Chapter 6

Zooming of Compressed Video

Problem of zooming can be looked upon as falling in to three classes: spatial for still images, temporal to interpolate missing frames in a video sequence and motion zooming in which a given video is zoomed to get a high resolution video. Zooming of still images is well addressed in the literature and is discussed in Chapter 4. Temporal zooming is also well addressed (for example see [63, 113, 116] and references therein). Motion zooming in non-compressed domain are treated by Schultz and Stevenson [121] Kaulgud and Desai [71] and compiled by Katsaggelos and Galatsanos [67]. Of late, motion zooming in compressed domain is drawing attention of researchers. Seagall and Katsaggelos [122, 123] propose a visual quality measurement constraint for enhancement of compressed video. Mateos *et. al.* [61] propose a scheme for simultaneous motion estimation and resolution enhancement.

In this chapter, we propose to zoom a video in compressed domain. We interpolate the motion vectors to locate a block in the zoomed video frame. We consider the motion vectors obtained by DCT and DWT based methods and see that the performance is similar in both the cases. This shows that the proposed method is relatively independent of compression methods.

This chapter is organized as follows: Section 1 gives a general introduction and addresses the need for video compression. In Section 2, we briefly discuss video compression methodologies, emphasizing the block-matching motion algorithm Its extension to multiresolution techniques appear in Section 3. Sections 4 outline our proposed method to interpolate motion vectors to zoom a video in compressed domain. Section 5 extends it to video stream compressed using MRME technique. Section 6 extends section 4 for temporal interpolation (estimating missing video frames). Section 7 and 8 provide discussion and summary, respectively.

6.1 Introduction

Due to limitations in the channel capacity, one might be forced to transmit a low resolution version of a given video. Then at the receiver, we need to extract a high resolution version of the transmitted video. Moreover, a raw video is not transmitted directly, but, a compressed version is transmitted instead. For video compression, motion information from the adjacent frames, which exploits temporal redundancy, is coded and quantized. At the receiver, suppose a user wants to obtain a higher resolution video, he has to generate it from this low-resolution compressed version. Conventional methods to reconstruct the data and then apply simple interpolation techniques are not only computationally demanding, but may result in a poor video quality. On the other hand, if we can manipulate the data in the compressed domain, unnecessary coding-decoding can be avoided. Moreover, while doing so, we will have a scheme that is independent of the compression standard. Motion vectors are, relatively, independent of the compression standards. Thus, if we can interpolate motion vectors at the receiver, we can reconstruct a higher resolution video stream at the receiver.

We assume that zooming is dual of compression. So, let us take a look at a generic video coder (Fig.6.1). As can be seen, two frames, $I(x, y, t)$ - the current frame, and $I(x, y, t-1)$ - the frame of the previous instant are used for motion estimation, motion compensation and motion vector calculations. That is, motion information is coded during compression and decoded during decompression. If we want to manipulate video in the compressed domain, we have to manipulate the motion vectors. This is possible if we can *interpolate* motion vectors to achieve our task. Mandal [81], Kim[118] and Zhang[156], assume that motion vectors at different levels are highly correlated in DWT domain. Taking this assumption as a cue, we predict motion vectors for the finer resolution in the DWT domain. Thus, the block diagram for our zooming methodology is as depicted in Fig 6.2. According to Fig.6.2, Motion vectors are *interpolated* during the decompression phase (shaded block). Inverse transform of twice the length of original sequence will result in a zoomed video frame.

In this chapter, we begin with an overview of video compression methodology, emphasizing block-matching motion algorithms, and their extension to multiresolution techniques. We follow it with the basic method for motion vector interpolation, where we assume that the coding and decoding blocks of Fig. 6.2 are absent (shown in dotted lines). We then extend the motion vector interpolation to a multiresolution motion estimation scheme.

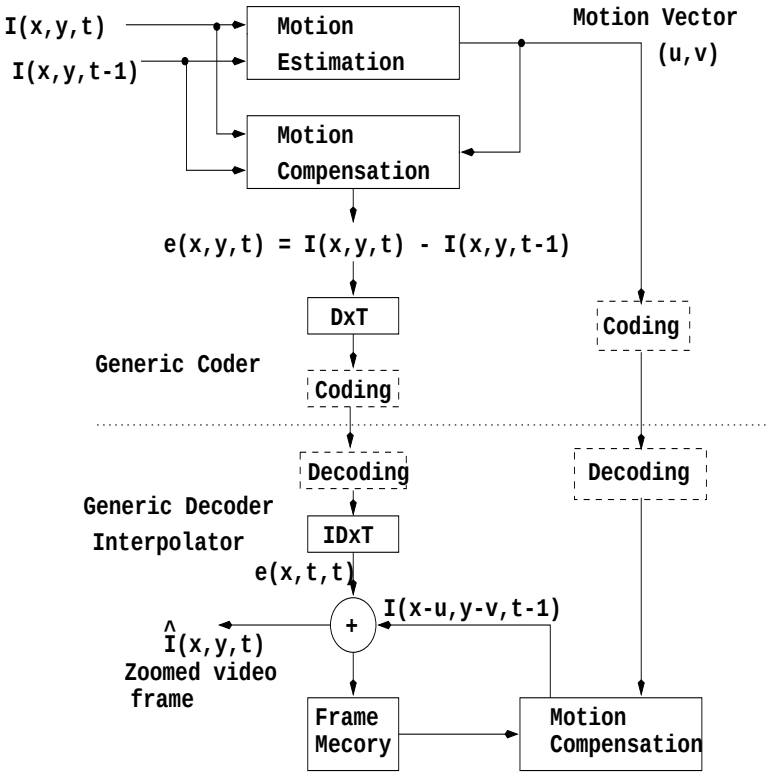


Figure 6.1: Generic hybrid Video coder and decoder

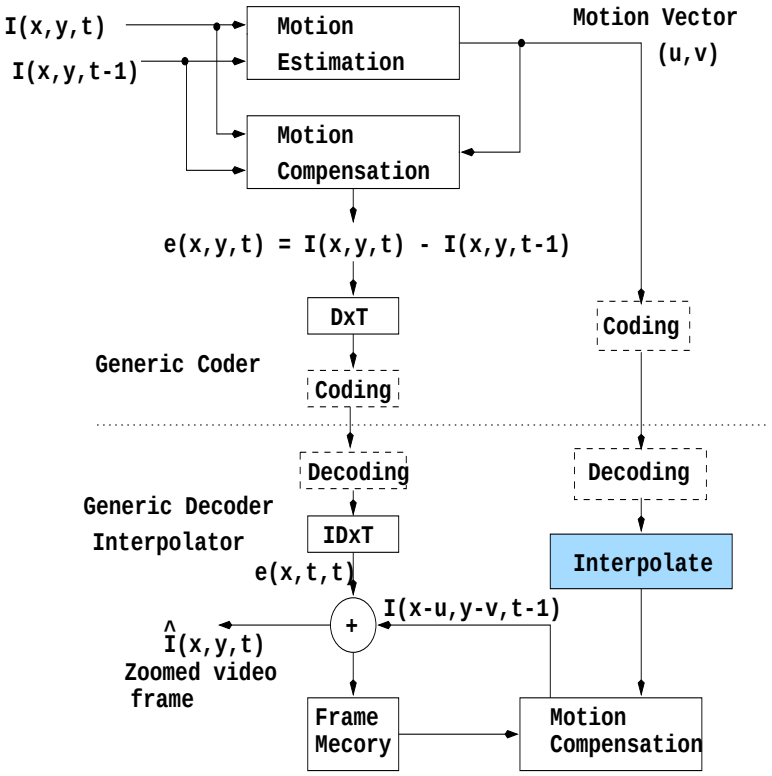


Figure 6.2: Generic hybrid video coder and decoder/interpolator. Motion vectors are interpolated in the shaded box during decoding.

6.2 Video Compression Techniques

Extraction of motion information is fundamental and essential for many image and video processing applications. A variety of motion estimation algorithms have been proposed [54, 125, 129, 142]. A good review of motion estimation is presented by Aggarwal and Nandhakumar [4] and Stiller and Konrad [133]. These algorithms are developed for different applications such as machine vision, robotics, image sequence analysis, coding and restoration. Motion estimation for video has some specific restrictions and requirements. Since the goal is to reduce the total transmission bit-rate, motion information should occupy a small amount of transmission bandwidth in addition to the picture information. To reduce computation and storage complexity, motion parameters of objects in a picture are estimated based on one or two neighboring frames. Most of the motion estimation algorithms assume rigid body, translational motion, occlusions etc. Motion estimation techniques can be divided into four main groups, namely,

- Gradient technique [48]
- Pel-recursive technique [147]

- Block-matching technique[57]
- Frequency domain technique [45, 47]

Gradient techniques are developed for image sequence analysis applications. They solve optical flow problems and result in a dense motion field. Both pel-recursive and block-matching techniques are developed in the framework of image sequence coding. Pel-recursive technique can be considered as a subset of the gradient technique. Here, every pixel is treated as the basic motion estimation unit, and the motion vector is computed over a small neighborhood surrounding the pixel. In the block matching technique, the image (or a video frame) is partitioned into non-overlapping blocks, assuming moving objects can be approximated reasonably well by regular shaped blocks. Then, a single displacement vector is estimated for the entire image block under the assumption that all the pixels in the block share the same displacement. This method is widely used because of its simplicity and small computational overhead. Frequency domain techniques are based on the relation between transformed coefficients of shifted images. Comparison of these techniques are given by Dufuax *et al* [26] and Hand [43].

6.2.1 Block Matching Motion Estimation

Block matching algorithms are based on matching the blocks between two images or video frames, such that disparity is minimized. For each block in the current time t, a corresponding block in the previous time $(t-1)$ is found within a search range Ω, minimizing a matching criterion. Basic principle of motion estimation and compensation is to find a vector $w = (w_x, w_y)$, where, w_x and w_y are translations in horizontal and vertical directions, respectively. The parameter w reconstructs the block $B_t(s)$ at time t and centered at pixel location $s = (s_x, s_y)$ from $B_{t-1}(s + v)$ with minimum errors. Procedure is shown in Fig.6.3

Parameters involved in the matching process are:

- matching function,
- block size,
- size of the search window Ω,
- search order of selected block and
- number of candidate blocks, (search points)

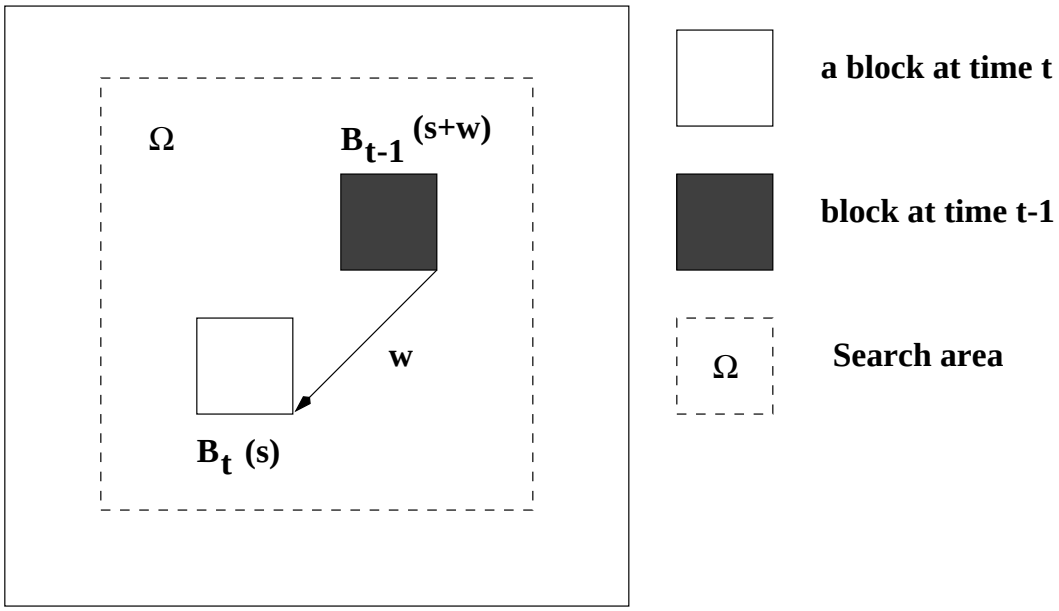


Figure 6.3: Block matching motion estimation.

