5-31-1991

A comparative study of image coding techniques : filter banks vs. discrete cosine transform

Hosam Fawzi Mutlag
New Jersey Institute of Technology

Abstract

Title of Thesis: A Comparative Study of Image Coding Techniques: Filter Banks vs. Discrete Cosine Transform.

Hosam Mutlag, Master of Science, 1991

Thesis directed by: Dr. Ali N. Akansu

Subband coding of still image frames using Binomial PR-QMF has been presented in this thesis. Simulation results have shown that the performance of subband coding using a low complexity coder is practically the same as the performance of the industry standard (8×8) DCT based image coder.

A low bit rate adaptive video coding technique is also introduced in this thesis. The redundancy within adjacent video frames is exploited by motion compensated interframe prediction. The Motion Compensated Frame Difference (MCFD) signals are filtered by employing Binomial PR-QMF structure into four subbands. Then, the subbands are quantized using an efficient Motion Based Adaptive Vector Quantization (MBAVQ) algorithm. Here, the adaptation scheme is based on block motion vectors rather than local signal energy which was used in earlier works of several researchers. The new technique results in a reduction in bit rate by nearly (40%) due to the drop of the extra bits used for local variances. Moreover, for the video test sequances considered, MBAVQ method gives superior SNR results over local Variance Based Adaptive Vector Quantiztion (VBAVQ) scheme especially for high motion frames.

A Comparative Study of Image Coding Techniques: Filter Banks vs. Discrete Cosine Transform

by
Hosam Fawzi Mutlag

Thesis submitted to the Faculty of the Graduate School of
the New Jersey Institute of Technology in partial fulfillment of
the requirements for the degree of
Master of Science in Electrical Engineering

APPROVAL SHEET

Title of Thesis: A Comparative Study of Image Coding Techniques: Filter Banks vs. Discrete Cosine Transform

Name of Candidate: Hosam F. Mutlag
Master of Science, 1991

Thesis and Abstract Approved: ______________________ Date 5/10/91
Dr. Ali N. Akansu
Assistant Professor
Department of Electrical and Computer Engineering

Signature of Other Members of the Thesis Committee: ______________________ Date 5/10/91
Dr. Yeheskel Bar-Ness
Professor
Deparment of Electrical and Computer Engineering

______________________ Date 5/10/91
Dr. Zoran Siveski
Assistant Professor
Department of Electrical and Computer Engineering

VITA

Name: Hosam Fawzi Mutlag.

Permanent address:

Degree and date to be conferred: Master of Science in Electrical Engineering, 1991.

Date of birth:

Place of birth:

Secondary education: Al Shati High School, Jeddah, Saudi Arabia.

Collegiate institutions attended	Dates	Degree	Date of Degree
N. J. Institute of Technology	9/89-5/91	M.S.E.E.	May 1991
University of Petroleum and Minerals, Dhahran, Saudi Arabia.	9/82-5/87	B.S.E.E.	May 1987

Major: Electrical Engineering.

Acknowledgement

I would like to express my gratitude to Dr. Ali Akansu for his valuable contribution, advices, patience and understanding. Furthermore, I appreciate his support and encouragement during the entire research period.

I am greateful to Dr. Yeheskil Bar-Ness and Dr. Zoran Siveski for their effort and time in reviewing this work.

Thanks and regards to my parents for their moral support and love.

Last but not least, I would like to thank the members of the Center for Communication and Signal Processing Research at New Jersey Institute of Technology and to all my friends for their help which made the period of this work a pleasent one.

To my parents

Contents

List of Tables iii

List of Figures iv

1 Introduction 1

2 PR_QMF 4

2.1 Theoretical Derivations 4

2.2 The Binomial Family 7

2.3 The Binomial QMF 9

2.4 M-Band Tree Decomposition 10

2.5 Two Dimensional Separable Case 10

3 Discrete Cosine Transform 15

4 Optimum Bit Allocation and Gain of Transform Coding 17

4.1 Bit Allocation 17

4.2 Gain of Transform Coding 19

5 Quantization 20

5.1 Introduction 20

5.2 The Optimum Mean Square or Lloyd-Max Quantizer 21

5.2.1 Properties of the Optimum Mean Square Quantizer 22

5.3 Vector Quantization 22

6 Source Entropy and Huffman Coding **25**

6.1 Entropy 25

6.2 Entropy Coding: The Huffman Coding Algorithm 25

6.2.1 The Huffman Coding Algorithm 26

7 Experimental Studies **27**

7.1 Subband Coding of Still Images 27

7.2 Subband Video Coding with Motion Based Adaptive Vector Quantization 29

8 Discussions and Future Research **49**

Appendix A **50**

Bibliography **118**

List of Tables

7.1 Results from bit allocation algorithm 48

List of Figures

2.1 Two Channel QMF Bank. 11

2.2 Low-pass and high-pass QMF filters from Binomial Network. 11

2.3 Four-band tree decomposition for one dimensional signal $X(n)$ 12

2.4 Four-band tree decomposition of a two-dimensional signal $X(n,m)$ . . 14

2.5 Four-band reconstruction of a two-dimensional signal $X(n,m)$ 14

7.1 SNR versus entropy for 7 subband decomposition using Laplacian quantizers for all the subbands 32

7.2 SNR versus entropy for 10 subband decomposition using Laplacian quantizers for all the subbands 33

7.3 SNR versus entropy for 7 subband decomposition using DPCM for the low frequency band and Laplacian quantizers for the remaining subbands 34

7.4 SNR versus entropy for 7 subband decomposition using different quantization schemes for the lowest frequency subband 35

7.5 SNR versus entropy for DCT, 7 and 10 subband decomposition using Laplacian quantization scheme for the lowest frequency subband . . . 36

7.6 SNR versus entropy for DCT and 10 subband decomposition using DPCM quantization scheme for the lowest frequency subband 37

7.7 Variance vs Motion for MCFD frames 10, 24 and 30 of CINDY . . . 38

7.8 SNR vs frame index of MBAVQ and VBAVQ for CINDY 39

7.9 Entropy vs frame index of MBAVQ and VBAVQ for CINDY 40

7.10 Energy Compaction Gain, G_{TC} vs frame index for (8×8) DCT and 4-tap Binomial QMF subband structure for MCFD of CINDY 41
7.11 SNR vs frame index of MBAVQ and VBAVQ for MONO 42
7.12 Entropy vs frame index of MBAVQ and VBAVQ for MONO 43
7.13 Motion vs frame index of MBAVQ and VBAVQ for MONO 44
7.14 SNR vs frame index of MBAVQ and VBAVQ for DUO 45
7.15 Entropy vs frame index of MBAVQ and VBAVQ for DUO 46
7.16 Motion vs frame index of MBAVQ and VBAVQ for DUO 47

Chapter 1

Introduction

Typical video has spatial resolution of approximately 512×512 pixels per frame. At 8 bits per pixel per color channel and 30 frames per second, this translates into a rate of nearly 180×10^6 bits/s. The large channel capacity and memory requirements for digital image transmission and storage makes it desirable to consider data compression techniques. Image data compression is concerned with minimizing the number of bits required to represent an image with an acceptable visual quality. Compression can be achieved by transforming the signal, projecting it on a basis of functions, and then encoding the transform coefficients. These transforms vary from the conventional block transforms to ideal subband filter banks. The principle of subband coding has recently been successfully applied to data compression of both still images and video. In subband coding, the signal to be coded is decomposed into narrow band pass signals (subbands). Each subband is then accordingly subsampled and encoded with a bit rate matched to the signal statistics in that subband.

Subband coding was first introduced by Crochiere, et. al.[1] in speech coding and then extended to multidimensional signals by Vetterli[2]. Then, this concept was applied to the coding of images [3] [4]. Perfect Reconstruction Quadrature Mirror Filters (PR-QMF) have been proposed as structures suitable for subband coding [5][6]. These filters employing a tree decomposition structure provide a basis for a multi-resolution signal representation [7] [8]. Recently, discrete wavelet transforms have been proposed as a new approach for multi-resolution signal decomposition. More recently, Binomial QMF-Wavelet Transform has been proposed for multiresolution signal decomposition [10] [11].

In this thesis, we have studied the performance of Binomial QMF in subband coding with comparison to the Discrete Cosine Transform (DCT). Simulation programs were developed for a complete subband coding system (codec) consisting of the following four distinct parts:

- An analysis filter bank splitting the input signal into subbands.
- An encoder which consists of:
    1. Bit allocation algorithm.
    2. Quantizers.
    3. Huffman encoder.
- A decoder whose purpose is to produce an approximation to the original subband signals.
- The synthesis filter bank that combines the decoded subband signals, to reconstruct the received signal.

A similar DCT based codec is also developed for comparison purposes.

We also propose a new Motion Based Adaptive Vector Quantization (MBAVQ) approach for subband video coding. In video signals, interframe images have significant frame to frame redundancy. For that reason, in video coding techniques, video frames are motion compensated using an efficient search algorithm to remove temporal redundancies. The resulting prediction error signals, Motion Compensated Frame Difference (MCFD), are coded using an efficient coding scheme. In our video coding scheme, the MCFD signals are divided into four subbands. Least significant band (highest frequency), is neglected and the other bands are vector quantized adaptively.

This thesis is organized as follows. Chapter 2 describes the Binomial-QMF which have been used. In the following chapter, Discrete Cosine Transform is discussed. In chapter 4, optimum bit allocation and gain of transform coding over PCM is explained. Quantization is reviewed in chapter 5. The next chapter discusses source entropy and Huffman coding. In chapter 7, Motion Based Adaptive Vector Quantization (MBA-VQ) video coding scheme is introduced. Also, a comprehensive set of simulation results is presented in this chapter. Finally, we conclude the thesis with a discussion of the results and future research.

Chapter 2

PR_QMF

2.1 Theoretical Derivations

The QMF bank is a multirate digital filter bank. There are decimators in the system which down-sample a signal sequence, and there are interpolators which perform up-sampling. The input-output relation for a two-fold decimator can be written in the transform domain as

$$Y(e^{jw}) = \frac{1}{2}\left[X(e^{jw/2}) + X(-e^{jw/2})\right] \tag{2.1}$$

where $Y(e^{jw})$ has a period of 2π. The effect of compression in time domain is an expansion (or stretching) in frequency domain. The transform domain relation for a two-fold interpolator is

$$\begin{aligned} Y(e^{jw}) &= X(e^{jw2}) \\ Y(z) &= X(z^2) \end{aligned} \tag{2.2}$$

The interpolator causes compression in the frequency domain.

Let us consider Figure (2.1) in which a two-channel QMF system is shown. Based on relations (2.1) and (2.2), it is possible to express $\hat{X}(z)$ in Figure 1 as

$$\begin{aligned} \hat{X}(z) &= \frac{1}{2}\left[H_0(z)F_0(z) + H_1(z)F_1(z)\right]X(z) \\ &\quad + \frac{1}{2}\left[H_0(-z)F_0(z) + H_1(-z)F_1(z)\right]X(-z) \end{aligned} \tag{2.3}$$

The reconstructed signal can in general be subject to three types of distortions:

- Aliasing distortion.
- Phase distortion.
- Amplitude distortion.

The second term in (2.3) represents the effects of aliasing and imaging. This term can be made to drop simply by choosing the synthesis filters to be

$$F_0(z) = -H_1(-z) \tag{2.4}$$

$$F_1(z) = H_0(-z) \tag{2.5}$$

The QMF bank becomes a linear and time-invariant system with transfer function

$$T(z) = \frac{\hat{X}(z)}{X(z)} = \frac{1}{2}\left[H_0(z)H_1(-z) - H_1(z)H_0(-z)\right] \tag{2.6}$$

If $|T(e^{jw})| =$ constant for all w, then there is no amplitude distortion. Also, if $T(z)$ is a linear-phase FIR function, then $\arg[T(e^{jw})] = kw$, and there is no phase distortion. This means $T(z)$ is a delay, i.e., $T(z) = Cz^{-n_0}$, so that the reconstructed signal $\hat{x}(n)$ is a delayed version of $x(n)$.

Smith and Barnwell [6] have shown that one can simultaneously eliminate both amplitude and phase distortions by choosing $(2N-1)$ odd and

$$H_1(z) = z^{(2N-1)}H_0(-z^{-1}) \tag{2.7}$$

Therefore,

$$T(z) = \frac{1}{2}z^{-(2N-1)}\left[H_0(z)H_0(z^{-1}) - H_0(-z)H_0(-z^{-1})\right] = Cz^{-(2N-1)} \tag{2.8}$$

The perfect reconstruction requirement of a two-band QMF reduces to[5]

$$\begin{aligned} Q(z) &= H(z)H(z^{-1}) + H(-z)H(-z^{-1}) = \mathrm{C} \\ &= R(z) + R(-z) \end{aligned} \tag{2.9}$$

where $H(z)$ is a low-pass filter of length $2N$.

The PR requirement, Eq.(2.9), can be readily recast in an alternate, time domain form [10]. First, one notes that $R(z)$ is a spectral density function and hence is representable by a finite series of the form

$$R(z) = \gamma_{2N-1}z^{2N-1} + \gamma_{2N-2}z^{2N-2} + \ldots + \gamma_0 z^0 + \ldots + \gamma_{2N-1}z^{-(2N-1)} \tag{2.10}$$

Then

$$R(-z) = -\gamma_{2N-1}z^{2N-1} + \gamma_{2N-2}z^{2N-2} - \ldots + \gamma_0 z^0 - \gamma_1 z^{-1} \ldots - \gamma_{2N-1}z^{-(2N-1)} \tag{2.11}$$

Therefore $Q(z)$ consists only of even powers of z. To force $Q(z) = \mathrm{C}$ which is a constant, it suffices to make all even indexed coefficients in $R(z)$ equal to zero, except for n=0. However, the γ_n coefficients in $R(z)$ are simply the samples of the autocorrelation $\rho(n)$ given by

$$\begin{aligned} \rho(n) &= \sum_{k=0}^{2N-1} h(k)h(k+n) = \rho(-n) \\ &\stackrel{\text{def}}{=} h(n) \odot h(n) \end{aligned} \tag{2.12}$$

where $\odot$ indicates a correlation operation. This follows from the z-transform relationships

$$R(z) = H(z)H(z^{-1}) \longleftrightarrow h(n) * h(-n) = \rho(n) \tag{2.13}$$

where $\rho(n)$ is the convolution of $h(n)$ with $h(-n)$, or equivalently, the time autocorrelation, Eq.(2.12). Hence, we need to set $\rho(n) = 0$ for n even, and $n \neq 0$. Therefore,

$$\rho(2n) = \sum_{k=0}^{2N-1} h(k)h(k+2n) = 0, \qquad n \neq 0 \tag{2.14}$$

If the normalization is imposed,

$$\sum_{k=0}^{2N-1} |h(k)|^2 = 1 \tag{2.15}$$

one obtains the PR requirement in time

$$\sum_{k=0}^{2N-1} h(k)h(k+2n) = \delta_n \tag{2.16}$$

2.2 The Binomial Family

The binomial family of orthogonal sequences [13] [14] is generated by successive differencing of the binomial sequence, which is defined on the finite interval $[0, N]$ by

$$x_0(k) = \begin{cases} \binom{N}{k} = \frac{N!}{(N-k)!k!} & 0 \leq k \leq N \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

The other members of the binomial family are obtained from

$$x_r(k) = \nabla^r \binom{N-r}{k} \qquad r = 0, 1, \ldots, N \tag{2.18}$$

where

$$\nabla f(n) = f(n) - f(n-1)$$

this is the backward difference operator. Taking successive differences yields Bonomial function family of length (N+1)

$$x_r(k) \;=\; \binom{N}{k} \sum_{\nu=0}^{r} (-2)^{\nu} \binom{r}{\nu} \frac{k^{(\nu)}}{N^{(\nu)}} \tag{2.19}$$

where $k^{(\nu)}$ is the forward factorial function, a polynomial in k of degree ν

$$k^{(\nu)} = \begin{cases} k(k-1)\ldots(k-\nu+1) & \nu \geq 1 \\ 1 & \nu = 1 \end{cases} \tag{2.20}$$

This family of binomially-weighted polynomials has a number of properties. Taking z transform, we obtain

$$\begin{aligned} X_0(z) &= (1+z^{-1})^N \\ X_r(z) &= (1-z^{-1})^r (1+z^{-1})^{N-r} \end{aligned} \tag{2.21}$$

The binomial matrix X is the $(N+1) \times (N+1)$ matrix

$$X = [x_r(k)]$$

where $x_r(k)$ is the entry in the r^{th} row and k^{th} column. The salient property of this matrix is that the rows are orthogonal to the columns,

$$\sum_{k=0}^{N} x_r(k) x_k(s) = (2)^N \delta_{r-s} \tag{2.22}$$

or

$$X^2 = (2)^N I \tag{2.23}$$

Additionally, the Bionomial filters are linear phase quadrature mirror filters. From eq. (2.21), we see that

$$X_r(-z) = X_{N-r}(z) \tag{2.24}$$

which implies

$$(-1)^k x_r(k) = x_{N-r}(k) \quad r = 0, 1, \cdots, N \tag{2.25}$$

Also,

$$z^{-N} X_r(z^{-1}) = (-1)^r X_r(z) \tag{2.26}$$

implies

$$x_r(N - k) = (-1)^r x_r(k) \tag{2.27}$$

Equations (2.25) and (2.27) demonstrate the symmetry and asymmetry of the rows and columns of the Bionomial matrix X. From equation (2.25), we can infer that the complementary filters X_r and $X_{N-r}(z)$ have magnitude responses which are mirror images about $\omega = \pi/2$

$$|X_r(\exp j(\frac{\pi}{2} - \omega))| = |X_{N-r}(\exp j(\frac{\pi}{2} + \omega))| \tag{2.28}$$

Moreover, the cross-correlation of the sequence $x_r(n)$, and $x_s(n)$ is defined as

$$\rho_{rs} = x_r(n) * x_s(-n) = \sum_{k=0}^{N} x_r(k) x_s(n+k) \rightleftharpoons R_{rs}(z) \tag{2.29}$$

and

$$R_{rs}(z) = X_r(z^{-1}) X_s(z) \tag{2.30}$$

Now for any real crosscorrelation,

$$\rho_{rs}(-n) = \rho(n) \qquad \forall s, r \tag{2.31}$$

Also, the quadrature mirror property of Eq.(2.25) implies that

$$\begin{aligned} \rho_{rs}(n) &= -\rho_{rs}(n) \qquad (s-r) \ \text{is odd} \\ \rho_{rs}(n) &= \rho_{rs}(n) \qquad (s-r) \ \text{is even} \end{aligned} \tag{2.32}$$

These properties are subsequently used in driving the perfect reconstruction Bionomial QMF in the next section.

