# H.264 STANDARD BASED SIDE INFORMATION GENERATION IN WYNER-ZIV CODING

by

SUBRAHMANYA VENKATRAV

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2010

## ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor Dr. K.R.Rao for the esteemed guidance and support. I also like to thank the other members of my advisory committee Dr. Manry and Dr. Mingyu Lu for reviewing the thesis document and offering insightful comments. I appreciate all members of the Multimedia Processing Lab for their support during my research work. Finally, I am grateful to my family for their support, patience, and encouragement during my graduate journey.

April 16, 2010

ABSTRACT

H.264 STANDARD BASED SIDE

INFORMATION GENERATION

IN WYNER-ZIV

CODING

Subrahmanya Venkatrav, M.S.

The University of Texas at Arlington, 2010

Supervising Professor: K. R. Rao

The traditional video coding involves a high complexity encoder and relatively low complexity decoder. The complexity of the encoder is mainly due to motion estimation [3] and other prediction techniques employed . There is a new paradigm emerging in video coding - Distributed video coding (DVC). This is based on Slepian-Wolf and Wyner-Ziv6 theorems. In Wyner-Ziv video coding, encoder is less complex compared to the decoder. The decoder is very complex due to prediction involved for side information (SI) generation.

There is lot of research done previously on linear density parity check codes (LDPC) and H.263 standard based SI generation for Wyner-Ziv coding. In our approach, we consider LDPC codes for error control and H.264 standard based SI generation. The basic Wyner-Ziv encoder and decoder are implemented to test this method. This is then compared against other SI generation methods.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

## 1.1. Wyner-Ziv video coding

The video compression is an area of intense research activity. The need for video compression arises from the fact that any video sequence contains large amount of data. In order store or stream a video sequence a compression algorithm is required. The compression can be lossless or lossy in which case quality of the video should not be degraded significantly.

The traditional video compression standards like MPEG2, MPEG4, H.263 and H.264 provide means of compressing video with required bit rate with degradation in quality which is inversely proportional to bit rate [3]. These coding standards achieve compression by removing redundancy in video. The video sequence is represented as frames and there is strong correlation between neighboring video frames and this is called temporal redundancy. There also redundancy within pixels of a frame and this is termed as spatial redundancy. The traditional video coding algorithms try to reduce this redundancy while encoding a video sequence. The encoder does this by employing prediction techniques and entropy coding.

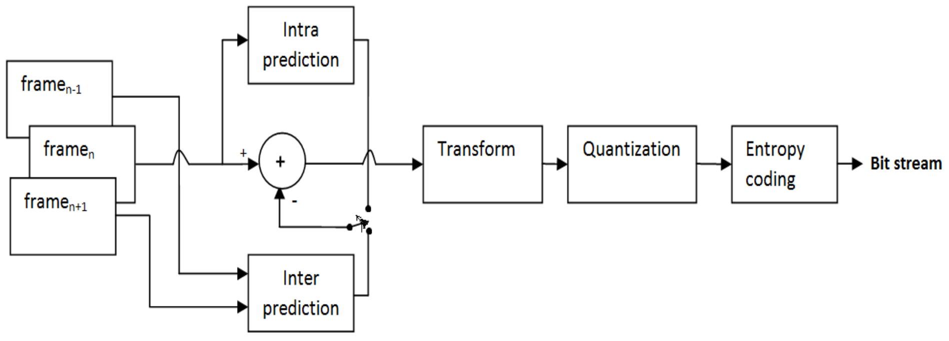

Figure 1.1 Traditional video encoding

The traditional video encoding uses Intra prediction (prediction within a frame) and inter prediction (prediction between frames) as shown in Figure 1.1. A transform is applied on prediction error and the transform coefficients are quantized. Entropy coding is done on quantized coefficients to obtain a video bit stream. The required bit rate for the output is attained by varying quantization parameter [3]. The encoder is very complex in nature due to intra and inter predictions involved. Typically for inter prediction is done via motion estimation which requires estimating the motion of an object involving lot of computations. On the other hand Wyner-Ziv decoding tries to achieve same amount of  compression with a different type of encoder and decoder setup.

The distributed video coding (DVC) or Wyner-Ziv video coding is a new paradigm in video coding based on Slepian and Wolf's [10] and Wyner and Ziv's Information theorems [9]. This is a lossy compression with decoder side information. It enables low complexity video encoding with most of the complexity shifted to the decoder side. This performs better than the conventional video coding under deteriorating channel conditions. The Wyner-Ziv encoding considered in this thesis consists of encoding some of the frames either using intra-frame prediction and rest using inter-frame prediction. The inter-frame encoding is highly complex in traditional video encoding techniques involving computational intensive tools like motion estimation. However in Wyner-Ziv inter-frame encoding is done via simple prediction methods and hence redundancy is not reduced fully. The prediction error is then encoded using error control codes. The compression is achieved by sending only partial bit stream  for inter frames. At the decoder side the prediction for inter frames is obtained using decoded  intra-frames. The partial inter bit stream sent is then used to correct errors in the prediction. The error correction techniques are used for this purpose.

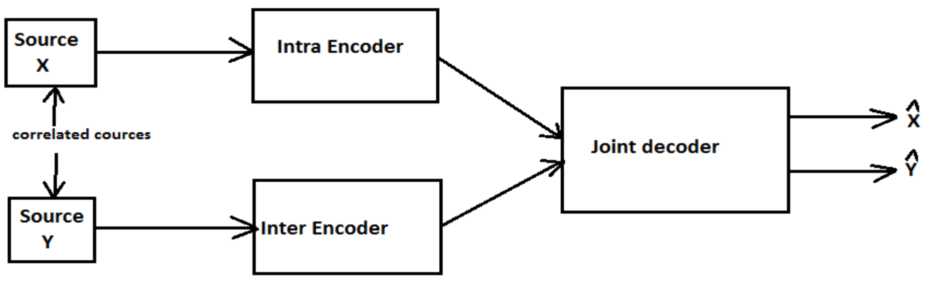

Figure 1.2 Distributed video coding [20]

The prediction techniques used in traditional video codecs can be applied for prediction in Wyner-Ziv coding In this thesis Wyner-Ziv coding also. This is already done where coding tools employed by H.263 standard [26] have been used in Wyner-Ziv coding [1]. In this thesis the prediction techniques employed by H.264 standard [6][7][8][19] are used. The design of encoder and decoder are discussed in detail in section 1.2 and 1.3 respectively.

## 1.2. Wyner-Ziv Encoder

The input for the encoder is YUV frames in 420 format. This format is shown in Figure 1.3. Each YUV 420 format has 3 components. The intensity component is called luma and represented as Y. The color components are called chroma and there are two components namely U and V. The chroma components are sub-sampled by 2 compared to luma component. The processing of each frame is done in-terms of blocks. The size of block depends on processing and generally one of three sizes are considered: 4x4, 8x8 and 16x16.

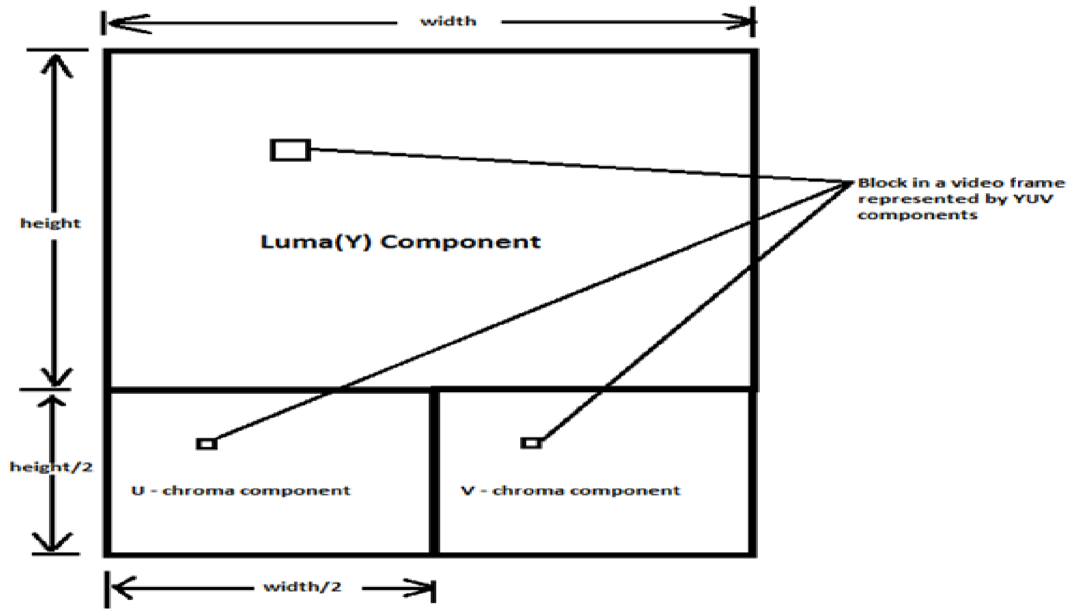

Figure 1.3 YUV 420 format

The Wyner-Ziv encoder encodes a frame in one of two types considered below:

Key frames: In this case the prediction is done within frame. In other words the decoding of these frames at the decoder side does not require any other frames. In this thesis key frames are encoded using H.264 intra frame [6][7][8][19]. This is briefly discussed in section 1.2.1.

WZ frames: In this case the difference between WZ frame and previous reconstructed key frame calculated and the output is quantized and encoded using low-density-parity-check (LDPC) code. LDPC is discussed in detail in section 4.1.1.2.

The overall encoding process is illustrated in Figure 1.4. Here even frames are encoded as key frames and odd frames are encoded as WZ frames. The encoding of key frame and WZ frame is described in detail in sections 1.2.1 and 1.2.2.

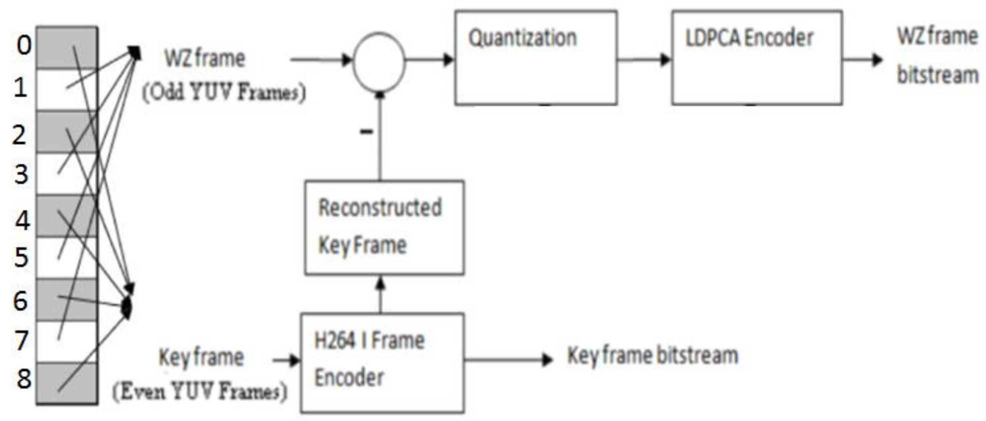

Figure 1.4 Block diagram of WZ encoder

### *1.2.1. Key frame encoding*

The intra frame encoding in H.264 standard [6][7][8][19] is used for encoding key frames. In this case each block is then predicted from neighboring blocks . This is done using intra prediction which is discussed in detail in chapter 5. The residual macro block is obtained by subtracting predicted block from original block. The 2D-Integer Discrete Cosine Transform is applied on residual block. The output is then quantized and encoded using entropy coder. This is illustrated in Figure 1.5.

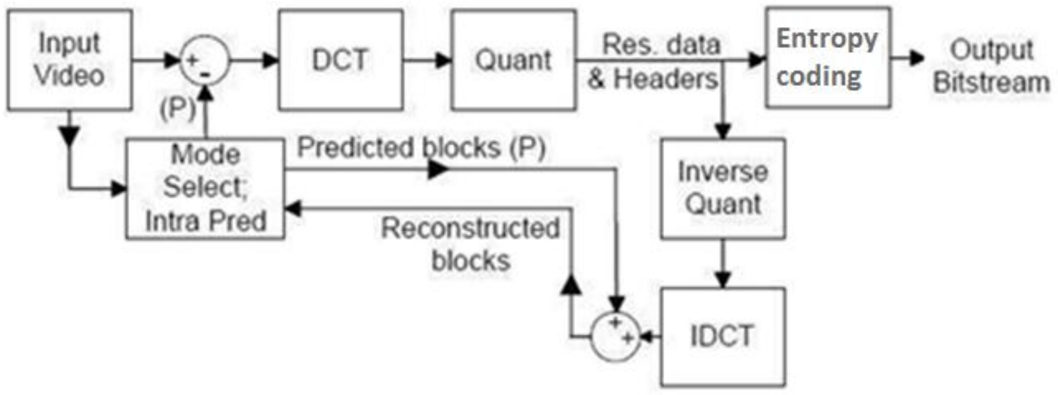

Figure 1.5 I-frame encoding in H.264[23]

1.2.1.1 Intra prediction

H.264 uses intra prediction to predict the block to be encoded using neighboring blocks in the same frame. The residual signal between the current block and the predicted block is encoded and sent to the decoder. The prediction is done in spatial domain unlike the other standards where prediction is done in transform domain. Intra-prediction uses the neighboring blocks based on the directional prediction mode. There are nine different directional prediction modes as shown in the Figure 1.6.

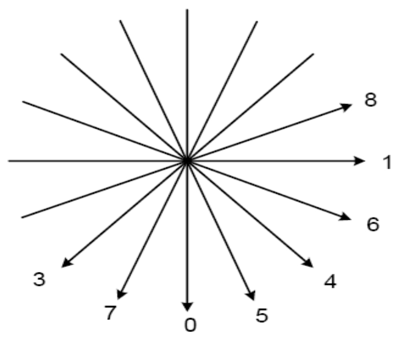

Figure 1.6 Nine spatial prediction modes used in H.264 standard [6]

The intra prediction for a block uses left, top, top left and top right blocks as shown in the Figure 1.7.

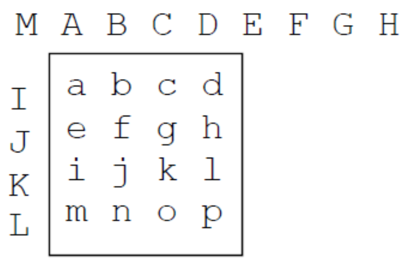

Figure 1.7 Block Intra prediction [6]

There are 16 blocks namely a, b…p and for each block a prediction block is generated using neighboring blocks. For example

1. Block ‘a’ has ‘I’ as left neighbor, ‘A’ as top neighbor, ‘M’ as top left neighbor and ‘B’ as top right neighbor.
2. Block ‘e’ has ‘J’ as left neighbor, ‘a’ as top neighbor, ‘I’ as top left neighbor and ‘b’ as top right neighbor.
3. Block ‘b’ has ‘a’ as left neighbor, ‘B’ as top neighbor, ‘A’ as top left neighbor and ‘C’ as top right neighbor.
4. Block ‘f’ has ‘e’ as left neighbor, ‘b’ as top neighbor, ‘a’ as top left neighbor and ‘c’ as top right neighbor.

The prediction block is generated using one of the direction modes shown in Figure 1.6. The direction mode to be used for the block can be derived from the direction mode of the co located block in the previous key frame. The prediction block generated is a linear combination of neighboring blocks and the linear combination depends on the direction mode. Following are the nine directional modes,

1. Vertical
2. Horizontal
3. DC
4. Diagonal Down Left
5. Diagonal Down Right
6. Vertical Right
7. Horizontal Down
8. Vertical Left
9. Horizontal Up

The prediction direction for each direction mode is shown in Figure 1.8.

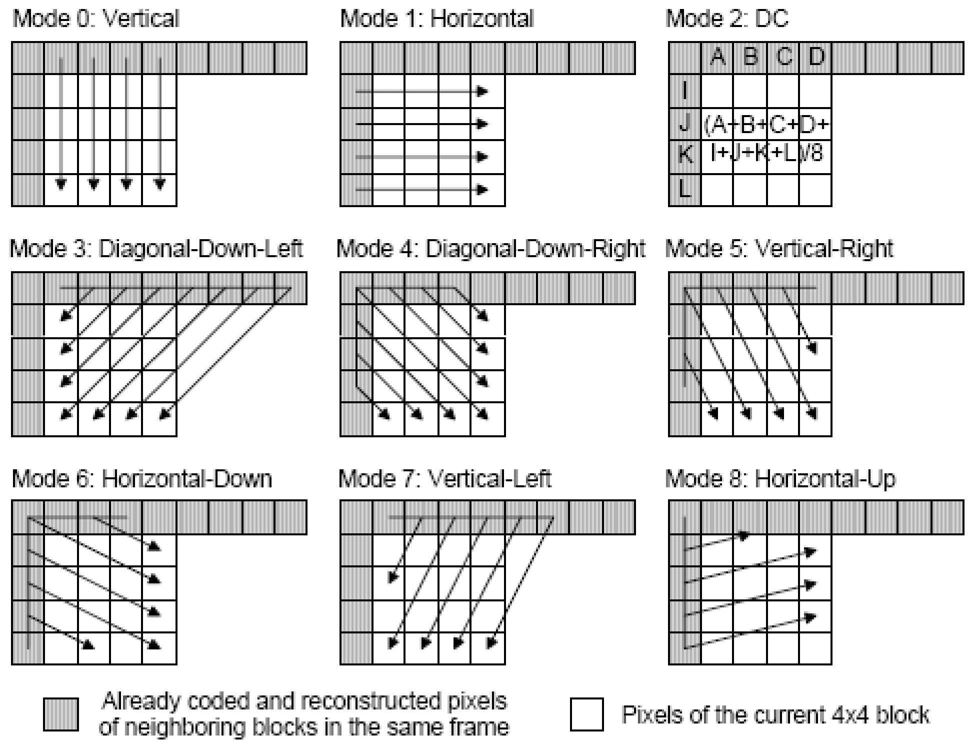
Mode 0: Vertical
Mode 1: Horizontal
Mode 2: DC
A
B
C
D
I
J
K
L
(A+B+C+D+ I+J+K+L)/8
Mode 3: Diagonal-Down-Left
Mode 4: Diagonal-Down-Right
Mode 5: Vertical-Right
Mode 6: Horizontal-Down
Mode 7: Vertical-Left
Mode 8: Horizontal-Up
Already coded and reconstructed pixels of neighboring blocks in the same frame
Pixels of the current 4x4 block

Figure 1.8 Intra prediction modes [24]

#### 1.2.1.2 Integer Transform

The integer transform is applied on 4x4 residual block between the current block and the intra predicted block. The Hadamard transform matrix is used and is given below [7][21],

$$\overline{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

The 2D transformation is applied as follows,

$$X = \overline{H} F \overline{H}^T$$

where F is the residual block and X is the transform domain output.

1.2.1.3 Quantization

This involves dividing the transformed block by a Qstep and multiplying by a scaling matrix (SF). The quantization can be written as [7][21],

$$Y = X \otimes round\left(\frac{SF}{Qstep}\right)$$

Here SF is the scaling matrix as follows,

$$SF = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

Where,

$$a = \frac{1}{2}, b = \sqrt{\frac{2}{5}}$$

The symbol $\otimes$ represents element by element multiplication of the corresponding matrices.

1.2.1.4 Scanning and Run-length encoding

The quantized coefficients are then scanned in a zig-zag order [6]. While scanning the continuous number of zero is counted and this count is called run. The non-zero value after a run of zeros is called level. The runs and levels are then encoded using entropy coding .

1.2.1.5 Entropy Coding

In this thesis only H.264 baseline profile is considered and in this case only Context Adaptive Variable Length encoding is supported. This uses Huffman coding with Exp-Golomb codes. Any value to be encoded is mapped to an Index which is an unsigned integer. This index is then encoded using the Exp-Golomb code word. The code word for a given index can be obtained as [6][7][8][19],

*code_word = [K zeros][1][K-bit DATA]*

such that,

*index = 2^K+ int(DATA)- 1*

where *int (DATA)* is the integer corresponding to the binary string. The table 1.1. shows the list of Exp-Golomb codes for first few indexes.

Table 1.1 Exp-Golomb codes

| Index | Code word |
|---|---|
| 0 | 1 |
| 1 | 010 |
| 2 | 011 |
| 3 | 00100 |
| 4 | 00101 |
| 5 | 00110 |
| 6 | 00111 |
| 7 | 0001000 |
| 8 | 0001001 |

The mapping of index to a value to be encoded is not fixed one instead it is context adaptive. There are twelve such mappings that standard specifies and selection among twelve is based on local statistics of the stream. The context adaptive VLC provides greater efficiency compared to fixed tables.

### *1.2.2. WZ frame encoding*

A simple prediction for WZ frame is previous key frame. The prediction error is calculated by subtracting WZ frame from key frame. The error is then encoded using Low Density Parity Check (LDPC) coding. The LDPC encoding is described in detail in section B. The following section briefly discusses the property of the residual signal.

#### 1.2.2.1 Residual signal Generation

Residual generation for WZ frame is shown in Figure 1.9 where frame- 0, frame-2 and frame- 4, … etc are encoded as I frames and frame-1, frame-3, frame-5, ... etc are encoded as

WZ frames. The residual is calculated as difference between WZ frame and the previous I frame.

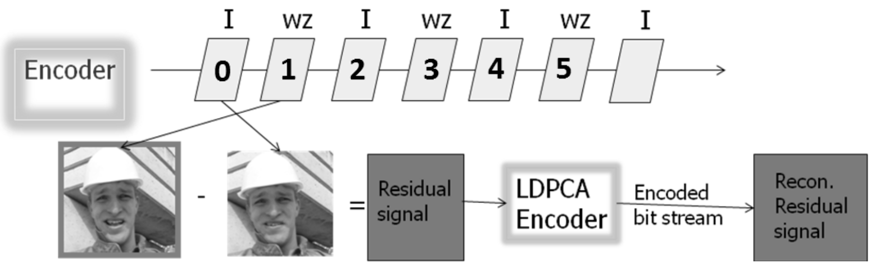

Figure 1.9 WZ frame encoding [4]

The probability density function (pdf) of the residual signal for three different video sequences is in shown in Figure 1.10. This pdf follows laplacian distribution closely. The pdf for luma and chroma residual signal is separately shown in Figure 1.11 and Figure 1.12 respectively. It can be observed that the chroma residual PDF follows laplacian more closely than luma.

