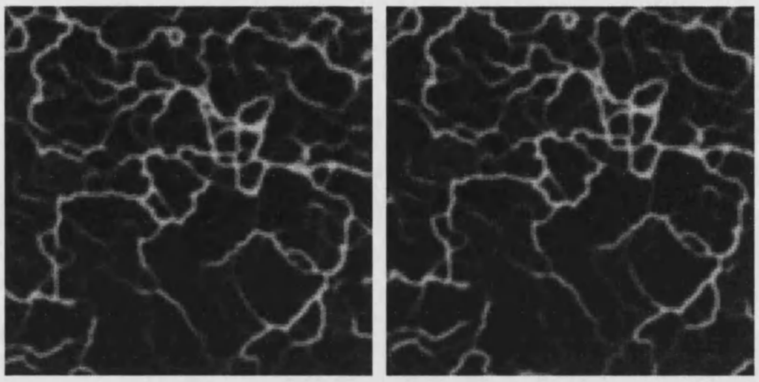
When all the candidate blocks have been considered the resulting best vector is assigned to the block from the input frame.

The pseudo-code below represents this algorithm.

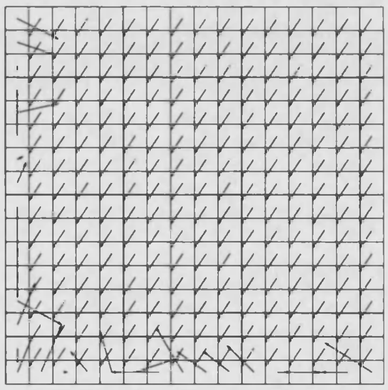
```
split input frame into blocks
for each block in the input frame
  choose candidate blocks from reference frame

  best match = first candidate block
  best vector = vector from input block to candidate block

  for each remaining candidate block
    compare blocks using metric
    if input block is a better than current best match
      best match = candidate block
      best vector = vector from input block to candidate block
    end if
  end for
end for
```

Having explained the ESA it is possible to look at some of the issues it raises. Consider the case where the only motion in the image it due to the camera tracking. This should lead to a motion field in which all the block are assigned the same motion vector. However, if this were the case then some of the edge blocks would need to reference source block which would lie partly *outside the image*. There are two solutions to this problem. The first is to simply prohibit blocks from having vectors which point outside the image. This approach is straightforward to implement, but can lead to some less than optimal results. Consider the two synthetic images shown in figure 2.5a which have been displaced with respect to one another. The vector field (as found by the ESA) is shown in figure 2.5b.

The field is almost entirely uniform except for the vectors along the bottom and left edges. As these are unable to take the same vector as the rest of the blocks as this would require them to reference vectors outside the image. As a result they must find a block within the image which produces the best match. This is



(a) Original Images



(b) Motion Vector Field

Figure 2.5: Motion vectors at the edge of the image.

unfortunate, as the inability to reference even a few pixels outside the image means that the block cannot chose a vector for which the majority of pixels would produce a perfect match. This is clearly unsatisfactory, and is resolved by assigning artificial values to pixels outside the image using a process called *padding*.

2.5 Padding

Padding allows us to extend the image beyond its boundaries by adding extra pixels. In video coding padding is most useful when the padded pixels are representative of those near the padded edge. We will briefly describe three of the more common methods used to pad an image, zero padding, extrusion padding and symmetric padding.

2.5.1 Zero Padding

One of the most simple ways to pad and image is to surround it with pixels with the value 0. This method is fast but in most cases the padded pixels will bear no relationship to the pixels within the image and as a result are unlikely to match. An example of zero padding is given in figure 2.6 (a).

2.5.2 Extrusion Padding

In this case the edge pixels are extruded away from the edge to determine the pixel values in the padded area. Corner regions are filled with the value of the corner pixel. An example of extrusion padding is given in figure 2.6 (b). There are many different ways to determine the value of the pixels to be extruded. One simple methods is just to use the value at the image edge. However, it is often possible to get a more representative pixel by choosing the median pixel from a group of pixels close to the edge, (this of course incurs additional computational costs).

2.5.3 Symmetric Padding

Symmetric padding involves filling the padded are with pixels reflected from the image. There are two ways of doing this; one with the edge pixels repeated and one where the edge pixels are not repeated. Examples are shown in figures 2.6 (c) and (d).

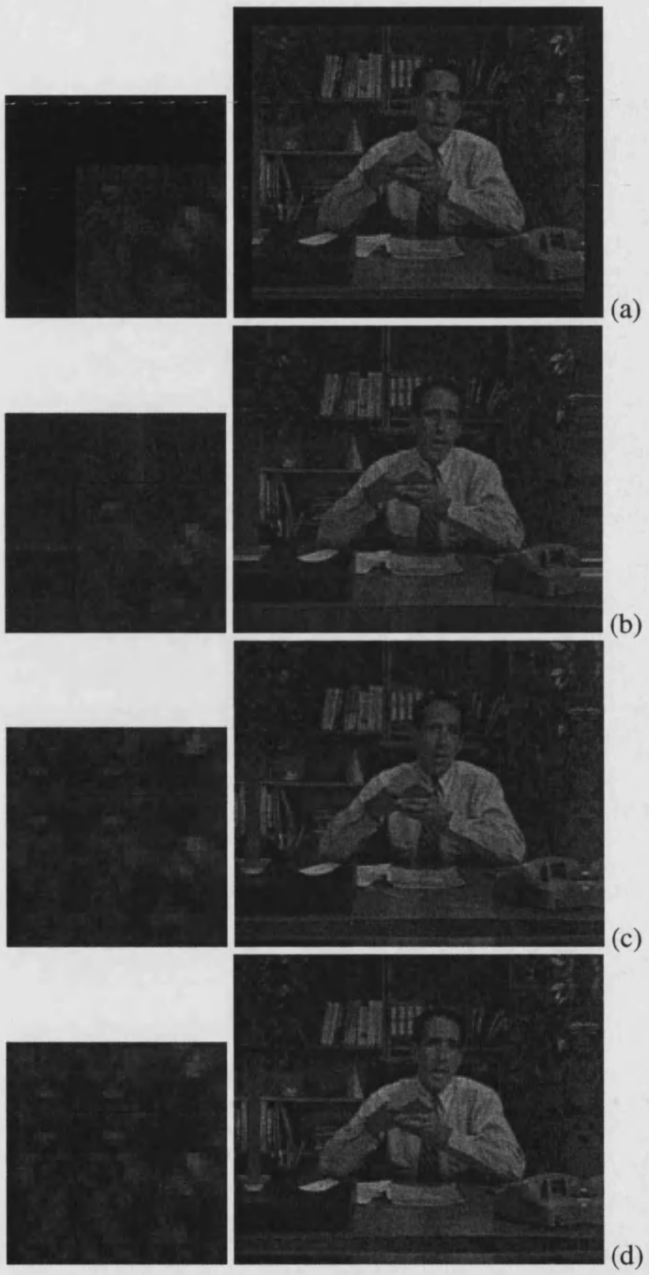


Figure 2.6: Figure (a) shows an example of zero-padding, (b) extrusion padding, (c) even-symmetric padding and (d) odd-symmetric padding. In each case the top-left corner is shown enlarged.

2.5.4 Comparing Padding Methods

The different padding method each have different levels of complexity and produce different results. Zero padding is the least complex but makes no attempt to correlate the extended region with the image itself. A slightly better approach is to pad the image with a 50% grey colour which is statistically more likely to be 'representative' of the image.

Extrusion padding is more complex than zero padding, but has the large advantage creating an extension which related to image content. It can cause problems due if the stripy pattern it produces intrudes into the image (during motion compensation) and is not adequately corrected by the residual.

Both types of symmetric padding have the advantage that they duplicate *texture* into the extended region. This reduces the chance that artifacts will be introduced when the extended region is copied into the image during motion compensation.

2.6 The Computational Expense of Block Matching

One of the main problems with the ESA as a block matching technique is that it is incredibly computationally intensive. Consider a CIF size image (352 $\times$ 288) divided into (16 $\times$ 16) blocks. There would be $22 \times 18 = 396$ such blocks in the image and comparing a single pair of blocks requires $16 \times 16 = 256$ pixel comparison calculations. Even a modest search range of ± 8 pixels at full pixel resolution gives $17 \times 17 = 289$ possible candidate vectors for each block. Combining these gives $396 \times 256 \times 289 \approx 30$ million pixel comparison calculations. It is also important to remember that for real-time encoding this process needs to be carried out up to 30 times a second, and that it is only *part* of the encoding process. It is clear that reducing the computational cost of block matching would be beneficial, allowing encoding to be carried out faster, by less powerful computers.

The three contributing factors to the total cost were the number of blocks, the block size and the number of candidate vectors. There are approaches which try to reduce each of these values.

One way to reduce the first value (number of blocks in the image) would be to increase the size of each block. However, this would lead to a proportional increase in the number of pixels per block leaving the total pixel comparison calculations unchanged. Instead some techniques find the exact motion vector for only a subset of the blocks [41, 42], (for example every other block as shown in figure 2.7). The vectors for the remaining blocks can then be inferred from the surrounding blocks using an averaging technique.

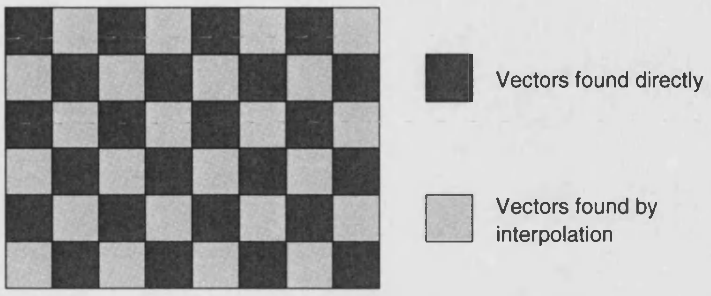


Figure 2.7: To reduce computational cost vectors are only found for every other block.

Another way to reduce the computational load is to match fewer pixels within each block [43]. As with the blocks in the image, one sensible approach would be to use every other pixel. This takes advantage of the fact that pixels will be locally correlated.

One final method would be to perform the matching algorithm on a scaled down version of the image, (for example by reducing a (352×288) CIF sized image to a (176×144) QCIF sized one). Although this can once again provide reduced computational cost, it does to at the expense of the accuracy of the motion vectors obtained. However, this method can be extended to produce a whole range of hierarchical methods [44, 45].

However, by far the most common method is to try and reduce the number of candidate vectors, and hence the number of matches performed. The following four approaches, (The Three Step Search, Four Step Search, Diamond Search and Circular Zonal Search) all try to achieve this in different ways.

2.7 The Three Step Search

The Three Step Search (TSS) [46] is a sub-optimal motion estimator which uses a coarse to fine approach. As the name suggests the search is performed in three steps. The search patterns for each of the steps is shown in figure 2.8. For the first step the pattern is centred on the $(0, 0)$ vector and the a match is evaluated at each of the nine indicated positions. The position which produces the most favourable match is used as the centre for the second step of the search. Once again each of the nine positions indicated by the step 2 pattern is considered[2]. Once again picking the most favourable match yields the centre for the third and final pattern of matches. The result of this match gives us the final motion vector.

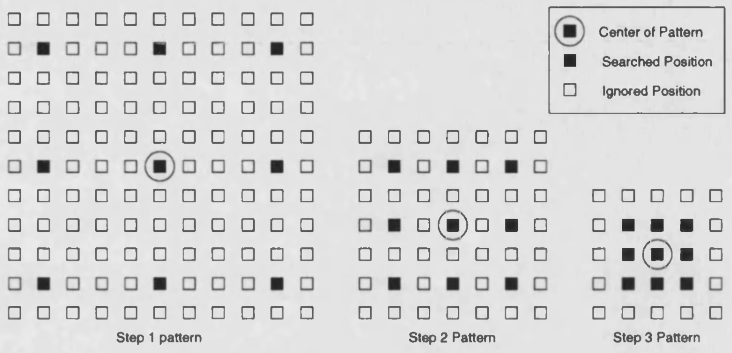


Figure 2.8: The search pattern used in the Three Step Search. The patterns become successively finer in order to 'home in' on the motion vector.

Figure 2.9 shows an example of the Three Step Search (TSS) in operation. In this example the first step indicates that the position with vector $(4,4)$ gives the best match. After the second step the centre has moved to vector $(6,4)$ and the final step produced the motion vector $(7,3)$. The dotted box indicates the total possible extent of the search, which is $(\pm7,\pm7)$ giving 225 difference possible matches. However, the TSS has only had to evaluate 25 of these in order to produce a motion vector. As a result the TSS is almost ten times as fast as the ESA. Due to the coarse to fine nature of the search

2.8 The Four Step Search

The Four Step Search (FSS) [47] uses a similar method to the TSS to find a predicted block within a $(\pm7,\pm7)$ search window. Although there are four steps, there are only two distinct search patterns which are used; these are shown in figure 2.10.

The procedure is best described in terms of pseudo code;

```
set the pattern centre to the (0, 0) vector
```

[2]The centre of the pattern is still considered although there is no need to recalculate the error metric.

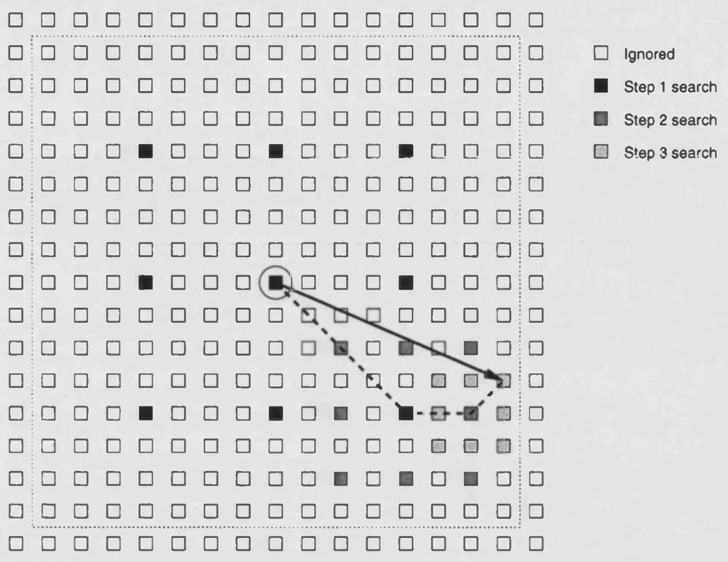


Figure 2.9: An example application of the TSS. Only 25 of the possible 225 search positions are evaluated in order to find the motion vector.

```
for i=1 to 3
  Apply pattern A at the choose a new centre based on
  the most favourable match
  if the new centre is the same as the old centre
    skip to label ::final::
  end if
end for

::final::
Apply pattern B. The motion vector is given by the most
favourable match.
```

As the above code shows pattern A is applied between one and three times, and is then followed by the application of pattern B. Although pattern A contains nine positions, the second and third applications of these patterns will overlap somewhat with previously searched positions. As a result each such application requires a maximum of five additional matches to be performed.

The total number of matches lies between 17 and 27, the exact number depends upon the location of the final best match, with smaller vectors requiring fewer matches. This technique seems more appropriate for sequences when the magnitude of the motion is small.

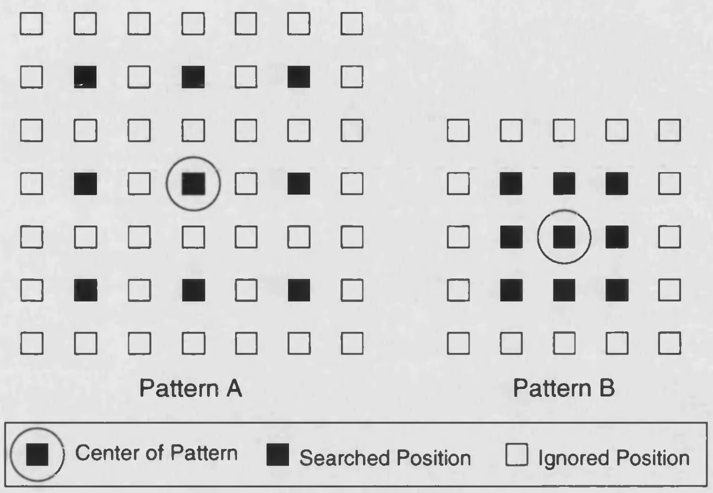


Figure 2.10: The two search patterns uses in the Four Step Search.

2.9 The Diamond Search Algorithm

The Diamond Search Algorithm (DSA)[48, 49] can be used to quickly produce an approximation to the optimal motion field. It makes the assumption that most motion vectors are small, and are therefore close to the $(0, 0)$ vector.

The Diamond Search Algorithm (DSA) uses two search patterns which are shown in figure 2.12. Initially search pattern 1 is applied with the centre at $(0, 0)$ and the motion vectors for each of the nine search positions are evaluated. If the minimum is found to be at the centre of the pattern, pattern 2 is then used, and the resulting best vector is selected to be used. If however the best vector did not lie at the centre of pattern 1, the pattern is reapplied, with the centre shifted to the recently located minimum. This process of shifting pattern 1 is repeated until the minimum is found to lie at the centre, at which point search pattern 2 is used to make the final choice of vector.

Tourapis[50] showed that the performance of the DSA could be improved by predicting a possible value of the motion vector based on the surrounding vectors, (see section A.1). Instead of placing the initial search pattern 1 at the origin, it was placed using this predicted vector. If the central point provided the minimum,

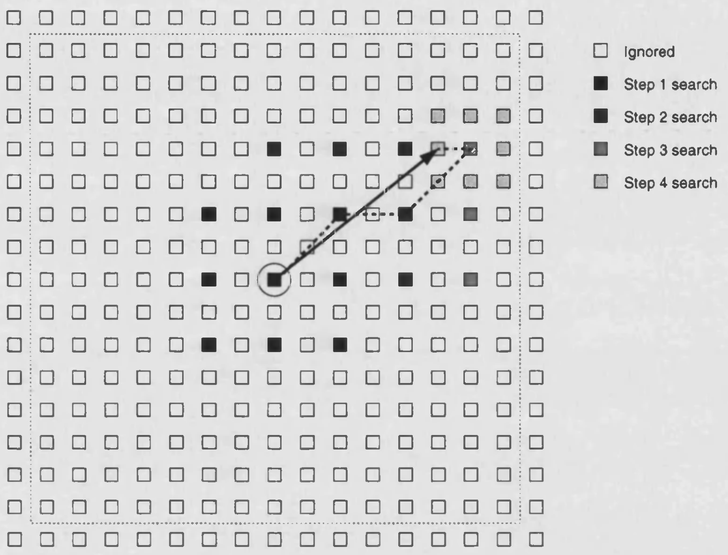


Figure 2.11: An example application of the FSS. Only 25 of the possible 225 search positions are evaluated in order to find the motion vector. Unlike the TSS the number of matches evaluated is not fixed and can range from 17 to 27.

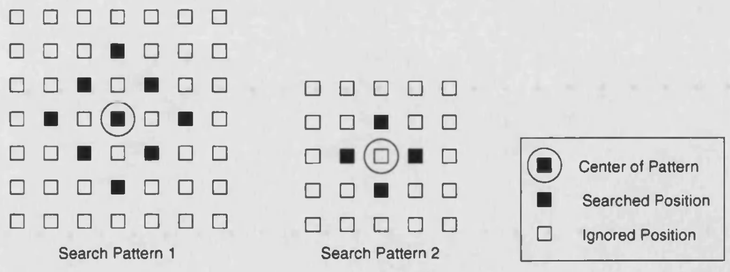


Figure 2.12: The search patterns used by the Diamond Search Algorithm.

pattern 2 was applied there to give the final result. If however, the initial application of pattern 1 produced a minimum around the edge of the pattern the next step would be to apply pattern 1 at the origin.

Evaluation of this pattern would lead to two cases. In the first case, the overall minimum would be found in the pattern centred at the origin. In this case the algorithm would proceed as in the original DSA. If however the overall minimum was found to lie within the original pattern (placed at the predicted vector) pattern 2 is immediately used around this point to produce the final vector.

Tourapis concludes that this method outperforms the original DSA (and other similar techniques) especially in sequences with a lot of motion (where more benefit would be derived from the predictive step).

2.10 The Circular Zonal Search Algorithm

The Circular Zonal Search (CZS)[51] algorithm uses a series of concentric *zones* around a central point. The first six zones are illustrated in figure 2.13. These and further zones can be created using the following formula,

$$r = \text{round}(\sqrt{\delta_x^2 + \delta_y^2}) + 1, \tag{2.1}$$

where r is the zone index, and δ_x, δ_y represent the horizontal and vertical distances from the zone centre respectively. The round function rounds to the nearest integer.

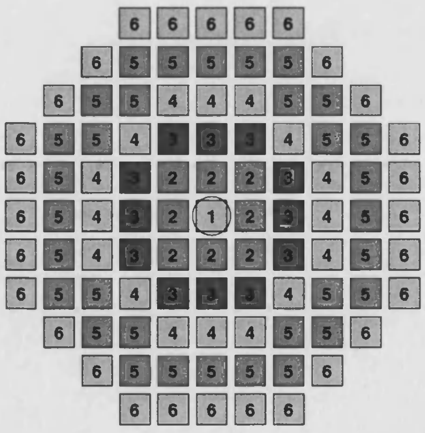


Figure 2.13: The first six zones used by the Circular Zonal Search.

These zones are centred around a predicted vector, which is taken from the block to the left of the block under inspection. (If there is no block to the left, then the predicted vector is taken as $(0,0)$).

In the first stage of the algorithm, M zones are constructed around the predicted point. Starting with the innermost zone, the MAD is found for each point in the zone. If the minimum of these MADs is lower than a predefined threshold (T_1), the selection is complete, and the point which provided the lowest MAD used to determine the motion vector. If the threshold is not met, then the next zone is considered. If after searching all M zones a suitable vector has yet to be found, the algorithm proceeds to a second stage.

Now the pattern is centred around the origin, and N zones are constructed. Two thresholds (T_2 and T_3) are used in this case. After each zone has been searched the minimum MAD (m) is compared to T_2. If $m < T_2$ then the search is complete and the position which resulted in m is used as the motion vector. If $T_2 < m < T_3$ then the next zone will be the final zone considered in the search, after which the position resulting in the lowest MAD will yield the vector. Otherwise the search continues with the next zone. If, after all N zones have been considered, there are no positions which satisfy the threshold criteria, the position with the lowest MAD is used to determine the vector.

2.11 Prediction Accuracy and Optimal Solutions

One of the problems with the methods detailed above is that there is no guarantee that they will find the same optimal solution as the ESA. There may be situations (such as off-line encoding) where it would be beneficial to have the optimal solution, but it would also be nice not to incur the full cost of the ESA. The Successive Elimination Algorithm (SEA) is an algorithm which fulfils both these criteria. It manages to do so by rejecting some candidate blocks without having to do a full pixel by pixel comparison.

2.12 The Successive Elimination Algorithm

If the metric to be used is the Sum of Absolute Difference (SAD) then the Successive Elimination Algorithm (SEA)[52] can be used to reduce the computational load incurred by the ESA. As with the ESA the algorithm iterates each block from from the input frame over a set of possible candidate matched from the reference frame. However, using the triangle inequality it is possible to dismiss some of the candidate blocks as better matches based only upon the sum of the pixel values in the candidate block, the sum of the pixel values of the input block, and the SAD of the current best matching candidate block.

The algorithm which forms the heart of the SEA stems from the mathematical inequality $||a| - |b|| \leqslant |a - b|$ which can be rewritten as

$$|a| - |b| \leqslant |a - b| \tag{2.2}$$

$$|b| - |a| \leqslant |a - b|. \tag{2.3}$$

Suppose $a = I(i, j)$ represents the intensity of the pixel at coordinates (i, j) in the input frame and $b = R(i - x, j - y)$ the intensity of a corresponding pixel in the reference frame, which has been offset by the motion vector (x, y).

Making these substitutions into equations 2.2 and 2.3 and summing over all the pixels in a block of size $M \times N$ gives

$$\sum_{i=1}^{M}\sum_{j=1}^{N}|I(i,j)| - \sum_{i=1}^{M}\sum_{j=1}^{N}|R(i-x,j-y)| \leqslant \sum_{i=1}^{M}\sum_{j=1}^{N}|I(i,j) - R(i-x,j-y)| \tag{2.4}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}|R(i-x,j-y)| - \sum_{i=1}^{M}\sum_{j=1}^{N}|I(i,j)| \leqslant \sum_{i=1}^{M}\sum_{j=1}^{N}|I(i,j) - R(i-x,j-y)|. \tag{2.5}$$

The first sum in equation 2.4 is the sum of the pixel intensities in the block from the input frame, which will be denoted by $\sum I$. Similarly the second sum represents the sum of pixel intensities of a candidate block from the reference frame, with the motion vector (x, y) denoted by $\sum R(x, y)$. The expression on the right hand side of the equation represents the SAD between the two blocks, and is denoted by $\mathrm{SAD}(x, y)$. With these definitions equations 2.4 and 2.5 become

$$\sum I - \sum R(x,y) \leqslant \mathrm{SAD}(x,y) \tag{2.6}$$

$$\sum R(x,y) - \sum I \leqslant \text{SAD}(x,y). \tag{2.7}$$

Suppose that for a motion vector (m,n) an initial candidate block with $\text{SAD}(m,n)$ has been found. A block with the motion vector (x,y) would prove to be a better match if the following criteria was satisfied

$$\text{SAD}(x,y) \leqslant \text{SAD}(m,n). \tag{2.8}$$

Substituting this into equations 2.6 and 2.7 and combining them yields

$$\sum I - \text{SAD}(m,n) \leqslant \sum R(x,y) \leqslant \sum I + \text{SAD}(m,n). \tag{2.9}$$

This is the key to the SEA algorithm. It states that, given a current matching candidate block with vector (m,n), the search for better matches is confined to those blocks which satisfy equation 2.9.

Precalculating all the block sums for the input frame is straightforward, as each pixel is part of only one block. However for the candidate blocks a naive approach could negate the advantages of using the SEA. However [52] also gives a method for fast calculation of the candidate blocks.

Suppose the dimensions of the image are $W \times H$ pixels, and that the block size is $A \times B$ pixels. The first step involves calculating *column sums* for each column in the image. Each column sum is initially made up of the first B pixels in each column.

Once these have been calculated the sum for the first block is simply the total of the first A column sums. The sum for the second block can now be found by taking the sum from the first block, subtracting from it the first column sum and adding column sum $A+1$. This process of subtracting and adding column sums in conjunction with the sum of the horizontally previous block can be continued to find sums for all the subsequent blocks in the first row, (see figure 2.14).

Before calculation of the second row of blocks can begin the column sums need to be updated. This new value is obtained by subtracting the first pixel from the column sum and adding pixel $B+1$. Once this is done the block sums can be found using the same method as for the first row. The process can then be repeated to find the block sums of all the blocks in subsequent rows.