2.3 The Binomial QMF

Now, it is a straight forward matter to impose PR condition of Eq.(2.16) on the binomial family [10]. First, the half-band filter is

$$h(n) = \sum_{r=0}^{\frac{N-1}{2}} \theta_r x_r(n)$$

or in the z transform

$$H(z) = \sum_{r=0}^{\frac{N-1}{2}} \theta_r (1+z^{-1})^{N-r}(1-z^{-1})^r = (1+z^{-1})^{(N+1)/2} F(z) \tag{2.33}$$

where $F(z)$ is FIR filter of order $(N-1)/2$. For convenience, take $\theta_0 = 1$, and later impose the normalization of Eq.(2.15). Substituting (2.33) into (2.12) gives

$$\begin{aligned} \rho(n) &= \left(\sum_{r=0}^{\frac{N-1}{2}} \theta_r x_r(n) \right) \odot \sum_{s=0}^{\frac{N-1}{2}} \theta_s x_s(n) \\ &= \sum_{r=0}^{\frac{N-1}{2}} \sum_{s=0}^{\frac{N-1}{2}} \theta_r \theta_s [x_r(n) \odot x_s(n)] \\ &= \sum_{r=0}^{\frac{N-1}{2}} \sum_{s=0}^{\frac{N-1}{2}} \theta_r \theta_s \rho_{rs}(n) \\ &= \sum_{r=0}^{\frac{N-1}{2}} \theta_r^2 \rho_{rr}(n) + \sum_{r=0, r\neq s}^{\frac{N-1}{2}} \sum_{s=0}^{\frac{N-1}{2}} \theta_r \theta_s \rho_{rs}(n) \end{aligned} \tag{2.34}$$

Eq.(2.32) implies that the second summation in Eq.(2.34) has only terms where the indices differ by an even integer. Therefore the autocorrelation for the binomial half-band low-pass filter is

$$\rho(n) = \sum_{n=0}^{\frac{N-1}{2}} \theta_r^2 \rho_{rr}(n) + 2 \sum_{l=1}^{\frac{N-3}{2}} \sum_{\nu=0}^{\frac{N-1}{2}-2l} \theta_\nu \theta_{\nu+2l} \, \rho_{\nu,\nu+2l}(n) \tag{2.35}$$

Finally, the PR requirement is

$$\rho(n) = 0, \qquad n = 2, 4, \ldots, N-1 \tag{2.36}$$

This condition gives a set of $\frac{N-1}{2}$ nonlinear algebraic equations, in the $\frac{N-1}{2}$ unknowns $\theta_1, \theta_2 \ldots, \theta_{\frac{N-1}{2}}$. From these, we can obtain the corresponding Binomial PR-QMFs.

These filters can be implemented using either the purely FIR structure, or the pole-zero cancellation configuration. The latter is shown in Fig.2 for $N = 5$. Wherein both low-pass and high-pass filters are simultaneously realized. Coefficient θ_0 can be taken equal to unity, leaving only θ_1 and θ_2 as tap weights. These are the only multiplications needed when using the Binomial network as the half-band QMF rather than the six $h(n)$ weights in a transversal structure.

2.4 M-Band Tree Decomposition

After a given signal $x(t)$ is sampled at f_s to give a signal $X(n)$, and split it into two signals $X_L(n)$ and $X_H(n)$, with the reduction of the sampling rate to $f_s/2$, the decomposition can be extended to more than two subbands by processing these signals in the same manner as the initial signal $X(n)$. Four signals are thus obtained with a reduction of the sampling rate to $f_s/4$. The decomposition/reconstruction can be generalized by repeating n times the previously described splitting using n-stage tree decomposition. In figure (2.3), a four subband decomposition is shown. The signal $X(n)$ is split into four signals X_{LL}, X_{LH}, X_{HL}, and X_{HH}.

2.5 Two Dimensional Separable Case

The extention of the QMF decomposition/reconstruction concept to two dimentional case is relatively straight forward in separable filter case. The conditions required for splitting two dimensional signals into more than two bands, with alias-free recon-

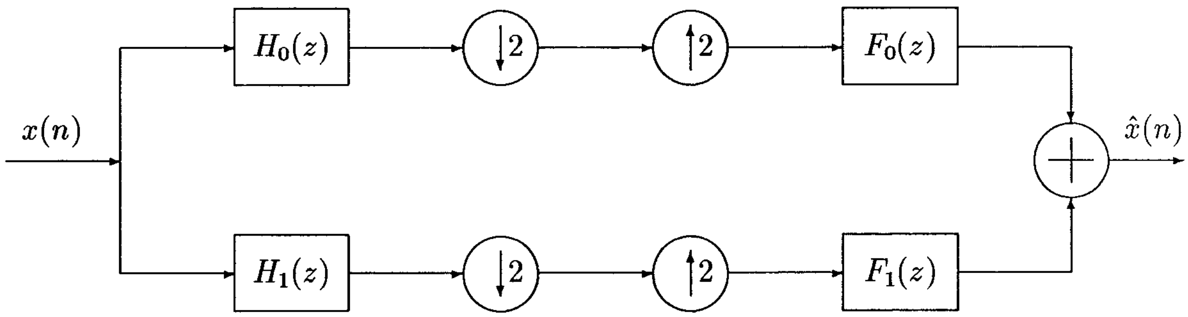

Figure 2.1: Two Channel QMF Bank.

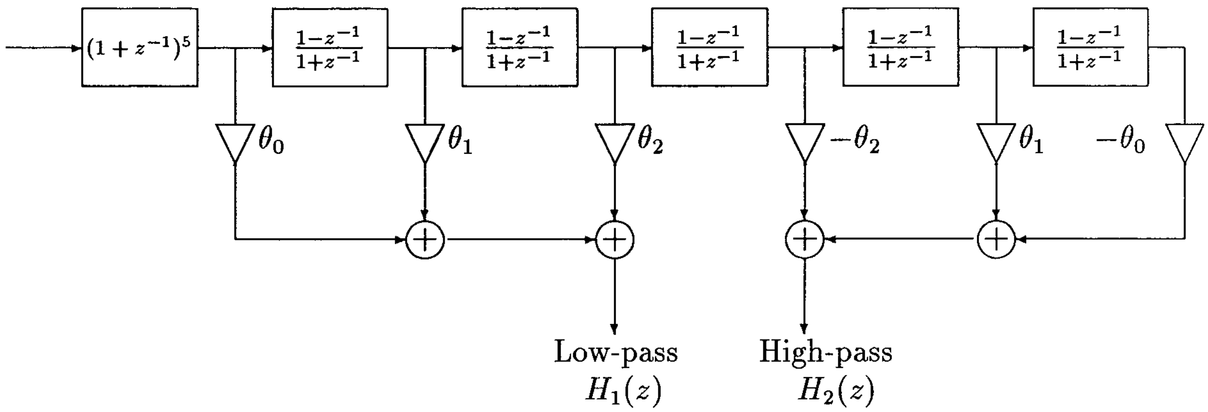

Figure 2.2: Low-pass and high-pass QMF filters from Binomial Network.

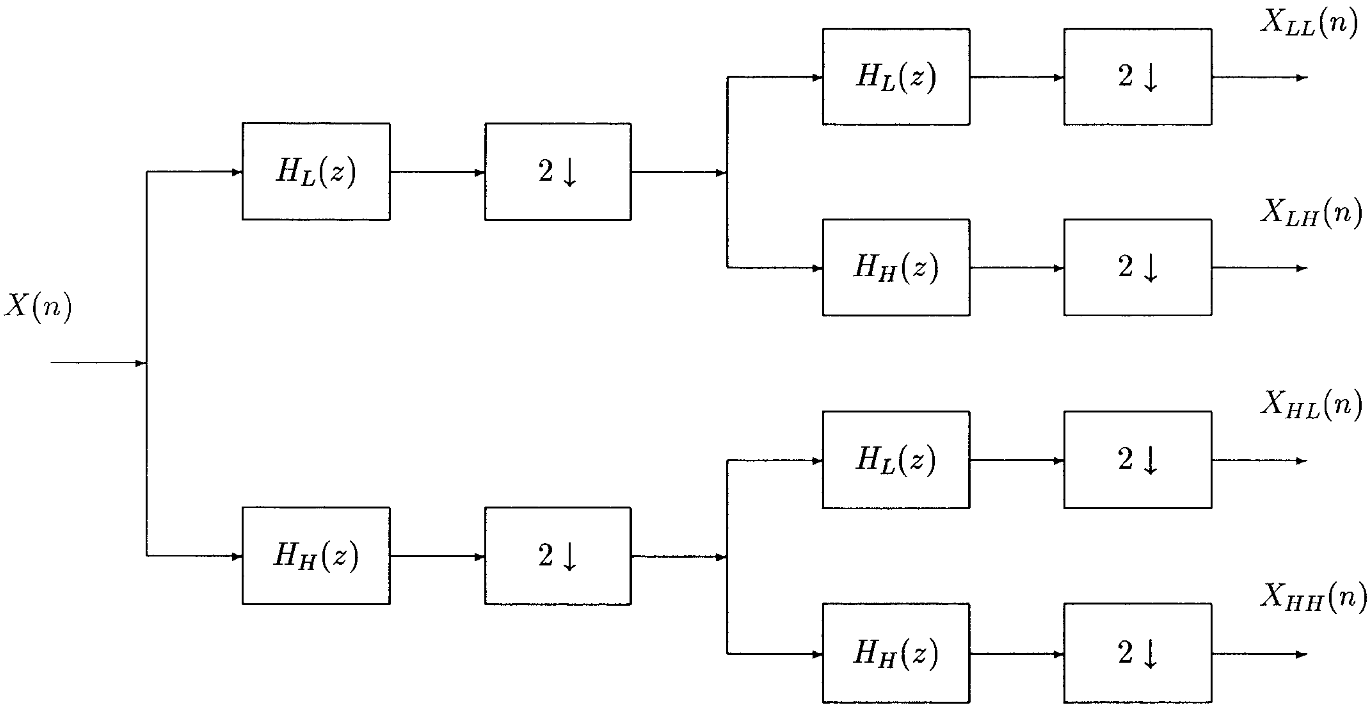

Figure 2.3: Four-band tree decomposition for one dimensional signal $X(n)$

struction of the original signal, is the separability feature of the filters, that is

$$h(n_1, n_2) = h_1(n_1)h_2(n_2) \tag{2.37}$$

These separability characteristics of the filter provide an alternative method of implementation for 2D-QMF banks. The analysis and synthesis are done as shown in figure (2.4) and figure (2.5) where the structure consists of one-dimensional filters. The computation is performed first along one axis (rows) and then along the other axis (columns). It can be shown that application of this filter structure will permit an alias-free reconstruction of the input signal at the receiver. The detailed analysis/synthesis and perfect reconstruction are given in [3]. Figure (2.4) shows four-band tree decomposition of a two-dimensional signal $X(n, m)$ while figure (2.5) shows the construction tree.

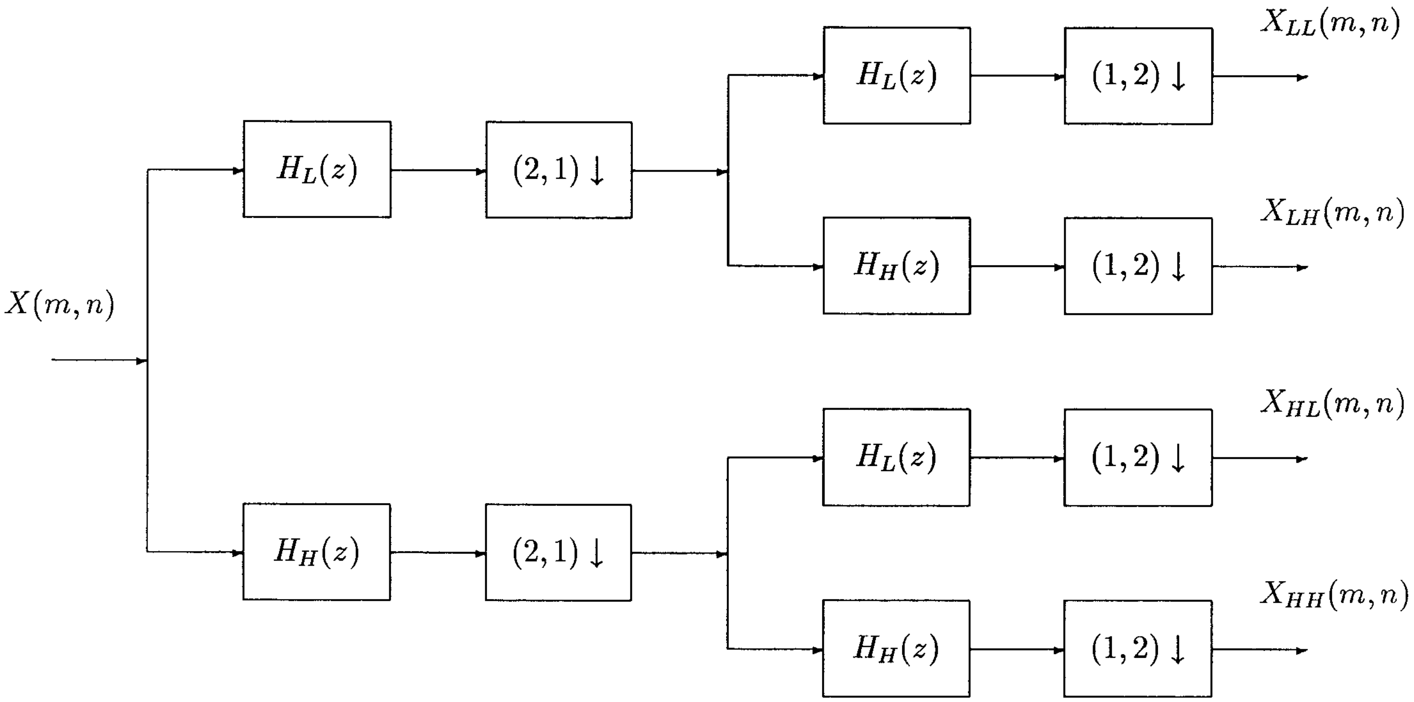

Figure 2.4: Four-band tree decomposition of a two-dimensional signal $X(n,m)$

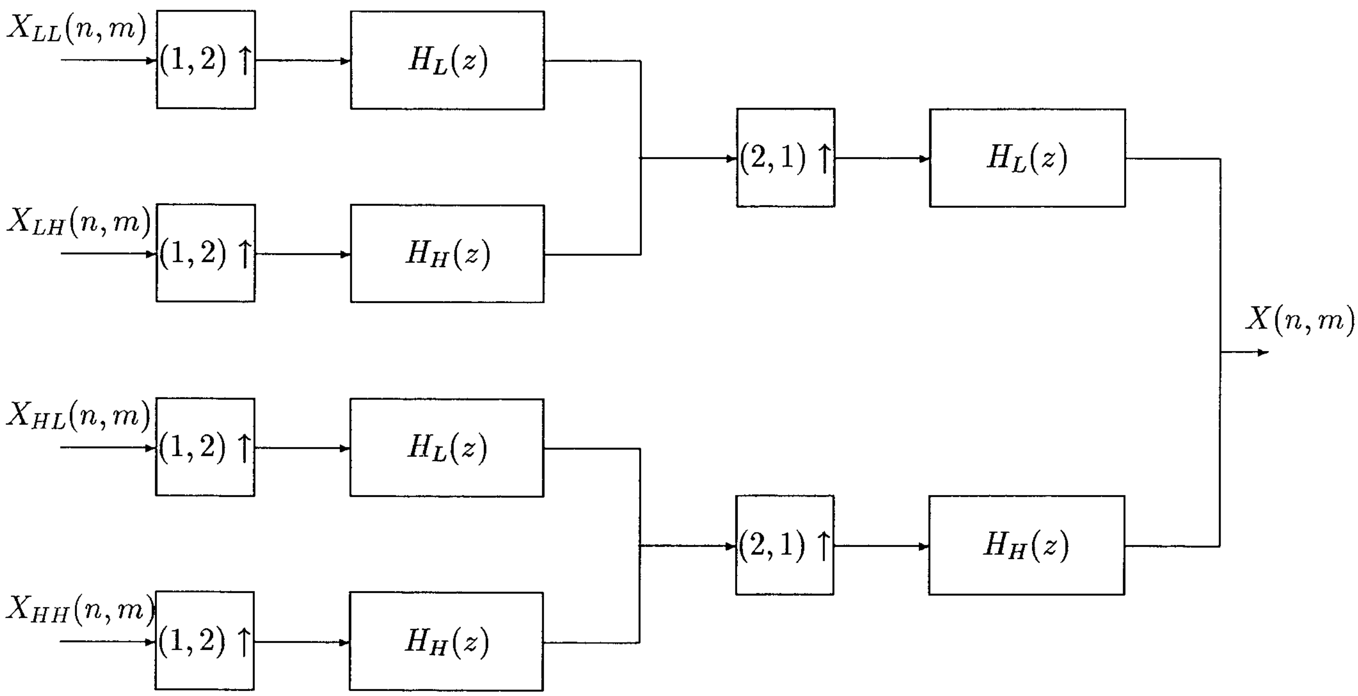

Figure 2.5: Four-band reconstruction of a two-dimensional signal $X(n,m)$

Chapter 3

Discrete Cosine Transform

The term image transform usually refers to a class of unitary matrices used for representing images. Just as one-dimensional signal can be represented by an orthogonal series of basis functions, an image can also be expanded in terms of a discrete set of basis arrays called basis images. These basis images can be generated by unitary matrices. To make transform coding practical, a given image is divided into small rectangular blocks, and each block is transform coded independently. For an $N \times M$ image divided into NM/pq blocks, each of size $p \times q$, the main storage requirements for implementing the transform are reduced by afactor of MN/pq. The computational load is reduced by a factor of $\log_2 MN/\log_2 pq$ for fast transform requiring $\propto N \log_2$ operations to transform an $N \times 1$ vector. For 256×256 images divided into 8×8 blocks, these factors are 1024 and 2.66, respectively. Although the operation count is not greatly reduced, the complexity of the hardware for implementing small size transform is reduced significantly. However, smaller block sizes yield lower compression. The $N \times N$ discrete cosine transform (DCT) matrix $C = \{c(k,n)\}$, is defined as[16]