Selection of a matching function has a direct impact on computational and motion vector accuracy. Frequently used matching function criteria are:

1. Maximum cross correlation (MCC):

$$w = \overset{arg\ max}{w \in \Omega} \sum_{s \in S} B_t(s) * B_{t-1}(s+w)$$

2. Minimum square error(MSE):

$$w = \overset{arg\ max}{w \in \Omega} \sum_{s \in S} (B_t(s) - B_{t-1}(s+w))^2$$

3. Minimum absolute difference (MAD):

$$w = \overset{arg\ max}{w \in \Omega} \sum_{s \in S} |B_t(s) - B_{t-1}(s+w)|$$

where S is the block size. Best match can be found only by performing a full search.

In order to decrease the computational load of full search algorithm, fast search techniques reported in literature [80, 132]. However, in these methods, convergence towards a global minimum is not guaranteed. The size of the block affects the performance of motion estimation. A small block size provides a good approximation to natural object boundaries. But it increases the number of transmission bits and is susceptible to noise.

Large block sizes may produce less accurate motion vectors. Block sizes of 8×8 and 16×16 are generally considered adequate for teleconferencing and entertainment applications. Another factor that governs the performance of a motion estimation scheme is the size of the search window Ω. A small search area can easily track the motion of slow moving objects (like video conferencing applications), while a large search area can track objects moving at high speeds. Larger the search area, more the search time and associated overhead.

The algorithms described above deal with video at a single resolution scale. In order to reduce computational complexity and to account for multi-scale characteristics of the motion scene, hierarchical and multi-grid algorithms have been proposed [26]. In a standard block matching technique, the motion is restricted to translation. However, block matching based motion estimation algorithms that relax this constraint have also been investigated [33, 124].

6.2.2 DWT based Techniques

Conventional block-matching motion estimation techniques use the DCT to match blocks in the transform domain. Of late, DWT is gaining popularity for video coding. With recent MPEG standards proposing use of wavelets, it is expected to gain wider acceptability. DWT for image and video coding entails a higher degree of computational complexity depending on the number of parameters required to characterize a wavelet. This implies more CPU cycles and/or more memory requirements for the use of DWT. Normally, motion compensation is done in the spatial domain on 8×8 or 16×16 blocks, which necessarily means reconstruction (inverse transform) of the previously encoded reference frame for motion estimation. Thus, if we were to follow a similar strategy using DWT, this would require the inverse DWT (IDWT) of the reference frame for motion estimation in the spatial domain. To avoid computing the IDWT, Zhang and Zafar [156] proposed a multiresolution motion estimation (MRME), which estimates motion in the wavelet transform domain. This technique estimates the motion vector hierarchically from lower to higher resolution sub-images. This approach exploits the cross-correlation between each layer of the wavelet pyramid in order to reduce the computational complexity of the motion estimation process. In the MRME scheme, the motion vectors for the detail sub-image of the coarsest resolution are determined (level 3) using a conventional block-matching based motion estimation algorithm. These motion vectors are used as the initial bases for calculating the motion vectors at the next finer resolution. Therefore, errors propagate

and expand to all subsequent finer resolution sub-images. This influences prediction at all levels to reduce errors.

Other techniques for MRME are also reported in the literature. Zafar and Zhang [155] propose a pel-recursive algorithm in a multiresolution form. Mandal *et al.* [81] propose three techniques to improve motion estimation in a wavelet based coder. First method is an adaptive threshold MRME (AMRME). This is similar to the intra-blocks used in spatial domain based algorithms. Here, a block is discarded, if the dissimilarity between the reference block and best match is greater than a threshold, and the motion vector corresponding to that block is not coded. Second method is Bidirectional MRME (BMRME). In this, the temporal (or direction) information is predicted for the blocks in the coarsest resolution sub-images and the same information was used for corresponding blocks in the finer resolution sub-images. The third is a *fast* MRME technique, where, the directional detail sub-images at each level of the wavelet pyramid are combined into a single sub-image. MRME is then performed on the newly formed sub-images. A two-stage variable block-size MRME algorithm is proposed by Kim *et al.* [118]. In this algorithm, a method to reduce the amount of motion information is developed and a bit-allocation method minimizing the sum of the motion information and the prediction error is obtained in the wavelet transform domain.

6.3 Multiresolution Motion Vector Estimation Techniques

Motion estimation in multiresolution framework was pioneered by Zhang and Zafar [155, 156], Kim *et al.* [118] and Mandal *et al.* [81] and a modified version of this was proposed by Jayashree [64]. Main points are given here, details can be found in the original references. In all of these, bit rate or budget is the main bottleneck for motion estimation and compensation for low bit-rate applications. More so in the wavelet domain because,

1. As the motion estimation and compensation is done on detailed sub-images (higher spatial frequency), a small error in motion vector estimation makes the prediction ineffective, and

2. Motion estimation cannot be done on arbitrarily large sized blocks.

In MRME, a prediction scheme with modified error measure is proposed to make vectors close to optimal for all resolutions, such that the energy of the motion compensated

residual image is minimized. The scheme achieves it without increasing the motion vector overhead, but with a slight increase in computational cost.

The scheme starts with a given video frame decomposition into three levels of a wavelet pyramid, as shown in Fig 6.4, and the algorithm is as follows:

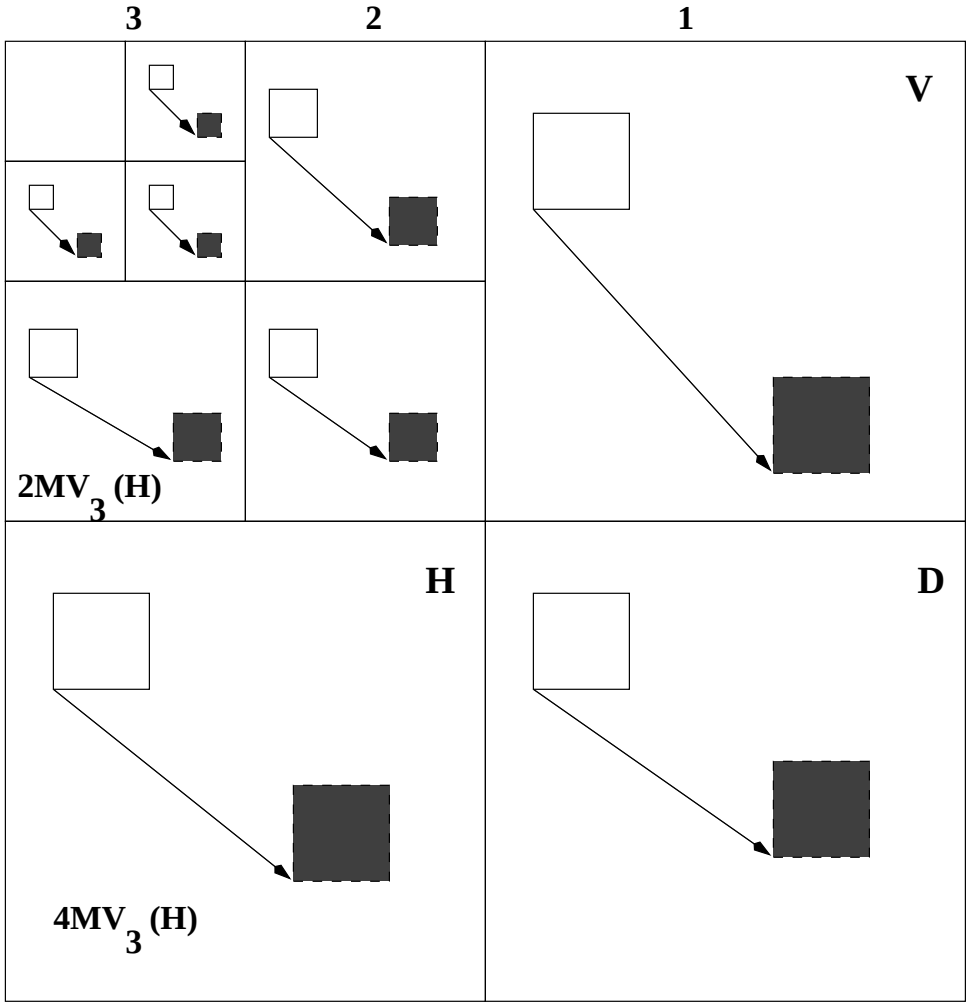


Figure 6.4: 3-level wavelet decomposition of a video frame. Empty blocks: position of block in current frame, shaded:position of the block used for motion compensation from previous frame. Motion vectors are obtained at level-3 are propagated to perform motion compensation at levels 2 and 1, as shown by arrow heads

1. The sum of absolute displacement (SAD) measure is used as an error criterion for estimating motion vectors. The SAD for the block B of size $O \times P$ centered at (o, p) in current frame i with respect to displacement of δo and δp in previous frame

$(i-1)$ is given by

$$\begin{array}{l} SAD(B_i(o,p), B_{i-1}(o+\delta o+\delta p)) = \\ \sum_{m=-o/2}^{o/2} \sum_{n=-p/2}^{p/2} |B_i(o+m, p+n) - B_{i-1}(o+m+\delta o, p+n+\delta p)| \end{array} \tag{6.1}$$

2. The MRME scheme proposed by Zhang and Zafar [156] estimates motion vectors for blocks of size 2×2 for all sub-images of level 3. The motion vectors (MV)s obtained at this level are appropriately scaled to perform motion compensation on blocks of size 4×4 and 8×8 levels 2 and 1 respectively. For example, $MV_3(H)(o,p)$, $MV_3(V)(o,p)$ and $MV_3(D)(o,p)$ represent the motion vectors for 2×2 sized blocks centered at (o,p) for level 3 horizontal, vertical and diagonal edge sub-images respectively. These motion vectors are propagated to other levels to perform motion compensation as:

$$\begin{array}{l} MV_i(H)(2^{3-i}o, 2^{3-i}p) = 2^{3-i} MV_3(H)(o,p) \\ MV_i(V)(2^{3-i}o, 2^{3-i}p) = 2^{3-i} MV_3(V)(o,p) \\ MV_i(D)(2^{3-i}o, 2^{3-i}p) = 2^{3-i} MV_3(D)(o,p) \\ for \;\; i = 1, 2 \end{array} \tag{6.2}$$

3. To make motion compensation more meaningful at levels 2 and 1, the SAD error criterion of (6.1) is modified as below while estimating Motion Vectors at level3:

$$\begin{array}{l} MSAD(B_i(o,p), B_{i-1}(o+\delta o, p+\delta p)) \\ \equiv SAD(B_i(o,p), B_{i-1}(o+\delta o, p+\delta p)) \\ +SAD(B_i(2*o, 2*p), b_{i-1}(2*(o+\delta o), 2*(p+\delta p))) \\ +SAD(B_i(4*o, 4*p), b_{i-1}(4*(o+\delta o), 4*(p+\delta p))) \end{array} \tag{6.3}$$

This measure is called *Multiresolution SAD* (MSAD). The first term in MSAD is same as the Eqn. (6.1), while the other two take into account the error incurred by 4×4 and 8×8 sized blocks at levels 2 and 1 centered at $(2*o, 2*p)$ and $(4*o \times 4*p)$ respectively. Thus, MVs at level 3 are computed first and then propagated to levels 1 and 2 as per Eqn. (6.2). Motion compensation is done at levels 1 and 2 with MVs obtained using MSAD Eqn. (6.3)

Motivation for doing estimation and compensation in the wavelet domain is to avoid the computation of inverse wavelet transform so as to make the overall video encoder-decoder fast. It is observed that traditional block-based motion estimation and compensation methods, like DCT,work well with hybrid video coders which incorporate block transforms, unlike the wavelet transform which is global in nature. To remove blocking

artifacts caused by motion compensation in the spatial domain, overlapping block motion compensation is suggested by Nogaki and Mohata [99]. This method gives better results when used along with wavelet transform, but with increased computational cost, due to the need to compute IDWT. In effect, this makes the decoder slow [65].