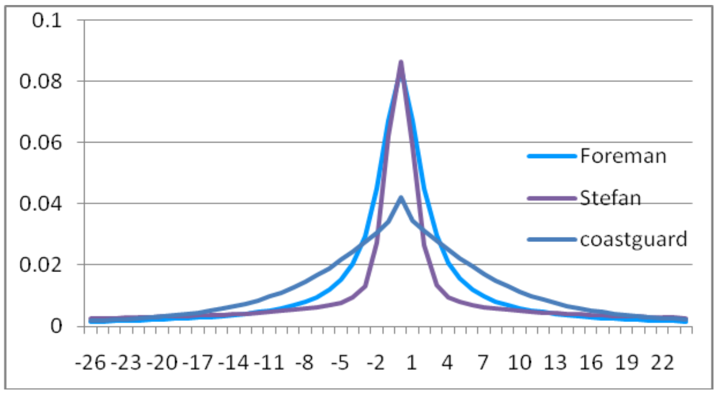

Figure 1.10 Pdf of residual frame [4]

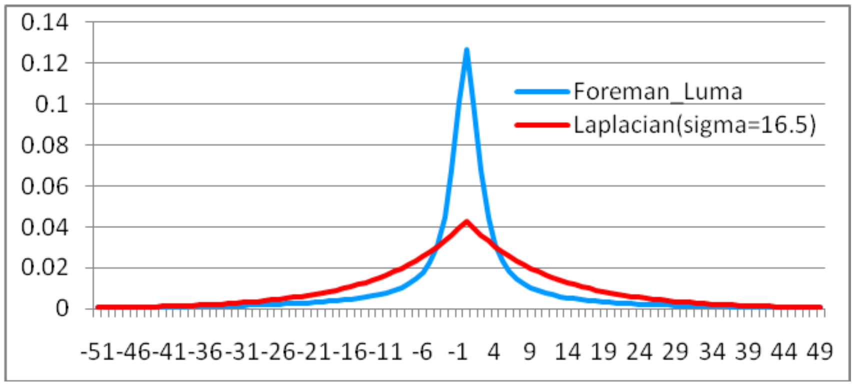

Figure 1.11 Pdf of luma residual frame [4]

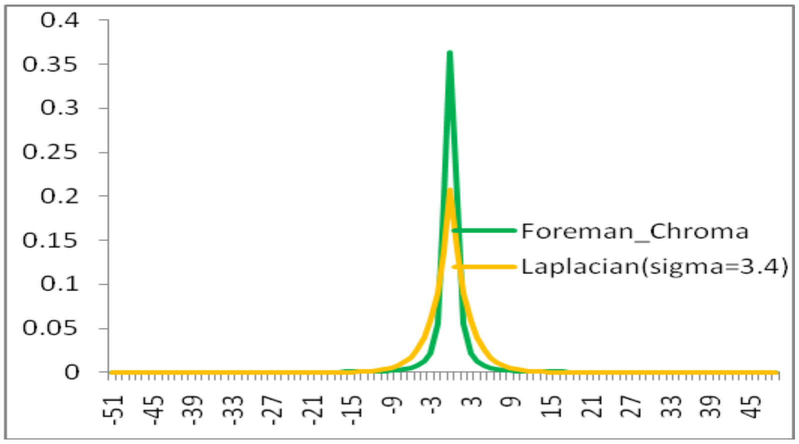

Figure 1.12 Pdf of chroma residual frame [4]

### 1.2.2.2 LDPC encoding

The LDPC encoder is implemented by Dr. Kim Jin Soo [4]. The soft decision encoding [25] is used for encoding message bits into message and parity bits. The LDPC parity bit generation is done using parity check matrices and this is illustrated in Figure 1.13. The parity check matrix for the case shown in the figure is

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

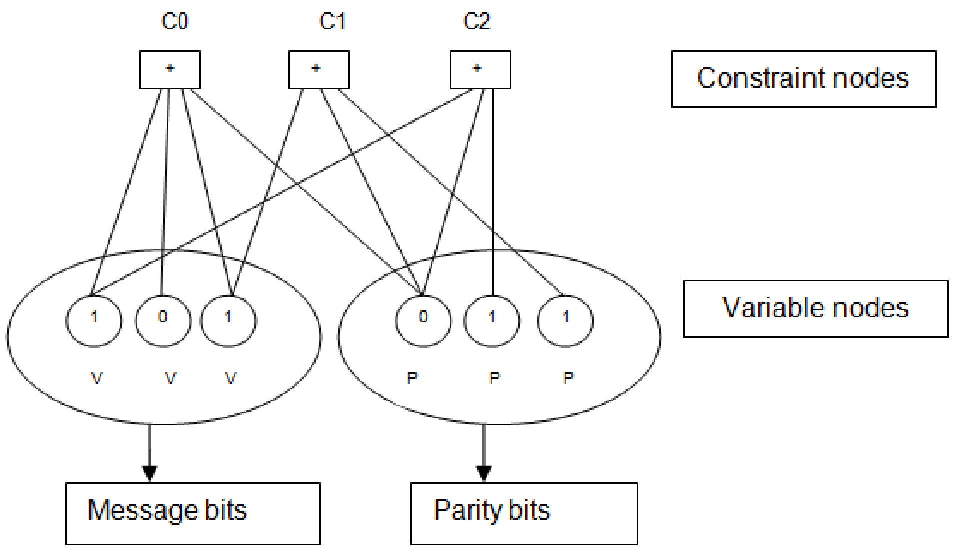

Figure 1.13 LDPC parity bits generation

The sum (modulo 2) of variable nodes connected to each constraint node should be zero. By using this parity bits for given message bits can be calculated iteratively. For example in the figure message bits 101 are considered. In this example,

C0 =0 = (V0+V1+V2+P0) % 2 => P0=0

C1 = 0 = (V2+P0+P2) % 2 => P2=1

C2 =0 = (V0+P0+P1) % 2 => P1=1

In general the parity check matrix is very large in length unlike the example shown in the figure. In our encoder implementation message bit size is 58320 and parity bit size is 6480. This implies the parity check matrix is of size 6480x64800.The LDPC encoder sends only parity bits to the decoder and this is how the compression is achieved.

## 1.3. Wyner-Ziv Decoder

The decoding of Wyner-Ziv stream involves, decoding of key frames and WZ frames. This is illustrated in Figure 1.14. The key frames in our thesis are encoded as H.264 Intra frames. At the decoder side these are decoded as per H.264 I-frame decoding. This is briefly explained in section 1.31.

The reconstructed key frames are used for decoding of WZ frames. In the figure, the WZ frame to be decoded is represented as $W_k$. The neighboring key frames $I_k$ and $I_{k+1}$ are used to generate a prediction for Wk. This prediction is termed as side information (SI). The previous key frame $I_k$ is subtracted from the SI frame to produce error frame $X_{er}$ which is subsequently quantized to form $Q_{er}$. The LDPC decoder uses the WZ parity bit stream sent by the encoder to correct the bit errors in $Q_{er}$. This corrected error frame $Q'_{er}$ is then inverse quantized and added to the key frame $I_k$ to obtain the WZ frame $W_k$.

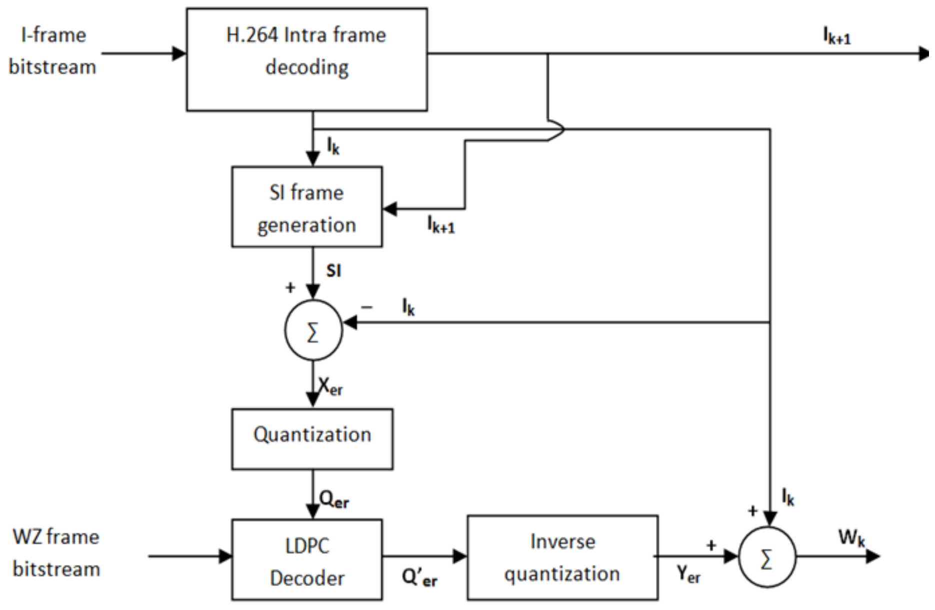

Figure 1.14 Wyner-Ziv decoder[1]

### *1.3.1 Key frame decoding*

The input for key frame decoder is the Intra frame bit stream generated by the encoder. The decoding is illustrated in Figure 1.15. The intra frame bit stream consists of entropy coded transform domain residuals for each block. This is decoded using entropy decoder and then inverse quantization is applied. The Inverse Integer transform is applied on this residual block to obtain special domain residual block. The intra prediction is done for the block using neighboring blocks and then the prediction block is added to decoded residual block to obtain reconstruction block. This way all the blocks in the frame decoded to form a reconstructed frame.

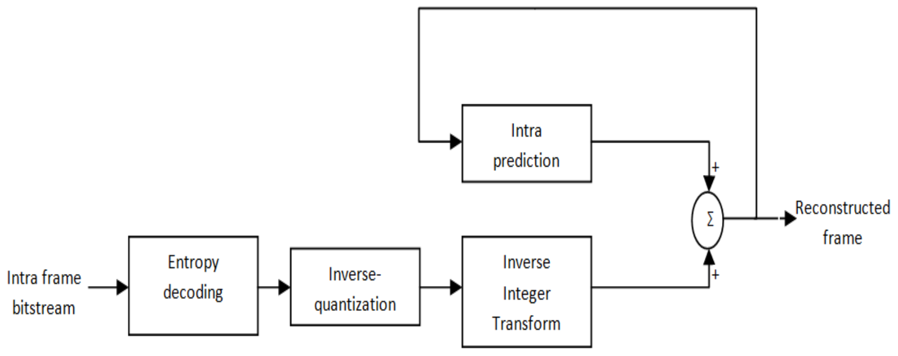

Figure 1.15 I-frame decoding in H.264

#### 1.3.1.1 Intra prediction

The intra prediction at the decoder side is same as encoder side as explained in the section 1.2.1.1 except that it need not be done for all the modes. The encoder sends the prediction mode in the bit stream used along with the residuals. Hence the prediction is done based only one mode. The prediction uses neighboring blocks which are previously decoded.

### 1.3.1.2 Entropy decoding

Here only CAVLC decoder is used since at the H.264 encoder used is a baseline encoder. The CAVLC decoder performs a reverse operation as compared to that of encoder. It should decode code-words and obtain the indices. The indices are then mapped to a value based on the type of parameter encoded. The parameter can be number of non-zero levels in a block or could be the levels itself.

The index can be obtained as [6][7][8][19],

*index = 2^K+ int(DATA)- 1*

where *int (DATA)* is the integer corresponding to the binary string. The parameters K and DATA are related to the codeword as follows,

*code_word = [K zeros][1][K-bit DATA]*

The table 1.1. illustrates decoding of codeword into index.

Table 1.2 First few Exp-Golomb codes and the corresponding indexes

| Code word | K | 2^K | DATA | int(DATA) | 2^K + int(DATA) - 1 | Index |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | | 0 | 0 | 0 |
| 010 | 1 | 2 | 0 | 0 | 1 | 1 |
| 011 | 1 | 2 | 1 | 1 | 2 | 2 |
| 00100 | 2 | 4 | 00 | 0 | 3 | 3 |
| 00101 | 2 | 4 | 01 | 1 | 4 | 4 |
| 00110 | 2 | 4 | 10 | 2 | 5 | 5 |
| 00111 | 2 | 4 | 11 | 3 | 6 | 6 |
| 0001000 | 3 | 8 | 000 | 0 | 7 | 7 |
| 0001001 | 3 | 8 | 001 | 1 | 8 | 8 |

The mapping of index to a value to be encoded is not fixed one instead it is context adaptive. There are twelve such mappings that standard specifies and selection among twelve is based on local statistics of the stream. The appropriate table is based on the context and the values are decoded accordingly.

### 1.3.1.3 Inverse Scanning and Run-length

The output of entropy decoder is an array of level-run pairs. The run indicates the run of zero valued quantized coefficients. The run-length decoding is applied to obtain the scanned position of each level in 4x4 block. The inverse scanning is applied on the position to obtain the actual position of levels in 4x4 block. The inverse scanning is done as per the table 1.2.

Table 1.3 The inverse scanning index table

| Input Index | Output Index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 4 |
| 3 | 8 |
| 4 | 5 |
| 5 | 2 |
| 6 | 3 |
| 7 | 6 |
| 8 | 9 |
| 9 | 12 |
| 10 | 13 |
| 11 | 10 |
| 12 | 7 |
| 13 | 11 |
| 14 | 14 |
| 15 | 15 |

### 1.3.1.4 Inverse Quantization

The inverse quantization is a simple multiplication by Qstep and a scaling matrix. The scaling matrix used at the decoder side is inverse of scaling matrix (SF) used at the encoder side. The Inverse Integer Transform is applied on this block to obtain spatial domain block. The quantization can be written as [7][21],

$$X = \text{Qstep} * Y \otimes \text{InvSF}$$

Where Y is quantized block and X is output of the inverse quantization, the symbol $\otimes$ denotes the element by element multiplication of the corresponding matrices and InvSF is matrix with elements of matrix SF inversed.

1.3.1.5 Inverse Integer transform

This is a 2-D transform done using the Hadamard inverse transformation matrix ($\overline{H}$) given below [6],

$$\overline{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

The inverse transformation using $\overline{H}$ is done as,

$$F = \overline{H}^T X \overline{H}$$

*1.3.2 WZ frame decoding*

The overall WZ frame decoding is discussed in section 1.3 and in this section some of the processes involved in WZ frame decoding are discussed.

1.3.2.1 SI generation

The SI generation is one of the core steps in WZ frame decoding and most of the decoder complexity is attributed to this step. This has significant effect on the performance of the decoder in terms of quality. This thesis focuses mainly on SI and it is discussed in detail in chapter 2.

1.3.2.2 Quantization

The quantization helps in reducing the range of values and hence the compression can be achieved. In this thesis a simple rounding division by a quantizer is considered. The quantization is done as follows,

$$Q_{er} = Floor((X_{er} + quantizer/2)/quantizer)$$

Where $X_{er}$ is the value to be quantized and $Q_{er}$ is the output of quantization.

### 1.3.2.3 LDPC decoding

LDPC decoding is done via belief propagation algorithm [25]. This is an iterative decoding algorithm and complexity of the algorithm depends on number of iterations. The belief propagation algorithm essentially involves,

1. Propagation of probability of a bit at the variable node being '0' or '1' to all check nodes
2. Response from the check nodes to a variable nodes which indicates the probability of bit being '0' or '1' at that variable node.

In Figure 1.16, $q_{ij}(b)$ indicates the belief propagation from variable node $c_i$ to check node $f_j$ . For example $q_{ij}(0)$ indicates the probability that '0' being sent from variable node $c_i$ to check node $f_j$ and $q_{ij}(1)$ indicates the probability that '1' being sent from variable node $c_i$ to check node $f_j$. Similarly $r_{ji}(b)$ indicates the response from check node $f_j$ to variable node $c_i$.

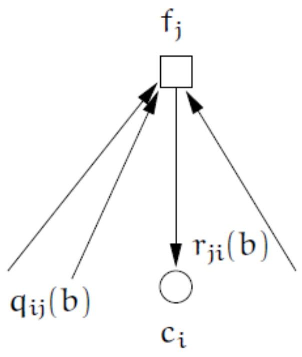

Figure 1.16 Belief propagation in LDPC decoder

The following notations are used for describing LDPC decoding using belief propagation algorithm:

| | |
|---|---|
| $y_i$ : | represents the bit value at variable node ci for the current iteration. |
| $h_{ji}$ : | represents the element of the parity check matrix. |
| $q_{ij}$ : | messages to be passed from bit node $c_i$ to check nodes $f_j$ |

$r_{ji}$ : messages to be passed from check node $f_j$ to bit node $c_i$.

$R_j = \{ i : h_{ji} = 1\}$ the set of column locations in the jth row having '1'

$R_{j/i} = \{ i' : h_{ji'} = 1\}\backslash\{i\}$ the set of column locations excluding ith column in the jth row having '1'.

$C_i = \{ j : h_{ji} = 1\}$ the set of row locations in the ith column having '1'

$C_{i/j} = \{ j' : h_{j'i} = 1\}\backslash\{j\}$ the set of row locations excluding jth row in the ith column having 1'.

The LDPC decoding involves following steps [25]:

1. Initialize $q_{ij}(b)$ as follows,

$$p_i = \Pr(c_i = 1/y_i)$$

$$p_i = \begin{cases} \dfrac{1}{1+e^{-2/\sigma^2}}, & y_i = 1 \\ \dfrac{1}{1+e^{2/\sigma^2}}, & y_i = 0 \end{cases}$$

$$q_{ij}(1) = p_i$$
$$q_{ij}(0) = 1 - p_i$$

Where, $\sigma^2$ is the variance of the noise.

2. Update $r_{ji}(b)$ as follows,

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in R_{j/i}} [1 - 2q_{i'j}(1)]$$

$$r_{ji}(1) = 1 - r_{ji}(0)$$

3. Update $q_{ij}(b)$ as follows,

$$q_{ij}(0) = K_{ij}(1-p_i)\prod_{j' \in C_{i/j}} r_{j'i}(0)$$

$$q_{ij}(1) = K_{ij}p_i\prod_{j' \in C_{i/j}} r_{j'i}(1)$$

Where $K_{ij}$ is a constant to be calculated such that $q_{ij}(0) + q_{ij}(1) = 1$

4. Calculate $Q_i(0)$ : Probability that $C_i$ is '0'

$Q_i(1)$ : Probability that $C_i$ is ‘1’

$$Q_i(0) = K_i(1 - p_i)\prod_{j \in C_i} r_{ji}(0)$$

$$Q_i(1) = K_i p_i \prod_{j \in C_i} r_{ji}(1)$$

5. Calculate new $C_i$ as follows:

$$C_i = \begin{cases} 1, & Q_i(1) < Q_i(0) \\ 0, & else \end{cases}$$

6. If with the new C vector $CH^T$ is zero where H is the parity check matrix or if the number of iterations exceed the maximum limit then stop or else steps 1-6 are iterated again.

### 1.3.2.4 Inverse quantization

The Inverse quantization helps to restore the values to original range. This is the reversal of quantization process and is done as follows,

$$Y_{er} = Q'_{er} * \text{quantizer}$$

### 1.3.2.5 Reconstruction

Reconstruction is generating actual frame by adding error frame to the previous key frame. This is the reverse process for residual generation at the encoder side. The output frame may exceed the dynamic range(0, 255) of the video frame and needs to be clipped. This is done as follows,

$$W_k = MIN(MAX(Y_{er} + I_{k,} 0), 255)$$

Where $Y_k$ is the reconstructed error frame, $I_k$ is previous key frame and $W_k$ is the reconstructed WZ frame.

# CHAPTER 2

# SIDE INFORMATION GENERATION

## 2.1. Forward and Backward Motion Estimation

The SI frame generation is key aspect of WZ decoding process. The quality of the decoding is dependent on the SI frame and in terms of complexity this is a major component of WZ decoder.

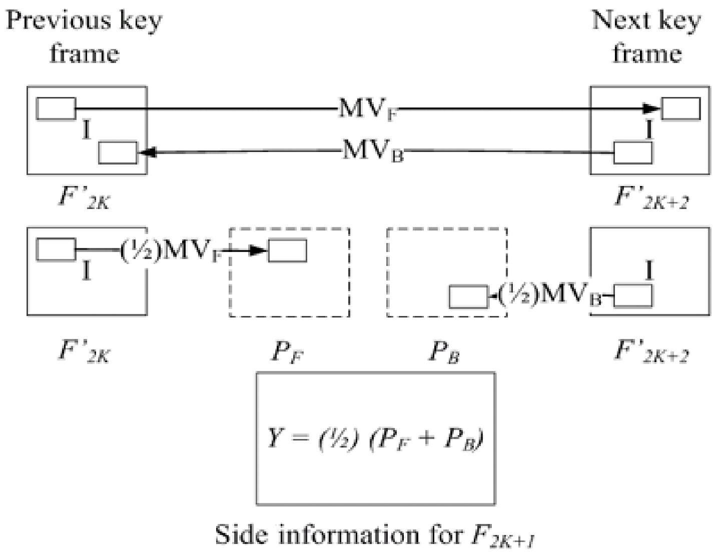

Figure 2.1 Side information generation using key frames [1]

The generation of SI is illustrated in Figure 2.1 and it involves,

1. Motion estimation (ME) between two key frames to obtain motion vectors (MV): The estimation is done in both forward and backward directions to obtain MVF and MVB respectively as shown in the figure. The block sizes used for ME are 16x16, 8x8 and 4x4.

