Winter 1-31-1994

A hybrid low bit-rate video codec using subbands and statistical modeling

Ferhat Cakrak
New Jersey Institute of Technology

ABSTRACT

A Hybrid Low Bit-rate Video Codec Using Subbands and Statistical Modeling

by
Ferhat Cakrak

A hybrid low bit-rate video codec using subbands and statistical modeling is proposed in this thesis. The redundancy within adjacent video frames is exploited by motion estimation and compensation. The Motion Compensated Frame Difference (MCFD) signals are decomposed into 7 subbands using 2-D dyadic tree structure and separable filters. Some of the subband signals are statistically modeled by using the 2-D AR(1) technique. The model parameters provide a representation of these subbands at the receiver side with a certain level of error. The remaining subbands are compressed employing a classical waveform coding technique, namely vector quantization (VQ).

It is shown that the statistical modeling is a viable representation approach for low-correlated subbands of MCFD signal.The subbands with higher correlation are better represented with waveform coding techniques.

A HYBRID LOW BIT-RATE VIDEO CODEC USING SUBBANDS AND STATISTICAL MODELING

by
Ferhat Cakrak

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

January 1994

APPROVAL PAGE

A HYBRID LOW BIT-RATE VIDEO CODEC USING SUBBANDS AND STATISTICAL MODELING

Ferhat Cakrak

Dr. Ali N. Akansu, Thesis Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Nirwan Ansari, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Zoran Siveski, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Ferhat Cakrak

Degree: Master of Science in Electrical Engineering

Date: January 1994

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1994

- Bachelor of Science in Electrical Engineering,
  Technical University of Istanbul, Istanbul, Turkey, 1989

Major: Electrical Engineering

This thesis is dedicated to my family

ACKNOWLEDGMENT

I would like to express my gratitude to Dr. A. N. Akansu for his valuable contribution, advice, patience and understanding. I also appreciate his support and encouragement during the entire research period.

I am very grateful to Dr. Nirwan Ansari and Dr. Zoran Siveski for their effort and time in reviewing this work.

I also would like to thank the members of the Center for Communications and Signal Processing Research at New Jersey Institute of Technology and to all my friends for their help and support.

TABLE OF CONTENTS

LIST OF FIGURES

Figure		Page

CHAPTER 1
INTRODUCTION

Interest in digital image processing has significantly increased over the past two decades. Advances in signal and image processing techniques allow sophisticated image processing algorithms to be realized in real time at a reasonable cost. However, the storage capacity and bandwidth of available communication channels have always been two major limitations. Since the amount of data in images is immense the compression of data has been of a great interest. Hence, there have been several image compression techniques proposed in the literature and the problem is still being actively pursued.

In video transmission and storage applications, one mostly has to deal with the images of moving objects. The motion occurring in such a multi-frame sequence is due to translation and rotation of objects with respect to the camera or moving camera and moving object case[1].

In video frames, we encounter the redundancy in temporal and spatial dimensions among the adjacent frames. A satisfactory data compression technique should not only remove temporal and spatial redundancies but also give a good visual perspective for a certain level of image quality [1]. The more visual quality we can sacrifice, the lower the bit-rate we need to transmit or to store an image.

There are several video coding techniques which provide satisfactory performance for compression. Most of these techniques employ transform coding of motion prediction error which is also known as the motion compensated frame difference ($MCFD$) signal. The $MCFD$ signal has been studied by several researchers and it is still being studied extensively in order to achieve better compression and visual performance. The statistical model based and subband coding techniques are the ones combined in the proposed codec structure of this thesis.

In model based image coding, an image or some regions of an image are statistically modeled and the model parameters are used for the representation. At the transmitter, the the statistical model parameters are estimated by analyzing the image. Then, these parameters are quantized and sent to receiver side. At the receiver side, the image is reconstructed using quantized model parameters. Although the modeling of speech is useful and works well, the modeling of images has not been satisfactory. Therefore, model-based image coding technique is still at the research stage and more needs to be done[3].

Subband coding, one of the most powerful waveform coding techniques, has found its applications in speech and image processing. To compress the data, the signal is divided into a set of uncorrelated frequency bands and subband signals are encoded after an optimal bit allocation.

In this thesis, the MCFD signal is studied by using subband coding and statistical modeling in subbands.

Chapter 2 deals with the motion compensated video coding and the statistical evaluation of MCFD signals. In Chapter 3, autoregressive source models and AR(1) modeling are studied. Chapter 4 deals with the theory of subband signal analysis and filter banks. In Chapter 5, vector quantization in subbands is covered. In Chapter 6, the experimental results are presented. The conclusions are given in Chapter 7.

CHAPTER 2

MOTION COMPENSATED VIDEO CODING

2.1 Introduction

In this chapter, the general idea behind the motion compensated video coding technique is given. The algorithms used for prediction are classified and explained briefly. In Section 2.2, the theory of block matching algorithms is given in detail. Section 2.3 deals with the statistical features of motion compensated frame difference (MCFD) signals.

Any good video coding technique should remove not only the temporal but the spatial redundancies. To eliminate redundancy in a video sequence, interframe predictive coding, one of the most powerful video coding techniques, is widely used. In a typical interframe coding process, the present video frame is predicted based on frame to frame motion and the previous frame. The prediction error, MCFD, along with motion information is transmitted. At the receiver side, the MCFD signals are decoded and added to the motion based prediction of the frame. The main feature in this coding technique is to predict the current frame based on the previous one. The better prediction gives the smaller error signal and the smaller transmission bit rate [2].

The video scenes usually contain moving objects. The motion in a typical video sequence is due to the rotation and translation of the objects. The current frame F_K is predicted by using the previous frame F_{K-1} and frame to frame motion. This process is called motion compensation, and the difference between the current frame and its motion compensated prediction is called motion compensated frame difference (MCFD) signal. Block diagram of motion compensated video coding technique is given in Figure 2.1.

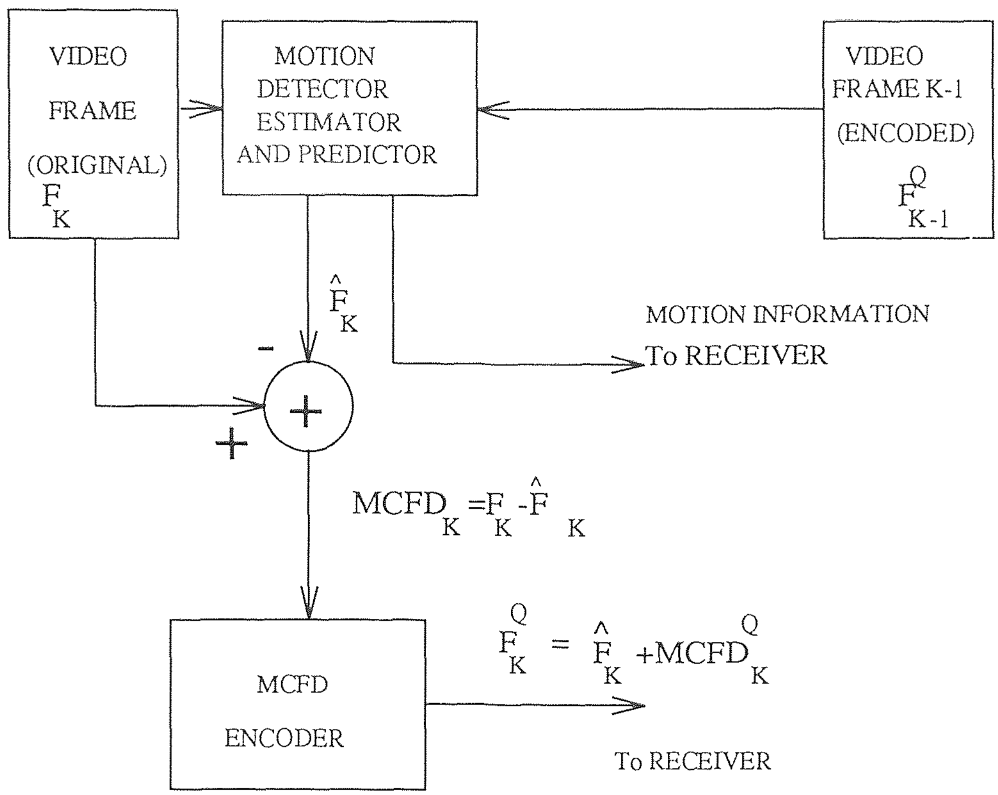


Figure 2.1 Block Diagram of Motion Compensated Video Sequence Coding Structure.

To predict the current frame by using the previous frame, there are several motion estimation algorithms proposed in the literature. Most common algorithms used in practice are as follows:

- Block Matching Algorithm
- Pel (pixel) Recursive Algorithm
- Knowledge Based Algorithm

First two algorithms use the 2D information of the successive video frames. The block matching algorithm tries to estimate the displacement vectors by means of comparing the gray levels of adjacent video frames in block fashion. On the other hand, pel recursive algorithm uses the coded neighbour pixels to predict the displacement of each pixel[2]. These two algorithms are based on the following assumptions:

- The motion of the moving objects is only translation.
- Intensity (illumination) is the same in spatial and temporal dimensions.
- Masking between objects and uncovered background is neglected.

The knowledge based algorithm employs the 3-D motion constraints. Although this algorithm is quite popular in model based coding, it is not quite practical due to the computational load and complexity of the algorithm.

2.2 Block Matching Algorithm (BMA)

In the current technology, block matching algorithms are widely used due to the their simplicity and effectiveness. The block matching algorithm (BMA) divides an image into fixed or variable size rectangular blocks, and assumes that each block can be represented by a displacement vector $D = (d_x, d_y)$ as shown in Figure 2.2. In order to maintain the validity of the assumption, block sizes are kept small, such as 8x8 or 16x16[2].

In this study, the motion compensation is based on the block matching algorithm, which can be implemented by using fixed or variable size blocks. In our approach, each video frame is divided into 8x8, fixed size, blocks. Each 8x8 block in current frame is compared with all possible blocks within a certain search region in the previous frame. The best matching block is found by the following procedure.

The motion detector compares each pixel of a predefined image block in the present frame K, with the corresponding pixel values of the previous frame $K-1$. If the condition given by Eq. 2.1 is satisfied, which means difference is above the predetermined threshold value, then the pixel m, n of block i, j in frame K is assumed moving. For an 8x8 block, if the number of moving pixels is above the predetermined

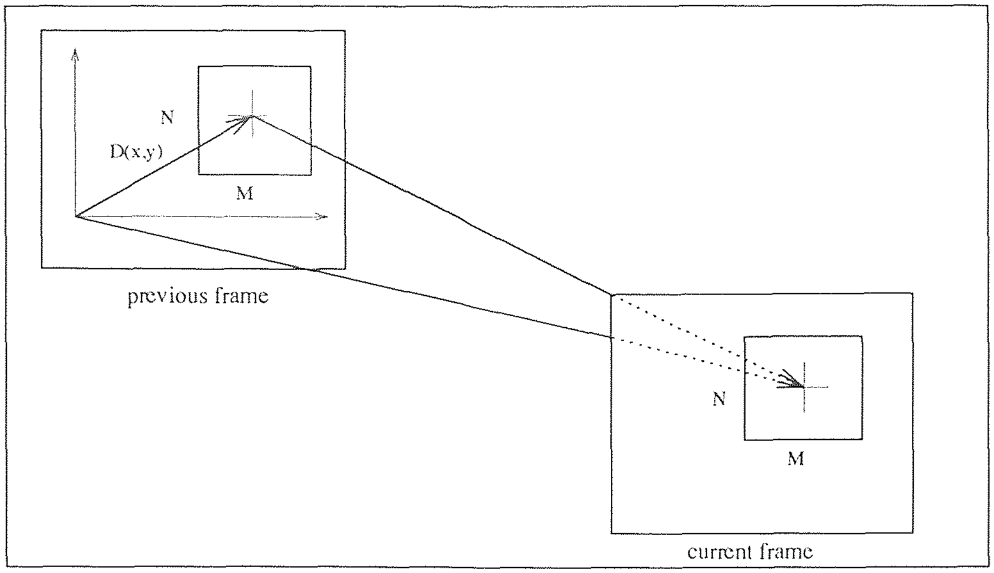


Figure 2.2 Block matching motion estimation

threshold value N_0 as given in Eq.2.2, then that block is assumed moving.

$$|P_{i,j}^{K}(m,n) - P_{i,j}^{K-1}(m,n)| \geq T_0 \qquad m,n = 1,2,..,8 \tag{2.1}$$

$$[number\ of\ moving\ pixels\ in\ an\ 8x8\ block] \geq N_0 \tag{2.2}$$

T_0 and N_0 are the predefined threshold values for each pixel and each 8x8 block respectively. These values can be adjusted depending on the application. In this study, $T_0 = 3$, $N_0 = 10$, have been found suitable values for the experimental video sequences. If the block has a motion, then motion estimation and compensation procedure is performed.

The BMA tries to find the best match for each 8x8 block belonging to the present frame K, using a predefined search region in the previous video frame. Predefined search region size has been taken as $(8 + 2p)$x$(8 + 2p)$ and fixed. It is assumed that the maximum displacement between two 8x8 blocks in two consecutive video frames is $\mp p$ pixels in two dimensions. For a video conferencing environment $p = 6$ is used. There are $(2p + 1)$x$(2p + 1)$ different blocks in search region in which each block is a candidate to be the correct displacement.

A comprehensive search algorithm scans all the candidate search points in the search area for the best match. The best match is found by minimizing the distortion measurement, like mean square error (MSE), or by maximizing a correlation feature, like the cross-correlation function, of the two blocks [2]. In Practice there are several fast search algorithms. Independent orthogonal search algorithm, a computationally efficient search algorithm, is used in this study [1].

Number of thresholded absolute difference ($NTAD$) given in Eq.(2.3) is employed as the objective function.

$$NTAD_{k,l} = \sum_{m=1}^{8} \sum_{n=1}^{8} [f(T_1, |B_{i,j}^{K}(m,n) - S_{i,j}^{K-1}i(m+k, n+l)|)] \quad k,l = 1,2,...,13 \quad (2.3)$$

where

$$f(T_1, D) = \begin{cases} 1 & T_1 \leq D \\ 0 & T_1 > D \end{cases}$$

and S_{ij}^{K-1} is the search region for block $B_{ij}^{K}(m,n)$. The best match is found by minimizing $NTAD(k,l)$. The parameter value $T_1 = 3$ is found the best for the video sequences used in this study.

After determining the best match, the image block B_{ij}^{K} is predicted as $\hat{B}_{ij}^{K}$ based on the corresponding 8x8 block in the previous video frame F^{K-1}. This process is repeated for all the blocks and the prediction of the current frame $\hat{F}_K$ is obtained as seen in Figure 2.1.

The prediction error, which is basically the difference signal between the original frame and its prediction, is encoded and transmitted to the receiver side along with the motion information to reconstruct the current frame F_K. The prediction error which is also called motion compensated frame difference $(MCFD)_K$ signal is given in Eq.(2.4) for the MXN video image sequences.

$$MCFD_K(i,j) = F_K(i,j) - \hat{F}_K(i,j) \qquad \begin{array}{l} i = 1, \ldots\ldots, M \\ j = 1, \ldots\ldots, N \end{array} \qquad (2.4)$$

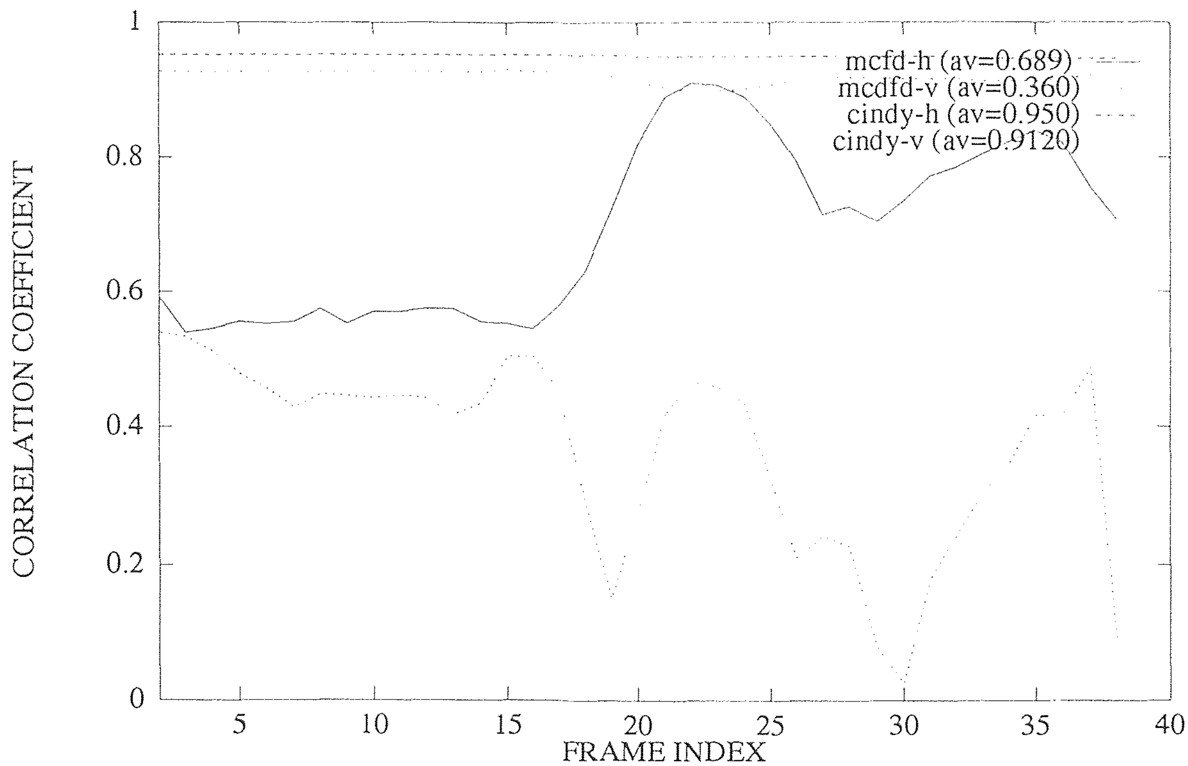


Figure 2.3 Frame by frame variation of the correlation coefficients for the test sequence "CINDY"

2.3 The Motion Compensated Frame Difference Signal (MCFD)

Although the motion information of the motion compensated video coding technique has to be encoded lossless, the MCFD signal may be encoded by using any entropy reduction technique.Transform coding, hybrid coding, and some other source coding techniques have been used to encode the MCFD signals.

It is well known that the performance of the transform coding decreases significantly for the low correlated signal sources such as MCFD signals. Therefore, transform coding is not a good choice for this kind of signals[1]. For the video test sequence "CINDY", frame by frame variation of the average first order horizontal and vertical correlation coefficients are given in Figure 2.3. It is seen from the figure that MCFD signals are low correlated.

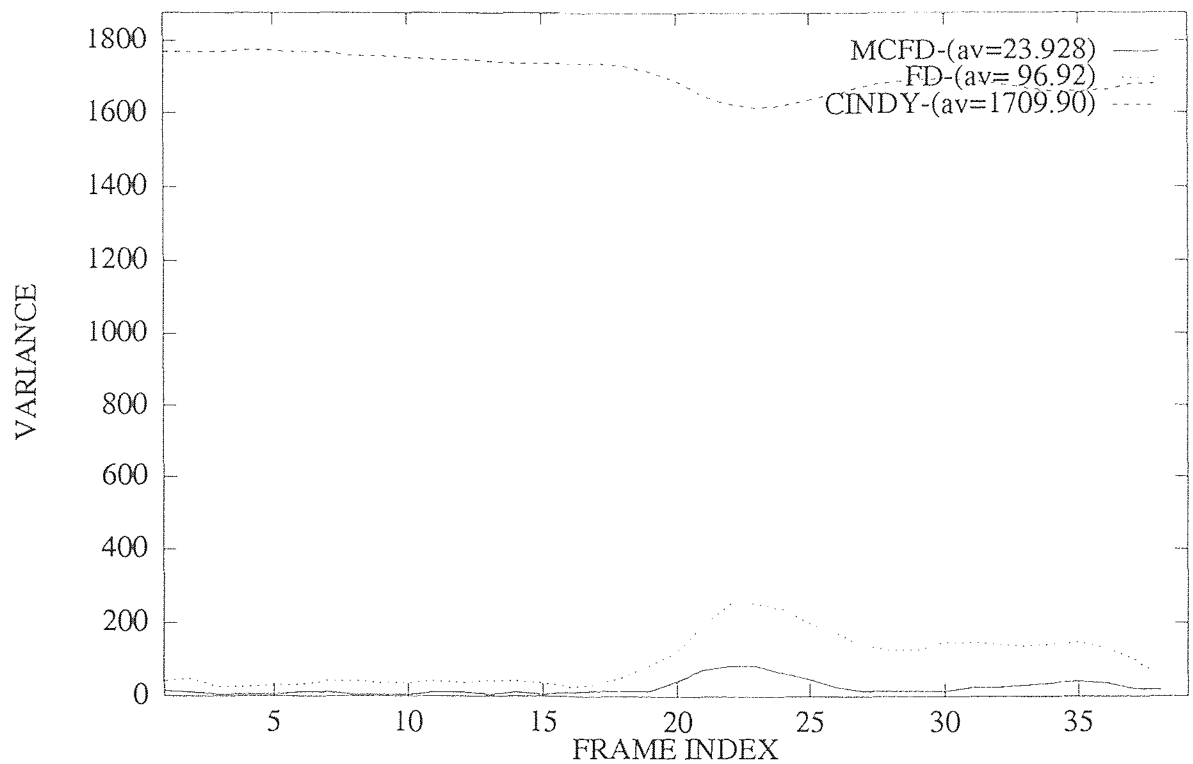


Figure 2.4 Frame by frame variation of variances for the test sequence "CINDY"

The variance of the $MCFD$ signals for first forty frames of the test sequence are displayed in Figure 2.4. It is seen that variance of the motion compensated frame difference signal for the first forty frames of the test sequence "CINDY" is just about 25% of the direct frame to frame difference signal.