The basic multiresolution estimation scheme proposed by Jayashree [64] is similar to the other proposed methods [81, 118, 156], wherein, motion vectors at finer levels of the wavelet transform are the refinements of the motion vectors at coarser level. In those, it was assumed that the motion vectors at different levels are highly correlated[1]. If the motion vectors at a coarser level are not accurately estimated, then it produces sub-optimal motion vectors at all other finer levels. Hence, they [64] use the above assumption of motion vectors being highly correlated at different resolutions while coding them rather than while estimating them. Their proposed scheme estimates the motion vectors independent of the motion vectors at the coarser level.

6.4 Motion Vector Interpolated Zooming of Compressed Video

In this section we propose to zoom a compressed video. First, we show that it is possible to interpolate motion vectors in the compressed domain, and such a technique is independent of the compression method. To show that motion vector interpolation is independent of the compression scheme, we consider compression in both DCT and DWT domains. For this purpose, we assume the coding/decoding blocks are not present in the Figures 6.1 and 6.2 (hence, they are shown in dotted lines). Later on, we extend it to the proposed multiresolution motion estimation based zooming technique. Comparing the outputs obtained by interpolating motion vectors for two different methods, we show that the proposed method is fairly general and is independent of the compression method. We have chosen DCT and DWT for the motion vector interpolation. We select these two transformations, as they are the popular standards for data compression. (A comparison of these two techniques is given by Zixiang Xiong *et al* [157].)

We assume that the given video frame is de-quantized, decoded and motion vectors are available. With this assumption, the task on hand is to interpolate these motion vectors. We will show how this is done in the DCT domain, and follow it with necessary modifications for the DWT.

[1]This assumption is the basis of our proposed video zooming technique

Let ${}^k b_p^{16}$ be the k^{th} macroblock of p^{th} frame of a given video sequence - in spatial domain. Superscript 16 denotes size of the macroblock, 16×16 in this case. Similarly, let us denote ${}^k b_{p-1}^{16}$ as the macroblock in the previous instant or the $(p-1)^{th}$ frame, corresponding to ${}^k b_p^{16}$. That is, ${}^k b_{p-1}^{16}$ has moved to ${}^k b_p^{16}$, in one time instant. Our aim is to generate ${}^k b_p^{32}$, a 32×32 macroblock from these two blocks. Since we need to work in a transformed domain, let us denote the transforms of these two blocks as: ${}^k B_p^{16}$ (DCT of ${}^k b_p^{16}$) and ${}^k B_{p-1}^{16}$ (DCT of ${}^k b_{p-1}^{16}$). To estimate motion in the neighborhood Ω, a block matching method is to be adopted. Some of the block matching schemes were mentioned in the section (6.2.1). We consider the MAD scheme. Define,

$$ {}^k w \stackrel{\triangle}{=} ({}^k w_x, {}^k w_y) = \underset{(i,j)\in\Omega}{arg\,min} \sum_{m=0}^{15}\sum_{n=0}^{15} |{}^k b_p^{16}(x+m, y+n) - {}^k b_{p-1}^{16}(x+i+m, y+j+n)| \quad (6.4) $$

where (x, y) is the upper left corner coordinates of the p^{th} block. Next let error ${}^k \epsilon_p$ be defined as

$$ {}^k \epsilon_p \stackrel{\triangle}{=} |\sum_{i=0}^{15}\sum_{j=0}^{15} \{{}^k B_p^{16}(i,j) - {}^k B_{p-1}^{16}(i,j)\}| \quad (6.5) $$

where (i, j) is the transform domain coordinates for the DCT of the macroblock p and the DCT of the motion compensated macroblock, $p-1$. We assume ${}^k w$ and ${}^k \epsilon_p$ are available during decompression. The task on hand now is to extrapolate these motion vectors from ${}^k w$ to ${}^k \hat{w}$ and use it to zoom the video. Based on extensive empirical studies, we have found that a linear extrapolation scheme works well, namely ${}^k \hat{w}_x = 2({}^k w_x + 1)$ and ${}^k \hat{w}_y = 2({}^k w_y + 1)$. The new macro block locations are now evaluated as,

$$ {}^k \hat{B}_p^{16}(i,j) = ({}^k B_{p-1}^{16}({}^k \hat{w}_x.2x + i, {}^k \hat{w}_y.2y + j) + {}^k \epsilon_p \quad (6.6) $$

It is mentioned in [15, 94] that zero-padding provides a good approximation to high resolution (zoomed) DCT. We use the same idea for zooming the 16×16 DCT block ${}^k B_p^{16}$ to size 32×32. Let

$$ {}^k B_p^{32} = \begin{bmatrix} {}^k \hat{B}_p^{16} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (6.7) $$

where, $\mathbf{0}$ is the 16×16 null matrix. Inverse of ${}^k B_p^{32}$ gives ${}^k b_p^{32}$ or a 32×32 video frame in spatial domain. In effect, we obtain a video frame zoomed by a factor of two Note that zooming is done in the *compressed* domain (as we are interpolating motion vectors), and *not* in the spatial domain. The motion vector interpolation is depicted pictorially in Fig.6.5.

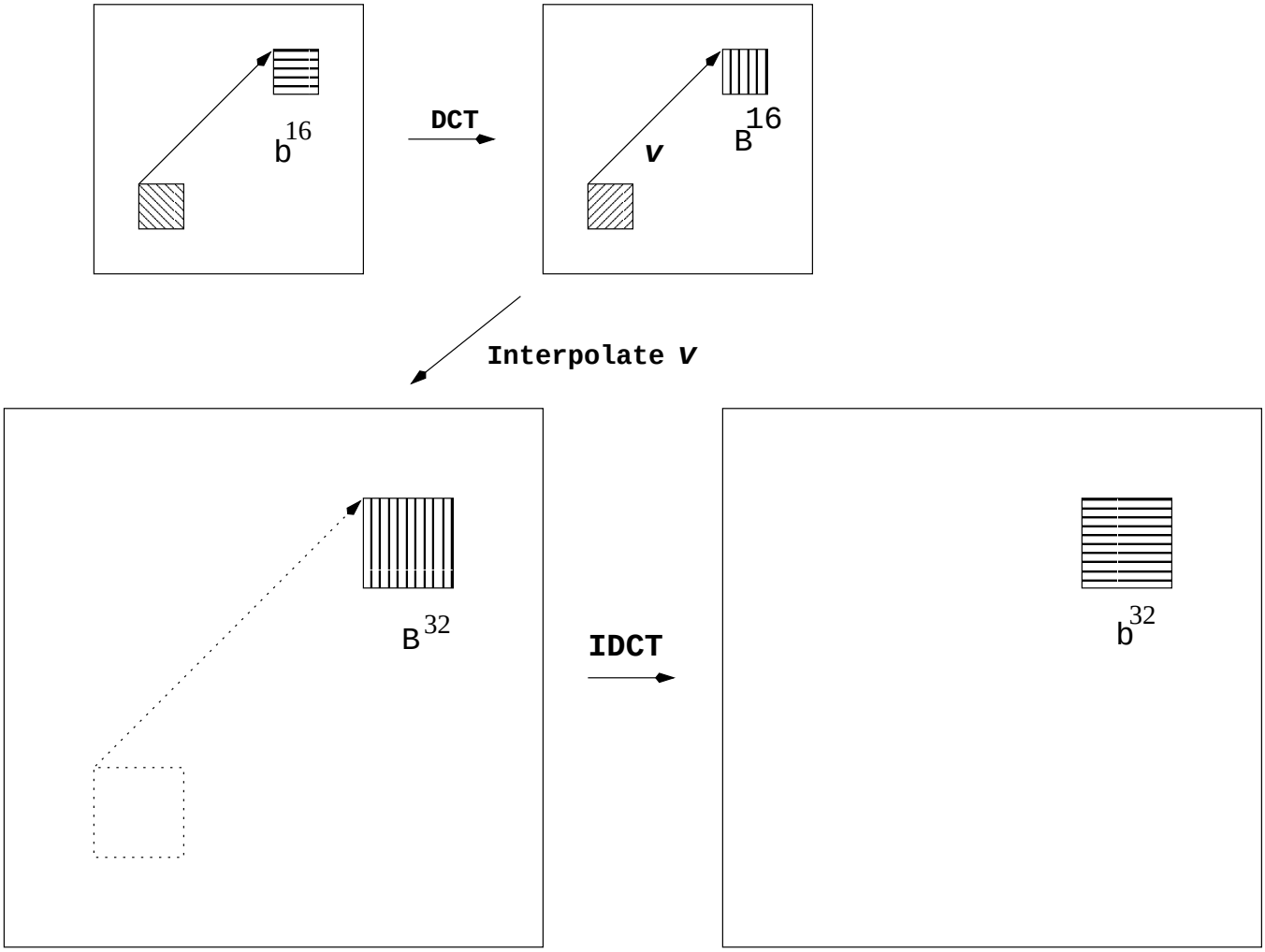


Figure 6.5: Motion Vector interpolation: On top we have a low resolution image in spatial and transformed domain. We have shown a block at the p^{th} instant and $p-1^{th}$ instant. Bottom figures show (left) block position after interpolating motion vectors in transform domain and (right) spatial domain reconstruction

Unlike the DCT, wavelets have a nice structure to distinguish boundaries of sub-bands. From the proposed still image zooming method, discussed in Chapter 4, we see that we can estimate wavelet coefficients at a finer scale, leading to a better zoomed image. We extend the same principle and **0** of Eqn. (6.7) is replaced by *estimated* wavelet coefficients as

$$ {}^{k}B_p^{32} = \begin{bmatrix} {}^{k}\hat{B}_p^{16} & {}^{k}\hat{B}_p^{16}(V) \\ {}^{k}\hat{B}_p^{16}(H) & {}^{k}\hat{B}_p^{16}(D) \end{bmatrix} \tag{6.8} $$

where, ${}^{k}\hat{B}_p^{16}(V)$ correspond to the estimated wavelet coefficients in the vertical direction. Similarly, H and D stand for horizontal and Diagonal directions, respectively. Methodology for estimating these wavelet coefficients remains the same as explained in Chapter 4 (Section 4.5).