2. The derivation of motion vectors for WZ frames: This is done by scaling MVs obtained in the previous step by the ratio calculated as distance between WZ frame to previous key frame to the distance between key frames themselves. In the figure, the scaling factor is ½ since the ratio of distance between key frame and WZ frame to distance between two key frames is ½.

3. Obtaining the estimation for block of a WZ frame: This is done by interpolation of blocks from the previous and next key frames. The motion vectors calculated in the previous step are used for the interpolation. The forward predicted frame (PF) is obtained using forward motion vector MVF and backward predicted frame (PB) is obtained using backward motion vector MVB. Then the side information frame Y is obtained as (PF + PB)/2.

The motion estimation for future key frame is done with respect to previous key frame followed by vice versa operation yielding two motion vectors namely forward (MVF) and backward (MVB) prediction MVs for a block. It is not necessary that the previous and future frames have to be key frames but any available decoded frames can be used for this purpose. For example consider the case where key frame distance is four then the decoding of frames is done as shown in Figure 2.2. Here S2 is decoded first using side information S2SI generated using reconstructed key frames K1R and K2 R. This is followed by S1 for which K1R and S2R are used for generating side information S1SI. Finally S3 is decoded for which side information S3SI is generated using S2R and K2R reconstructed frames.

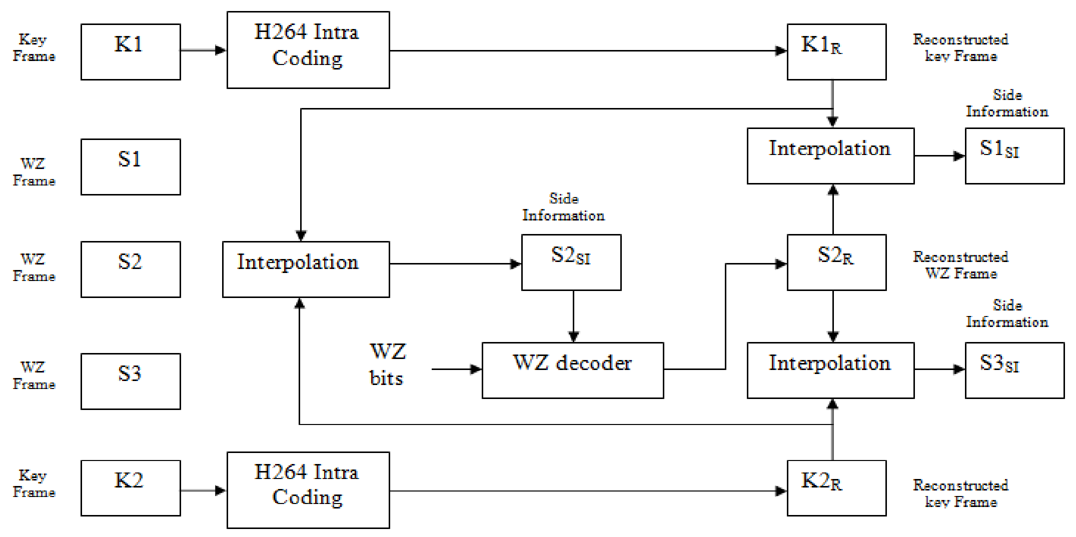

Figure 2.2 Side information generation using key frames [2]

The forward and backward motion vectors of the current frame are obtained by appropriately scaling and changing the signs of forward and backward prediction MVs obtained earlier. For the case of key frame distance shown in the figure, the operations for generating MVs can be summarized as,

1. Obtaining MVs for S2: Perform ME between K2(frame under consideration) and K1 (reference) to obtain a set of forward MV based on SAD. Similarly perform ME between K1(frame under consideration) and K2 (reference) to obtain a set of backward MV. Now for each block in K2 with a given forward MV calculate the position of same block in S2. This position is calculated by shifting the position of block in K2 by ½ * forward MV. Also the forward MV for this block in S2 will be ½ * forward MV. Similarly for each block in K1 with a given backward MV calculate the position of same block in S2. This position is calculated by shifting the position of block in K1 by ½ * backward MV. Also the backward MV for this block in S2 will be ½ * backward MV.

2. Obtaining MVs for S1: Operations involved in the calculation of forward and backward MV for S1 are similar to those of S2 with the exceptions that prediction MVs are generated using K1 and S2.

3. Obtaining MVs for S3: Here prediction MVs are generated using S2 and K2.

## 2.2 Sub-pixel motion estimation for SI generation

The side information generated can be improved by using sub-pixel motion vectors for both forward and backward predictions. In order to derive these sub-pixel positions interpolation between pixels needs to be performed. For half pixel motion estimation there are three pixel positions that need to be evaluated. For quarter pixel motion estimation there are twelve pixel positions that need to be evaluated. The generation of sub-pixel positions is done as per H.264 standard [6][7][8][19] and is briefly described below.

### *2.2.1 Half-pixel positions*

The position of half pixels is shown along with full pixels in Figure 2.3. The pixel positions numbered H33, G33 and D33 are half pixel positions and need to be derived. These are generated by interpolating full pixel or half pixel values using a six tap filter [1 -5 20 20 -5 1]/32. Following equations can be used [6],

1. H33 = [F13 + -5 * F23 + 20 * F33 + 20 * F43 + -5 * F53 + F63 + 15] >> 5
2. G33 = [F31 + -5 * F32 + 20 * F33 + 20 * F34 + -5 * F35 + F36 + 15] >> 5
3. D33 = [H31 + -5 * H32 + 20 * H33 + 20 * H34 + -5 * H35 + H36 + 15] >> 5

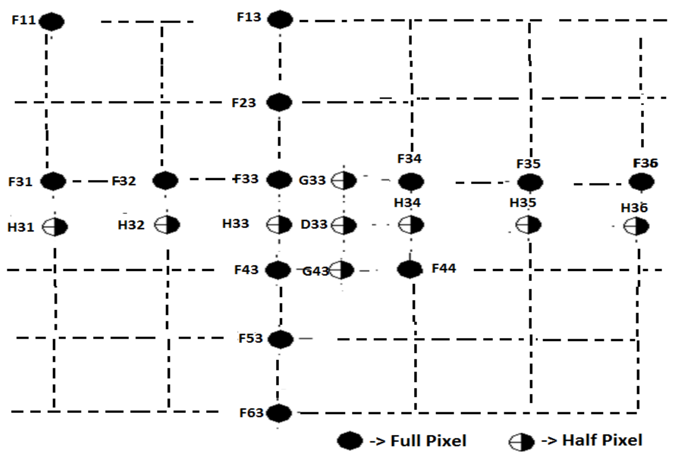

Figure 2.3 Full and half pixel positions

### *2.2.2 Quarter-pixel positions*

The quarter pixels are obtained by averaging nearest full pixel or half pixel positions. The following equations are used for obtaining quarter pixel positions [6],

q1 = ( F33 + G33 + 1 ) >> 1

q2 = ( G33 + F34 + 1 ) >> 1

q3 = ( F33 + H33 + 1 ) >> 1

q4 = ( H33 + G33 + 1 ) >> 1

q5 = (G33 + D33 + 1 ) >> 1

q6 = ( G33 + H34 + 1 ) >> 1

q7 = ( H33 + D33 + 1 ) >> 1

q8 = ( D33 + H34 + 1 ) >> 1

q9 = ( H33 + F43 + 1 ) >> 1

q10 = ( H33 + G43 + 1 ) >> 1

q11 = ( D33 + G43 + 1 ) >> 1

q12 = ( G43 + H34 + 1 ) >> 1

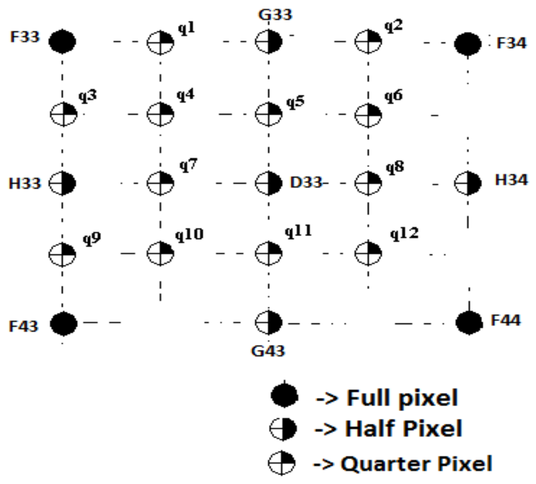

Figure 2.4 Full, half and quarter pixel positions

The forward and backward predicted data obtained for each partition block is averaged to obtain the final prediction block. In case for a block if there is no motion vector, then intra prediction can be used to predict the block from neighboring pixels. The improvement in the quality of SI generated with the sub pixel motion estimation over full pixel motion estimation can be measured both visually and quantitatively [3]. The quantitative measurement can be done by PSNR of the predicted frame with reference to the original frame. The objective is to get a good improvement in the quality of SI frame.

## 2.3 Motion Compensation (MC)

Motion compensation involves obtaining the forward and backward prediction blocks for all the blocks in current frame using the motion vectors available for that block. Depending on the motion vectors available for a block there are four possibilities of MC.

4. FORW_MV is available: Forward motion compensation is done to obtain forward predicted frame (PF).
5. BACK_MV is available: Backward motion compensation is done to obtain backward predicted frame (PB).

6. FORW_MV and BACK_MV are available: Both forward and backward motion compensations are done.

7. No MV exists: No motion compensation is done for the block. This block is candidate for intra prediction which is discussed in later sections.

Interpolation filters and the steps used to obtain the prediction block for sub-pel MVs are described below.

### *2.3.1 Luma motion compensation*

In section 2.1 the motion vectors are derived as FORW_PRED_MV / 2 and BACK_PRED_MV / 2. Therefore,

Luma forward MV = [ FORW_PRED_MVx / 2, FORW_PRED_MVy / 2]

Luma backward MV = [ BACK_PRED_MVx / 2, BACK_PRED_MVy / 2]

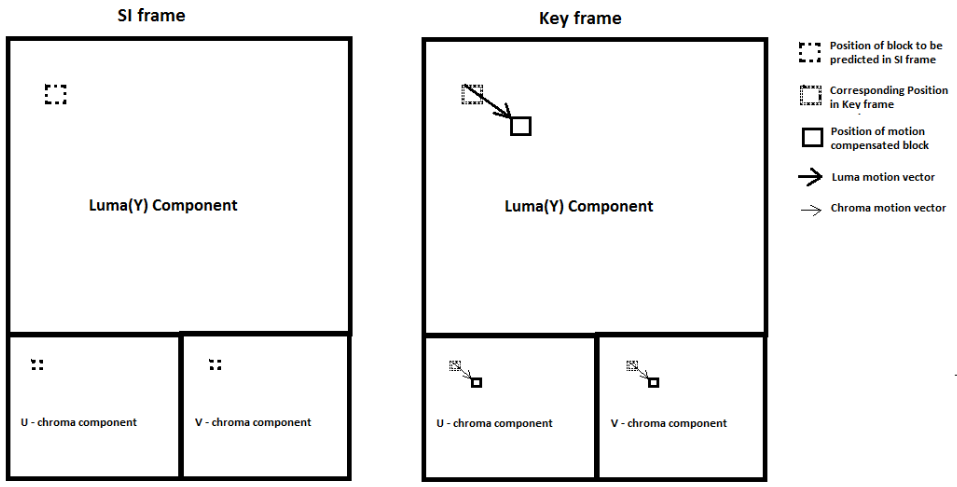

Figure 2.5 Deriving position of motion compensated block

The derivation of position of motion compensated block using these motion vectors is shown in Figure 2.5. Luma motion compensation uses these motion vectors and generates a corresponding block. The method used for interpolation is same as that used in motion estimation which is described in section 2.2. The motion compensation has to be performed individually for both forward and backward reference frames.

### *2.3.2 Chroma motion compensation*

The chroma motion compensation needs bilinear interpolation. The four neighboring pixels needed for the interpolation are obtained using luma motion vectors. The four pixels are named A, B, C and D in the Figure 2.6 and belong to the reference frame. In this thesis only YUV 420 frame format is considered where chroma frame is subsampled by 2 compared to luma. Hence the chroma motion vectors are derived of luma motion vectors as,

Chroma forward MV = [ FORW_PRED_MVx / 4, FORW_PRED_MVy / 4]

Chroma backward MV = [ FORW_PRED_MVx / 4, FORW_PRED_MVy / 4]

The motion compensation has to be performed for each block in forward and backward reference frames individually. Additionally the motion compensation for U and V components in the chroma are done independently but use the same motion vector. The position (Ax, Ay) of pixel A for block having position (chroma_blk_x, chroma_blk_y) can be derived as shown in the figure.

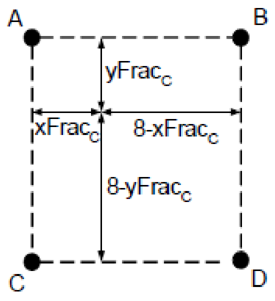

Figure 2.6 Chroma motion compensation [22]

The luma motion vectors are represented in quarter pixel units. Hence the chroma motion vector will be in 1/8 of pixel units. Hence to obtain position of A which is in full pixel units motion vector has to be divided by 8. Hence Ax and Ay can be obtained as,

Ax = chroma_blk_x + floor(Chroma forward MVx / 8)

$$Ay = chroma\_blk\_y + floor(\text{Chroma forward MVy} / 8)$$

The rest of the pixel positions can be obtained as,

$$Bx = Ax + 1$$

$$By = Ay$$

$$Cx = Ax$$

$$Cy = Ay + 1$$

$$Dx = Ax + 1$$

$$Dy = Ay + 1$$

The fractional components of chroma motion vector can be obtained as,

$$xFracc = \text{Chroma forward MVx} \ \& \ 7$$

$$yFracc = \text{Chroma forward MVy} \ \& \ 7$$

The bilinear interpolation of four pixels to obtain the predicted pixel is done as follows,

$$\begin{aligned}\text{Predicted Pixel} = \; & ( ( 8 - xFrac_c) * ( 8 - yFrac_c) * A \; + \\ & xFrac_c * ( 8 - yFrac_c) * B \; + \\ & ( 8 - xFrac_c) * yFrac_c * C \; + \\ & xFrac_c * yFrac_c * D \; + 32 ) >> 6\end{aligned}$$

Similarly all the pixels in the block can be generated using the neighboring four pixels as explained above. The process is same for both U and V chroma components.

## 2.4 Bidirectional Interpolation

The final prediction for a block with two motion vectors is obtained by the interpolation of forward and backward prediction blocks obtained in previous step. The method used here is simple average with rounding.

$$\text{SI frame} = (\text{Forward prediction Frame} + \text{Backward Prediction frame} + 1)/2$$

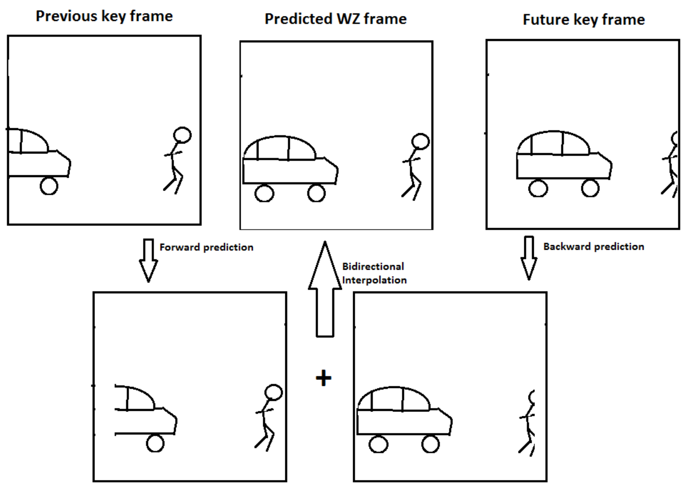

Figure 2.7 Bidirectional interpolation

## 2.5 Intra prediction

Intra prediction is used when a block does not contain a motion vector. Intra prediction is done as defined in H.264 standard [6] using the neighboring blocks in the same predicted frame. There are nine different directional prediction modes as discussed in the section 1.2.1.A. The prediction block is generated using one of the direction modes shown in Figure 1.6.

### *2.5.1 Derivation of Intra prediction modes*

The direction mode to be used for the block can be derived from the direction mode of the co located block in the previous key frame. The co-located block means that the offset of the block in the previous frame from the start of the frame is same as that of the current block to be predicted from its start of frame.

The block size used for intra prediction is fixed at 4x4 irrespective of the Intra prediction block size of the co-located block in the previous frame. The care should be taken to derive the directional mode when the current block and the co-located block in the key frame have different block sizes. Figure 2.8 shows the derivation of directional mode when both the blocks have the same size.

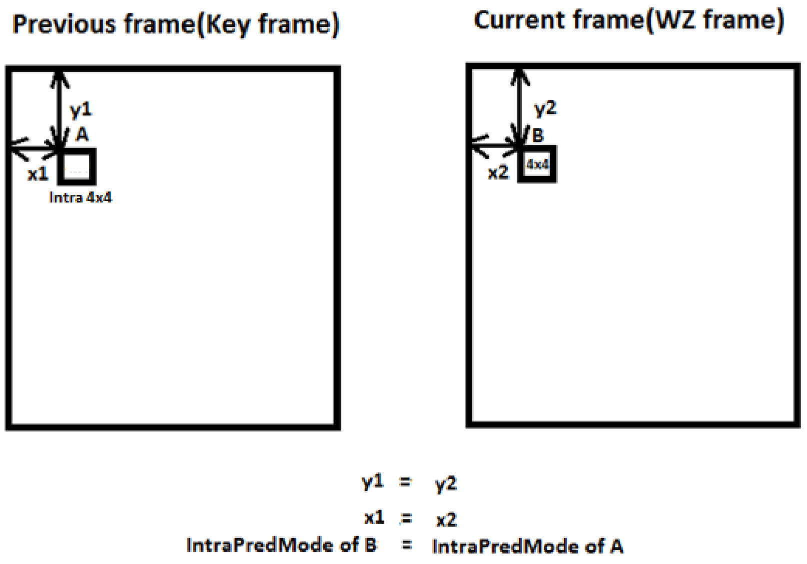
Previous frame(Key frame)
Current frame(WZ frame)
y1
A
x1
Intra 4x4
y2
B
x2
4x4
y1 = y2
x1 = x2
IntraPredMode of B = IntraPredMode of A

Figure 2.8 Deriving intra prediction mode when co-located block is intra 4x4

In Figure 2.9 the co-located block in the key frame is coded using Intra-16x16 mode. This implies it has only one direction mode for the entire 16x16 block. In this case the 16x16 block in the key frame has same offset as that of 4x4 block in the WZ frame. Here the directional mode for 4x4 block used is same as that of 16x16 block as shown in the figure.

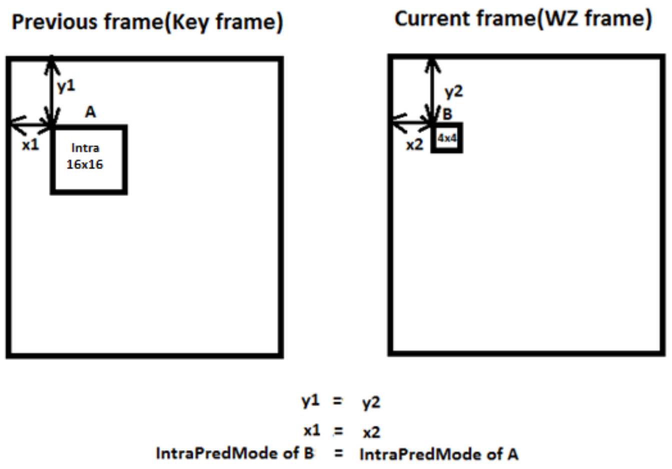
Previous frame(Key frame)
Current frame(WZ frame)
y1
A
x1
Intra 16x16
y2
B
x2
4x4
y1 = y2
x1 = x2
IntraPredMode of B = IntraPredMode of A

Figure 2.9 Deriving intra prediction mode when co-located block is intra 16x16 aligned

In Figure 2.10 the co-located block in the key frame is coded using Intra-16x16 mode. But it has different offset compared to 4x4 block in the WZ frame. But when two frames overlapped the 16x16 block encompasses the 4x4 block hence the directional mode used for 4x4 block is same as that of 16x16 block.

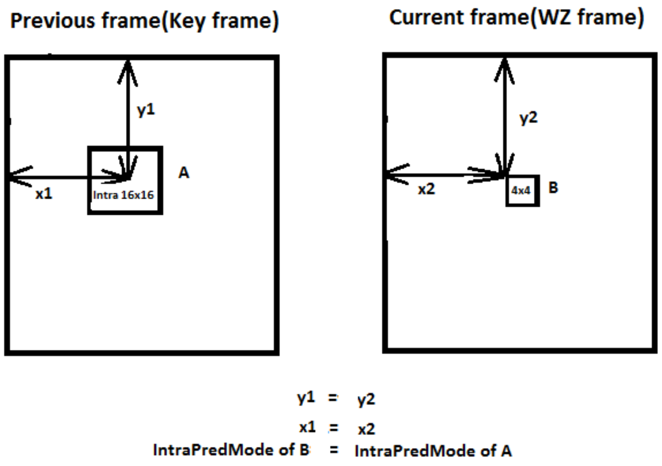
Previous frame(Key frame)
Current frame(WZ frame)
y1
x1
Intra 16x16
A
y2
x2
4x4
B
y1 = y2
x1 = x2
IntraPredMode of B = IntraPredMode of A

Figure 2.10 Deriving intra prediction mode when co-located block is intra 16x16 non-aligned

# CHAPTER 3

# SOFTWARE IMPLEMENTATION

The testing of various schemes for SI generation requires implementation of both Wyner-Ziv encoder and decoder. The implementation is done in ‘C’ and the Microsoft Visual Studio is used for the development and testing. The encoding and decoding of key frames are kept the same as that in H.264 standard and for this case the relevant part of JM reference software [27] is used. JM software is freely available without any license or royalty regulations. This is implemented by JVT and contains both encoder and decoder implementations for H.264. The exact details of how this software is incorporated in Wyner-Ziv encoder and decoder implementation is described in detail in further sections.