The pseudo-code for this algorithm is shown below. It is very similar to that for the ESA except for the precalculation of the block sums and an additional com-

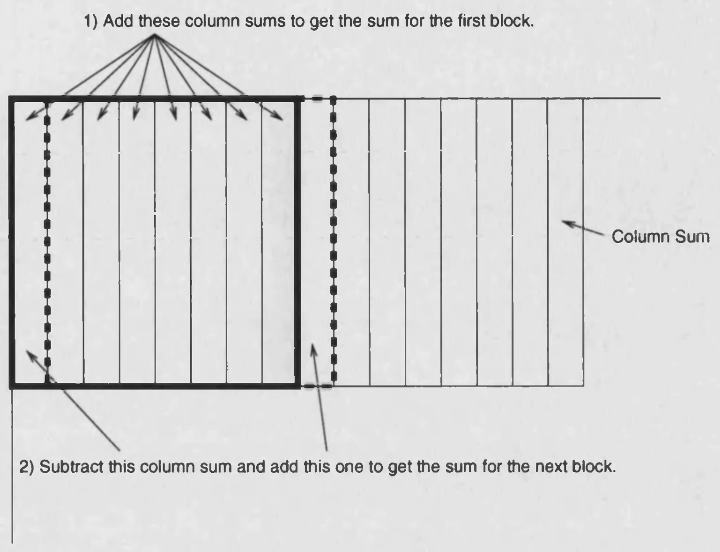

1) Add these column sums to get the sum for the first block.
Column Sum
2) Subtract this column sum and add this one to get the sum for the next block.

Figure 2.14: The Column Sums

parison within the inner loop. With these additions it would appear that the SEA could in fact have a worse performance than the ESA. This is possible but unlikely. The most computationally intensive part of the algorithm is the metric comparison. The SEA allows this to be avoided in cases where there is no chance of a better match being obtained. The more blocks which can be discarded this way, the faster the algorithm.

As the SEA comparison is always made against the current best guess block, an initial good guess is very important, as it can reduce the number of metric comparisons considerably. This initial guess is often taken to be the vector from the previously matched, neighbouring input block, or the vector obtained from matching the input block in the previous frame.

```
split input frame into blocks
calculate block sums for input frame
calculate block sums for reference frame
for each block in the input frame
  choose candidate blocks from reference frame

  best match = initial guess candidate block
  best vector = vector from input block to candidate block
  for each remaining candidate block
    if candidate block satisfies SEA equation
      compare blocks using SAD
      if input block is a better than current best match
        best match = candidate block
        best vector = vector from input block to candidate block
      end if
    end for
  end if
end for
```

2.13 Comparison of Techniques

One way of comparing the computational cost of the various block matching techniques is to look at the number of block matches required to find the motion vector for a single image block. If the search range is restricted to ± 7 there are $(2 \times 7 + 1)^2 = 225$ possible positions to search in order to find the optimal vector. The ESA, by definition, has to search all of these. The sub-optimal motion estimators only search a subset of these positions. The TSS evaluates 25 positions, whereas the Four Step Search (FSS) uses between 17–27 positions depending on the steps used.

The DSA does not have a fixed range but typically uses between 13–30 to cover the ± 7 region. In a similar fashion the number of positions searched by the Circular Zonal Search (CZS) varies depending upon the accuracy of the initial predicted vector. A good prediction could yield a vector in as few as 10 matches whereas a poor prediction could require all 255 positions to be evaluated. (This depends on the thresholds chosen for the search.)

The SEA also has a variable number of matches depending on the nature of the sequence being used. In[52] the authors claim the SEA can reduce the number of matches by 85%. During this work the values seemed to range between 40% and 60%. Although this is much higher than the other methods discussed in this chapter is it important to remember that the SEA is able to produce the *optimal* solution without necessarily checking all match positions.

2.14 Compression of Vector Fields

One of the most important aspect of vector field compression is that it must be done in a lossless fashion. It is essential that the encoder and the decoder both use exactly the same vector field to construct the predicted image. If this were not the case then there would be no guarantee that the transmitted residual would accurately represent the difference between the prediction and the actual image.

A common way to transmit motion vector data is by means of a predictive scheme. The motion vector to be transmitted is predicted from previously transmitted neighbouring vector (spatially or temporally), and the difference between this prediction and actual vector is losslessly encoded. A good example of such a scheme is given by H.263 and is described in appendix A.1. A consequence of such a scheme is that smoother field tend to compress more efficiently as the prediction error is smaller.

It is possible to provide a coarse degree of control over the cost by changing the size of the blocks used to describe the field. The larger the block size the fewer blocks each image will be split into, and therefore less vectors in the field. A vector field with fewer vectors should cost less to encode, at the expense of producing a less accurate predicted image.

Using the pair of images shown in figure 2.15 (which were QCIF sized) the ESA was used to produce vector fields with the following block sizes, 2×2, 4×4, 8×8 and 16×16. In each case the scan range was 16×16. The motion vector fields for each of these block sizes was compressed with the H.263 coder, (see section A.1) and the MAD of the predicted image was found. Finally the residual image produced in each case was compressed with a standard JPEG encoder.

The results are shown in table 2.1. Additionally, the motion fields produced are shown in appendix B.

It is worth noting that the number of pixels matches is *not* related to the size of the blocks used, but *is* directly related to the number of search positions. Smaller blocks require less pixel matches to be evaluated, but there are proportionally more blocks to find matches for.

Firstly it is important to note the correlation between the MAD error in the residual and the cost of compressing it. This demonstrates that in this case the greater the MAD the greater the cost of transmitting the residual *at a given quality*.

As expected, decreasing the block size allows a more accurate predicted image to be produced. This is accompanied by a rapid increase in the cost of encoding

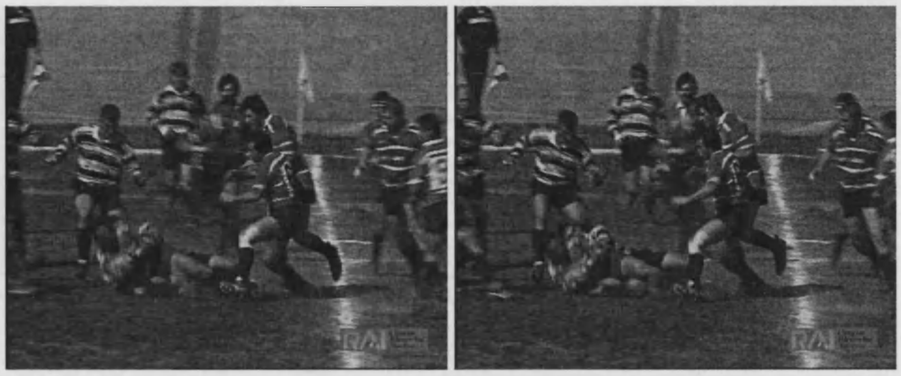


Figure 2.15: A pair of frames from the rugby sequence.

Block Size	Number of Blocks	MVF Cost (bits)	Cost per Block (bits)	MAD Error in Residual	JPEG Cost of Residual (bits)
2 × 2	6336	76308	12.04	2.190	5792
4 × 4	1584	10304	6.51	4.841	13952
8 × 8	396	2162	5.46	7.410	19096
16 × 16	99	508	5.13	10.304	22728

Table 2.1: The effect which varying the block size has on the bit cost of the motion description and the error in the residual.

the vector field. However, it would be unrealistic to claim that this method gave any fine control over the number of bits used by the vector field. Suppose we were encoding a sequence at 25fps for transmission at a bit rate of 64kb/s. This would allow just over $2,600$ bits per frame. In the example above, only the 8×8 and 16×16 block sizes could be used, as any smaller block sizes causes the bit allocation for the vector field to dramatically exceed the bit allowance.

There are also rate-constrained approaches[53] which often rely on Lagrangian techniques[54] to produce a locally optimised motion field. These do not provide direct control over the cost of the motion vector field, and are often closely related to other parts of the encoding system.

2.15 Conclusion

In this chapter we have looked at various techniques used to produce motion vector fields suitable for digital video encoding. Many of these stem from the ESA which can be considered the grandfather of block matching algorithms. Each set of techniques was designed to overcome limitations or failings in earlier methods. However, none of the methods examined so far are able to control the number of bits given to a motion vector field in any reliable way. It would be useful to do so as this would allow both terms on the right hand side of equation 1.21 to be easily varied. If this were the case then it would be possible to find the trade off in the bit distribution between the motion description and the residual which resulted in the optimum image quality.

In the next chapter we will examine techniques which move towards the aim of having more control over the cost of the motion description, and the effect this has on coding efficiency.

Chapter 3

Bandwidth Controllable Motion Estimation

Previously we have seen that existing methods for producing motion vector fields fail to give a great deal of control over the cost of encoding the vector field. As a result they are unable to find the optimal trade-off between the cost of the MVF and the residual. In this chapter we will examine two novel methods for which aim to address this problem.

The first method, the Extended Block Algorithm (EBA) attempts to do so by controlling the *smoothness* of the field, the smoothness implying correlation which in turn leads to increased compression. The second method, Embedded Quad-tree Motion Estimation (EQME) aims to produce an *embedded* representation of the motion description. This embedded nature means that the more data which is transmitted, the more accurate the motion description. However, it is possible to truncate the stream at certain points, and therefore control the cost of the motion description.

3.1 The Extended Block Algorithm

The Extended Block Algorithm (EBA) aims to produce a smooth motion field which also is more likely to reflect the true motion of the scene. Finding the vectors which describe the true motion can have several advantages. In textured areas such as grass or sky, some techniques produce motion vectors which can have a very chaotic, random appearance. Figure 3.2 shows a motion field produced using traditional block matching techniques (the frames used to generate this field, shown in figure 3.1, contain a large amount of grass). Such a field has high entropy and is difficult to compress.

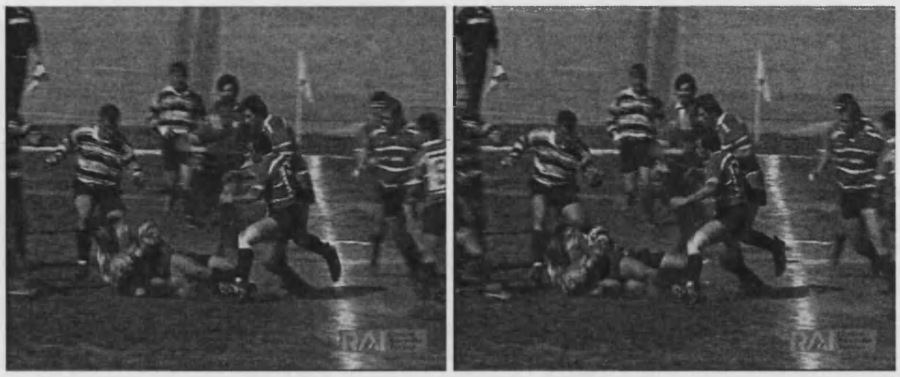


Figure 3.1: The frames used to produce the vectors in figure 3.2.

Finding the true motion of the scene would yield a more correlated field, which would aid in compression. Another advantage of a true motion field is that it becomes meaningful to interpolate the vectors over time. This enables the contents of a block to be estimated at an earlier or later time without needing to send further motion information.

In traditional block matching algorithms [55] each block is treated independently from its neighbours. However, in the Extended Block Algorithm (EBA) each block is coupled with those around it to a certain degree, enabling a more correlated field to be produced.

The pseudo-code for the algorithm is identical to that for the ESA. The difference comes in the way the blocks are compared using the metric. In the ESA each pixel in the input block was partnered with a corresponding pixel from the candidate block. The same is true in the EBA except that each block is *extended* by a certain number of pixels before the metric is calculated. For example, the block shown in figure 3.3 has been extended by three pixels in each direction. All the pixels in this extended block are used to calculate the motion vector. However, when creating the motion compensated frame the vector is only applied to the original pixels in the block, and not those from the extension.

When near the edge of the image, adding the extension may cause the block to extend beyond the image boundary. In this case the extension is cropped to fit onto the image. In this particular implementation the opposite edge is also cropped to the same amount so that the extension is symmetrical, (see figure 3.4). Another

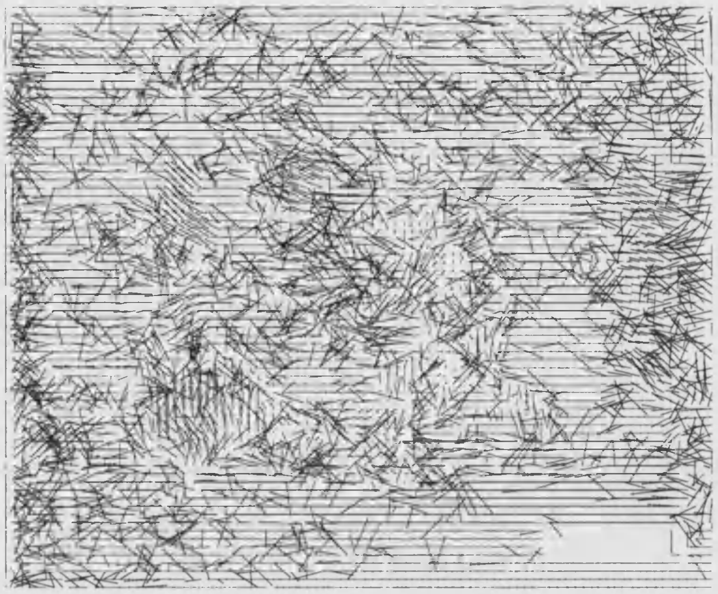


Figure 3.2: Even though the grass is all moving in the same direction the motion vectors are very chaotic.

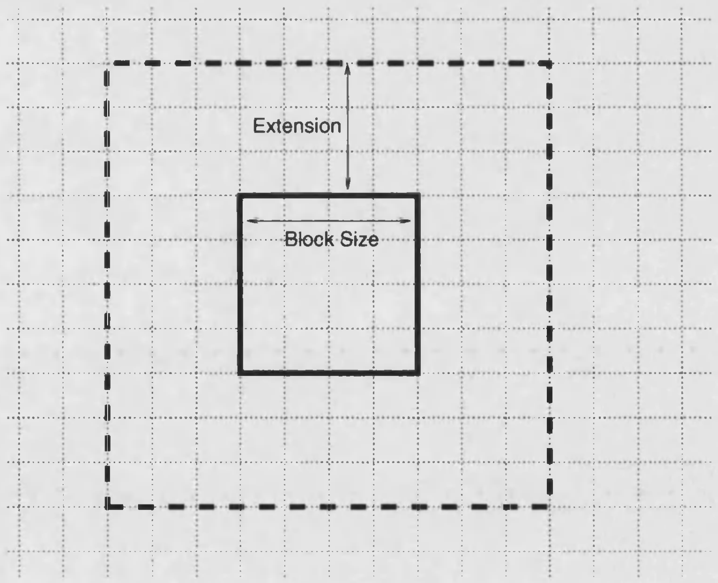


Figure 3.3: The block is extended by a given number of pixels before the metric is calculated. In this case the 4 $\times$ 4 block is extended by 3 pixels.

solution to this problem would be to extend the image using some form of padding method, such as symmetrical padding or null padding.

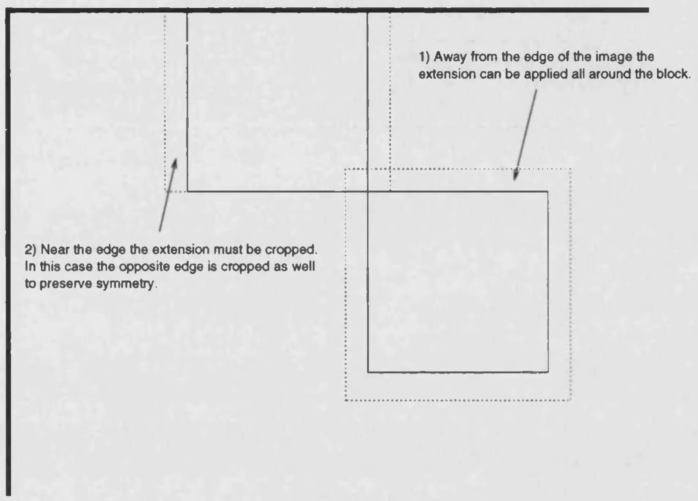

1) Away from the edge of the image the extension can be applied all around the block.
2) Near the edge the extension must be cropped. In this case the opposite edge is cropped as well to preserve symmetry.

Figure 3.4: What happens to the extensions at the edges?

If the metric used is normalised, then blocks of different sizes may be compared. However, often all the candidate blocks are required to have the same dimensions. In this case, (when near the edge of the frame), the extension applied to *all* the candidate blocks is that of the most cropped block. So for example if one of the candidate blocks is a corner block, no extension will be applied to the blocks.

The method can also be modified to take advantage of an SEA type implementation when the metric to be used is the SAD. However, there are two factors which need to be taken into account if this is to be done. Firstly the blocks are no longer all the same size, so the metric must be normalised. In the case of the SAD this means dividing by the number of pixels in the block to yield the MAD, and the block sums become the sum norms. Secondly, the calculation of these sum norms also requires attention, as the blocks in the input frame are no longer independent as the extensions cause them to overlap.

3.1.1 Computational Cost of the EBA

In terms of the number of block matches required to produce a vector field from a pair of images, the EBA has the same computational complexity at the ESA. However, due to the extension in the size of the blocks the number of pixel comparisons is greater when using the EBA. Table 3.1 shows this increase in terms of compared pixels for various block sizes and extensions. In the case where the extension is half the block size the computational cost is over twice that of the standard ESA.

Block Size	Extension	ESA Pixel Comparisons	EBA Pixel Comparisons	Relative Cost
8	2	64	100	156%
8	4	64	144	225%
8	8	64	256	400%
16	2	256	324	127%
16	4	256	400	156%
16	8	256	576	225%
N	e	N^2	$(N+e)^2$	$\left(1+\frac{e}{N}\right)^2$

Table 3.1: The increased computational cost of using the EBA.

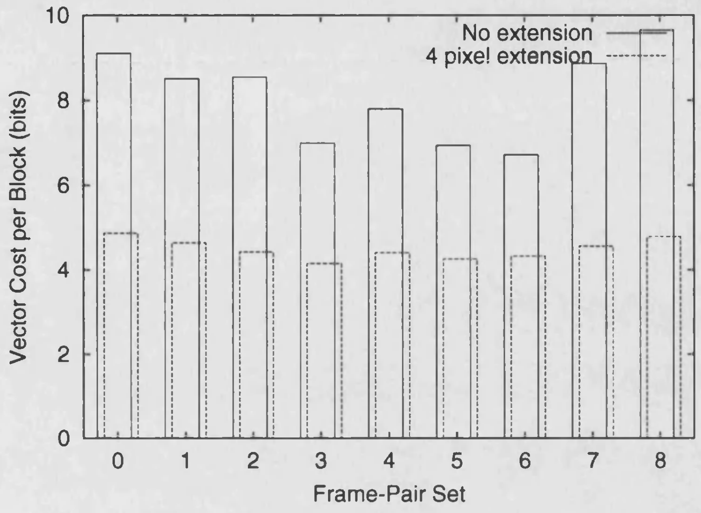
3.1.2 Results

In order to demonstrate the smoothing effect of the EBA two motion vector fields were produced from input images. In both cases the block size was 4×4 pixels and the scan range was 16×16. However, the first vector field (figure 3.7) has no extension (and is therefore identical to the result produced by the ESA), whereas the second field (figure 3.8) has an extension of 4 pixels applied in both the horizontal and vertical directions.

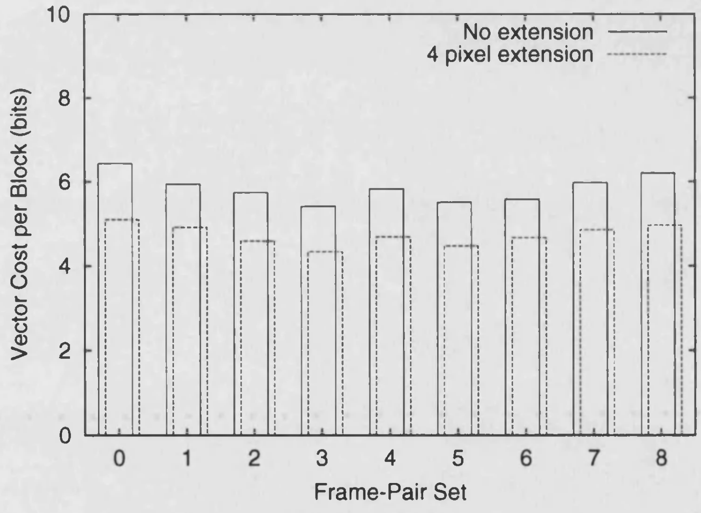
It is clear to see that the vector field produced by the extended blocks is much smoother than that without. However, the MAD between the input frame and the motion compensated frame has risen from 4.28 grey-levels/pixel with no extension to 6.51 grey-levels/pixel with the extension.

In order to obtain an estimate for the bandwidth required by the each field the following method was used. In raster scan order, each vector, (starting with the second), was subtracted from the previous one. For each vector component the entropy of each differenced value was found. Summing over all possible values yields an estimate of the number of bits to encode each vector. The results are shown in table 3.2, along with the actual bit cost for encoding the field as produced by an H.263 encoder. It can be seen that in this case around 4 bits/vector can be saved by using the EBA.

The procedure was repeated for another eight pairs of frames from the rugby sequence. Each pair was evaluated with block sizes of 4 and 8, both with and without a 4 pixel extension. The results are shown in figure 3.5.



Block Size 4



Block Size 8

Figure 3.5: The effect of the EBA on vector bit cost.

	Calculated Entropy	
	Extended Blocks	Traditional Blocks
x Vector Component	2.59	4.60
y Vector Component	2.19	4.31
Total	4.78	8.91
Actual Bit Cost	4.87	9.10

Table 3.2: Comparing the number of bits per vector component required when a vector field is created with traditional block matching methods and the EBA. The actual bit cost of compressing the field is shown along side the cost based on entropy calculations.

From these results we can see that the EBA consistently reduces the cost per block for all the pairs of frames used. The reduction is more evident when the block size is smaller. As previously seen the smaller block size (with no extension) produces a less correlated vector field so there is greater scope for improvement.

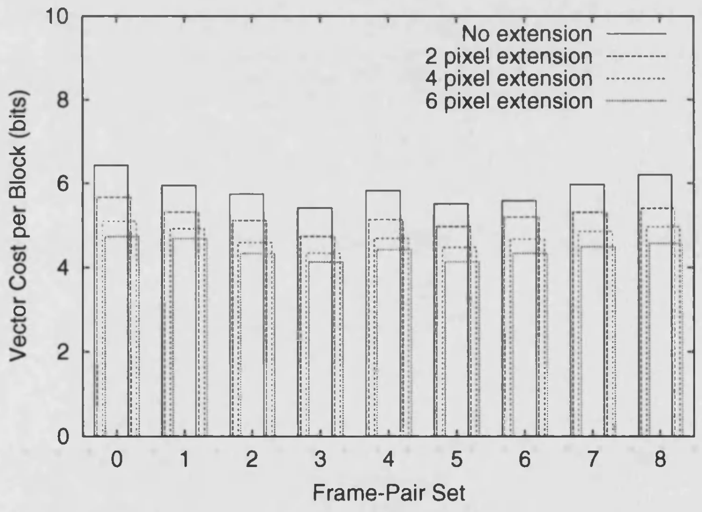


Figure 3.6: The effect of changing the amount of extension when using the EBA. (Block Size 8)

Figure 3.6 shows the effect of varying the amount of extension when using the EBA, (in this case the block size was 8). It can be seen that the greater the extension the lower the average cost in bits per block. This is due to the smoothing nature of the EBA creating greater correlation between the vectors as the extension increases.

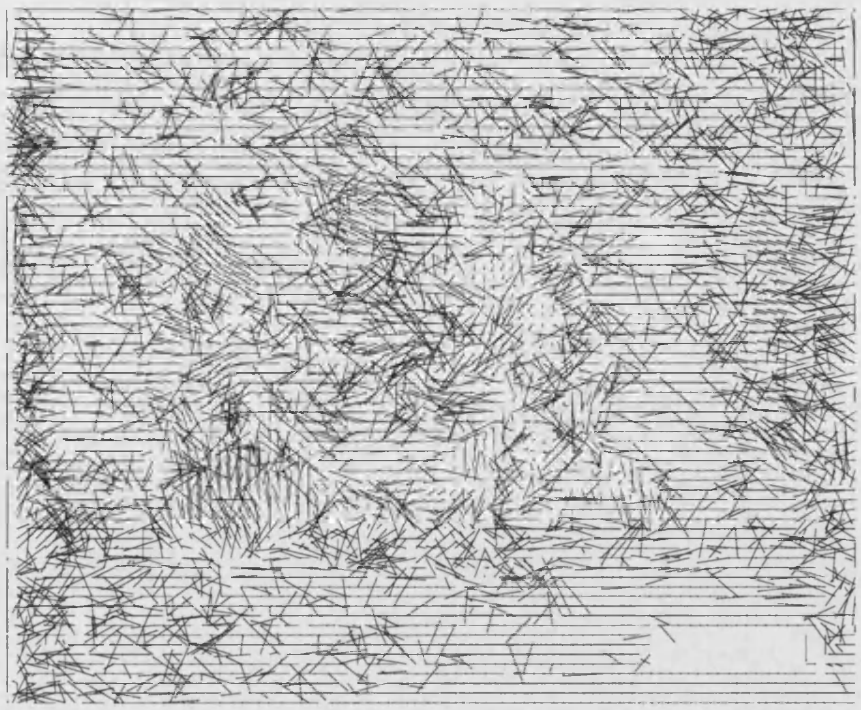


Figure 3.7: The motion vector field produced using 4×4 pixel blocks with a scan range of ± 16 pixels, (no extension).

3.2 Embedded Quad-tree Motion Estimation

The Embedded Quad-tree Motion Estimation (EQME) [56] uses a coarse to fine approach to produce an embedded motion description. The global motion of the input frame with respect to the reference frame is found, and is then recursively refined, producing locally more optimal vectors.

In common with block based motion estimation techniques the input frame is divided into an array of blocks, each $M \times N$ pixels in size. The maximum density of the vector field is limited by the number of blocks.

Initially all the blocks in the input frame are considered as one group and are thus constrained to share a common motion vector. For each allowed motion vector a comparison metric is applied between the group of blocks and the corresponding

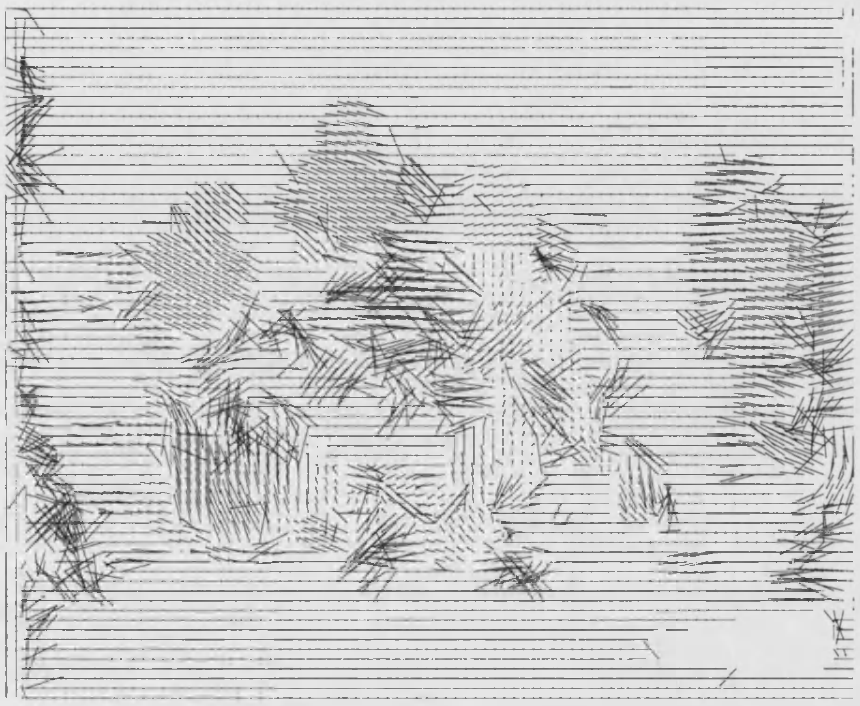


Figure 3.8: The motion vector field produced using 4×4 pixel blocks with a scan range of ± 16 pixels extended by 4×4 pixels.

group (as determined by the motion vector) in the reference frame. For all non-zero motion vectors some of the required pixels in the reference frame will lie outside the image. The value of these pixels are determined using an image padding scheme such as zero-padding or symmetric-padding. The optimal global motion vector is the one which results in the most favourable outcome of the comparison metric. The value of this vector becomes the first element in the motion description stream.