6.4.1 Performance Measure

Normal method of comparing performance is (P)SNR evaluation. For this, we need the original image or video. Another option is to down-sample the given image or video to generate a low resolution image or video, zoom it and then compare it with the original. In the absence of availability of original image, alternate methods are to be formulated. We compare the performance in the following way:

For Signal to Noise Ratio (SNR) computation, we generate a zoomed video sequence *without using motion vectors*, in the DCT framework. Each frame is divided into 16 $\times$ 16 blocks and these blocks are extended to 32 $\times$ 32 directly by zero-padding, without using motion information. This represents the worst case scenario. We use this video represented as $(\tilde{X})$ as the reference and the zoomed video $\hat{X}$ obtained using either DCT or DWT schemes as the output of the zoomed video. SNR is measured for each frame p as

$$ SNR(frame\ p) = \frac{\sum_{i,j} \tilde{X}(i,j)^2}{\sum_{i,j}(\tilde{X}(i,j) - \hat{X}(i,j))^2} \tag{6.9} $$

Fig.s 6.6 - 6.8 show the SNR plots for Suzie video and Fig.s 6.9 - 6.11 shows that for Claire video. The plots and resulting video clips suggest that the proposed method for video zooming works quite well. In particular, the DWT based scheme works better.

We also plot the error plots e, where, error is defined as

$$ e = \sum_{i,j}(\tilde{X} - \hat{X}) \tag{6.10} $$

Fig.s(6.12-6.14) show the error plots for Suzie video. These error plots and resulting videos suggest that the proposed method of motion vector interpolation and one that

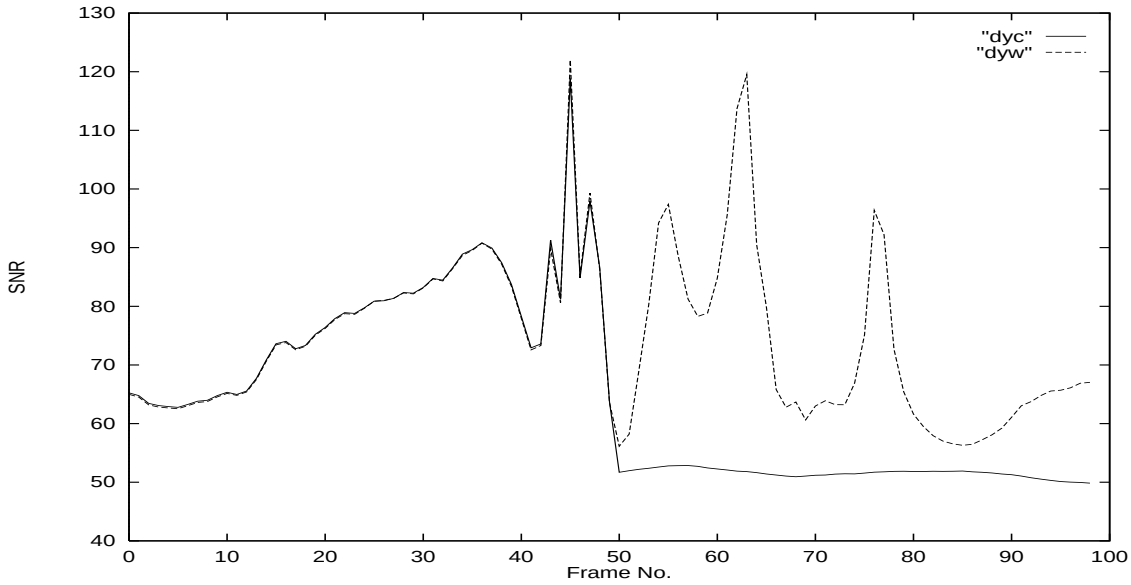


Figure 6.6: SNR plot for Y component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

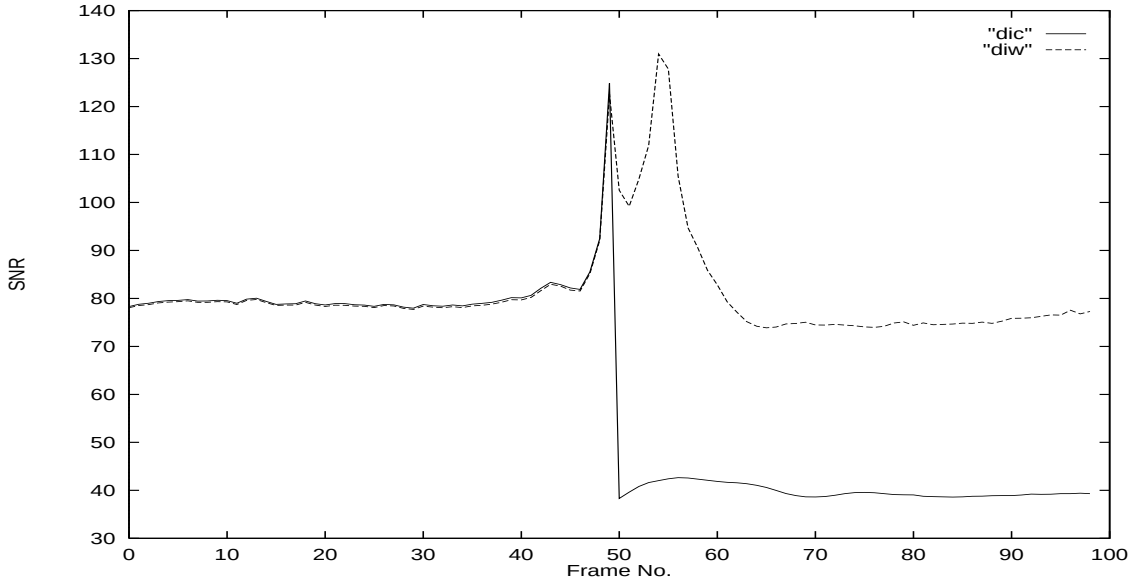


Figure 6.7: SNR plot for I component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

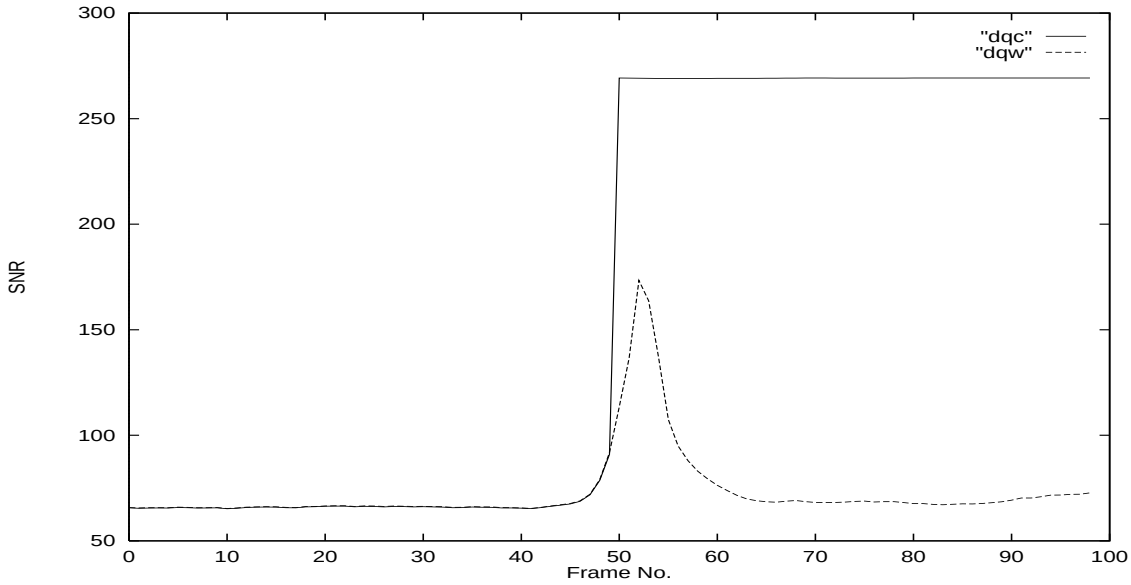


Figure 6.8: SNR plot for Q component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

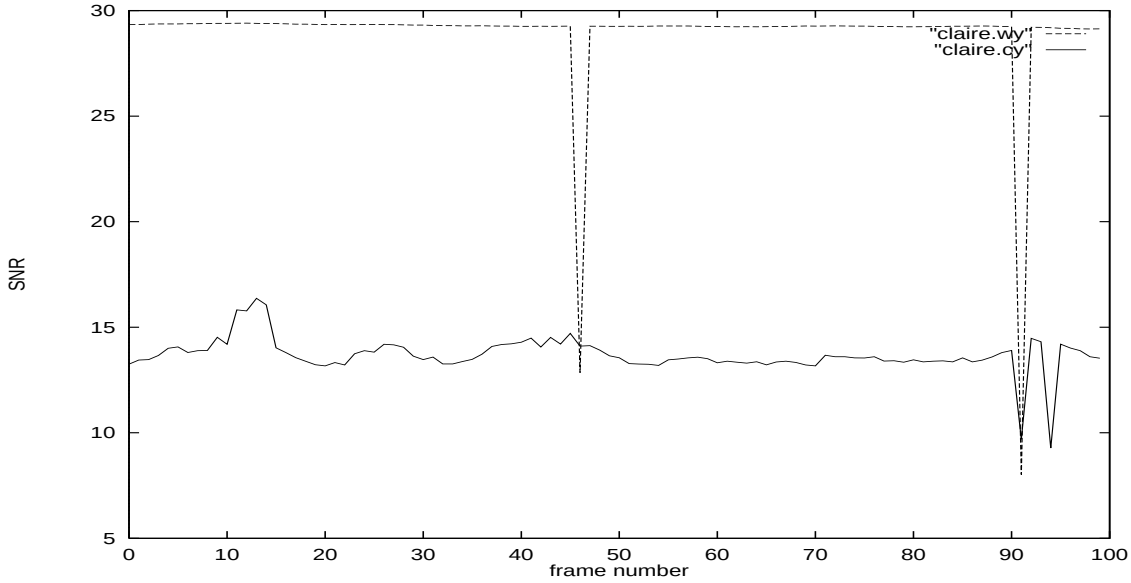


Figure 6.9: SNR plot for Y component of Claire video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

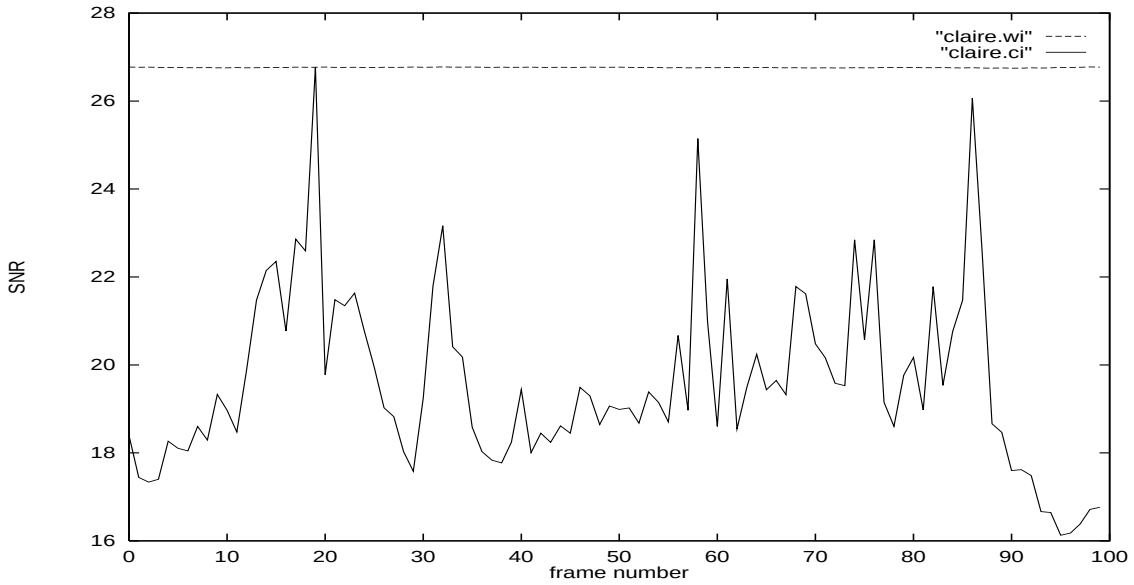


Figure 6.10: SNR plot for I component of Claire video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