There are several contributors to the prediction error (MCFD) signal. All types of motion is approximated by translation. Additionally, using the encoded version of previous frame for prediction brings the effects of the quantization noise into the progress.Furthermore, the threshold values used, T_0, T_1,N_0, are not global optimum values. Last, no abrupt scene change is included in this study. Hence,the effects of abrupt scene changes are not seen here.

CHAPTER 3

STATISTICAL MODEL BASED IMAGE CODING TECHNIQUES

3.1 Introduction

It is highly desirable to define signal sources by a set of statistical parameters. These parameters are used in transmission and storage applications for modeling of source characteristics. The main objective here is to use as few parameters as possible to represent a signal source, keeping certain level of signal quality for a given application. As mentioned earlier in Chapter 1, in statistical model based image coding, an image or a portion of it is statistically modeled and model parameters are quantized and sent to the receiver side to reconstruct the image. Although the statistical modeling works well in speech coding applications, it does not give satisfactory results in still frame image coding applications. Therefore, it is still an active research area and more needs to be done.

Autoregressive modeling is widely used in 1-D applications like linear predictive coding (LPC) of speech. Although its performance is not satisfactory for 2-D applications, like still frame image modeling, it is a reasonable technique to represent MCFD signals. Hence, it can be used for modeling of the MCFD signals, i.e., by employing the subband coding technique some frequency bands of the MCFD signal can be statistically modeled.

In this chapter, Section 3.2 covers autoregressive (AR) processes. In Section 3.3, first-order autoregressive, $AR(1)$, process is studied. In Section 3.4, AR(1) modeling technique for images and the statistics of test images are presented.

3.2 Autoregressive (AR) Process

An autoregressive (AR) process is generated by passing the white noise $\eta(n)$ innovations through an all-pole filter. A wide-sense zero-mean white noise process

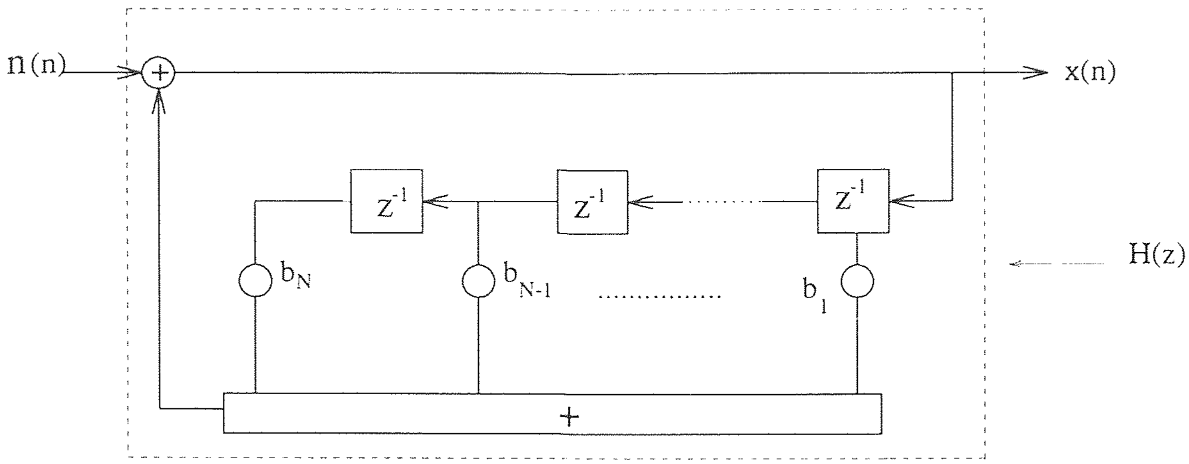


Figure 3.1 Filter Model of AR(N) Process

and its spectrum are defined as;

$$S_{NN}(e^{j\omega}) = \sigma_N^2 \tag{3.1}$$

where σ_N^2 is the variance of a zero mean, wide-sense stationary white noise sequence. Its autocorrelation sequence is given as

$$R_{NN}(m) = E[\eta(n)\eta(n+m)] = \sigma_N^2\delta(m) \tag{3.2}$$

where

$$\delta(m) = \begin{cases} 1 & for \quad m = 0 \\ 0 & otherwise \end{cases}$$

is the kronecker delta function. For any shift m there is no correlation between the samples of the white noise process and it has a flat power spectral density function as seen in Eq.(3.1).

Filter used in this process is called all-pole since it has N multiple zeros at z=0 as seen in Eq.(3.3),

$$H(z) = \frac{1}{1 - \sum_{i=1}^{N} b_i z^{-i}} = \frac{z^N}{z^N - \sum_{i=1}^{N} b_i z^{N-i}} \tag{3.3}$$

The difference equation generating the AR process $\{x(n)\}$ is given by

$$x(n) = \eta(n) + \sum_{i=1}^{N} b_i x(n-i) \tag{3.4}$$

The process $\{x(n)\}$ is called $AR(N)$ or Nth order Markov process. $\{b_i\}$ are the correlation coefficients. The filter model realizing an $AR(N)$ process is given in Figure 3.1.

The impulse response of an all-pole filter is in infinite duration. However, the autocorrelation function (acf) can be calculated recursively for the given set of prediction coefficients b_i ; $i = 1, 2, ..., N$. This is done by multiplying $x(n)$ in Eq.(3.4) with $x(n-m)$ and taking the expectations of both sides. Here, we should note that the white noise innovations $\eta(n)$ are uncorrelated with its past outputs by definition. As a result, we have the following equations

$$E[\eta(n)x(n-m)] = 0 \qquad for \quad k > 0 \tag{3.5}$$

where $E[.]$ donates expectation and

$$\sigma_x^2 = E\{|x(n)|^2\} = R_{xx}(0) = \sum_{i=1}^{N} b_i R_{xx}(i) + \sigma_\eta^2 \tag{3.6}$$

where σ_x^2 is the signal power. The recursive relation of autocorrelation sequence is given as

$$R_{xx}(m) = \sum_{i=1}^{N} b_i R_{xx}(m-i) \qquad m > 0 \tag{3.7}$$

The all-pole model leads to N unknowns and N linear equations. These equations can be solved by using Levinson algorithm or the Cholesky decomposition[4][5].

3.3 First-order Autoregressive AR(1) Process

The first-order Markov or AR(1) process, with zero mean, is obtained easily from Eq.(3.4) with N=1 and $b_1 = \rho$. Thus, we have the difference equation of AR(1) source model in time as

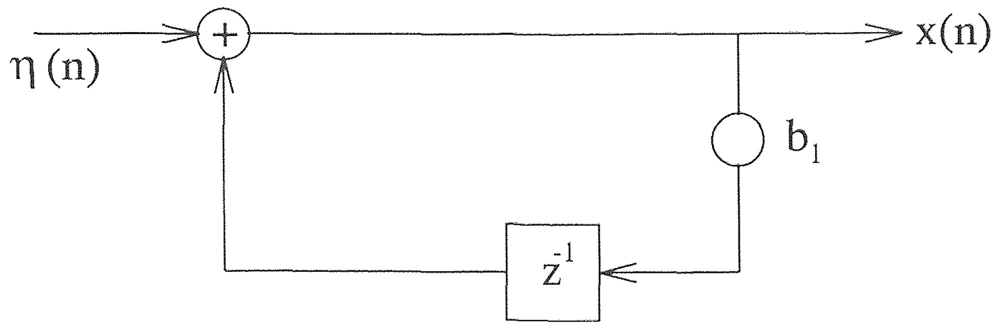


Figure 3.2 Filter Model of AR(1) Process

$$x(n) = \rho x(n-1) + \eta(n) \tag{3.8}$$

where ρ is the first order correlation or prediction coefficient and $\{\eta(n)\}$ is the white noise sequence as given in Eqs.(3.1) and (3.2). The corresponding first-order filter function is found as

$$H(z) = \frac{1}{1-\rho z^{-1}} \tag{3.9}$$

with the frequency response

$$H(e^{j\omega}) = H(z)_{z=e^{j\omega}} = (1-\rho e^{-j\omega})^{-1} \tag{3.10}$$

and the unit sample response

$$h(n) = \rho^n \qquad\qquad n = 0, 1 \ldots\ldots\ldots \tag{3.11}$$

The filter diagram for the first-order autoregressive process is given in Figure 3.2 where b_1 denotes the first-order correlation or prediction coefficient.

The autocorrelation function of an AR(1) signal is found as

$$R_{xx}(m) = \sigma_x^2 \rho^{|m|} \qquad\qquad m = 0, \mp, 1, \mp, 2, \ldots. \tag{3.12}$$

The signal variance is expressed as

$$\sigma_x^2 = R_{xx}(0) = \frac{\sigma_N^2}{1-\rho^2} \tag{3.13}$$

where σ_N^2 is the noise variance.

The power spectral density (psd) of an AR(1) process is given by

$$S_{xx}(e^{j\omega}) = \sigma_N^2 |H(e^{j\omega})|^2 = \frac{1-\rho^2}{1+\rho^2 - 2\rho\cos\omega}\sigma_x^2 \tag{3.14}$$

The process generated by Eq.(3.8) is stationary if the filter is stable. Therefore, $|\rho| < 1$ and

$$\sum_{n=0}^{\infty} |h(n)| = (1-|\rho|)^{-1} < \infty \tag{3.15}$$

Otherwise, a non-stationary process results.

The AR(1) source model is a good analytical tool in a variety of applications, such as speech and statistical model based signal representation.

3.4 First-Order Correlation Models for Images

A random field in which the mean is independent of spatial coordinates and the autocorrelation function is translation-invariant is called *homogeneous*. Properties of an *homogeneous* field is described by

$$E\{m,n\} = \mu(m,n) = constant \tag{3.16}$$

$$E\{x(m,n)x(m+k,n+l)\} = R_{xx}(m,n) \quad for \quad all \quad m,n \tag{3.17}$$

A *homogeneous* random field is white-sense stationary with the following power spectral density and autocorrelation function descriptions;

$$S_{xx}(e^{j\omega_1}, e^{j\omega_2}) = \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} R_{xx}(m,n) e^{-j\omega_1 m} e^{-j\omega_2 n} \tag{3.18}$$

$$R_{xx}(m,n) = [\frac{1}{2\pi}]^2 \int_{-\pi}^{\pi}\int_{-\pi}^{\pi} S_{xx}(e^{j\omega_1}, e^{j\omega_2}) e^{j\omega_1 m} e^{j\omega_2 n} \quad d\omega_1 d\omega_2 \tag{3.19}$$

If all the values of $R_{xx}(m,n)$ are zero in both spatial directions except $R_{xx}(0,0)$, a white noise process results. The autocorrelation function of a white noise process is defined by

$$R_{xx}(m,n) = \sigma_N^2 \delta(m,n) = \begin{cases} \sigma_N^2 & m = n = 0 \\ 0 & otherwise \end{cases} \tag{3.20}$$

The process has a flat power spectral density (psd) function as given in Eq.(3.21).

$$S(e^{j\omega_1}, e^{j\omega_2}) = \sigma_N^2 \tag{3.21}$$

The correlation models in two dimensional sources are divided into two groups; separable and non-separable correlation models[4]. Experimental studies have indicated that natural objects are better represented by non-separable correlation models[5]. On the other hand, artificial images are better represented by the separable correlation models. In this study, separable correlation model is employed. Hence, we are only concentrated on 2-D AR(1) modeling. More information about 2-D correlation models can be found in references [4] and [5].

3.4.1 First-Order Autoregressive AR(1) Source Model for Images

An important autocorrelation model in two dimensions is the first-order autoregressive AR(1) source model which has the autocorrelation function

$$R_{xx}(m,n) = \sigma_x^2 \rho_h^{|m|} \rho_v^{|n|} \qquad m,n = 0, \mp, 1, \mp, 2, ... \tag{3.22}$$

and variance

$$\sigma_x^2 = R_{xx}(0,0) = \frac{\sigma_N^2}{(1-\rho_h^2)(1-\rho_v^2)} \tag{3.23}$$

where m and n are spatial shifts in horizontal and vertical directions, and ρ_h and ρ_v are the corresponding first-order horizontal and vertical correlation coefficients, respectively. The correlation model in Eq.(3.22) is called separable because it can be expressed as the product of two one-dimensional autocorrelations. One can verify that

$$R_{xx}(m,n) = \sigma_x^2 \rho^{|m|+|n|} \qquad if \quad \rho_h = \rho_v = \rho \tag{3.24}$$

A 2-D AR(1) signal can be expressed by the difference equation

$$x(m,n) = \rho_h x(m-1,n) + \rho_v x(m,n-1) - \rho_h \rho_v x(m-1,n-1) + \eta(m,n) \quad (3.25)$$

where $\{\eta(m,n)\}$ is the zero mean, white noise array with the variance σ_N^2. The transfer function of 2-D AR(1) filter in Z-domain is given as

$$H(z_1,z_2) = \frac{1}{1 - \rho_h z_1^{-1} - \rho_v z_2^{-1} - \rho_h \rho_v z_1^{-1} z_2^{-1}} \quad (3.26)$$

Although the real-world images are not stationary, they can be assumed stationary over a small region. By using this assumption, statistical parameters namely ρ_h, ρ_v ,σ_x^2 and $\mu(m,n)$ of each region can be estimated, and these parameters can be used for image representation.

In this thesis, the first-order autoregressive source model is used for the statistical modeling of the MCFD frame in the subbands.

First, the MCFD signal is decomposed into a set of frequency bands (4 and 7 band 2-D filter banks) and then, some of the subband signals are modeled by using AR(1) technique. The following steps are performed for each subband.

The MCFD subband frame is divided into fixed size blocks, i.e., 8x8, 4x4, and for each block, the mean is calculated and subtracted from the respective block. Then, the statistical model parameters, namely ρ_h, ρ_v and σ_x^2, are estimated for each zero mean block. These 4 parameters are quantized and encoded for transmission. At the receiver side, these parameters are used to reconstruct the corresponding image blocks by using Eq.(3.25). The white noise array $\eta(m,n)$ is generated by using the respective local block variances with zero mean. The relationship between local block variances and white noise array variances is given in Eq.(3.23).

After reconstructing each statistically modeled, zero mean, block, mean is added to each respective block to recover the statistically modeled MCFD subband frame.

It is obvious that even in the case of perfect quantization, original signal can not be recovered. Modeling brings some error due to stationarity assumption. Frame by frame average variances of the LH band before and after AR(1) modeling is given in Figure 3.3 for the first forty frames of the test sequence "CINDY" for 4x4 block size.

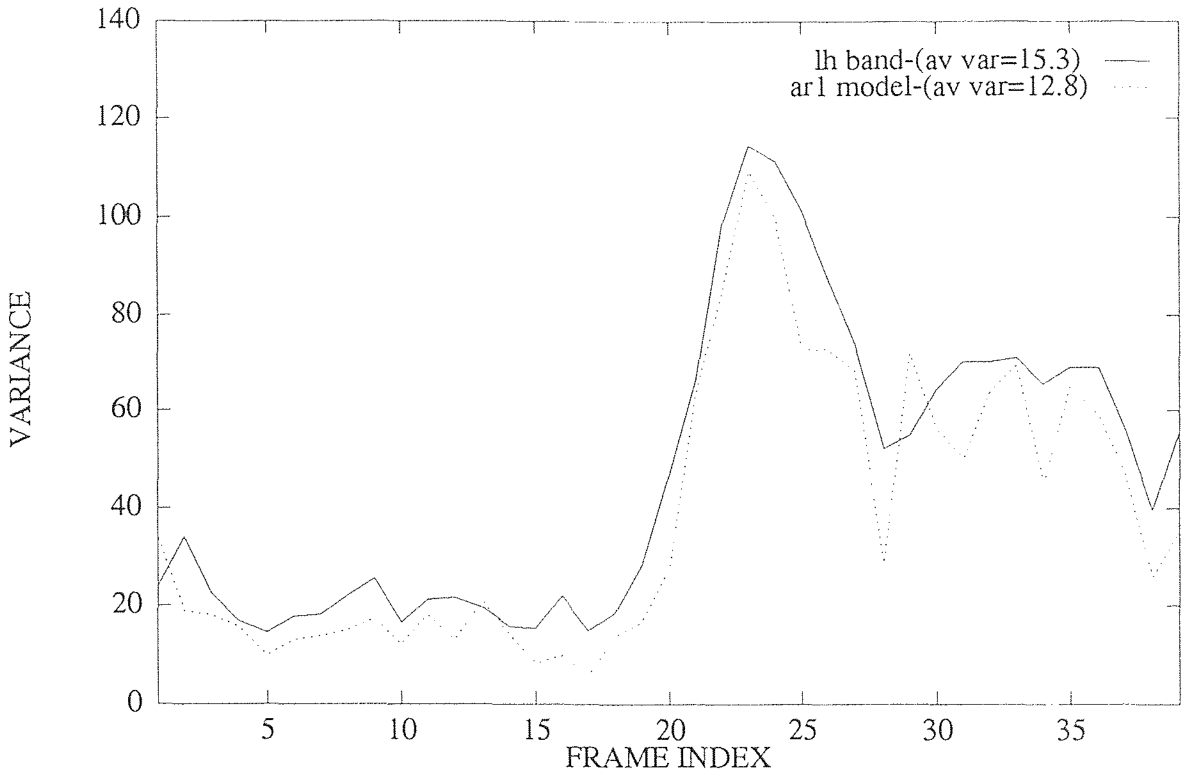


Figure 3.3 Frame by frame variation of variances for the LH band of the test sequence "CINDY'

CHAPTER 4

THEORY OF SUBBAND SIGNAL ANALYSIS AND FILTER BANKS

4.1 Introduction

In recent years, subband coding has been widely used for image and video coding applications. To compress the data, the signal is decomposed into a set of uncorrelated frequency bands and each of these subband signals are encoded for transmission after the optimal bit allocation. At the receiver side these encoded subband signals are decoded to reconstruct the signal.

In general, a subband system can be decomposed into two parts, analysis and synthesis. The analysis part of a subband system consists of anti-aliasing subband filters and downsamplers. The synthesis part consists of upsamplers and interpolation filters.

In this chapter, section 4.2 deals with the main building blocks of a subband system, i.e., downsamplers, upsamplers , anti-aliasing and interpolation filters. The two channel perfect reconstruction quadrature mirror filter (PR-QMF) banks are studied in section 4.3. In section 4.4, M-band tree decomposition is covered along with 4 and 7 band filter banks. Section 4.5 deals with the two dimensional separable filter case which is used in this study.

4.2 Main Building Blocks in Subband Analysis

In this section, main building blocks, i.e., downsamplers, upsamplers, anti-aliasing and interpolation filters, are studied along with their frequency domain characterizations.

4.2.1 Downsamlers and Upsamplers

Figure 4.1 shows the block diagrams of a downsampler and upsampler. The input-output relation of a downsampler with rate M is given by[6]

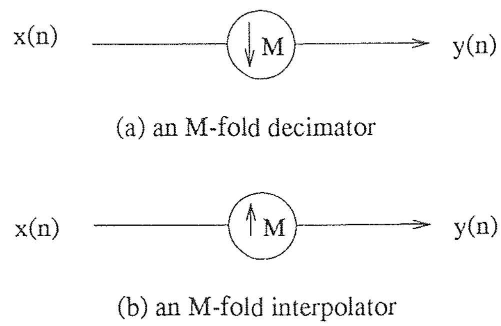


Figure 4.1 Block diagrams for downsampler and upsampler.

$$y(n) = x(Mn). \tag{4.1}$$

Eq.(4.1) shows that the output at time n is equal to the input at time Mn. As a result, only the input samples with the sample numbers equal to multiples of M are retained. The sampling rate reduction process is illustrated in Figure 4.2 for $M = 2$ case.

The input–output relation of an M–fold upsampler is given by

$$y(n) = \begin{cases} x(\frac{n}{M}) & if\ n\ is\ a\ multiple\ of\ M \\ 0 & otherwise \end{cases} \tag{4.2}$$

Eq.(4.2) indicates that the output $y(n)$ is obtained by inserting $M - 1$ zeros between adjacent samples of $x(n)$. This process is shown in Figure 4.3 for $M = 2$.

Although the downsamplers and upsamplers make the system time varying, they are linear systems.

The transform domain description of the upsampler is given by

$$Y(Z) = X(Z^M), \qquad Y(e^{j\omega}) = X(e^{j\omega M}) \tag{4.3}$$

where $z = e^{j\omega}$. The stretching effect of upsampling in time domain corresponds to a compression in frequency domain as shown in Figure 4.4 for $M = 2$. As seen from the figure, $Y(e^{j\omega})$ has $M - 1$ images of the basic spectrum.Consequently, the upsampler causes an imaging effect.

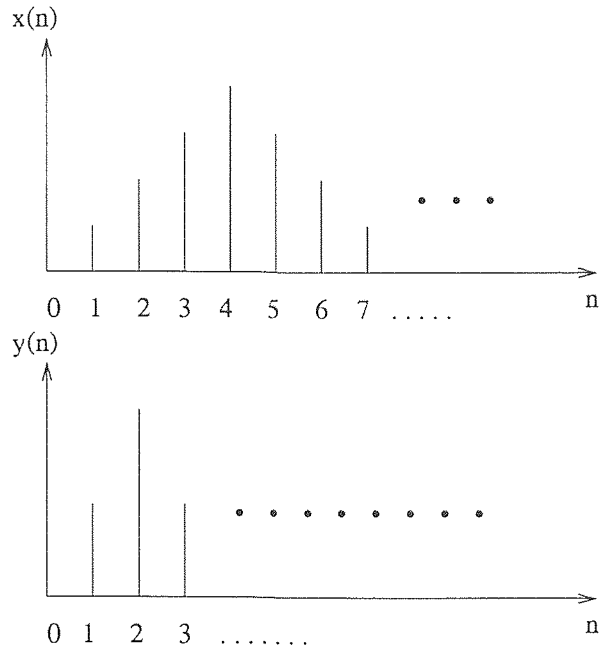


Figure 4.2 Downsampling with M=2.