Once the global motion vector has been found the refining stage commences. In this process, the group of blocks from the input frame is split to form four new groups, each of which is a quadrant of the old group, see figure 3.9. For each of these new groups the optimal motion vector is found in the same way as for the global motion. Each vector is then subtracted from the vector of the parent quadrant before being entropy coded and added to the stream, see figure 3.10.

Each new quadrant is itself then split and the process recursively repeated until each 'group' contains only one block. The resulting embedded motion stream now contains the motion vectors for successively smaller blocks, each level containing blocks a quarter of the area of the previous one. This bit stream can be truncated at any point to give a motion description to an exact number of bits.

3.2.1 Results

Embedded Quad-tree Motion Estimation (EQME) streams were generated for pairs of reference/input frames selected from three commonly used video test sequences. In each case a fixed number of bits, b_{total}, was allocated to encode the input frame, given the reference frame. Of these b_{motion} bits were taken from the beginning of the EQME stream and used the produce a motion compensated frame. Subtracting this motion compensated frame from the reference frame produced the residual, which was coded using the remaining $b_{residual} = b_{total} - b_{motion}$ bits. The resulting decoded residual was then combined with the motion compensated frame to produce the transmitted frame.

For each pair of frames, the proportion of the bits allocated to the motion description b_{motion}/b_{total} was varied between 0 and 1 and the Normalised RMS Error between the transmitted and original frames found, see figures 3.11 to 3.13. For each figure $b_{total} = 10,000 \pm 1\%$ and the image size was 176×144.

When implementing the EQME a number of decisions regarding the various parameters need to be made. These results were obtained for a particular implementation of the EQME method, with parameters as listed below.

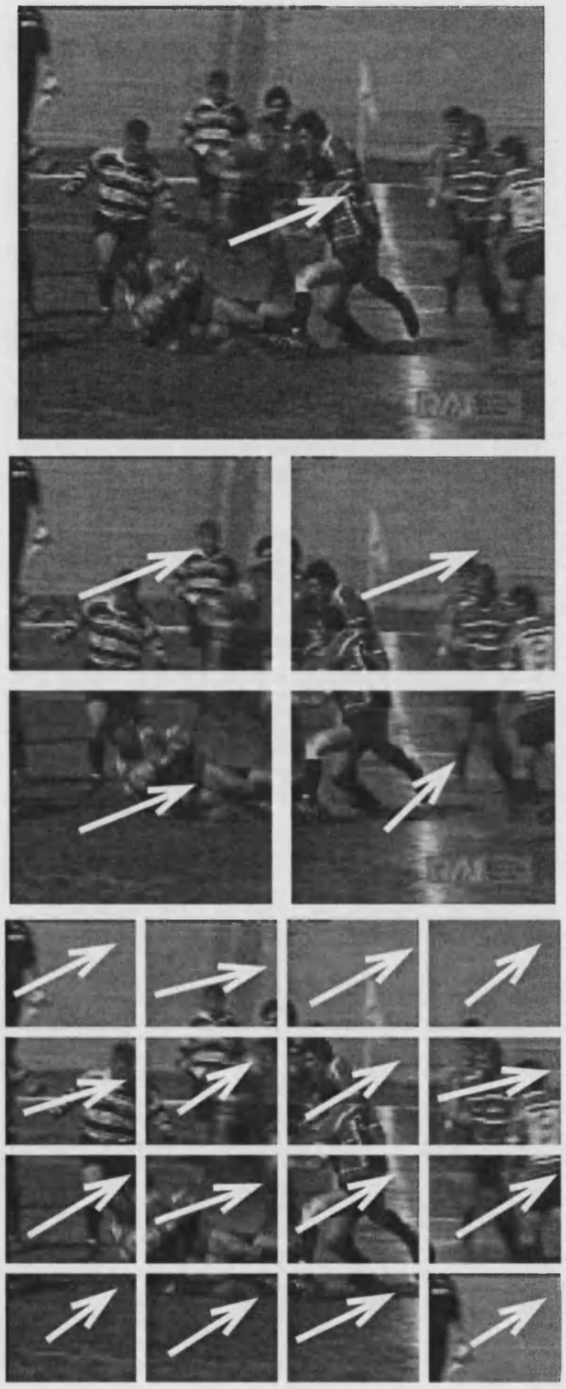


Figure 3.9: Each block is subdivided into four others, and the motion vectors are found for these.

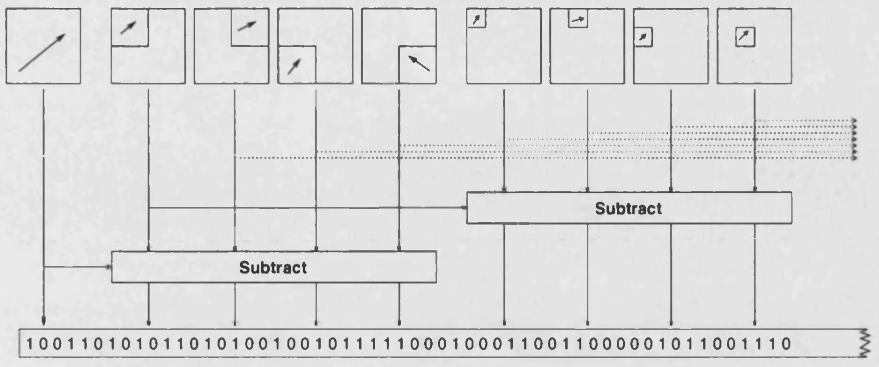


Figure 3.10: Each new block is differentially encoded with respect to its parent and added to the embedded stream.

- The block size was 4×4.
- The allowed range of the motion vectors was $\pm 15 \times \pm 15$.
- The global motion was encoded raw, and the refining difference vectors were entropy coded using an adaptive Huffman scheme.
- If a group of blocks could not be spit into four equally sizes quadrants, the top-left quadrant was made the largest.
- The comparison metric used was the Sum Absolute Difference (SAD).
- The reference frame was zero-padded in order to account for the unconstrained nature of the vectors.

The results illustrate that the optimal distribution of bits between the motion description and the residual can vary considerably. For example, figure 3.11 was produced from the *Carphone* sequence in which the motion is relatively simple. There is a small amount of global motion, with the remaining motion being quite coarse. Here the motion description initially plays an important role in aligning the two frames. However, due to the nature of the motion, more detailed refinement is more efficiently handled by the residual coder. In this case once the optimal proportion has been reached (0.2), the error rapidly rises as further bits are taken from the residual and allocated to the motion description.

Figure 3.12 is taken from the beginning of the *Foreman* sequence, which was filmed using a hand held camera. As a result there is significant camera wobble

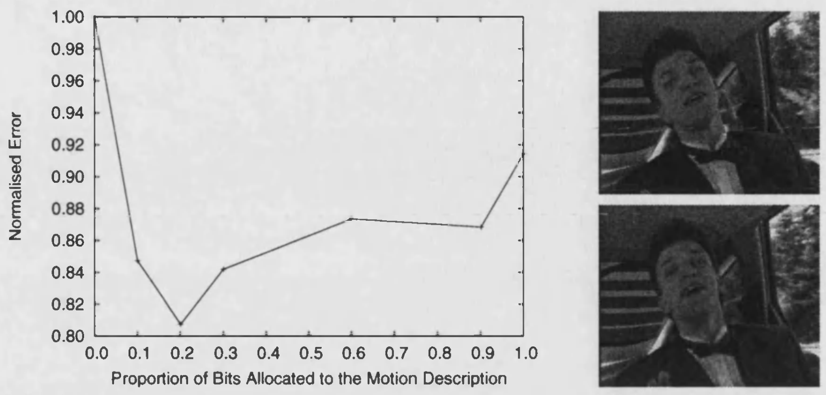


Figure 3.11: Carphone frames 305 and 309.

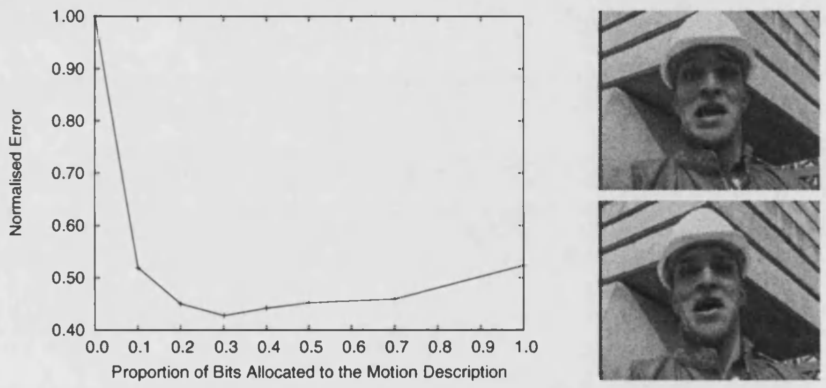


Figure 3.12: Foreman frames 66 and 70.

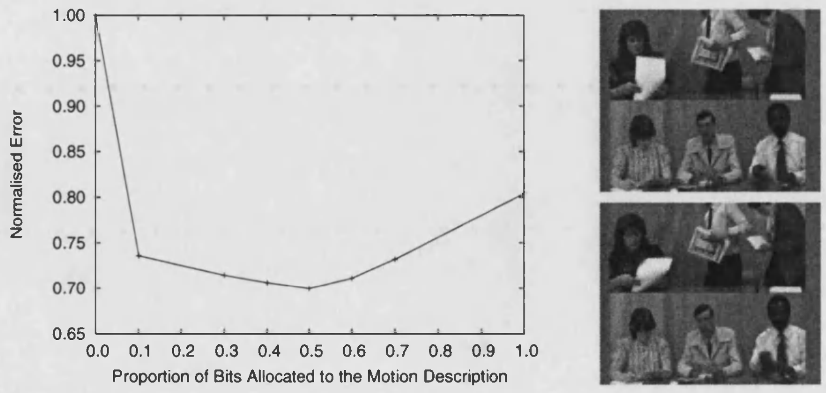


Figure 3.13: Trevor frames 48 and 52.

between the frames, in addition to the movement of the foreman himself. As this motion is slightly more complicated, the optimal distribution requires more bits for its motion description than in figure 3.11. However, once the optimal point is reached further allocation of bits to the motion does not increase the error as rapidly as for Carphone. The results in figure 3.13 come from the *Trevor* sequence, which contains six people moving independently. This leads to localised motion which requires an even greater proportion of the bits to be allocated to the motion description in order to generate the optimal transmitted frame.

Although the benefits of using the optimal distribution (in terms of RMS error) may be small for a single pair of frames, the cumulative benefit over a sequence of differentially coded frames could become significant.

3.3 Discussion and Conclusions

The EBA has been shown to generate smoother vector fields with lower entropy (and therefore higher compression) than those generated by the ESA. However, it is extremely computationally expensive, even more so that the ESA. Controlling the smoothness by varying the amount of extension leads to a certain degree of control over the bandwidth required to encode the field, however, this control is still coarse. Even so, the generation of such a dense, smooth motion vector field may have applications outside that of video coding such as cloud tracking [57], glacier surface motion, and medical imaging [58].

It is also likely that the EBA will produce more visually pleasing results which are less prone to blocking artifacts. As large contiguous areas of the image share the same vector, the resulting residual is less likely to have high frequency 'edges' and will therefore be easier to encode.

This effect is similar to that achieved by overlapping block motion estimation in which the applied vector for any given block is the weighted average of the calculated vector for that block and its immediate neighbours. This also has the effect of smoothing out the vector field, but lacks the EBAs ability to vary the bandwidth associated with the motion description.

The results for the EQME are important in that they confirm the hypothesis that this *is* an optimal trade-off between the bits assigned to the motion description and those assigned to the residual. They also show that this optimal proportion is not fixed, and varies depending on the contents of the sequence.

One problem with the stream produced by the EQME is that, although bandwidth-controllable, it is not particularly efficient at compressing the field. For example, each quadrant is always split at the refining stage, which is inefficient for large areas with constant motion. Quad-tree decomposition has already been applied in other areas of image processing [59, 60] and evolution of this method could benefit from the incorporation of similar decomposition schemes.

3.4 Summary

In this chapter we have examined two methods which give a certain amount of control over the bandwidth used to encode the motion description. However, each of the methods has drawbacks which prevent them from competing with existing techniques. The EBA produced smooth, easily compressed fields, but lacks fine grained control over the allocated bandwidth, whereas the the EQME has good bandwidth control, but is not efficient at compressing the description.

It was noticed that the smoother fields produced by the EBA often contained fewer unique vectors and that they were also more easily compressible. This lead to the hypothesis which suggests that the fewer unique vectors a field contains, the cheaper it is to encode.

The next chapter explores this hypothesis and examines techniques which attempt to overcome the shortfalls of both the EBA and the EQME thus achieving good control over the bandwidth, while also compressing the resulting field efficiently.

Chapter 4

Restricted Vector Set Motion Descriptions

One way to measure the complexity of a vector field is to count the number of unique vectors which occur within it. For example, a uniform field has only one unique vector, a field in which the left half moves one way and the right half another has two unique vectors and a 11×9 field in which all the vectors are different would have 99 unique vectors.

In order to investigate the relationship between the number of unique vectors in a field and the cost of encoding that field, the following experiment was conducted. Each of the following sequences, ('Carphone', 'Claire, 'Container', 'Foreman', 'News', 'Salesman', 'Silent', 'Suzie' and 'Trevor') were encoded using a standard H.263 encoder at 128 kb/s using an IPPP... encoding scheme. For each of the 2885 P-frames produced the cost of the vector field was found using H.263 encoding along with the number of unique vectors in the field. The average costs were found for each number of unique vectors and the results are shown in figure 4.1.

Up to about 40 vectors the graph is reasonably smooth, with each number of vectors having sufficient data to produce a reasonable average. Beyond this point however there are fewer data for each point causing the graph to appear more chaotic. The raw data for these results can be found in Appendix D.1. Despite this the graph still displays a visibly linear trend.

It can be seen that the larger the number of unique vectors the more, on average, it costs to encode the field. This can be attributed to the decrease in correlation which tends to occur with an increase in vectors.

From this we can conclude that any method which is able to reduce the number

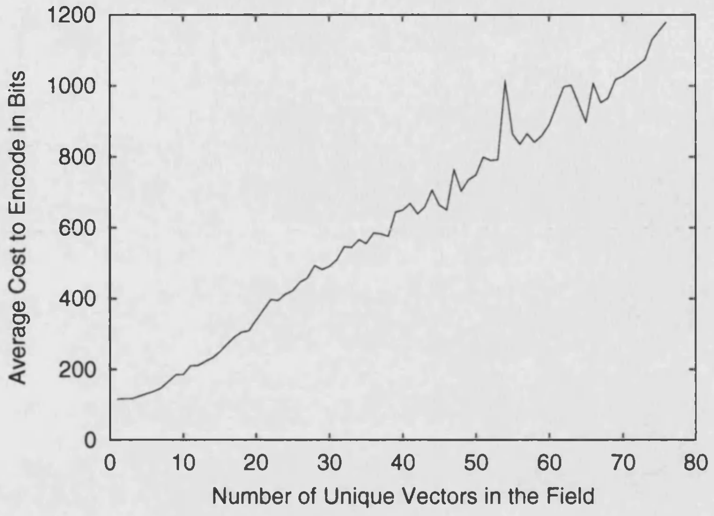


Figure 4.1: The more unique vectors a field contains, the larger the size of the encoded field.

of unique vectors in a field should also have the effect of reducing the encoded cost of the field. In this section we explore techniques which try to restrict the number of vectors present in a field. These methods try to select a suitable set of candidate vectors for a field, and are hence called *vector selection strategies*.

Three such vector selection strategies are presented below. In essence they are no more than motion estimators, however they can be constrained so that the motion vector fields they produce contain no more than N unique vectors.

4.1 Histogram Method

As its name implies the Histogram Method[61] picks vectors based on how frequently they occur. It does not operate directly on the current and reference images, rather its input is a motion vector field which has already been generated from them.

There are three stages which make up the Histogram Method;

1. Prepare the histogram,
2. Pick the vectors,
3. Assign the stragglers.

Vector	Frequency	Vector	Frequency
(0, 0)	18	(-2, -2)	2
(2, -1)	12	(7, 6)	1
(1, 0)	8	(2, 6)	1
(1, -1)	4	(6, 5)	1
(-1, -1)	4	(-8, 3)	1
(0, -2)	3	(3, 0)	1
(3, 2)	2	(0, -1)	1
(2, 1)	2	(-2, -1)	1
(1, 1)	2	(4, -3)	1
(2, 0)	2	(1, -3)	1
(-1, 0)	2	(-8, -3)	1
(4, -2)	2	(-5, -4)	1
(2, -2)	2	(1, -5)	1
(1, -2)	2	(8, -8)	1

Table 4.1: Example Vector Frequency for a typical motion field.

4.1.0.1 Prepare the Histogram

The vector field which is used as an input to the Histogram Method will contain many different vectors and (possibly) several instances of each vector. The first thing which needs to be done is to create a list of all the individual vectors which comprise the field along with a count of how frequently each vector occurs. A typical list is shown in table 4.1.

4.1.0.2 Picking the Vectors

The next task is to select the N vectors which will form the final motion vector field. We simply select the N most frequently occurring vectors. In some cases there will be more than one vector with a given frequency. Such ties can be resolved in a number of ways, for example by selecting the vector closest to the origin, or the one whose block present the least error.

4.1.0.3 Assign the Stragglers

However, this leaves us with a problem. What do we do with the blocks whose vectors were not selected to form part of the new motion vector field? We shall call these blocks the *stragglers*, as we see in table 4.1 nearly half the vectors occur only once. The solution is to assign them to one of the blocks which is already present in the list. For each straggler block the following procedure is used. Each vector from the list is applied to the block and the prediction error is calculated. The block is assigned to the list vector for which this prediction error is smallest. This part of the process requires access to the original images from which the input motion vector field was generated, (or a table of the error metrics for each block/vector combination).

This will result in a complete motion vector field which is similar to the original, but with only N unique vectors.

4.2 Metric Method

The Metric Method[62, 61] attempts to find the the optimal N vectors in a sequential manner. It will first find the single vector which minimises the error when applied to all the image blocks. The it will find a second vector which when used along side the first vector will minimise the error, and so on. The vectors are selected in a sequential fashion, so that the best N vectors contains the best $N-1$ vectors, plus an extra one.

Unlike the Histogram Method the Metric Method does not take an existing vector field as an input. However, it does require a list of possible candidate vectors. These are the vectors which will be considered for the vector list. Possible ways to pick candidate vectors include;

- Specifying a range for the vectors. For example $(\pm 16, \pm 16)$.
- Using the vectors from a field generated by a block matching algorithm

Once the candidate vectors have been selected the prediction error when each of these vectors is applied to each image block is calculated and stored in a table. Such a table is shown in figure 4.2.

First we want to find the single vector which gives the minimum total prediction error when applied to all the blocks in the image. This vector can be easily found by calculating the sum of each row of the table and finding the minimum. The vector which provided this sum is the single most optimum vector for the image.

The sum of each row in the table is found; this is the total prediction error when each single vector is applied to *all* the blocks in the image. The vector for which this sum is a minimum is the most optimal single vector for the image.

Having found this vector we can now look for the next best vector from the remaining candidates in the table. Each of these remaining vectors is considered

		Block					
		1	2	3	⋯	$B-1$	B
Vector	1	383	886	777	⋯	915	793
	2	335	386	492	⋯	649	421
	3	362	27	690	⋯	59	763
	⋮	⋮	⋮	⋮	⋱	⋮	⋮
	$M-2$	211	368	567	⋯	429	782
	$M-1$	530	862	123	⋯	67	135
	M	929	802	22	⋯	58	69

Figure 4.2: Example of a metric table. Each cell of the table contains a measure of the error introduced into the predicted image when a given vector is applied to a given block.

in turn. As before a sum is made along each row of the table, but now contribution from each image block is the minimum of the prediction error for the vector under consideration and that of the previously selected optimum vector. Once again, the minimum sum indicates which candidate vector should be selected as the second best vector. This process continues until N vectors have been selected.

There is a subtle consequence of this selection process which means that N iterations will not necessarily produce a field with N unique vectors. This is due to the fact that later vectors can cause earlier vectors to become obsolete. This is most easily explained by way of an example.

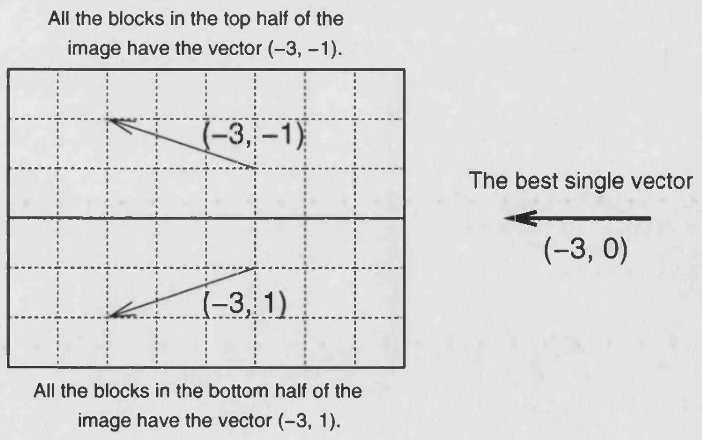

All the blocks in the top half of the image have the vector (–3, –1).
(–3, –1)
(–3, 1)
All the blocks in the bottom half of the image have the vector (–3, 1).
The best single vector
(–3, 0)

Figure 4.3: This hypothetical field is composed of only two different vectors. However, the vector which best describes the global motion is neither of them.

Suppose we have a pair of images for which the most optimal vector field contains just two unique vectors, in similar proportions, (figure 4.3). It is quite possible that the best global vector for these images is neither of the two most optimal vectors. If we assume all three of these vectors are present in the candidate list, the first iteration of the Metric Method will result in the vector $(-3,0)$ being selected. The next two iterations will add the other two vectors, say $(-3,1)$ followed by $(-3,-1)$, resulting in the optimal motion vector field. However, this field will no longer contain the original vector $(-3,0)$, so after three iterations of the Metric Method we have a field which contains only two vectors.

Another interesting property of this example is that after two iterations we have a field which contains the two vectors $(-3,0)$ and $(-3,1)$. Suppose the selection process stopped at this stage, it is clear by inspection that a much better choice would have been the pair of vectors $(-3,1)$ and $(-3,-1)$. However, because the Metric Method can only pick vectors sequentially it is stuck with the erroneous vector $(-3,0)$. (In our example this vector becomes obsolete in the next iteration but there is no guarantee that this will happen so quickly, if at all.)

4.3 Full Search Method

One drawback of the Metric Method is that the best pair of vectors does not necessarily contain the best single vector as was illustrated in figure 4.3 and the corresponding example.

As with the Metric Method the Full Search Method[61] takes a list of candidate vectors as its input. However, where as the Metric Method asks the question, "What is the N^{th} best vector given that we have already selected these $N-1$ vectors?", the Full Search Method simply asks, "What are the N best vectors?".

Although this question is more straightforward than that posed by the Metric Method the removal of the restriction leads to a much higher computational cost. Now there is no need for the best pair of vectors to contain the best single vector. As a result there are many more such pairs to evaluate in order to find the optimum.

A table is constructed as it was in the Metric Method. To find the most optimum N vectors, each combination of N from the candidate list must be tested. The total row sum resulting from the minimum prediction error of each vector under consideration. When all such combinations have been considered the one with the most favourable prediction error is the optimum.

4.4 Comparing the Vector Selection Strategies

The three methods presented above all have different merits and disadvantages. The Histogram Method is very easy to implement and computationally cheap. However, the frequency tables produced by many typical vector fields often yield vectors with the same frequency (as seen in figure 4.1) and there is no way to prioritise these vectors without further calculation (which could possibly be done using the Metric Method). As a consequence, its total SAD is always higher than for the other techniques, apart from a few cases for which the histogram has distinct vectors.

Although the Metric Method generally performs better than the Histogram Method, the best N vectors do not necessarily contain the best $N-1$ vectors. This can be seen in figure 4.4 which shows the best $N = 1, 2, 3, 4, 5$ vectors as found by the Full Search Method and the Metric Method. Both methods pick the best single vector $(1, 0)$ but this vector is *not* found in the best pair of vectors (as determined by the Full Search Method). As the Metric Method can only find the second vector *given* that it has selected vector $(1, 0)$, it fails to produce an optimal result. It is not until $N = 4$ that this vector reappears in the list for the Full Search Method.

Metric Method		**Restricted Full Search Method**	
N	Vectors	N	Vectors
1	(1, 0)	1	(1, 0)
2	(1, 0) (2, -1)	2	(2, -1) (0, 0)
3	(1, 0) (2, -1) (0, 0)	3	(2, -1) (0, 0) (1, -2)
4	(1, 0) (2, -1) (0, 0) (1, -2)	4	(1, 0) (2, -1) (0, 0) (1, -2)
5	(1, 0) (2, -1) (0, 0) (1, -2) (2, 1)	5	(1, 0) (2, -1) (0, 0) (1, -2) (2, 1)

Figure 4.4: The best $N = 1, 2, 3, 4, 5$ vectors for the Metric Method and the Restricted Full Search Method.

Another feature of the Metric Method is that earlier vectors can be made obsolete by later vectors. For example, figure 4.5(a) shows a hypothetical mapping of vectors that minimises the SAD for $N = 4$. The index of the vector indicates the order in which it was added to the mapping. Notice that by this iteration the vector with index 1 only occurs once in the mapping. Suppose that, given that these vectors have been selected, most improvement can be gained by replacing this vector with a new vector with index 4 (see figure 4.5(b)) bringing the total number of vectors in the list to five. However, as vector 1 no longer appears in the mapping it has become obsolete and after five iterations the vector list only contains four useful vectors, and is still considered $N = 4$. As the Metric Method cannot jump straight to this second, more optimal solution without considering the first one, it must always check *past* the number of vectors required in order to ensure that all the cases for the required value of N have been considered.