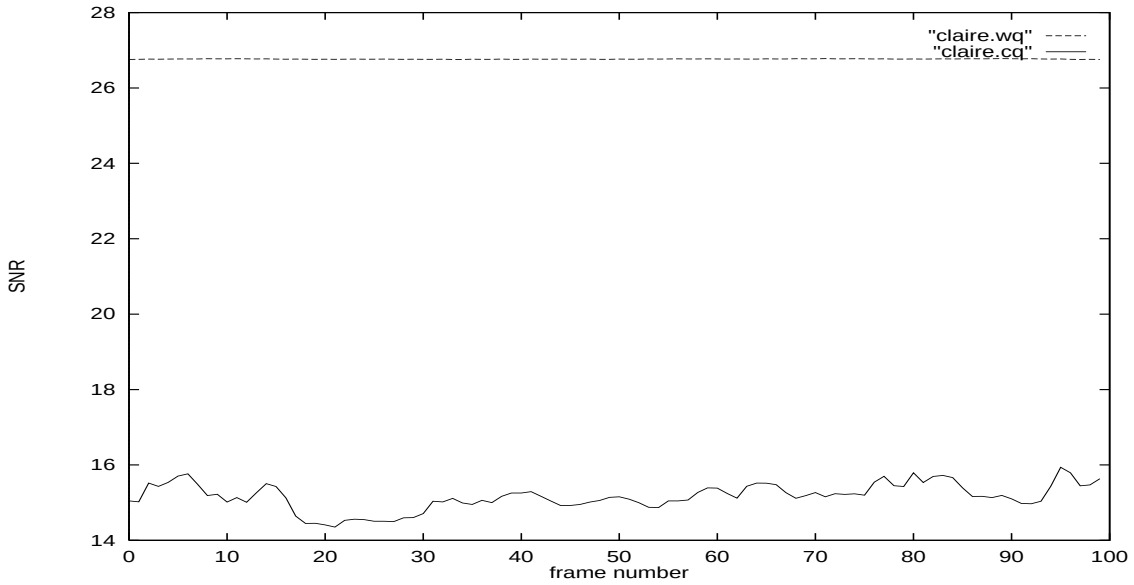


Figure 6.11: SNR plot for Q component of Claire video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

does not consider motion vectors give similar results. That is, the proposed video zooming in compressed domain works satisfactorily, especially for wavelet domain motion vector interpolation.

We have considered QCIF video clips for our experiment; but the technique should work satisfactorily for other video formats as well.

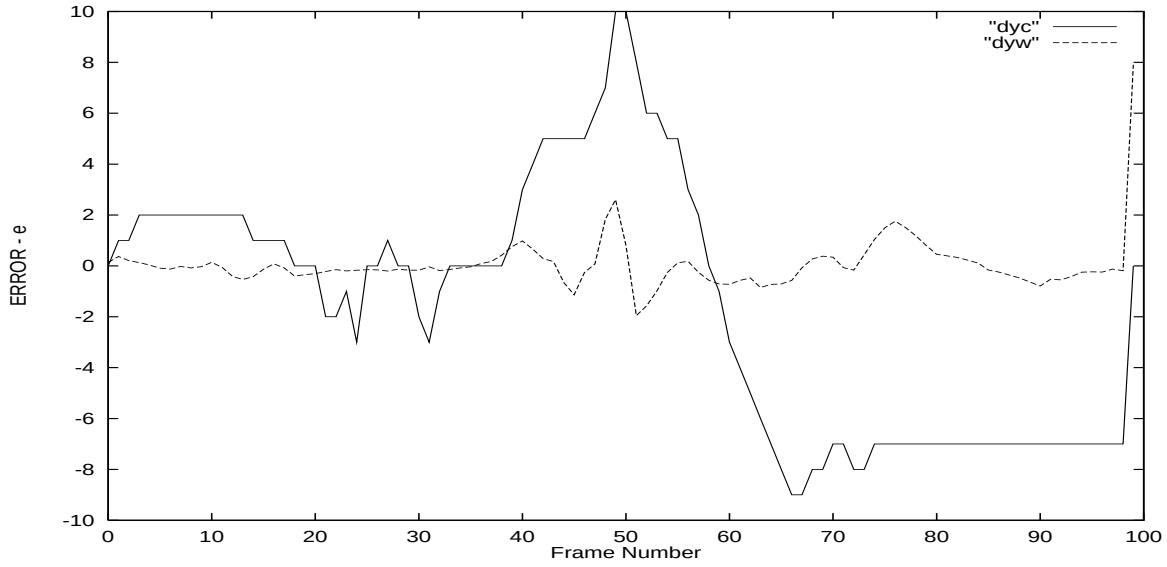


Figure 6.12: Error plot (Eqn. 6.10) for Y component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

6.5 Zooming of Video Stream Compressed by MRME Method

In this section, we extend the motion vector interpolation to a video clip, which is actually in a compressed form. We use the MRME based video compression algorithm developed by Jayashree [64]. We start with an MRME based compressed video. We partially decode it to obtain the motion vectors and then exploit the DWT based algorithm described in section 6.4 for video zooming. First we highlight MRME algorithm, and then mention the necessary modification for zooming.

The multiresolution motion estimation and compensation scheme discussed earlier saves the computation of IDWT, making the video encoder fast, but at the cost of increased bit-rate due to increase in the number of motion vectors. These motion vectors to be transmitted as side-information are predictive coded. The motion activities for a

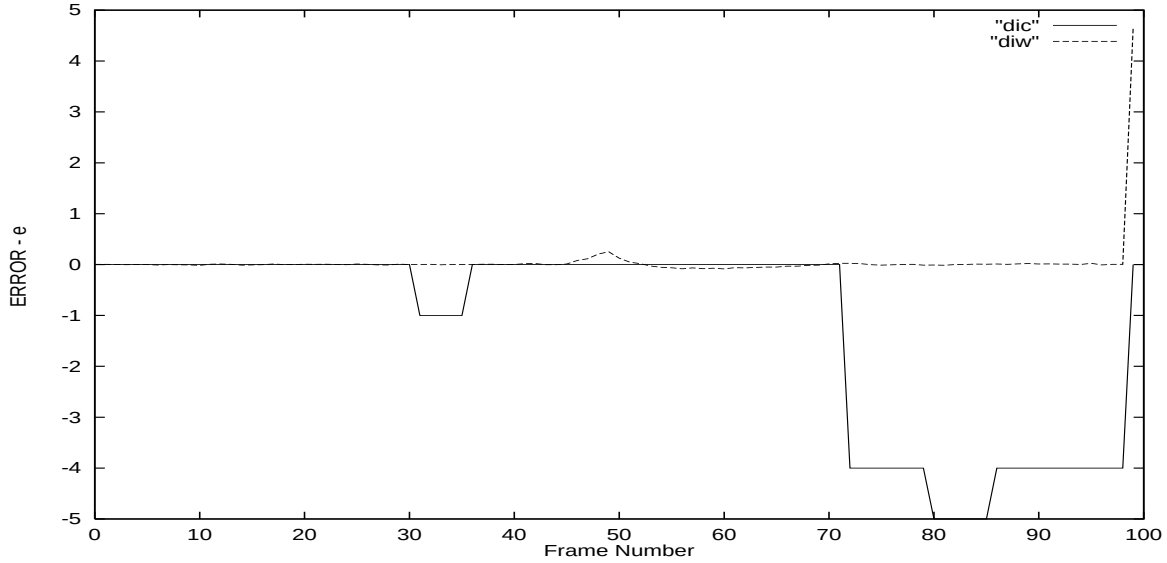


Figure 6.13: Error plot for I component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

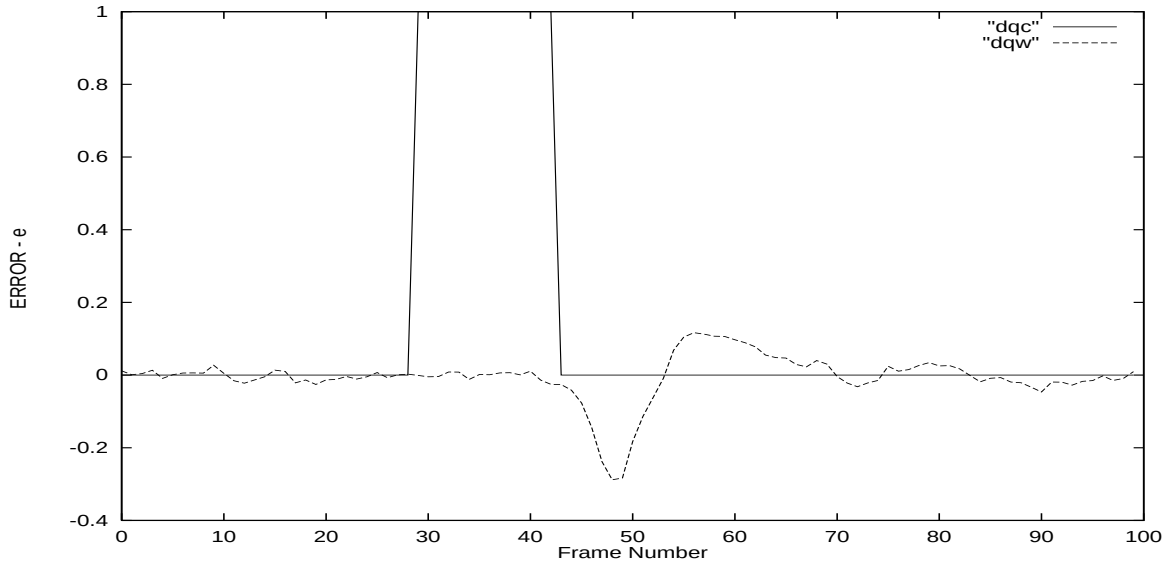


Figure 6.14: Error plot for Q component of Suzie video. Thick lines for DCT based interpolation and thin lines for DWT based interpolation