The transform domain description of the idownsampler is given by

$$Y(Z) = \frac{1}{M}\sum_{k=0}^{M-1} X(Z^{1/M}W^k), \quad Y(e^{j\omega}) = \frac{1}{M}\sum_{k=0}^{M-1} X(e^{j(\omega-2\pi k)/M}) \tag{4.4}$$

where $W = e^{-2\pi j/M}$. For $M = 2$, this equation becomes

$$Y(e^{j\omega}) = \frac{1}{2}[X(e^{j\omega/2}) + X(e^{j\omega/2})] \tag{4.5}$$

As seen above, a downsampler causes compression in time and brings the stretching effect in frequency domain as shown in Figure 4.5 for $M = 2$. Figure 4.5 also shows that spectrum of the the original signal after downsampler contains the $2\pi-$shifted versions of the original spectrum. As a consequence, aliasing effect is observed.

4.2.2 Anti-aliasing Filters

To avoid the aliasing effect of downsampling operation, the downsampler is preceded by a band limiting filter which is called anti$-$aliasing or decimation filter. For example, a low-pass filter with the stopband edge $\omega_s = \frac{\pi}{M}$ can serve as such a filter for the signal downsamled by M as seen in Figure 4.6.

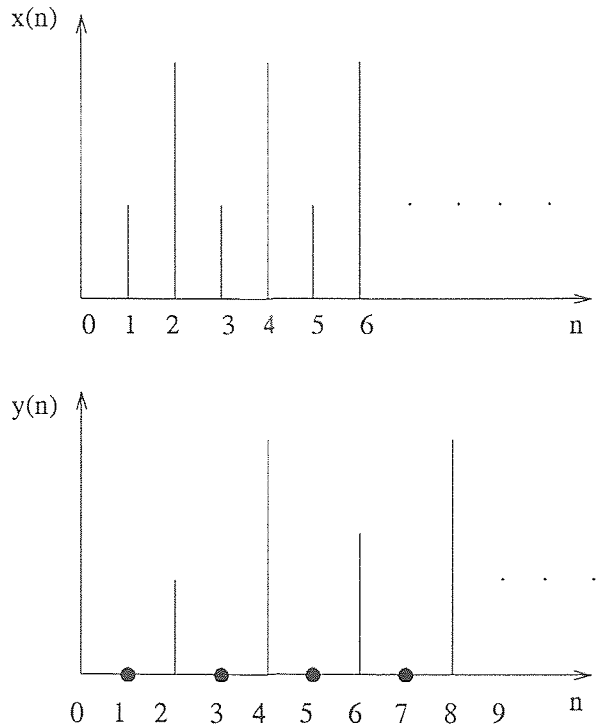


Figure 4.3 Upsampling with M=2.

4.2.3 Interpolation Filters

To eliminate the imaging effects at the output of the upsampler, it is followed by an interpolation filter. The low−pass filter in figure 4.6, again, may serve as an interpolation filter.

4.3 Two Channel Perfect Reconstruction Quadrature Mirror Filter (PR–QMF) Banks

Consider the two channel QMF structure given in Figure 4.7. Based on the Eqs.(4.3) and (4.5), we can express $\hat{X}(Z)$ as[7]

$$\begin{aligned}\hat{X}(Z) &= \frac{1}{2}[H_0(Z)F_0(Z) + H_1(Z)F_1(Z)]X(Z) + \\ &\quad \frac{1}{2}[H_0(-Z)F_0(Z) + H_1(-Z)F_1(Z)]X(-Z)\end{aligned} \tag{4.6}$$

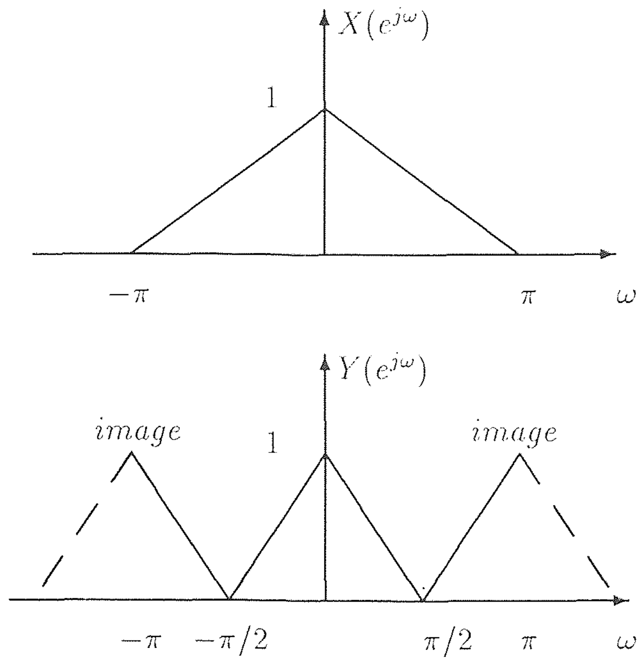


Figure 4.4 Frequency domain representation of the Upsampling by 2, with the input signal (top) and the upsampled signal (bottom).

The second term in Eq.(4.6) represents the effects of imaging and aliasing. These terms can be eliminated simply by choosing the synthesis filters to be

$$F_0(Z) = H_1(-Z), \qquad F_1(Z) = -H_0(-Z) \tag{4.7}$$

When the aliasing effect is eliminated, the QMF bank becomes a time-invariant system with the transfer function

$$T(Z) = \frac{\hat{X}(Z)}{X(Z)} = \frac{1}{2}[H_0(Z)H_1(-Z) - H_1(Z)H_0(-Z)] \tag{4.8}$$

Ideally, $T(Z)$ is desired to be a delay, i.e., $T(Z) = Z^{-n_0}$, so that the reconstructed signal is a delayed version of $x(n)$. Unfortunately, $T(Z)$ is not a delay in general and it represents a distortion overall transfer function. One can express $T(Z)$ in the form of

$$T(Z) = T(e^{j\omega}) = |T(Z^{j\omega})| \; arg[T(e^{j\omega})] \tag{4.9}$$

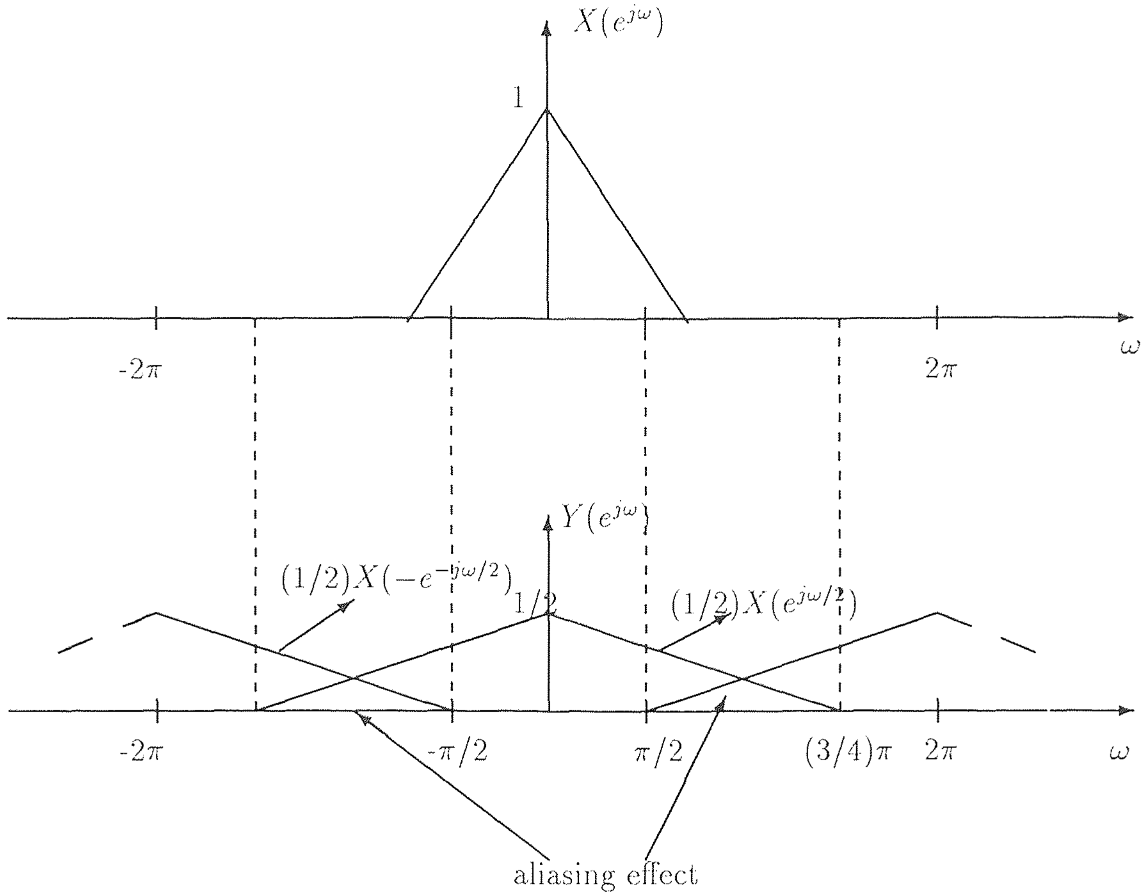
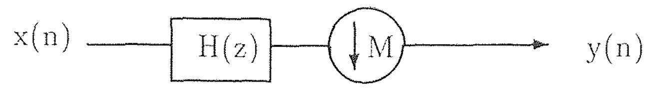


Figure 4.5 Frequency domain representation of the downsampling by 2, with the input signal (top) and the downsampled signal (bottom).

where $|T(e^{j\omega})|$ and $arg[T(e^{j\omega}]$ represent amplitude and phase distortions, respectively. If $|T(e^{j\omega})|$ is constant for all ω, then there is no amplitude distortion. Also, if $T(Z)$ is a linear phase FIR function, then $arg[T(e^{j\omega})] = k\omega$, and there is no phase distortion. As a result, $T(Z)$ becomes a delay, i.e., $T(Z) = CZ^{-n_0}$, so that $\hat{x}(n)$, reconstructed signal, is a delayed version of $x(n)$, i.e., $\hat{x}(n) = cx(n - n_0)$[7].

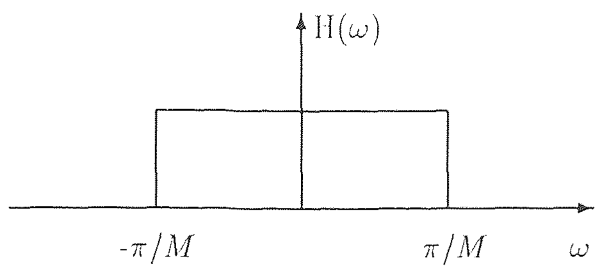
Smith and Barnwell[8] have shown first time that amplitude and phase distortions can be eliminated simultaneously by choosing

$$H_1(Z) = Z^{N-1} H_0(-Z^{-1}) \tag{4.10}$$



(a) An anti-aliasing filter followed an M-fold decimator.

(b) An M-fold interpolator followed by an interpolation filter



(c) A low-pass filter with the stopband edge π/M

Figure 4.6 Interpolation and decimation filters.

where $(N-1)$ is odd and N is the order of $H_1(Z)$. Thus, we have

$$T(Z) = \frac{1}{2}\, Z^{-(N-1)}[H_0(Z)H_O(Z^{-1}) - H_0(-Z)H_0(-Z^{-1})] = CZ^{-(N-1)} \qquad (4.11)$$

Therefore, the perfect reconstruction requirement reduces to finding an $H(Z) = H_0(Z)$ so that

$$\begin{aligned} Q(Z) &= H(Z)H(Z^{-1}) + H(-Z)H(-Z^{-1}) = C \\ &= R(Z) + R(-Z) \end{aligned} \qquad (4.12)$$

The perfect reconstruction requirement in time is expressed as[5]

$$\rho(2n) = \sum_{k=0}^{N-1} h(k)(k+2n) \qquad (4.13)$$

where $\rho(2n)$ is the autocorrelation function.

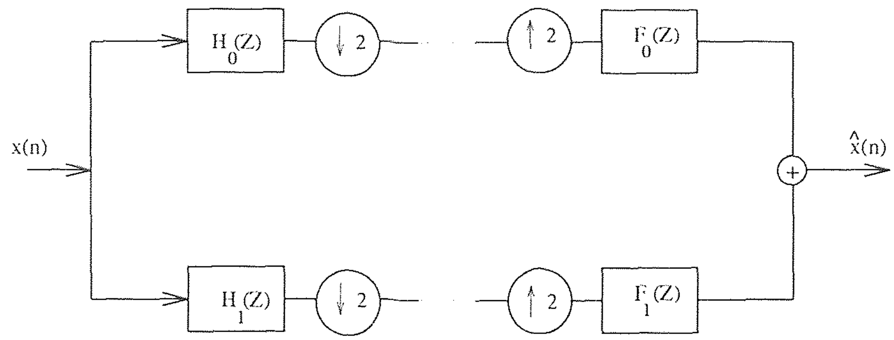


Figure 4.7 The two-channel QMF bank.

In summary, we have the perfect reconstruction conditions for the 2 band PR-QMF banks as follows;

$$\begin{aligned} F_0(Z) &= -H_1(-Z) \\ F_1(Z) &= H_0(-Z) \\ H_0(Z) &= H(Z) \end{aligned} \tag{4.14}$$

and

$$\begin{aligned} R(Z) = H(Z)H(Z^{-1}) &\longleftrightarrow \rho(n) = h(n) * h(-n) \\ R(Z) + R(-Z) = 1 &\longleftrightarrow \rho(2n) = \rho(n) \end{aligned} \tag{4.15}$$

4.4 M-Band Tree Decomposition

Once a given signal x(t) is sampled at f_s and split into two subband signals, $X_L(n)$ and $X_H(n)$, each of these subband signals can be further decomposed into more than 2 subbands in the same manner as the initial signal x(n). Four subband signals, thus are obtained after reduction of the sampling rate to $f_s/4$. The spectrum of each of these subbands, $X_{LL}(n)$, $X_{LH}(n)$, $X_{HL}(n)$ and $X_{HH}(n)$, represents the subspectrum of x(n) in the corresponding subband. This decomposition-reconstruction structure can be repeated p times yielding a p-stage hierarchical tree decomposition. The initial

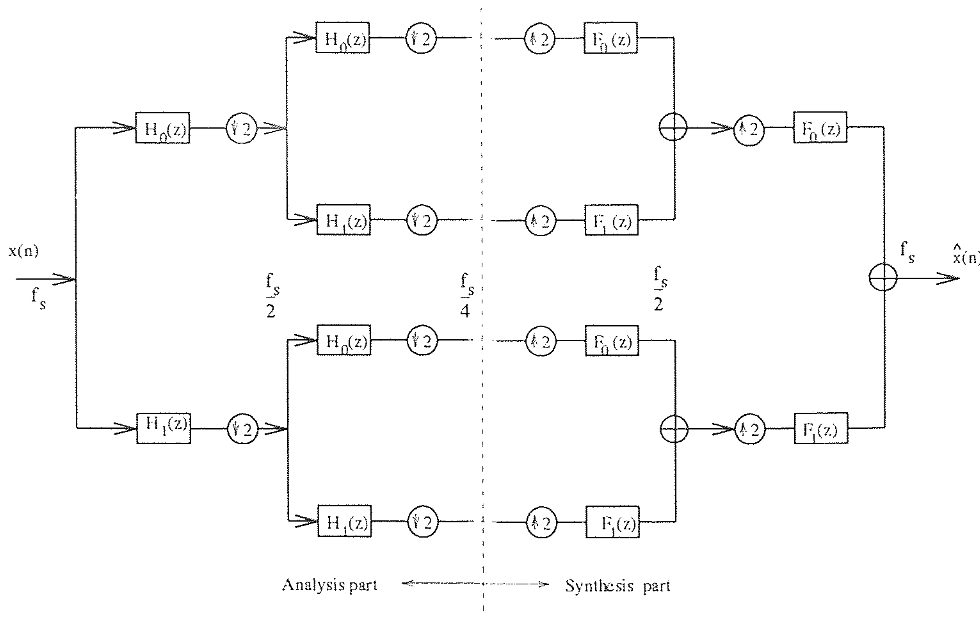


Figure 4.8 4 band regular tree decomposition.

signal is thus decomposed into $M=2^p$ subbands. The subband tree decomposition technique is shown in Figure 4.8 for 4-band regular tree and in Figure 4.9 for 7-band dyadic tree decomposition[9].

4.5 Two Dimensional Separable Case

The two dimensional (2-D) filter bank is a direct extension of 1-D filter bank in separable filter case. In the separable case, the filters used can be expressed as the product of two one-dimensional filters as given in Eq.(4.16).

$$h(n_1, n_2) = h_1(n_1)h_2(n_2) \tag{4.16}$$

The separability feature of the filter provides an alternative method of implementation of 2D-QMF banks. Figure 4.10 shows a four band analysis/ synthesis filter bank structure. As shown, the structure consists of a set of one dimensional filters which operate separately along the rows and the columns of the input signal.

It can be shown that the use of such filters will result an alias-free reconstruction of the input signal at the receiver side[10].

The decomposition of input signal can be extended for more than four subbands, i.e., 7, 10, 13 etc., by repeating the process as explained in section 4.4.

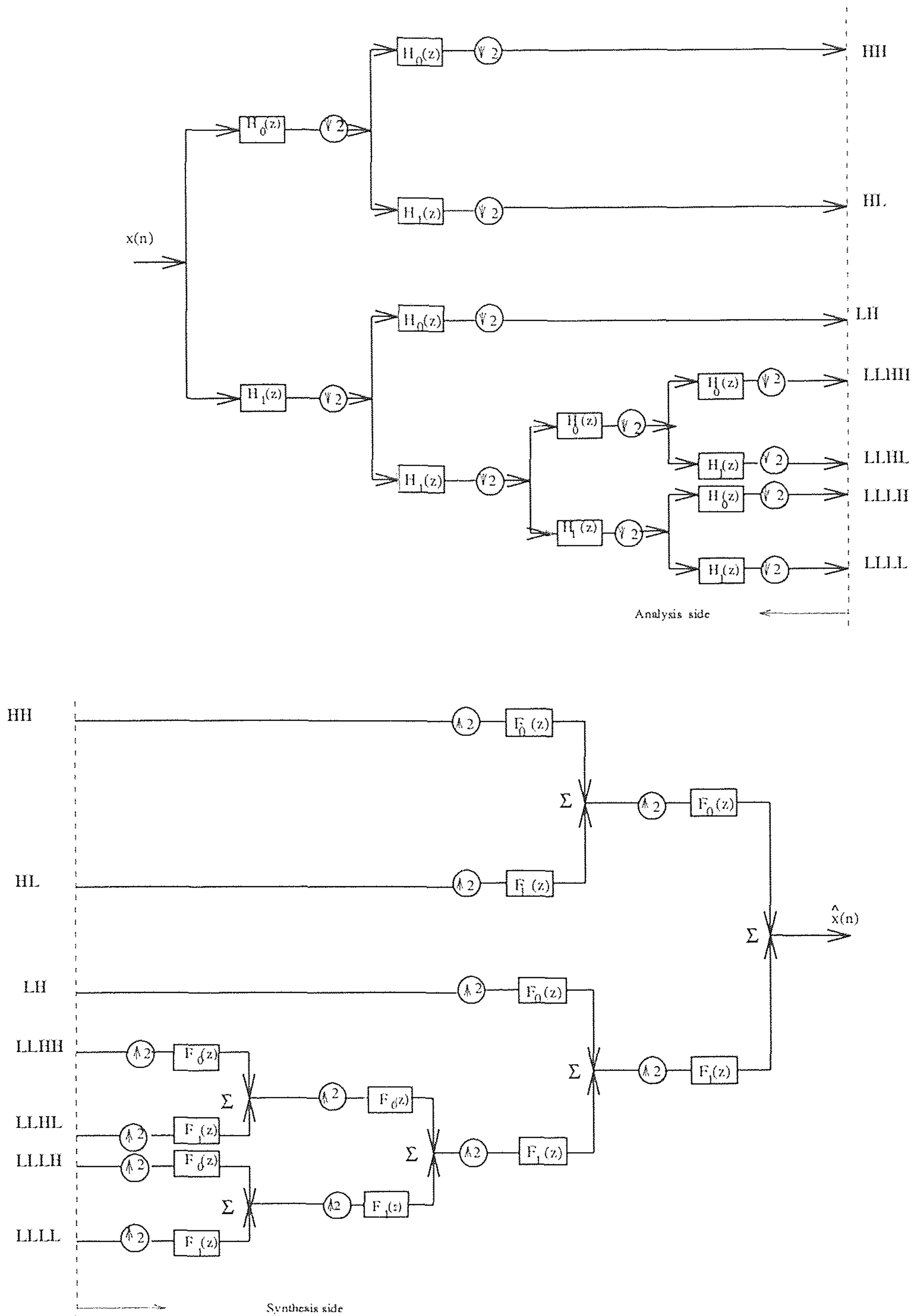


Figure 4.9 7 band dyadic tree decomposition.

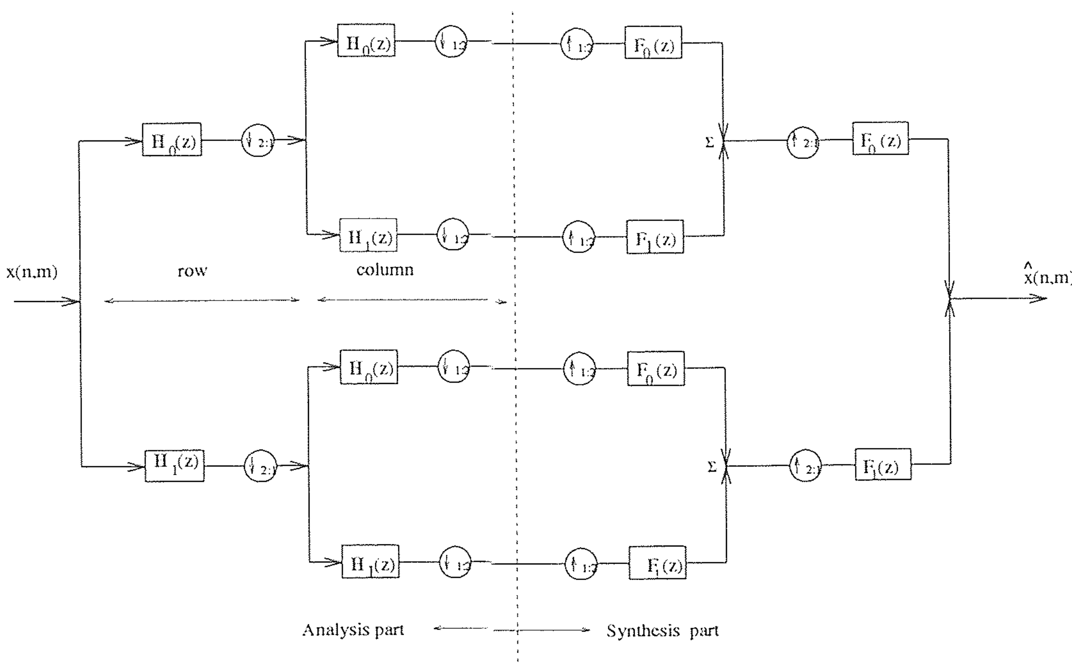


Figure 4.10 4 band regular tree decomposition and reconstruction of a 2-D signal x(m,n).