3	3	0	0	0
3	2	1	0	0
3	2	2	2	0
3	2	2	2	0

(a)

3	3	0	0	0
3	2	4	0	0
3	2	2	2	0
3	2	2	2	0

(b)

Figure 4.5: An example vector mapping: (a) after the best four vectors have been found and (b) after the next iteration in which the replaced vector has become obsolete.

Figure 4.6 compares the total SAD of the predicted image constructed using N vectors selected by the Histogram, Metric and Restricted Full Search Methods for two frames from the Foreman sequence. The frames were split into 80 blocks and the reference frame was degraded as it would have been if used in a rate-constrained video codec. The candidate vector set for the Restricted Full Search Method (RFSM) was found by running the Metric Method to completion, giving 28 vectors.

As expected the RFSM *always* gives the best result, but it is more often than not matched by the Metric Method. Also the higher computational cost of the RFSM makes it less practical to use than the Metrical Method. The Histogram Method does not perform as well for all non-trivial ($N > 2$) cases.

4.5 Restricted Vector Results

The Metric Method was applied to four sequences (Foreman, Carphone, News and Container) to produce around $20,000$ motion vector fields containing between 2 and 30 unique vectors. Each field was encoded using the motion field compressor from the H.263 codec. An average result was found for each value of N, the number of unique vectors in the field. These are displayed in figure 4.7.

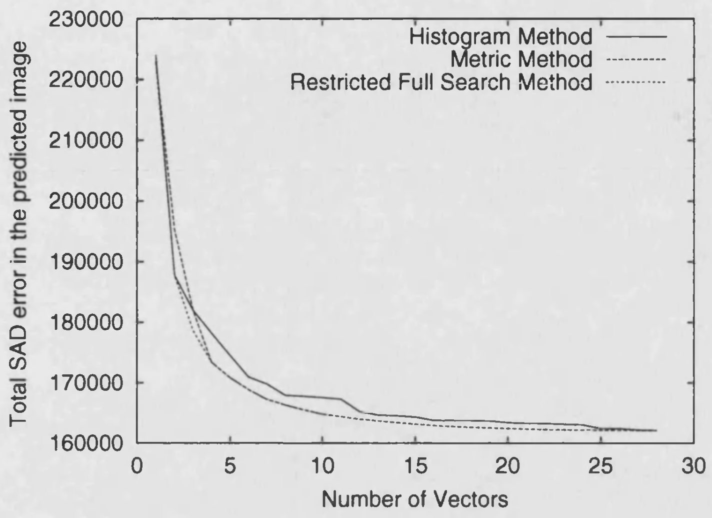


Figure 4.6: Comparing the different vector selection strategies.

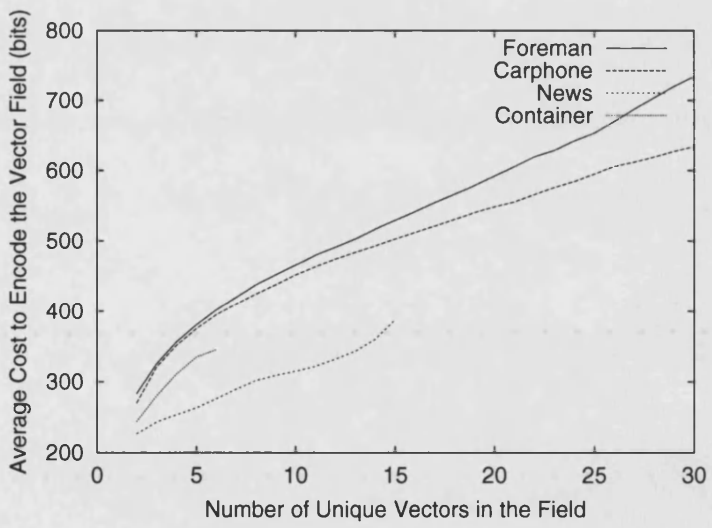


Figure 4.7: The result of limiting the number of vectors in a field using the metric method, and compressing with a standard H.263 encoder.

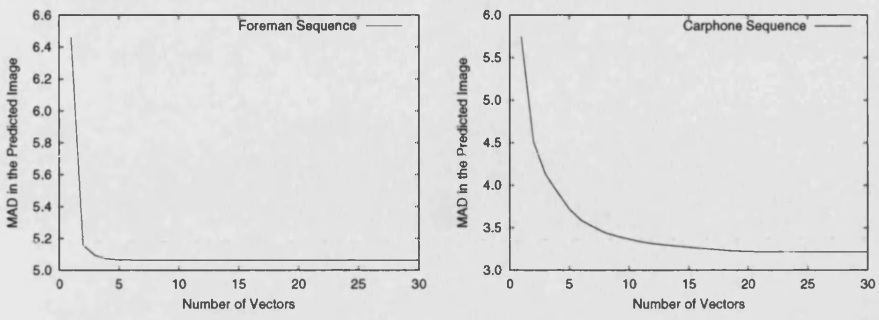


Figure 4.8: Examples of how the MAD varies with the number of vectors selected.

The results clearly demonstrate that there is a strong correlation between the number of unique vectors in the motion vector field (as produced by the Metric Method) and the number of bits needed to encode the field using the H.263 encoder. The results for the news and container sequences are limited due to the lack of complex motion in these sequences. For example, very few frames in the news sequence require more than 15 vectors to adequately describe the motion between the frames.

4.6 Dynamically Selecting the Number of Vectors

The methods described in this chapter allow us to construct a motion vector field which contains *N* unique vectors. Increasing *N* produces a more complicated motion vector field, but the resulting residual contains less error. In this section we will look at ways in which a sensible value of *N* can be determined for different pairs of frames.

A good value of *N* is related to the complexity of motion in the sequence. If the motion is simple, a low value of *N* will usually be sufficient, as adding more vectors to the field will not yield a significant reduction in the prediction error. However, for more complex motion additional vectors will be beneficial, so we would like to include them in the field.

Figure 4.8 shows how the prediction error changes with *N* for a pair of frames from two different sequences, (the foreman sequence and carphone sequence). The frames from the foreman sequence are taken towards the end of the sequence, where most of the movement in between the frames is due to the camera motion. As a result nearly all the motion is well represented with only a couple of vectors.

Any additional vectors added to the field have an extremely small effect. In the carphone sequence there is a greater variety of movement, so added more vectors does improve the quality of the predicted image, (perhaps up to 15 vectors).

This example serves to demonstrate the need to tailor the number of vectors picked by the selection strategies (N) to the content of the sequences being compressed.

Rather than try to determine an absolute value of N we will take an iterative approach to the problem. That is we will create fields with $N = 1, 2, 3, \ldots$ vectors, and after each new field is created we evaluate some criteria to determine whether or not we should continue.

4.6.1 Number of Blocks Changed

The motion vector field generated for N vectors will differ from that created with $N - 1$ vectors. One way to measure this difference is to count the number of blocks for which the vectors have changed. When this number becomes small it is an indication that each additional vector is having only a small impact on the prediction error.

4.6.2 Change in Prediction Error

As N increases the quality of the prediction will improve and prediction error should decrease. In the case of the Metric Method and the Full Search Method the amount of improvement decreases with increasing N, (the prediction error monotonically decreases). This *change* in the prediction error indicates directly the effect of adding vectors to the motion vector field. When this change becomes small the bit cost of encoding the more complex vector field may outweigh the benefits of the reduction in prediction error.

Figures 4.9 to 4.12 show the effect of using the prediction error as a criteria for dynamic picking. In each case the picking continued until the improvement in prediction error dropped below a certain threshold. The threshold of 0 allowed the picking to continue until an optimal field was found. It is clear from these results that the number of vectors in the field can be controlled by this limit. The result for the container sequence all coincide. This is due to the very limited motion in the sequence.

In figures 4.13 to 4.16 we see the impact the prediction error limiting vector picking scheme has on encoded sequences. These results seem to indicate that

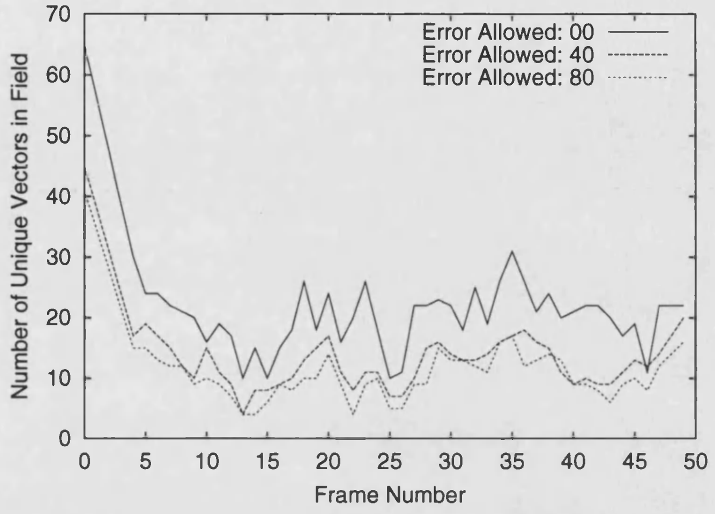


Figure 4.9: The number of vectors used in the first 50 frames of the foreman sequence. Dynamic picking was enabled and vectors were picked until the improvement each frame dropped below a certain limit.

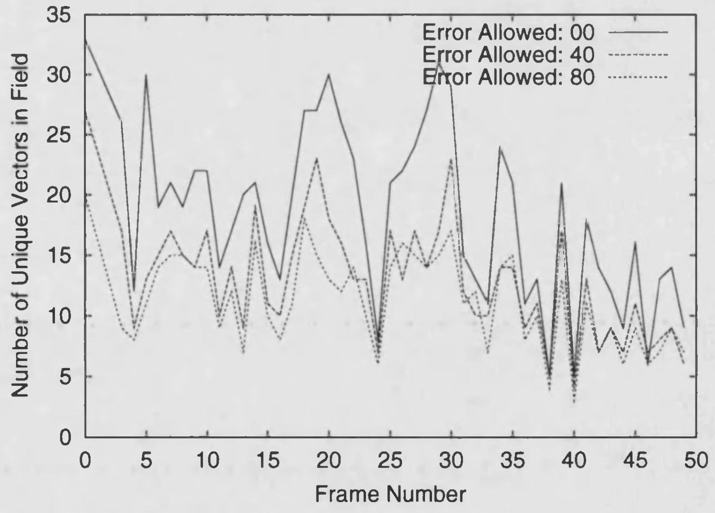


Figure 4.10: The number of vectors used in the first 50 frames of the carphone sequence. Dynamic picking was enabled and vectors were picked until the improvement each frame dropped below a certain limit.

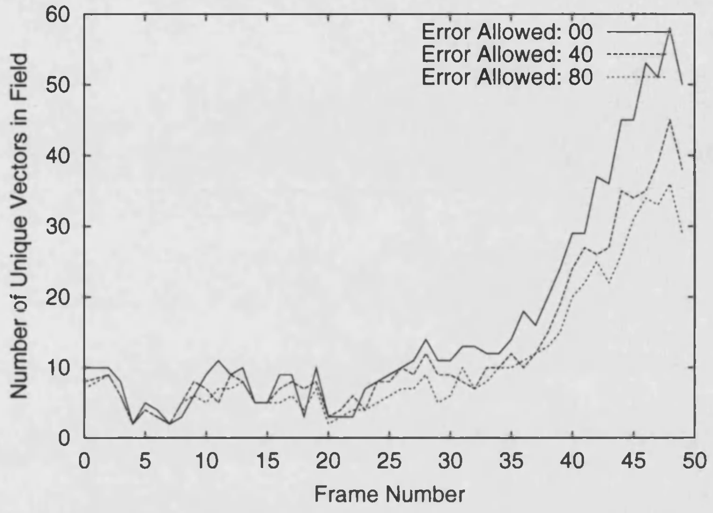


Figure 4.11: The number of vectors used in the first 50 frames of the suzie sequence. Dynamic picking was enabled and vectors were picked until the improvement each frame dropped below a certain limit.

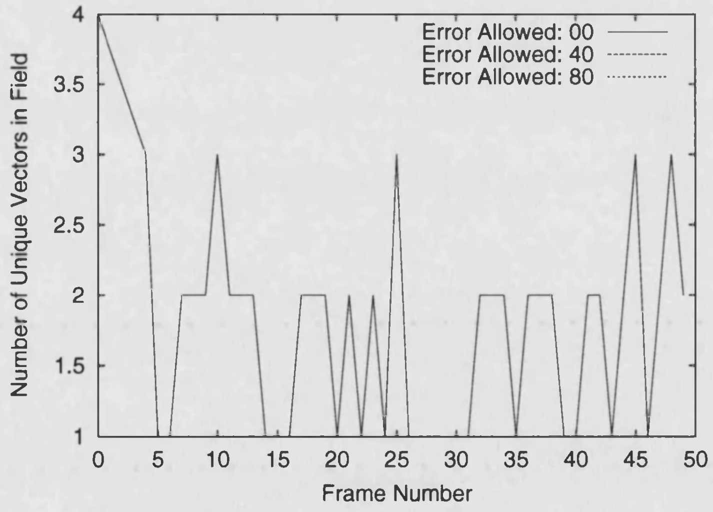


Figure 4.12: The number of vectors used in the first 50 frames of the container sequence. Dynamic picking was enabled and vectors were picked until the improvement each frame dropped below a certain limit.

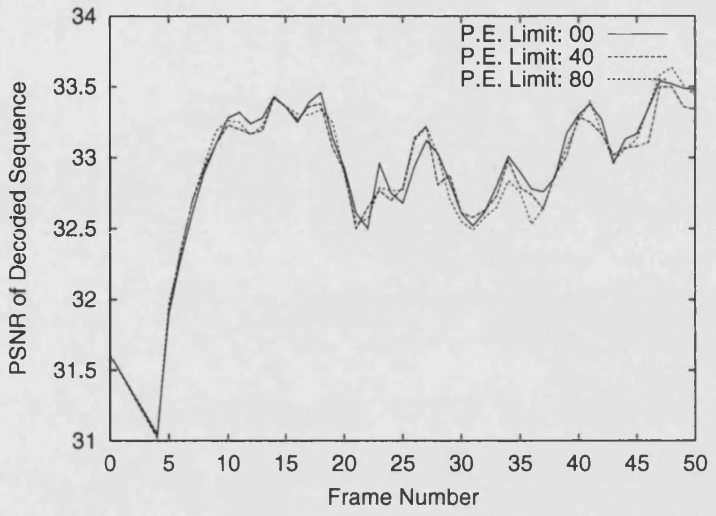


Figure 4.13: The result of coding the first 50 frames of the foreman sequence using prediction error limited vector picking.

although the number of vectors in the fields varies, the quality of the overall results remain more or less the same. Once again the result for the container sequence shows only one line as the vector selection strategy has no effect due to the very small amount of motion in the sequence.

4.7 Summary

In this chapter we have further looked at ways of controlling the number bits allocated to the motion description. Three vector selection strategies have been proposed, the Histogram Method, the Metric Method and the Full Search Method. Each of these tried to limit the number of vectors which are used to create the motion vector field. It has been shown the there is a strong correlation between the number of unique vectors in the field and the cost in bit required to encode the field, (figure 4.7).

In the next chapter we will explore an alternative representation for the motion description which may prove more favourable when the number of vectors in a motion description has been limited, (by one of the above techniques for example).

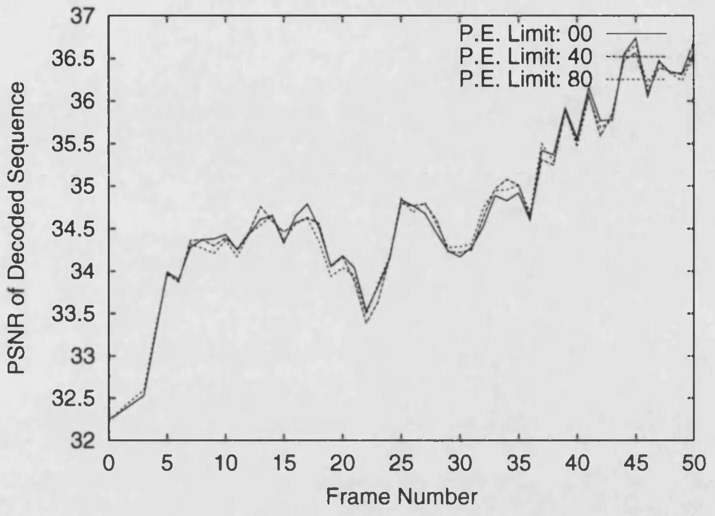


Figure 4.14: The result of coding the first 50 frames of the carphone sequence using prediction error limited vector picking.

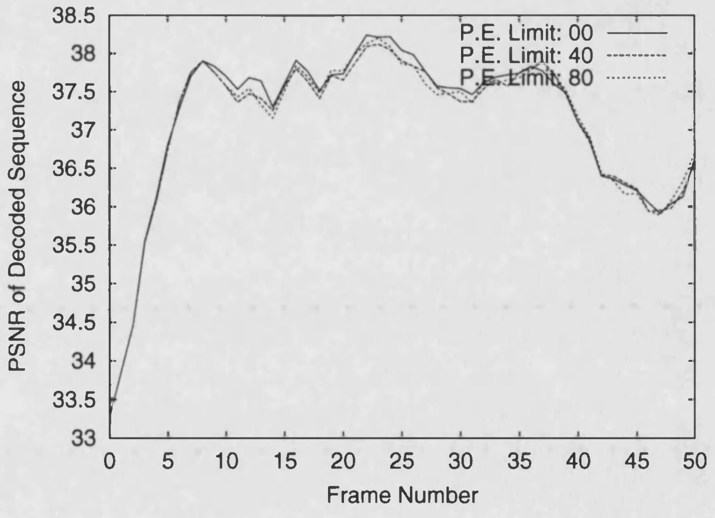


Figure 4.15: The result of coding the first 50 frames of the suzie sequence using prediction error limited vector picking.

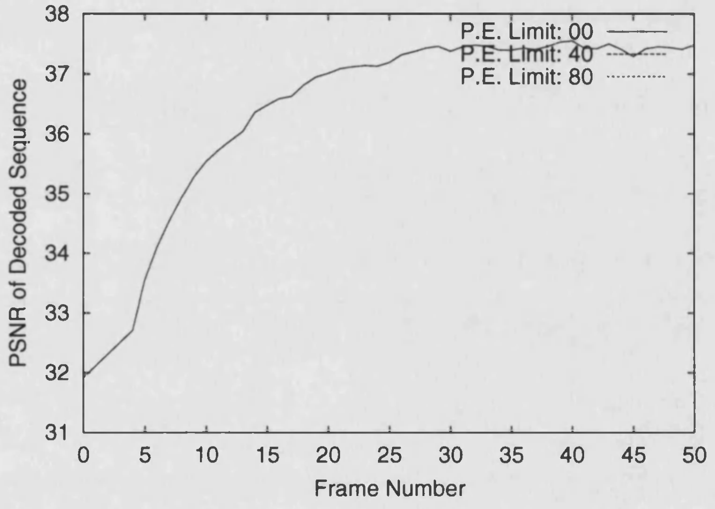


Figure 4.16: The result of coding the first 50 frames of the container sequence using prediction error limited vector picking.

Chapter 5

The List Mapping Motion Description

The traditional method for describing motion vector fields involves simply stating the horizontal and vertical vector components for each image block. The List Mapping Motion Description[63, 64] is an alternative method for describing such fields.

Rather than assign each image block a pair of vector components, it is instead assigned a single index number. This index number refers to a corresponding table of vectors in which the actual vector components can be found. In the context of the List Mapping Motion Description (LMMD) this table is referred to as the vector *list* and the array of indices as the vector *mapping*. Figure 5.1 shows a traditional motion vector field and the equivalent LMMD is shown in figure 5.2.

This form of motion description should be more suitable in cases where there are a limited number of unique motion descriptions, such as those produced by the methods described in chapter 4.

As the LMMD is a form of global optimisation the whole image has to be encoded before any part of it can be transmitted. This is different to existing codecs such as H.263 which can transmit a block as soon as it is encoded, (when only local optimisation is used). The result of this is that the LMMD has frame level latency rather than block level latency which could affect some real time coding applications.

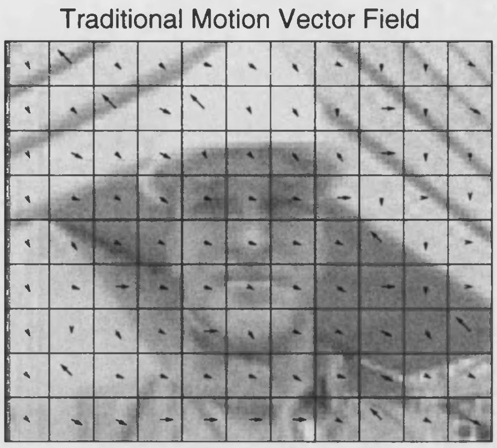
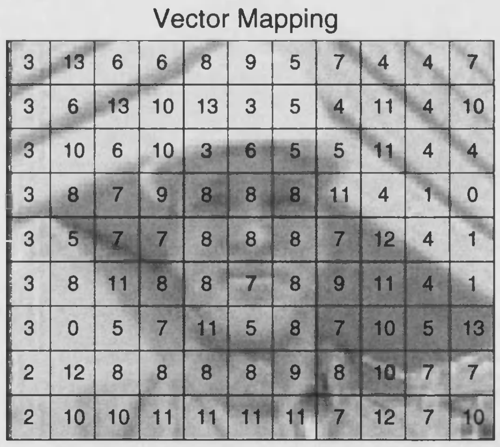


Figure 5.1: This is the traditional representation of a motion description. Each image block is assigned a motion vector.



3	13	6	6	8	9	5	7	4	4	7
3	6	13	10	13	3	5	4	11	4	10
3	10	6	10	3	6	5	5	11	4	4
3	8	7	9	8	8	8	11	4	1	0
3	5	7	7	8	8	8	7	12	4	1
3	8	11	8	8	7	8	9	11	4	1
3	0	5	7	11	5	8	7	10	5	13
2	12	8	8	8	8	9	8	10	7	7
2	10	10	11	11	11	11	7	12	7	10

Vector List

0	(0, 1)	7	(2, 1)
1	(1, 0)	8	(3, 1)
2	(1, 1)	9	(3, 2)
3	(1, 2)	10	(4, 2)
4	(0, 3)	11	(5, 0)
5	(2, 3)	12	(−4, −4)
6	(2, 2)	13	(−5, −5)

Figure 5.2: The LMMD uses a vector list and a mapping to describe the motion description. It should be noted that this is completely equivalent to the vector field shown in 5.1

5.1 Encoding the LMMD

One of the main features of the LMMD is that each vector is only stated once, in its entry in the list. From this it follows that each vector only needs to be encoded once. However, it is also necessary to encode the vector index for each image block. Although this mapping has been reduced from two values per image block (the vector components) to one value (the index) it is much harder to take advantage of any correlation in the index mapping. In section A.1 it was shown how H.263 uses the correlation in traditional vector fields can be used to reduce the cost of encoding them. This works by predicting the vector to be encoded from those previously transmitted and only encoding the (hopefully small) difference between this prediction and the actual vector. In the case of the LMMD it makes no sense to take the difference between two indices, as (at the most general level) there is no well defined correlation between the index and the vector it refers to.

5.2 List Encoding

The following simple method is used to encode the vector list. The components of the first vector in the list is encoded using Reversible Variable Length Coding (RVLC). Each subsequent vector is subtracted from the previously encoded vector and the components of these difference vectors are encoded using the same RVLC scheme. An example of this scheme is shown in figure 5.3.

The cost in bits (c) when encoding a given vector component v_a using the RVLC is given by:

$$c = 2(\lfloor \log_2 v_a \rfloor) + 1 \tag{5.1}$$

Although the encoding method for the list has been fixed, there is still scope for optimising the cost of transmitting the list. This encoding method can be thought of as equivalent as tracing a path from one vector to the next, starting at the origin. Although the cost of encoding the path is not the same as the Euclidean length of the path it does have the following properties;

- The cost of going from A to B via Z is always greater than or equal to the cost of going from A to B directly.

- The cost of going from A to B is the same as going from B to A.

Vector	Previous Vector	Difference	RVLC bit cost
$(-5,-5)$	–	$(-5,-5)$	14
$(-4,-4)$	$(-5,-5)$	$(1,1)$	6
$(0,1)$	$(-4,-4)$	$(4,5)$	14
$(0,3)$	$(0,1)$	$(0,2)$	6
$(1,0)$	$(0,3)$	$(1,-3)$	8
$(1,1)$	$(1,0)$	$(0,1)$	4
$(1,2)$	$(1,1)$	$(0,1)$	4
$(2,1)$	$(1,2)$	$(1,-1)$	6
$(2,2)$	$(2,1)$	$(0,1)$	4
$(2,3)$	$(2,2)$	$(0,1)$	4
$(3,1)$	$(2,3)$	$(1,-2)$	8
$(3,2)$	$(3,1)$	$(0,1)$	4
$(4,2)$	$(3,2)$	$(1,0)$	4
$(5,0)$	$(4,2)$	$(1,-2)$	8
		Total Cost	94

Figure 5.3: An example of encoding a vector list using differential RVLC.

The optimum encoding can be obtained by finding the shortest path (in terms of bit cost) between all the vectors. This is an example of the classic Travelling Salesman Problem, (see appendix C for details).

5.2.1 Branch and Bound Insertion Algorithm

It is possible to achieve an optimal solution to the Travelling Salesman Problem (TSP) without having to evaluate all possible routes between the cities. The Branch and Bound Insertion Algorithm (BABIA)[65] uses the principle that a journey from A to B *via* C will be no shorter than going from A to B directly to eliminate some routes.

Suppose we have a 10 city problem which we wish to solve. The first step of the BABIA algorithm involves finding any tour which traverses all 10 cities and measuring its length. This is our initial estimate for the optimum tour. We now create a sub-tour which visit any two of the cities. From this sub-tour we can create 24 new three[1] city sub-tours by inserting each of the remaining 8 cities into one of three places:

1. Before the first city,

[1]The number of tours may be reduced depending upon the symmetries of the problem.

2. Between the first and second city,

3. After the second city.

Each of these tours can be further extended by placing each of the 7 remaining cities in one of the four appropriate places, and so on. Now, owing to the Euclidean principle stated above, each of the three city tours must be longer than (or equal to) the two city tour, and each four city tour must be longer than (or equal to) the three city tour from which it was generated. As a result, if we ever generate a sub-tour which is longer than our estimate of the optimum tour we know that the sub-tour in question cannot possibly be part of the optimum tour, so we can ignore it, and all tours descended from it. Each time our algorithm produces a tour which covers all 10 cities, we check its length and compare it with our current best tour. If the new complete tour is shorter than our current estimate, it becomes our new estimate. As more an more 10 city tours are evaluated our estimate improves, which allows us to prune sub-tours earlier.

While the solution to the TSP will give the optimal ordering for the vector list, the computational expense makes it impractical for long vector list, even with the use of algorithms[66] such as BABIA.

The following methods were investigated as faster alternatives to the optimal solution.

5.2.2 Genetic Algorithm Solution to the TSP

Much research has been done in trying to use genetic algorithms to solve the TSP[67, 68, 69, 70]. For the simple genetic algorithm used in this work the following crossover technique was used when mating two existing 'parent' tours (p_1 and p_2) to produce two new child tours (c_1 and c_2). The length of the parent tours is taken to be L.