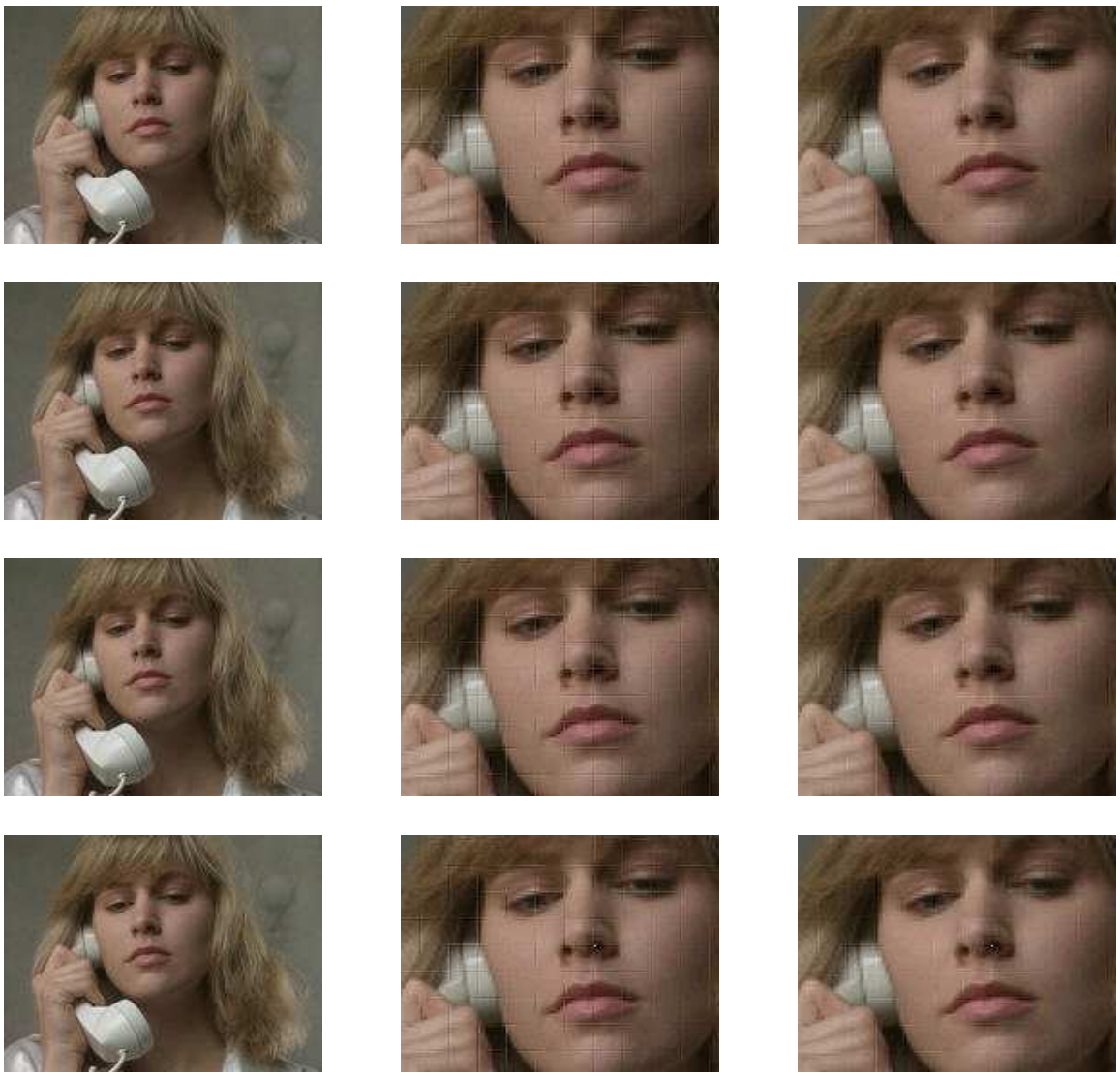


Figure 6.15: Zoomed video frames of Suzie video. Left column is the original frames. Middle column is DCT zoomed frames and right column is DWT zoomed video frames. The image are shown correspond to frame numbers 6 to 10

particular detail sub-image at different resolutions are highly correlated since they actually specify the same motion structure at different scales. Hence to keep the increased bit-rate under control, and to efficiently code these motion vectors, the zerotree data structure of Shapiro [127] can be applied on the motion vectors generated from the above MRME scheme. Here, the underlying image on which zerotree data structure is applied, is not a regular wavelet transform, but the points that specify motion vectors of a wavelet transformed frame.

6.5.1 Coding of Motion Vectors using Zerotree

The basic idea is: if a motion vector is insignificant at a coarser location, then it is likely that the motion vectors at the same location in the next finer resolution for the same detail sub-image are also likely to be insignificant. To exploit these dependencies and predict the significant motion vectors, we use the zerotree data structure. In principle, it is like the SPIHT coding algorithm for wavelet-transformed coefficients, with scalars (wavelet coefficients) being replaced by vectors (motion vectors). Successive approximation quantization of motion vectors is not considered, as that leads to sub-optimal motion vectors. We only predict the zero-motion vectors and once they are predicted, they are sent as zerotree roots to the decoder.

Following sets of symbols are used to present the algorithm:

- $\mathcal{O}(i,j)$: set of offspring nodes of (i,j)
- $\mathcal{D}(i,j)$: set of all descendants of (i,j)
- $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$
- $\mathcal{H}$: coordinates of all spatial orientation tree roots (nodes corresponding to highest level)
- $S_n(i,j)$ is a function to indicate the presence of non-zero motion vector in a set of coordinates $\{i,j\}$.

The offspring structure at level-3 for different detail sub-images is as follows:

- For vertical edge sub-image $\mathcal{O}(i,j) = \{(2i,j),(2i+1,j)\}$
- For horizontal edge sub-image $\mathcal{O}(i,j) = \{(i,2j),(i,2j+1)\}$
- For diagonal edge sub-image $\mathcal{O}(i,j) = \{(i,j)\}$

From level 3 onwards, each node has four off-spring nodes at next the level for all detail sub-images, namely,

$$\mathcal{O}= \{(2i, 2j), (2i, 2j+1), (2i+1, 2j), (2i+1, 2j+1)\}$$

There are no off-spring present at level-1 diagonal detail sub-image (Fig. 6.16), as no

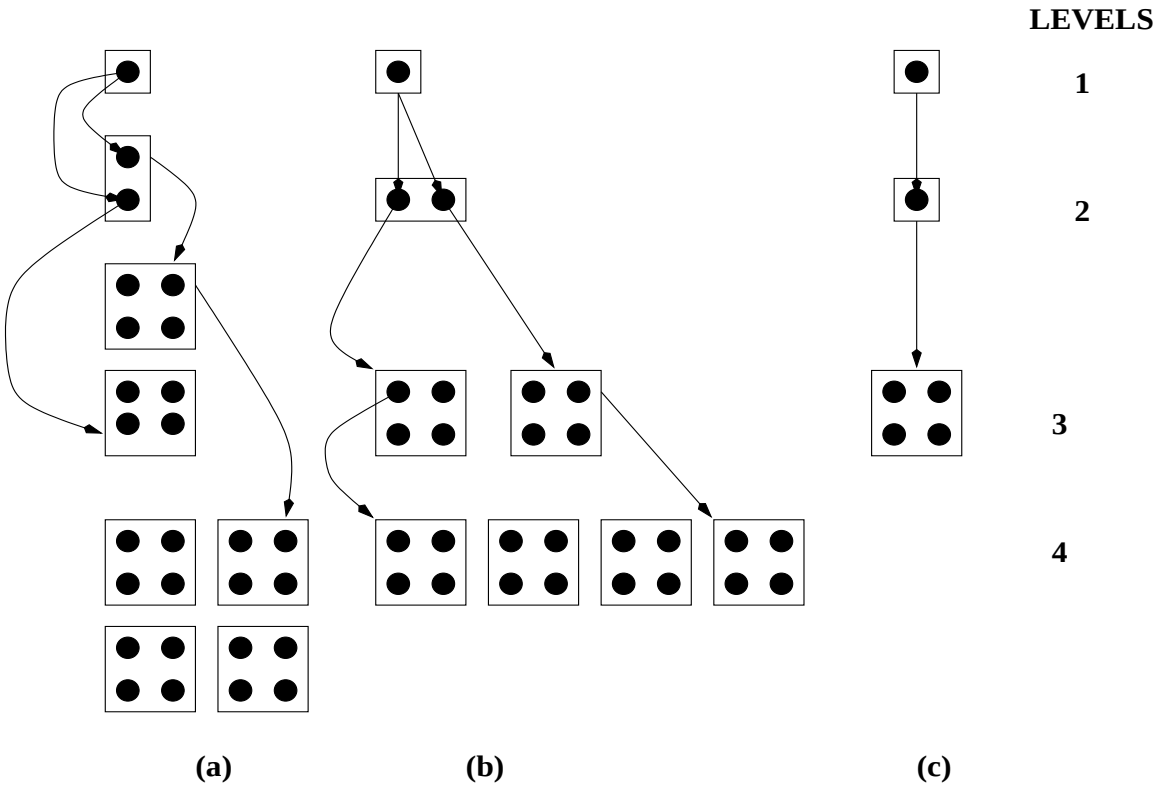


Figure 6.16: Motion vector tree across wavelet pyramid. Motion vector (x, y) (a) vertical (b) horizontal and (c) diagonal sub-images.

motion estimation is performed on D_1. With this new offspring structure, the zerotree motion vector coding algorithm is similar to SPIHT.

The set partitioning rules are now:

1. initial partition is formed with the sets (i, j) and $\mathcal{D}(i, j) \in \mathcal{H}$,
2. if $\mathcal{D}(i, j)$ contains a non-zero motion vector then it is partitioned into $\mathcal{L}(i, j)$ plus single-element sets with $(k, l) \in \mathcal{O}(i, j)$, and
3. if $\mathcal{L}(i, j)$ contains a non-zero motion vector then it is partitioned into sets $\mathcal{D}(k, l)$ with $(k, l) \in \mathcal{O}(i, j)$.

Here we use *list of insignificant sets* (LIS) and *list of insignificant pixels* (LIP), instead of *list of significant pixels* (LSP). Each entry is identified by a coordinate (i, j). An entry in LIS represents either set $\mathcal{D}(i, j)$ (called type D) or $\mathcal{L}(i, j)$ (type L)

Algorithm

1. Initialization:
   Add the coordinates $(i, j) \in \mathcal{H}$ to both LIP and LIS as type D entries.

2. Sorting phase:
   (a) for each entry (i, j) in LIP do:
   output $S_n(i, j)$; if $S_n(i, j) = 1$ code motion vector corresponding to location (i, j).
   (b)for each entry in LIS do:
   (i) if the entry is of type D, then output $S_n(\mathcal{D}(i, j))$; if $S_n(\mathcal{D}(i, j)) = 1$ then

   - for each $(k, l) \in \mathcal{O}(i, j)$ output $S_n(k, l)$
     -if $S_n(k, l) = 1$, code motion vector corresponding to location (k, l)
   - if $\mathcal{L}(i, j) \neq 0$ then move (i, j) to the end of LIS as an entry of type L (ii)
     if the entry is of type L then output $S_n(\mathcal{L}(i, j))$
   - if $S_n(\mathcal{L}(i, j)) = 1$ then add each $(k, l) \in \mathcal{O}(i, j)$ to the end of LIS as an entry of type D; remove (i, j) from LIS

We can observe the absence of quantization and refinement steps in the above algorithm.

6.5.2 Video Encoder - Decoder

A brief description of the proposed video encoder is given here. Details are available in [64, 65]. Each frame of the video sequence is encoded either as intra (I) or inter (P - forward prediction) frame. Intra frame is used for the first and every fifteenth frame, and inter frame is for the in-between frames. The intra frames are wavelet transformed with five levels and coded using SPIHT [119] and adaptive arithmetic coder [51, 148]. SPIHT is based on the following two ideas: (i) exploiting self-similarity inherent in the wavelet transform to predict the location of significant information across scales and (ii) successive approximation quantization of those significant wavelet coefficients. The inter-frames of the video sequence are wavelet transformed with five levels and encoded by forward prediction from the preceding wavelet transformed and SPIHT decoded frames. The motion estimation and compensation algorithm of the previous section can be used to predict the blocks in the current frame. Motion vectors thus estimated are coded using

the zerotree motion vector coder. Motion compensated wavelet coefficients are quantized and coded using SPIHT and adaptive arithmetic coder. The block diagram of the video encoder scheme is shown in Fig.6.17. The video decoder is an exact reverse of the video encoder.