$$c(k,n) = \begin{cases} \frac{1}{\sqrt{N}} & k = 0, 0 \leq n \leq N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{\pi(2n+1)}{2N} & 1 \leq k \leq N-1, 0 \leq n \leq N-1 \end{cases} \tag{3.1}$$

The two-dimensional discrete cosine transform of a two-dimensional sequence

$\{u(m,n), 0 \leq m \leq N-1, 0 \leq n \leq N-1\}$, is obtained by

$$v(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} c(k,m)u(m,n)c(l,n) \tag{3.2}$$

The inverse transformation is given by

$$u(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} c(k,m)v(k,l)c(l,n) \tag{3.3}$$

or in vector notation

$$V = CUC^T \tag{3.4}$$

and

$$U = C^T VC \tag{3.5}$$

The cosine transform is real and orthogonal, that is,

$$C = C^* \rightarrow C^{-1} = C^T \tag{3.6}$$

To implement one-dimensional DCT of size N using fast transform algorithms, it requires $O(N \log N)$ operations, where one operation is a real multiplication and a real addition. For two-dimensional DCT of size $N \times N$, the transformation is equivalent to $2N$ one-dimensional DCT. Hence, the total number of operation is $O(N^2 \log_2 N)$.

Chapter 4

Optimum Bit Allocation and Gain of Transform Coding

4.1 Bit Allocation

Bit allocation is a major concern in any coding scheme where a given quota of bits must be efficiently distributed among a number of different quantizers. Encoding in subbands offers several advantages. By appropriately allocating the bits in different bands, the number of quantizer levels and hence reconstruction error variance can be separateley controlled in each band and the shape of the overall reconstruction error spectrum can be controlled as a function of frequency. Using this approach, the noise spectrum can be shaped according to the subjective noise perception of the human eye. Moreover, the error in coding a subband is confined to that subband.

The overall MSE incurred in an orthonormal, equal bandwidth, subband coding scheme with K subbands is given by

$$D = \sum_{i=1}^{K} D_i(r_i), \tag{4.1}$$

where $D_i(r_i)$ is the distortion rate performance of the encoder operating on the ith subband at r_i bits/sample. The average bit rate is given by

$$R = \frac{1}{K}\sum_{i=1}^{K} r_i \tag{4.2}$$

In equation (4.2) we have assumed that all subbands have the same number of pixels.

If this is not the case, the r_i's should be multiplied by appropriate weighting coefficients to account for the variability in the number of pixels in any subband. The bit allocation we have used is based on the Lagrange multiplier technique in which it is assumed that all bands have the same pdf type and the distortion rate performance of the quantizers are given by

$$D_i(r_i) = \epsilon_i^2 \sigma_i^2 2^{-2r_i} \tag{4.3}$$

where, σ_i^2 is the variance of the ith subband and ϵ_i^2 is the quantization correction factor for that band. If we assume equal band distortions $D_i = D_j$, the following bit allocation is obtained [16]

$$r_i = r_{ave} + \frac{1}{2}\log_2 \left[\frac{\sigma_i^2}{\left[\prod_{i=1}^{K} \sigma_i^2\right]^{\frac{1}{K}}} \right] \tag{4.4}$$

where r_{ave} is the average bit rate.

$\{r_i\}$ are not restricted to be non-negative here. In practice, they are truncated to zero if they become negative. A negative bit allocation result implies that if that band is completely discarded its reconstruction error contribution is still less than the corresponding distortion for the given rate. Equation (4.4) holds only for regular tree structures of subband coding. For irregular tree structure, with N_1 bands in the first level of the tree and only band p is decomposed further into N_2 bands in the second level of the tree, the corresponding optimum bit allocation expressions are found as [18]

$$r_{k1} = r_{ave} + \frac{1}{2}\log_2 \frac{\sigma_{k1}^2}{\left[\prod_{k1=1,k1\neq p}^{N_1} \sigma_{k1}^2\right]^{\frac{1}{N_1}} \left[\prod_{k2=1}^{N_2} \sigma_{pk2}^2\right]^{\frac{1}{N_1 N_2}}} \tag{4.5}$$

and,

$$r_{pk2} = r_{ave} + \frac{1}{2}\log_2 \frac{\sigma_{pk2}^2}{\left[\prod_{k1=1,k1\neq p}^{N_1} \sigma_{k1}^2\right]^{\frac{1}{N_1}} \left[\prod_{k2=1}^{N_2} \sigma_{pk2}^2\right]^{\frac{1}{N_1 N_2}}} \tag{4.6}$$

4.2 Gain of Transform Coding

An N band orthonormal transform implies the variance preservation condition:

$$\sigma_x^2 = \frac{1}{N} \sum_{k=0}^{N-1} \sigma_k^2 \tag{4.7}$$

where σ_x^2 is the input signal variance with zero mean and σ_k^2 are the band variances. All orthonormal, variance preserving, signal decomposition techniques can be evaluated by employing the energy compaction criterion, namely the gain of transform coding over pulse code modulation G_{TC} [16]. If we assume that all the bands and the input signal have the same *pdf* type, the distortion ratio of PCM over transform coding at the same bit rate can be easily obtained as

$$G_{TC} = \frac{D_{PCM}}{D_{TC}} = \frac{\sigma_x^2}{\left[\prod_{k=1}^{N} \sigma_k^2\right]^{\frac{1}{N}}} \tag{4.8}$$

This equation holds for regular binary tree structures of subband technique. For irregular tree case equation (4.8) can be modified properly. Assuming an N_1 band in the first level of the tree, and only band p is decomposed further into N_2 bands in the second level of the tree, the corresponding gain is [18]

$$G_{TC} = \frac{\sigma_x^2}{\left(\left[\prod_{k1=1,k1\neq p}^{N_1} \sigma_{k1}^2\right]\left[\prod_{k2=1}^{N_2} \sigma_{pk2}^2\right]^{\frac{1}{N_2}}\right)^{\frac{1}{N_1}}} \tag{4.9}$$

Chapter 5

Quantization

5.1 Introduction

The subsequent step is quantization. Quantization is the most common form of data compression and is fundamental to any digitization scheme or data compression system. Quantization causes entropy reduction and hence it is an irreversible operation. A quantizer maps a continuous variable u into a discrete $\dot{u}$, which takes values from a finite set $\{r_1, \cdots, r_L\}$ of numbers. This mapping is generally a stair case function and the quantization rule is as follows:

- Define $\{t_k, k = 1, .. \cdots, L+1\}$ as a set of increasing transition or decision levels with t_1 and t_{L+1} as the minimum and maximum values, respectively, of u.
- If u lies in interval $[t_k, t_{k+1})$, then it is mapped to r_k, the kth reconstruction level.

We will consider only zero memory quantizers, which operate on one input sample at a time, and the output value depends only on that input. Each quantizer is operating at a different rate by using different quantization tables. The rate is usually measured by

$$R_k = \log_2 N_k \tag{5.1}$$

where R_k and N_k are the rate and the size of the table for the kth quantizer respectively. In zero memory quantizer (scalar quantization), the quantization is performed such that quantization noise is minimized, with a given number of quantization levels.

5.2 The Optimum Mean Square or Lloyd-Max Quantizer

This quantizer minimizes the mean square error for a given number of quantization levels. Let u be a real scalar random variable with a continuous probability density function $p_u(u)$. It is desired to find the decision level t_k and the reconstruction levels r_k for an L level quantizer such that the mean square error [17]

$$\mathcal{E} = E[(u-\dot{u})^2] = \int_{t_1}^{t_{L+1}} (u-\dot{u})^2 p_u(u)du \tag{5.2}$$

is minimized. Rewriting this as

$$\mathcal{E} = \sum_{\imath=1}^{L} \int_{t_\imath}^{t_\imath+1} (u-r_\imath)^2 p_u(u)du \tag{5.3}$$

By differentiating with respect to t_k and r_k and equating the result to zero, we get

$$\frac{\partial \mathcal{E}}{\partial t_k} = (t_k - r_{k-1})^2 p_u(t_k) - (t_k - r_k)^2 p_u(t_k) = 0 \tag{5.4}$$

$$\frac{\partial \mathcal{E}}{\partial r_k} = 2\int_{t_k}^{t_{k+1}} (u - r_k) p_u(u) du = 0 \quad 1 \le k \le L \tag{5.5}$$

Using the fact that $t_{k-1} \le t_k$, this gives

$$t_k = \frac{(r_k + r_{k-1})}{2} \tag{5.6}$$

$$r_k = \frac{\int_{t_k}^{t_{k+1}} u p_u(u) du}{\int_{t_k}^{t_{k+1}} p_u(u) du} = E[u | u \epsilon \mathcal{J}_k] \tag{5.7}$$

where $\mathcal{J}_k$ is the kth interval $[t_k, tk+1)$. These results state that the optimum transition levels lie halfway between the optimum reconstruction levels, which, in turn, lie at the center of mass of the probability density in between the transition levels. Both (5.6) and (5.7) are nonlinear equations that have to be solved simultaneously. In practice, these equations can be solved by an iterative scheme such as the Newton method.

A commonly used probability densities for quantization of image-related data is the Laplacian densities, which is defined as,

$$p_u = \frac{\alpha}{2} \exp(-\alpha|u - \mu|) \tag{5.8}$$

where μ denotes the mean of u. The variance is given by

$$\sigma^2 = \frac{2}{\alpha} \tag{5.9}$$

5.2.1 Properties of the Optimum Mean Square Quantizer

This quantizer has several interesting properties.

- The quantizer output is an unbiased estimate of the input, that is,

$$E[\dot{u}] = E[u] \tag{5.10}$$

- The quantization error is orthogonal to the quantizer output, that is,

$$E[(u - \dot{u})\dot{u}] = 0. \tag{5.11}$$

- The variance of the quantizer output is reduced by the factor $1 - f(B)$, where $f(B)$ denotes the mean square distortion of the B-bit quantizer for unity variance inputs, that is,

$$\sigma_{\dot{u}}^2 = [1 - f(B)]\sigma_u^2 \tag{5.12}$$

- It is sufficient to design mean square quantizers for zero mean unity variance distributions.

5.3 Vector Quantization

When Shannon [20] proved that signals from an information source could be coded at a bit rate no greater than the entropy of the source, he showed that this would be achieved, not by coding individual samples but by collecting samples into groups and encoding the groups. This is the basis of vector quantization.

A vector quantization scheme involves an encoder, a decoder and a codebook. The codebook is a lookup table with a k-bit address and 2^k entries. Each entry in the table is a vector of samples. Both the encoder and the decoder have copies of the codebook. The encoder looks at each incoming vector of samples $\bar{x}$ and selects the code-word $\bar{y} = \bar{C}(i)$ which is the best match to $\bar{x}$. Formally, let $d(\bar{x}, \bar{y})$ be the distortion measure. The Euclidean distance $(\bar{x} - \bar{y})^T(\bar{x} - \bar{y})$ is frequently used as the measure but any positive definite quadratic form in $(\bar{x} - \bar{y})$ will serve. The encoder transmits that vector index i for which $d(\bar{x}, \bar{C}(i))$ is minimum. The decoder receives i and presents $\bar{C}(i)$ as its output. Clearly the success of this technique depends on having a well chosen codebook. This codebook is, in general, not complete: i.e., the number of vectors in the codebook is finite while the number of possible vectors is usually, for all practical purposes, infinite. The distortion we must live with is that which arises from assigning an incoming vector to a codebook entry that does not quite match. The quantization process is reduced to a simple search and comparison procedure. The major difficulty in implementing vector quantizers is the time required to search a very large codebook. There are several algorithms for designing a codebook. We employed LBG algorithm which is summarized in the next subsection.

Lloyd's Algorithm

A procedure for designing codebooks was developed by Lloyd [21]. This algorithm was designed as a clustering technique for use in pattern recognition and related fields. It was extended to vector quantization by Linde, Buzo, and Gray [22] and therefore called LBG algorithm. The algorithm consists of an iterative technique for refining an initial codebook. The algorithm is as follows:

1. Encode a selection of test data with the current codebook and measure the average distortion. If the distortion is small enough the algorithm terminates.

2. For each address i in the codebook, find the centroid of all the input vectors which were mapped into i and make this centriod the new $\bar{C}(i)$. Go to step 1.

The test data used in this process are refered to as the training set. Each word in the codebook is used to represent a cluster of possible input vectors from this set.

Chapter 6

Source Entropy and Huffman Coding

6.1 Entropy

The entropy of a source with L symbols is defined as the average information generated by the source, i.e., Entropy

$$H = -\sum_{k=1}^{L} p_k \log_2 p_k \qquad \text{bits/symbol} \tag{6.1}$$

where p_k is the probability of having kth symbol. For a digital image considered as a source of independent pixels, its entropy can be estimated from its histogram. For the given L levels, the entropy of a source is maximized for uniform distributions, i.e., $p_k = 1/L, k = 1, \cdots, L$. In that case

$$\max H = -\sum_{k=1}^{L} \frac{1}{L} \log_2 \frac{1}{L} = \log_2 L \qquad \text{bits} \tag{6.2}$$

The entropy of a source gives the lower bound on the number of bits required to encode its output. Obviously, this definition assumes a stationary source.

6.2 Entropy Coding: The Huffman Coding Algorithm

The quantizer output is generally coded by a fixed-length binary code word having B bits. If the quantized pixels are not uniformaly distributed, then their entropy will be less than B, and there exists a code that uses less than B bits per pixel. In entropy, the goal is to encode a block of M pixels containing MB bits with probabilities $p_i, i = 0, 1, \cdots, L-1, L = 2^{MB}$, by $(-\log_2 p_i)$ bits, so that the average bit rate is

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

This gives a variabe-length code for each block, where highly probable blocks (or symbols) are represented by small length codes, and vice versa. If $(-\log_2 p_i)$ is not an integer, the achieved rate exceeds H but approaches it asymtotically with increasing block size. For a given block size, a technique called *Huffman coding* is the most commonly used fixed to variable length encoding method [17].

6.2.1 The Huffman Coding Algorithm

1. Arrange the symbol probabilities p_i in a decreasing order and consider them as leaf nodes of a tree.

2. While there is more than one node:

   - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes.

   - Arbitrarily assign 1 and 0 to each pair of branches merging into a node.

3. Read sequentially from the root node to the leaf node where the symbol is located.

The preceding algorithm gives the *Huffman code book* for any given set of probabilities. Coding and decoding is done simply by looking up values in a table.

Chapter 7

Experimental Studies

Since we are emphasizing the low complexity of the Binomial filter banks introduced in[10], it would be attractive to apply these filters in complete coder structures characterized by low overall complexity. Both still image coders and video coders are implemented and tested.

7.1 Subband Coding of Still Images

In a still image coder, the original image constitutes the input to the analysis filter bank. The subband signals are represented in a bit efficient manner using optimum bit allocation expressions given in equations (4.5) and (4.6). A problem with these two equations is that they may give negative values. Moreover, the r_{kj} may not match any of the existing quantizers bit rate. Actually, we need $2^{r_{kj}}$ to be an integer equales to the number of levels in one of the used quantizers.