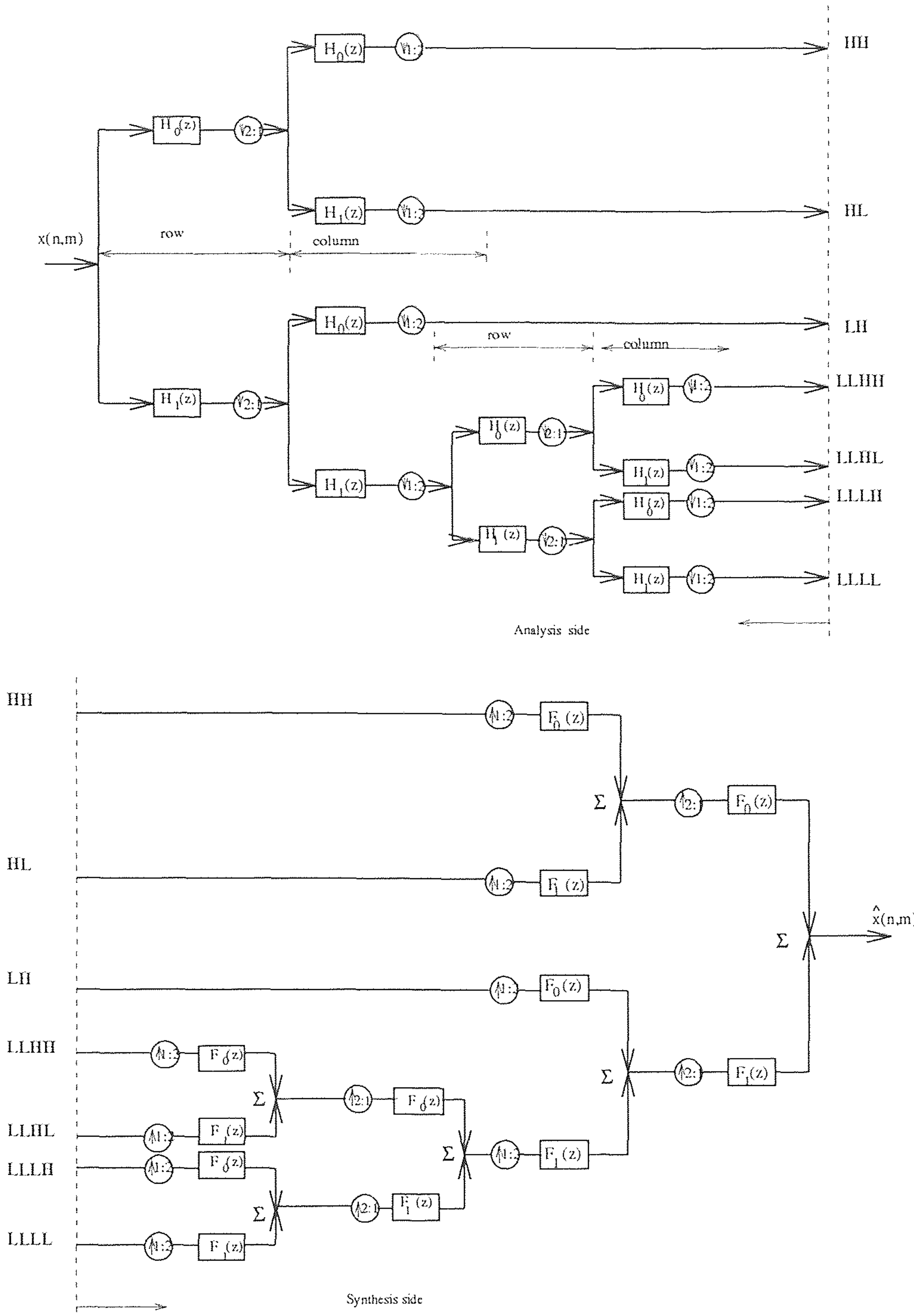


Figure 4.11 7 band dyadic tree decomposition and reconstruction of a 2-D signal x(m,n) using 2-D separable filters

CHAPTER 5

VECTOR QUANTIZATION IN SUBBANDS

5.1 Introduction

Many different coding techniques can be used for the encoding of the subband signals. In this study, vector quantization is used to independently encode the subband signals.

In this chapter, the concentration is given to vector quantization and its applications in subbands. Section 5.2 covers the general concept of vector quantization. In section 5.3, codebook design and the LGB algorithm[11] is covered. Section 5.4 deals with the vector quantization in subbands.

5.2 Vector Quantization

Vector quantization, also known as block quantization, is a direct extension of scalar quantization. The basic principle here is to map an N-dimensional input vector x onto another N-dimensional vector y, i.e.,

$$y = VQ(x) \tag{5.1}$$

where $VQ(.)$ is the vector quantization operator. Reconstruction vector, y_i, takes its value from one of the finite set of vectors

$$Y = \{y_i, \quad i = 1, ..., L\}. \tag{5.2}$$

The set Y is referred as the codebook containing the L code-vectors. For an L-length codebook the bits per code vector is given by

$$R = \frac{1}{N} log_2 L. \tag{5.3}$$

The vector quantization procedure can be described as follows. First, the N-dimensional vector x is constructed from the input signal. Next, the best fitting code

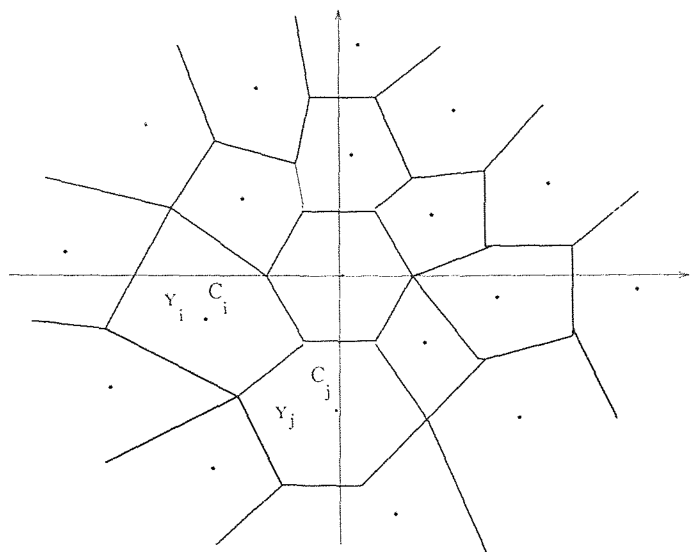


Figure 5.1 Illustration of clusters and the vector quantization for two-dimensional space.

vector y_i minimizing the distortion measure is searched in the codebook . This can be expressed as

$$VQ(x) = y_i, \iff d(x, y_i) \leq d(x, y_j) \quad j = 1, ..., L \tag{5.4}$$

where $d(x, y)$ represents the distortion measure or distance measure between the vectors x and y. The vector quantization process is shown in Figure 5.1 where two-dimensional space is divided into cells. The shape of each cell is uniquely determined by the location of the code vectors and the distortion measure.

5.3 Codebook Design

A codebook of size L divides the N-dimensional space into cells $\{C_i\}, i = 1, ..., L$ associating each cell C_i a code vector y_i. The vector quantizer assigns the code vector y_i to the vector x if x falls into C_i. The optimal quantizer is found by minimizing the distortion over all possible L-level quantizers. The overall average distortion of a vector quantizer is defined as[12]

$$D = \sum_{i=1}^{L} P(x \in C_i) \int_{x \in C_i} d(x, y_i)\ p(x)\ dx \tag{5.5}$$

where $P(x \in C_i)$ is the probability that x lies inside C_i and, $p(x)$ is the multi-dimensional probability density function (pdf) of x. The integral is taken over all components of x. However, in practice the pdf is usually unknown. In that case we use a training set, consisting of a large number of vectors $V_n, n = 1, 2, ..., M$. The codebook is designed using an iterative algorithm known as the K-means algorithm. Since this algorithm was first proposed by *L*inde, *G*ray and *B*uzo it is called as the LGB algorithm[11].

5.3.1 The LGB Algorithm

The basic steps of LGB algorithm is implemented as follows[11]:

- 1. Initialization: Iteration index is set to m=0. An initial codebook size L with the codebook vectors $y_i^0, i = 1, ..., M$ is chosen.
- 2. Clustering: The training vectors $V_n, i = 1, ..., M$ are classified into the clusters C_i by using the nearest neighbor rule.
- 3. Updating: New codebook vector y_i^{m+1} is calculated for each cell C_i^m by calculating the centroid of the training vectors classified to that cell:

  $$y_i^{m+1} = cent(C_i^m) = \frac{1}{M} \sum_{V_n \in C_i} V_n, \quad i = i, ..., L, \tag{5.6}$$

  where M_i is the number of training vectors classified to cell C_i.
- 4. Stop: New average distortion is calculated and if the distortion is below the predetermined threshold then the iteration is stopped. Otherwise iteration index is increased by 1 and the clustering operation is performed.

There are several ways to choose the initial codebook for the LGB algorithm. In our approach, we used splitting technique which works as follows. The initial codebook contains only one vector which is the centroid of all training sequence. The second codebook with codebook size L=2 is created by adding and subtracting a perturbance (a splitting vector) to the initial codebook. After optimizing this codebook, the splitting technique is repeated for larger size codebooks, i.e., L=4, 8, 16,...,512 etc. In our experiment L=512 is found as the optimal codebook size with respect to overall distortion and, iteration is stopped.

5.4 Vector Quantization In Subbands

As explained earlier in section 5.3, the LGB algorithm is used in this study to generate the codebooks. First, the MCFD signal is decomposed into 7 band using dyadic tree structure. After studying the subband signals, some of the subbands are found insignificient and entirely discarded. The remaining subbands excluding LH band are adaptively vector quantized based on the motion vectors[13]. The LH band is statistically modeled and the model parameters are vector quantized.

The 2-D 7 band dyadic tree structure, given in Figure 4.11, is used in this study. The filter used is the 8-tap separable filter[14]. After discarding the insignificiant bands, namely LLHH, HL and HH bands, the remaining bands are treated as follows.

5.4.1 The Adaptive Vector Quantization Based on the Motion Vectors

In this study, we employed adaptive vector quantization based on the block motion vectors (MBAVQ) to quantize LLLL, LLLH and LLHL bands as suggested in Ref [13].

In our approach, the motion compensation is based on block matching algorithm using brute-force method. The block size is set to $8x8$ and the maximum displacement in two directions, horizontally and vertically, is set to ∓ 6 pixels.

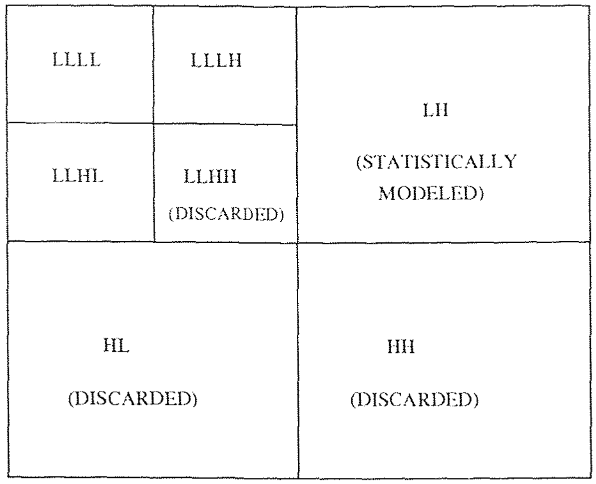


Figure 5.2 2-D subbands used in video codec

Each 8x8 block in full frame resolution corresponds to a 4x4 block in 4 subbands. Since the LL band is further decomposed into 4 subbands as seen in Figure 5.2, each 4x4 block in LL band corresponds to a 2x2 block in these subbands. The 4x4 and 2x2 blocks correspond to a block motion vector of size 8x8 in full frame resolution. In his thesis, Mutlag stated that there is a relation between motion vector magnitude and the MCFD variance of the corresponding block[13]. In general, large magnitude motion vectors represent high variance blocks in the MCFD signal while blocks with small motion vectors have small variances. By using this relation the codebooks are created depending on the motion vector magnitude. The magnitude of motion vector $\hat{m}$ is given by

$$\hat{m} = max(|i|, |j|) \tag{5.7}$$

where i and j are the horizontal and vertical displacements respectively. The block motion vectors are classified into 3 groups depending on their motion magnitudes:

- Group 1: $\hat{m}$=1 or 2
- Group 2: $\hat{m}$=3 or 4

- Group 3: $\hat{m}$=5 or 6.

Codebooks are generated using the subblocks corresponding to these groups. As a result, 9 codebooks are generated for the LLLL, LLLH and LLHL bands using the LGB algorithm.

For the LLLL band, the codebook contains 512 vectors in which each vector is in dimension 4. For the LLLH and LLHL bands the codebook size is 512 and each codeword is in dimension 16. For the latter case, 4 motion vectors are averaged and these motion vectors are used to create the codebooks.

5.4.2 Vector Quantization for the AR(1) Model Parameters

As mentioned earlier, the LH band is AR(1) modeled and model parameters are vector quantized and encoded for transmission. The model parameters, ρ_h, ρ_v, σ^2, and η, are vector quantized as follows: for means and variances two 256 length codebooks in which each codeword is in dimension 16 are generated. For the prediction coefficients, the codebook size is set to 512 where each codeword is also in dimension 16.

In conclusion, 12 codebooks are generated to vector quantize the subbands and model parameters.

CHAPTER 6

EXPERIMENTAL STUDIES

The performance of the proposed low bit-rate hybrid subband codec was simulated. The first forty frames of monochrome video test sequences CINDY, MONO, DUO, QUARTET and TOPGUN are used. The video frames are 512x400 pixel size except 240x352 for TOPGUN sequence, and 8 bits/pixel.

Simulations are carried out in the following manner. First, the MCFD signals are split into 7 subbands using 2-D dyadic tree structure and separable 8-tap filters[14]. Next, the statistical modeling of some subband signals are studied. The last step, the quantization, is carried out after modeling those subbands.

6.1 Subband Decomposition of the MCFD Signals

In this study, 7 band 2-D dyadic tree structure is employed for decomposition of the MCFD signal. This analysis/ synthesis subband tree structure is given in Figure 4.11. The filters employed are 8-tap separable filters. The frequency response of these filters are given in Figure 6.1. After studying the subband signals, LLHH, HL, HH bands are found insignificiant and completely discarded.

6.2 Statistical Modeling In Subbands

The remaining subbands are studied for statistical modeling using 2-D AR(1) technique. The LH band is statistically modeled. The model parameters are quantized and sent to the receiver side for the reconstruction of LH band. The modeling procedure was explained in detail in Chapters 3 and 5.

Frame by frame variation of subband variances for LH band before and after 2-D AR(1) modeling of the test sequence "CINDY" was displayed in Figure 3.3. As

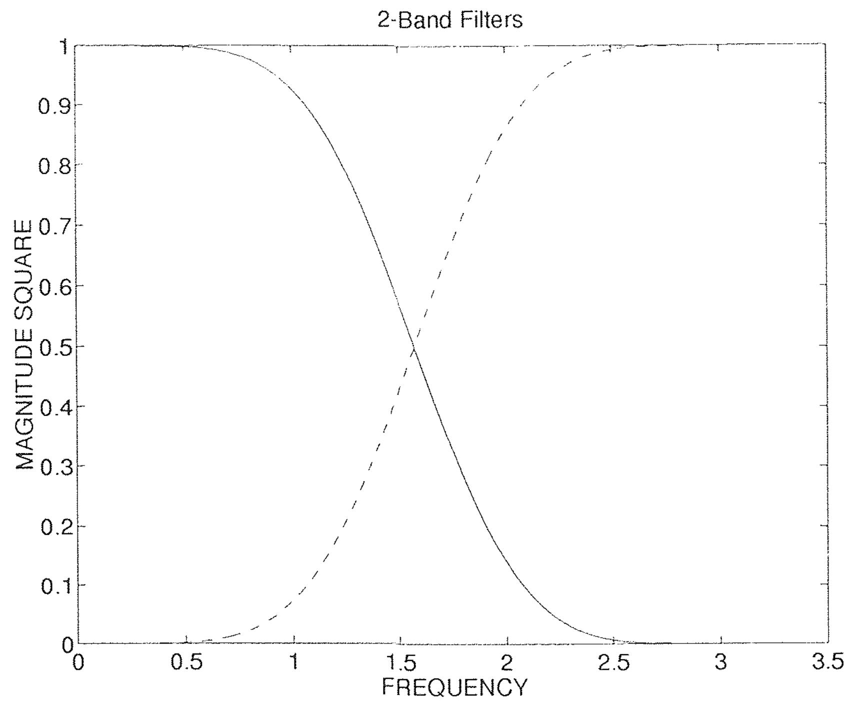


Figure 6.1 Frequency responses of the 8 tap separable low-pass and high-pass filters.

seen from the figure, modeling brings some error, but this error can be tolerated for the low bit-rate coding applications.

6.3 Quantization

Vector quantization is used to encode the subbands. The Motion Based Adaptive Vector Quantization (MBAVQ) is used for the subbands which are not modeled. Codebooks are generated using the LGB algorithm. For the statistically modeled LH band, the model parameters for each 4x4 blocks are quantized using the classical VQ algorithm. The more information about quantization was given in chapter 5.

The peak-to-peak signal to noise ratio is used as the objective performance criterion and defined as

$$SNR_{pp}(dB) = 10log_{10}\left(\frac{255^2}{E\{(X(i,j)-\hat{X}(i,j))^2\}} \right) \tag{6.1}$$

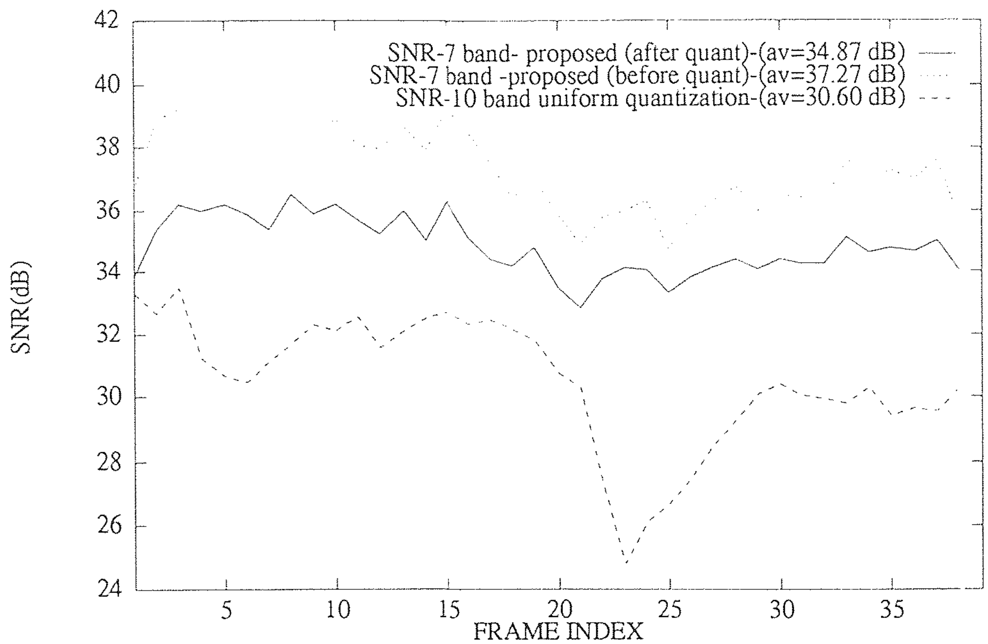


Figure 6.2 Frame by frame variation of the average SNR_{pp} values in dB for the test sequence "CINDY".

where the denominator term is the mean square coding error.

Frame by frame variation of peak-to peak SNR values before and after quantization for the test sequence "CINDY" is given in Figure 6.2 along with the average $SNR_{pp}(dB)$ values for the first forty frames. Figure 6.2 also shows the $SNR_{pp}(dB)$ values for the 10 band case which is used for comparison.

The total bit-rate for the proposed hybrid video codec can be expressed as

$$B = B_M + B_{SB} + B_{AR(1)} \tag{6.2}$$

where

- B_M= average bits/pixel for motion information.
- B_{SB}= average bits/pixel for the subband signals which are not modeled.
- $B_{AR(1)}$=average bits/pixel for the AR(1) modeled subbands.

The measure of information is expressed by the entropy. The first order entropy is defined as

$$H = -\sum_i p(i) log_2 p(i) \tag{6.3}$$

where p(i) is the probability of the source symbol i. The frame by frame variation of entropy values for the first forty frames of test sequence "CINDY" is given in Figure 6.3 along with the average entropy values.

The performance of the proposed algorithm was also tested for the video sequences which are not a part of the training sequence. "TOPGUN" which is not a part of training sequence, gave the superior SNR results for the proposed algorithm compared to the other algorithms used for comparison. Figures 6.4 and 6.5 display the $SNR_{pp}(dB)$ and entropy values of the tested 100 frames of the video sequence "TOPGUN". These figures also show the performance of the algorithms used for comparison.