Algorithm:

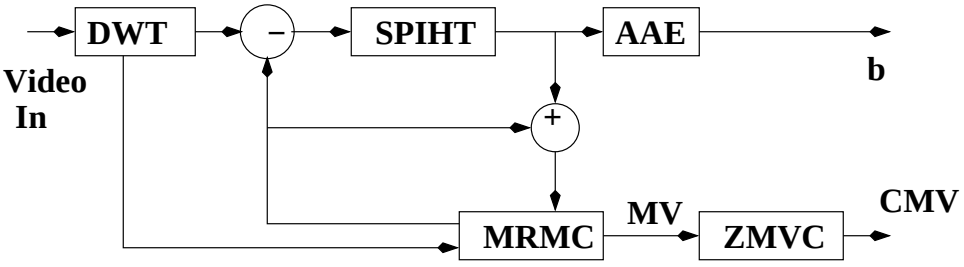
```
(a) For intra frame:
    - take DWT of the frame ;
    - quantize and code wavelet coefficients using SPIHT  and adaptive
arithmetic coder till target bit-rate /distortion is achieved   ;

(b) For inter-frame:
    - take DWT of the frame ;
    - Do multiresolution motion estimation and compensation in wavelet domain
using a variable block structure.
/* Search area for motion vectors is + or - 7.*/
Motion vectors for D1 detail sub-image are set to zero. ;
    - code motion vectors using zerotree motion vector coder. ;
    - quantize and code predicted wavelet coefficients using SPIHT and
adaptive arithmetic coder till bit-rate/distortion is achieved ;
```

Zooming is performed during the decoding. Algorithm for decoding is the reverse of the above algorithm, with interpolating motion vectors is:

```
(a)  For intra frame:
- Decode and de-quantize coded bit stream ;
- Interpolate wavelet coefficients (Chapter 4) ;
- Take 2x IDWT
(b) For inter-frame
- Decode and de-quantize coded bit stream
- Generate the motion vectors ;
- Interpolate motion vectors (section 6.3)
- Take 2x IDWT
```

As per the above algorithm, given the video is compressed and zoomed while decompressing. Compression and decompression are carried out in wavelet domain, hence, Eqn. (6.8) is valid while zooming. SNR values for Claire video is plotted in Fig. 6.18. The



DWT : Discrete Wavelet Transform
SPIHT: Set Partitioning in Hierarchical Trees
MRMC : Multiresolution Motion Compensation
b: Output bits
MV: Motion Vectors
CMV: Coded MVs
ZMVC: Zerotree Motion Vector Coder

Figure 6.17: Proposed Video encoder - block diagram

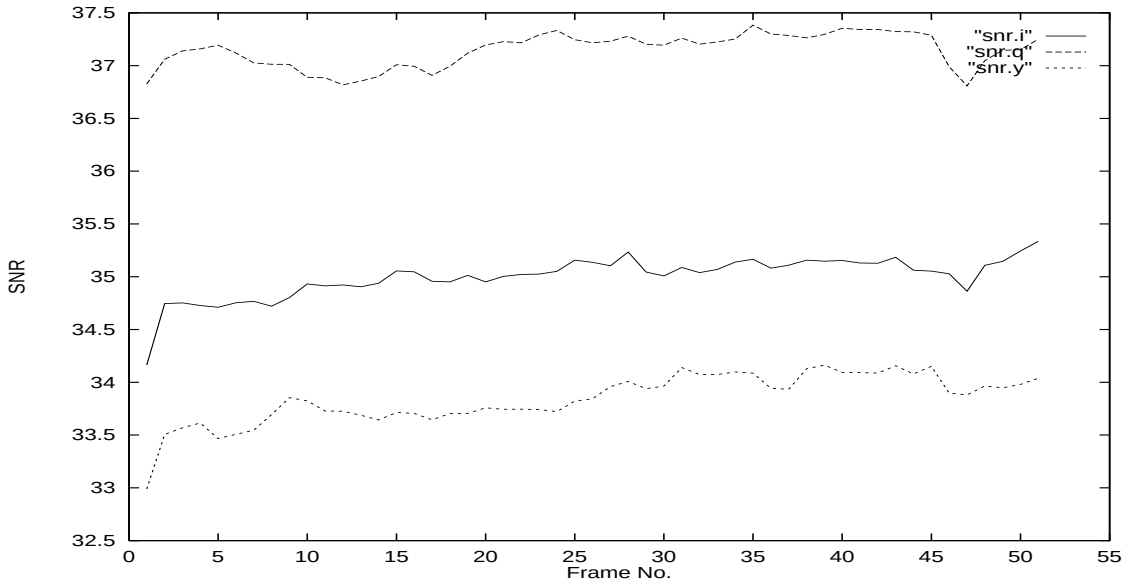


Figure 6.18: SNR plot for Claire video compressed at 12 Kbps. Top line is for Q component, middle is for I component and bottom line for Y component, for the first 50 frames

SNR values of this graph are for decompressed video and *not* for zoomed video. Decompressed video has notable amount of distortions due to very low-bit rate coding (11 Kbps in compressed domain, as opposed to 3 Mbps in uncompressed QCIF domain). The same distortions are present in the zoomed video. Zoomed video clip (images) are shown in Fig. 6.19

The video images show that the proposed motion vector interpolation method works well, and the limitations and distortions are due to the compression and decompression techniques. The proposed method retains all the properties of a regular decompressed image while zooming.

6.6 Temporal Video Frame Interpolation using Motion Vectors

Motion vector interpolated video frame zooming method can be extended interpolating temporal video frames. This involves estimating *missing* frames in a given video stream. This is achieved by developing a DWT based method for temporal interpolation of motion vectors. For comparison, we also implement a linear interpolation scheme.

Suppose, we have two frames $p-1$ and $p+1$, and we need to estimate the *missing* frame p. As in the previous section, let ${}^{k}b_{p+1}^{16}$ and ${}^{k}b_{p-1}^{16}$ be the k^{th} macroblock of frames

Figure 6.19: MRME based decompression and zooming. Top row shows decompressed Claire video after compressing at 11Kbps. Bottom row is zoomed version of the same compressed video clip

$p+1$ and $p-1$ respectively. Let ${}^kB^{16}_{p+1}$ and ${}^kB^{16}_{p-1}$ be their respective DWTs. The MAD is now defined as

$${}^kw = ({}^kw_x, {}^kw_y) = \underset{(i,j)\in\Omega}{arg\,min} \sum_{m=0}^{15}\sum_{n=0}^{15} |{}^kb^{16}_{p+1}(x+m, y+n) - {}^kb^{16}_{p-1}(x+i+m, y+j+n)| \quad (6.11)$$

Error is defined as,

$${}^k\epsilon_p = |\sum_{i=0}^{15}\sum_{j=0}^{15}\{{}^kB^{16}_{p+1}(i,j) - {}^kB^{16}_{p-1}(i,j)\}| \quad (6.12)$$

Motion vectors for the p^{th} frame are estimated as ${}^k\hat{w}_x = \lfloor\frac{{}^kw_x}{2}\rfloor$ and ${}^k\hat{w}_y = \lfloor\frac{{}^kw_y}{2}\rfloor$. The new block location in the p^{th} frame will be,

$${}^k\hat{B}^{16}_p = ({}^kB^{16}_{p-1}({}^k\hat{w}_x.x + i, {}^k\hat{w}_y.y + j) + {}^k\epsilon_p \quad (6.13)$$

Since the size of the estimated video frame p is the same as the available frames viz., $p-1$ and $p+1$, we do not need Eqn.s (6.7 and 6.8).

We compare this scheme with linear frame interpolation, where a pixel at location (i,j) of frame p is the average of pixels at the same (i,j) locations in frame $p-1$ and $p+1$. Note that the linear frame interpolation is done in the *spatial* domain and not in the compressed domain. Performance is compared using SNR as defined in Eqn. (6.9). Plots of this method are shown in Fig.s 6.20 - 6.25. Fig.s 6.20 - 6.22 are the plots for the Claire video and 6.23 - 6.25 for the Suzie video. We see that visually there is not much difference between linear and motion vector (MV) interpolated methods. But the PSNR plots show that linear interpolation is slightly better than the proposed method. This is due to the fact that MV based scheme is carried out in compressed domain. During MV estimation, coding and quantization errors may have crept in. However, the difference is small, and MV based scheme is comparable, if not better, to the linear interpolation scheme. Here, we note that for SNR computation, original frame is available, unlike the zooming application.

6.7 Discussion

Results of the all three methods of motion vector interpolation for video zooming are shown in the appropriate figures. From the SNR and the error (e) plot, we see that the proposed motion vector interpolation technique works well for DCT and DWT based methods, with, DWT performing better than DCT method. This justifies our claim that the proposed technique is independent of the compression method.

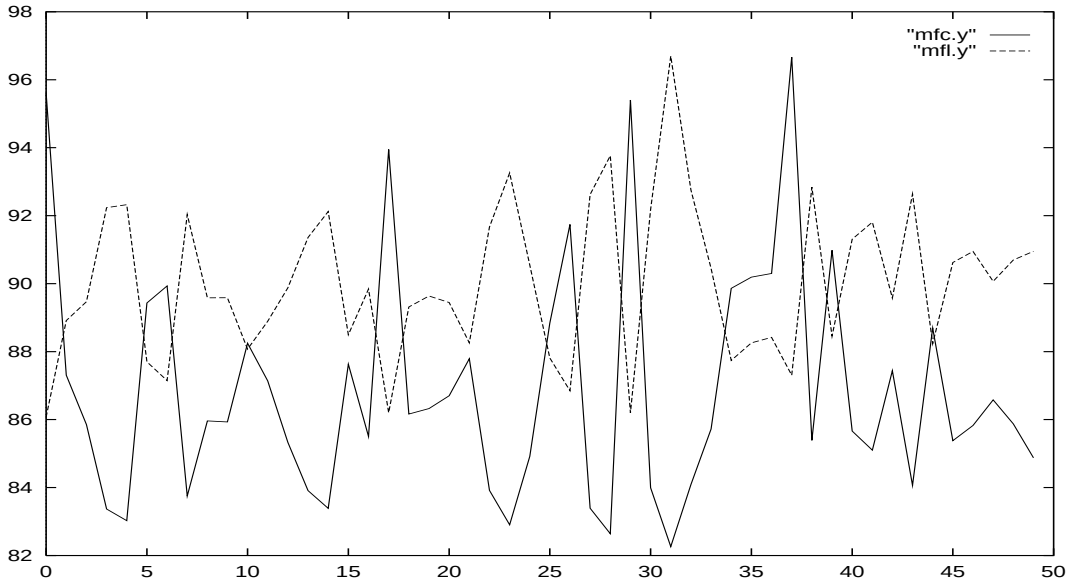


Figure 6.20: SNR plot for Y component of frame interpolated Claire video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