To solve these problems, subbands with negative r_{kj} are truncated to zero and dropped from our future calculations. Then the average bit rate r_{ave} is recalculated for the remaining bands. After that, a quickly converging iterative algorithm has been developed to find the integer number of levels corresponding to each of the remaining bands.

- First, we start with the band which has the highest frequency. The bit rate corresponding to this band is calculted using (4.5), (4.6) and the new average bit rate r_{ave} recalculated previously.

- Then, choose the quantizer with the number of levels equals or less than $2^{r_{kj}}$.
- Excessive bit rate resulting from this integer truncation is used to reevaluate average bit rate r_{ave} for the remaining bands and this band is dropped from our next calculations.
- This new average bit rate r_{ave} is used in equations (4.5) and (4.6) to calculate the number of levels for the next lower band by repeating the same preceding procedure.

We should notice that by assigning the excessive bits, resulting from truncation, to lower bands, we are putting more emphasis on low frequencies. This feature enhances the image quality because of the mechanism of the human visual system which is more sensitive to low frequencies. For different average bit rates and the "Lady" test image, this adaptive bit allocation results are shown in Table (7.1). These numbers reflect the iterative adjustment to remove negative and noninteger values.

Next step is to quantize each band using the corresponding number of levels found using the Bit Allocation Algorithm. The bands are quantized using normalized Laplacian quantizers [17] with the following numbers of levels: $3, 5, 7, \cdots, 29, 31, 33, 35, 64, 128$. In this thesis, we have considered two methods of encoding the lowest frequency band; differential quantization such as DPCM and PCM quantization. In our experiments, we observed that the pixel to pixel correlation of upper band signals is very low. Consequently for these bands, PCM quantizer schemes are preferred to differential coding methods. We have therefore applied differential coding to the lowest band of the input signal in our experiments. In the DPCM encoder used here, a linear predictor constructs its predicted pixel as the weighted summation of previous pixels. Thus:

$$\bar{u}(m,n) = \frac{1}{4}[u(m-1,n) + u(m,n-1) + u(m-1,n-1) + u(m-1,n+1)] \quad (7.1)$$

The prediction error is then quantized by a symmetric non-uniform quantizer.

After quantization, the bands are entropy coded using Huffman coding scheme. To reduce complexity, only first order Huffman coding is used to code each quantized band.

Now, we can transmit the coded image over the channel. Here, we assume error free channel. It is suitable to use progressive transmission scheme for subband coding. The main objective of progressive transmission is to allow the receiver to recognize a picture as quickly as possible at minimum cost, by sending a low resolution level picture first. Further details of the picture are obtained by sequentially receiving the remaining encoded bands. At the receiver, the bands are decoded and reconstructed using the synthesizing filter bank.

A similar 8×8 two-dimensional DCT based coder is also developed for comparison purposes.

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

$$SNR_{pp}(dB) = 10 \log_{10} \left[\frac{255^2}{mse} \right] \tag{7.2}$$

where mse is the mean square coding error.

7.2 Subband Video Coding with Motion Based Adaptive Vector Quantization

In our low complexity video coder, we employ block matching for motion compensation using Brute-force method. The block size is set to 8×8 and the maximum displacement, both horizontally and vertically, is set to ± 6 pixels. The motion compensated frame difference (MCFD) signal is encoded with subband coding. The MCFD signal is split into four subbands, namely LL - HL - LH - HH bands, using 4-tap separable Binomial PR-QMF filters. The subbands are vector quantized adaptively after discarding the highest frequency band HH.

Each 4×4 block in the subbands corresponds to a motion vector of a block with size 8×8 in the original resolution. From experiments, we have found that there is a relation between motion vector magnitude and prediction error variance of the corresponding block. In general, large magnitude motion vectors represent high variance blocks in MCFD while blocks with no motion have small variances. Using this relation, vector quantizers (codebook) were classified depending on the magnitude of block motion. The magnitude of block motion $\hat{m}$ is defined as follow:

$$\hat{m} = \max(|i|, |j|) \tag{7.3}$$

where i and j are the horizontal and vertical motion respectively. We classified motion into three groups as follow:

- Group 1: $\hat{m} = 1$ or 2.
- Group 2: $\hat{m} = 3$ or 4.
- Group 3: $\hat{m} = 5$ or 6.

Using blocks corresponding to these groups for each of the three bands, codebooks were generated using LBG algorithm. As a result of this, we got 9 codebooks each of 512 length. This technique were compared to the work done by Kadur in his thesis [23][19]. Kadur used local variances of each block in subbands to classify them. This means that we need to transmit extra bits for local variances for each 4×4 block. The total bit rate for this technique can be written as

$$B = B_M + B_{SB} + B_G \tag{7.4}$$

where

- B_M = average bits/pixel for motion information.
- B_{SB} = average bits/pixel for subband signal.

- B_G = average bits/pixel for the transmission of local variances.

Using our MBAVQ scheme, B_G is not needed. This means a significant reduction in bit rate (40%) in general.

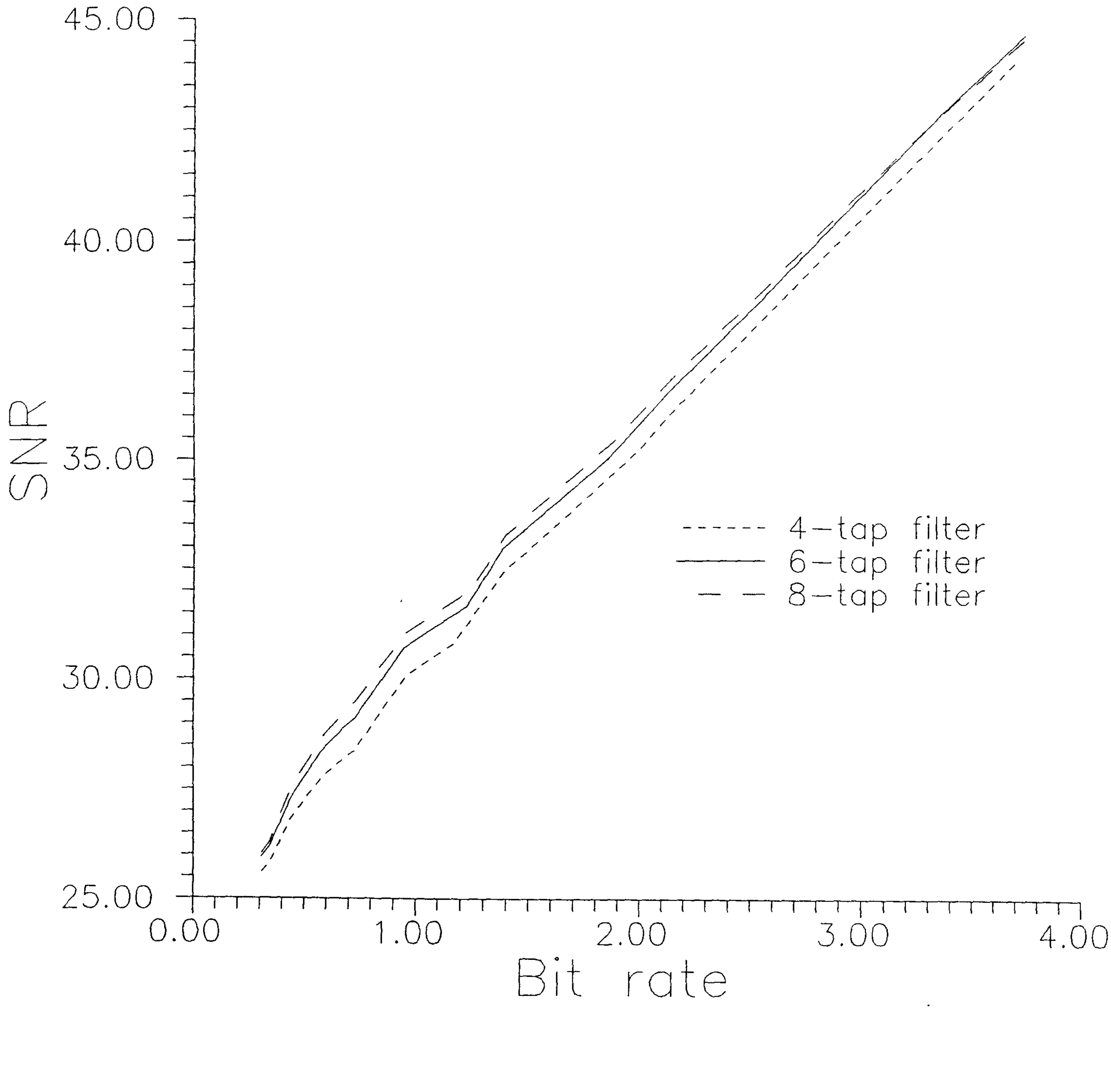

Figure 7.1: SNR versus entropy for 7 subband decomposition using Laplacian quantizers for all the subbands

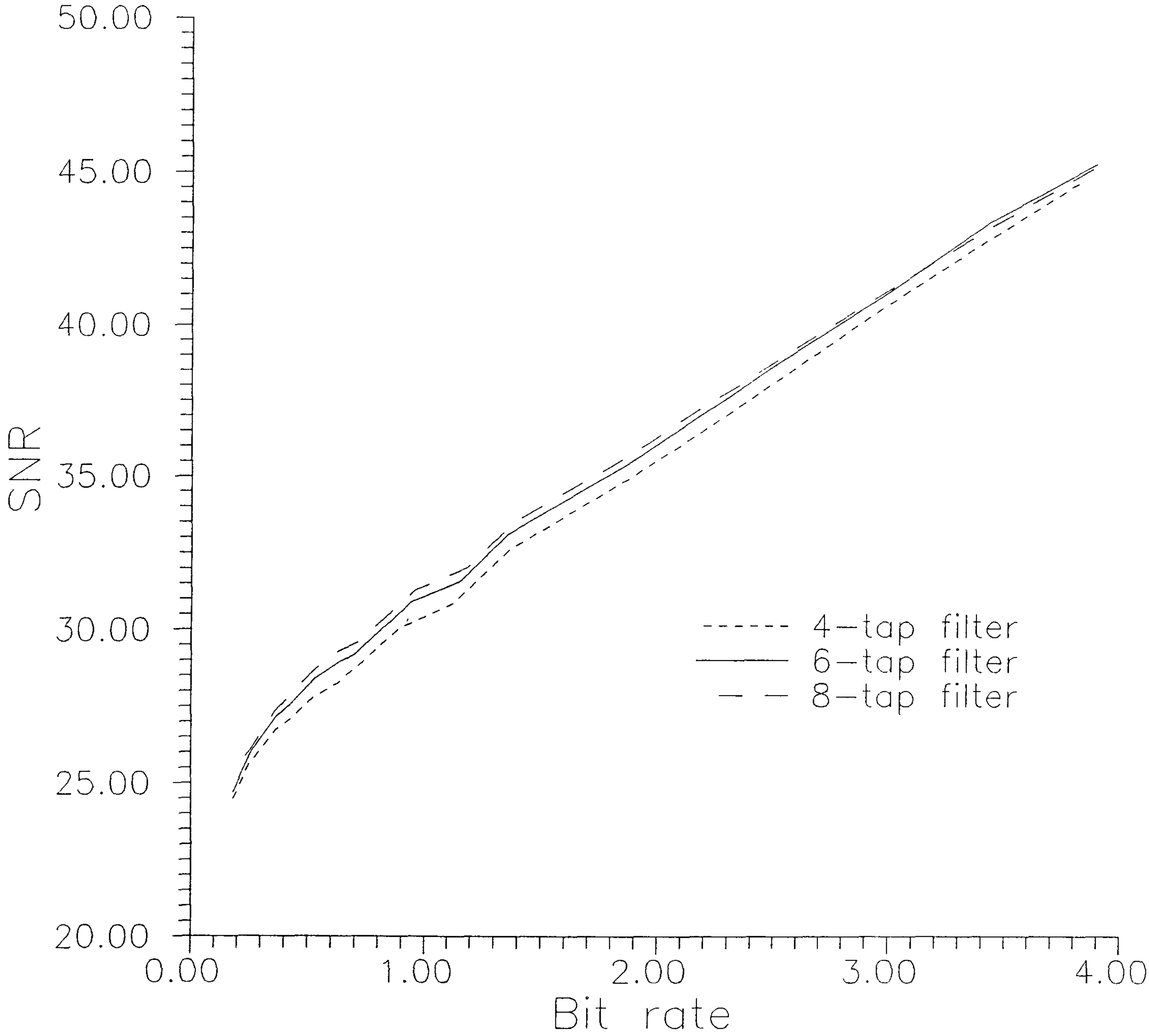

Figure 7.2: SNR versus entropy for 10 subband decomposition using Laplacian quantizers for all the subbands

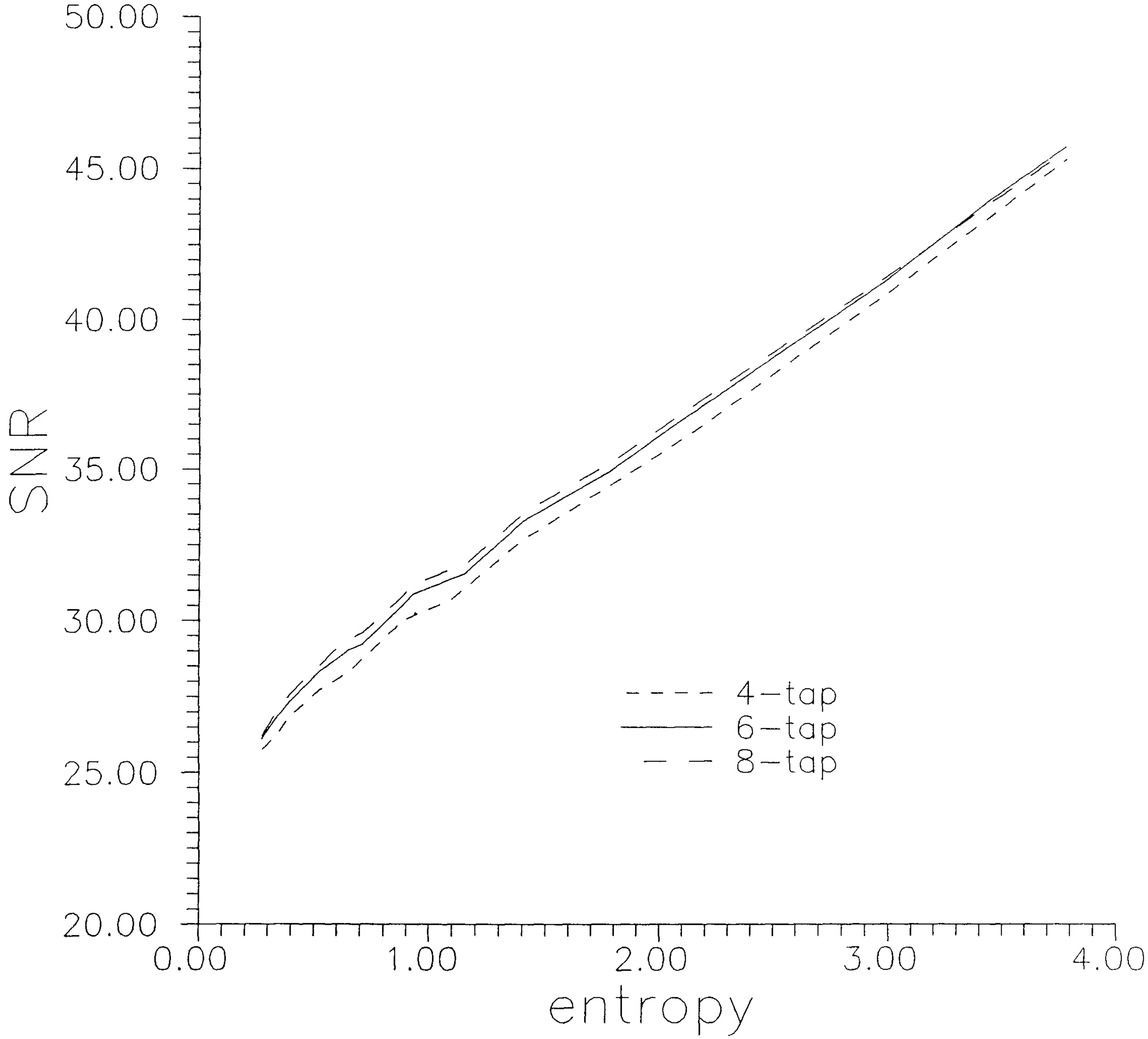

Figure 7.3: SNR versus entropy for 7 subband decomposition using DPCM for the low frequency band and Laplacian quantizers for the remaining subbands