## 3.1 Encoder Implementation

Wyner-Ziv encoder implementation is divided into two parts. In the first part key frame encoding is done which is based on H.264 intra frame encoding. The second part is encoding WZ frame which involves implementation of residual generation and LDPC encoding. The frames numbered 0, 2, 4, 6…etc are encoded as key frames and rest of the frames are encoded as WZ frames.

### *3.1.1 Key frame*

The key frame is encoded as I-frame in H.264 and this is done using JM software [27]. The source code for I- frame encoding is available as part of JM encoder. This is used without any modifications. The bit stream generated for all the I-frames is stored in a file for decoding later. All the tool sets available in JM encoder for I-frame encoding are enabled.

### *3.1.2 WZ frame*

The WZ frame encoding is implemented in two phases. The first phase is residual generation and quantization. The residual generation is obtained by subtracting the reconstructed key frame from the WZ frame. A buffer is maintained which stores a recent reconstructed key frame which is used in residual generation. The quantization is implemented using addition and shifts as the quantizer used is multiple of 2.

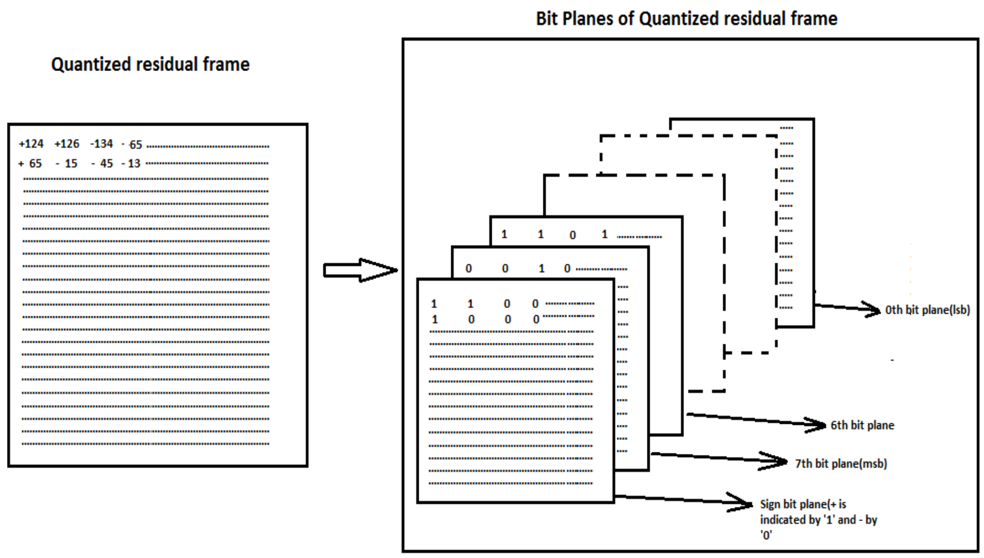

Figure 3.1 Bit plane extraction of residual frame

In the second phase LDPC encoding is implemented and this is done by Dr. Kim Jin Soo [4]. The 8 bit-planes and the sign bit plane for the entire residual frame are extracted. The bit-planes are appended one after another to form a stream of bits which is fed to LDPC encoder. The encoder groups bits into a block of 58320 bits and produces output codeword with 58320 original bits and 6480 parity. The original bits from codeword are discarded and the parity bits are stored in a file which will be later used by the decoder.

## 3.2 Decoder Implementation

The encoder produces output in two files one is the intra frame bit stream file and the second is WZ frame bit stream file. The decoding of key frames is implemented first and then the decoding of WZ bit stream is added which is more complex in nature.

### *3.2.1 Key frame*

The key frames are encoded as Intra frame and the decoding is done using JM software [27]. JM software contains source code for intra frame decoding and it is used as is without any modification. A buffer is maintained which stores two recent reconstructed frames which are required for WZ frame decoding.

### *3.2.2 WZ frame*

The first step in WZ decoding is generation of SI frame. The reconstructed key frames stored in the buffer are used for this. The generation of SI frame requires motion estimation. This is done by using full-search ME method where each point in the search range is visited and the best point is the point with lowest SAD. The motion estimation is block based and the various block sizes are considered. The SI frame is then quantized and fed to LDPC decoder. The parity bit stream generated by the encoder is used by LDPC to recursively correct errors in the SI frame. The SI frame is then Inverse quantized and the WZ frame is reconstructed by adding SI frame to key frame. The LDPC decoding is implemented by Dr.Kim Jin Soo[4]. The rest of the development for WZ frame decoding is done independently.

## 3.3 Comparison of complexity

The Wyner-Ziv encoder implemented has very low complexity. It involves residual generation, quantization, LDPC decoder. In comparison H.264 encoder has motion estimation, transform, quantization, de-blocking and context adaptive binary arithmetic coding (CABAC). The motion estimation and CABAC contribute significant parts in H.264 encoder complexity which are not required in Wyner-Ziv encoder.

The Wyner-Ziv decoder requires motion estimation, motion compensation, quantization and inverse quantization and LDPC decoding. In comparison H.264 decoder uses CABAC decoding, inverse transform, inverse quantization, de-blocking and motion compensation. The motion estimation at the decoder side increases complexity of Wyner-Ziv decoder but this is expected from the point of view of applications of Wyner-Ziv coding.

# CHAPTER 4

# TESTING OF WYNER-ZIV CODING

The testing of Wyner-Ziv coding involves generating H.264 reference stream and Wyner-Ziv stream. The streams are then decoded and the reconstructed video sequences are then compared against each other using video quality metrics. The quality metrics used are PSNR and SSIM. The quality metrics used are described in appendices A and B.

The visual quality of WZ reconstructed sequence is dependent on wide variety of parameters. It depends on the input test sequence is used for comparison. It also depends on the encoder parameters like bit rate, ME search range, ME block size, quantizer etc. Both the reference stream and test streams are generated by varying these parameters and compared with each other.

## 4.1 Generation of H.264 reconstructed sequence

The first step in generating reconstructed YUV sequence is to encode the original YUV sequence itself. The original YUV sequence is encoded using JM encoder [27] to obtain the reference bit stream . The encoder parameters used for I frame encoding are kept the same for both H.264 reference stream generation and Wyner-Ziv key bit stream generation. The table 4.1 lists the parameters used for Intra frame encoding. The P frame encoding is done with an ME search range of 16 (Figure 5.14) and the rest of the parameters used are shown in the table 4.2.

Table 4.1 Parameters used in H.264 for I frame encoding

| Parameter | Value | Remarks |
|---|---|---|
| FrameRate | 30 | Frame Rate per second (1-100) |
| ProfileIDC | 66 | Profile IDC (66   baseline, 77   main, 88   extended) |
| LevelIDC | 30 | Level IDC   (e.g. 20     level 2.0) |
| IntraPeriod | 1 | Period of I-Frames (0   only first) |
| IDRIntraEnable | 0 | Force IDR Intra  (0   disable 1   enable) |
| QPFirstFrame | 28 | Quant. param for first frame (intra) (0-51) |
| QPRemainingFrame | 28 | Quant. param for remaining frames (0-51) |
| FrameSkip | 1 | Number of frames to be skipped in input (e.g 2 will code every third frame) |
| ChromaQPOffset | 0 | Chroma QP offset (-51..51). |
| UseHadamard | 1 | Hadamard transform (0   not used, 1   used) |
| MbLineIntraUpdate | 0 | Error robustness(extra intra macro block updates)(0   off, N: One GOB every N frames are intra coded) |
| RandomIntraMBRefresh | 0 | Forced intra MBs per picture |
| LoopFilterParametersFlag | 0 | Configure loop filter (0   parameter below ingored, 1   parameters sent) |
| LoopFilterDisable | 0 | Disable loop filter in slice header (0   Filter, 1   No Filter) |
| LoopFilterAlphaC0Offset | 0 | Alpha & C0 offset div. 2, {-6, -5, ... 0, +1, .. +6} |
| LoopFilterBetaOffset | 0 | Beta offset div. 2, {-6, -5, ... 0, +1, .. +6} |
| RestrictSearchRange | 2 | restriction for (0: blocks and ref, 1: ref, 2: no restrictions) |
| RDOptimization | 1 | rd-optimized mode decision (0:off, 1:on, 2: with losses) |
| LossRateA | 10 | expected packet loss rate of the channel for the first partition, only valid if RDOptimization     2 |
| LossRateB | 0 | expected packet loss rate of the channel for the second partition, only valid if RDOptimization     2 |
| LossRateC | 0 | expected packet loss rate of the channel for the third partition, only valid if RDOptimization     2 |
| NumberOfDecoders | 30 | Numbers of decoders used to simulate the channel, only valid if RDOptimization     2 |
| RestrictRefFrames | 0 | Doesnt allow reference to areas that have been intra updated in a later frame. |

Table 4.2 Parameters used in H.264 for P frame encoding

| Parameter | Value | Remarks |
|---|---|---|
| FrameRate | 30 | Frame Rate per second (1-100) |
| ProfileIDC | 66 | Profile IDC (66   baseline, 77   main, 88   extended) |
| LevelIDC | 30 | Level IDC   (e.g. 20     level 2.0) |
| IntraPeriod | 1 | Period of I-Frames (0   only first) |
| IDRIntraEnable | 0 | Force IDR Intra  (0   disable 1   enable) |
| QPFirstFrame | 28 | Quant. param for first frame (intra) (0-51) |
| QPRemainingFrame | 28 | Quant. param for remaining frames (0-51) |
| FrameSkip | 1 | Number of frames to be skipped in input (e.g 2 will code every third frame) |
| ChromaQPOffset | 0 | Chroma QP offset (-51..51). |
| UseHadamard | 1 | Hadamard transform (0   not used, 1   used) |
| SearchRange | 16 | Max search range |
| NumberReferenceFrames | 10 | Number of previous frames used for inter motion search (1-16) |
| PList0References | 0 | P slice List 0 reference override (0 disable, N <    NumberReferenceFrames) |
| MbLineIntraUpdate | 0 | Error robustness(extra intra macro block updates) (0   off, N: One GOB every N frames are intra coded) |
| RandomIntraMBRefresh | 0 | Forced intra MBs per picture |
| InterSearch16x16 | 1 | Inter block search 16x16 (0   disable, 1   enable) |
| InterSearch16x8 | 1 | Inter block search 16x8  (0   disable, 1   enable) |
| InterSearch8x16 | 1 | Inter block search  8x16 (0   disable, 1   enable) |
| InterSearch8x8 | 1 | Inter block search  8x8  (0   disable, 1   enable) |
| InterSearch8x4 | 1 | Inter block search  8x4  (0   disable, 1   enable) |
| InterSearch4x8 | 1 | Inter block search  4x8  (0   disable, 1   enable) |
| InterSearch4x4 | 1 | Inter block search  4x4  (0   disable, 1   enable) |
| UseFME | 0 | Use fast motion estimation (0   disable, 1   enable) |
| NumberBFrames | 0 | Number of B frames inserted (0   not used) |
| QPBPicture | 30 | Quant. param for B frames (0-51) |
| DirectModeType | 0 | Direct Mode Type (0:Temporal 1:Spatial) |
| DirectInferenceFlag | 0 | Direct Inference Flag (0: Disable 1: Enable) |
| BList0References | 0 | B slice List 0 reference override (0 disable, N <    NumberReferenceFrames) |
| BList1References | 0 | B slice List 1 reference override (0 disable, N <    NumberReferenceFrames) |
| StoredBPictures | 0 | Stored B pictures (0   off, 1   on) |
| LoopFilterParametersFlag | 0 | Configure loop filter (0   parameter below ingored, 1   parameters sent) |
| LoopFilterDisable | 0 | Disable loop filter in slice header (0   Filter, 1   No Filter) |
| LoopFilterAlphaC0Offset | 0 | Alpha & C0 offset div. 2, {-6, -5, ... 0, +1, .. +6} |
| LoopFilterBetaOffset | 0 | Beta offset div. 2, {-6, -5, ... 0, +1, .. +6} |
| RestrictSearchRange | 2 | restriction for (0: blocks and ref, 1: ref, 2: no restrictions) |
| RDOptimization | 1 | rd-optimized mode decision (0:off, 1:on, 2: with losses) |
| LossRateA | 10 | expected packet loss rate of the channel for the first partition, only valid if RDOptimization     2 |
| LossRateB | 0 | expected packet loss rate of the channel for the second partition, only valid if RDOptimization     2 |
| LossRateC | 0 | expected packet loss rate of the channel for the third partition, only valid if RDOptimization     2 |
| NumberOfDecoders | 30 | Numbers of decoders used to simulate the channel, only valid if RDOptimization     2 |
| RestrictRefFrames | 0 | Doesnt allow reference to areas that have been intra updated in a later frame. |

## 4.2 Generation of Key and WZ reconstructed frames

The Wyner-Ziv stream is generated with Key frame parameters same as Intra-frame parameters in reference stream. The parameters used for key frame encoding are same as listed in table 4.1. The WZ frames are encoded with quantizer, key frame distance and bit rate as variables. The encoding bit rate is varied by varying the size of parity bits. At the decoder side, ME search range, ME block size and SI generation scheme can be changed. The H.264 reconstructed sequence and Wyner-Ziv reconstructed sequence are then compared with the original video sequence by using video quality metrics. The metrics used are PSNR and SSIM which are discussed in appendices A and B respectively.

# CHAPTER 5

# RESULTS

The main focus of this thesis is to devise a method for generation of Side Information (SI) in order to improve the quality of the video at the output side of decoder. As discussed in chapter 3, various schemes are considered and the quality comparison is done between them. The measures used for quality comparison are PSNR and SSIM. Apart from considering various schemes for generation of SI the experiments are done to find suitable values for other parameters like ME search range and ME block size. Even though this thesis mainly focuses on key frame distance of 1, where there is only one WZ frame in between two key frames, the impact of higher key frame distance is also studied. In the last section the comparison is done between H.264 base line coding and Wyner-Ziv coding for a fixed bit rate.

## 5.1 PSNR and SSIM with varying SI prediction scheme

The key element in SI generation is motion estimation (ME) as discussed in the chapter 2. In order to estimate gain in video quality by using ME a simple non ME based method is also used. The details on the SI prediction schemes tested are given below.

Average: In this method the SI frame is generated by averaging the two nearest key frames. This method performs reasonably good for several sequences tested. This is because of the lack of fast moving objects in these sequences. The QCIF sequences coastguard, all SIF sequences and CIF sequences have fast moving objects and show significant drop in quality when using this scheme compared to other schemes.

Full-pixel ME: In this method the motion estimation is applied as per section 2.1, but only forward ME is used. The MV is derived in this way for most of the blocks. For those blocks with no MV the predictor MV is used. The predictor MV is derived as median of top, left and top-left blocks as shown in Figure 5.1. Then MC is applied for all blocks in the frame as per section 2.3.

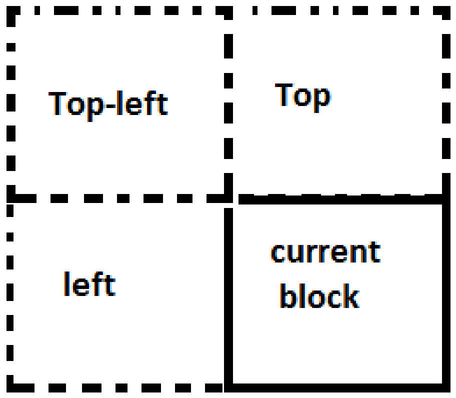

Figure 5.1 Candidates for MV prediction for a block

Half-pixel ME: In this method the half-pixel motion estimation is applied as per section 2.2.1 and only forward ME is used. Also in this method, MV prediction and MC are used as explained earlier. It can be observed from the results that the quality improves with this method when compared to full-pixel ME.

Quarter-pixel ME: In this method the quarter-pixel motion estimation is applied as per section 2.2.2 and only forward ME is used. Also in this method, MV prediction and MC are used as explained earlier. It can be observed that the quality is better with this method when compared to both full-pixel and half-pixel ME methods.

Bi-directional ME: In this method the motion estimation is applied as per section 2.2.2 and both the forward ME and backward ME are used. The resulting two predictions are then interpolated as per section 2.4. Also in this method, MV prediction and MC are used as explained earlier. It can be observed that the quality is better with this method when compared to other ME methods.

Choice of Bi-directional ME and intra-prediction: In this method the bi-directional ME is applied and motion vectors are derived for most of the blocks in a frame. For blocks having motion vectors MC is applied and for blocks without motion vector intra-prediction is applied as per section 2.4. The percentage of blocks with no motion vectors is found to be negligible for the test sequences used in this experiment. Hence it can be seen that no quality improvement is seen in this method over bidirectional ME.

The tables 5.1-5.3 are plotted with the prediction scheme along x-direction and PSNR along y-direction and these plots are given in figures 5.2 - 5.4. The tables 5.4-5.6 are plotted with the prediction scheme along x-direction and SSIM along y-direction and these plots are given in figures 5.5 - 5.7.

Table 5.1 PSNR vs. SI prediction schemes for QCIF (176x144) test sequences

| SI prediction scheme | PSNR(in dB) | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| Average | 34.151404 | 33.659223 | 36.089604 | 39.099904 | 40.349291 |
| Full-pixel ME | 35.343268 | 29.322668 | 33.98107 | 38.588379 | 39.679823 |
| Half-Pixel ME | 35.578079 | 30.806866 | 35.120504 | 38.588379 | 39.679823 |
| Quarter-Pixel ME | 35.578079 | 32.690123 | 35.826314 | 38.588379 | 39.679823 |
| Bi-directional | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |
| Intra-prediction | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |

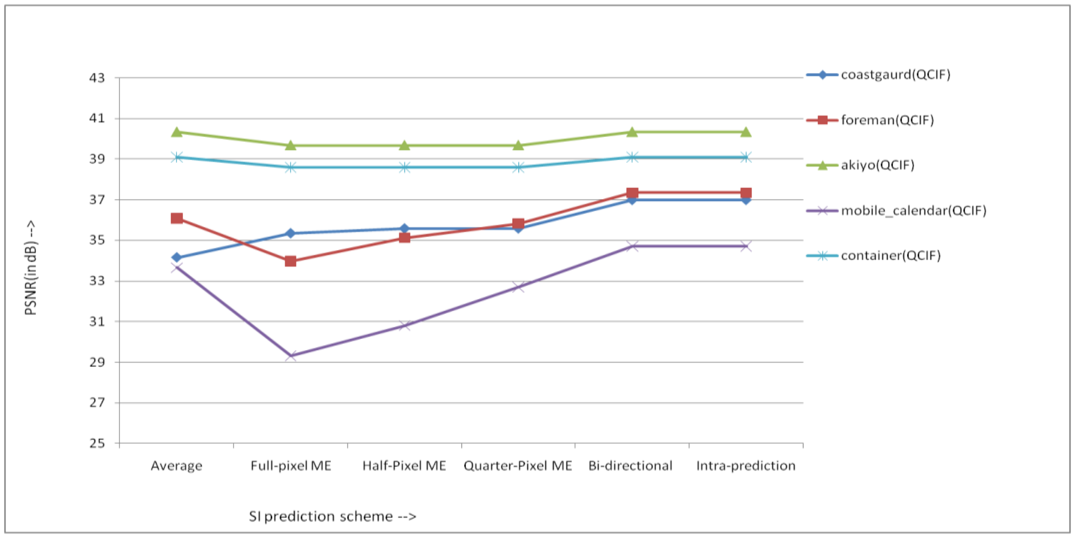

Figure 5.2 PSNR plot for different SI prediction schemes for QCIF (176x144) test sequences

Table 5.2 PSNR vs. SI prediction schemes for SIF (352x240) test sequences

| **SI prediction scheme** | **PSNR(in dB)** | | |
|---|---|---|---|
| | **tennis(SIF)** | **garden(SIF)** | **mobile_calendar(SIF)** |
| Average | 27.716877 | 22.121075 | 26.255596 |
| Full-pixel ME | 29.742313 | 27.600019 | 24.589719 |
| Half-Pixel ME | 30.727177 | 28.992665 | 27.677574 |
| Quarter-Pixel ME | 30.888045 | 29.742313 | 29.154533 |
| Bi-directional | 32.110204 | 31.796119 | 30.97077 |
| Intra-prediction | 32.110204 | 31.796119 | 30.97077 |

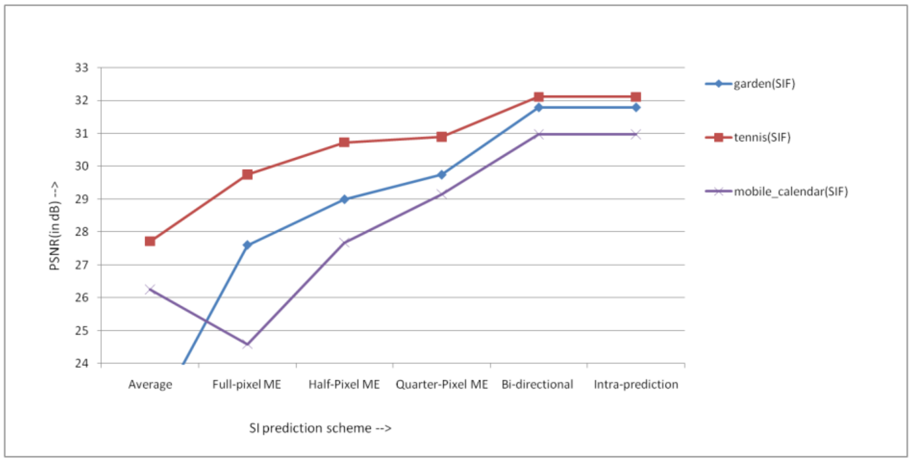

Figure 5.3 PSNR plot for different  SI prediction schemes for SIF (352x240) test sequences