25th and 26th frames of the test sequence "CINDY" are given in Figures 6.6 - 6.10 along with the MCFD frames for coded and original cases.

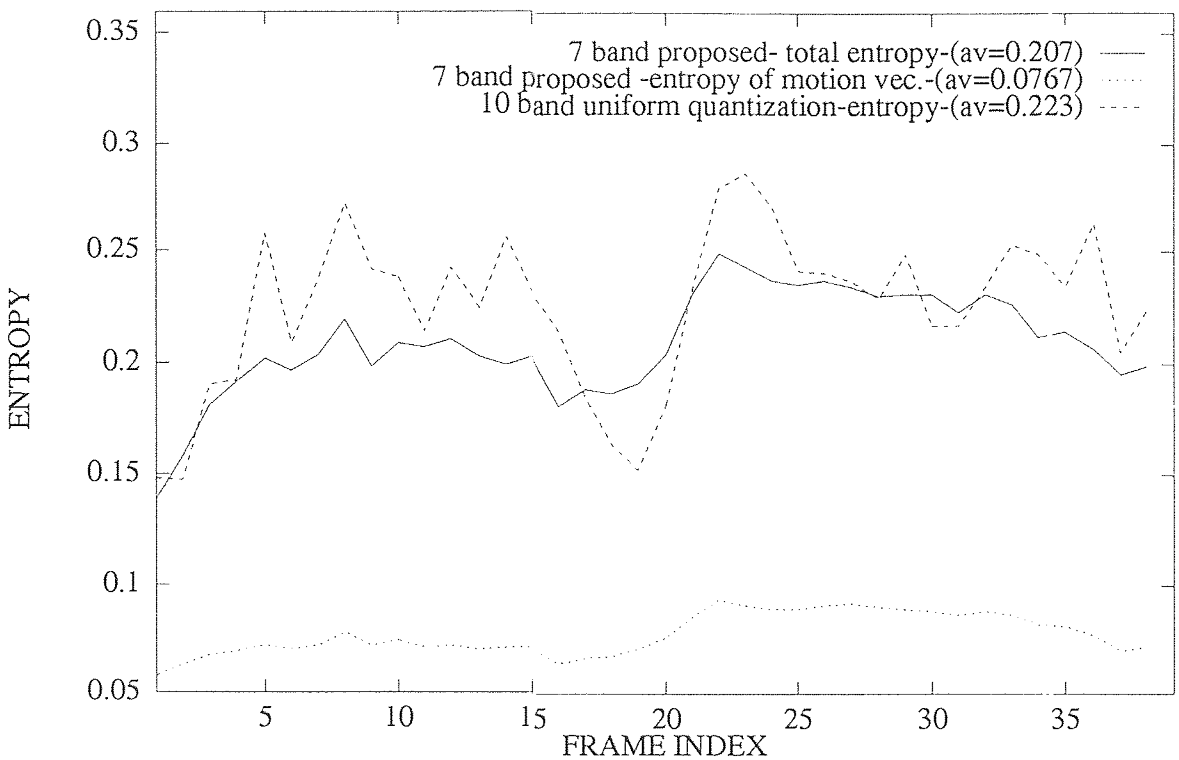


Figure 6.3 Frame by frame variation of the first order entropy values for the test sequence "CINDY".

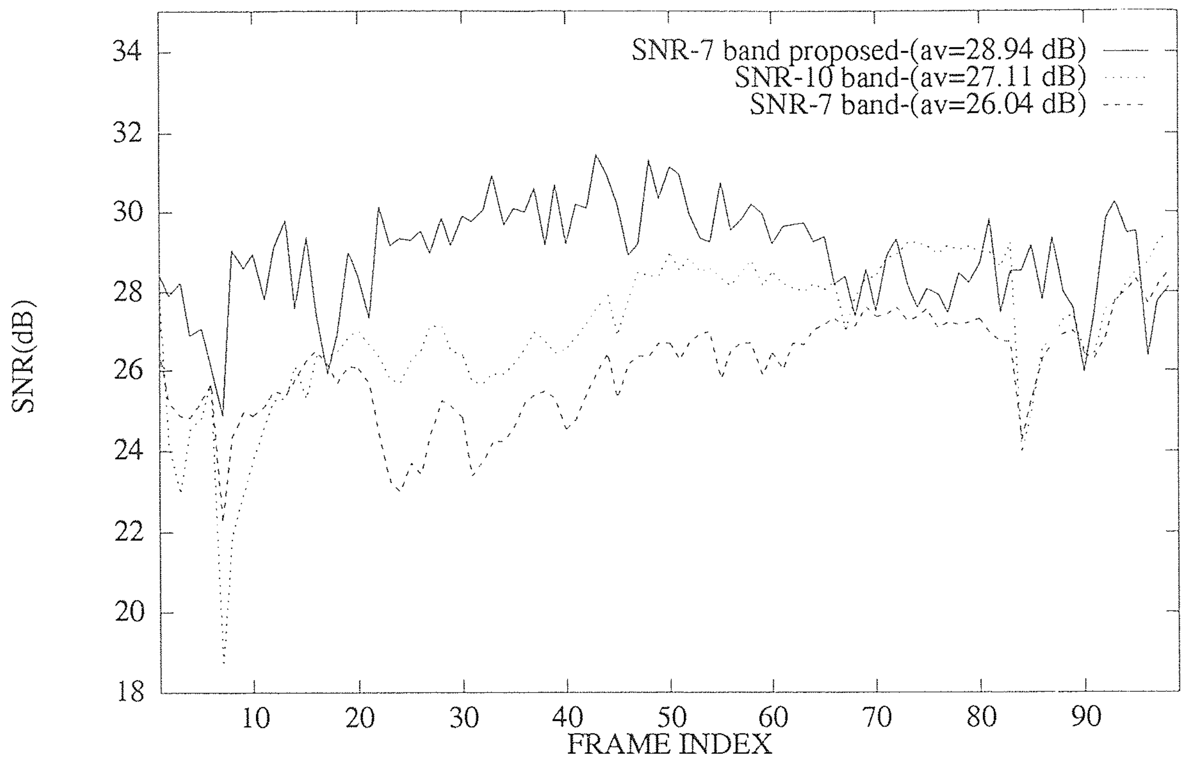


Figure 6.4 Frame by frame variation of the average SNR_{pp} values in dB for the test sequence "TOPGUN".

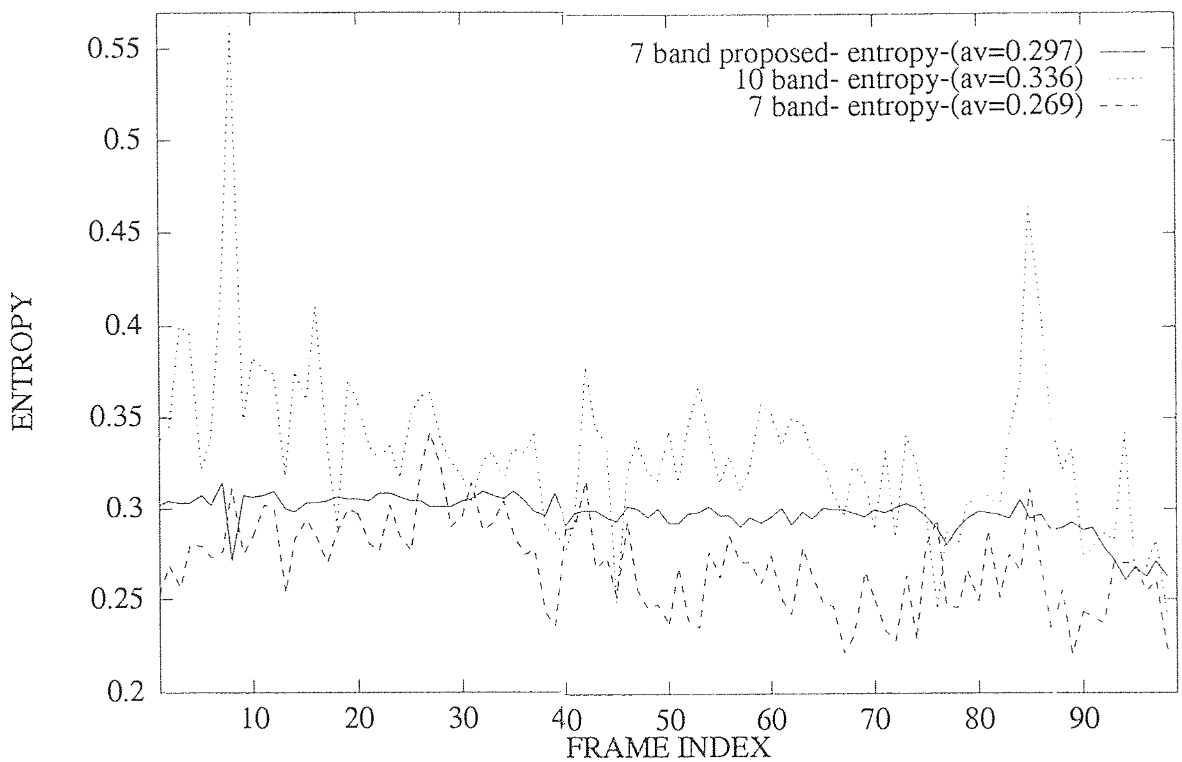


Figure 6.5 Frame by frame variation of the first order entropy values for the test sequence "TOPGUN".

Figure 6.6 25th frame of the test sequence "CINDY"

Figure 6.7 The direct difference between the frames 25 and 26 of test sequence "CINDY"

Figure 6.8 26th frames of the test sequence "CINDY", the original (top) and coded (bottom) ($SNR_{pp} = 34.1$, $bpp = 0.24$)

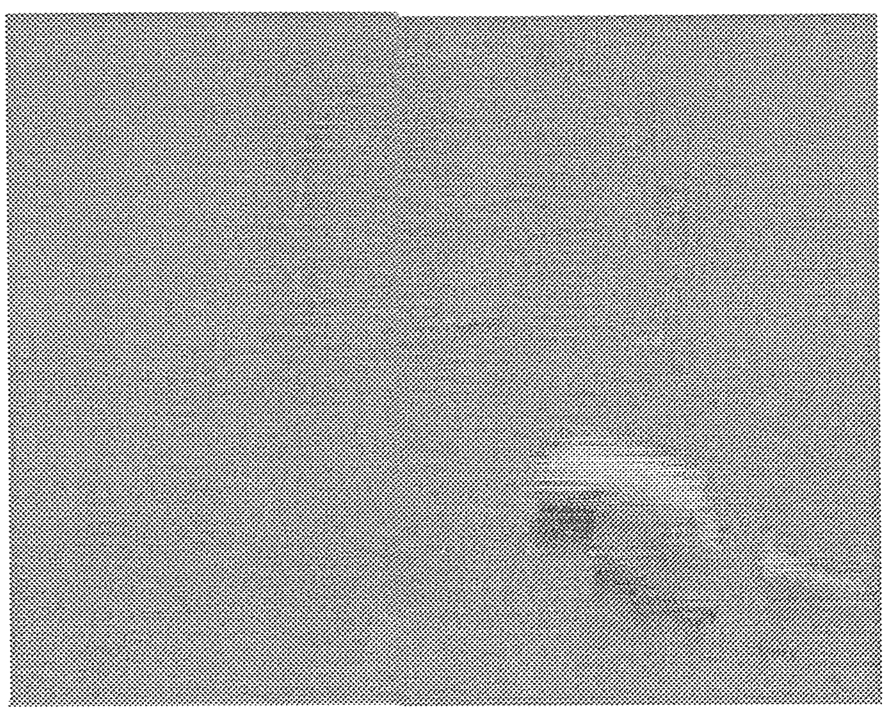


Figure 6.9 26th MCFD frames of "CINDY", the original (top) and coded (bottom) ($SNR_{pp} = 34.1$, $bpp = 0.24$).

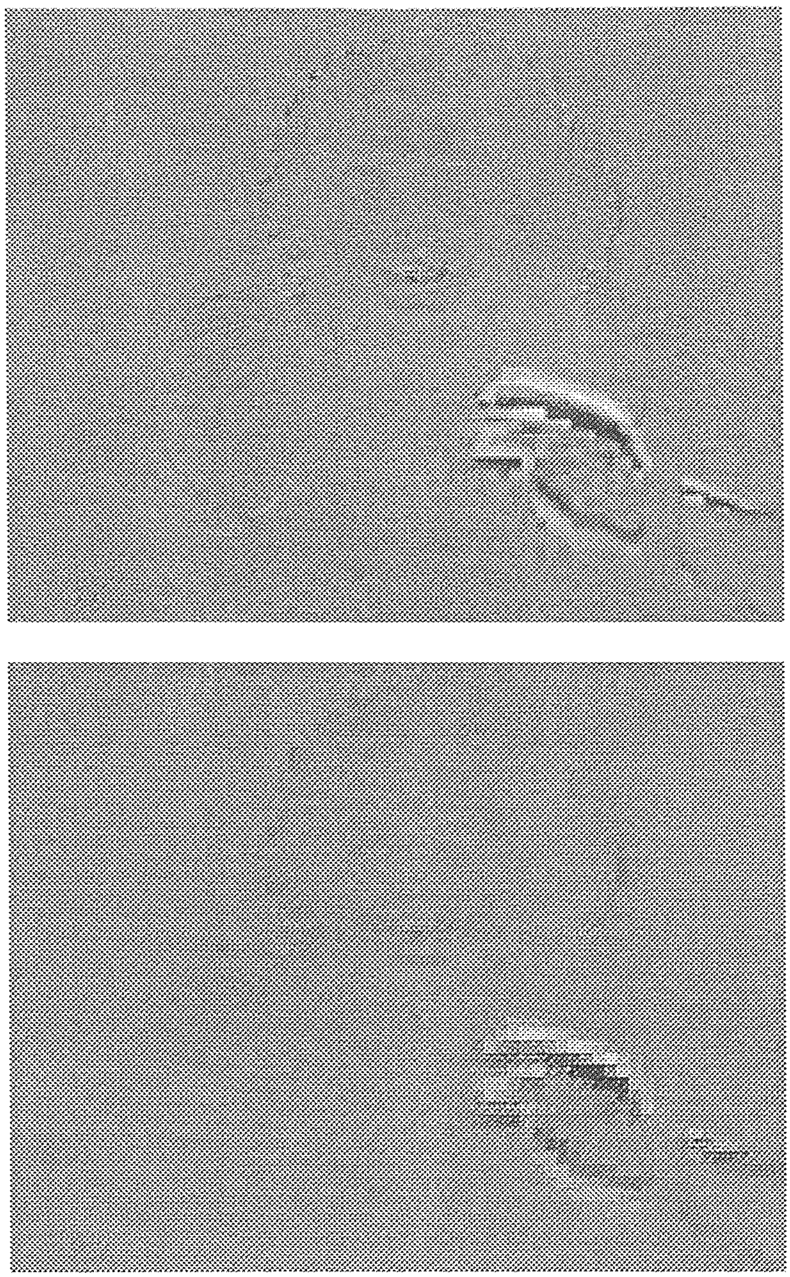


Figure 6.10 LH band of the 26th MCFD frames of "CINDY", the original (top) and statistically modeled (bottom).

CHAPTER 7

CONCLUSIONS AND DISCUSSIONS

A hybrid low bit-rate video codec is proposed in this study. The proposed technique achieves a good objective and visual performance at the low bit-rates B$\leq$0.30 bits/pixel with the SNR_{pp} 30-36 dB range.

It is well known that the training sequence dependency of vector quantization is a very important problem. Furthermore, the modeling approach used introduces some error due to stationarity assumption. In spite of these drawbacks, the statistical modeling is a viable approach for the low-correlated MCFD signal subbands and, the study to improve and develop better modeling techniques is an open field. In conclusion, the future work is to find better modeling approaches to have better performance and visual quality for low bit-rate video coding applications.