1. A random number, r, which lies between 1 and $L -$ `splice_length`, is generated.

2. The cities between r and $r +$ `splice_length` are copied from p_1 to the corresponding locations in c_1 and also from p_2 to c_2

3. The remaining locations in c_1 (those before and after the spiced region) are then filled from p_2 by copying across all unused cities in a sequential fashion. Similarly, the remaining locations in c_1 are filled from p_1.

5.2.2.1 Crossover Example

Suppose the parent tours p_1 and p_2 are as follows:

$$\begin{aligned} p_1 &= 6\ 1\ 0\ 9\ 2\ 3\ 7\ 8\ 4\ 5 \\ p_2 &= 5\ 7\ 4\ 8\ 0\ 3\ 1\ 9\ 2\ 6 \end{aligned}$$

and that `splice_length` $= 3$ and $r = 4$.

The above procedure can now be used to construct c_1 and c_2. In the following example spliced values are in italic, filled values in bold and empty values are represented with a dot ($\cdot$).

- Copy across the spliced regions to the corresponding locations in the children.

$$\begin{aligned} p_1 &= 6\ 1\ 0\ 9\ 2\ 3\ 7\ 8\ 4\ 5 \\ c_1 &= \cdot\ \cdot\ \cdot\ \cdot\ \mathit{2}\ \mathit{3}\ \mathit{7}\ \cdot\ \cdot\ \cdot \end{aligned}$$

$$\begin{aligned} p_2 &= 5\ 7\ 4\ 8\ 0\ 3\ 1\ 9\ 2\ 6 \\ c_2 &= \cdot\ \cdot\ \cdot\ \cdot\ \mathit{0}\ \mathit{3}\ \mathit{1}\ \cdot\ \cdot\ \cdot \end{aligned}$$

- Then fill in the remaining locations. This example will show how c_1 is completed using the elements from p_2.

  First all the cites already present in c_1 are removed from a copy of p_2.

$$p_2 = 5\ \cancel{7}\ 4\ 8\ 0\ \cancel{3}\ 1\ 9\ \cancel{2}\ 6$$

  The remaining cities are then copied in a sequential manner to the free spaces in c_1. This gives

$$c_1 = \mathbf{5\ 4\ 8\ 0}\ \mathit{2}\ \mathit{3}\ \mathit{7}\ \mathbf{1\ 9\ 6}$$

  Repeating the process for c_2 would yield:

$$c_2 = \mathbf{6\ 9\ 2\ 7}\ \mathit{0}\ \mathit{3}\ \mathit{1}\ \mathbf{8\ 4\ 5}$$

5.2.3 Sorting by Component

The most straightforward method of sorting the list is to arrange the vectors in order of increasing x or y component, with any ties resolved by considering the other component.

5.2.4 Sorting by Magnitude and Angle

Each vector is converted from Cartesian form (x,y) to polar form (r,θ) and these values are used to sort the vectors. There are two approaches which can be taken in this case;

- The list can be sorted by magnitude, r, and any ties resolved by sorting by angle, θ.
- The list can be sorted by angle, θ, and any ties resolved by sorting by magnitude, r.

5.2.5 Sector Sort

This method derives its name from the 'sectors' that the Cartesian space is divided into in order to group together vectors in the list. It is essentially an extension of the angle/magnitude sort with a more coarse division by angle.

Figure 5.4 shows an example where the vectors have been split in to six equally sized segments. The vectors in each segment are sorted according to magnitude, and the segment containing the vector with the smallest magnitude is selected as the 'initial' sector. If there is a tie for the initial sector then the sector whose median is closest to the polar 0° axis is selected.

The vectors from the initial sector are taken (in ascending order of magnitude) and used to form the beginning of the reordered list. The next (non-empty) sector in an anti-clockwise direction is then considered. This sector is added to the reordered list in either ascending or descending magnitude depending on which gives the lower encoding cost. This is determined by considering the bit cost of encoding the difference between the last vector in the reordered list and the first and last vectors in the sector currently under consideration.

5.2.6 List Coding Examples

For each of the methods described a figure is given showing the path traced when an example set of 14 vectors are traversed in the order into which they are sorted,

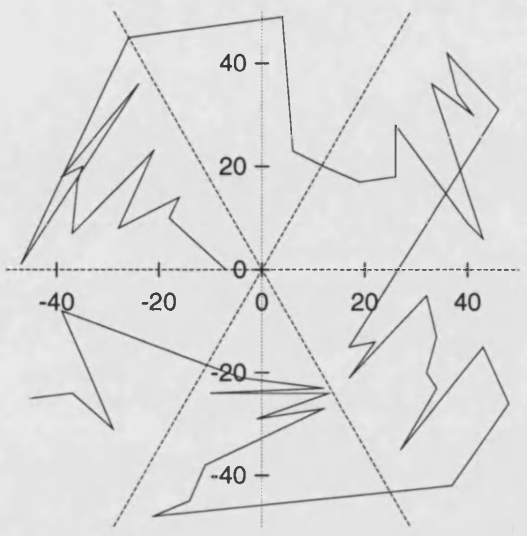


Figure 5.4: An example of a vector list sorted into sectors as used by the Sector Sort.

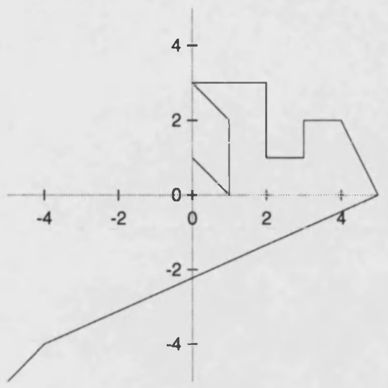
see Figure 5.5. The length of this path is an indication of the cost of encoding the list.

5.3 Mapping Encoding

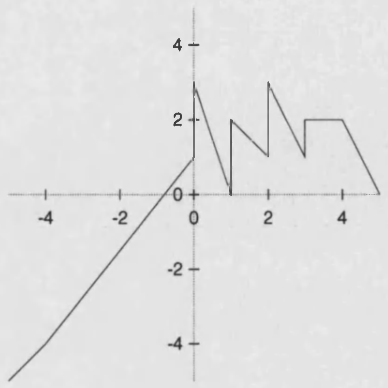
After the vector list has been encoded it is necessary to encode the array which maps the vectors to each individual image block. This mapping contains a list of indices to the vectors in the list and often contains correlation which can be exploited during coding.

The primary form of correlation in the mapping is due to neighbouring blocks being assigned to the same vector and therefore having the same index. It is also interesting to note however, that due to the nature of the list encoding it is possible that neighbouring indices represent similar vectors. This stems from the fact the the list encoding schemes try to reduce the cost of encoding the list by ordering the list to minimise the distance between neighbouring vectors in the list. As the index used in the mapping corresponds to the position of the vector in the list it is worth making some attempt to take advantage of possible correlation between indices.

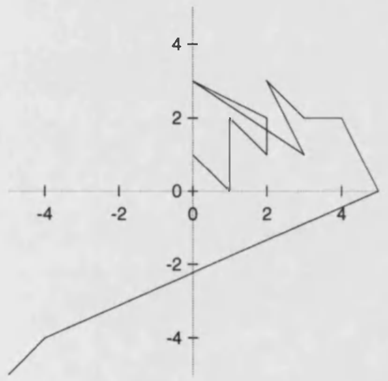
A number of techniques for encoding the mapping are proposed. As with the



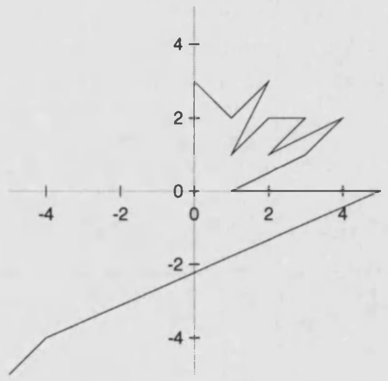
(a) TSA: 80 bits



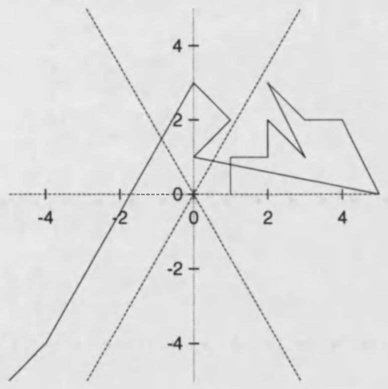
(b) Component: 94 bits



(c) Magnitude/Angle: 94 bits



(d) Angle/Magnitude: 98 bits



(e) Sector: 90 bits

Figure 5.5: Comparing different methods for sorting the vector list. For each method the path traced by the sorted list is shown, along with the cost of encoding the list.

vector list techniques, each have their strengths and weaknesses.

5.3.1 Raw Encoding

This method simply encodes each element of the mapping with a fixed length code. As the number of vectors in the list is already known, length of code needed to uniquely identify any index in the mapping can be deduced.

For a mapping containing indices between $0 \leqslant n \leqslant N$ the cost, in bits, to encode each index is given by;

$$N_{\text{bits}} = \lfloor \log_2(N) \rfloor + 1 : N > 0 \tag{5.2}$$

Apart from the fact that no attempt is made to take advantage of any correlation between indices, this method is rather inefficient for many values of N. However, it is fast and the cost of encoding can be determined without the need to iterate over the mapping (the cost simply being N_{bits} times the number of blocks in the mapping).

5.3.2 Linearised Map Encoding

5.3.2.1 The Snake Scan

The remaining method operate on a linearised version of the mapping. The most traditional method of transforming a two dimensional array into a vector is to select the elements in raster order. However, for this work an alternative method known as the snake scan will be used. This technique begins the same way as a raster scan; by picking data sequentially across the first raster, from left to right. Rather than repeating this process, the next line is linearised from *right to left*. The third line is then traversed left to right, and so on. This method ensures that the neighbours of each index in the linearised vector are also neighbours in the original two dimensional mapping. It is also possible to form the vector by scanning *down* the first column, and *up* the second. These two scan are shown in figure 5.6. The resulting vector can be encoded by the following techniques.

Run Length Encoding Runs of symbols in the vectors are encoded using a standard run length scheme. The symbol is sent raw (see equation 5.2), and then length of run encoded. For QCIF size images (with 99 indices in the mapping) the the maximum length of run permitted is 8.

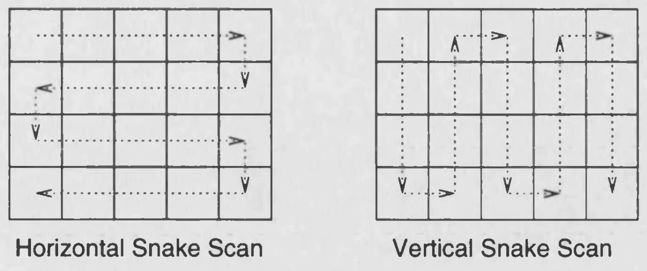


Figure 5.6: Paths of the horizontal and vertical snake scans.

Predictive Encoding The first symbol is sent raw. Each subsequent symbol is compared to the previous one, and a bit flag is used to indicate whether or not they are the same. If they differ the symbol needs to be sent raw.

Both of the above methods can be used with either the horizontal or vertical scanning giving a total of five techniques, with the addition of the raw encoding.

5.4 Summary

In this chapter we have introduced the LMMD as an alternative to the traditional motion vector field. The LMMD is comprised of two parts, the vector list and the mapping. Techniques for coding each of these have been presented, (both optimal and sub-optimal). In the next chapter the LMMD will be combined with the Metric Method vector selection strategy and compared with a standard H.263 encoder.

Chapter 6

LMMD Results

6.1 Optimising the LMMD List

Five of the methods described in section 5.2 were used to optimise the LMMD list. These were;

1. The *xy* sort,
2. The magnitude sort,
3. The angle sort,
4. The sector sort,
5. The Travelling Salesman Algorithm.

In order to evaluate these method a test set of list data was generated from the following five QCIF sequences:

1. Foreman (400 frames)
2. Carphone (382 frames)
3. Football (90 frames)
4. News (300 frames)
5. Container (300 frames)

Each sequence was encoded using a standard H.263 encoder running in 'Unrestricted Motion Vector' mode, (Annex D). The bit rate used was 128 kbits/s and the sequences were run at 30 fps with no frames skipped (after the initial I-frame) unless otherwise specified. For each input/reference frame pair in the sequences the Metric Method (section 4.2) was used to produce vector lists for between 2 and 30 unique vectors. (Some frames did not have 29 unique vectors.) A summary of the motion descriptions produced is given in table 6.1.

	Number of Motion Descriptions				
N	Foreman	Carphone	Football	News	Container
2	395	379	85	260	163
3	394	379	85	246	69
4	385	378	85	237	34
5	374	378	85	210	15
6	363	372	85	160	8
7	352	368	85	125	3
8	338	366	85	102	1
9	331	356	85	92	
10	324	345	85	87	
11	318	337	85	78	
12	315	329	85	62	
13	311	323	85	48	
14	300	316	85	35	
15	288	305	85	20	
16	273	293	85	12	
17	260	280	85	4	
18	247	263	85	4	
19	234	249	85	3	
20	218	236	85	2	
21	204	228	85	2	
22	187	207	85	1	
23	178	191	85	1	
24	167	181	85	1	
25	157	162	84	1	
26	145	142	84		
27	134	138	83		
28	122	128	83		
29	112	118	83		
Total	9084	8499	2372	1793	293

Table 6.1: Summary of the data used to generate the results.

As the Foreman, Carphone and Football sequences contain significant motion they all manage to produce motion descriptions with up to 29 unique vectors. However, the other sequences contain more simple movement and as a result produce less motion data.

The vector list from each motion description was encoded using each of the five techniques mentioned above. However, there were constraints placed on the Travelling Salesman Algorithm (TSA) in order to reduce computation time. If the number of unique vectors (N) less than 16 the TSA was used to yield the optimal solution. For $N \geqslant 16$ however, the BABIA algorithm (see section 5.2.1 used was interrupted after 1 second and the current 'best result' was used.

For each sequence two results were plotted. First the average cost in bits for each N was found for each of the methods. Also the number of times each method produced the best optimisation was recorded. The sum of the 'Normalised Number of Wins' will sometimes be greater than one, as more than one technique may produce the best list. This can happen frequently for smaller N. These results are displays in figures 6.1 to 6.5.

As expected the TSA produced the best optimisation in all the cases it was allowed to run to completion ($N \leqslant 15$). The amount by which it beats the other methods also increases with N. However as N increases past 16 it fails to produce such good results. Towards the end of the graph ($N \geqslant 25$) the XY sort starts to produce the best results.

While in general the average cost of encoding the list seems to increase smoothly with the number of vectors in the list, there is a kink in the graph for the news sequence at 17 vectors, (figure 6.4). This is due to the fact that there are only four data samples being averaged to produce the lines. One of these is an anomalous, expensive result and causes a skew in the mean.

6.2 Encoding the LMMD Mapping

As with the LMMD lists, five mapping encoding methods (section 5.3) were evaluated using the motion descriptions from table 6.1. These were;

1. Raw Encoding
2. Horizontal Runlength Encoding
3. Horizontal Difference Encoding

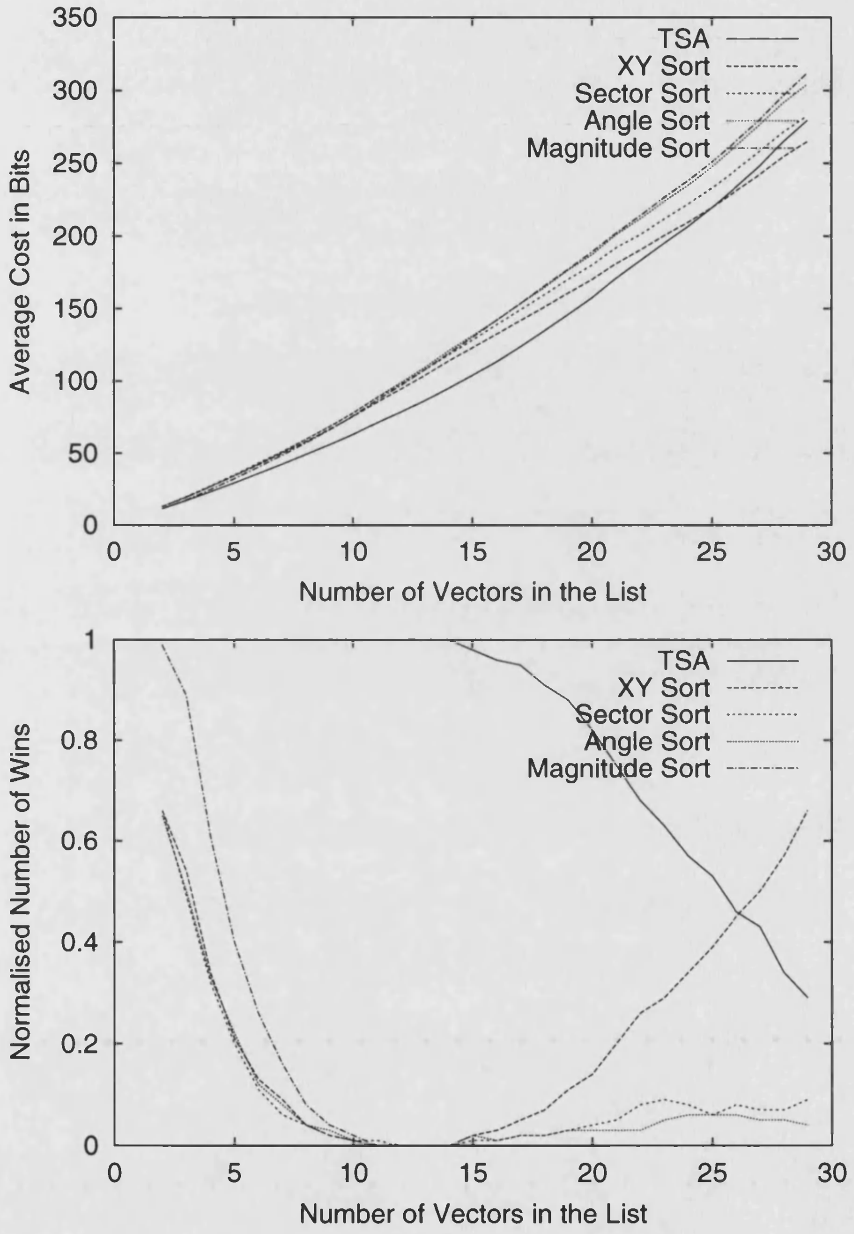
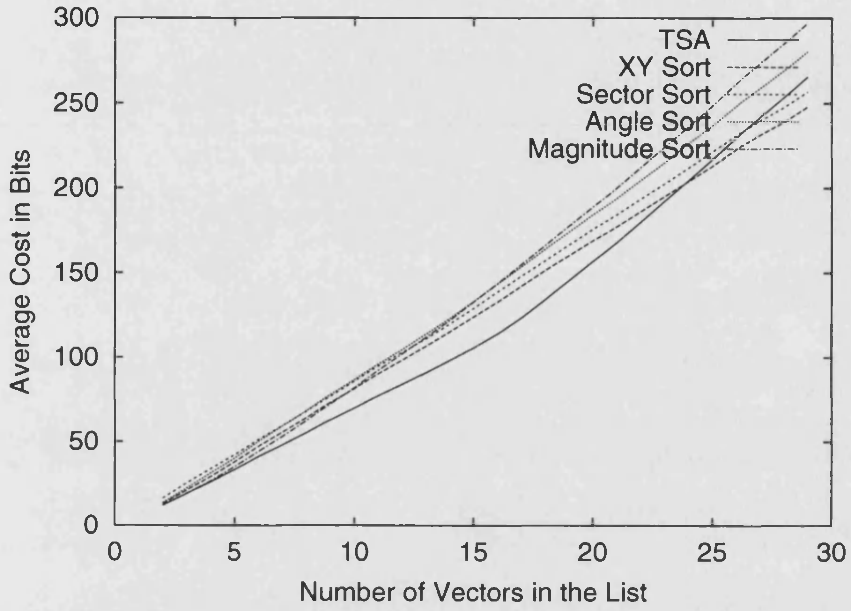


Figure 6.1: Optimised list encoding results for the foreman sequence.



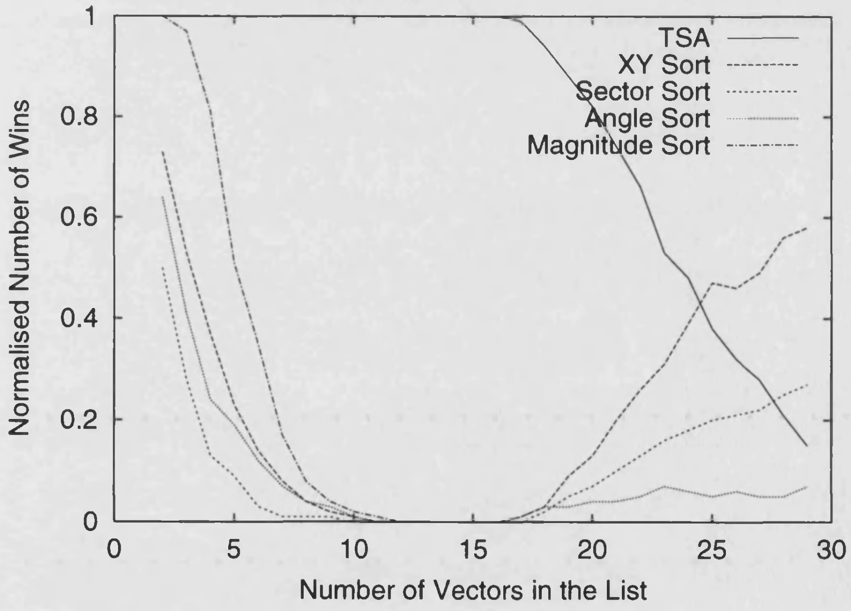


Figure 6.2: Optimised list encoding results for the carphone sequence.

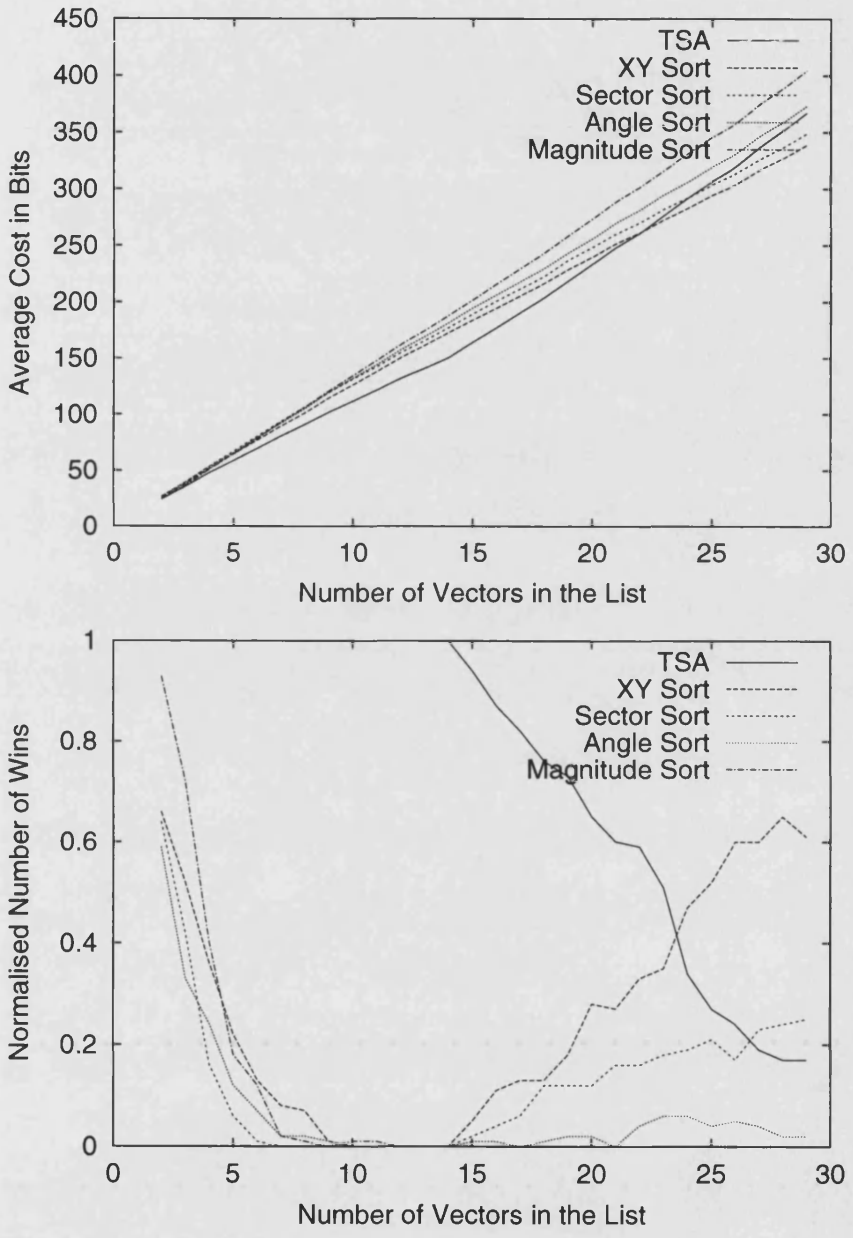


Figure 6.3: Optimised list encoding results for the football sequence.