Table 5.3 PSNR vs. SI prediction schemes for CIF (352x288) test sequences

| SI prediction scheme | PSNR(in dB) | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(CIF) |
| Average | 31.796119 | 23.302068 | 34.328691 | 25.366186 |
| Full-pixel ME | 33.659223 | 27.267205 | 34.328691 | 28.785819 |
| Half-Pixel ME | 34.151404 | 28.685977 | 35.578079 | 29.870056 |
| Quarter-Pixel ME | 34.328691 | 29.265896 | 35.578079 | 30.00167 |
| Bi-directional | 36.089604 | 31.228843 | 36.669523 | 31.503225 |
| Intra-prediction | 36.089604 | 31.228843 | 36.669523 | 31.503225 |

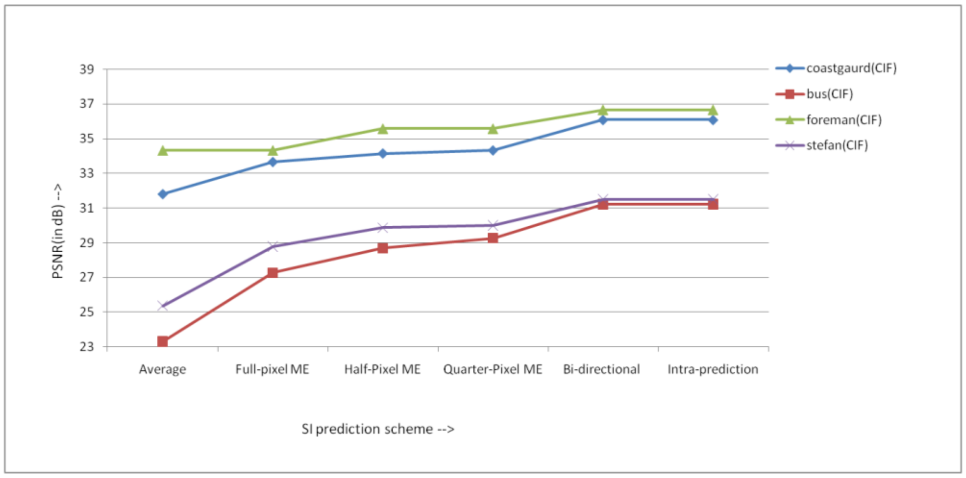

Figure 5.4 PSNR plot for different SI prediction schemes for CIF (352x288) test sequences

Table 5.4 SSIM vs. SI prediction schemes for QCIF (176x144) test sequences

| SI prediction scheme | SSIM | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| Average | 0.9497 | 0.9878 | 0.9840 | 0.9743 | 0.9922 |
| Full-pixel ME | 0.9539 | 0.9754 | 0.9804 | 0.9731 | 0.9915 |
| Half-Pixel ME | 0.9555 | 0.9798 | 0.9827 | 0.9732 | 0.9916 |
| Quarter-Pixel ME | 0.9563 | 0.9854 | 0.9837 | 0.9731 | 0.9918 |
| Bi-directional | 0.9629 | 0.9891 | 0.9856 | 0.9743 | 0.9923 |
| Intra-prediction | 0.9629 | 0.9891 | 0.9856 | 0.9743 | 0.9923 |

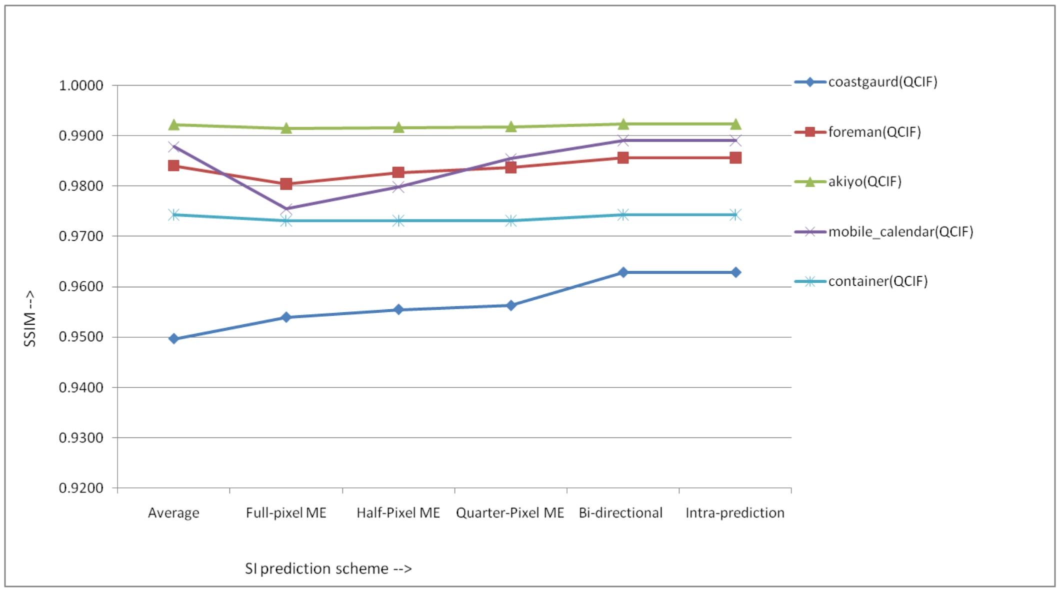

Figure 5.5 SSIM plot for different SI prediction schemes for QCIF (176x144) test sequences

Table 5.5 SSIM vs. SI prediction schemes for SIF (352x240) test sequences

| **SI prediction scheme** | **SSIM** | | |
|---|---|---|---|
| | **tennis(SIF)** | **garden(SIF)** | **mobile_calendar(SIF)** |
| Average | 0.9138 | 0.8683 | 0.9552 |
| Full-pixel ME | 0.9346 | 0.9541 | 0.9420 |
| Half-Pixel ME | 0.9410 | 0.9643 | 0.9670 |
| Quarter-Pixel ME | 0.9430 | 0.9683 | 0.9756 |
| Bi-directional | 0.9515 | 0.9752 | 0.9816 |
| Intra-prediction | 0.9515 | 0.9752 | 0.9816 |

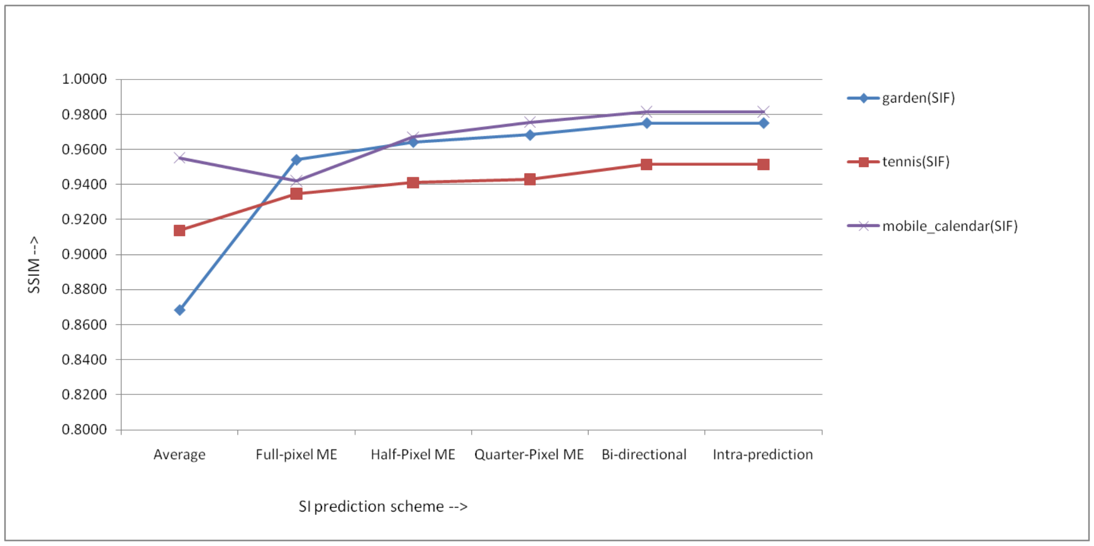

Figure 5.6 SSIM plot for different SI prediction schemes for SIF (352x240) test sequences

Table 5.6 SSIM vs. SI prediction schemes for CIF (352x288) test sequences

| SI prediction scheme | SSIM | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(CIF) |
| Average | 0.9333 | 0.8670 | 0.9817 | 0.8911 |
| Full-pixel ME | 0.9491 | 0.9443 | 0.9815 | 0.9423 |
| Half-Pixel ME | 0.9517 | 0.9535 | 0.9837 | 0.9504 |
| Quarter-Pixel ME | 0.9522 | 0.9577 | 0.9840 | 0.9514 |
| Bi-directional | 0.9618 | 0.9663 | 0.9856 | 0.9604 |
| Intra-prediction | 0.9618 | 0.9663 | 0.9856 | 0.9603 |

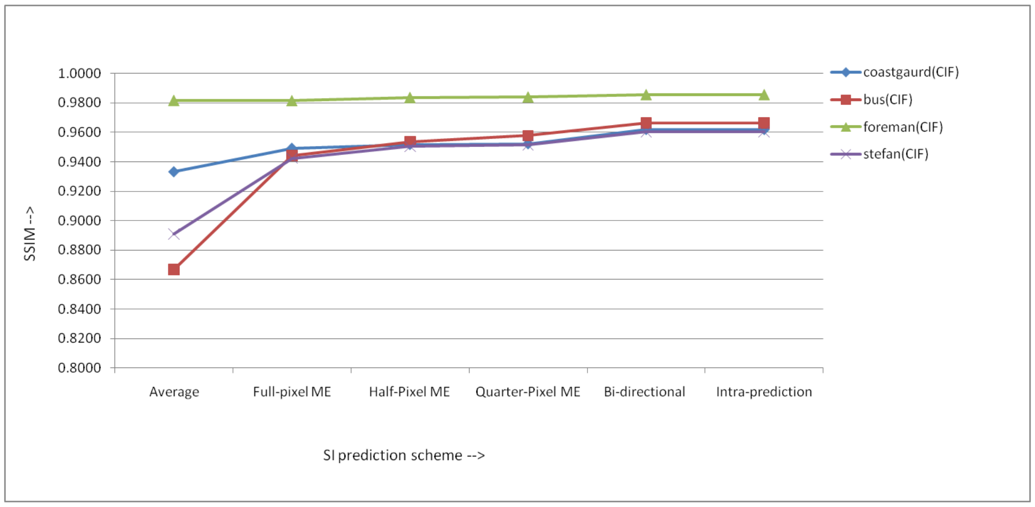

Figure 5.7 SSIM plot for different SI prediction schemes for CIF (352x288) test sequences

## 5.2 PSNR and SSIM with different ME block sizes

In this experiment ME block size for SI generation is varied and the quality of the decoded frame is measured. The block sizes considered are 16x16, 8x8 and 4x4. It can be observed from the tables 5.7-5.12 that the block size of 16x16 is best suited for ME in terms of the video quality. The figures 5.8-5.10 show plots with ME block size along x-direction and PSNR along y-direction. The figures 5.11-5.13 show plots with ME block size along x-direction and SSIM along y-direction.

Table 5.7 PSNR vs. ME block sizes for QCIF (176x144) test sequences

| Block size | PSNR(in dB) | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| 16x16 | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |
| 8x8 | 36.089604 | 33.98107 | 35.826314 | 38.588379 | 40.349291 |
| 4x4 | 32.690123 | 29.742313 | 32.816014 | 36.089604 | 38.588379 |

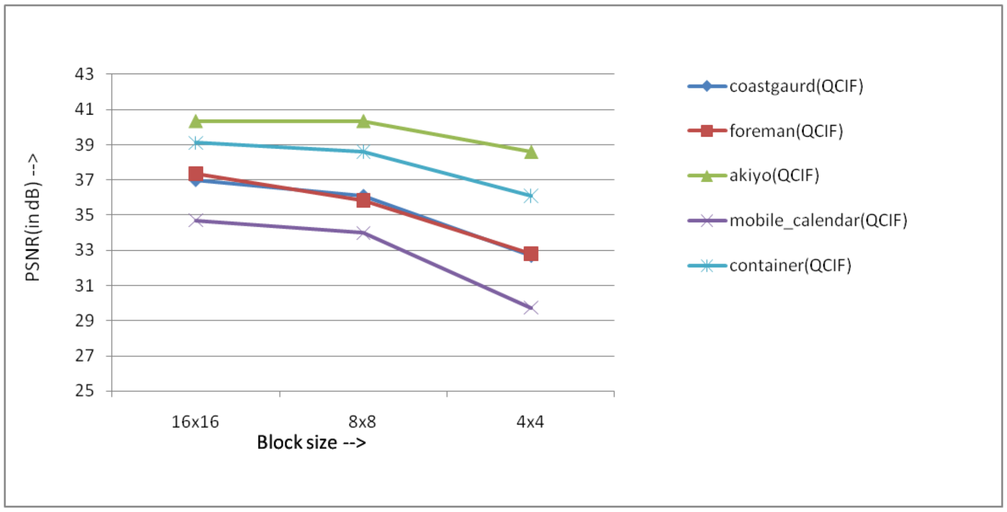

Figure 5.8 PSNR plot for different ME block sizes for QCIF (176x144) test sequences

Table 5.8 PSNR vs. ME block sizes for SIF(352x240) test sequences

| Block size | PSNR(in dB) | | |
|---|---|---|---|
| | tennis(SIF) | garden(SIF) | mobile_calendar(SIF) |
| 16x16 | 32.110204 | 31.796119 | 30.97077 |
| 8x8 | 31.696277 | 31.055102 | 30.00167 |
| 4x4 | 30.069004 | 29.099904 | 27.338991 |

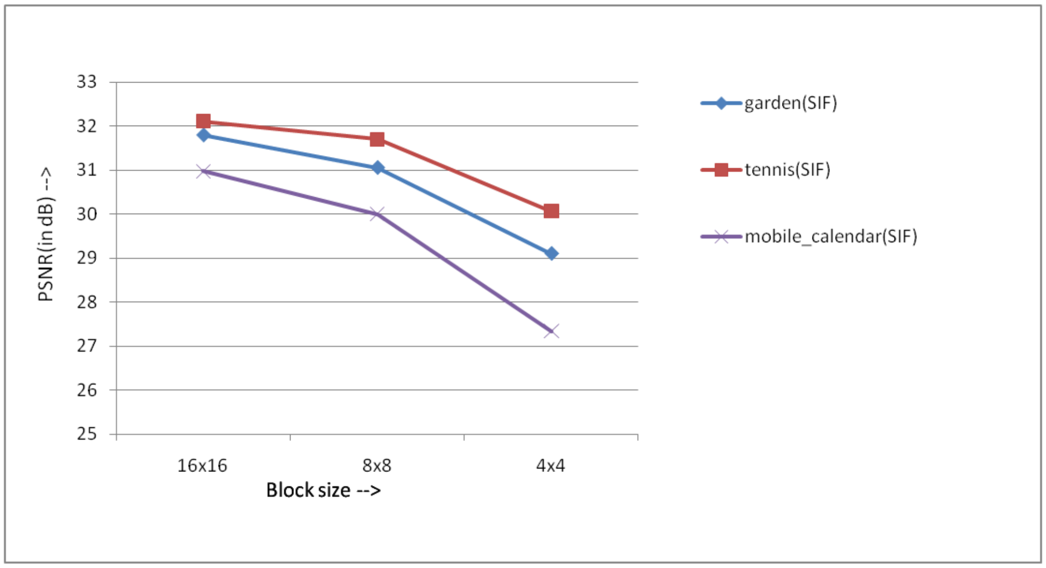

Figure 5.9 PSNR plot for different ME block sizes for SIF (352x240) test sequences

Table 5.9 PSNR vs. ME block sizes for CIF (352x288) test sequences

| Block size | PSNR(in dB) | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(SIF) |
| 16x16 | 36.089604 | 31.228843 | 36.669523 | 31.503225 |
| 8x8 | 33.359591 | 29.679823 | 36.089604 | 30.727177 |
| 4x4 | 30.00167 | 27.677574 | 34.513525 | 29.497575 |

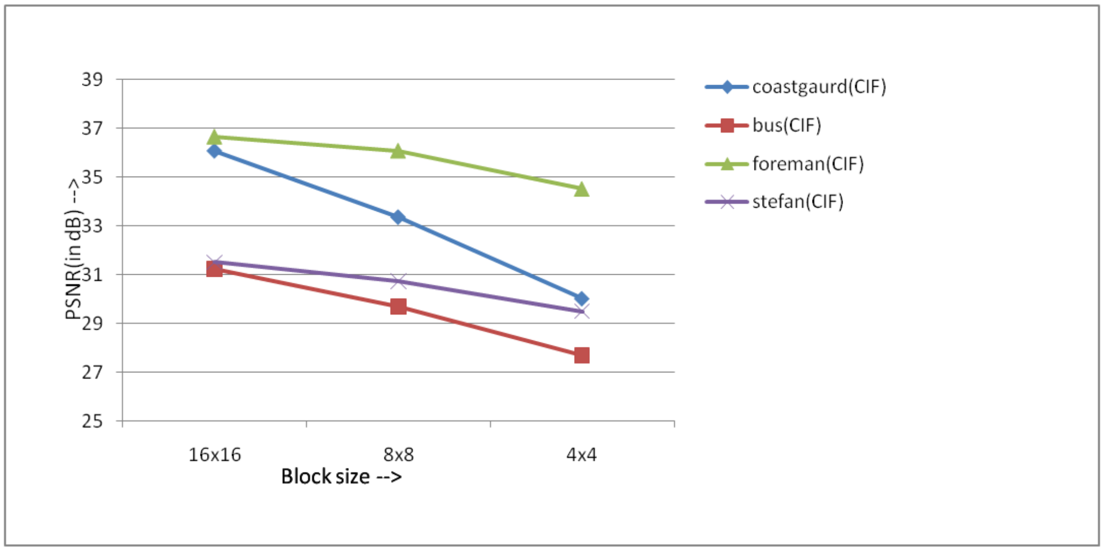

Figure 5.10 PSNR plot for different ME block sizes for CIF (352x288) test sequences

Table 5.10 SSIM vs. ME block sizes for QCIF (176x144) test sequences

| Block size | SSIM | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| 16x16 | 0.9629 | 0.9891 | 0.9856 | 0.9743 | 0.9923 |
| 8x8 | 0.9551 | 0.9881 | 0.9836 | 0.9724 | 0.9923 |
| 4x4 | 0.9215 | 0.9749 | 0.9771 | 0.9681 | 0.9908 |

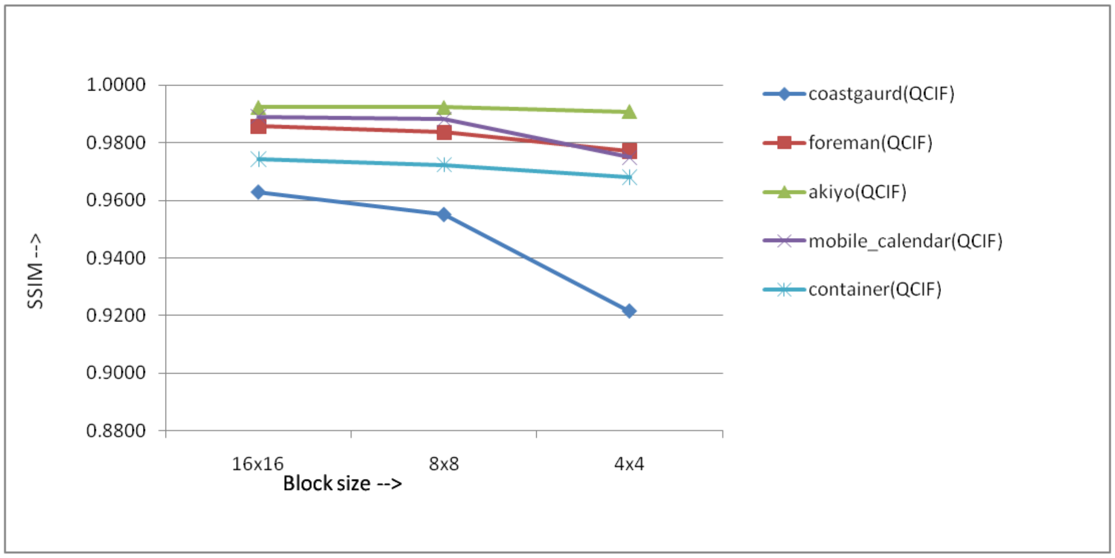

Figure 5.11 SSIM plot for different ME block sizes for QCIF (176x144) test sequences

Table 5.11 SSIM vs. ME block sizes for SIF(352x240) test sequences

| Block size | SSIM | | |
|---|---|---|---|
| | tennis(SIF) | garden(SIF) | mobile_calendar(SIF) |
| 16x16 | 0.9515 | 0.9752 | 0.9816 |
| 8x8 | 0.9499 | 0.9729 | 0.9787 |
| 4x4 | 0.9277 | 0.9642 | 0.9630 |