APPENDIX A

Simulation Program for the 7 Band Dyadic Tree Structure

```
c SOURCE CODE FOR THE PROPOSED ADAPTIVE SUBBAND VIDEO CODING WITH
c MOTION COMPENSATION using MBAVQ
c    nx      Row size of the picture
c    ny      Column size of the picture
c    frame1  Previous Frame
c    frame2  Current Frame
c    pics    Search frame from the previous frame
c    recon   Prediction of the current frame with motion compensation
c    ibs     Block size (8 is used)
c    ip      Assumed maximum displacement (Max of 6)
c    frm2msk ibs*ibs size mask of the current frame to be
c              motion compensated
c    frm1msk ibs*ibs size mask of the previous frame in the
c              same geometrical position (used for motion detection)
c    err1    Motion Compensated Frame Difference Signal
c    searg   Search Region (ibs+ip)*(ibs+ip)
c    mask    Same as frm2msk


parameter(nx=400,ny=512)
        integer motionv(50,64),ifld
        common /motionv/ motionv
        common /ifld/ ifld
        common /ifl/ ifl
        real outimg(400,512)

        real frame1(nx,ny),frame2(nx,ny),pics(416,528)
        real recon2(nx,ny),frm1msk(8,8),frm2msk(8,8),err1(nx,ny)
real recon3(400,512), errtemp(400,512)
        integer ifrm1(nx,ny),ifrm2(nx,ny)
        real dirdif(400,512)

        real b1v1(512,4),b1v2(512,4),b1v3(512,4)
        real b2v1(512,16),b2v2(512,16),b2v3(512,16)
        real b3v1(512,16),b3v2(512,16),b3v3(512,16)
        real b4v1(256,16),b4v2(256,16),b4v3(512,16),b4v4(512,16)


        common /entropy1/entropy1
        common /entropy2/entropy2
```

```
       common /entropy3/entropy3
       common/arbitrate/arbitrate

      common /vqcodebook1/ b1v1,b1v2,b1v3
      common /vqcodebook2/ b2v1,b2v2,b2v3
      common /vqcodebook3/ b3v1,b3v2,b3v3
      common /vqcodebook4/ b4v1,b4v2,b4v3,b4v4

character*1 pim(nx,ny)
character*1 pim1(nx,ny)
        integer mcvector(50,64)

        common /AAA/ searg(24,24),mask(8,8)



        print*, 'READING CODEBOOKS'

        open (15,file='B1NEW/b1v1.12')
        do 10 i=1,512
          read(15,*) (b1v1(i,j),j=1,4)
10      continue
        close(15)

        open (16,file='B1NEW/b1v2.34')
        do 11 i=1,512
          read(16,*) (b1v2(i,j),j=1,4)
11      continue
        close(16)

        open (17,file='B1NEW/b1v3.56')
        do 12 i=1,512
          read(17,*) (b1v3(i,j),j=1,4)
12      continue
        close(17)

        open (20,file='QUANTIZER/b2v1.12')
        do 1110 i=1,512
          read(20,*) (b2v1(i,j),j=1,16)
1110      continue
        close(20)

        open (21,file='QUANTIZER/b2v2.34')
        do 1111 i=1,512
          read(21,*) (b2v2(i,j),j=1,16)
```

```
1111      continue
        close(21)

        open (22,file='QUANTIZER/b2v3.56')
        do 1112 i=1,512
          read(22,*) (b2v3(i,j),j=1,16)
1112      continue
        close(22)

        open (23,file='QUANTIZER/b3v1.12')
        do 1113 i=1,512
          read(23,*) (b3v1(i,j),j=1,16)
1113      continue
        close(23)

        open (24,file='QUANTIZER/b3v2.34')
        do 1114 i=1,512
          read(24,*) (b3v2(i,j),j=1,16)
1114      continue
        close(24)

        open (25,file='QUANTIZER/b3v3.56')
        do 1115 i=1,512
          read(25,*) (b3v3(i,j),j=1,16)
1115      continue
        close(25)
        open (26,file='QUANTIZER/b4v4.mn')
        do 1116 i=1,256
          read(26,*) (b4v1(i,j),j=1,16)
1116      continue
        close(26)

        open (27,file='QUANTIZER/b4v4.var')
        do 1117 i=1,256
          read(27,*) (b4v2(i,j),j=1,16)
1117      continue
        close(27)

        open (28,file='QUANTIZER/b4v4.rh')
        do 1118 i=1,512
          read(28,*) (b4v3(i,j),j=1,16)
1118      continue
        close(28)
```

```
        open (29,file='QUANTIZER/b4v4.rv')
        do 1119 i=1,512
          read(29,*) (b4v4(i,j),j=1,16)
1119      continue
        close(29)
       print*, 'CODEBOOKS ARE READ'


c  mfld:  final  field to be read
c  ifld:  starting field number
        mfld =34
        ifld =33

         call read_frm(ifld,pim)
         ifl=0
c************************************
c   Frame One is read into frame1 array
c*************************************
      do 100 i=1,nx
        do 100 j=1,ny
        ifrm1(i,j)=ichar(pim(i,j))
        if(ifrm1(i,j).lt.0) ifrm1(i,j)=256+ifrm1(i,j)
        frame1(i,j)=float(ifrm1(i,j))
  100    continue

c call write_in_frm(ifld,frame1)
c call write_out_frm(ifld,frame1)

        write(35,*) 'Original Image'

c   The loop to process mfld number
c     of frames begins here

 6000   ifld = ifld+1
         ifl=ifl+1
        write(*,*) 'Frame Number = ', ifld,ifl
      write(35,*) 'Frame Number =',ifld

        call read_frm(ifld,pim1)


c************************************
c   Current frame is read into frame2
c*************************************
```

```
      do 110 i=1,nx
        do 110 j=1,ny
        ifrm2(i,j)=ichar(pim1(i,j))
        if(ifrm2(i,j).lt.0) ifrm2(i,j)=256+ifrm2(i,j)
        frame2(i,j)=float(ifrm2(i,j))
 110     continue

c        call write_in_frm(ifld,frame2)



c********************************
c Auto-Correlation, Mean, Variance
c are  calculated in
c the subroutine autocor
c********************************
c        print *,'Frame k'
        write(35,*) 'Frame k'
        call autocor(frame1,nx,ny)
c        print *,'Frame k+1'
        write(35,*) 'Frame K+1'
        call autocor(frame2,nx,ny)

         do 2000  i=1,400
         do 2000  j=1,512
          dirdif(i,j)=frame2(i,j)-frame1(i,j)
2000      continue

c         write(35,*) 'Direct Difference Frame'
        call autocor2(dirdif,nx,ny)


c ip: displacement
        ip=6

c  ibs:  the mask block size
        ibs=8

c  imthd: Enter 1 for Brute-force method and 2 for Orthogonal src
imthd=1

c  imdetect: Enter 1 if motion-detection is required'
imdetect=1
```

```
c***********************************
c  Search Array pics is Initialized
c***********************************
  do 101 i=1,nx+2*ip
  do 101 j=1,ny+2*ip
      pics(i,j)=0.0
 101 continue
c*************************************************
c  Search Array is generated from the previous frame.
c  Borders are filled with first(or last) ip
c  rows(or clums) of the previous frame
c*************************************************
do 155 i=1,nx
  do 155 j=1,ny
   pics(i+ip,j+ip)=frame1(i,j)
 155 continue

        do 111 i=1,ip
         do 111 j=1,ny
          pics(i,j)=frame1(i,j)
          pics(i+nx+ip,j)=frame1(i+nx-ip,j)
 111    continue

        do 112 i=1,nx
          do 112 j=1,ip
            pics(i,j)=frame1(i,j)
            pics(i,j+ny+ip)=frame1(i,j+ny-ip)
 112    continue
c*********************************************
c   Prediction of the Current frame is Initialized
c*********************************************

        do 240 i4=1,nx
          do 240 j4=1,ny
             recon2(i4,j4)=0.0
 240   continue

c***********************************************
c   The current frame is devided into 8*8 blocks and
c   motion compensated. mcount keeps count of number
c   of moving blocks.
c***********************************************
```

```
mcount=0
        do 200 i=1,nx/ibs
  do 200 j=1,ny/ibs
       iact=(i-1)*ibs+1
    jact=(j-1)*ibs+1
if (imdetect .eq. 1) then

do 401 k=1,ibs
        do 401 l=1,ibs
     frm1msk(k,l)=frame1(iact-1+k,jact-1+l)
frm2msk(k,l)=frame2(iact-1+k,jact-1+l)
401 continue
c*********************************************************
c      First the motion is detected
c*********************************************************

call motiondetect(frm1msk,frm2msk,ibs,indx)

         if (indx .eq. 1) then
         mcount=mcount+1
         do 410 i1=1,ibs+ip*2
 do 410 j1=1,ibs+ip*2
 searg(i1,j1)=pics(i1+iact-1+ip-ip,j1+jact-1+ip-ip)
410      continue

 do 420 i2=1,ibs
 do 420 j2=1,ibs
  mask(i2,j2)=frame2(iact-1+i2,jact-1+j2)
420  continue

*****************************************************
 if motion is detected, it is estimated and predicted
*****************************************************
   call matct(ip,ibs,imthd,n,nn,Num)
        motionv(i,j)=max(abs(n-7),abs(nn-7))
        mcvector(i,j)=Num
        do 430 i3=1,ibs
        do 430 j3=1,ibs
        recon2(iact-1+i3,jact-1+j3)=pics(iact+ip-1+(n-ip)-1+i3,
     +  jact+ip-1+(nn-ip)-1+j3)
 430    continue

        else
        motionv(i,j)=0
```

```
        mcvector(i,j)=0
c        write(55,*) mcvector(i,j)
        do 402 k1=1,ibs
        do 402 l1=1,ibs
        recon2(iact-1+k1,jact-1+l1)=frame1(iact-1+k1,jact-1+l1)
 402    continue
        endif

  else


        do 210 i1=1,ibs+ip*2
do 210 j1=1,ibs+ip*2
        searg(i1,j1)=pics(i1+iact-1+ip-ip,j1+jact-1+ip-ip)
 210    continue

        do 220 i2=1,ibs
        do 220 j2=1,ibs
        mask(i2,j2)=frame2(iact-1+i2,jact-1+j2)
 220    continue

   call matct(ip,ibs,imthd,n,nn,Num)


        do 230 i3=1,ibs
        do 230 j3=1,ibs
        recon2(iact-1+i3,jact-1+j3)=pics(iact+ip-1+(n-ip)-1+i3,
     +  jact+ip-1+(nn-ip)-1+j3)
 230    continue
endif
c        write(55,*) motionv(i,j)

 200 continue


*****************************
    MCFD signal is generated
c****************************
              do 650 i=1,400
      do 650 j=1,512
650       recon3(i,j)=0.0

              do 700 i=1,400
              do 700 j=1,512
```

```
700       recon3(i,j)=recon2(i,j)

             do 599 i=2,nx-1
             do 600 j=2,ny-1
          recon3(i,j)=(1.0/16.0)*(recon2(i-1,j-1)+2.0*recon2(i-1,j)
     #    + recon2(i-1,j+1)+2.0*recon2(i,j-1)+4.0*recon2(i,j)
     #    +2.0*recon2(i,j+1)+recon2(i+1,j-1)+2.0*recon2(i+1,j)
     #    +recon2(i+1,j+1))
600          continue
599          continue

         do 300 i=1,nx
         do 300 j=1,ny
         err=(err+abs(frame2(i,j)-recon3(i,j)))
         err1(i,j)=frame2(i,j)-recon3(i,j)
300     continue
        print *,'the value of err=',err
        write(35,*) 'the value of err=',err

        if (imdetect .eq. 1) then
          print *,'Number of blocks motion detected = ', mcount
          write(35,*)'Number of blocks motion detected  =',mcount
        endif

             print *,'Error Signal'
             write(35,*) 'Error Signal'
             call autocor(err1,nx,ny)
             print *,'Predicted singal'
             write(35,*) 'Predicted signal'
             call autocor(recon2,nx,ny)

             do 350 i = 1,nx
             do 350 j = 1,ny
             errtemp(i,j) = err1(i,j)
             outimg(i,j)=0.0
350          continue

          call bitrates2(mcvector,entropy,50,64)

c         call writeimgs(err1,400,512,'diff25')

*****CODING OF MCFD SIGNAL IS CARRIED OUT HERE******

**********7-BAND  FILTER BANK IS  CALLED************
```

```
         call subband(err1,outimg,400,512)

        print *,'Error Signal after the vector quantization'
        write(35,*)'Error Signal after the vector quantization'
        call autocor1(outimg,nx,ny)
        vecmean = 0.0
        vecvar = 0.0
        do 351 i = 1,nx
        do 351 j = 1,ny
        vecmean = vecmean + (errtemp(i,j) - outimg(i,j))
        vecvar = vecvar + (errtemp(i,j) - outimg(i,j))**2
351     continue
        vecmean = vecmean/(nx*ny)
        vecvar = vecvar/(nx*ny) - vecmean**2

       write(35,*) 'Variance of quantization error',vecvar

*********************************************************
Quantized MCFD signal is added to the motion compensated
prediction of the current frame and put into frame1 and
this  becomes the  previous frame for the next current frame
**********************************************************
        xmse = 0.0

        do 1000 i=1,nx
        do 1000 j=1,ny
         frame1(i,j)=recon3(i,j)+outimg(i,j)
         xmse = xmse + (frame2(i,j)-frame1(i,j))**2
 1000   continue

        call write_out_fr(frame1)
c       call write_out_frm(ifld,frame1)
        xmse = xmse/(nx*ny)
        snr = 10*log10(255**2/xmse)
        write(*,*) 'SNR = ',snr
        write(35,*) 'SNR = ',snr
        write(100,*) ifld ,snr

        write(*,*)  'Mean Square Error after Vector
     +  Quantization=',xmse
        write(36,*) ifld,xmse
        write(*,*) 'mean square error=',xmse
```

```
      tbitllll=entropy1/(400*512)
      tbitlllh=entropy2/(400*512)
      tbitllhl=entropy3/(400*512)
      tbitlh=arbitrate/(400*512)

      xmbitrate = entropy/64.0
      tbitsub= (entropy1+entropy2+entropy3+arbitrate)/(400*512)
      tbitrate=xmbitrate+tbitsub
      write(200,*) xmbitrate


      write(201,*) tbitsub
      write(35,*) 'total bitrate=',tbitrate
      write(202,*) tbitrate
      write(*,*) 'total bitrate=',tbitrate

******************************************
*if all the frames are not processed go back
*******************************************


       if(ifld.lt.mfld) go to 6000
   stop
     end

********************
* subroutine matct
********************

       subroutine matct(ip,ibs,imthd,n,nn,Num)

common /AAA/ searg(24,24),mask(8,8)
real test(13,13)

do 50 i=1,2*ip+1
       do 50 j=1,2*ip+1
       test(i,j)=0.0
       do 50 ii=1,ibs
       do 50 jj=1,ibs
       test(i,j)=abs(mask(ii,jj)-searg(i+ii-1,j+jj-1))+test(i,j)
 50    continue

c   Brute force technique
```

```
        if (imthd .eq. 1 ) then
          tmin=1.0e20
           do 100 i=1,2*ip+1
            do 100 k=1,2*ip+1
              if(test(i,k) .lt. tmin) then
      tmin=test(i,k)
        n=i
             nn=k
      endif
 100  continue

else

call ortho(test,ip,ibs,icent,jcent)

n=icent
nn=jcent

endif

c
c  Generates anumber between 1 & 169, The number idicates
c  the motion information
c
        Num=(n-1)*(ip*2+1)+nn
c       write(*,*) Num,n,nn

        return
end



subroutine ortho(test,ip,ibs,icent,jcent)
********************************************
*  Independent Orthognal Search Technique  *
********************************************


real test(ip*2+1,ip*2+1)

icent=ip+1
jcent=ip+1
l=ip/2.+.5
istep=0
```

```
10 if ((test(icent,jcent) .lt. test(icent,jcent-l)) .and.
     +     (test(icent,jcent) .lt. test(icent,jcent+l)))  then
             icent=icent
     jcent=jcent
else if ((test(icent,jcent-l) .lt. test(icent,jcent)) .and.
     +     (test(icent,jcent-l) .lt. test(icent,jcent+l)))  then
             icent=icent
     jcent=jcent-l
else if ((test(icent,jcent+l) .lt. test(icent,jcent)) .and.
     +     (test(icent,jcent+l) .lt. test(icent,jcent-l)))  then
             icent=icent
     jcent=jcent+l
        endif


  istep=istep+1


if ((test(icent,jcent) .lt. test(icent-l,jcent)) .and.
     +     (test(icent,jcent) .lt. test(icent+l,jcent)))  then
             icent=icent
     jcent=jcent
else if ((test(icent-l,jcent) .lt. test(icent,jcent)) .and.
     +     (test(icent-l,jcent) .lt. test(icent+l,jcent)))  then
             icent=icent-l
     jcent=jcent
else if ((test(icent+l,jcent) .lt. test(icent,jcent)) .and.
     +     (test(icent+l,jcent) .lt. test(icent-l,jcent)))  then
             icent=icent+l
     jcent=jcent
endif


istep=istep+1

if (l .ne. 1) then
   l=(l/2.0+.5)
   go to 10
        endif
return
end
       subroutine motiondetect(frm1msk,frm2msk,ibs,indx)
```

```
****************************************************
*  Subroutine calculates if motion is present in the  *
*  (ibs*ibs) subblock                                  *
****************************************************

       real frm1msk(ibs,ibs),frm2msk(ibs,ibs)

kount=0
do 10 i=1,ibs
    do 10 j=1,ibs
    thrsh=abs(frm1msk(i,j)-frm2msk(i,j))
if (thrsh .gt. 3) kount=kount+1
  10 continue
if (kount .gt. 10 ) then
indx=1
        else
indx=0
endif
c print *,'index',indx
return
end

****************************************************
*This Subroutine calculates the prediction coefficients,*
*means and variances of each ibx*iby block and  forms   *
*the  AR(1) model of  corresponding subband  frame      *
****************************************************
       subroutine ar1(fror,nx,ny,armod,ibx,iby)
       character*1  pimm(200*256)
       real fror(1:nx,1:ny)
       real   autoc(12800)
       real autoc1(12800)
       real uframe1(-12:212,-12:268)
       real  rmean1(3200),fror0(-10:210,-10:266)
       real framar1(1:200,1:256)
       real  rh(3200),rv(3200),var(3200)
       real armod(1:200,1:256)
       real fror1(-5:205,-5:261)
       real mean,sigma
       integer hist41(512)
       integer hist42(512)
       integer hist43(512)
       integer hist44(512)
       common/arbitrate/arbitrate
```

```
      real yy(1:16)
      common /yy/yy

      do 13 i=-5 ,205
      do 13 j=-5 ,261
      fror1(i,j)=0.0
13    continue

      do 44 i=1,3200

      rmean1(i)=0.0
      rh(i)=0.0
      rv(i)=0.0
      var(i)=0.0

44    continue

c      write(1,*) ((fror(i,j),j=1,256),i=1,200)

      do 39 i=1,200
      do 39 j=1,256
      fror1(i,j)=fror(i,j)
      framar1(i,j)=0.0
      armod(i,j)=0.0
39     continue
**********************************************
* IN FOLLOWING LOOP ZERO MEAN  FRAME OPTAINED*
**********************************************
       kk=0
      do 10  i=0,nx-ibx,ibx
      do 11  j=0,ny-iby,iby
        ii=i
        jj=j
        kk=kk+1
        rmn=0.0
        do 12 k=ii+1,ii+ibx
        do 12 l=jj+1,jj+iby
        rmn=rmn+fror1(k,l)
12     continue
       rmean1(kk)=rmn/(ibx*iby)

      do 14  m=ii+1,ii+ibx
      do 14  n=jj+1,jj+iby
```

```
      fror0(m,n)=fror1(m,n)-rmean1(kk)
14    continue

11    continue
10    continue

        do 50 m=-12,nx+12
        do 50 n=-12,ny+12
        uframe1(m,n)=0.0
50      continue
        mm=0

*****************************************************
*IN FOLLOWING LOOP ,ZERO MEAN MCFD FRAME MODELED BY *
*USING THE AR1 MODEL PARAMETERS. FIRST, VARIANCE,   *
*AUTOCORRELATION , RH,RV OF EACH ibx*iby BLOCKS ARE *
*CALCULATED                                         *
*****************************************************
        do 511 i=0, nx-ibx,ibx
        do 512 j=0, ny-iby,iby
        ii=i
        jj=j
        mm=mm+1
        rautoc=0.0
        do 71 k=ii+1,ii+ibx
        autoc(k)=0.0
        do 72 l=jj+1, jj+iby-1
        autoc(k)=autoc(k)+fror0(k,l)*fror0(k,l+1)
72      continue
 rautoc=rautoc+autoc(k)
71      continue

        rautoc = rautoc/(ibx*iby)


        rautoc1=0.0
        do 23 l=jj+1, jj+iby
        autoc1(l)=0.0
        do 24 k=ii+1, ii+ibx-1
        autoc1(l)= autoc1(l)+fror0(k,l)*fror0(k+1,l)
24      continue
        rautoc1=rautoc1+autoc1(l)
23      continue
```

```
        rautoc1 = rautoc1/(ibx*iby)

        var1=0.0
        do 25  k=ii+1, ii+ibx
        do 26  l=jj+1, jj+iby
        var1=var1+fror0(k,l)*fror0(k,l)
26      continue
25      continue

        var(mm) = var1/(ibx*iby)
        rh(mm) = rautoc/var(mm)
        rv(mm)= rautoc1/var(mm)
c        write(50,*) rh(mm),rv(mm)

512     continue
511     continue
*******************************************
* QUANTIZATION OF THE AR1 MODEL PARAMETERS*
* ARE CARIED OUT HERE                     *
*******************************************
        print*, 'quantizing ar1 parameters'

        call vec_quant4(rmean1,1,hist41,256)
        call vbitrates4(hist41,bs1,256)

        call vec_quant4(var,2,hist42,256)
        call vbitrates4(hist42,bs2,256)


        call vec_quant4(rh,3,hist43,512)
        call vbitrates4(hist43,bs3,512)


        call vec_quant4(rv,4,hist44,512)
        call vbitrates4(hist44,bs4,512)

        arbitrate=bs1+bs2+bs3+bs4
        write(*,*) 'BITRATE FOR AR1=', arbitrate


c        write(100,*) (hist41(i),i=1,512)


        N=16
```

```
      sigma=1.0
      mean=0.0
      nu=0
      call gauss(N,mean,sigma,iseed)

      do 518 i=0, nx-ibx,ibx
      do 518 j=0, ny-iby,iby
      ii=i
      jj=j
      nu=nu+1
      varno=(1.0-rh(nu)**2)*(1.0-rv(nu)**2)*var(nu)
      sigma1=sqrt(varno)

      nn=0
      do 27 k=ii+1, ii+ibx
      do 28 l=jj+1, jj+iby
      nn=nn+1
      uframe1(k,l)=rh(nu)*uframe1(k,l-1)+rv(nu)*uframe1(k-1,l)
     +  - rh(nu)*rv(nu)*uframe1(k-1,l-1)+sigma1*yy(nn)
28    continue
27    continue
*************************************************
*MEANS ARE  ADDED TO  THE AR1 MODELED MCFD FRAME*
*************************************************
      do 29 k=ii+1, ii+ibx
      do 30 l=jj+1, jj+iby
      framar1(k,l)=uframe1(k,l)+rmean1(nu)
      uframe1(k,l)=0.0
30    continue
29    continue

518   continue

          do 510 i=1,nx
          do 510 j=1,ny
          armod(i,j)=framar1(i,j)
510       continue

c     open(99,file='ard25',access='direct',form='unformatted'
c    + ,recl=nx*ny)
c     do  690 i=1,nx
c     do  691 j=1,ny
c     ip=int(armod(i,j))+128
c     if(ip.gt.255) ip=255
```

```
c        if (ip.lt.0) ip=0
c        if(ip.gt.128) ip=ip-255
c        kk=j+(i-1)*ny
c        pimm(kk)=char(ip)
c691     continue
c690     continue
c       write(99,rec=1) (pimm(j),j=1,nx*ny)
c        close(99)
         write(*,*) 'OKAY'
        return
        end
*****************************
*  GAUSSIAN NOISE GENERATOR *
*****************************
        subroutine gauss(N,mean,sigma,iseed)
        real x(1:16),mean,yy(1:16)
        common /yy/yy
        do 1 i=1,N
        x(i)=gran(mean,sigma,iseed)
        yy(i)=x(i)
1       continue
        return
        end
****************************************
        function gran(mean,sigma,iseed)
        real mean
        u=0
        do 1 i=1,12
        u=u+ran(iseed)
1       continue
        gran=sigma*(u-6)+mean
        return
        end
c***************************************

        subroutine write_out_fr(frm)
        real frm(400,512)
        real frm1(1:512,1:512)
        character*1 image(512*512)
        common /ifl/ ifl
        parameter(nx=400,ny=512)
        open(21,file='fr25',access='direct',form=
     &  'unformatted',recl=512*512)
```

```
            do 300 i=0,513
             do 300 j=0,513
300              frm1(i,j)=0.0

             do 200 i=1,400
       do 200 j=1,512
             frm1(i,j)=frm(i,j)
200    continue

           do 599 i=1,nx
           do 600 j=1,ny
      frm1(i,j)=(1.0/16.0)*(frm(i-1,j-1)+2.0*frm(i-1,j)
     #+ frm(i-1,j+1)+2.0*frm(i,j-1)+4.0*frm(i,j)
     #+2.0*frm(i,j+1)+frm(i+1,j-1)+2.0*frm(i+1,j)
     #+frm(i+1,j+1))
600        continue
599        continue

c          do 499 i=3,nx-2
c          do 500 j=3,ny-2
c       frm1(i,j) =(1.0/256.0)*((frm(i-2,j-2)+frm(i-2,j+2)
c      # +frm(i+2,j+2))
c      # +4.0*(frm(i-2,j-1)+frm(i-2,j+1)+frm(i-1,j-2)
c      # +frm(i-1,j+2))
c      # +4.0*(frm(i+1,j-2)+frm(i+1,j+2)+frm(i+2,j-1)
c      # +frm(i+2,j+1))
c      # +6.0*(frm(i-2,j)+frm(i,j-2)+frm(i,j+2)
c      # +frm(i+2,j))
c      # +16.0*(frm(i-1,j-1)+frm(i-1,j+1)
c      # +frm(i+1,j-1)+frm(i+1,j+1))
c      # +24.0*(frm(i-1,j)+frm(i,j-1)+frm(i,j+1)
c      # +frm(i+1,j))
c      # +36.0*frm(i,j))+(1./256.)*frm(i+2,j-2)
c500       continue
c499       continue
        do 10 i=1,512
        do 10 j=1,512
         ip=int(frm1(i,j)+.5)
         if(ip.gt.255) ip=255
         if(ip.lt.0) ip=0
         if(ip.gt.127) ip=ip-256
         image((i-1)*512+j) = char(ip)
 10     continue
```

```
      write(21,rec=ifl)(image(j),j=1,512*512)

      close(21)

      return
      end
*********************************************




      subroutine write_out_frm(ifld,frm)
      real frm(400,512)
      real pic1(1:512,1:512)
      character*1 image(512*512)

      open(21,file='cindy.out',access='direct',form=
     & 'unformatted',recl=512*512)
           do 300 i=0,513
             do 300 j=0,513
300             pic1(i,j)=0.0

             do 200 i=1,400
             do 200 j=1,512
             pic1(i,j)=frm(i,j)
200   continue

           do 599 i=1,nx
           do 600 j=1,ny
      pic1(i,j)=(1.0/16.0)*(frm(i-1,j-1)+2.0*frm(i-1,j)
     #+ frm(i-1,j+1)+2.0*frm(i,j-1)+4.0*frm(i,j)
     #+2.0*frm(i,j+1)+frm(i+1,j-1)+2.0*frm(i+1,j)
     #+frm(i+1,j+1))
600        continue
599        continue



       do 10 i=1,400
       do 10 j=1,512
        ip=int(pic1(i,j)+.5)
        if(ip.gt.255) ip=255
        if(ip.lt.0) ip=0
        if(ip.gt.127) ip=ip-256
```

```
         image((i-1)*512+j) = char(ip)
 10     continue

        write(21,rec=ifld)(image(j),j=1,512*512)

        close(21)

        return
        end
**********************
*  SUBROUTINE AUTOCOR*
**********************

        subroutine autocor(frame,nx,ny)
        real frame(nx,ny)
        real autoc(400),autoc1(512)

        rautoc=0.0
        do 11 k=1,nx
          autoc(k)=0.0
          do 12 l=1,ny-1
           autoc(k)=autoc(k)+frame(k,l)*frame(k,l+1)
 12       continue
         rautoc=rautoc+autoc(k)/ny
 11     continue

        rautoc1=0.0
        do 13 l=1,ny
          autoc1(l)=0.0
          do 14 k=1,nx-1
            autoc1(l)=autoc1(l)+frame(k,l)*frame(k+1,l)
 14       continue
          rautoc1=rautoc1+autoc1(l)/nx
 13     continue

        rac=0.0
        rmean=0.0
        do 23 l=1,ny
          do 24 k=1,nx
            rac=rac+frame(k,l)*frame(k,l)
            rmean=rmean+frame(k,l)
 24       continue
 23     continue
```

```
        rmean=rmean/(nx*ny)
        var=rac/(nx*ny)-rmean*rmean

        rautoc=rautoc/nx-rmean*rmean
        rautoc1=rautoc1/ny-rmean*rmean

        write(35,*) 'Mean Variance'
        write(35,*) rmean,var

        rh=rautoc/var
        rv=rautoc1/var

        write(35,*) 'Autocor-H,Autocor-V'
        write(35,*) rh,rv

        return
        end
**********************************************
* subroutine initial                         *
* This subprogram initialize the main program*
**********************************************
        subroutine initial
        character*80 input_file
        common /ina/ input_file

        write (*,1)
write (*,3)
        read (5,4) input_file


 1      format ('   ')
 3      format (' Enter the name of the file contains
     &  ',/,'the order of the filtes there coefficients,
     &  input Image, and output file:')
 4      format( a80)

        return
        end
c*******************************************
       subroutine writeimgs(pic,nx,ny,name)
       real pic(nx,ny)
       character*1 pim(400*512)
       character*20 name
```

```
        open(1,file=name,access='direct',
     + form='unformatted',recl=400*512)
       do 20 i=1,nx
         do 20 j=1,ny
           ip=int(pic(i,j))+128
           if(ip.gt.255) ip=255
           if(ip.lt.0) ip=0
           if(ip.gt.128) ip=ip-256
            mm=j+(i-1)*ny
            pim(mm)= char(ip)
 20    continue
       write(1,rec=1) (pim(j),j=1,nx*ny)
       close (1)

       return
       end
c******************************************
       subroutine writeimg(pic,nx,ny,name)
       real pic(nx,ny)
       character*1 pim(400*512)
       character*20 name

         open(1,file=name,access='direct',
     + form='unformatted',recl=nx*ny)
       do 20 i=1,nx
         do 20 j=1,ny
           ip=int(pic(i,j))
           if(ip.gt.128) ip=ip-256
            mm=j+(i-1)*ny
            pim(mm)= char(ip)
 20    continue
       write(1,rec=1) (pim(j),j=1,nx*ny)
       close (1)

       return
       end
c*********************************************
       subroutine writeint(c,nx,ny,name)
       real c(nx,ny)
       character*20 name

       open(1,file=name)
```

```
      do 10 i=1,nx
       write(1,*) (c(i,j),j=1,ny)
10    continue

      close (1)

      return
      end
**********************************************
      subroutine writeint1(pic,nx,ny,name)
      integer pic(nx,ny)
      character*20 name

      open(1,file=name)

      do 10 i=1,nx
       write(1,*) (pic(i,j),j=1,ny)
10    continue

      close (1)

      return
      end
c ************************************************
      subroutine read_frm(ifld,pic)
      character*1 pic(400,512)

      open(1,file='/images/cindy',access='direct',form=
     &  'unformatted',recl=512)

c       open(1,file='/images/mono',access='direct',form=
c     &  'unformatted',recl=512)
c
c       open(1,file='/images/quartet',access='direct',form=
c     & 'unformatted',recl=512)
c
c       open(1,file='/images/duo',access='direct',form=
c     &  'unformatted',recl=512)
c
       icod1 = (ifld-1)*400
       icod2 = (ifld-1)*400 + 200

      do 10 i=1,200
       read(1,rec=icod1+i)(pic(2*i-1,j),j=1,512)
```

```
       read(1,rec=icod2+i)(pic(i*2,j),j=1,512)
10    continue
       close(1)

       return
       end


************************************************************

subroutine write_in_frm(ifld,pic)
        real pic(400,512)
        character*1 image(512*512)

        open(22,file='cindy.in',access='direct',form=
     &  'unformatted',recl=512*512)

do 10 i=1,400
        do 10 j=1,512
         ip=int(pic(i,j)+.5)
         if(ip.gt.255) ip=255
         if(ip.lt.0) ip=0
         if(ip.gt.127) ip=ip-256
         image((i-1)*512+j) = char(ip)
 10 continue

        do 20 i=204801,262144
          image(i) = char(003)
 20     continue

        write(22,rec=ifld)(image(j),j=1,512*512)

close(22)

return
end
************************************************************


subroutine write_1frm(pic,name)
        real pic(400,512)
        character*1 image(512*400)
        character*20 name
```

```
        open(22,file=name,access='direct',form=
     &  'unformatted',recl=512*400)
do 10 i=1,400
        do 10 j=1,512
         ip=int(pic(i,j)+.5)
         if(ip.gt.255) ip=255
         if(ip.lt.0) ip=0
         if(ip.gt.127) ip=ip-256
         image((i-1)*512+j) = char(ip)
 10 continue

c        write(22,rec=1)(image(j),j=1,512*400)

close(22)

return
end

***************************************************
* SUBROUTINE SUBBAND-7 BAND ANALYSIS AND SYNTHESIS*
***************************************************
*  LL-LH-HL-HH Bands used         *
*  Synthesize band signals        *
*  Write out reconstructed imageaa*
**********************************
subroutine subband(inimg,outimg,nx,ny)

        integer raw,col
c
c       raw=number of rows of input image
c       col=number of columns of input image
*******************************************************
* CHANGE raw and col values for different sized images*
*******************************************************
        parameter(raw=400,col=512)


        real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)
        character*80 input_file
        common /ifl/ ifl
        common /ina/ input_file
        common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     &       ,ltap3,mtap3,ltap4,mtap4
```

```
        real inimg(nx,ny)
        real a1(200,256),a2(200,256),a3(200,256),a4(200,256)
        real outimg(400,512)
        real b1(raw/4,col/4),b2(raw/4,col/4)
        real b3(raw/4,col/4),b4(raw/4,col/4)
        real e1(raw/2,col/2)

c       input_file='in8'
c       input_file='in81'
        input_file='FILTERS/inmf8'
c       input_file='insb8'
c       input_file='inst8'
c       input_file='inunc8'
c       input_file='inotf8'
c       input_file='inofa8'
c       input_file='inofb8'
c       input_file='inotaf8'
c       input_file='inofsa8'
c       input_file='inofsb8'
c       input_file='inoffl8'
        call readf

        write(*,*) 'subband analysis'
       call analysis256(inimg,a1,a2,a3,a4)
c      a1:LL
c      a2:LH
c      a3:HL
c      a4:HH
        call analysis128(a1,b1,b2,b3,b4)
c       b1:LLLL
c       b2:LLLH
c       b3:LLHL
c       b4:LLHH

        write(*,*) 'synthesis'
        call synthesis128(b1,b2,b3,b4,e1)
        call synthesis256(e1,a2,a3,a4,outimg,inimg)


return
        end
****************************************************
*    SUBROUTINE READ                               *
*    THIS SUBROUTINE READS THE FILTER COEFFICIENTS *
```

```
****************************************************

        subroutine readf
        real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)

        common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     &       ,ltap3,mtap3,ltap4,mtap4

        call openf
       write(*,*) 'reading filter coefficients'
        read(11,*) ltap1