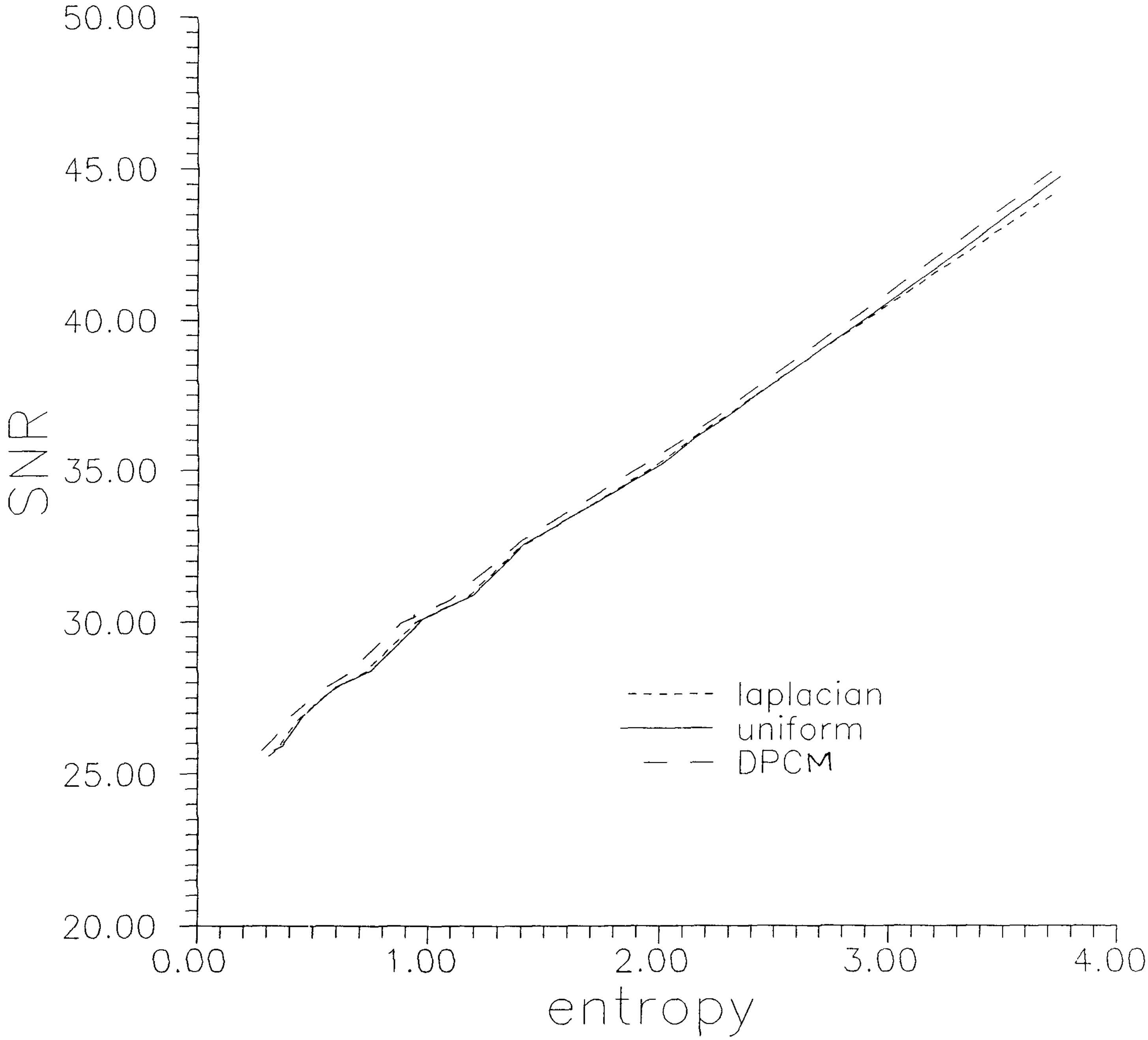

Figure 7.4: SNR versus entropy for 7 subband decomposition using different quantization schemes for the lowest frequency subband

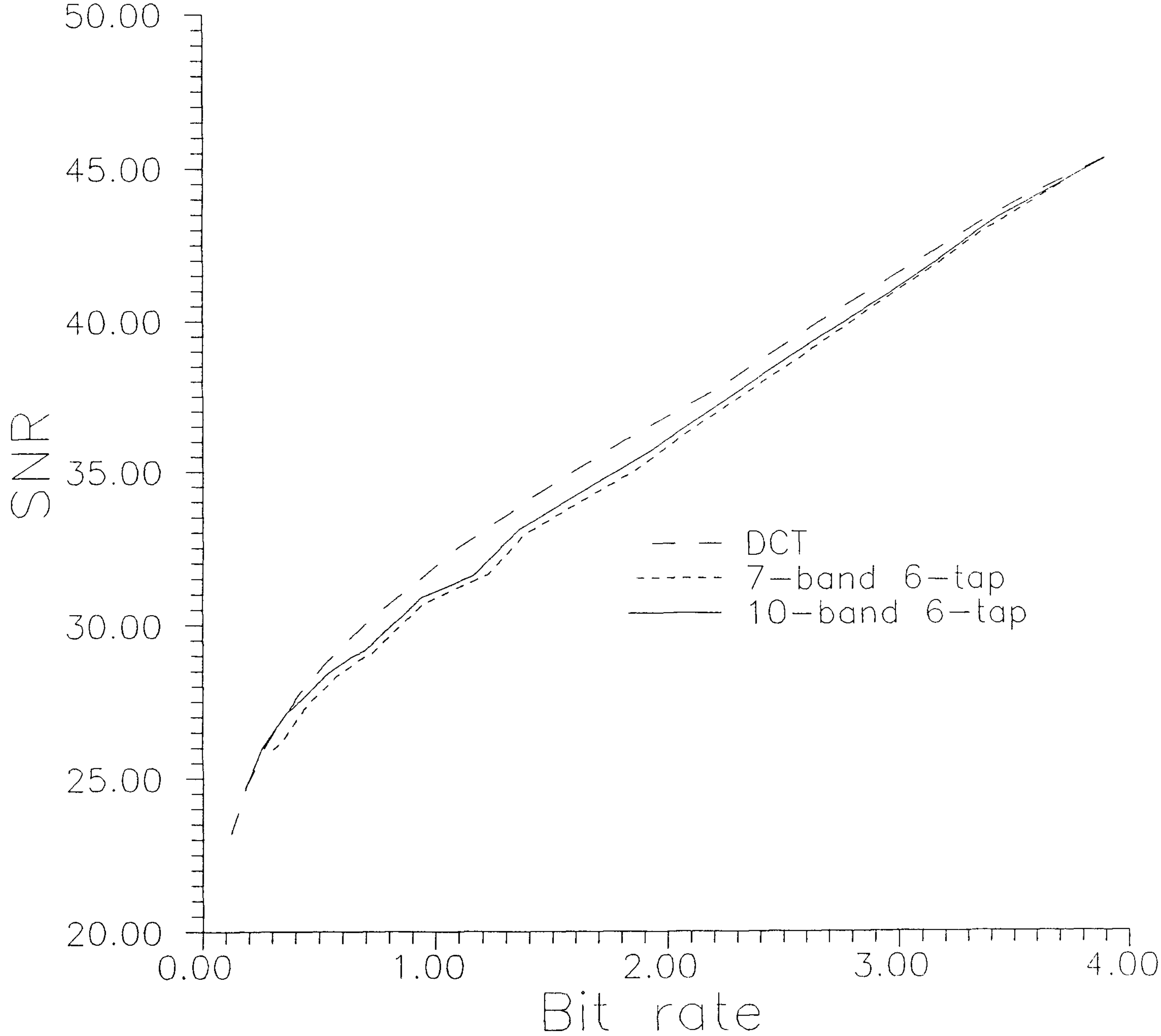

Figure 7.5: SNR versus entropy for DCT, 7 and 10 subband decomposition using Laplacian quantization scheme for the lowest frequency subband

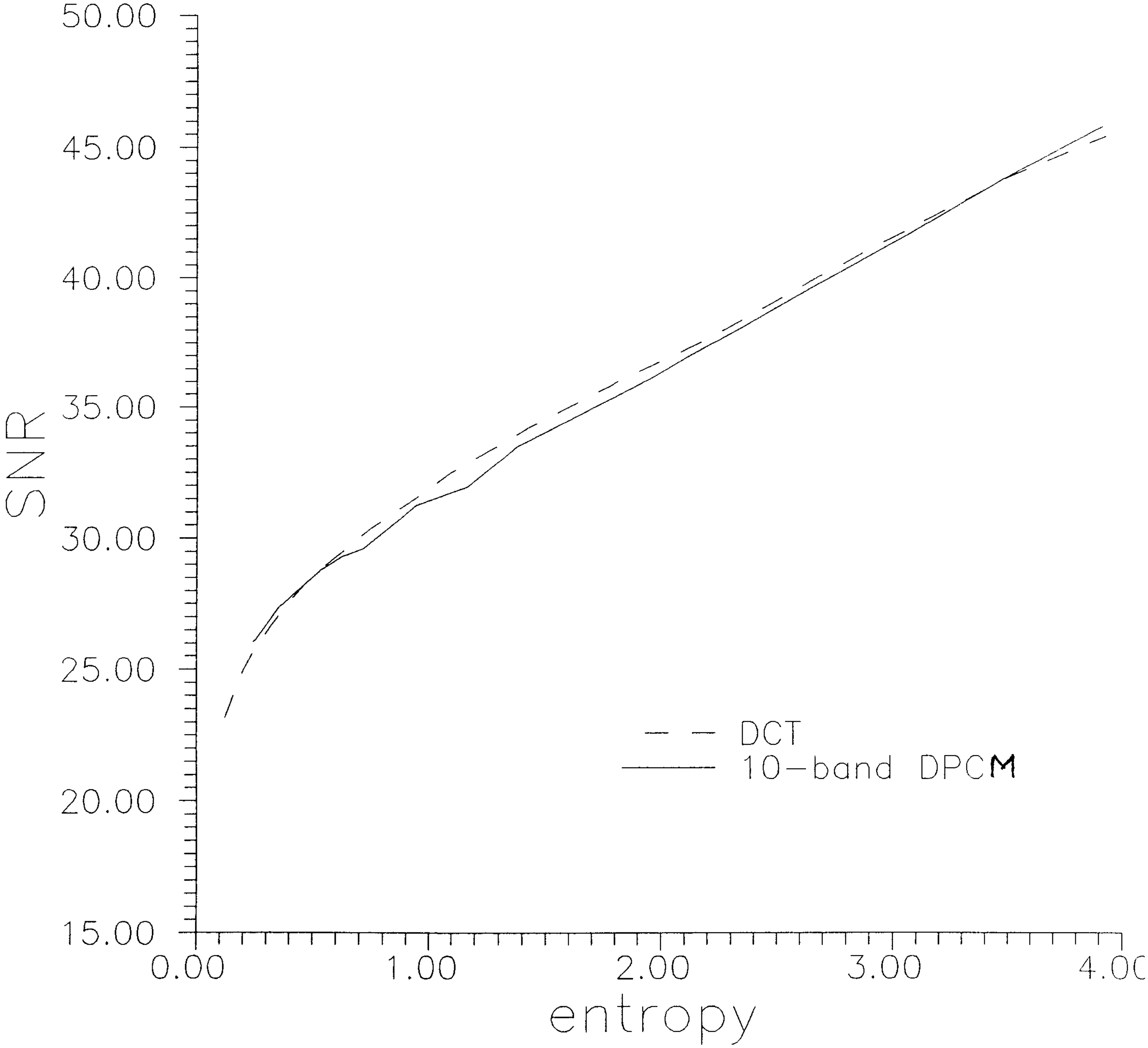

Figure 7.6: SNR versus entropy for DCT and 10 subband decomposition using DPCM quantization scheme for the lowest frequency subband