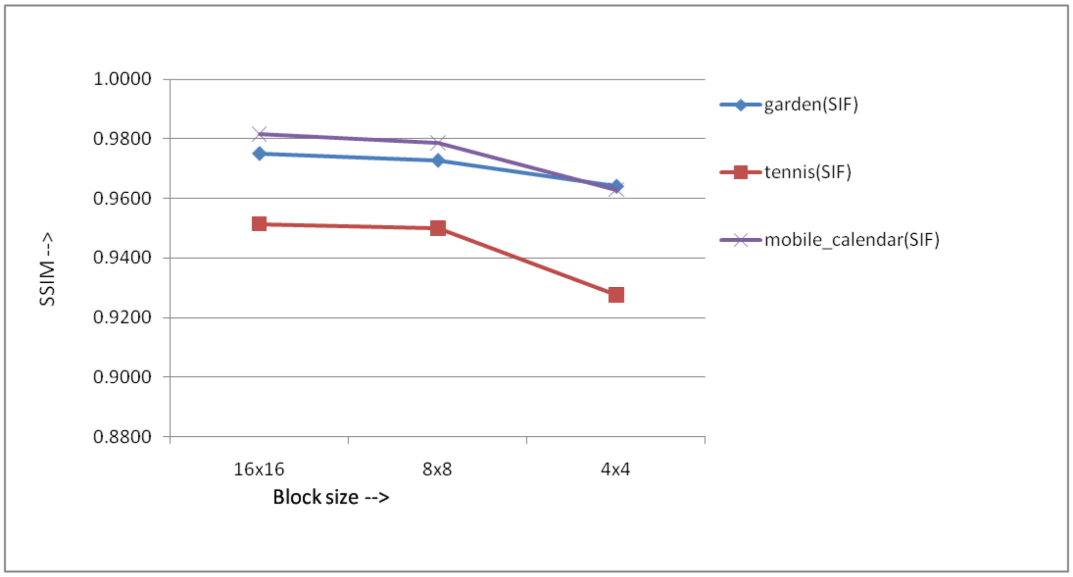

Figure 5.12 SSIM plot for different ME block sizes for SIF (352x240) test sequences

Table 5.12 SSIM vs. ME block sizes for CIF (352x288) test sequences

| Block size | SSIM | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(CIF) |
| 16x16 | 0.9618 | 0.9663 | 0.9856 | 0.9603 |
| 8x8 | 0.9381 | 0.9590 | 0.9845 | 0.9550 |
| 4x4 | 0.8926 | 0.9423 | 0.9816 | 0.9415 |

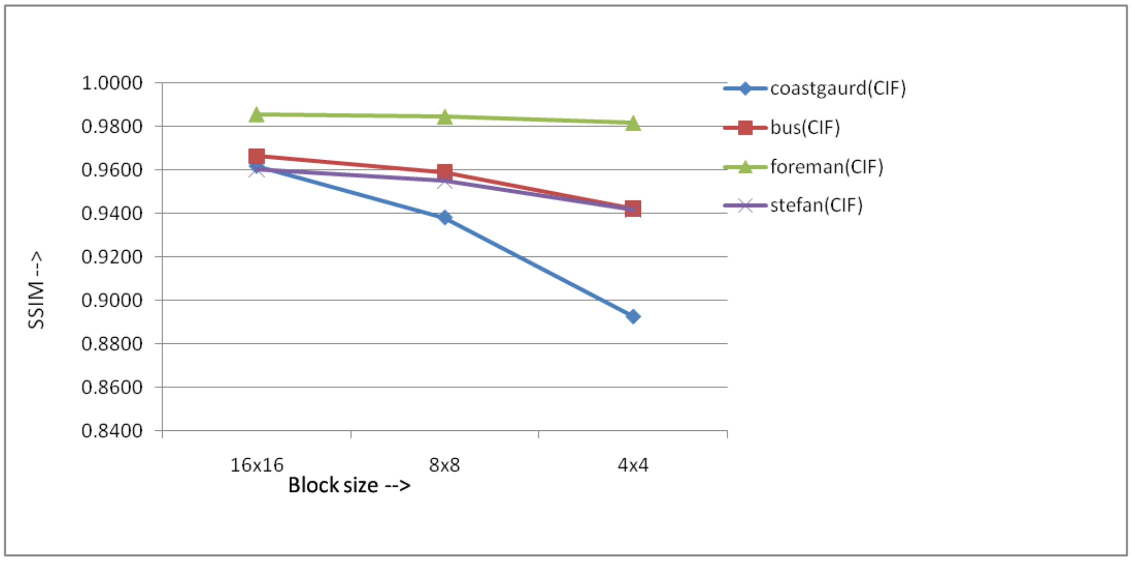

Figure 5.13 SSIM plot for different ME block sizes for CIF (352x288) test sequences

## 5.3 PSNR and SSIM with different ME search ranges

In this experiment ME search range for SI generation is varied and the quality of the decoded frame is measured. The search ranges considered are 16, 8 and 4 pixels. The search range determines the size of the window around within which a block is searched. Figure 5.14 shows the size of search window in terms of search range.

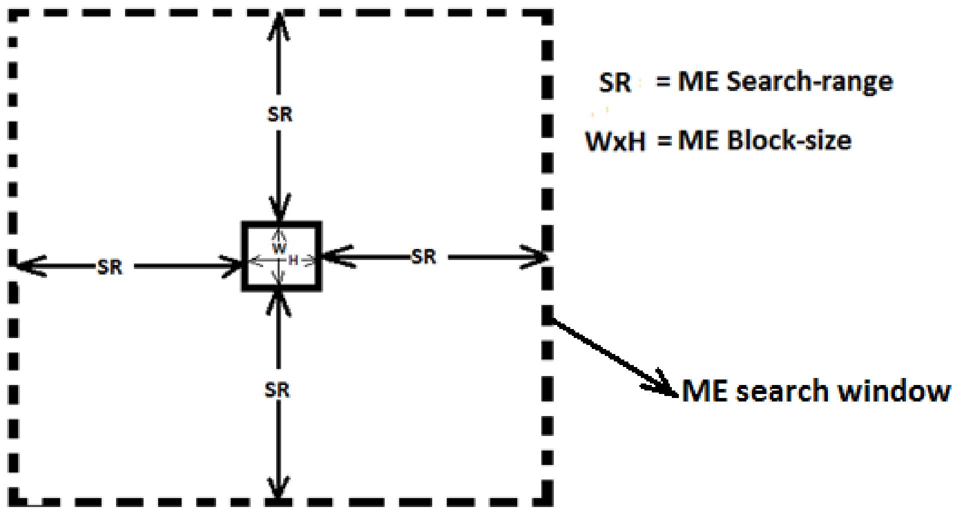

Figure 5.14 ME search window for a given search-range and block-size

It can be observed from the tables 5.13 - 5.18 that increase in ME search range increases video quality. This is especially true for test sequences with fast moving objects. The figures 5.15 - 5.17 show plots with ME search range along x-direction and PSNR along y-direction. The figures 5.18 - 5.20 show plots with ME search range along x-direction and SSIM along y-direction.

Table 5.13 PSNR vs. ME search ranges for QCIF (176x144) test sequences

| Search range | PSNR(in dB) | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| 16 | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |
| 8 | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |
| 4 | 36.99137 | 34.706577 | 37.338991 | 39.099904 | 40.349291 |

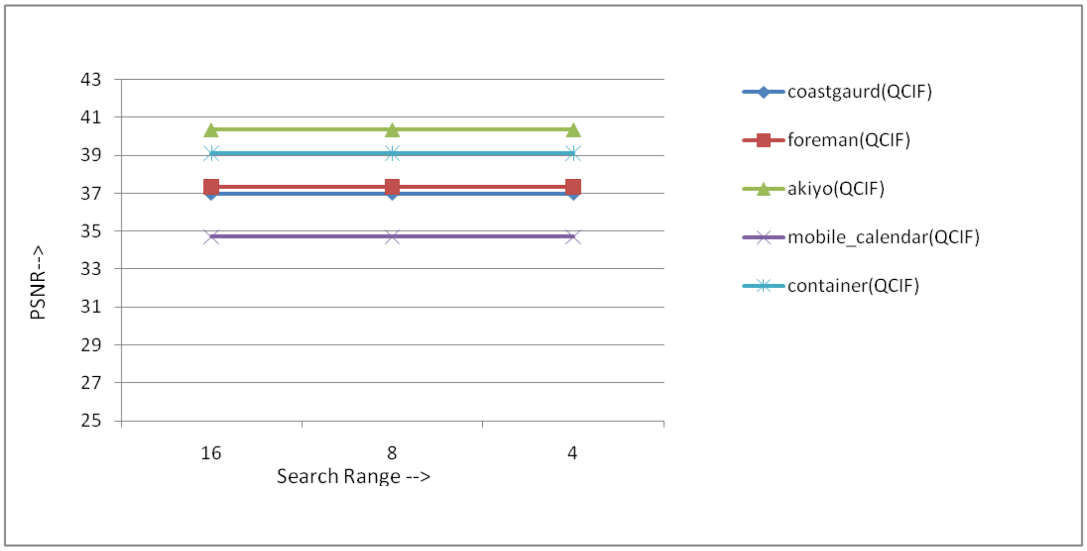

Figure 5.15 PSNR plot for different ME search ranges for QCIF (176x144) test sequences

Table 5.14 PSNR vs. ME search ranges for SIF (352x240) test sequences

| Search range | PSNR(in dB) | | |
|---|---|---|---|
| | tennis(SIF) | garden(SIF) | mobile_calendar(SIF) |
| 16 | 32.110204 | 31.796119 | 30.97077 |
| 8 | 32.110204 | 31.696277 | 31.141104 |
| 4 | 30.648923 | 28.685977 | 31.141104 |

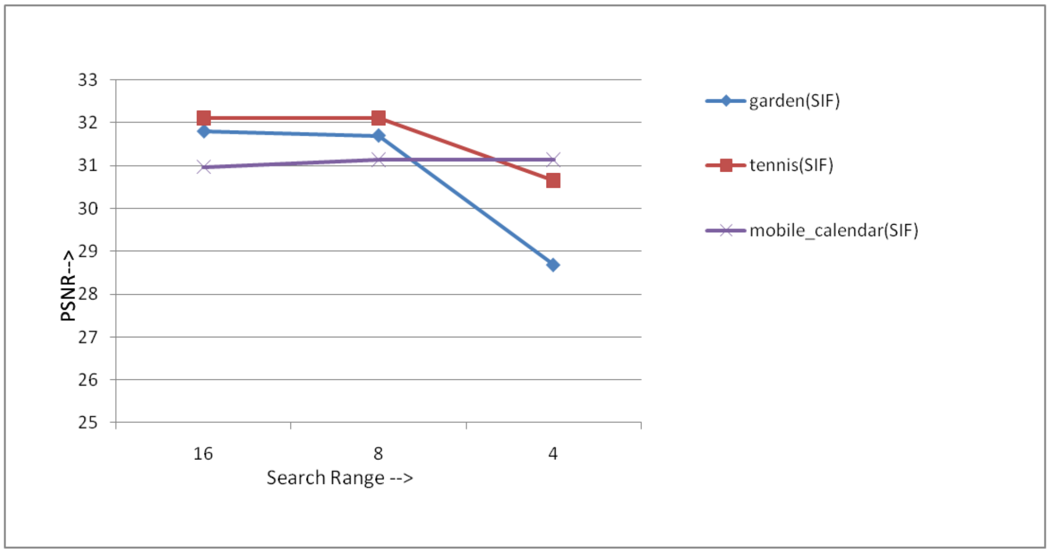

Figure 5.16 PSNR plot for different ME search ranges for SIF (352x240) test sequences

Table 5.15 PSNR vs. ME search ranges for CIF (352x288) test sequences

| Search range | PSNR(in dB) | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(CIF) |
| 16 | 36.089604 | 31.228843 | 36.669523 | 31.503225 |
| 8 | 36.089604 | 25.412388 | 36.669523 | 28.218543 |
| 4 | 34.706577 | 23.849456 | 36.669523 | 26.341034 |

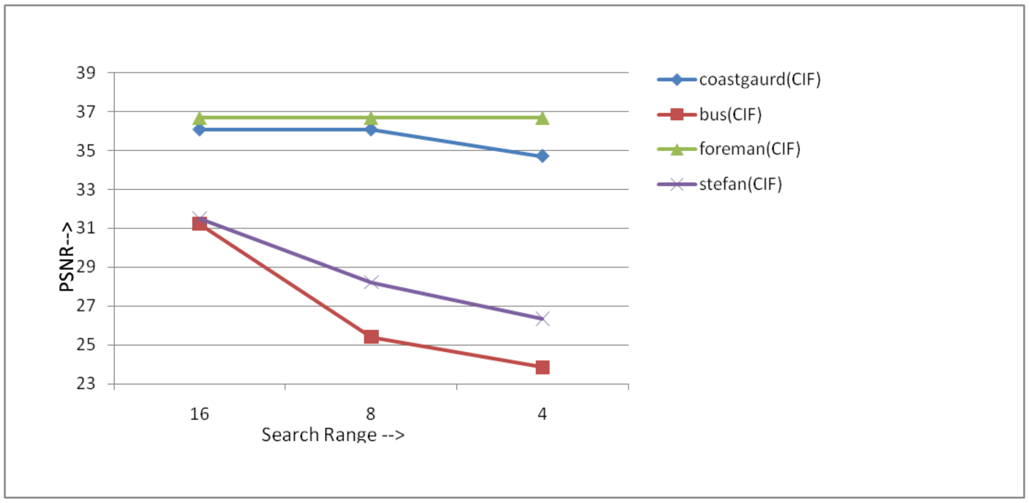

Figure 5.17 PSNR plot for different ME search ranges for CIF (352x288) test sequences

Table 5.16 SSIM vs. ME search ranges for QCIF (176x144) test sequences

| Search range | SSIM | | | | |
|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile_calendar(QCIF) | foreman(QCIF) | container(QCIF) | akiyo(QCIF) |
| 16 | 0.9629 | 0.9891 | 0.9856 | 0.9743 | 0.9923 |
| 8 | 0.9629 | 0.9891 | 0.9858 | 0.9743 | 0.9923 |
| 4 | 0.9629 | 0.9891 | 0.9858 | 0.9744 | 0.9923 |

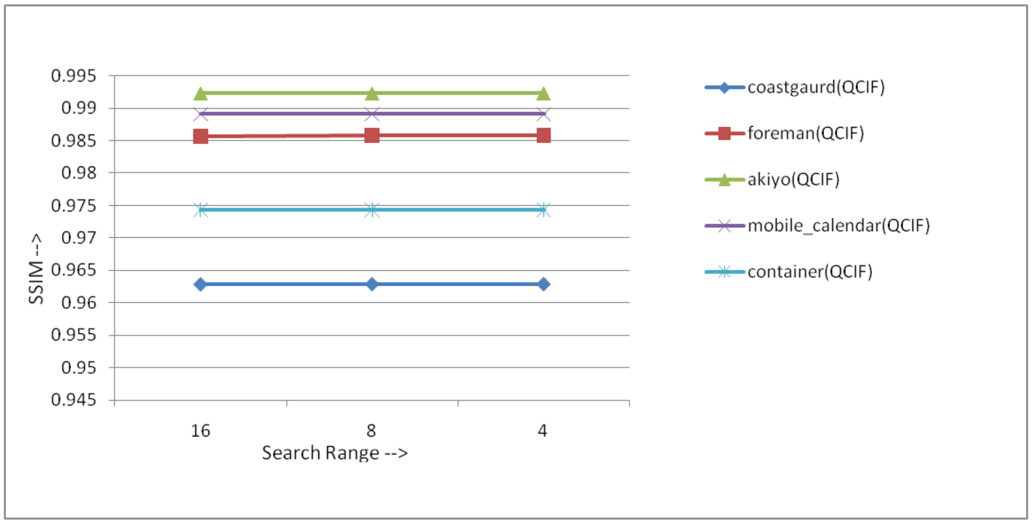

Figure 5.18 SSIM plot for different  ME search ranges for QCIF (176x144) test sequences

Table 5.17 SSIM vs. ME search ranges for SIF (352x240) test sequences

| Search range | SSIM | | |
|---|---|---|---|
| | tennis(SIF) | garden(SIF) | mobile_calendar(SIF) |
| 16 | 0.9515 | 0.9752 | 0.9816 |
| 8 | 0.9513 | 0.9748 | 0.9821 |
| 4 | 0.9435 | 0.9615 | 0.9822 |

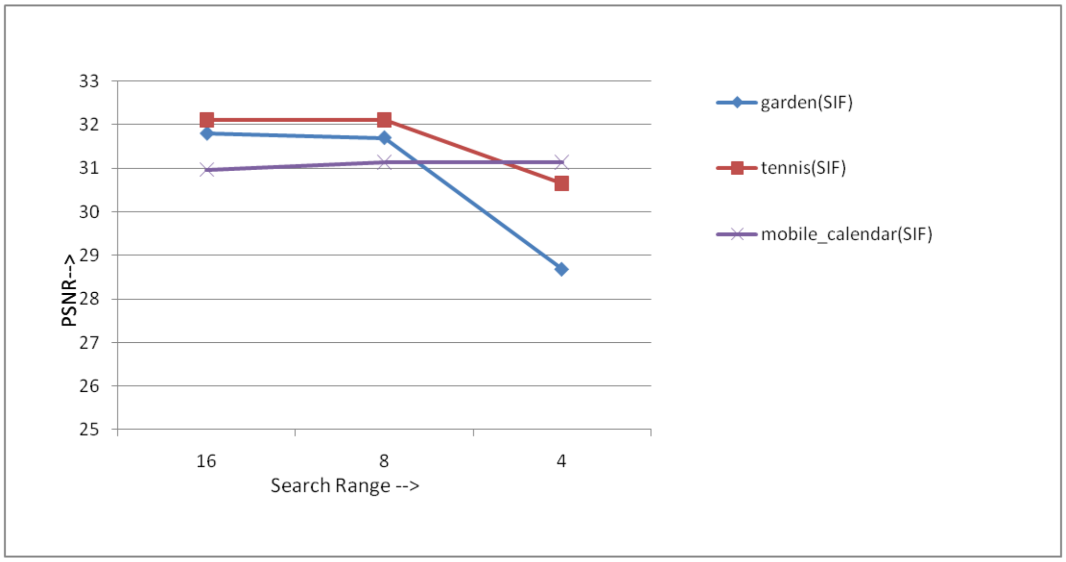

Figure 5.19 SSIM plot for different  ME search ranges for SIF (352x240) test sequences

Table 5.18 SSIM vs. ME search ranges for CIF (352x288) test sequences

| Search range | SSIM | | | |
|---|---|---|---|---|
| | coastgaurd(CIF) | bus(CIF) | foreman(CIF) | stefan(CIF) |
| 16 | 0.9618 | 0.9663 | 0.9856 | 0.9603 |
| 8 | 0.9624 | 0.9106 | 0.9858 | 0.9353 |
| 4 | 0.9582 | 0.8820 | 0.9857 | 0.9121 |

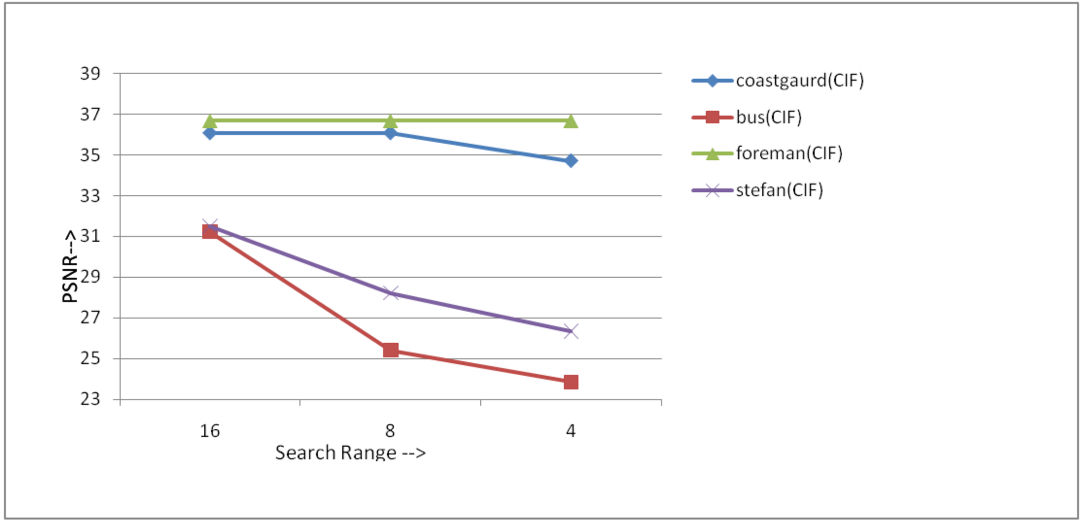

Figure 5.20 SSIM plot for different ME search ranges for CIF (352x288) test sequences

## 5.4 PSNR and SSIM with different key frame distances

In this experiment the key frame distance is varied and the quality of the decoded frame is measured. The key frame distance is the number of WZ frames in between two key frames. The key frame distance is varied from 1 to 5. This is done for many test sequences and the results are given in the tables 5.19 and 5.20. The increase in key frame distance results in poor prediction and hence decrease in video quality. Figure 5.21 shows plot with key frame distance along x-direction and PSNR along y-direction. Figure 5.22 shows plot with key frame distance along x-direction and SSIM along y-direction.