c        write(*,*) ltap1
        read(11,*) mtap1
        do 10 i=ltap1,mtap1
   10         read(11,*) coff1(i)
c
        read(11,*) ltap2
        read(11,*) mtap2
        do 20 i=ltap2,mtap2
   20         read(11,*) coff2(i)
c

        read(11,*) ltap3
        read(11,*) mtap3
        do 30 i=ltap3,mtap3
   30         read(11,*) coff3(i)
c

        read(11,*) ltap4
        read(11,*) mtap4
        do 40 i=ltap4,mtap4
   40         read(11,*) coff4(i)
c

        close (11)

        RETURN
        END
**********************************************
        subroutine openf
        character*80 input_file
        common/ina / input_file
        write(*,*) 'Opening input_file'
        open (11,file=input_file,status='old')
```

```
        write(*,*) 'file opened'
        return
        end
*********************************************
        subroutine analysis256(inimg,llband,lhband,hlband,hhband)

        integer raw,col
        parameter(raw=400,col=512)


        real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)

        common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     &       ,ltap3,mtap3,ltap4,mtap4



c these are the four subband
        real llband(raw/2,col/2),lhband(raw/2,col/2)
     &  ,hlband(raw/2,col/2),hhband(raw/2,col/2)



c  these are the high and low bands
        real lband(raw,col/2),hband(raw,col/2)


c input and output images
        real inimg(raw,col)




        nx=raw
        ny=col

        call rfilter(coff1,inimg,lband,nx,ny,ltap1,mtap1)

        call rfilter(coff2,inimg,hband,nx,ny,ltap2,mtap2)

        ny=ny/2

        call cllfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)
```

```
      call clhfilter(coff2,lband,lhband,nx,ny,ltap2,mtap2)

      call chlfilter(coff1,hband,hlband,nx,ny,ltap1,mtap1)

      call chhfilter(coff2,hband,hhband,nx,ny,ltap2,mtap2)

      return
      end


*************************************************************

      subroutine analysis128(inimg,llband,lhband,hlband,hhband)

      integer raw,col
      parameter(raw=200,col=256)
      common /ifld/ ifld
      integer motionv(50,64),ifld
      common /motionv/ motionv

      real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)
      common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     & ,ltap3,mtap3,ltap4,mtap4



c these are the four subband
      real llband(raw/2,col/2),lhband(raw/2,col/2)
     &  ,hlband(raw/2,col/2),hhband(raw/2,col/2)



c  these are the high and low bands
      real lband(raw,col/2),hband(raw,col/2)


c input and output images
      real inimg(raw,col)
      nx=raw
      ny=col

      call rfilter(coff1,inimg,lband,nx,ny,ltap1,mtap1)

      call rfilter(coff2,inimg,hband,nx,ny,ltap2,mtap2)
```

```
      ny=ny/2

      call cllllfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)

      call clllhfilter(coff2,lband,lhband,nx,ny,ltap2,mtap2)

      call cllhlfilter(coff1,hband,hlband,nx,ny,ltap1,mtap1)

      call cllhhfilter(coff2,hband,hhband,nx,ny,ltap2,mtap2)

      return
      end
c_________________________________________________________

     subroutine rfilter(f,a1,a2,raw,col,ltap,mtap)
      integer col,raw,ltap,mtap

      real a1(raw,col),a2(raw,col/2),f(-20:20)

      do 20 i=1,raw
        do 20 j=2,col,2
           a2(i,j/2)=0
          do 20 k=ltap,mtap
           jk=j+k
           if(jk.le.0) jk=col+jk
           if(jk.gt.col) jk=jk-col
           a2(i,j/2)=a2(i,j/2)+a1(i,jk)*f(k)
20    continue

      return
      end
c________________________________________________________

     subroutine cfilter(f,a1,a2,raw,col,ltap,mtap)
      integer col,raw,ltap,mtap,jk
      real a1(raw,col),a2(raw/2,col),f(-20:20)

      do 20 i=1,col
        do 20 j=2,raw,2
           a2(j/2,i)=0
          do 20 k=ltap,mtap
           jk=j+k
           if(jk.le.0) jk=raw+jk
```

```
            if(jk.gt.raw) jk=jk-raw

            a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
 20     continue

        return
        end
*********** LL FILTER********************************
        subroutine cllfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        common /ifl/ ifl
        real a1(raw,col),a2(raw/2,col),f(-20:20)
        character*1  pimm(200*256)
        parameter(nx=200,ny=256)

          do 20 i=1,col
          do 20 j=2,raw,2
             a2(j/2,i)=0
            do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw

             a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
 20     continue
c        open(99,file='LL',access='direct',form='unformatted'
c     +  ,recl=nx*ny)
c        do  690 i=1,nx
c        do  691 j=1,ny
c        ip=int(a2(i,j))+128
c        if(ip.gt.255) ip=255
c        if (ip.lt.0) ip=0
c        if(ip.gt.128) ip=ip-255
c        kk=j+(i-1)*ny
c        pimm(kk)=char(ip)
c691     continue
c690     continue
c        write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c        close(99)

        return
        end

*********** LLLL FILTER********************************
```

```
      subroutine cllllfilter(f,a1,a2,raw,col,ltap,mtap)
      integer col,raw,ltap,mtap,jk
      real a1(raw,col),a2(100,128),f(-20:20)
      common /hist1/ hist1
      real hist1(3,512)
      common /ifl/ ifl
      common /entropy1/entropy1
      character*1  pimm(100*128)
      parameter(nx=100,ny=128)

        do 20 i=1,col
        do 20 j=2,raw,2
           a2(j/2,i)=0
          do 20 k=ltap,mtap
           jk=j+k
           if(jk.le.0) jk=raw+jk
           if(jk.gt.raw) jk=jk-raw
           a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
 20   continue


      call vec_quan1(a2)

c      open(99,file='LLLL',access='direct',form='unformatted'
c    + ,recl=nx*ny)
c      do  690 i=1,nx
c      do  691 j=1,ny
c      ip=int(a2(i,j))+128
c      if(ip.gt.255) ip=255
c      if (ip.lt.0) ip=0
c      if(ip.gt.128) ip=ip-255
c      kk=j+(i-1)*ny
c      pimm(kk)=char(ip)
c691   continue
c690   continue
c      write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c      close(99)
       call vbitrates1(hist1,entropy1,3,512)
       write(*,*) 'entropy1=',entropy1


      return
      end
```

```
*********** LLLH FILTER********************************
        subroutine clllhfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        real pimm1(12800),pimm2(12800)
        real a1(raw,col),a2(raw/2,col),f(-20:20)
        common /hist2/ hist2
        common /entropy2/entropy2
        real hist2(3,512)

        common /ifl/ ifl
        character*1  pimm(100*128)
        parameter(nx=100,ny=128)

          do 20 i=1,col
          do 20 j=2,raw,2
             a2(j/2,i)=0
            do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw

             a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
 20      continue

        call vec_quan2(a2)

c        open(99,file='LLLH',access='direct',form='unformatted'
c     +  ,recl=nx*ny)
c        do  690 i=1,nx
c        do  691 j=1,ny
c        ip=int(a2(i,j))+128
c        if(ip.gt.255) ip=255
c        if (ip.lt.0) ip=0
c        if(ip.gt.128) ip=ip-255
c        kk=j+(i-1)*ny
c        pimm(kk)=char(ip)
c691     continue
c690     continue
c        write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c        close(99)

c        do 1 i=1,100
c        do 1 j=1,128
```

```
c        kk=j+(i-1)*128
c        pimm1(kk)=a2(i,j)
c        write(60,*) pimm1(kk)
c1       continue
        call vbitrates2(hist2,entropy2,3,512)
        write(*,*) 'entropy2=',entropy2

        return
        end



********** LLHL FILTER******************************
        subroutine cllhlfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        real a1(raw,col),a2(raw/2,col),f(-20:20)
        common /ifl/ ifl
        common /hist3/ hist3
        common /entropy3/entropy3
        real hist3(3,512)

        character*1  pimm(100*128)
        parameter(nx=100,ny=128)

          do 20 i=1,col
          do 20 j=2,raw,2
             a2(j/2,i)=0
            do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw
            a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
  20    continue

        call vec_quan3(a2)

c        open(99,file='LLHL',access='direct',form='unformatted'
c     +  ,recl=nx*ny)
c        do  690 i=1,nx
c        do  691 j=1,ny
c        ip=int(a2(i,j))+128
c        if(ip.gt.255) ip=255
c        if (ip.lt.0) ip=0
c        if(ip.gt.128) ip=ip-255
```

```
c         kk=j+(i-1)*ny
c         pimm(kk)=char(ip)
c691      continue
c690      continue
c         write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c         close(99)
        call vbitrates3(hist3,entropy3,3,512)
        write(*,*) 'entropy3=',entropy3


        return
        end


********** LLHH FILTER****************************
        subroutine cllhhfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        real a1(raw,col),a2(raw/2,col),f(-20:20)
        common /ifl/ ifl
        parameter(nx=100,ny=128)

          do 20 i=1,col
          do 20 j=2,raw,2
             a2(j/2,i)=0
            do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw

c             a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
              a2(j/2,i)=0.0
  20    continue

        return
        end


*********LH FILTER***********************************

         subroutine clhfilter(f,a1,a2,raw,col,ltap,mtap)
         integer col,raw,ltap,mtap,jk
         real a1(raw,col),a2(raw/2,col),a6(200,256),f(-20:20)
        common /ifl/ ifl
         character*1  pimm(200*256)
```

```
         parameter(nx=200,ny=256)

          do 20 i=1,col
          do 20 j=2,raw,2
             a2(j/2,i)=0
          do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw
c              a2(j/2,i)=0
             a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
20       continue

c         open(99,file='LH',access='direct',form='unformatted'
c     +   ,recl=nx*ny)
c         do  690 i=1,nx
c         do  691 j=1,ny
c         ip=int(a2(i,j))+128
c         if(ip.gt.255) ip=255
c         if (ip.lt.0) ip=0
c         if(ip.gt.128) ip=ip-255
c         kk=j+(i-1)*ny
c         pimm(kk)=char(ip)
c691     continue
c690     continue
c         write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c         close(99)

         call ar1(a2,200,256,a6,4,4)
         do 35 i=1,200
         do 35 j=1,256
          a2(i,j)=a6(i,j)
35       continue

         return
          end
**********HL FILTER*******************************
       subroutine chlfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        real a1(raw,col),a2(raw/2,col),f(-20:20)
        common /ifl/ ifl
        character*1  pimm(200*256)
        parameter(nx=200,ny=256)
```

```
        do 20 i=1,col

          do 20 j=2,raw,2
             a2(j/2,i)=0
            do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw

c             a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
             a2(j/2,i)=0.0
  20    continue
c         call ar1(a2,200,256,a7,8,8)
c       do 35 i=1,200
c       do 35 j=1,256
c35        a2(i,j)=a7(i,j)
c        write(13,*) ((a2(i,j),j=1,256),i=1,200)
c       open(99,file='HL',access='direct',form='unformatted'
c     +  ,recl=nx*ny)
c        do  690 i=1,nx
c        do  691 j=1,ny
c        ip=int(a2(i,j))+128
c        if(ip.gt.255) ip=255
c        if (ip.lt.0) ip=0
c        if(ip.gt.128) ip=ip-255
c        kk=j+(i-1)*ny
c        pimm(kk)=char(ip)
c691     continue
c690     continue
c        write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c        close(99)

        return
        end
***********HH FILTER***************************
        subroutine chhfilter(f,a1,a2,raw,col,ltap,mtap)
        integer col,raw,ltap,mtap,jk
        real a1(raw,col),a2(200,256),f(-20:20)
        common /ifl/ ifl
real a8(200,256)
        character*1  pimm(200*256)
        parameter(nx=200,ny=256)

        do 20 i=1,col
```

```
          do 20 j=2,raw,2
             a2(j/2,i)=0
           do 20 k=ltap,mtap
             jk=j+k
             if(jk.le.0) jk=raw+jk
             if(jk.gt.raw) jk=jk-raw

c              a2(j/2,i)=a2(j/2,i)+a1(jk,i)*f(k)
               a2(j/2,i)=0.0
  20     continue
c           write(*,*) ifl
c          call ar1(a2,200,256,a8,16,16)
c           do 35 i=1,200
c           do 35 j=1,256
c35         a2(i,j)=a8(i,j)

c         open(99,file='HH',access='direct',form='unformatted'
c      +  ,recl=nx*ny)
c         do  690 i=1,nx
c         do  691 j=1,ny
c         ip=int(a2(i,j))+128
c         if(ip.gt.255) ip=255
c         if (ip.lt.0) ip=0
c         if(ip.gt.128) ip=ip-255
c         kk=j+(i-1)*ny
c         pimm(kk)=char(ip)
c691      continue
c690      continue
c         write(99,rec=ifl) (pimm(j),j=1,nx*ny)
c         close(99)

         return
         end


******************************************
        subroutine vec_quan1(pic)
******************************************
c       pic: picture o be coded (100X128)

        real pic(100,128)
        integer motionv(50,64)
        real tvector(4)
```

```
      common /hist1/ hist1
      integer hist1(3,512)
      common /motionv/ motionv
      real b1v1(512,4),b1v2(512,4),b1v3(512,4)
      common /vqcodebook1/ b1v1,b1v2,b1v3

      do 191 i=1,3
       do 191 j=1,512
       hist1(i,j)=0
191   continue


        nn=0
      do 10 i=1,50
      do 10 j=1,64
        if(motionv(i,j).ge.5) then
          do 20 k=1,2
          do 20 l=1,2
            tvector(2*(k-1)+l)=pic((i-1)*2+k,(j-1)*2+l)
 20       continue
          mm=nn+3
       else if(motionv(i,j).ge.3) then
          do 21 k=1,2
          do 21 l=1,2
            tvector(2*(k-1)+l)=pic((i-1)*2+k,(j-1)*2+l)
 21       continue
          mm=nn+2
        else if(motionv(i,j).ge.1) then
          do 22 k=1,2
          do 22 l=1,2
            tvector(2*(k-1)+l)=pic((i-1)*2+k,(j-1)*2+l)
 22       continue
          mm=nn+1
        else
          do 33 k=1,2
          do 33 l=1,2
             pic((i-1)*2+k,(j-1)*2+l)=0.0
 33       continue
           mm=0
          endif

        if(mm.eq.1) then
          call vquantizer(tvector,b1v1,ivecnum)
          hist1(1,ivecnum) = hist1(1,ivecnum)+1
```

```
         else if(mm.eq.2)then
           call vquantizer(tvector,b1v2,ivecnum)
           hist1(2,ivecnum) = hist1(2,ivecnum)+1

         else if(mm.eq.3)then
           call vquantizer(tvector,b1v3,ivecnum)
           hist1(3,ivecnum) = hist1(3,ivecnum)+1
         endif
          if(mm.ne.0) then
          do 44 k=1,2
          do 44 l=1,2
             pic((i-1)*2+k,(j-1)*2+l) = tvector(2*(k-1)+l)
44        continue
          endif

10        continue

         return
         end
*************************************
        subroutine vec_quan2(pic)
*************************************
c       pic: picture to be coded (100X128)

        real pic(100,128)
        integer motionv(50,64)
        integer motionv1(25,32)

        real tvector(16)
        common /hist2/ hist2
        integer hist2(3,512)
        common /motionv/ motionv
        real b2v1(512,16),b2v2(512,16),b2v3(512,16)
        common /vqcodebook2/ b2v1,b2v2,b2v3

        do 191 i=1,3
         do 191 j=1,512
         hist2(i,j)=0
191     continue

        do 50 i=1,25
         do 50 j=1,32
         notl=0
```

```
      do 51 k=i*2-1,2*i
      do 51 l=2*j-1,2*j
        notl =notl+motionv(k,l)
51    continue
      motionv1(i,j)=(notl/4)
50    continue



       nn=0
      do 10 i=1,25
      do 10 j=1,32
        if(motionv1(i,j).ge.5) then
          do 20 k=1,4
          do 20 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
 20       continue
          mm=nn+3
       else if(motionv1(i,j).ge.3) then
          do 21 k=1,4
          do 21 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
 21       continue
          mm=nn+2
        else if(motionv1(i,j).ge.1) then
          do 22 k=1,4
          do 22 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
 22       continue
          mm=nn+1
        else
          do 33 k=1,4
          do 33 l=1,4
             pic((i-1)*4+k,(j-1)*4+l)=0.0
 33       continue
           mm=0
          endif

        if(mm.eq.1) then
          call vquantizer3(tvector,b2v1,ivecnum)
          hist2(1,ivecnum) = hist2(1,ivecnum)+1

        else if(mm.eq.2)then
          call vquantizer3(tvector,b2v2,ivecnum)
```

```
            hist2(2,ivecnum) = hist2(2,ivecnum)+1

          else if(mm.eq.3)then
            call vquantizer3(tvector,b2v3,ivecnum)
            hist2(3,ivecnum) = hist2(3,ivecnum)+1
          endif
           if(mm.ne.0) then
           do 44 k=1,4
           do 44 l=1,4
              pic((i-1)*4+k,(j-1)*4+l) = tvector(4*(k-1)+l)
44         continue
           endif

10         continue

          return
          end

*************************************
        subroutine vec_quan3(pic)
c       pic: picture to be coded (100X128)

        real pic(100,128)
        integer motionv(50,64)
        integer motionv1(25,32)

        real tvector(16)
        common /hist3/ hist3
        integer hist3(3,512)
        common /motionv/ motionv
        real b3v1(512,16),b3v2(512,16),b3v3(512,16)
        common /vqcodebook3/ b3v1,b3v2,b3v3

        do 191 i=1,3
         do 191 j=1,512
         hist3(i,j)=0
 191    continue
        do 50 i=1,25
         do 50 j=1,32
         notl=0
         do 51 k=i*2-1,2*i
         do 51 l=2*j-1,2*j
           notl =notl+motionv(k,l)
 51      continue
```

```
      motionv1(i,j)=(notl/4)
50    continue

       nn=0
      do 10 i=1,25
      do 10 j=1,32
        if(motionv1(i,j).ge.5) then
          do 20 k=1,4
          do 20 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
20        continue
          mm=nn+3
        else if(motionv1(i,j).ge.3) then
          do 21 k=1,4
          do 21 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
21        continue
          mm=nn+2
        else if(motionv1(i,j).ge.1) then
          do 22 k=1,4
          do 22 l=1,4
            tvector(4*(k-1)+l)=pic((i-1)*4+k,(j-1)*4+l)
22        continue
          mm=nn+1
        else
          do 33 k=1,4
          do 33 l=1,4
             pic((i-1)*4+k,(j-1)*4+l)=0.0
33        continue
           mm=0
          endif

        if(mm.eq.1) then
          call vquantizer3(tvector,b3v1,ivecnum)
         hist3(1,ivecnum) = hist3(1,ivecnum)+1

        else if(mm.eq.2)then
          call vquantizer3(tvector,b3v2,ivecnum)
          hist3(2,ivecnum) = hist3(2,ivecnum)+1

        else if(mm.eq.3)then
          call vquantizer3(tvector,b3v3,ivecnum)
          hist3(3,ivecnum) = hist3(3,ivecnum)+1
        endif
```

```
         if(mm.ne.0) then
         do 44 k=1,4
         do 44 l=1,4
            pic((i-1)*4+k,(j-1)*4+l) = tvector(4*(k-1)+l)
44       continue
         endif

10       continue

        return
        end


**********************************************
      subroutine vec_quant4(enimg,n,hist4,L)
      real testv(16)
      real enimg(3200)
      integer hist4(L)
      integer ivecnum4
      real b4v1(256,16),b4v2(256,16),b4v3(512,16),b4v4(512,16)
      common /vqcodebook4/ b4v1,b4v2,b4v3,b4v4

      do 17 j=1,L
      hist4(j)=0
17    continue



      do 100 i=0,3184,16
         k=0
              do 150 ii=i+1,i+16
              k=k+1
              testv(k) = enimg(ii)
 150          continue

        if (n.eq.1) then

        call vquant4(testv,b4v1,ivecnum4,L)

        else if (n.eq.2) then

        call vquant4(testv,b4v2,ivecnum4,L)
```

```
          else if (n.eq.3) then

          call vquant4(testv,b4v3,ivecnum4,L)

          else if (n.eq.4) then

          call vquant4(testv,b4v4,ivecnum4,L)

           endif

          hist4(ivecnum4)=hist4(ivecnum4)+1

            k1=0
            do 170 jj=i+1,i+16
            k1=k1+1
            enimg(jj) = testv(k1)
170         continue

100       continue

        return
        end
```

```
***************************************************
        subroutine vquantizer(testv,codebook,ivecnu)
c  Best Matching of vector
        real testv(4)
        real codebook(512,4)

        rdiff = 1000000.0
        ivecnu = 0

        do 110 m = 1,512
            adiff = 0
            do 120 n = 1,4
             adiff = adiff + (testv(n) - codebook(m,n))**2
 120        continue
            if (adiff .lt. rdiff) then
               rdiff = adiff
```

```
              ivecnu = m
            endif
 110    continue

        do 130 n = 1,4
            testv(n) = codebook(ivecnu,n)
 130    continue

        return
        end


*****************************************************
        subroutine vquantizer3(testv,codebook,ivecnu)
c  Best Matching of vector
        real testv(16)
        real codebook(512,16)

        rdiff = 1000000.0
        ivecnu = 0

        do 110 m = 1,512
            adiff = 0
            do 120 n = 1,16
             adiff = adiff + (testv(n) - codebook(m,n))**2
 120        continue
            if (adiff .lt. rdiff) then
               rdiff = adiff
               ivecnu = m
            endif
 110    continue

        do 130 n = 1,16
            testv(n) = codebook(ivecnu,n)
 130    continue

        return
        end
**************************************************

        subroutine vquant4(testv,codebook,ivecnu,L)
c  Best Matching of vector
        real testv(16)
        real codebook(L,16)
```

```
      rdiff = 1000000.0
      ivecnu = 0

      do 110 m = 1,L
          adiff = 0
          do 120 n = 1,16
           adiff = adiff + (testv(n) - codebook(m,n))**2
120       continue
          if (adiff .lt. rdiff) then
             rdiff = adiff
             ivecnu = m
          endif
110   continue

      do 130 n = 1,16
          testv(n) = codebook(ivecnu,n)
130   continue

      return
      end

*****************************************************
c this subroutine calculate the entropy of each band
c and find the probability of each code
******************************************************

      subroutine vbitrates1(ic,bitrate,raw,col)

c      common /gtotal/ gtotal
      integer ic(raw,col),raw,col
      real entropy(3),sum(512),pr(512)

      gtotal=0
      bitrate=0
      do 10 m=1,3

      total=0
      do 20 n=1,512
       sum(n)=ic(m,n)
       total=total+sum(n)
20    continue
c
```

```
      entropy(m)=0.0
      do 30 n=1,512
         pr(n)=sum(n)/total
         if(pr(n).gt.0) then
           br=pr(n)*xlog2(1.0/pr(n))
           entropy(m)=entropy(m)+br
         endif
 30   continue

      bitrate=bitrate+entropy(m)*total
      write(*,*) 'total = ',total
      gtotal=gtotal+total
 10   continue

      write(*,*) 'gtotal = ',gtotal
      write(*,*) 'ventropy = ',(entropy(i),i=1,3)
      write(*,*) 'bitrate = ', bitrate

      return
      end
**********************************************

      subroutine vbitrates2(ic,bitrate,raw,col)

      integer ic(raw,col),raw,col
      real entropy(3),sum(512),pr(512)

      gtotal=0
      bitrate=0
      do 10 m=1,3

      total=0
      do 20 n=1,512
       sum(n)=ic(m,n)
       total=total+sum(n)
 20   continue


      entropy(m)=0.0
      do 30 n=1,512
         pr(n)=sum(n)/total
         if(pr(n).gt.0) then
           br=pr(n)*xlog2(1.0/pr(n))
           entropy(m)=entropy(m)+br
```

```
          endif
 30     continue

        bitrate=bitrate+entropy(m)*total
        write(*,*) 'total = ',total
        gtotal=gtotal+total
 10     continue
c        bitrate=(gtotal/800)*(9./256)

        write(*,*) 'gtotal = ',gtotal
        write(*,*) 'ventropy = ',(entropy(i),i=1,3)
        write(*,*) 'bitrate = ', bitrate
        return
        end

************************************************

        subroutine vbitrates3(ic,bitrate,raw,col)

        integer ic(raw,col),raw,col
        real entropy(3),sum(512),pr(512)

        gtotal=0
        bitrate=0
        do 10 m=1,3

        total=0
        do 20 n=1,512
         sum(n)=ic(m,n)
         total=total+sum(n)
20      continue
c

        entropy(m)=0.0
        do 30 n=1,512
           pr(n)=sum(n)/total
           if(pr(n).gt.0) then
             br=pr(n)*xlog2(1.0/pr(n))
             entropy(m)=entropy(m)+br
           endif
 30     continue

        bitrate=bitrate+entropy(m)*total
        write(*,*) 'total = ',total
```

```
      gtotal=gtotal+total
 10   continue

c     bitrate=(gtotal/800)*(9./256.)


      write(*,*) 'gtotal = ',gtotal
      write(*,*) 'ventropy = ',(entropy(i),i=1,3)
      write(*,*) 'bitrate = ', bitrate
      return
      end


*************************************************

      subroutine vbitrates4(ic,bitrate,col)

      integer ic(col),col
      real entropy,sum(512),pr(512)

      bitrate=0

      total=0
      do 20 n=1,col
       sum(n)=ic(n)
       total=total+sum(n)
20    continue

      entropy=0.0
      do 30 n=1,col
         pr(n)=sum(n)/total
         if(pr(n).gt.0) then
           br=pr(n)*xlog2(1.0/pr(n))
           entropy=entropy+br
         endif
 30   continue

      bitrate=entropy*total

      write(*,*) 'entropyar1 = ',entropy
      write(*,*) 'total = ',total
      write(*,*) 'bitrate = ', bitrate

      return
```

```
      end



****************************************

      function xlog2(x)
      real x
      xlog2=alog(x)/alog(2.0)
      return
      end

************************************************
      subroutine bitrates2(ic,entropy,raw,col)
************************************************
      integer ic(raw,col),raw,col
      real entropy,sum(0:169),pr(0:169)


      do 20 n=0,169
       sum(n)=0.0
 20   continue
c
      do 10 i=1,raw
      do 10 j=1,col
         k=ic(i,j)
         sum(k)=sum(k)+1
 10   continue
      entropy=0.0
      total=real(raw*col)
      do 30 n=0,169
         pr(n)=sum(n)/total
         if(pr(n).gt.0) then
           br=pr(n)*xlog2(1.0/pr(n))
           entropy=entropy+br
         endif
 30   continue

      write(*,*) 'xmentropy = ',entropy
c     write(*,*) 'pr = ',(pr(i),i=1,169)

      return
      end
```

```
**************************************************


      subroutine ccfilter(f,a1,a2,raw,col,ltap,mtap)
       integer col,raw,ltap,mtap,jk
       real a1(raw,col),a2(raw,col),f(-20:20)
       do 20 i=1,col
         do 20 j=1,raw
           a2(j,i)=0
           do 20 k=ltap,mtap
            jk=j+k
            if(jk.le.0) jk=raw+jk
            if(jk.gt.raw) jk=jk-raw
            a2(j,i)=a2(j,i)+a1(jk,i)*f(k)
 20    continue
       return
       end
c________________________________________________________

      subroutine rcfilter(f,a1,a2,raw,col,ltap,mtap)
       integer col,raw,ltap,mtap,jk

       real a1(raw,col),a2(raw,col),f(-20:20)
       do 20 i=1,raw
         do 20 j=1,col
            a2(i,j)=0
           do 20 k=ltap,mtap
            jk=j+k
            if(jk.le.0) jk=col+jk
            if(jk.gt.col) jk=jk-col
            a2(i,j)=a2(i,j)+a1(i,jk)*f(k)
 20    continue

       return
       end
c_____________________________________________________

       subroutine cinter(in,out,nraw,ncol)
       integer nraw,ncol
       real in(nraw,ncol),out(nraw*2,ncol)
       do 20 j=1,ncol
         do 20 i=1,nraw
             out(2*i-1,j)=in(i,j)
             out(2*i,j)=0.0
```

```
c             out(2*i,j)=in(i,j)
c             out(2*i-1,j)=0.0
 20     continue
        return
        end
c_____________________________________________________

        subroutine rinter(in,out,nraw,ncol)
        integer nraw,ncol
        real in(nraw,ncol),out(nraw,2*ncol)
        do 20 j=1,ncol
          do 20 i=1,nraw
              out(i,2*j-1)=in(i,j)
              out(i,2*j)=0.0
c             out(i,2*j)=in(i,j)
c             out(i,2*j-1)=0.0
 20     continue
        return
        end
c_______________________________________________________

       subroutine synthesis256(llband,lhband,hlband,hhband,outimg,
     +     inimg)
        character*1  pimm(400*512)
        real inimg(400,512)
        integer raw,col
        parameter(raw=400,col=512)
        parameter(nx=400,ny=512)

c input and output images
        real outimg(raw,col)


c these are the four subband
        real llband(raw/2,col/2),lhband(raw/2,col/2)
     &  ,hlband(raw/2,col/2),hhband(raw/2,col/2)



        real lli(raw,col/2),lhi(raw,col/2),hli(raw,col/2),
     &  hhi(raw,col/2),llo(raw,col/2),lho(raw,col/2),hlo(raw,col/2),
     &  hho(raw,col/2)
```

```
      real li(raw,col/2),lo(raw,col),hi(raw,col/2),
     & ho(raw,col)

      real  limg(raw,col),himg(raw,col)



      real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)

      common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     &       ,ltap3,mtap3,ltap4,mtap4

c________________________________
      nraw=raw
      ncol=col



      call cinter(llband,lli,raw/2,col/2)
      call ccfilter(coff4,lli,llo,raw,col/2,ltap4,mtap4)

      call cinter(lhband,lhi,raw/2,col/2)
      call ccfilter(coff3,lhi,lho,raw,col/2,ltap3,mtap3)

      call cinter(hlband,hli,raw/2,col/2)
      call ccfilter(coff4,hli,hlo,raw,col/2,ltap4,mtap4)

      call cinter(hhband,hhi,raw/2,col/2)
      call ccfilter(coff3,hhi,hho,raw,col/2,ltap3,mtap3)

c
      do 10 i=1,raw
        do 10 j=1,col/2
          li(i,j)=llo(i,j)+lho(i,j)
c           li(i,j)=llo(i,j)
c         hi(i,j)=hlo(i,j)+hho(i,j)
c           hi(i,j)=hlo(i,j)
            hi(i,j)=0.0
 10   continue

      call rinter(li,lo,raw,col/2)
      call rcfilter(coff4,lo,limg,raw,col,ltap4,mtap4)

      call rinter(hi,ho,raw,col/2)
```

```
      call rcfilter(coff3,ho,himg,raw,col,ltap3,mtap3)

        do 20 i=1,raw
        do 20 j=1,col
          outimg(i,j)=limg(i,j)
20      continue


c       open(99,file='im',access='direct',form='unformatted'
c    +  ,recl=nx*ny)
c      do  690 i=1,nx
c      do  691 j=1,ny
c      ip=int(outimg(i,j))+128
c      if(ip.gt.255) ip=255
c      if (ip.lt.0) ip=0
c      if(ip.gt.128) ip=ip-255
c      kk=j+(i-1)*ny
c      pimm(kk)=char(ip)
c691   continue
c690   continue
c      write(99,rec=1) (pimm(j),j=1,nx*ny)
c      close(99)

      return
      end

***************************************
* SYNTHESIS FILTER FOR  128*100 IMAGES*
***************************************

      subroutine synthesis128(llband,lhband,hlband,hhband,outimg)

      integer raw,col
      parameter(raw=200,col=256)

c input and output images
      real outimg(raw,col)


c these are the four subband
      real llband(raw/2,col/2),lhband(raw/2,col/2)
     &  ,hlband(raw/2,col/2),hhband(raw/2,col/2)
```

```
      real lli(raw,col/2),lhi(raw,col/2),hli(raw,col/2),
     & hhi(raw,col/2),llo(raw,col/2),lho(raw,col/2),hlo(raw,col/2),
     & hho(raw,col/2)


      real li(raw,col/2),lo(raw,col),hi(raw,col/2),
     & ho(raw,col)

      real  limg(raw,col),himg(raw,col)
      real coff1(-20:20),coff2(-20:20),coff3(-20:20),coff4(-20:20)

      common /a/ coff1,coff2,coff3,coff4,ltap1,mtap1,ltap2,mtap2
     &       ,ltap3,mtap3,ltap4,mtap4

c_________________________________
      nraw=raw
      ncol=col

      call cinter(llband,lli,raw/2,col/2)
      call ccfilter(coff4,lli,llo,raw,col/2,ltap4,mtap4)

      call cinter(lhband,lhi,raw/2,col/2)
      call ccfilter(coff3,lhi,lho,raw,col/2,ltap3,mtap3)

      call cinter(hlband,hli,raw/2,col/2)
      call ccfilter(coff4,hli,hlo,raw,col/2,ltap4,mtap4)

      call cinter(hhband,hhi,raw/2,col/2)
      call ccfilter(coff3,hhi,hho,raw,col/2,ltap3,mtap3)


c
      do 10 i=1,raw
        do 10 j=1,col/2
          li(i,j)=llo(i,j)+lho(i,j)
          hi(i,j)=hlo(i,j)
 10   continue

      call rinter(li,lo,raw,col/2)
      call rcfilter(coff4,lo,limg,raw,col,ltap4,mtap4)

      call rinter(hi,ho,raw,col/2)
      call rcfilter(coff3,ho,himg,raw,col,ltap3,mtap3)
```

```
      do 20 i=1,raw
        do 20 j=1,col
          outimg(i,j)=1*(limg(i,j)+himg(i,j))
 20     continue

      return
      end

************************************************************
```