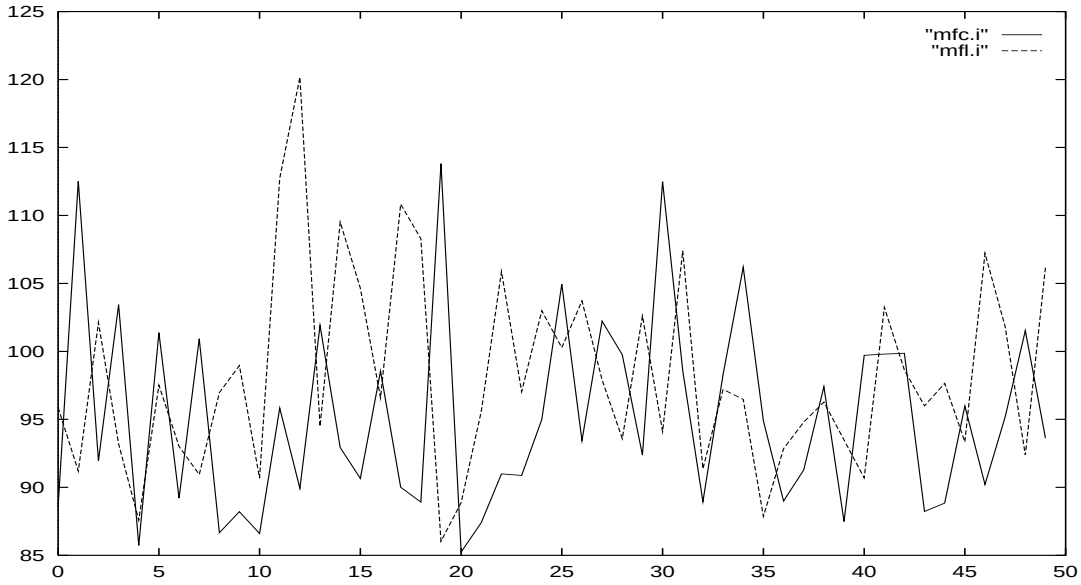


Figure 6.21: SNR plot for I component of frame interpolated Claire video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

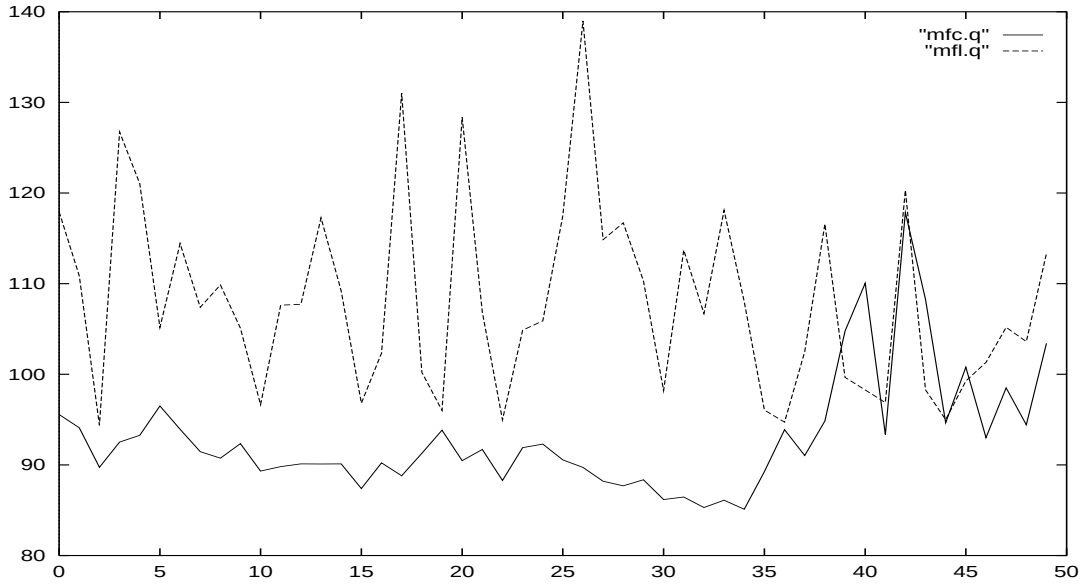


Figure 6.22: SNR plot for Q component of frame interpolated Claire video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

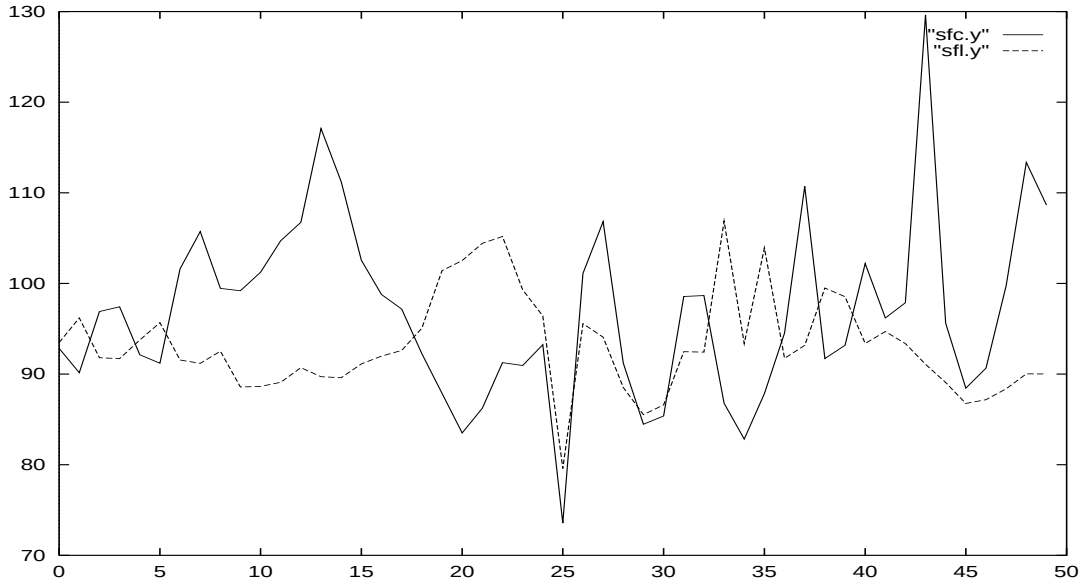


Figure 6.23: SNR plot for Y component of frame interpolated Suzie video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

We see that for the MRME method, the image is not as sharp as that of the DWT and DCT. The reason is that during compression and decompression, some high frequency components are lost. That is, performance of the proposed zooming algorithm is dependent on coding and decoder and decoder efficiencies. Zoomed video retains the properties of the decoded video.

Extension of the proposed method to temporal interpolation also gives results comparable to the interpolation carried out in spatial domain. Here, the performance of the proposed method is slightly inferior to the linear interpolation of spatial domain. This may be due to the coding and quantization errors that are present while estimating the motion information. However, the difference in performance is not significant and visually both the methods look similar.

6.8 Summary

This chapter introduced concepts of video compression, with emphasis on motion coding using block matching techniques. We extended the ideas of video compression to zooming by interpolating the motion vectors in the transform domain. It is shown that motion vector based interpolation works satisfactorily, for DCT and DWT. We extended the motion vector interpolation to MRME, for a real compressed image and obtained encouraging results. We extended the motion vector interpolation to temporal video interpolation. This scheme gives results comparable to linear frame interpolation of spatial domain.

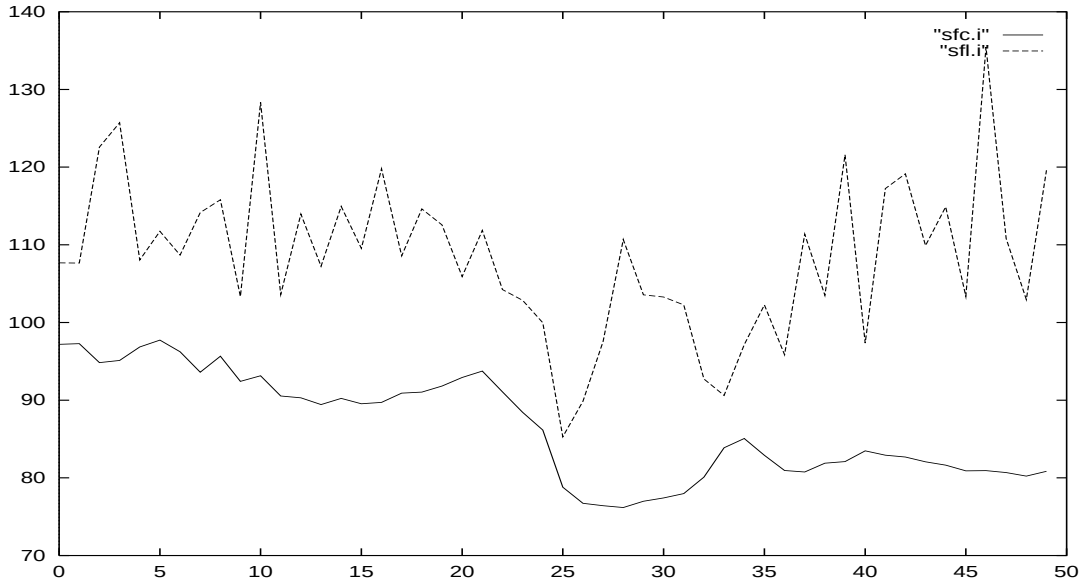


Figure 6.24: SNR plot for I component of frame interpolated Suzie video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

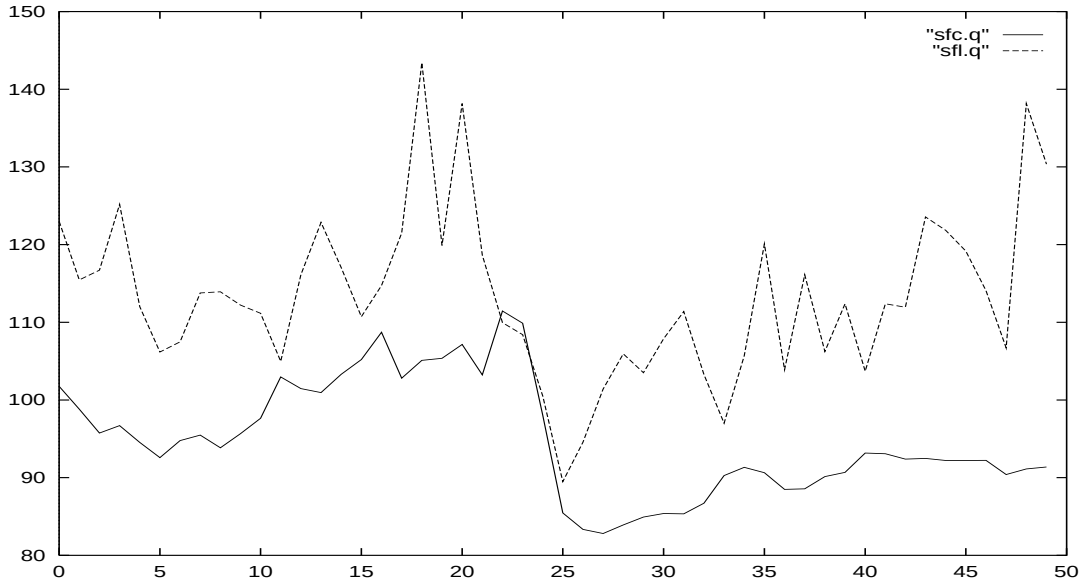


Figure 6.25: SNR plot for Q component of frame interpolated Suzie video. Thick lines for DWT based interpolation and thin lines for linear frame interpolation. Y axis is the SNR and X axis is the frame No.

Figure 6.26: Frame interpolated Claire video clips. Left column is the original frames. Middle column is linear frame interpolation and right column is DWT frame interpolation. Video clips show frame numbers 11,13,15 and 17