Table 5.19 PSNR vs. key frame distances for several test sequences

| key frame distance | PSNR (in dB) | | | | | |
|---|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile(QCIF) | tennis(SIF) | garden(SIF) | bus(CIF) | foreman(CIF) |
| 1 | 36.99137 | 34.706577 | 32.110204 | 31.796119 | 31.22884 | 36.669523 |
| 2 | 35.826314 | 33.359591 | 29.557479 | 28.785819 | 24.43865 | 33.659223 |
| 3 | 30.806866 | 26.958091 | 25.554018 | 20.178958 | 20.2139 | 28.735611 |
| 4 | 30.069004 | 25.602273 | 24.706577 | 19.883039 | 19.34559 | 27.756539 |
| 5 | 28.263086 | 23.536879 | 23.64374 | 17.96047 | 18.35814 | 25.982365 |

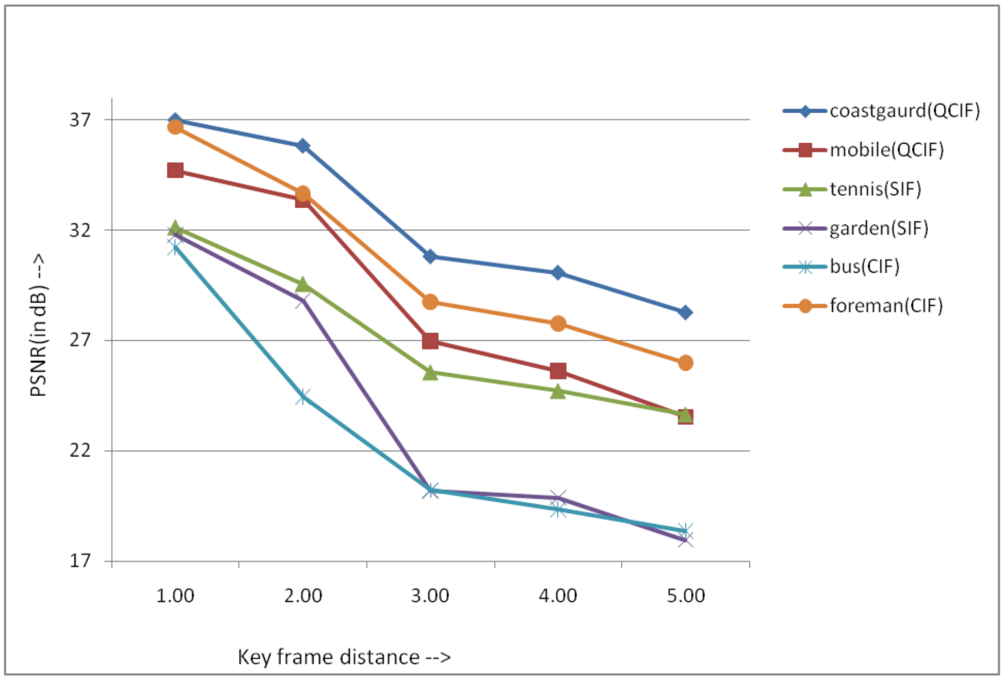

Figure 5.21 PSNR plot for different key frame distances for several test sequences

Table 5.20 SSIM vs. key frame distances for several test sequences

| key frame distance | SSIM | | | | | |
|---|---|---|---|---|---|---|
| | coastgaurd(QCIF) | mobile(QCIF) | tennis(SIF) | garden(SIF) | bus(CIF) | foreman(CIF) |
| **1** | 0.9629 | 0.9891 | 0.9515 | 0.9752 | 0.9663 | 0.9856 |
| **2** | 0.9567 | 0.9873 | 0.9301 | 0.9640 | 0.8994 | 0.9813 |
| **3** | 0.9224 | 0.9603 | 0.8799 | 0.8626 | 0.8007 | 0.9615 |
| **4** | 0.9113 | 0.9492 | 0.8549 | 0.8476 | 0.7508 | 0.9547 |
| **5** | 0.8870 | 0.9230 | 0.8250 | 0.7775 | 0.7131 | 0.9361 |

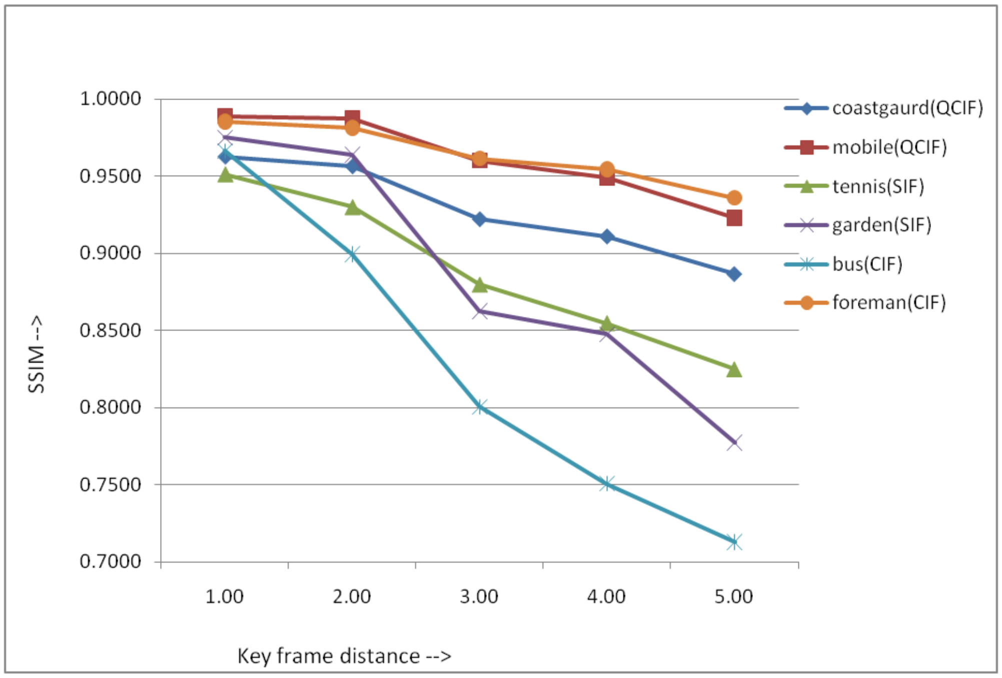

Figure 5.22 SSIM plot for different  key frame distances for several test sequences

## 5.5 PSNR and SSIM with different rates

This experiment focuses on rate-distortion (RD) curves for the WZ codec compared to that of H.264 codec. The RD curves are obtained by plotting bit-rate of the encoded stream along the x-direction and the PSNR of the decoded YUV along the Y-direction. The figures 5.23 – 5.28 show the RD curves obtained for several sequences ranging  from QCIF to CIF resolution. The figures 5.29 – 5.34 show the SSIM vs. bit rate curves obtained for several sequences ranging  from QCIF to CIF resolution. The table 5.21 combines the result in a single table. It is intuitive that the increase in bit-rate causes increase in quality which is evident from the RD curves. Additionally the H.264 codec has better RD curves compared to Wyner-Ziv. The distortion caused by H.264 compression is significantly less compared to that of Wyner-Ziv compression method. H.264 compression method produces better results due to the fact that it achieves better compression by employing number of toolsets like quantization, transform , entropy coding etc.

Table 5.21 PSNR and SSIM at different bit rates for several test sequences

| Test YUV | H.264 baseline | | | Wyner-Ziv | | |
|---|---|---|---|---|---|---|
| | Rate(kbps) | Psnr(in dB) | SSIM | Rate(kbps) | Psnr(in dB) | SSIM |
| **coastgaurd(QCIF)** | 1504 | 43.36 | 0.9885 | 1437 | 41.14 | 0.9854 |
| | 1210 | 42.11 | 0.9846 | 1184 | 39.10 | 0.9765 |
| | 1004 | 41.14 | 0.9826 | 1019 | 36.99 | 0.9629 |
| | 804 | 39.68 | 0.9772 | 819 | 33.66 | 0.9286 |
| | 604 | 37.72 | 0.9678 | 665 | 29.50 | 0.8462 |
| **mobile_calendar(QCIF)** | 1510 | 37.72 | 0.9936 | 2150 | 38.13 | 0.9956 |
| | 1207 | 35.58 | 0.9902 | 1811 | 36.37 | 0.9931 |
| | 1006 | 33.98 | 0.9864 | 1564 | 34.71 | 0.9891 |
| | 809 | 31.90 | 0.9797 | 1216 | 31.32 | 0.9770 |
| | 605 | 28.35 | 0.9596 | 884 | 26.46 | 0.9395 |
| **tennis(SIF)** | 6037 | 43.36 | 0.9892 | 7149 | 33.22 | 0.9677 |
| | 4822 | 41.14 | 0.9863 | 6060 | 32.82 | 0.9607 |
| | 4042 | 40.35 | 0.9830 | 5273 | 32.11 | 0.9515 |
| | 3223 | 38.59 | 0.9780 | 4145 | 30.42 | 0.9217 |
| | 2416 | 36.37 | 0.9689 | 2994 | 28.04 | 0.8509 |
| **garden(SIF)** | 6012 | 38.59 | 0.9899 | 5283 | 33.08 | 0.9863 |
| | 4811 | 36.37 | 0.9848 | 4339 | 32.57 | 0.9819 |
| | 4013 | 34.51 | 0.9795 | 3692 | 31.80 | 0.9752 |
| | 3212 | 32.57 | 0.9713 | 2807 | 29.81 | 0.9571 |
| | 2411 | 29.56 | 0.9530 | 2163 | 25.96 | 0.9131 |
| **bus(CIF)** | 6011 | 43.36 | 0.9922 | 5886 | 32.00 | 0.9783 |
| | 4816 | 42.11 | 0.9900 | 4936 | 31.70 | 0.9732 |
| | 4023 | 40.35 | 0.9876 | 4305 | 31.23 | 0.9663 |
| | 3225 | 38.59 | 0.9839 | 3524 | 30.07 | 0.9516 |
| | 2418 | 36.37 | 0.9770 | 2847 | 27.41 | 0.9157 |
| **foreman(CIF)** | 5995 | 48.13 | 0.9965 | 4383 | 38.59 | 0.9921 |
| | 4836 | 48.13 | 0.9959 | 3678 | 37.72 | 0.9891 |
| | 4017 | 48.13 | 0.9956 | 3271 | 36.67 | 0.9856 |
| | 3221 | 45.12 | 0.9945 | 2849 | 34.71 | 0.9788 |
| | 2411 | 43.36 | 0.9934 | 2552 | 31.70 | 0.9640 |

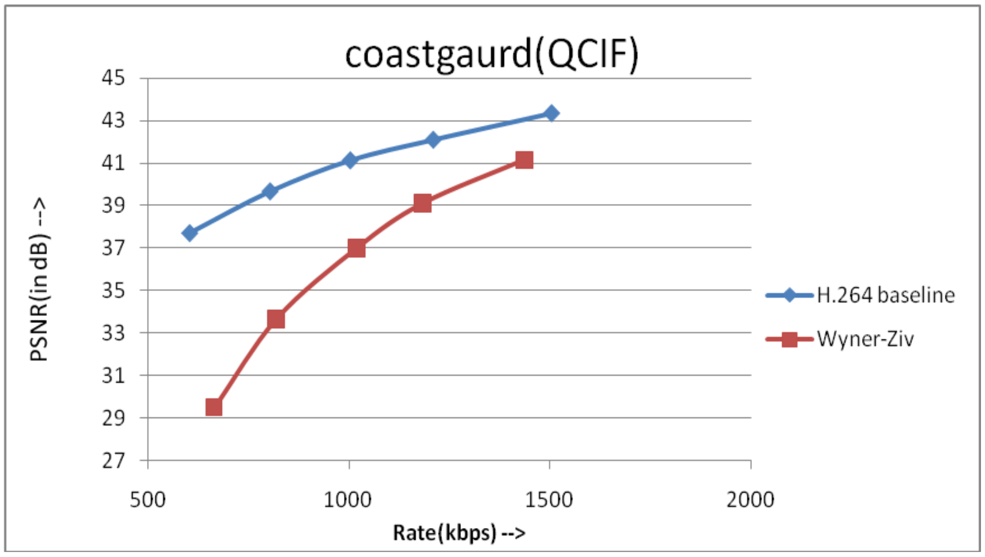

Figure 5.23 Rate-distortion plot for H.264 baseline and WZ codec, for the coastguard (QCIF) sequence

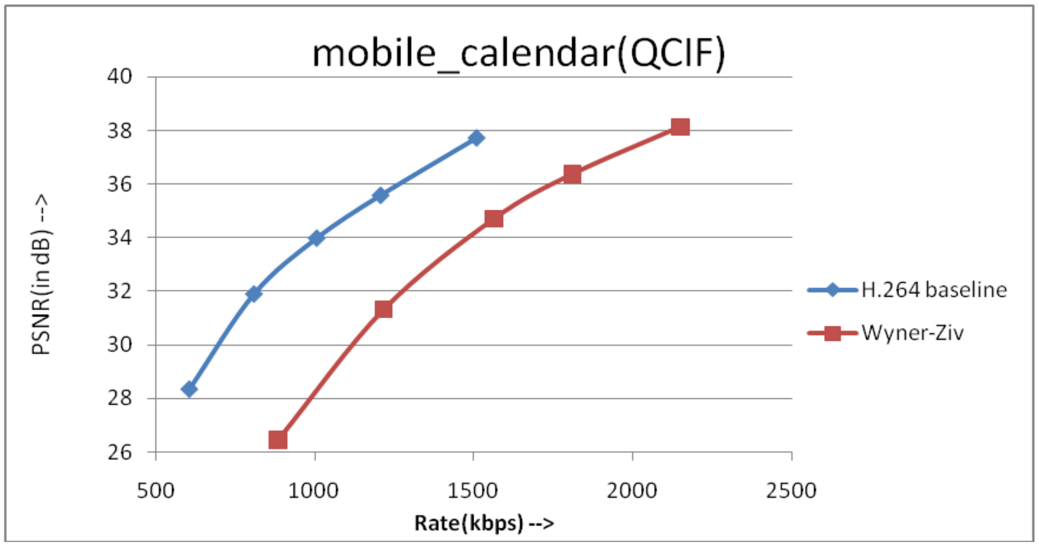

Figure 5.24 Rate-distortion plot for H.264 baseline and WZ codec, for the mobile_calendar (QCIF) sequence

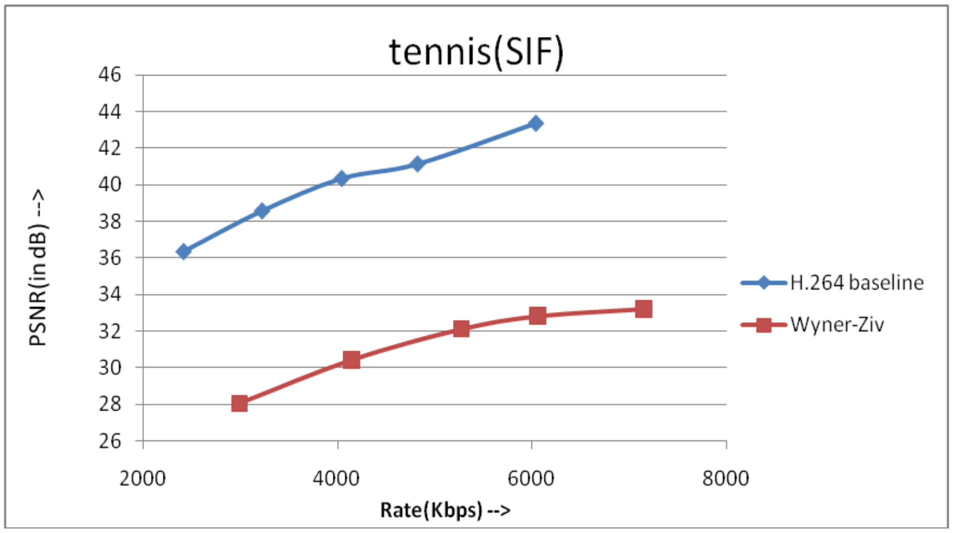

Figure 5.25 Rate-distortion plot for H.264 baseline and WZ codec, for the tennis (SIF) sequence

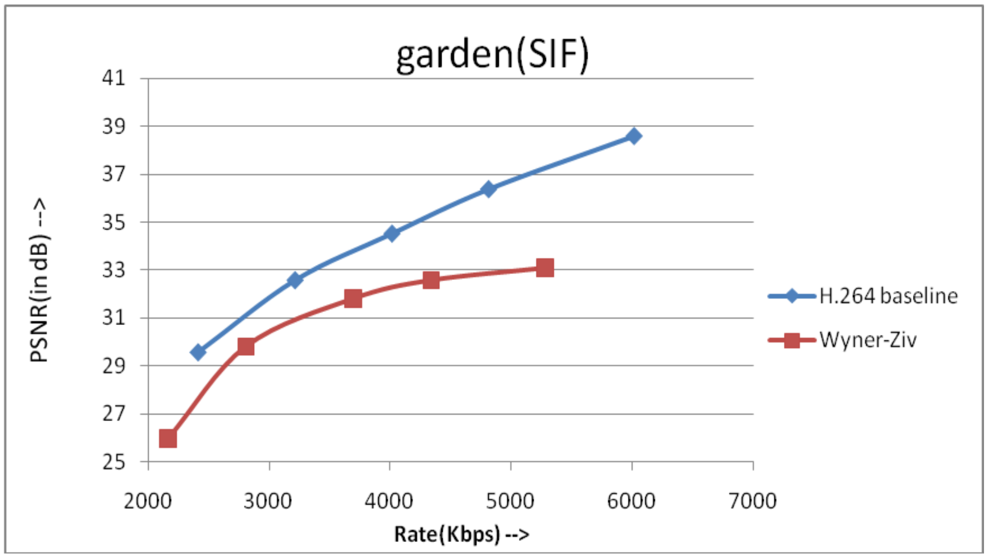

Figure 5.26 Rate-distortion plot for H.264 baseline and WZ codec, for the garden (SIF) sequence

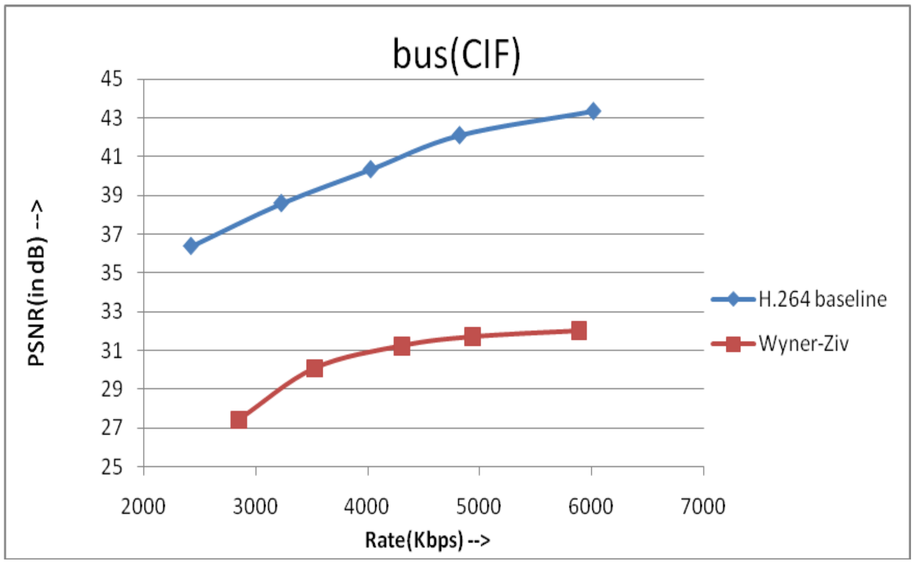

Figure 5.27 Rate-distortion plot for H.264 baseline and WZ codec, for the bus (CIF) sequence

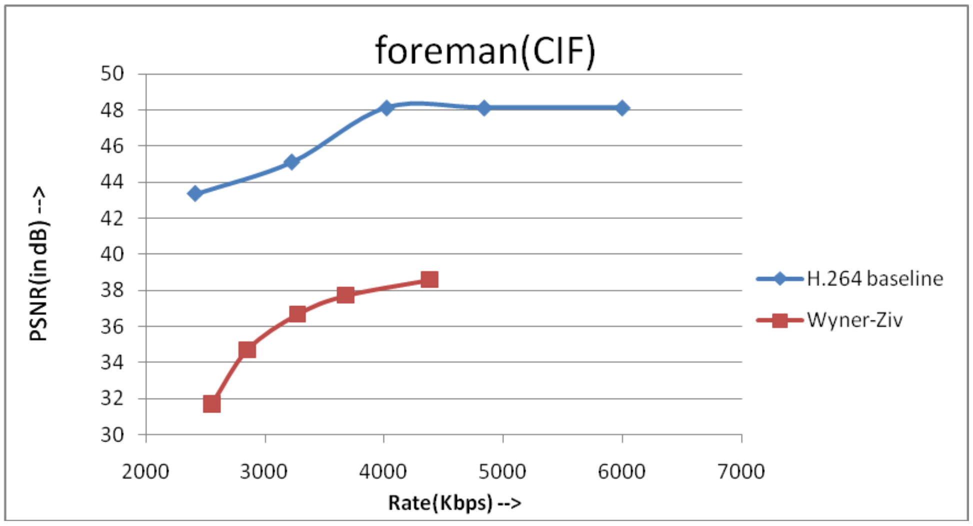

Figure 5.28 Rate-distortion plot for H.264 baseline and WZ codec, for the foreman (CIF) sequence

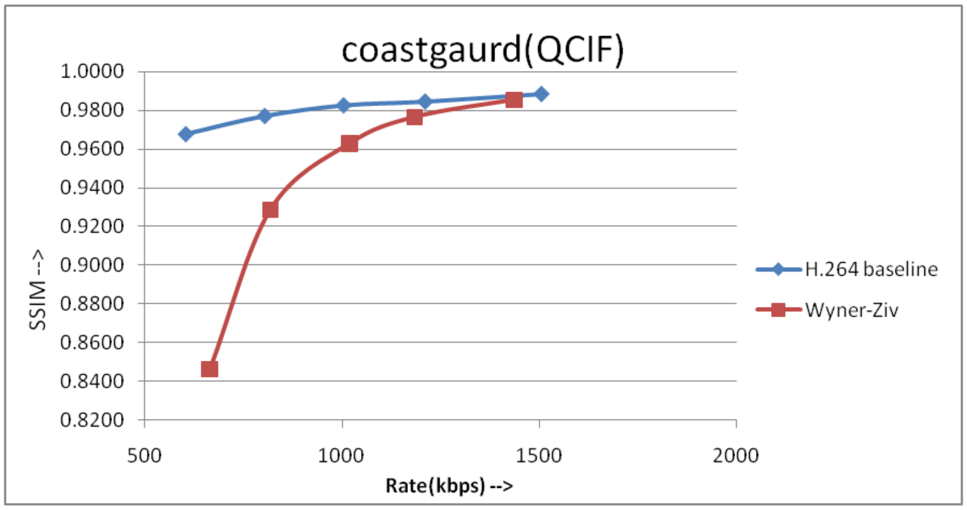

Figure 5.29 SSIM vs. bit rate for H.264 baseline and WZ codec, for the coastguard (QCIF) sequence