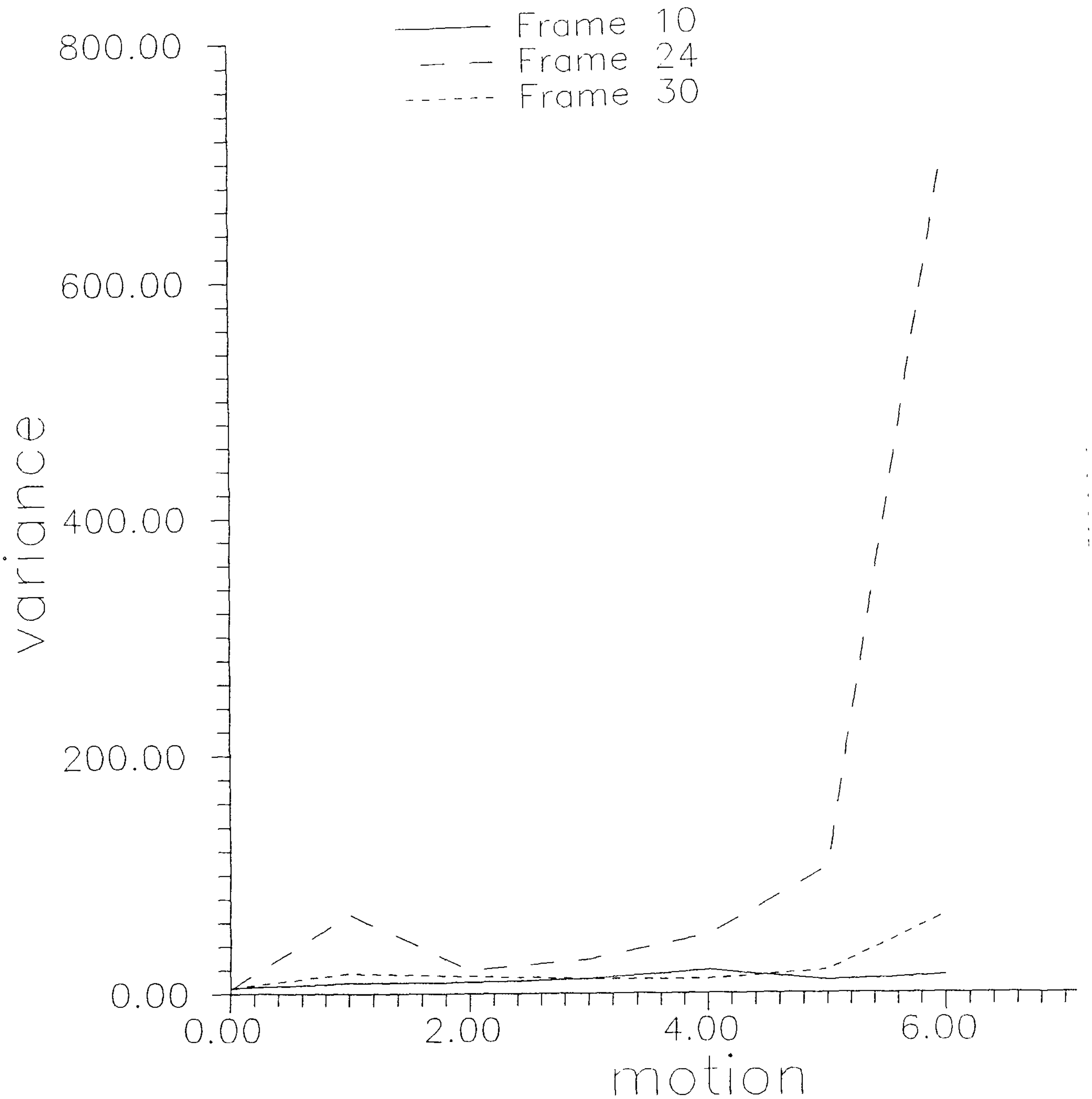

Figure 7.7: Variance vs Motion for MCFD frames 10, 24 and 30 of CINDY

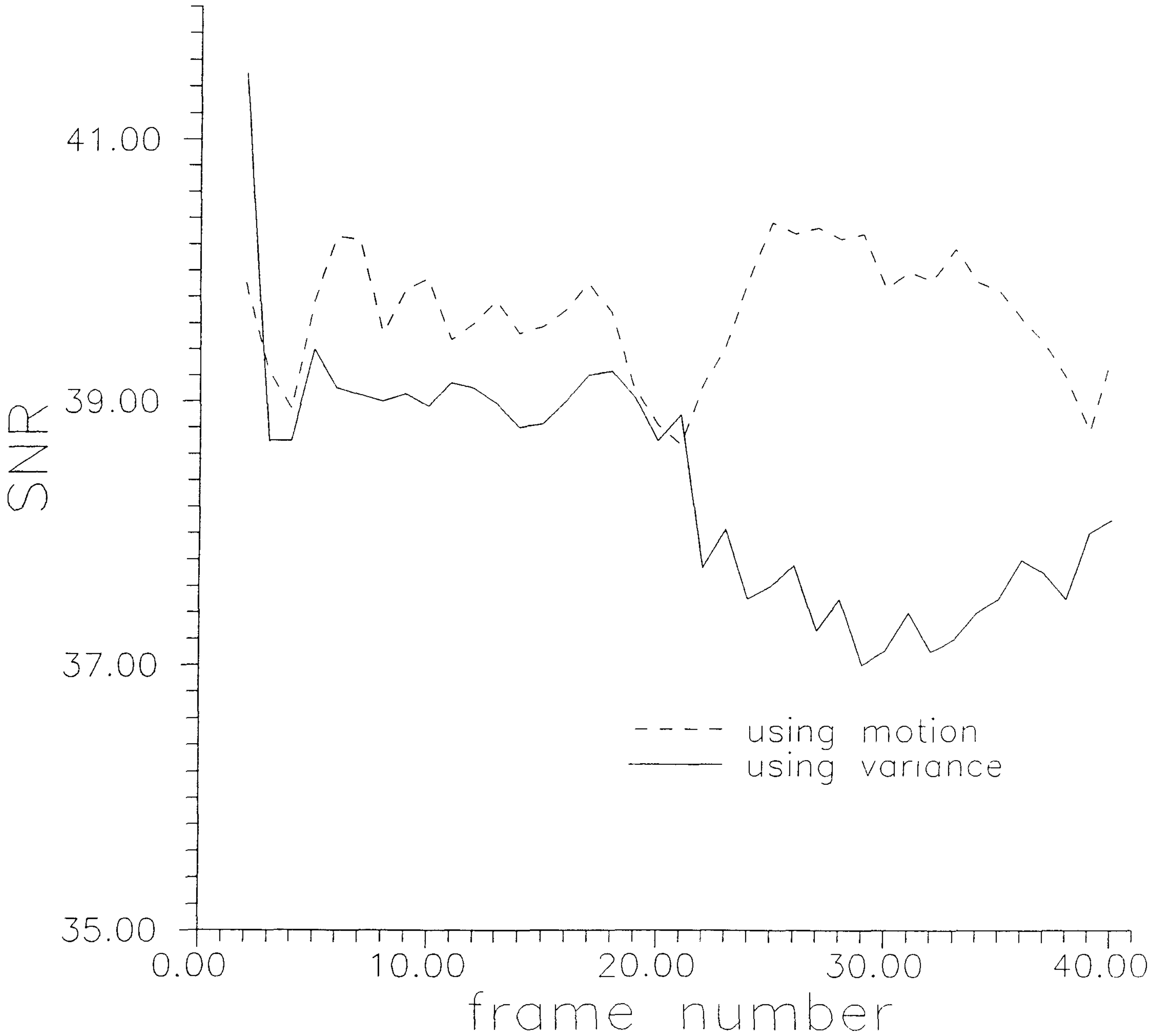

Figure 7.8: SNR vs frame index of MBAVQ and VBAVQ for CINDY

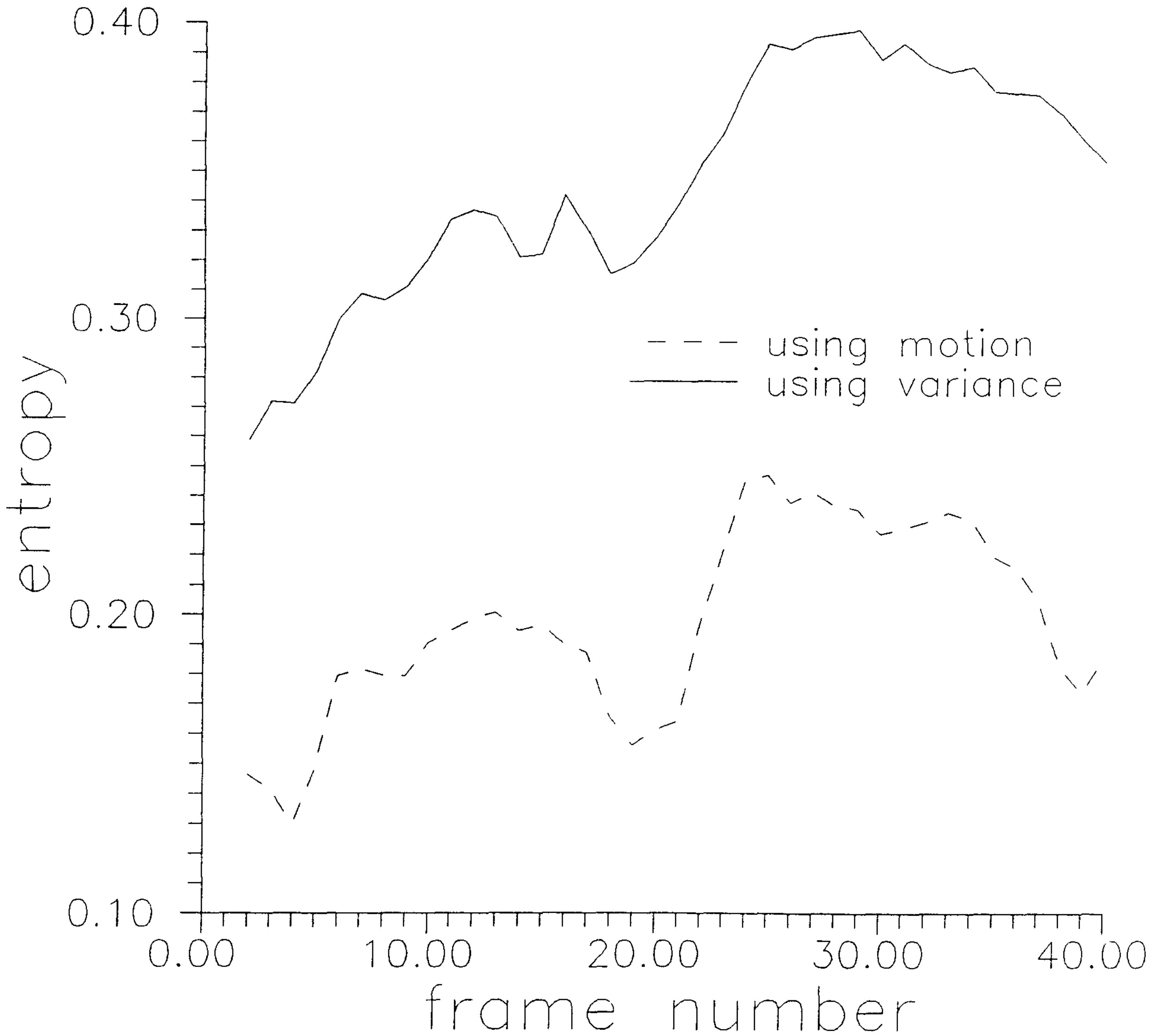

Figure 7.9: Entropy vs frame index of MBAVQ and VBAVQ for CINDY

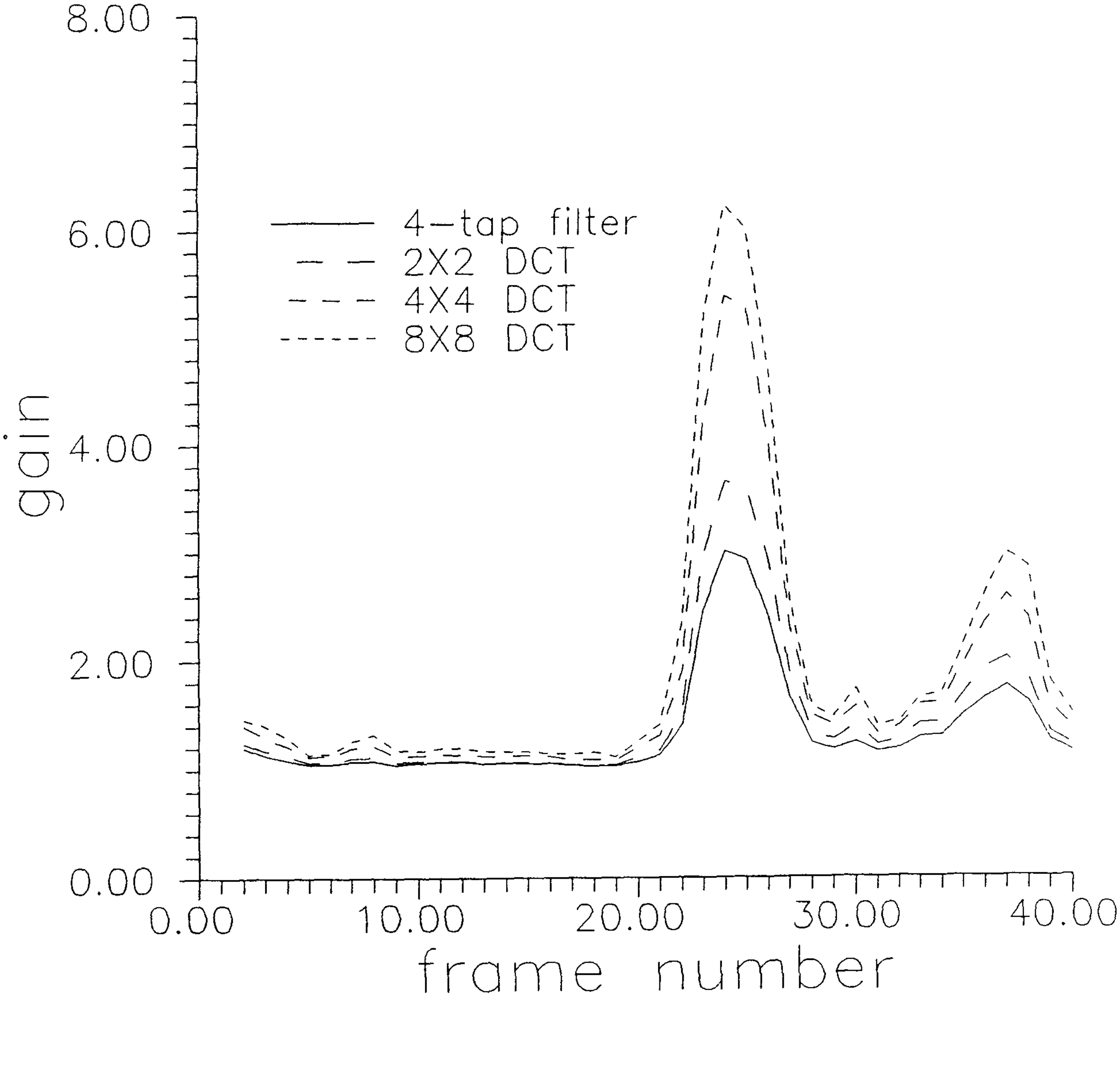

Figure 7.10: Energy Compaction Gain, G_{TC} vs frame index for (8×8) DCT and 4-tap Binomial QMF subband structure for MCFD of CINDY