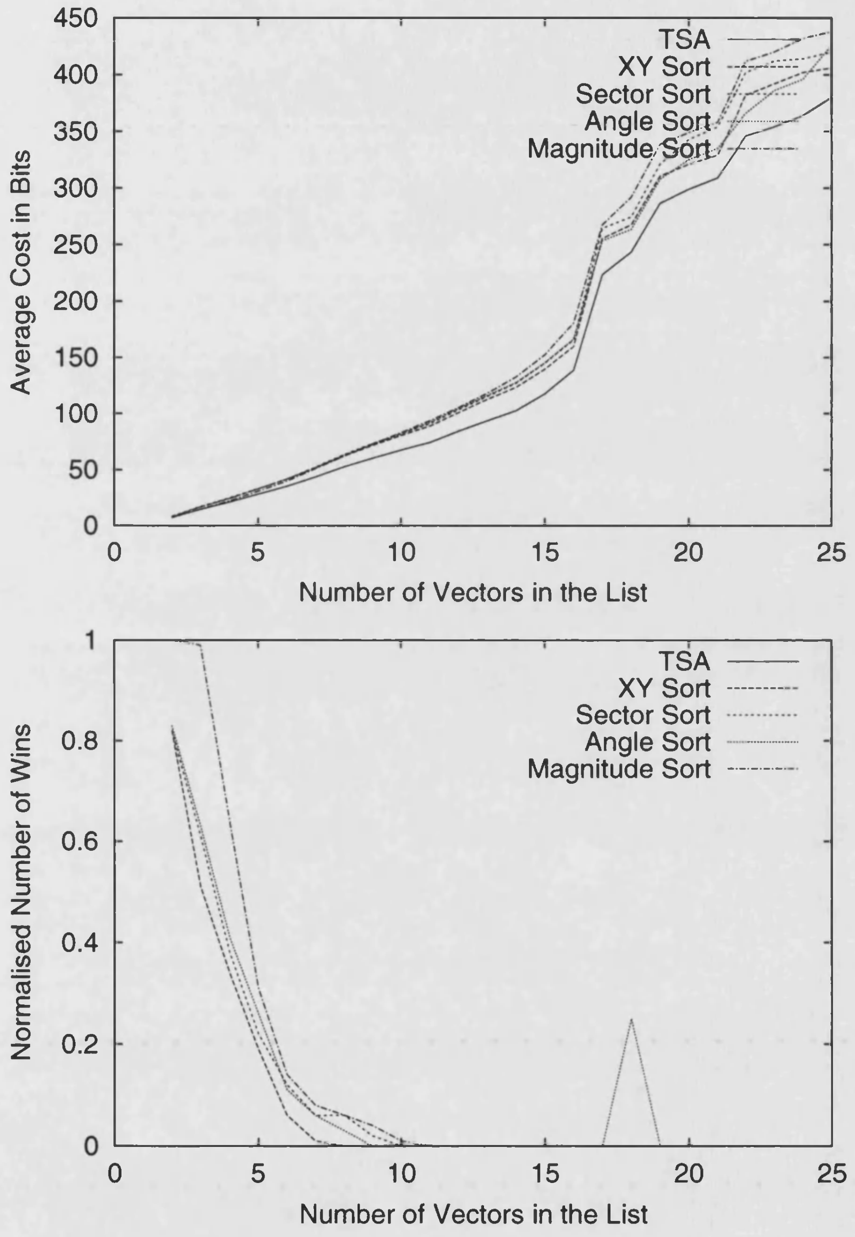
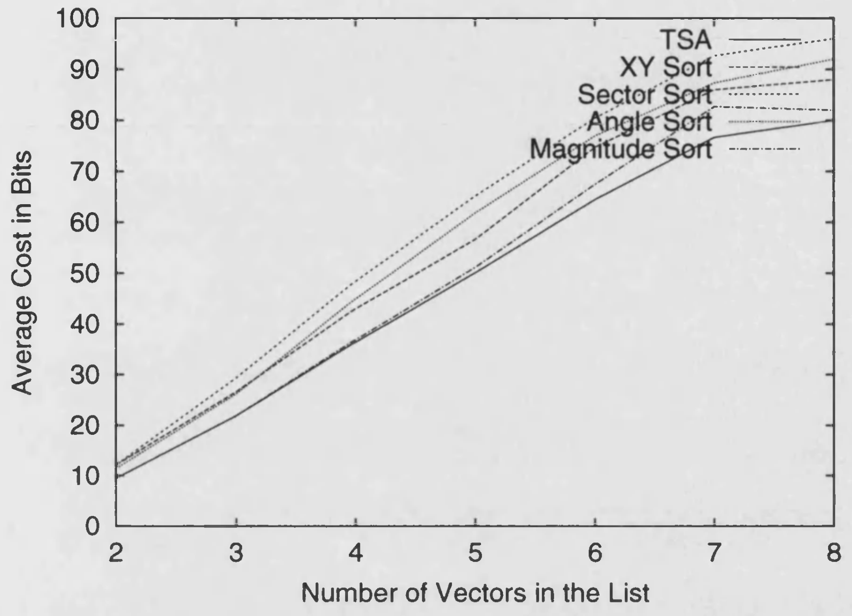


Figure 6.4: Optimised list encoding results for the news sequence.



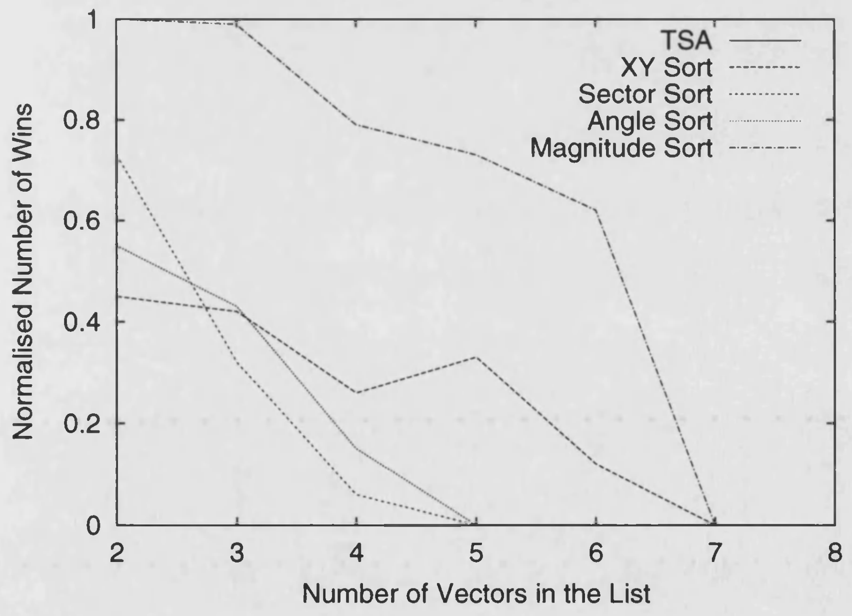


Figure 6.5: Optimised list encoding results for the container sequence.

4. Vertical Runlength Encoding

5. Vertical Difference Encoding

The results are shown in figures 6.6 to 6.10. In the high motion sequences (Foreman, Carphone and Football) the vertical DPCM method wins the majority of the time, although the horizontal DPCM also provides the best result on 10% to 20% of occasions. In these sequences the raw encoding starts to play a role as the number of unique vectors starts to rise above 25. As the number of vectors increases the correlation in the mapping is likely to decrease, and so the runlength and DPCM methods can cost more than they gain.

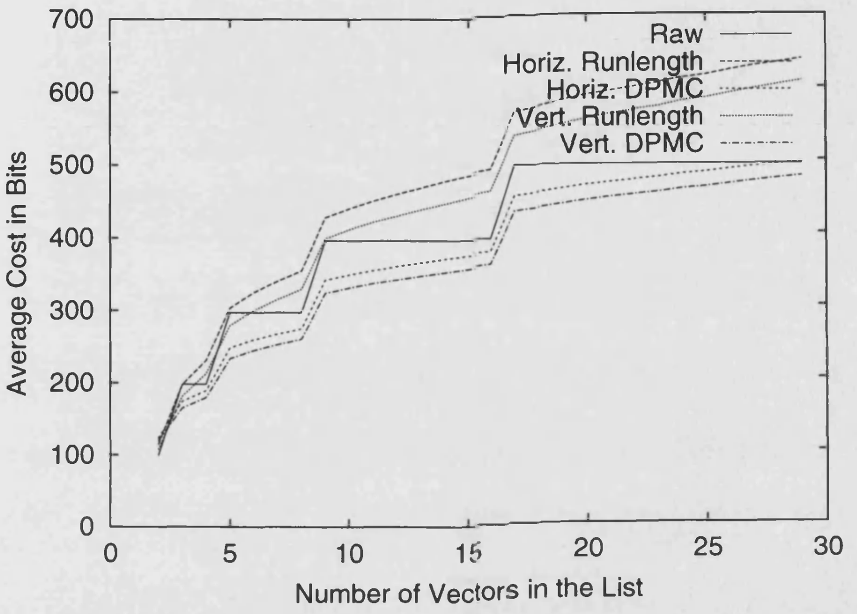
For the news sequence there is no method out performs the others until there are over 15 vectors in the list. At this point the horizontal DPCM becomes dominant. The results below this point however, indicate that a mixture of the methods is required in order to gain the most benefit from the LMMD style encoding.

Finally looking at the container sequence we see that once again, no single method is dominant. We must also be wary of the apparent drop in the cost of encoding the vector list when there are over six vectors. This is likely to be the result of averaging data from too few samples, (the number of which can be seen in table 6.1).

The range of results shows that although the DPCM methods are predominantly more successful they are not exclusively so. As it is quite difficult to predict which scan will work most effectively with a given type of sequence the combined approach is sensible. Of course, each method used in the comparison increases the computational cost, but this increase is insignificant when compared with other areas of complexity within the codec.

For a given list/mapping pair the cost of encoding the mapping is roughly two to three times more expensive than encoding the list.

The cost of encoding the mapping is very sensitive to $\lfloor \log_2 N \rfloor$ and this can be seen by the jumps in the cost which occur at $N = 2, 4, 8, 16$. This is due to the fact that all the methods are sensitive to the raw encoding cost of a mapping index. It should be possible to smooth the graph out a bit by using a more sophisticated raw coding technique.



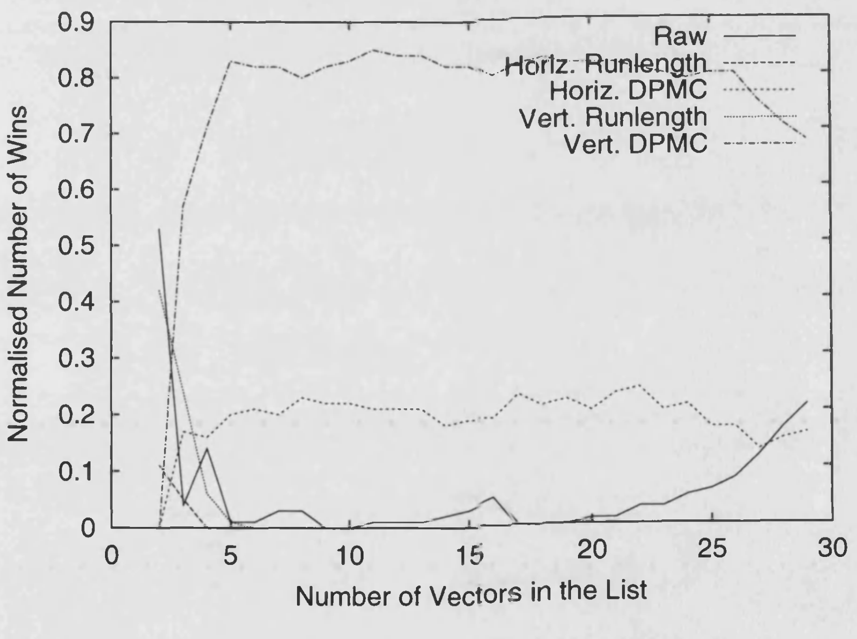


Figure 6.6: Mapping encoding results for the foreman sequence.

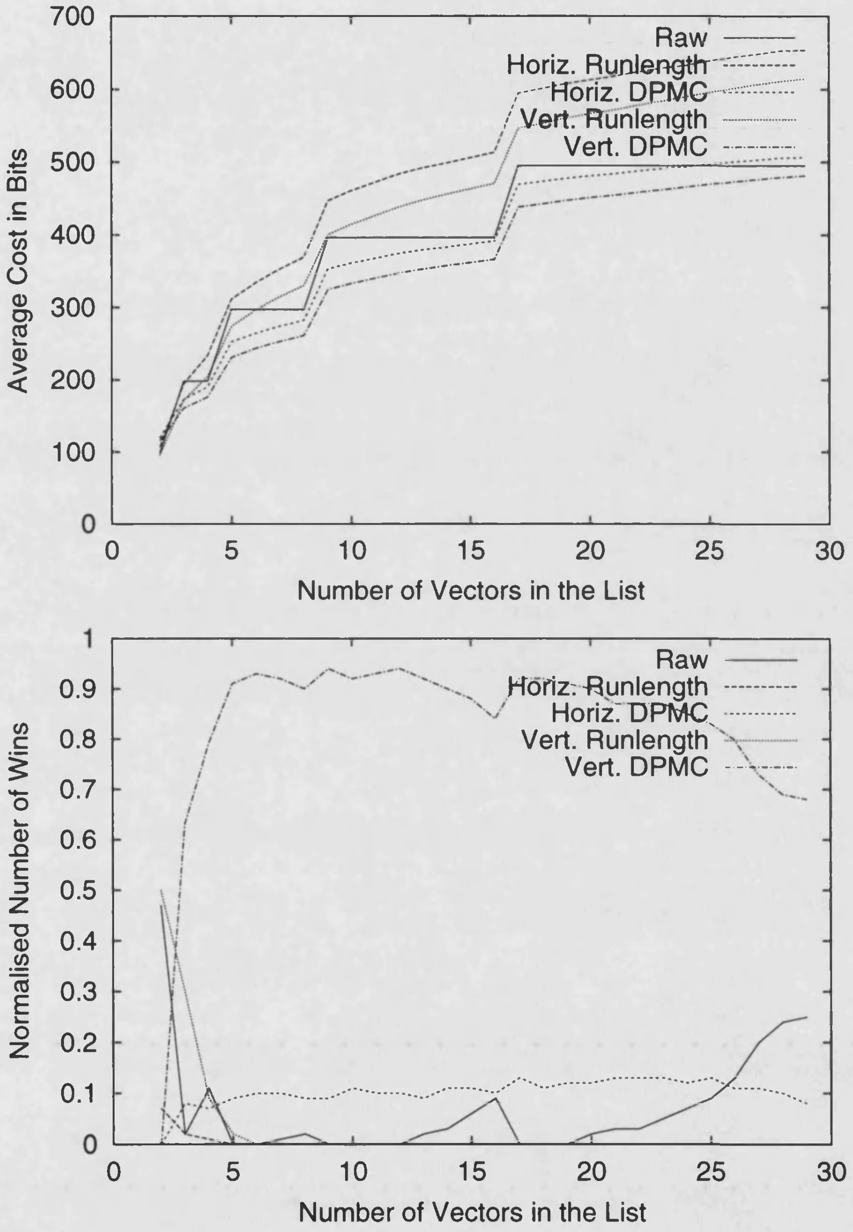


Figure 6.7: Mapping encoding results for the carphone sequence.

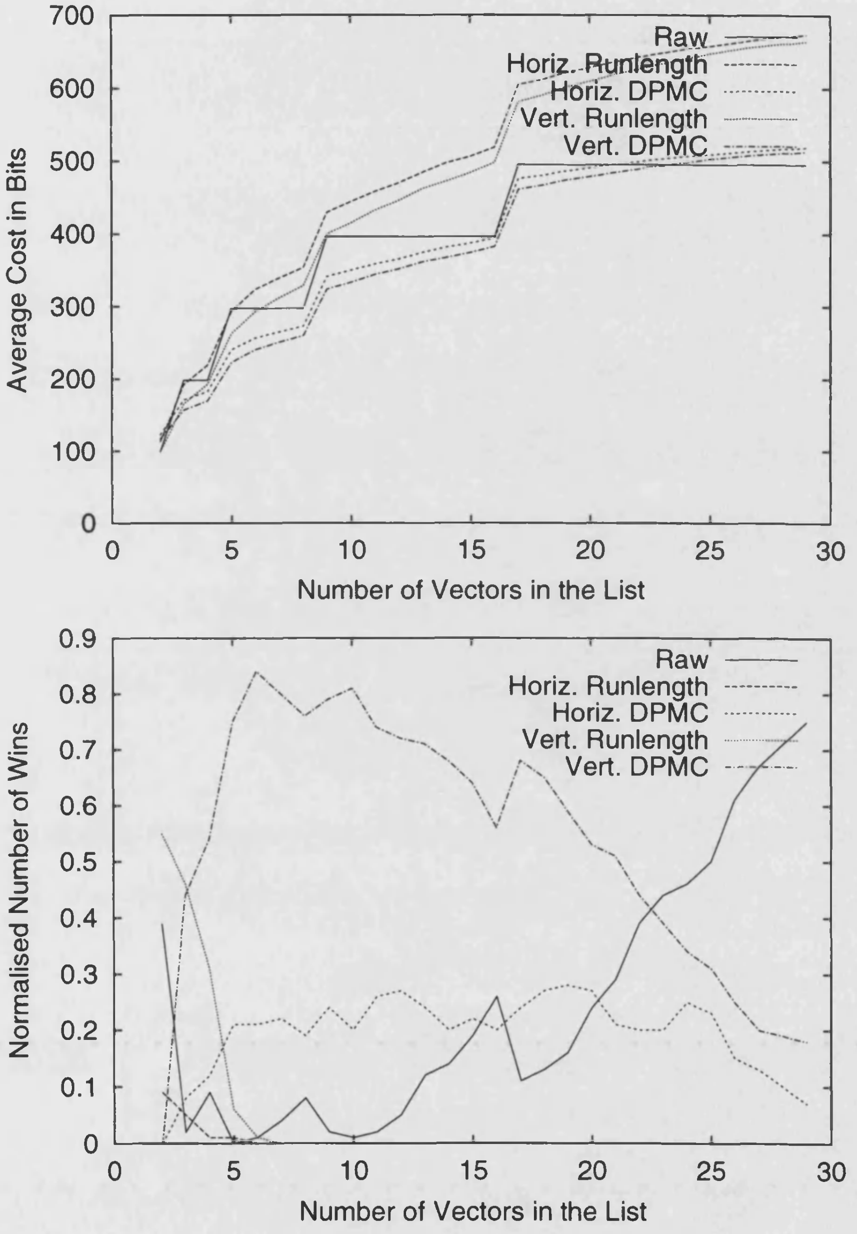


Figure 6.8: Mapping encoding results for the football sequence.

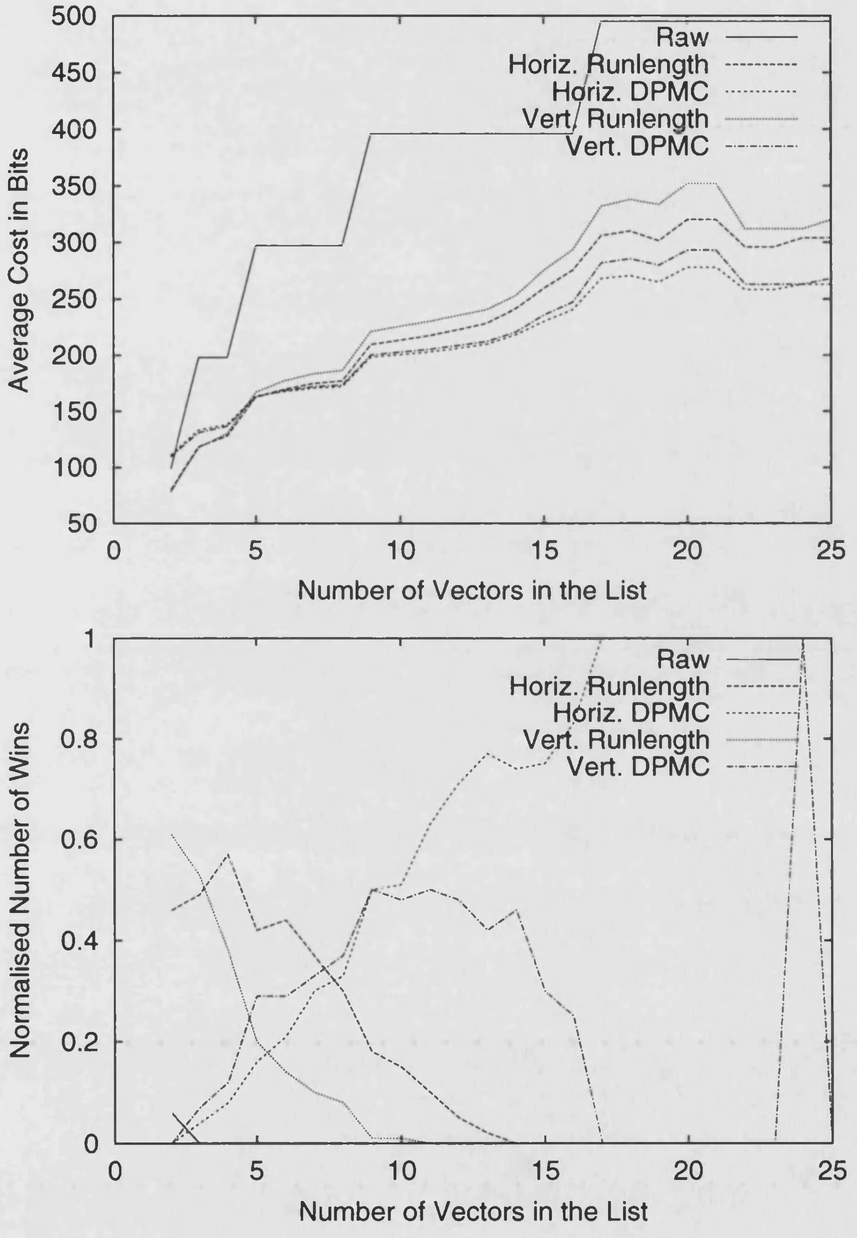


Figure 6.9: Mapping encoding results for the news sequence.

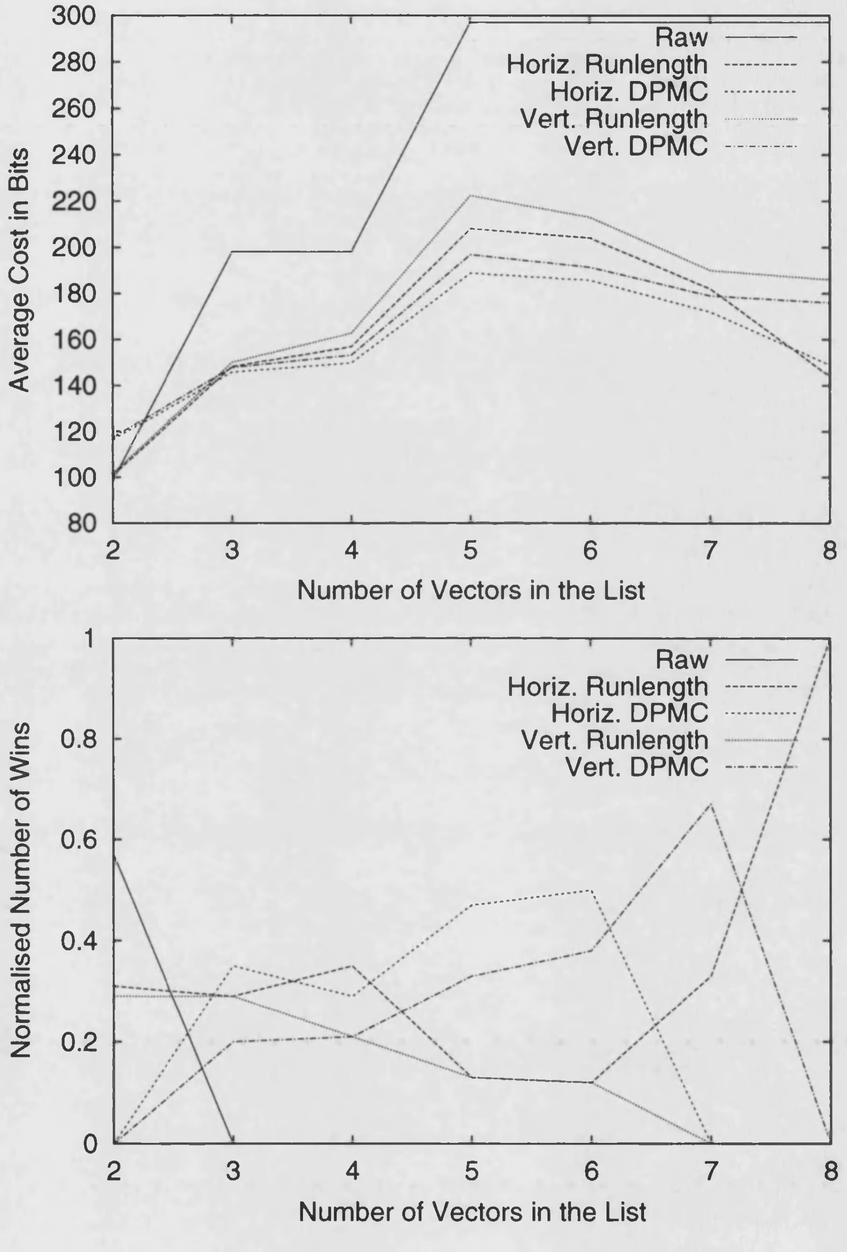


Figure 6.10: Mapping encoding results for the container sequence.

6.3 Combining the List/Mapping Encodings

Until now we have examine the list and mapping encoding separately. We now combine them to produce a complete motion description which contains sufficient information to produce a predicted image from a reference image.

Using, once again, the data from table 6.1 complete motion descriptions were evaluated using the LMMD style encoding. For each LMMD the equivalent vector field was generated and encoded using H.263 style encoding as described in section A.1. In each case the motion vectors for all the image blocks were encoded, (something which isn't guaranteed in a complete H.263 encoder). The results for each of the four sequences are shown in figures 6.11 to 6.15.

In all cases the LMMD encoding does well for $N \leqslant 16$, however there are two factors contributing to its poorer performance after that point. One is the increased cost of the mapping. Also after this point the performance of the TSA declines due to the fact it is no longer given sufficient time to generate the optimal encoding of the list.

For the sequences with less motion (news and container) the LMMD encoding always beats traditional H.263. (This should be the case as the LMMD has been designed to perform better with lower N.)

6.4 H.263 Codec Based Results

In order to test the LMMD completely it was integrated as part of an H.263 codec. The codec used was as described in [1] with the LMMD style motion description coding replacing the vector field encoding. Instead of sending the motion vector data within the macroblock structure, as the standard H.263 codec does, the list and mapping data was sent at the picture level, before any macroblocks have been transmitted.

Seven different tests were carried out the details of which are shown in table 6.2. Test 0 uses the standard H.263 encoding with no modifications, it provides a baseline against which the other tests can be evaluated. Tests 1–3 use the Metric Method vector selection strategy to reduce the number of vectors in the field. The field is still encoded using a standard H.263 encoder to produce a H.263 compliant stream. In test 1 there is no dynamic selection of N, the number of vectors in the field, the Metric Method is allowed to use as many vectors as it is supplied with. Test 2 implements 'No single vector' type dynamic picking, (section 4.6.1). This method

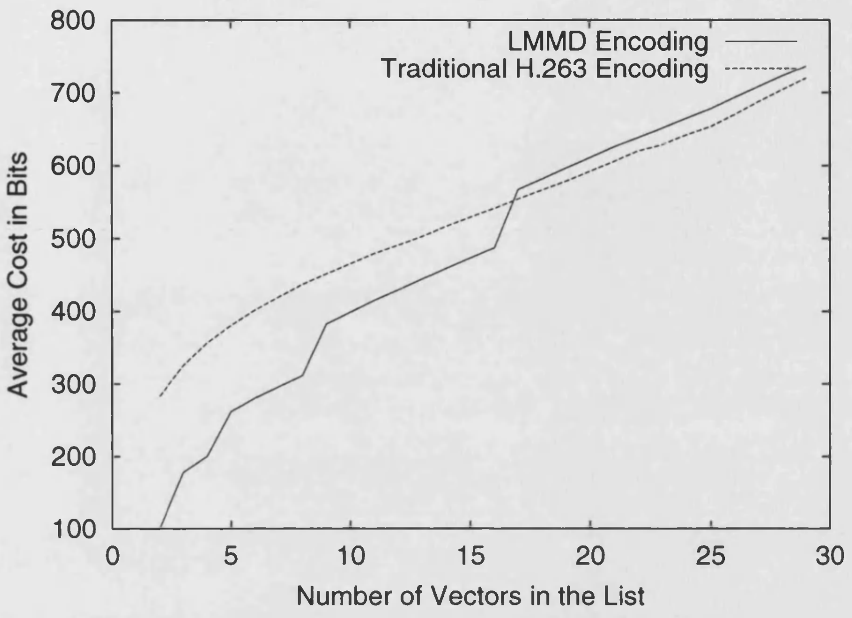


Figure 6.11: Results comparing the LMMD style encoding against H.263 style encoding for the foreman sequence.