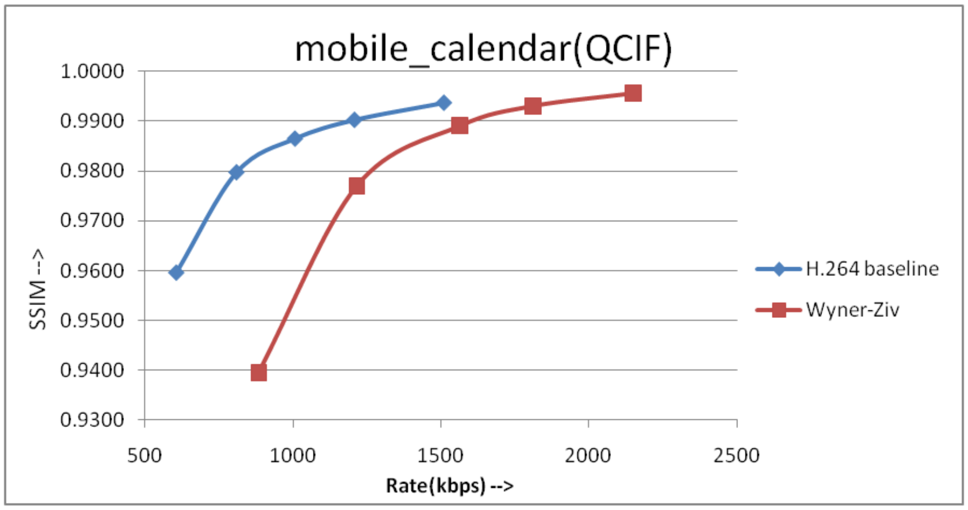

Figure 5.30 SSIM vs. bit rate for H.264 baseline and WZ codec, for the mobile_calendar (QCIF) sequence

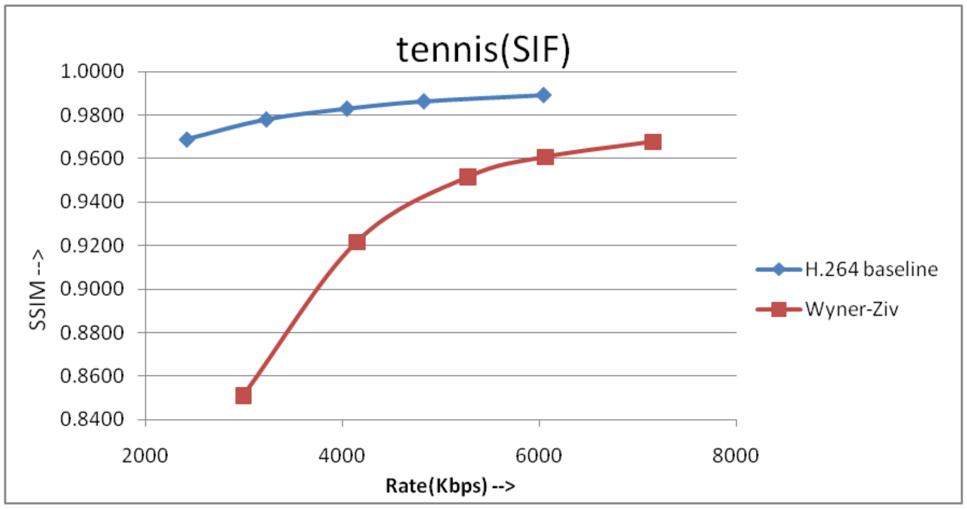

Figure 5.31 SSIM vs. bit rate for H.264 baseline and WZ codec, for the tennis (SIF) sequence

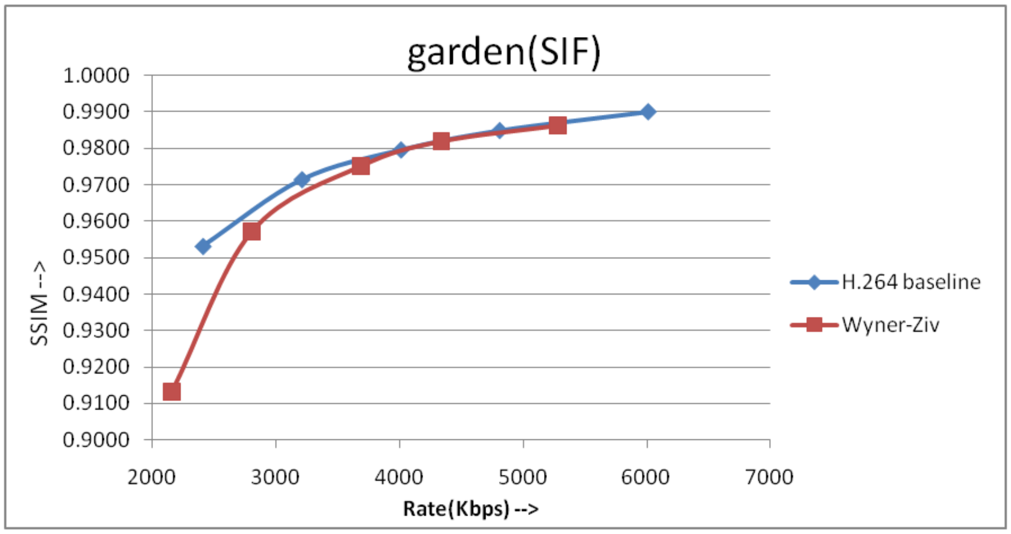

Figure 5.32 SSIM vs. bit rate for H.264 baseline and WZ codec, for the garden (SIF) sequence

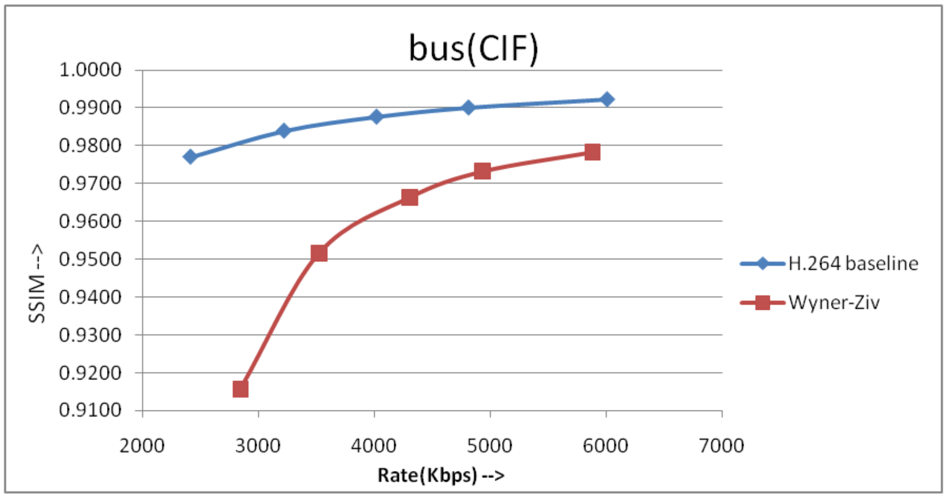

Figure 5.33 SSIM vs. bit rate for H.264 baseline and WZ codec, for the bus (CIF) sequence

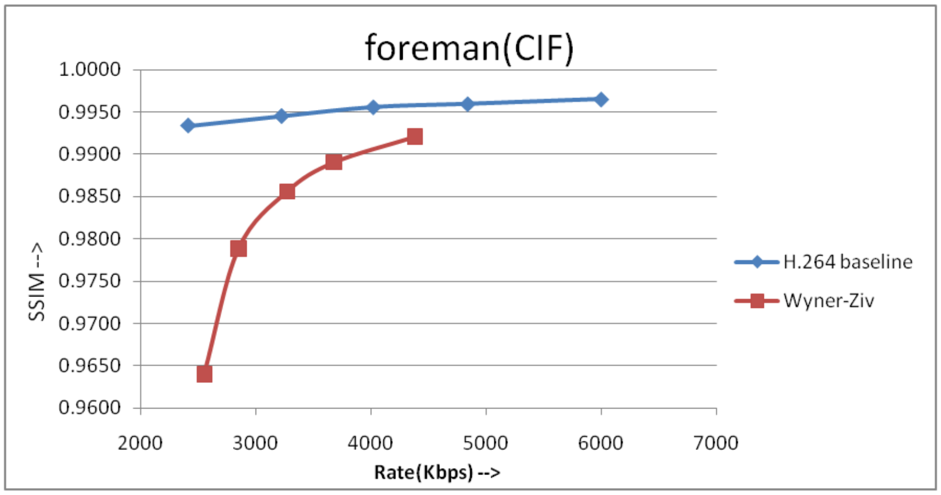

Figure 5.34 SSIM vs. bit rate for H.264 baseline and WZ codec, for the foreman (CIF) sequence

# CHAPTER 6

# CONCLUSIONS

In this thesis basic Wyner-Ziv encoder and decoder are implemented and the different methods for Side Information(SI) generation are considered. In section 5.1 different methods for SI generation and in section 5.2-4 different values for the parameters involved in SI generation are then compared using quality of the video at the decoder side. Also in section 5.5, Wyner-Ziv coding is compared with the H.264 coding which is traditional state of the art video coding standard. The following conclusions can be made based on the experiments carried out,

1. The H.264 based quarter-pixel motion estimation and intra prediction provide superior performance over other H.263 based ME where ME is based on half-pixel motion estimation.
2. ME block size of 16x16 is better suited in terms of video quality over smaller block sizes of 8x8 and 4x4 for video resolution tested(QCIF, SIF and CIF). The smaller the block size the probability of correctly identifying the location of moving object is reduced and this causes drop in performance. The higher block sizes may yield better results for higher resolution where the size of the objects will be comparatively bigger.
3. Higher ME search range produces better results and a search range of 16 (Figure 5.14) is suitable for the video resolutions tested (QCIF, SIF and CIF). At higher resolution, higher ME search range may be needed as the amount of motion will be comparatively large.
4. The quality of the video drops sharply as the key frame distance is increased. This is a severe issue because increasing key frame distance is essential to decrease the bit rate. In key frames only intra-prediction is done which reduces the spatial redundancy but no inter prediction is done hence the temporal redundancy is not removed. This causes key frames to consume more bits while encoding hence the number of key frames should be kept as small as possible.

In traditional video codecs the key frame or intra frame distance up to 50 is used to reduce the bit rate where as in WZ coding even a key frame distance of 5 severely degrades the performance.

5. The H.264 coding performs much better than WZ coding for the same motion estimation and intra prediction. This indicates that the traditional video coding methods have superior edge over WZ coding in terms of video quality at the same bit rate. The WZ coding has the same theoretical rate-distortion limits as that of traditional video coding but in practice it is difficult to achieve the theoretical bound.

## APPENDIX A

## PEAK SIGNAL TO NOISE RATIO (PSNR)

PSNR is measure of video quality often used in image and video compression. The PSNR of a compressed frame I(i, j) given the reference frame $\hat{I}$(i, j) can be calculated as,

$$\text{MSE} = \frac{1}{M*N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[I(i,j) - \hat{I}(i,j)]^2$$

$$\text{PSNR} = 10log_{10}(\frac{R^2}{MSE})$$

Where, the size of the frame is MxN and R is the maximum value in the image data type. The case where image data is stored in 8 bits, R = 255. The quality of the compressed image is considered high if the PSNR is high. In this way different compression methods can be compared. In this thesis PSNR is calculated for

- H.264 compressed stream
- Wyner-Ziv compressed stream

The uncompressed YUV data is used as the reference while calculating the PSNR.

## APPENDIX B

## STRUCTURAL SIMILARITY (SSIM)

PSNR is a good measure of video quality but it does not account for human visual perception. On the other hand SSIM [18] measures the video quality on the assumption that human visual perception is highly adapted for extracting structural information from a scene. This method compares two images based on,

- Luminance
- Contrast
- Structure

The SSIM measurement involves comparison of two images in terms of above three aspects. The SSIM is obtained in such a way that each of the metrics is compared independent of one another. This is illustrated in Figure B.1. The contrast comparison between signals x and y is done after effect of luminance is removed and similarly the structural comparison is done after removing effect of luminance and contrast.

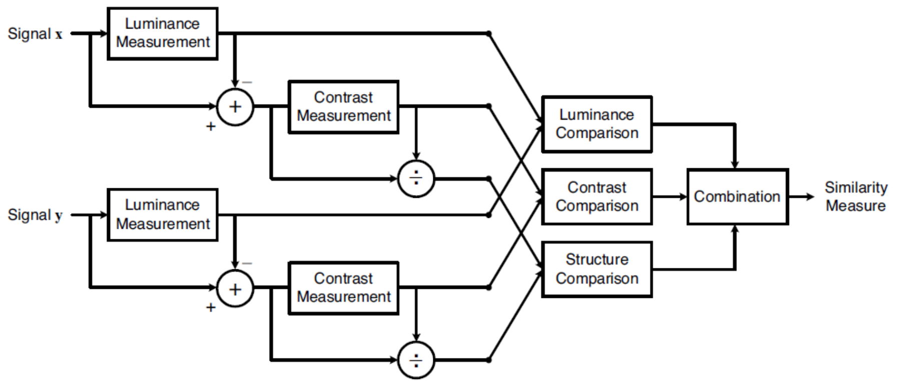

Figure. B.1 Diagram of the structural similarity (SSIM) measurement system [18]

The SSIM can be calculated as,

$$\mathrm{SSIM}(x, y) = \frac{(2m_x m_y + C_1)(2\sigma_{xy} + C_2)}{(m_x{}^2 + m_y{}^2 + C_1)(\sigma_x{}^2 + \sigma_y{}^2 + C_2)}$$

where,

$m_x$ = mean of x
$m_y$ = mean of y
$\sigma_x^2$ = variance of x
$\sigma_y^2$ = variance of y
$\sigma_{xy}$ = correlation coefficient for x and y

The constants $C_1$ and $C_2$ are calculated as $C_1 = (K_1L^2)$ and $C_2 = (K_2L^2)$ where $K_1 << 1$ and $K_2 << 1$.

## APPENDIX C

## YUV TEST SEQUENCES USED

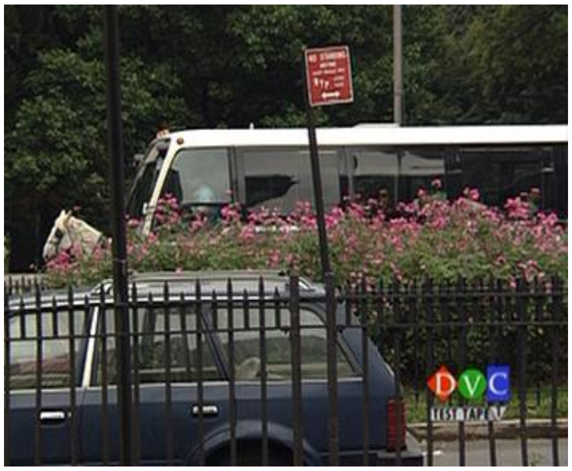

Figure C.1 Bus (CIF) [28]

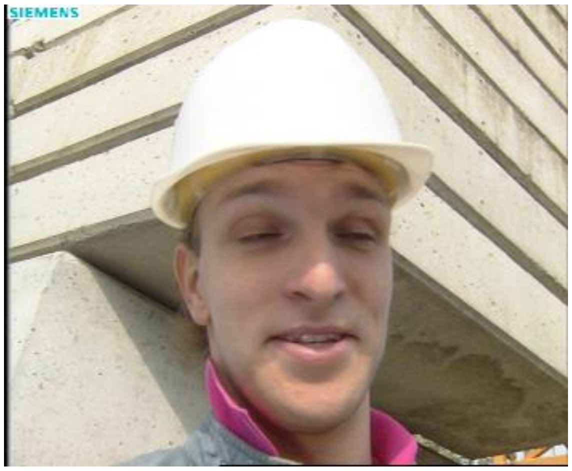

Figure C.2 Foreman (CIF) [28]

Figure C.3 Coastgaurd (CIF) [28]

Figure C.4 Stefan (CIF) [28]

Figure C.5 Garden (SIF) [29]

Figure C.6 Mobile_calendar (SIF) [29]

Figure C.7 Tennis (SIF) [29]

Figure C.8 Akiyo (QCIF) [28]

Figure C.9 Coastgaurd (QCIF) [28]

Figure C.10 Container (QCIF) [28]

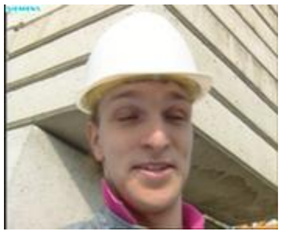

Figure C.11 Foreman (QCIF) [28]

Figure C.12 Mobile_calendar (QCIF) [28]

## REFERENCES

1. E. Peixoto, R. L. de Queiroz, and D. Mukherjee, “Mobile video communications using a Wyner-Ziv transcoder,” Proc. SPIE 6822, VCIP, 68220R Jan. 2008.
2. A. Aaron, E. Setton and B. Girod, "Towards practical Wyner-Ziv coding of video," Proceedings. 2003 International Conference on Image Processing, 2003. ICIP 2003., vol.3, pp. III-869-872, 14-17 Sept. 2003.
3. K. R. Rao and J. J. Hwang, Techniques and standards for image, video, and audio coding, Prentice Hall PTR, 1996.
4. Jin-Soo Kim, "Brief overview of Wyner-Ziv CODEC" (Private Communication)
5. A. Aaron, D. Varodayan, and B. Girod, “Wyner-Ziv residual coding of video,” Proc. International Picture Coding Symposium, Beijing, P. R. China , April 2006.
6. T. Wiegand and G.J Sullivan, “The H.264/AVC video coding standard”, IEEE SP Magazine, vol. 24, pp. 148-153, March 2007.
7. G. J. Sullivan, P. Topiwala, and A. Luthra, "The H.264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions", SPIE Conf. on applications of digital image processing XXVII, vol. 5558, pp. 53-74, Aug. 2004.
8. S.K. Kwon, A. Tamhankar, and K.R. Rao “Overview of H.264/MPEG-4 Part 10” J. VCIR, Vol. 17, pp. 186-216, April 2006, Special Issue on “Emerging H.264/AVC Video Coding Standard,”.
9. A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," IEEE Trans., Information Theory, vol.22, pp. 1-10, Jan 1976.
10. D. Slepian and J. Wolf, “Noiseless coding of correlated information sources,” IEEE Transactions on Information Theory 19, pp. 471–480, July 1973.
11. D. Varodayan, A. Aaron and B. Girod, "Rate-adaptive distributed source coding using low-density parity-check codes," Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005, pp. 1203-1207, Oct. 28 – Nov. 1, 2005.
12. Z. Li and E.J. Delp, "Wyner-Ziv video side estimator: conventional motion search methods revisited," IEEE International Conference on Image Processing, 2005. ICIP 2005, vol.1, pp. I-825-8, 11-14 Sept. 2005.
13. L. Liu and E. J. Delp, "Wyner-Ziv video coding using LDPC codes," Proceedings of the 7th Nordic Signal Processing Symposium, 2006. NORSIG 2006.
14. D. Kubasov, K. Lajnef and C. Guillemot, "A hybrid encoder/decoder rate control for Wyner-Ziv video coding with a feedback channel," IEEE 9th Workshop on Multimedia Signal Processing, 2007. MMSP 2007., pp.251-254, 1-3 Oct. 2007.
15. C. Brites and F. Pereira, "Encoder rate control for transform domain Wyner-Ziv video coding," IEEE International Conference on Image Processing, 2007. ICIP 2007., vol.2, pp.II -5-II -8, 16-19 Sept. 2007
16. A. Roca, et al, "Rate control algorithm for pixel-domain Wyner-Ziv video coding ," Proc. SPIE, vol. 6822, 68221T (2008).
17. D. Mukherjee, “Optimal parameter choice for Wyner-Ziv coding of Laplacian sources with decoder side information,” HP Labs Technical Report HPL-2007-34, 2007.
18. Z. Wang, et al, "Image quality assessment: From error visibility to structural similarity," IEEE Trans., Image Processing, vol.13, pp. 600-612, April 2004

19. I. Richardson, “The H.264 advanced video compression standard,” Hoboken, NJ: Wiley, 2010.
20. D. Rebollo-Monedero, S. Rane, A. Aaron and B. Girod, "High-rate quantization and transform coding with side information at the decoder," EURASIP Signal Processing Journal, Special Issue on Distributed Source Coding.
21. S.-K Kwon, A. Tamhankar and K.R. Rao, ‘Overview of H.264 / MPEG-4 Part 10” ISME, Hong Kong, Oct. 2004.
22. JVT documents
ftp://standards.polycom.com
http://ftp3.itu.ch/av-arch/jvt-site/
23. M. Lee, A. Moore, “H.264 Encoder Design“, Group 3, May 17, 2006
24. J. S. Park and H. J. Song, “Selective Intra Prediction Mode Decision for”, World Academy of Science, Engineering and Technology 13 2006.
25. B. M.J. Leiner, “LDPC Codes – a brief Tutorial”, April 8, 2005
26. G. Cote, B. Erol, M. Gallant, and F. Kossentini, “H.263+: Video coding at low bit rates,” IEEE Circuits and Systems for Video Technology 8, pp. 849–866, November 1998
27. (2008, August) H.264/avc JM reference software. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. [Online]. Available: http://iphome.hhi.de/suehring/tml/
28. YUV test sequences: http://trace.eas.asu.edu/yuv/index.html
29. YUV test sequences: http://www.cipr.rpi.edu/resource/sequences/sif.html

## BIOGRAPHICAL INFORMATION

Subrahmanya Venkatrav received his Bachelor of Engineering degree in Electronics and Communication Engineering from National Institute of Technology, Karnataka, in 2004. His current research interests include multimedia coding, video compression and embedded signal processing. He is currently pursuing his Master's degree in Electrical Engineering at The University of Texas at Arlington. He is a member of the multimedia processing research group, guided by Dr. K. R. Rao.