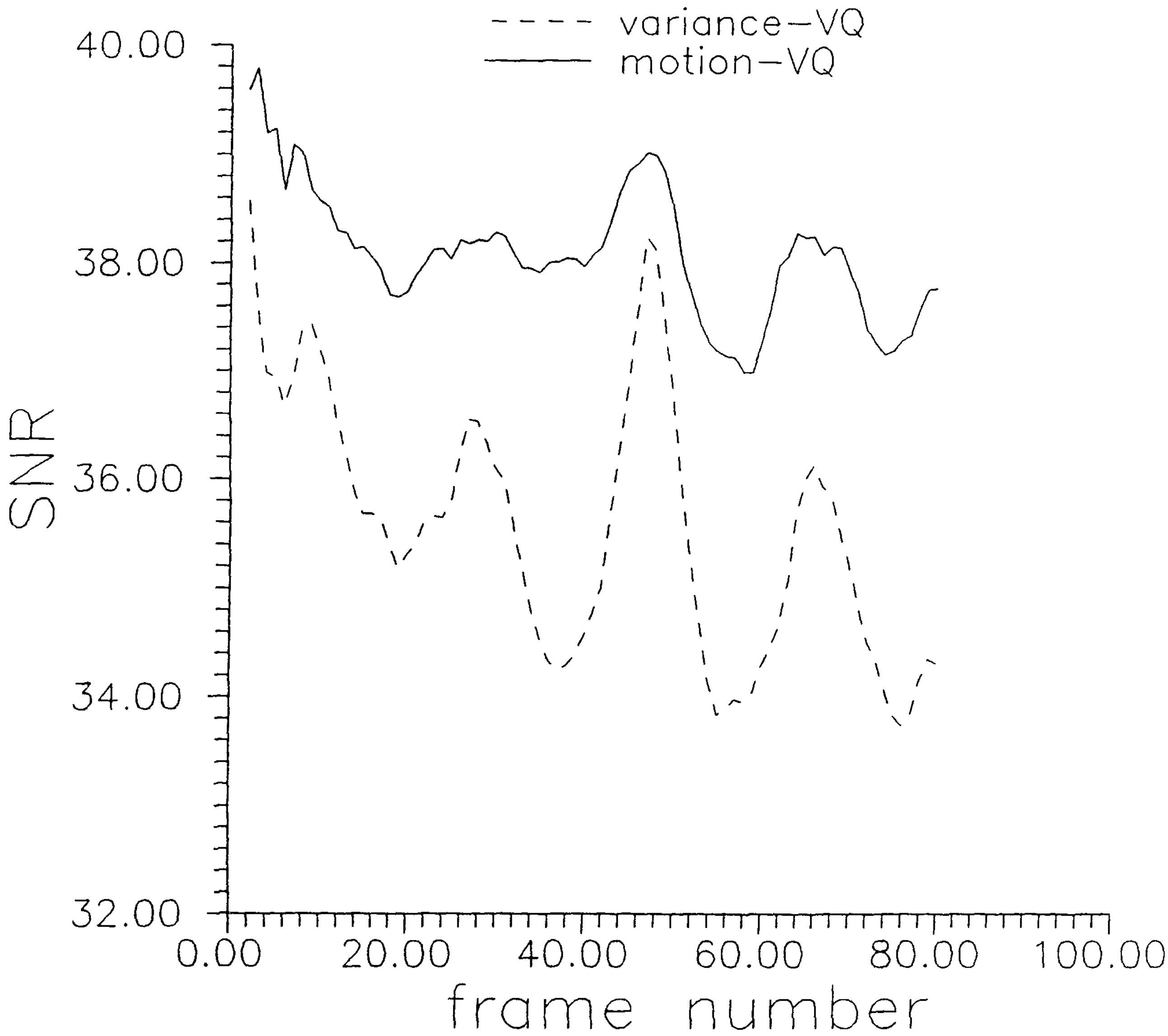

Figure 7.11: SNR vs frame index of MBAVQ and VBAVQ for MONO

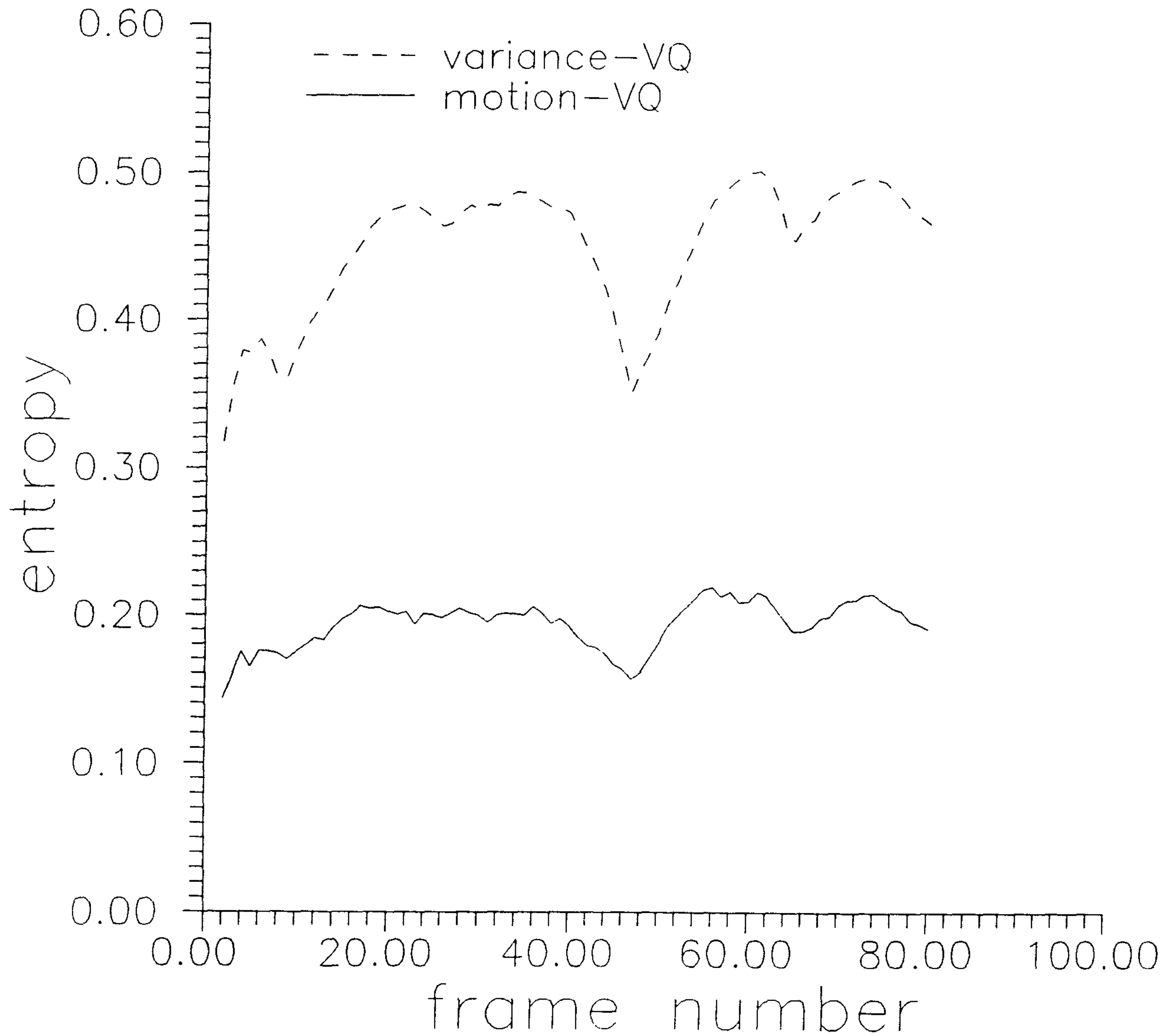

Figure 7.12: Entropy vs frame index of MBAVQ and VBAVQ for MONO

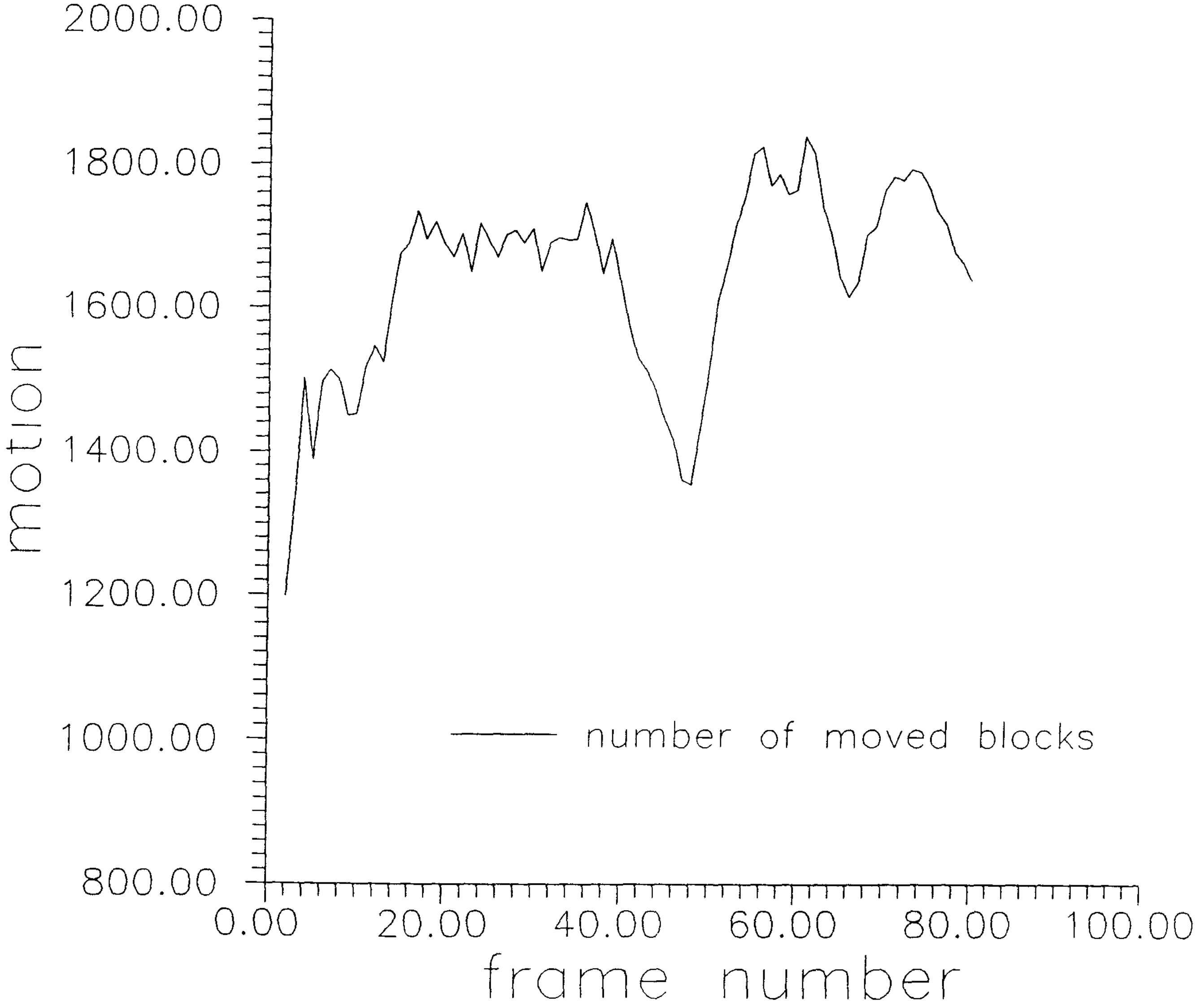

Figure 7.13: Motion vs frame index of MBAVQ and VBAVQ for MONO

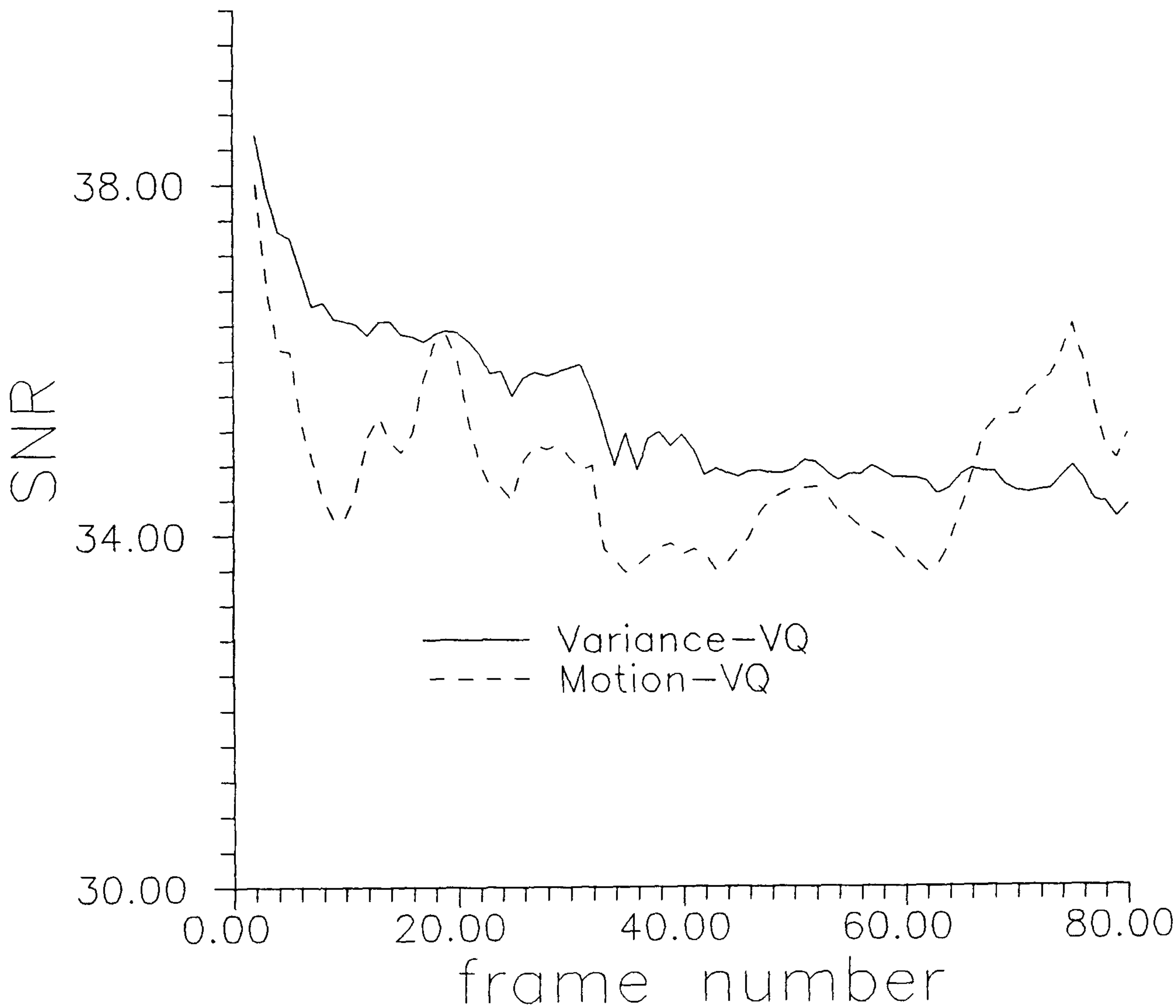

Figure 7.14: SNR vs frame index of MBAVQ and VBAVQ for DUO

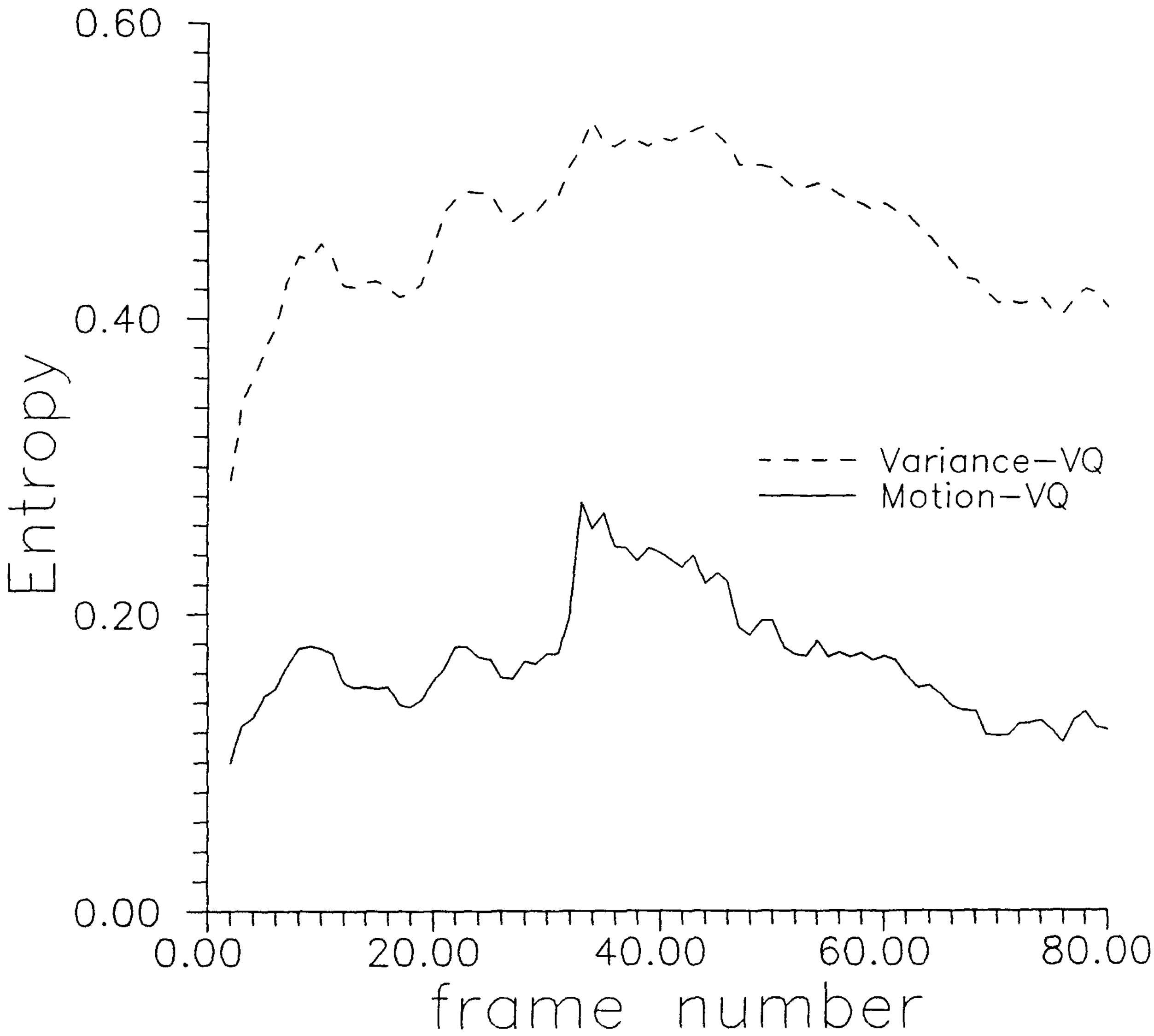

Figure 7.15: Entropy vs frame index of MBAVQ and VBAVQ for DUO

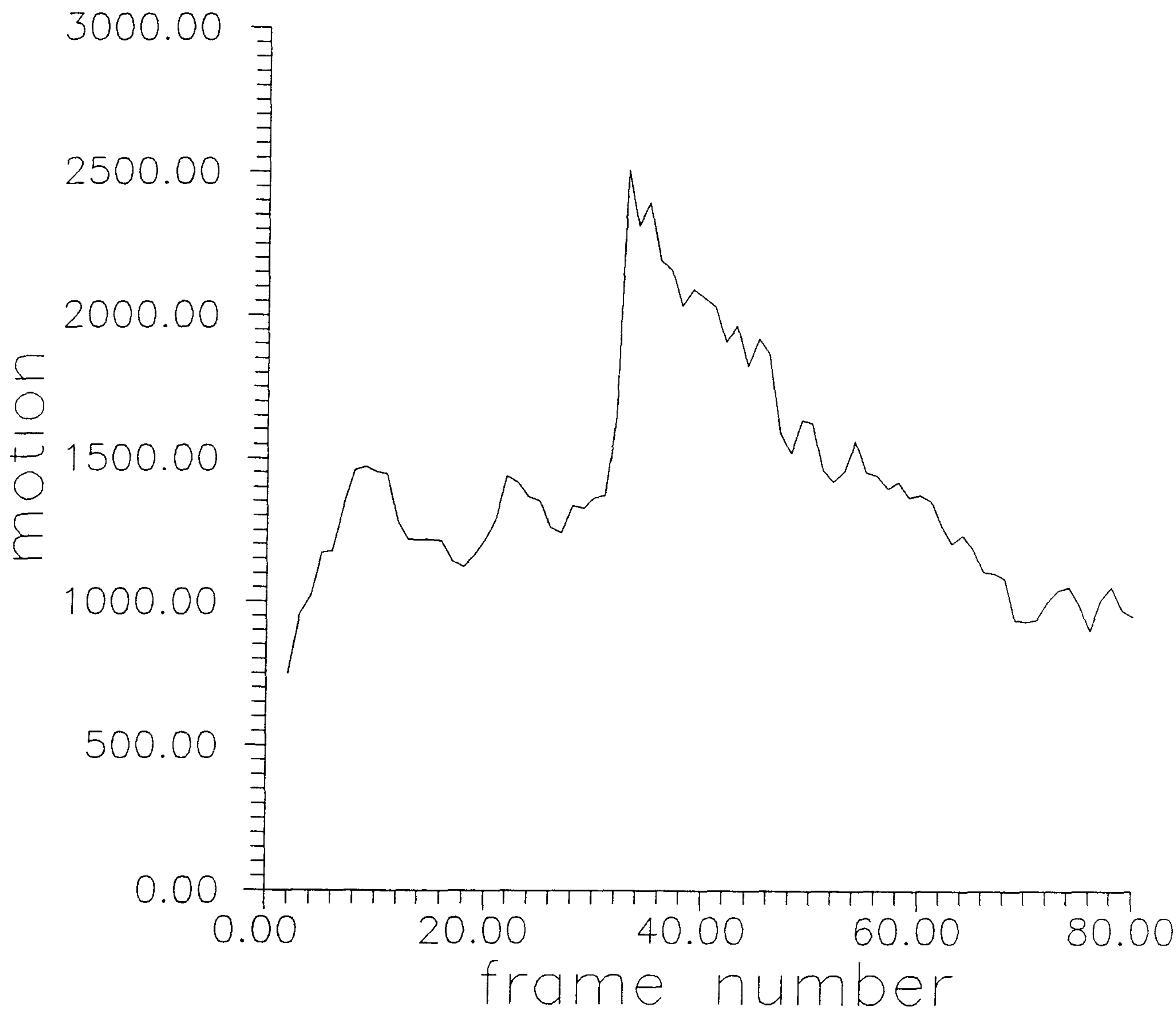

Figure 7.16: Motion vs frame index of MBAVQ and VBAVQ for DUO

	number of levels in the quantizer						
bit rate	band 1	band 2	band 3	band 4	band 5	band 6	band 7
0.5	35	3	7	0	0	0	0
0.75	64	5	11	3	0	0	0
1.0	64	7	15	5	0	3	0

Table 7.1: Results from bit allocation algorithm

Chapter 8

Discussions and Future Research

Subband coding of digital images using Binomial PR-QMF has been presented for still images. Simulation results has shown that the performance of subband coding using a low complexity coder is practically the same as the performance of the industry standard (8×8) DCT based codecs.

For video signals an efficient Motion Based Adaptive Vector Quantization (MBAVQ) subband coding method has been introduced. This new approach is compared with the Variance Based Adaptive Vector Quantization of subband video [19]. The new technique resulted in a reduction in bit rate by nearly (40%) due to the drop of the extra bits used for local variances. Moreover, this method gave superior SNR results especially for high motion frames. This clearly demonstrates that subband coding with MBAVQ should be considered as an atractive and powerful method for video coding. The modelling of MCFD signal and its relation with motion compensation techniques are open problems for future research.