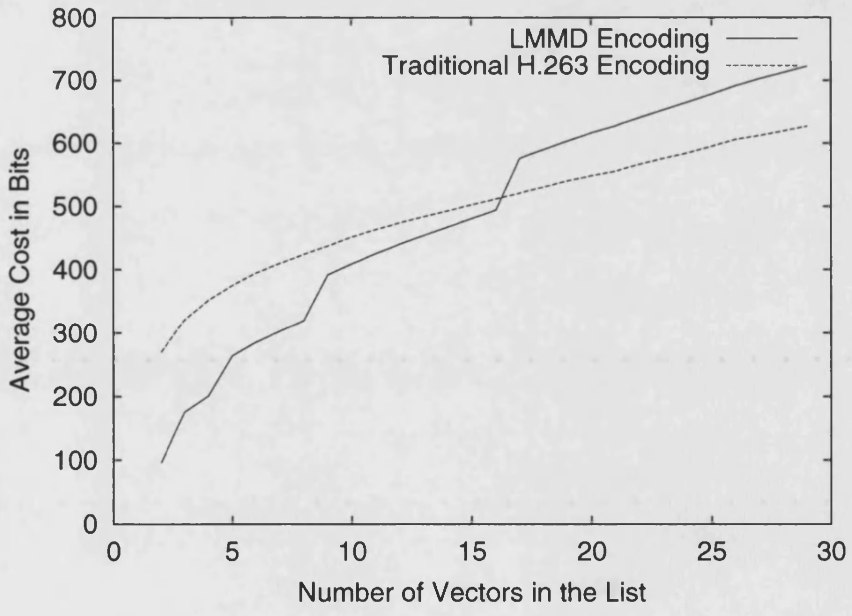


Figure 6.12: Results comparing the LMMD style encoding against H.263 style encoding for the carphone sequence.

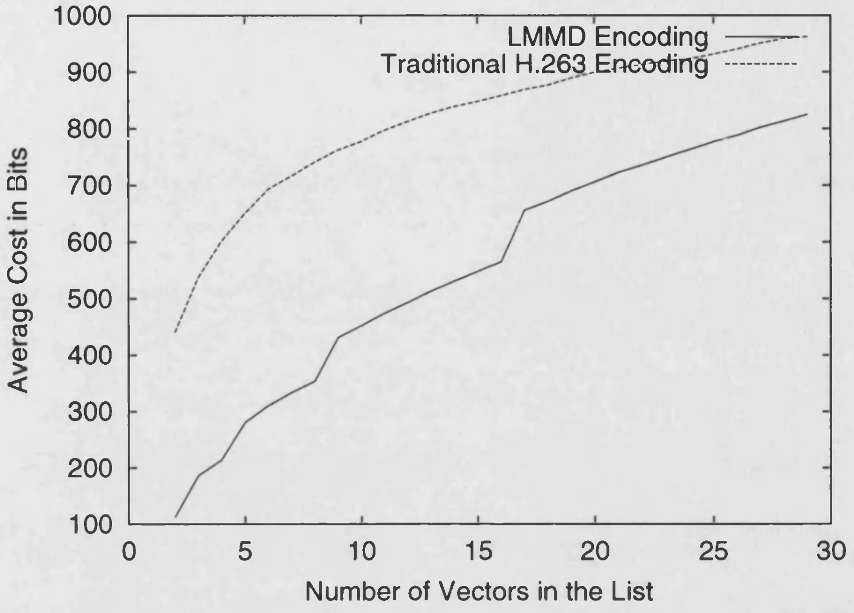


Figure 6.13: Results comparing the LMMD style encoding against H.263 style encoding for the football sequence.

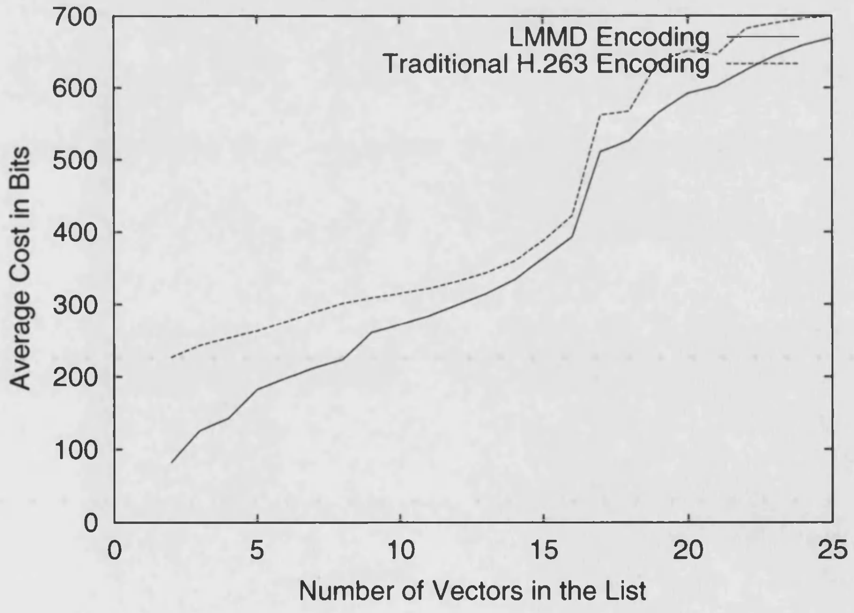


Figure 6.14: Results comparing the LMMD style encoding against H.263 style encoding for the news sequence.

stops selecting vectors when any addition vectors added to the list only cause one block to change its vector. The third test (Test 3) uses the 'Small Improvement' method to determine how many vectors should be included in the list, (see section 4.6.2). Vectors ceased to be added to the list when they caused the SAD of the residual to improve by less than 40. The final three tests 4–6 mirror test 1–3, except that the standard H.263 vector encoding is replaced with LMMD style encoding.

Each of these tests was carried out on the first 50 frames of the QCIF foreman sequence at 20 kbits/s, 32 kbits/s, 64 kbits/s and 128 kbits/s. The encoder was set up to encode the first I-frame with a quantisation parameter of 13, which provided a sensible image from which the subsequent frames could be predicted. After the initial I-frame all the rest of the frames were transmitted in INTER mode, so the resulting GOP was of type 'IPPP...'. The I-frame was not rate constrained it consumed more than one frames worth of bits, as a result the first encoded P-frame was frame 4 at 128 kbits/s, frame 8 at 64 kbits/s and 16 at 32 kbits/s. At 20 kbits/s the encoder was unable to encode single frames within the required bit budget. As a result it skipped every other frame, doubling the bit budget for each frame, but halving the number of frames encoded. For this reason these results have not been used.

The test were run at 30 fps, giving around 4300 bits/frame at 128 kbits/s, 2150 bits/frame at 64 kbits/s and 1075 bits/frame at 32 kbits/s. It was hoped that at these rates any saving made using the modifications to the standard H.263 encoder would be apparent.

ID	Description
0	Standard H.263
1	Metric Method: No dynamic picking
2	Metric Method: 'No single vector' dynamic picking
3	Metric Method: 'Small Improvement' dynamic picking
4	Metric Method with LMMD Encoding: No dynamic picking
5	Metric Method with LMMD Encoding: 'No single vector' dynamic picking
6	Metric Method with LMMD Encoding: 'Small Improvement' dynamic picking

Table 6.2: The seven tests. Test 0 is the baseline, and test 1–3 and 4–5 form pairs with and without LMMD encoding.

Figure 6.16 compared the baseline test 0 with tests 1–3. At 64 kbits/s none of the modifications appear to offer any improvement over the baseline. However, at 128 kbits/s all of the tests give an overall performance gain in terms of average

PSNR.

Figure 6.17 shows the results of test 4–6 which use the LMMD style encoding. At both bit rates there is now a marked improvement in terms of PSNR.

The final set of results, shown in figures 6.18, 6.19 and 6.20 compare similarly produced fields, encoded using standard H.263 encoding and LMMD style encoding. At 128 kbits/s the LMMD encoding provides very little overall benefit, but at 64 kbits/s it is the LMMD encoding which appears to be generating much of the improvement. The same is also true of the results at 32 kbits/s.

Finally, figure 6.21 shows the original H.263 encoding (test 0) versus test 6 for 32 kbits/s. In nearly all cases the proposed encoding technique improves upon that given by an unmodified H.263 encoder. These improvements are often in the region of 0.4 dB and in those cases where it is worse the difference is negligible.

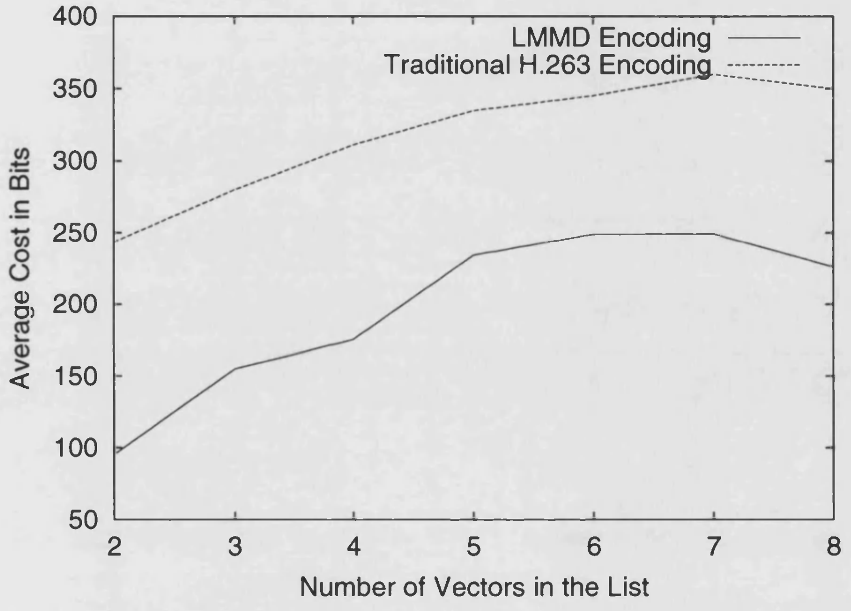
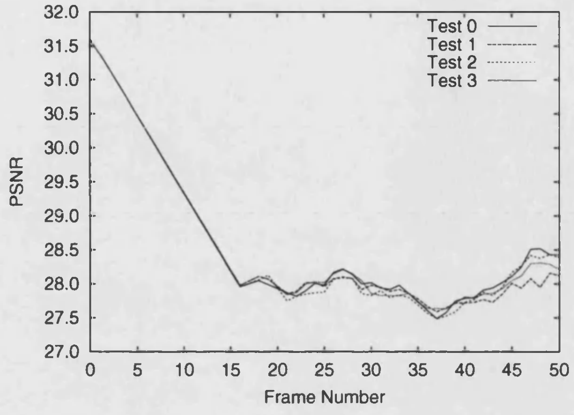
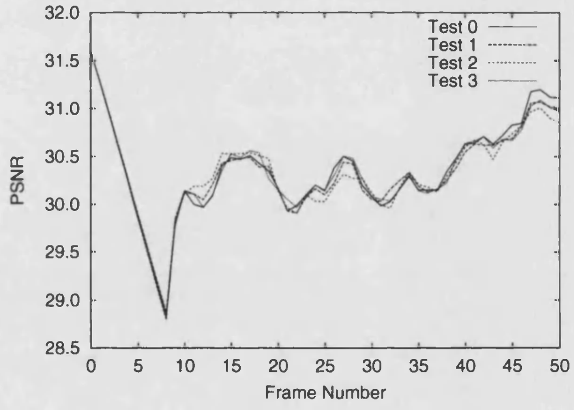


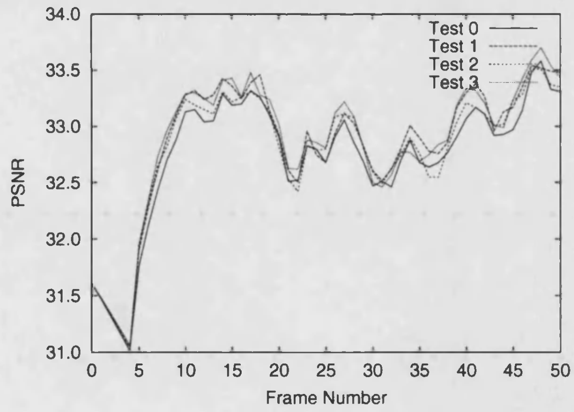
Figure 6.15: Results comparing the LMMD style encoding against H.263 style encoding for the container sequence.



32 kbits/s

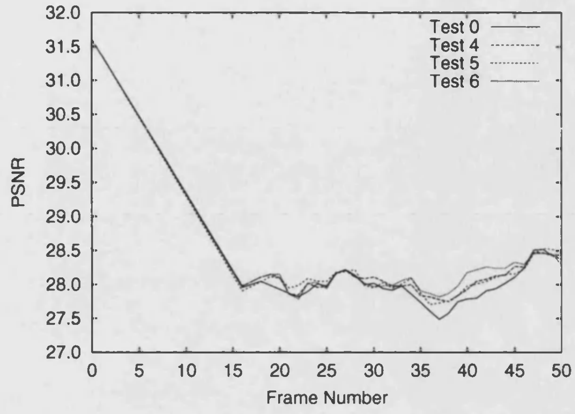


64 kbits/s

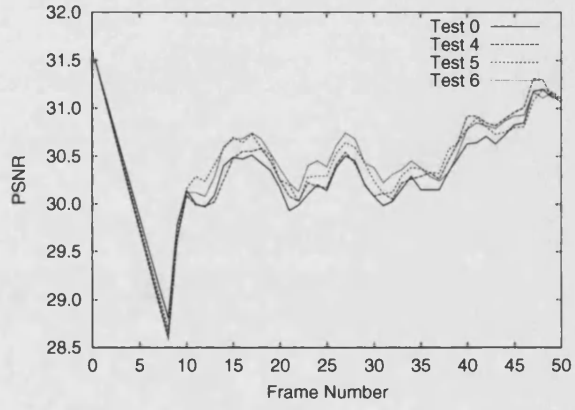


128 kbits/s

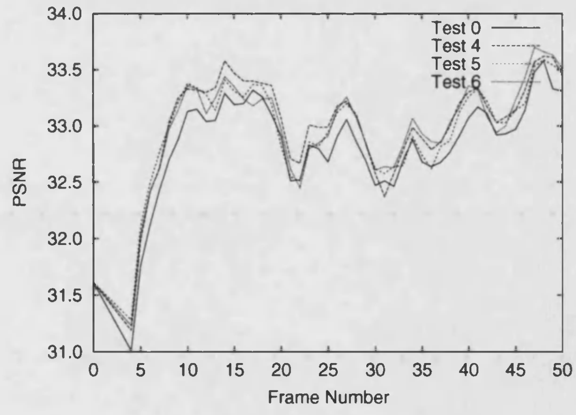
Figure 6.16: Tests 0, 1, 2, and 3 carried out on the first 50 frames of the foreman sequence.



32 kbits/s



64 kbits/s



128 kbits/s

Figure 6.17: Tests 0, 4, 5, and 6 carried out on the first 50 frames of the foreman sequence.

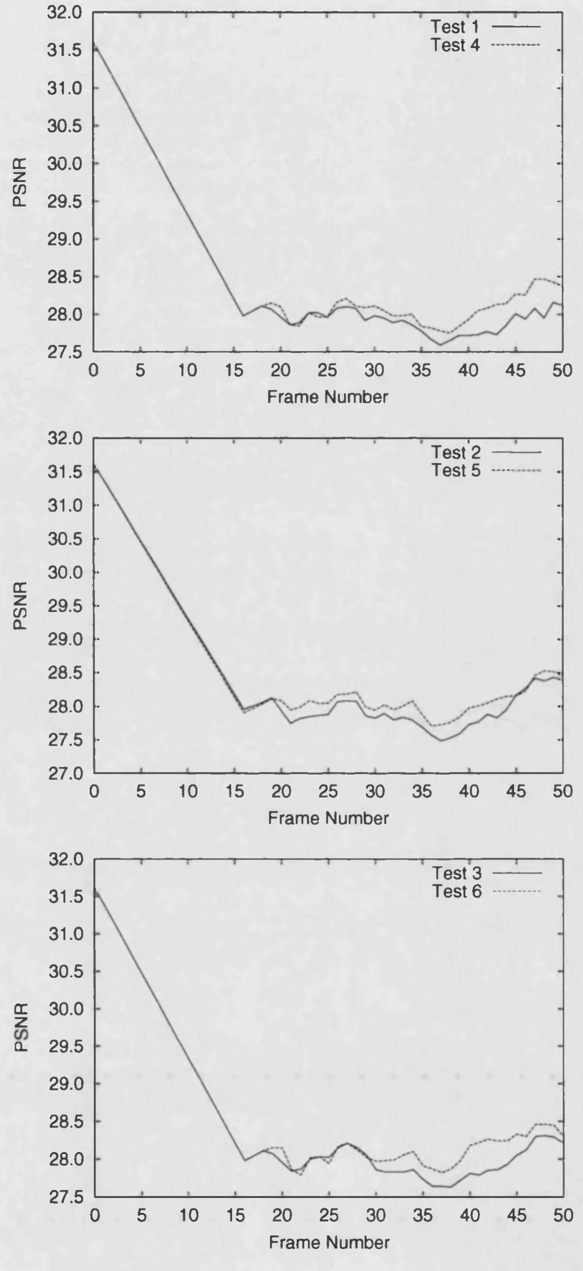


Figure 6.18: These graphs show the effect of adding LMMD encoding to fields produced by the Metric Method for the 32 kbits/s results.

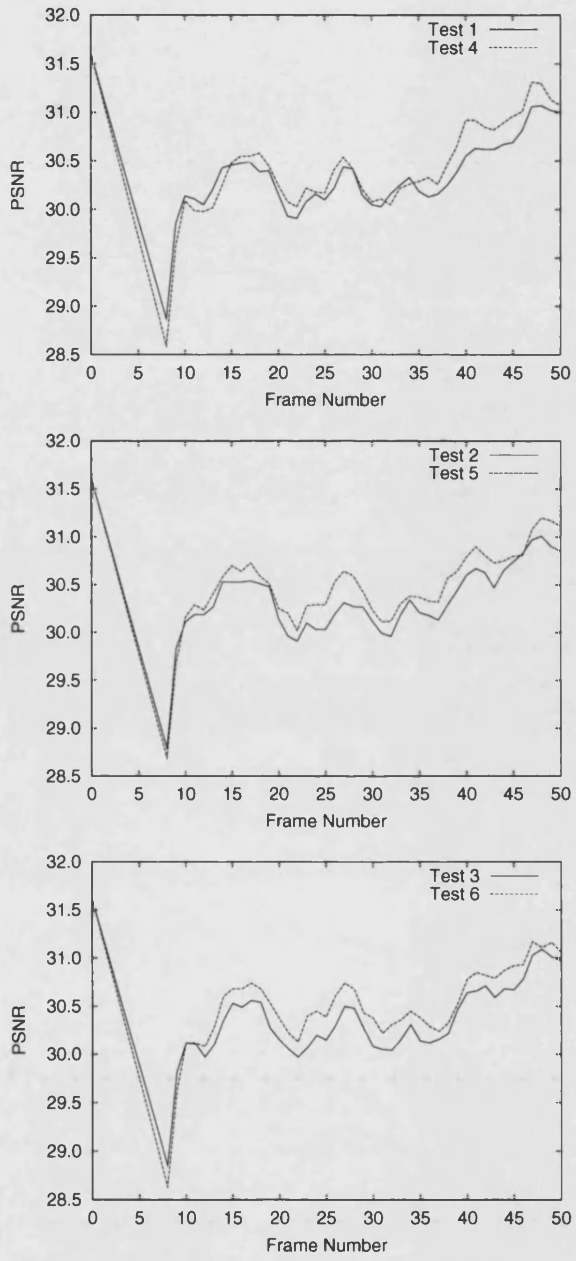


Figure 6.19: These graphs show the effect of adding LMMD encoding to fields produced by the Metric Method for the 64 kbits/s results.

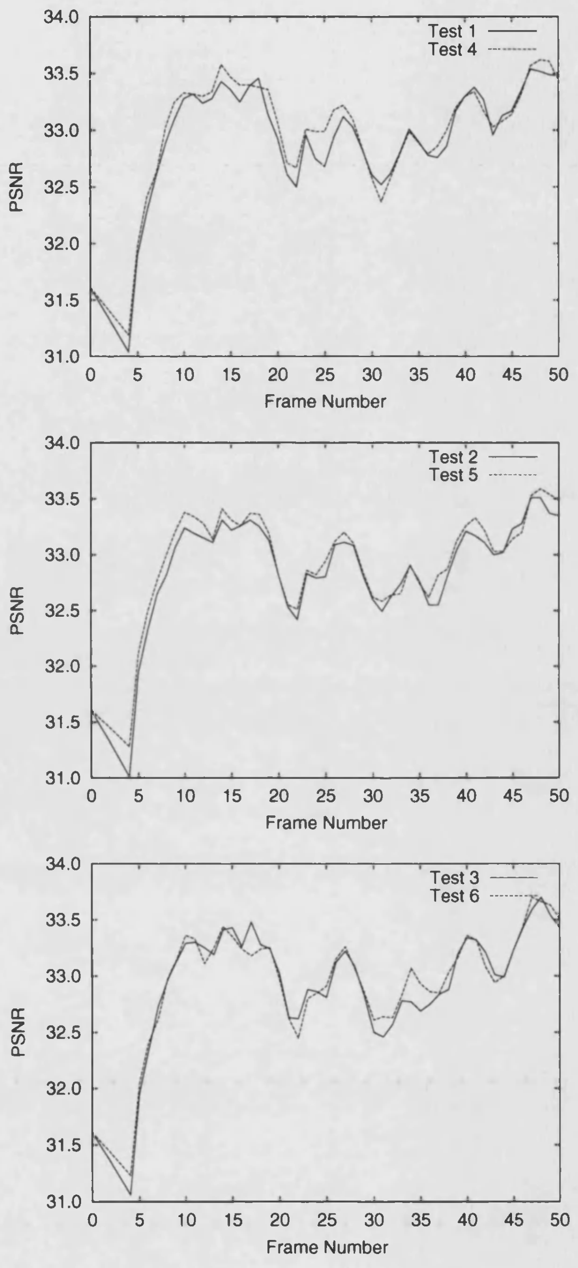


Figure 6.20: These graphs show the effect of adding LMMD encoding to fields produced by the Metric Method for the 128 kbits/s results.

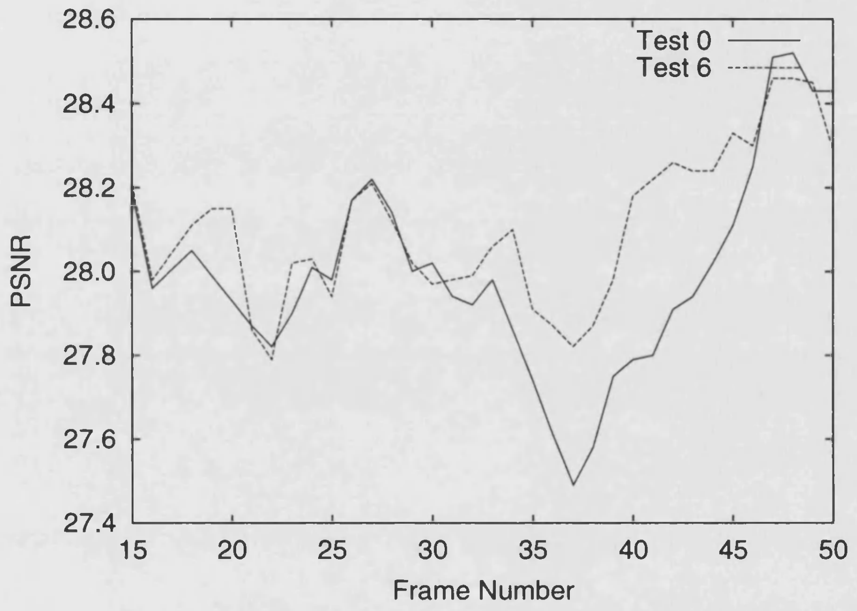


Figure 6.21: Comparing test 6 with unmodified H.263 encoding (test 0) at 32 kbits/s.

Chapter 7

Conclusions

7.1 Main Aims

This thesis has investigated the trade off between the bits allocated to the motion description and the residual in the context of a video codec.

7.2 Varying Block Sizes

Initial efforts showed that very coarse control could be achieved by varying the size of the blocks used in block matching algorithms. The level of control was not suitable for use in a video codec though.

7.3 The Extended Block Algorithm

Producing a smoother field by extending the blocks used in the block matching algorithms provided a mechanism for reducing the cost of encoding the motion description. This technique was more useful for dense motion fields with small block sizes. As a result it was not really applicable at low bit rates where there were insufficient bits available to encode such a dense field.

7.4 Embedded Quadtree Motion Estimation

The Embedded Quad-tree Motion Estimation (EQME) allowed the motion vector field bandwidth to be controlled at quite a fine level. It managed this by producing an embedded motion description which could be truncated with a resolution of a few bits. Experiments with the EQME confirmed that there was an optimum trade off in bits between the motion description and the residual, and that it should be possible to achieve this balance. Although the EQME was scalable it did not prove to be a very efficient way of describing the motion description.

The preceding experiments did indicate that the fewer unique vectors in the MVF the more easily the field compressed. This lead to the hypothesis that having control over the number of unique vectors in the field may give control over the cost of encoding that field.

7.5 Vector Selection Strategies

The investigation then moved to three vector selection strategies. The purpose of these was to select a specific number of vectors which could be used to construct a motion vector field. By varying the number of unique vectors in the field good control could be gained over the cost of the motion description. Of the three methods proposed the Metric Method provided the best trade off in terms of quality of result versus computational complexity.

When the Metric Method was used to modify the motion field produced by a standard H.263 codec it was found that it could improve the quality of the results (in terms of PSNR).

7.6 The List Mapping Motion Description

As the previously explored vector selection strategies could produce motion vector fields with few vectors, a new motion description was developed in order to further exploit the enhanced correlation in such a field. The LMMD described the motion vector field as a vector list, and a mapping indicating which listed vector belonged to which image block. Techniques for encoding both the list and the mapping were explored.

Experimental results showed that the LMMD was more efficient than a traditional H.263 encoder at representing fields with less than 16 unique vectors. When combined with a vector field produced by the Metric Method the quality of low bit rate video (32 and 64 kbits/s) could be improved. However, the addition of the LMMD didn't greatly improve of simple application of the Metric Method at higher bit rates (128 kbits/s).

7.7 Future Work

A great deal of time was spent looking at ways to optimise the vector list in the LMMD, and the results were very satisfactory. Less time was devoted to trying to improve the mapping encoding. Owing to the fact that the mapping often requires over twice as many bits to encode as the list it seems there is a great deal of room for improvement in this area. The investigations done so far have been aimed at the lower bit rates, and thus concentrated on smaller image sizes. Increasing the image size to say CIF (352×288) would increase the number of (16×16) image blocks in each frame. This would lead to a mapping which may well be more correlated than those produced for QCIF images.