REFERENCES

1. M. S. Kadur, " Adaptive Subband Video Coding with Motion Compensation," *M.Sc. Thesis*, NJIT, May 1989.

2. V. Seferidis and M. Chanbari, "Generalized Block Matching Motion Estimation," *SPIE, Visual Communications and Image Processing'92*, vol. 1818, pp.110–119, 1992.

3. Joe S. Lim, *Two–Dimensional Signal and Image Processing*, Prentice- Hall Inc., Englewood Cliffs, NJ, 1990.

4. N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1984.

5. A. N. Akansu and R. A. Haddad, *Multiresolution Signal Decomposition; Transforms, Subbands, and Wavelets*, Acedemic Press, Inc., San Diago, CA, 1992.

6. P. P. Vaidyanathan, "Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial,",*Proceedings of the IEEE*, vol. 78, no.1, pp.56–93, January 1990.

7. P. P. Vaidyanathan, " Quadrature Mirror Filter Banks, M-band Extentions and Perfect-Reconstruction Techniques,",*IEEE ASSP Magazine*, pp. 4–20, July 1987.

8. M. J. T. Smith and T .P. Barnwell, III,"A Procedure for Designing Exact Reconstruction Filter Banks for Tree Structured Subband Coders," *In Proc. IEEE int. conf. Acust., Speech, Signal Processing*, pp.27.1.1–27.1.4, San Diago, CA, March 1984.

9. A. Croisier, D. Esteban and C. Galand,"Perfect Channel Splitting by Use of Interpolation/Decimation/Tree Decomposition Techniques," *Int'l Conf. on Information Sciences and Systems*, Patras, Greece, 1976.

10. H. Gharavi and A. Tabatabai, "Application of Quadrature Mirror Filtering to the Coding of Monochrome and Color Images," *Proceedings of ICASSP*, pp. 2384–2387, Dallas, April 6–9, 1987.

11. Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. on Communications*, Vol. COM-28, no.1, pp. 84–85, January 1985.

12. J. Makhoul, S. Roucos and H. Gish, "Vector Quantization in Speech Coding," *Proc. of the IEEE*, vol.73, no.11, pp.1551-1588, Nov 1985.

13. H. M. Mutlag, "A comperative Study of Image Coding Techniques: Filter Banks vs. Discrete Cosine Transform," *M.Sc. Thesis*, NJIT, May 1991

14. J. D. Johnston, "A filter Family Designed for Use in Quadrature Mirror Filter Banks," *Int. Conf. on ASSP, ICASSP*, pp. 291–294, Denver, 1980.