The work on the EBA and the EQME was produced earlier during the PhD and as a result the approach is slightly less mature. It would be nice to take another look at these techniques in the light of the later work. The dense motion fields produced by the EBA could well be useful at higher bit rates. If the efficiency of compression offered by the EQME could be improved it could play the part in a fully embedded video compression system.

Appendix A

The H.263 Codec

The H.263 codec [1] was used as the reference codec against which the modifications described in this work were tested. H.263 is one of the two most prolific codecs in use today, the other being the Motion Picture Expert Group (MPEG) family of codecs. However, MPEG[8, 71] encoding is usually used for high bit-rate applications whereas H.263 performs better in low bit-rate scenarios; such as those of interest in this thesis.

A.1 Overview

The H.263 video codec allows for frames to be encoded in any one of a number of different *modes*, such as INTRA mode (I-frames), INTER mode (P-frames) and B mode (B-frames). However, the codec used for this thesis only implements INTRA and INTER modes as these were the one most relevant to the work. Actually, INTRA mode is only used for the first transmitted frame, from then on all frames are transmitted in INTER mode. (Don't forget however, that INTER frames can have INTRA blocks.)

H.263 is a block based codec. This means that the image is broken into blocks and each block is encoded (more or less) independently. This is a rough overview of how I-blocks are encoded. See the specifications[1] for the full details. The 16×16 block is first split into four 8×8 blocks. Each of these is then transformed using the DCT. The transform coefficients for each block are quantised using a given quantisation parameter. The quantised coefficients are then linearised using a zig-zag scan pattern and this linear sequence then encoded using run/level based Variable Length Code (VLC).

The process for encoding P-blocks is very similar, except instead of encoding the actual image pixels themselves the difference between these pixels and some prediction is encoded. The prediction is described using an offset which points to a previously (temporally) transmitted block. This offset is known as the motion vector for the block. Each block has its own motion vector. In the case of the P-blocks the motion vector needs to be transmitted along with the transform coefficients.

Rather than encoding the actual vector for each P-block, the H.263 codec encodes the difference between the vector for that block and a predicted vector. This predicted vector is formed using three neighbouring blocks; the block to the left, the block above and the block diagonally above and right. This is shown in figure A.1.

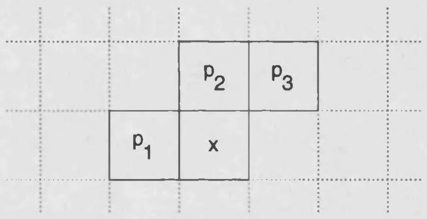


Figure A.1: The three neighbouring blocks used to generate the predicted vector.

In some cases the vector for a block is not known, for example if the block was not encoded, or it lies outside the frame. In these cases the following rules are used to determine the value of the vector.

1. If the block was not coded, or it was coded in INTRA mode the motion vector is taken to be $(0,0)$.
2. If the left hand block lies outside the image, it is taken to have a vector $(0,0)$.
3. If the upper blocks lie outside the top of the image they are assigned the same vector as the left hand block.
4. If the right hand block lies outside the right hand edge of the image it is assigned the vector $(0,0)$.

These rules are summarised in figure A.2.

Once the vectors of the three candidate blocks have been established the predicted vector is formed by taking the median of each of the vector components

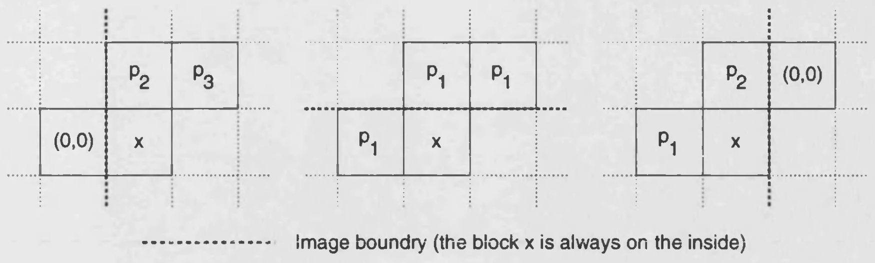


Figure A.2: Rules for obtaining the candidate predicted vectors.

from these blocks. For example if the three candidate vectors were $(2,0)$, $(2,4)$ and $(1,1)$ the predicted vector would be;

$$\begin{aligned} v &= (\text{median}(2,2,1), \text{median}(0,4,1)) \\ &= (2,1) \end{aligned}$$

In this thesis all vector components differences were encoded using a RVLC as shown in table A.1.

Value Range	Value to be encoded	Number of bits	Code
0	0	1	1
1	1	3	$0s0$
2–3	x_0+2	5	$0x_01s0$
4–7	x_1x_0+4	7	$0x_11x_01s0$
8–15	$x_2x_1x_0+8$	9	$0x_21x_11x_01s0$
⋮	⋮	⋮	⋮

Table A.1: RVLC scheme used to encode motion vectors. Each x_n represents a binary digit and the value of s is used to represent the sign of the value, 0 for positive and 1 for negative. Taken from [1].

Rate control for the transform coefficients can be achieved by varying the quantisation parameter. Increasing the parameter causes more information to be lost, but produces transform coefficients which can be encoded more compactly. Conversely, higher quality can be achieved by reducing the quantisation parameter but this results in more bits being needed to encode the transform coefficients.

Appendix B

Fields for Varying Blocksizes

Here we show the four different vector fields produced by the experiment described in section 2.14. The fields are produced using the ESA (search range 16×16) on the pair of QCIF sized images shown in figure 3.1. Each field represents a different block size: 2×2, 4×4, 8×8 and 16×16.

It is interesting to see that in figure B.1 (block size of 2×2) the field is very chaotic and disorganised. This is due to the fact that each 2×2 block does not really contain enough context to enable the best match to reflect the true motion. Instead each group of four pixels is matched with the most similar group within the search range, leading to a wild variation in motion vectors. The resulting field is less correlated, and therefore compresses less well when compared with the other fields, (as seen in the increase in the 'bit cost per block' result).

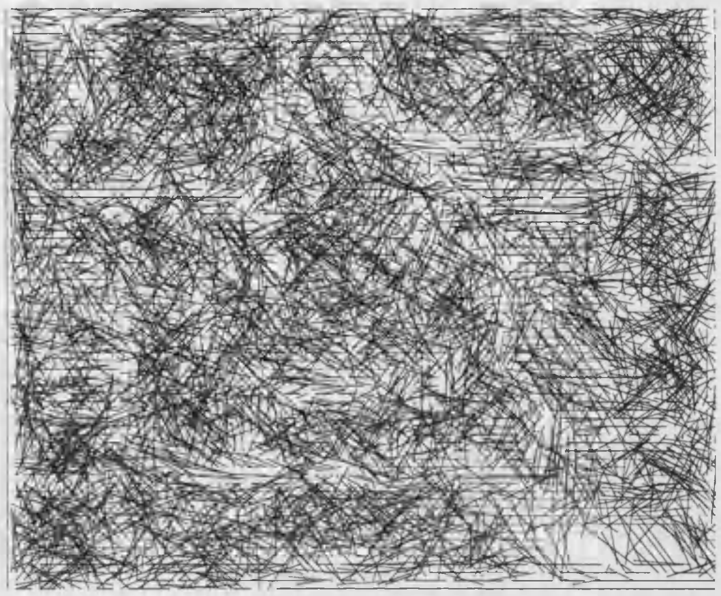


Figure B.1: Motion vector field produced with a block size of 2×2.

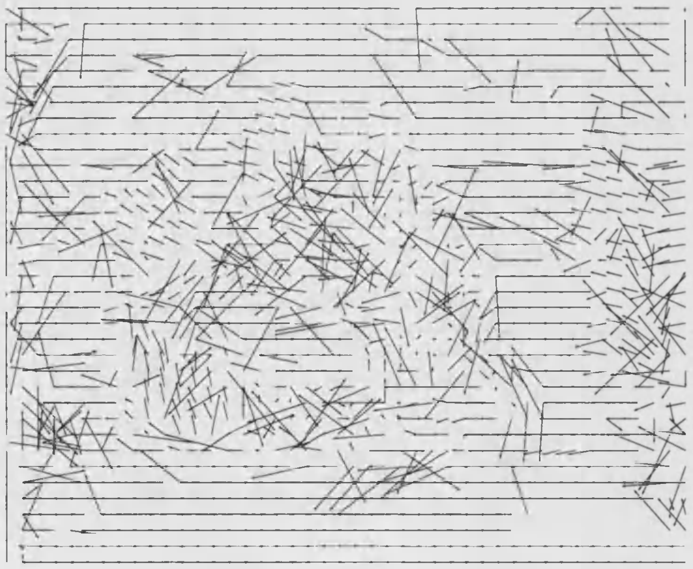


Figure B.2: Motion vector field produced with a block size of 4×4.

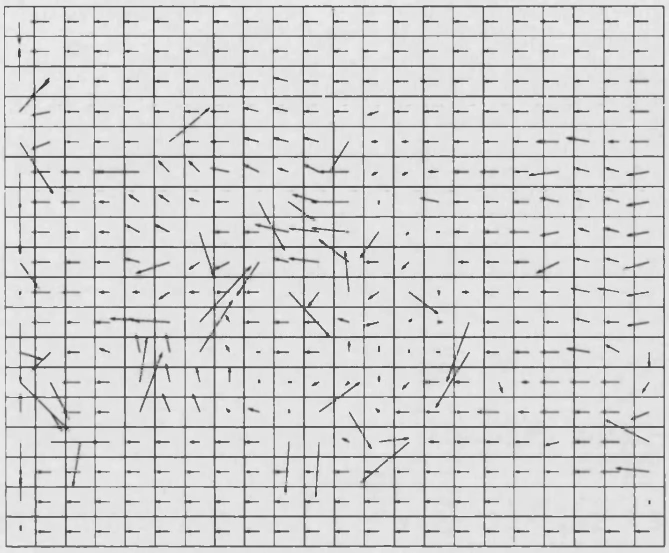


Figure B.3: Motion vector field produced with a block size of 8 × 8.

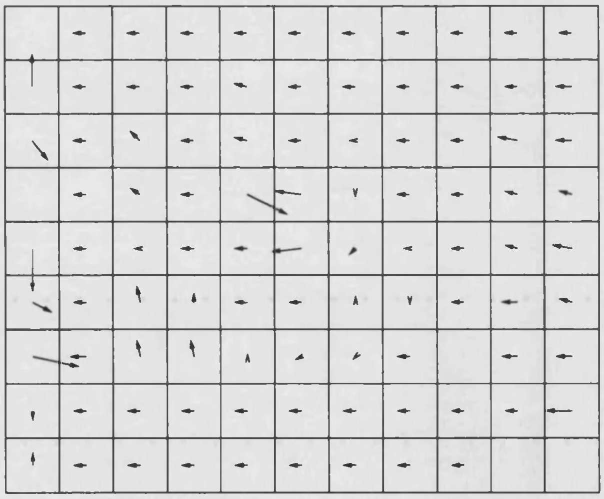


Figure B.4: Motion vector field produced with a block size of 16 × 16.

Appendix C

The Travelling Salesman Problem

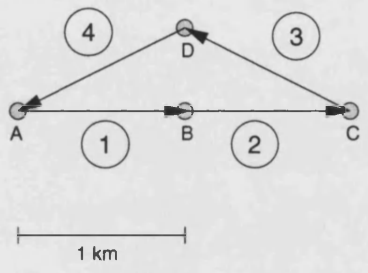
In the Travelling Salesman Problem (TSP) our salesman has to visit a number of different cities which are a known distance apart. He would like to know which order to visit the cities in so as to minimise his distance travelled.

A route between the cities may be *open* or *closed.* A open route visits each city exactly once and is dependant on which city is chosen as the start point. A closed route requires the salesman to eventually arrive back at the starting city. In this case the starting position is immaterial.

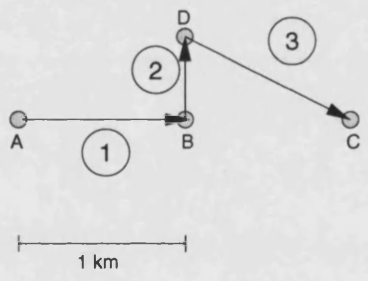
One might suspect that the best open route would form part of the best closed route. This is not the case and can be demonstrated by means of the simple counter example shown in figure C.1. Assume the salesman starts at city A, his best closed route takes him first to city B, followed by cities C and D. In the best open route however, he travels from city B to city D, a journey not present in the closed route. (It is interesting to note that if the best open route were started from city B or D, then the route would indeed be part of the best close route.)

The number of possible open routes between N cities is given by $N!$, and the number of closed routes by $(N-1)!$. In the case where journeys are symmetrical (the distance from A to B is the same as the distance from B to A) these numbers can be halved.

A brute force computational approach would simply evaluate each of these combinations to find the best. Even with todays powerful computers the sheer size of the problem limits the maximum number of cites which can be considered.



Journey	Distance
1	1.00 km
2	1.00 km
3	1.12 km
4	1.12 km
Total	4.24 km



Journey	Distance
1	1.00 km
2	0.50 km
3	1.12 km
Total	2.62 km

Figure C.1: Comparing optimum open and closed routes. Here it is clear that the best open route (starting at city A) is not part of the best closed route.

Appendix D

Data

D.1 Average Cost of Field vs. No. of Unique Vectors

NV	Cost	Count	NV	Cost	Count	NV	Cost	Count
1	115.7	263	25	422.2	31	49	736.0	1
2	116.9	346	26	447.0	23	50	748.8	8
3	117.2	262	27	458.3	29	51	799.5	4
4	124.1	174	28	493.6	22	52	790.0	4
5	131.2	180	29	482.2	24	53	792.0	3
6	137.8	136	30	491.6	33	54	1012.0	1
7	147.9	106	31	509.9	33	55	864.3	6
8	166.6	89	32	546.7	15	56	835.6	5
9	184.8	68	33	545.0	23	57	865.7	7
10	185.4	59	34	566.6	18	58	841.0	2
11	210.1	69	35	555.5	19	59	860.0	1
12	210.7	80	36	585.5	12	60	891.2	5
13	222.2	80	37	582.8	13	62	998.0	2
14	232.6	70	38	575.8	10	63	1002.0	1
15	249.8	71	39	644.2	12	65	898.0	1
16	271.8	47	40	650.2	12	66	1008.0	3
17	291.2	50	41	668.8	8	67	953.0	2
18	305.1	52	42	639.4	10	68	966.0	1
19	309.1	38	43	658.8	8	69	1018.0	4
20	339.3	52	44	706.6	7	70	1027.0	2
21	368.3	31	45	664.4	11	73	1074.0	1
22	397.7	33	46	650.3	7	74	1130.0	1
23	394.9	34	47	764.4	10	76	1180.0	1
24	412.5	31	48	703.3	6			

Table D.1: The average cost of a vector field with a given number of unique vectors (NV).

Appendix E

List of Acronyms

BMA Block Matching Algorithm

CRT Cathode Ray Tube

CZS Circular Zonal Search

DSA Diamond Search Algorithm

DCT Discrete Cosine Transform

DWT Discrete Wavelet Transform

EBA Extended Block Algorithm

EQME Embedded Quad-tree Motion Estimation

ESA Exhaustive Search Algorithm

FSS Four Step Search

HVS Human Visual System

JPEG Joint Photographic Expert Group

LMMD List Mapping Motion Description

MAD Mean of Absolute Differences

MPEG Motion Picture Expert Group

MSE Mean Square Error

NSS N-Step Search

RFSM	Restricted Full Search Method
PSNR	Peak Signal-to-Noise Ratio
RMSE	Root Mean Square Error
RVLC	Reversible Variable Length Coding
SAD	Sum of Absolute Difference
SEA	Successive Elimination Algorithm
TCM	Transform Coding Model
TSA	Travelling Salesman Algorithm
TSP	Travelling Salesman Problem
TSS	Three Step Search
VLC	Variable Length Code

Appendix F

Author's Publications

1. S. Tredwell and A. N. Evans, "Embedded quad-tree motion estimation for low bit rate video coding," in *IEE European Workshop on Distributed Imaging, Ref. 1999/109,* November 1999, pp. 3/1–5.

2. S. Tredwell and A.N. Evans, "Selecting motion vectors for bandwidth controllable motion description coding," in *Proceedings of Irish Machine Vision and Image Processing Conference*, September 2001.

3. S. Tredwell and A.N. Evans, "A sequential vector selection algorithm for controllable bandwidth motion description encoding," in *Proceedings of IEEE International Symposium on Intelligent Multimedia, Video and Speech Processing*, May 2001, pp. 209–212.

4. S. Tredwell and A.N. Evans, "Block grouping algorithm for motion description encoding," in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 2001, vol. 5, pp. 211–214.

5. S. Tredwell and A. N. Evans, "Improved motion description coding using the list mapping motion description," in *Proceedings ICIP2002, Rochester, USA*, September 2002, vol. 1, pp. 657–660.

References

[1] ITU, “Recommendation H.263 (02/98) – video coding for low bit rate communication,” ITU-T H.263, International Telecommunication Union, 1998.

[2] A. Manduca and A. Said, “Wavelet compression of medical images with set partitioning in hierarchical trees,” in *Proc. SPIE 2707*, 1996, pp. 192–200.

[3] R. C. Gonzales and R. E. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

[4] C. Poynton, “Colorfaq,” http://www.inforamp.net/ poynton/ColorFAQ.html.

[5] D. H. Brainard and B. A. Wandell, “Calibrated processing of image color,” *Color Research and Application*, vol. 15, pp. 266–271, 1990.

[6] G. Sharma and H. J. Trussell, “Digital color imaging,” *IEEE Transactions on Image Processing*, vol. 6, no. 7, pp. 901–932, 1997.

[7] G. Sharma, M. J. Vrhel, and H. J. Trussell, “Color imaging for multimedia,” *Proceedings of the IEEE*, vol. 86, no. 6, pp. 1088–1108, 1998.

[8] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*, Chapman & Hall, 1996.

[9] M. Nelson and J. Gailly, *The Data Compression Book*, M&T Books, New York, 2nd edition, 1996.

[10] G. C. K. Abhayaratne and D. M. Monro, “Embedded to lossless image coding (ELIC),” in *Proc. IEEE Nordic Signal Processing Symposium (NORSIG 2000)*, June 2000, pp. 255–258.

[11] N. Vasconcelos and F. Dufaux, “Pre and post-filtering for low bit-rate video coding,” in *ICIP (1)*, 1997, pp. 291–294.

[12] N. Young and A. N. Evans, "Psycho-visually tuned area-morphology tools for improved image compression," in *Proc. International Symposium on Mathematical Morphology , Manly, Sydney, Australia*, April 2002, pp. 185–195.

[13] G. K. Wallace, "JPEG: Still image compression standard," *Communications of the ACM*, vol. 30, no. 6, pp. 30–44, April 1991.

[14] N. Ahmed, T. Natarajan, and K. R. Pao, "Discrete cosine transform," *IEEE Transactions on Communication*, vol. COM-23, pp. 90–93, January 1974.

[15] D. Gabor, "Theory of communication," *J. Inst. Elect. Eng.*, vol. 93, pp. 429–457, 1946.

[16] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications in Pure and Applied Mathematics*, vol. 41, pp. 909–996, 1988.

[17] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674–693, 1989.

[18] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, "Wavelets for computer graphics: A primer, part 1," *IEEE Computer Graphics and Applications*, vol. 15, no. 3, pp. 76–84, 1995.

[19] A. Graps, "An introduction to wavelets," *IEEE Computational Science and Engineering*, vol. 2, no. 2, pp. 383- 390, 1995.

[20] T. Edwards, "Discrete wavelet transforms: Theory and implementation," 1991.

[21] C. E. Heil and D. F. Walnut, "Continuous and discrete wavelet transforms," *SIAM Review*, vol. 31, no. 4, pp. 628–666, 1989.

[22] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG-2000," in *Data Compression Conference*, 2000, pp. 523–544.

[23] Z. Xiong, K. Ramchandran, and M. Orchard, "Joint optimization of scalar and tree-structured quantization of wavelet image decompositions," in *27th Annual Asilomar Conf. on Signal, Syst. and Computers*, November 1993, pp. 891–895.

[24] D. Yu and M. W. Marcellin, “A fixed-rate quantizer using block-based entropy-constrained quantization and run-length coding,” in *Designs, Codes and Cryptography*, 1997, pp. 310–316.

[25] J. M. Shapiro, “Embedded image coding using zerotree of wavelet coefficients,” *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, 1993.

[26] D. M. Monro and G. J. Dickson, “Zerotree coding of DCT coefficients,” in *Proc. IEEE ICIP 1997*, 1997, vol. 2, pp. 625–628.

[27] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[28] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, pp. 623–656, October 1948.

[29] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” in *Proceedings of the IRE*, September 1952, vol. 40, pp. 1098–1101.

[30] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” in *Communications of the ACM*, February 1987, vol. 30, pp. 520–540.

[31] G. G. Langdon, “An introduction to arithmetic coding,” *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, 1984.

[32] P. G. Howard and J. S. Vitter, “Analysis of arithmetic coding for data compression,” *Information Processing and Management*, vol. 28, no. 6, pp. 749–764, 1992.

[33] İ. Avcıbaş, B. Sankur, and K. Sayood, “Statistical evaluation of image quality measures,” *Journal of Electronic Imaging*, vol. 11, no. 2, pp. 206–223, April 2002.

[34] J. L. Barron, D. J. Fleet, and S. Beauchemin, “Performance of optical flow techniques,” *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[35] B. K. P. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, pp. 185–204, 1981.

[36] E. H. Adelson and J. R. Bergen, "The extraction of spatio-temporal energy in human and machine vision," in *Proceedings of Workshop on Motion: Representation and Analysis*, May 1986, pp. 151–155.

[37] G. A. Thomas, "Television motion measurement for DATV and other applications," Tech. Rep. BBC RD 1987/11, BBC, September 1987.

[38] John Watkinson, *The Engineer's Guide to Motion Compensation*, Snell and Wilcox, 1994.

[39] T. Wiegand, X. Zhang, and B. Girod, "Long-term memory motion-compensated prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 70–84, February 1999.

[40] J. Jain and A. Jain, "Displacement measurement and its application in interframe video coding," *IEEE Transactions on Communication*, vol. COM-29, no. 12, pp. 1799–1808, December 1981.

[41] F. Moschetti and E. Debes, "Data adapting motion estimation and subsambling," in *Proceedings of the SPIE International Conference on Visual Communications and Image Processing*, June 2000.

[42] B. Lui and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 2, pp. 148–157, April 1993.

[43] Y. Chan and W. Siu, "New adaptive pixel decimation for block motion vector estimation," *IEEE Transactions on Circuits and Systems on Video Technology*, vol. 6, no. 1, pp. 113–118, February 1996.

[44] M. Bierling, "Displacement estimation by hierarchical blockmatching," *Visual Communications and Image Processing*, pp. 942–951, May 1998.

[45] K. M. Nam, J. Kim, R. Park, and Y. S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Transactions on Circuits and Systems on Video Technology*, vol. 5, no. 4, pp. 344–351, August 1995.

[46] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, August 1994.

[47] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, June 1996.

[48] K. K. Ma and S. Zhu, "A new diamond search algorithm for fast block matching motion estimation," in *Proceedings of the International Conference on Information, Communications and Signal Processing*, 1997, vol. 1, pp. 292–296.

[49] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-based diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits, Systems and Video Technologg*, vol. 8, pp. 369–377, August 1998.

[50] A. Tourapis, G. Shen, M. Liou, O. Au, and I. Ahmad, "A new predictive diamond search algorithm for block based motion estimation," in *Proc. of Visual Comm. and Image Proc.*, June 2000.

[51] A. M. Tourapis, O. C. Au, and M. L. Liou, "Fast motion estimation using circular zonal search," in *Proceedings of Visual Communications and Image Processing 1999 (VCIP'99)*. SPIE, 1999.

[52] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, no. 1, pp. 105–107, 1995.

[53] A. Kaup and H. Mooshofer, "Performance and complexity analysis of rate constrained motion estimation in MPEG–4," in *Proceeding of Multimedia Systems and Applications II*, Septemper 1999, pp. 202–211.

[54] G. M. Schuster and A. K. Katsaggelos, "An optimal quadtree-based motion estimation and motion-compensated interpolation scheme for video compression," *IEEE Transactions on Image Processing*, vol. 7, no. 11, pp. 1505–1523, November 1998.

[55] C. Cafforio and F. Rocca, "Methods for measuring small displacements of television images," *IEEE Transactions on Information Theory*, vol. IT-22, no. 5, pp. 573–579, September 1976.

[56] S. Tredwell and A. N. Evans, "Embedded quad-tree motion estimation for low bit rate video coding," in *IEE European Workshop on Distributed Imaging, Ref. 1999/109*, November 1999, pp. 3/1–5.

[57] A. N. Evans, "On the use of ordinal measures for cloud tracking," *International Journal of Remote Sensing*, vol. 21, no. 9, pp. 1939–1944, June 2000.

[58] J. L. Prince and E. R. McVeigh, "Motion estimation from tagged MR image sequences," *IEEE Transactions on Medical Imaging*, vol. 11, no. 2, pp. 443–447, June 1992.

[59] G. M. Schuster and A. K. Katsaggelos, "Optimal decomposition for quad-trees with leaf dependencies," in *Proceedings of SPIE - the International Society for Optical Engineering*, 1997, pp. 59–70.

[60] G. M. Schuster and A. K. Katsaggelos, "An optimal quad-tree-based motion estimation and motion compensated interpolation scheme for video compression," *IEEE Transactions on Image Processing*, vol. 11, no. 11, pp. 1505 – 1523, November 1998.

[61] S. Tredwell and A.N. Evans, "Selecting motion vectors for bandwidth controllable motion description coding," in *Proceedings of Irish Machine Vision and Image Processing Conference*, September 2001.

[62] S. Tredwell and A.N. Evans, "A sequential vector selection algorithm for controllable bandwidth motion description encoding," in *Proceedings of IEEE International Symposium on Intelligent Multimedia, Video and Speech Processing*, May 2001, pp. 209–212.

[63] S. Tredwell and A.N. Evans, "Block grouping algorithm for motion description encoding," in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 2001, vol. 5, pp. 211–214.

[64] S. Tredwell and A. N. Evans, "Improved motion description coding using the list mapping motion description," in *Proceedings ICIP2002, Rochester, USA*, September 2002, vol. 1, pp. 657–660.

[65] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[66] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *Complex Systems*, vol. 5, pp. 299–326, 1991.

[67] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Articial Intelligence Review*, vol. 13, pp. 129–170, 1999.

[68] B. Freisleben and P. Merz, "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in *International Conference on Evolutionary Computation*, 1996, pp. 616–621.

[69] B. Freisleben and P. Merz, "New genetic local search operators for the traveling salesman problem," in *Proceedings of theFourthConference on Parallel Problem Solving from Nature(PPSN IV)*, Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, Eds., Berlin, 1996, vol. 1141, pp. 890–899, Springer.

[70] Merz and Freisleben, "Genetic local search for the TSP: New results," in *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1997.

[71] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46– 58, April 1991.