APPENDIX A

```
c     SOURCE CODE FOR THE PROPOSED ADAPTIVE SUBBAND VIDEO CODING WI
c     MOTION COMPENSATION using MBAVQ
c******************************************************************
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 compensati
c    ibs     Block size (8 is used)
c    ip      Assumed maximum displacement (Max of 8)
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
c******************************************************************
      parameter(nx=400,ny=512)
        character*80 input_file
         common /ina/ input_file
         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
         integer motionv(50,64),ifld
         common /motionv/ motionv
         common /ifld/ ifld
         real b1v1(512,16),b1v2(512,16),b1v3(512,16)
         real b2v1(512,16),b2v2(512,16),b2v3(512,16)
         real b3v1(512,16),b3v2(512,16),b3v3(512,16)
         common /vqcodebook/ b1v1,b1v2,b1v3,b2v1,b2v2,b2v3,b3v1,b3v2
      &                      b3v3
         integer mcvector(50,64)
         common /bitrat/ entropy2,entropy
         common /gtotal/ gtotal
c_________________________________________________________________
           real frame1(nx,ny),frame2(nx,ny),pics(416,528),
      +  recon2(nx,ny),frm1msk(8,8),frm2msk(8,8),err1(nx,ny),
      +  errtemp(nx,ny)
         integer ifrm1(nx,ny),ifrm2(nx,ny)
         character*1 pim(nx,ny)
        character*1 pim1(nx,ny)
```

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

c***************************
c       Codebooks are read
c***************************

c_____________________________________________________________

        call readf

c       write(*,*) (coff1(i),i=ltap1,mtap1)

        call readvq
c_____________________________________________________________


c  mfld:  final  field to be read
c  ifld:  starting field number
        mfld = 80
        ifld = 1
        call read_frm(ifld,pim)

c*****************************************
c   Frame One is read into frame1 array
c*****************************************
          do 10 i=1,nx
        do 10 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))
  10    continue

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

        write(35,*) 'Original Image'
c********************************************************
c   The loop to process mfld number of frames begins here
c********************************************************

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

        call read_frm(ifld,pim1)


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

```
        do 11 i=1,nx
      do 11 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))
 11   continue

   call write_in_frm(ifld,frame2)

c       if(ifld.eq.11) then
c        call write_1frm(frame2,'frame11')
c       endif

c**********************************************************
c    Auto-Correlation, Mean, Variance are calculated in
c     the subroutine autocor
c**********************************************************


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 100 i=1,nx+2*ip
       do 100 j=1,ny+2*ip
        pics(i,j)=0.0
 100 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 110 i=1,nx
       do 110 j=1,ny
        pics(i+ip,j+ip)=frame1(i,j)
 110 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

c************************************************************
c    if motion is detected, it is estimated and predicted
c************************************************************
     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+i
     +                                   jact+ip-1+(nn-ip)-1+j3)
 430         continue

               else
              motionv(i,j)=0
              mcvector(i,j)=0
           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+i
     +                                   jact+ip-1+(nn-ip)-1+j3)
 230         continue
     endif

 200 continue
```

```
c*****************************************************
c    MCFD signal is generated using
c*****************************************************

        err=0.0
        do 300 i=1,nx
         do 300 j=1,ny

         err=(err+abs(frame2(i,j)-recon2(i,j)))
c        irecon2(i,j)=recon2(i,j)
      err1(i,j)=frame2(i,j)-recon2(i,j)

 300    continue

c       if(ifld.eq.3) then
c         call write_1frm(err1,'diff3')
c       endif
c_______________test_____________________
c
c       if(ifld.eq.3) then
c         call writeimgs(err1,400,512,'diff3.img')
c       endif
c
c       if(ifld.eq.10) then
c         call writeimgs(err1,400,512,'diff10.img')
c       endif
c_______________________________________________

     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


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

c_____________________________________________________

     call bitrates(mcvector,entropy,50,64)

c*****************************************************
c    the coding of the MCFD signal is carried out here
c*****************************************************

c____________________________________________
```

```
c
c    if(ifld.eq.10) then
c    call writeint1(motionv,50,64,'motion.10')
c    call writeint(err1,400,512,'diff.10')
c    endif
c    if(ifld.eq.24) then
c    call writeint1(motionv,50,64,'motion.24')
c    call writeint(err1,400,512,'diff.24')
c    endif
c
c    if(ifld.eq.30) then
c    call writeint1(motionv,50,64,'motion.30')
c    call writeint(err1,400,512,'diff.30')
c    endif
c_______________________________________________

     call qmf(err1)

            print *,'Error Signal after the vector quantization'
            write(35,*)'Error Signal after the vector quantization'
        vecmean = 0.0
     vecvar = 0.0
        do 351 i = 1,nx
         do 351 j = 1,ny
           vecmean = vecmean + (errtemp(i,j) - err1(i,j))
        vecvar = vecvar + (errtemp(i,j) - err1(i,j))**2
 351    continue
     vecmean = vecmean/(nx*ny)
     vecvar = vecvar/(nx*ny)
     vecvar = vecvar - vecmean**2
     print *, 'Variance of error signal before quant minus after
quant=',
     +  vecvar
        write(35,*) 'Variance of error signal before quant minus af
quant
     +   = ',vecvar

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


     errqnt = 0.0
     errqnt1 = 0.0
        errmean = 0.0
          do 1000 i=1,nx
         do 1000 j=1,ny
            frame1(i,j)=recon2(i,j)+err1(i,j)
        errqnt1 = errqnt1 + (frame2(i,j)-frame1(i,j))**2
```

```
 1000   continue

      call write_out_frm(ifld,frame1)

         xmers=errqnt1/(nx*ny)
         write(35,*) 'MEAN SQUARE ERROR ',xmers
         write(*,*) 'MEAN SQUARE EROR',xmers
         snr=10.0*alog10(255**2/xmers)
         write(35,*) 'S N R = ',snr
         write(36,*) 'S N R = ',snr
         fbitrate = entropy/(64.0) + entropy2/(400*512)
         write(*,*) entropy,entropy2,fbitrate,'final'
         write(38,*) ifld,fbitrate

         write(45,*) ifld,fbitrate,snr,gtotal
         write(*,*) ifld,fbitrate,snr,gtotal

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

      if (ifld .lt. mfld) goto 6000

      stop
      end

c**************************************************
c  subroutine matct
c**************************************************

         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
 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)
c*****************************************************
c   Independent Orthognal Search Technique
c*****************************************************


      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-1,jcent)) .and.
      +     (test(icent,jcent) .lt. test(icent+1,jcent)))  then
              icent=icent
           jcent=jcent
      else if ((test(icent-1,jcent) .lt. test(icent,jcent)) .and.
      +     (test(icent-1,jcent) .lt. test(icent+1,jcent)))  then
              icent=icent-1
           jcent=jcent
      else if ((test(icent+1,jcent) .lt. test(icent,jcent)) .and.
      +     (test(icent+1,jcent) .lt. test(icent-1,jcent)))  then
              icent=icent+1
           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)
c*****************************************************************
c   Subroutine calculates if motion is present in the
c   (ibs*ibs) subblock
c*****************************************************************

        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. 3.0) kount=kount+1
   10 continue
      if (kount .gt. 10) then
           indx=1
      else
           indx=0
      endif
c     print *,'index',indx
      return
      end
```

```
c************************** from here

c THE NEW QMF
c_____________________________________________________________

        subroutine qmf(image)

        integer raw,col

        parameter(raw=400,col=512)

        real image(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 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)

        common /bnds/ llband,lhband
     &  ,hlband,hhband


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

        common /band4/ lband,hband

c input and output images

        real inimg(raw,col),outimg(raw,col)
        common /i/ inimg
        common /o/ outimg



        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)
        common/band2/ lli,lhi,hli,
     &  hhi,llo,lho,hlo,
     &  hho


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

```
     &  ho(raw,col)
        common /bnd1/ li,lo,hi,
     &  ho

        real  limg(raw,col),himg(raw,col)
        common /bnd0/ limg,himg

c motion vector
     integer motionv(50,64)
        common /motionv/ motionv
        common /ifld/ ifld
        common /hist/ hist
        integer hist(9,512)
        common /bitrat/ entropy2,entropy
c_______________________________________________________________

        do 12 i=1,raw
          do 12 j=1,col
          inimg(i,j)=image(i,j)
  12    continue

        call analysis

c
c       if(ifld.eq.2) then
c         call writeint(llband,raw/2,col/2,'band1.2')
c         call writeint(lhband,raw/2,col/2,'band2.2')
c         call writeint(hlband,raw/2,col/2,'band3.2')
c         call writeint1(motionv,50,64,'motionv.2')
c       elseif(ifld.eq.7) then
c         call writeint(llband,raw/2,col/2,'band1.7')
c         call writeint(lhband,raw/2,col/2,'band2.7')
c         call writeint(hlband,raw/2,col/2,'band3.7')
c         call writeint1(motionv,50,64,'motionv.7')
c       elseif(ifld.eq.10) then
c         call writeint(llband,raw/2,col/2,'band1.10')
c         call writeint(lhband,raw/2,col/2,'band2.10')
c         call writeint(hlband,raw/2,col/2,'band3.10')
c         call writeint1(motionv,50,64,'motionv.10')
c       elseif(ifld.eq.15) then
c         call writeint(llband,raw/2,col/2,'band1.15')
c         call writeint(lhband,raw/2,col/2,'band2.15')
c         call writeint(hlband,raw/2,col/2,'band3.15')
c         call writeint1(motionv,50,64,'motionv.15')
c       elseif(ifld.eq.20) then
c         call writeint(llband,raw/2,col/2,'band1.20')
c         call writeint(lhband,raw/2,col/2,'band2.20')
c         call writeint(hlband,raw/2,col/2,'band3.20')
c         call writeint1(motionv,50,64,'motionv.20')
c       elseif(ifld.eq.22) then
c         call writeint(llband,raw/2,col/2,'band1.22')
```

```
c          call writeint(lhband,raw/2,col/2,'band2.22')
c          call writeint(hlband,raw/2,col/2,'band3.22')
c          call writeint1(motionv,50,64,'motionv.22')
c        elseif(ifld.eq.24) then
c          call writeint(llband,raw/2,col/2,'band1.24')
c          call writeint(lhband,raw/2,col/2,'band2.24')
c          call writeint(hlband,raw/2,col/2,'band3.24')
c          call writeint1(motionv,50,64,'motionv.24')
c        elseif(ifld.eq.25) then
c          call writeint(llband,raw/2,col/2,'band1.25')
c          call writeint(lhband,raw/2,col/2,'band2.25')
c          call writeint(hlband,raw/2,col/2,'band3.25')
c          call writeint1(motionv,50,64,'motionv.25')
c        elseif(ifld.eq.27) then
c          call writeint(llband,raw/2,col/2,'band1.27')
c          call writeint(lhband,raw/2,col/2,'band2.27')
c          call writeint(hlband,raw/2,col/2,'band3.27')
c          call writeint1(motionv,50,64,'motionv.27')
c        elseif(ifld.eq.30) then
c          call writeint(llband,raw/2,col/2,'band1.30')
c          call writeint(lhband,raw/2,col/2,'band2.30')
c          call writeint(hlband,raw/2,col/2,'band3.30')
c          call writeint1(motionv,50,64,'motionv.30')
c        elseif(ifld.eq.35) then
c          call writeint(llband,raw/2,col/2,'band1.35')
c          call writeint(lhband,raw/2,col/2,'band2.35')
c          call writeint(hlband,raw/2,col/2,'band3.35')
c          call writeint1(motionv,50,64,'motionv.35')
c        elseif(ifld.eq.38) then
c          call writeint(llband,raw/2,col/2,'band1.38')
c          call writeint(lhband,raw/2,col/2,'band2.38')
c          call writeint(hlband,raw/2,col/2,'band3.38')
c          call writeint1(motionv,50,64,'motionv.38')
c        endif
c

        call segma(llband,raw/2,col/2,x1)

        call segma(lhband,raw/2,col/2,x2)

        call segma(hlband,raw/2,col/2,x3)

        call segma(hhband,raw/2,col/2,x4)

        write(35,*) 'var=',x1,x2,x3,x4,(x1+x2+x3+x4)/4.0

c_____________________________________________________________

        do 191 i=1,9
        do 191 j=1,512
         hist(i,j)=0
```

```
191     continue
c________________________________________________________________

        call vec_quan(llband,0)
        call vec_quan(lhband,1)
        call vec_quan(hlband,2)
c       call vec_quan(hhband)

        call vbitrates(hist,entropy2,9,512)

        do 20 i=1,raw/2
        do 20 j=1,col/2
           hhband(i,j)=0.0
20   continue

        call recon

        call m_error(inimg,outimg,col,raw,output)

        write(*,*) 'error',output
        write(35,*) ' mean error square of the error signal'
        write(35,*) 'between after subband and vector quantization'
        write(35,*) 'mse=',output

        call segma(llband,raw/2,col/2,x1)

        call segma(lhband,raw/2,col/2,x2)

        call segma(hlband,raw/2,col/2,x3)

        call segma(hhband,raw/2,col/2,x4)

        write(35,*) 'var=',x1,x2,x3,x4,(x1+x2+x3+x4)/4.0

        call segma(inimg,raw,col,x5)

        write(35,*) 'inimg var',x5

        call segma(outimg,raw,col,x6)

        write(35,*) 'outimg var=',x6

        do 10 i=1,raw
         do 10 j=1,col
         image(i,j)=outimg(i,j)
 10     continue


c----------------------------
        return
```

```
        end
c_____________________________________________________________________
c     SUBROUTINE READ
C     THIS SUBROUTINE READ THE DATA OF FILTERS COEFFICEINTS
C

        subroutine readf
c        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
        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

  5     format(a80)
        close (11)

        RETURN
        END
c_____________________________________________________________________
c       subroutine initial
c       This subprogram initialize the main program
c
        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
      &  ',/,' filtes there coefficients,
      &  input Image, and output file:')
 4       format( a80)

         return
         end

c_______________________________________________________________________
         subroutine openf
         character*80 input_file
         common/ina/ input_file
         open (11,file='../filters.dir/in24',status='old')
         return
         end


c_______________________________________________________________________
         subroutine analysis

         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)

         common /bnds/ llband,lhband
      &  ,hlband,hhband


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

```
      common /band4/ lband,hband

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

      common /i/ inimg

      common /o/ outimg


      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 cfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)

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

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

      call cfilter(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
c_____________________________________________________________

     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 recon

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

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

      common /i/ inimg

      common /o/ outimg

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)

      common /bnds/ llband,lhband
     & ,hlband,hhband



      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)
         common/band2/ lli,lhi,hli,
     &   hhi,llo,lho,hlo,
     &   hho


         real li(raw,col/2),lo(raw,col),hi(raw,col/2),
     &   ho(raw,col)
         common /bnd1/ li,lo,hi,
     &   ho

         real  limg(raw,col),himg(raw,col)
         common /bnd0/ limg,himg



         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)+hho(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)=(limg(i,j)+himg(i,j))
c           if(outimg(i,j).gt.255)outimg(i,j)=255
c           if(outimg(i,j).lt.0)outimg(i,j)=0
   20    continue

        return
        end


c_______________________________________________________________

       subroutine segma(image,raw,col,output)

       integer raw,col
       real image(raw,col),output
c      double precision s1,s2

       s1=0.0
       s2=0.0

       do 10 i=1,raw
          do 10 j=1,col
           s1=s1+image(i,j)
           s2=s2+(image(i,j))**2
  10   continue

       output=s2/(raw*col)-(s1/(raw*col))**2

       return
       end
c_______________________________________________________________


       subroutine m_error(image1,image2,raw,col,output)

       integer raw,col
       real image1(raw,col),image2(raw,col),output

       s=0.0

       do 10 i=1,raw
          do 10 j=1,col
           s=s+(image1(i,j)-image2(i,j))**2
  10   continue

       output=s/(raw*col)

       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=nx*ny)
      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
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
c_____________________________________________________

      subroutine writeint(pic,nx,ny,name)
      real pic(nx,ny)
      integer ipic(400,512)
      character*20 name

      open(1,file=name)

      do 5 i=1,nx
      do 5 j=1,ny
        ipic(i,j)=int(pic(i,j)+0.5)
 5    continue

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

      close (1)

      return
      end


c_____________________________________________________

      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 vec_quan(pic,nband)
c  pic: picture o be coded (200X256)
c  nband: the band number (0,1,2,3)

       real pic(200,256)
       integer motionv(50,64)
```

```
      real tvector(16)

      common /hist/ hist
      integer hist(9,512)

        common /motionv/ motionv
      real b1v1(512,16),b1v2(512,16),b1v3(512,16)
      real b2v1(512,16),b2v2(512,16),b2v3(512,16)
      real b3v1(512,16),b3v2(512,16),b3v3(512,16)
      common /vqcodebook/ b1v1,b1v2,b1v3,b2v1,b2v2,b2v3,b3v1,b3v2
     &                      b3v3

      nn=nband*3

      do 10 i=1,50
      do 10 j=1,64
        if(motionv(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(motionv(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(motionv(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 vquantizer(tvector,b1v1,ivecnum)
          hist(1,ivecnum) = hist(1,ivecnum)+1

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

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

```
          call vquantizer(tvector,b1v3,ivecnum)
          hist(3,ivecnum) = hist(3,ivecnum)+1

        else if(mm.eq.4)then
          call vquantizer(tvector,b2v3,ivecnum)
          hist(4,ivecnum) = hist(4,ivecnum)+1

        else if(mm.eq.5)then
          call vquantizer(tvector,b2v2,ivecnum)
          hist(5,ivecnum) = hist(5,ivecnum)+1

        else if(mm.eq.6)then
          call vquantizer(tvector,b2v3,ivecnum)
          hist(6,ivecnum) = hist(6,ivecnum)+1

        else if(mm.eq.7)then
          call vquantizer(tvector,b3v1,ivecnum)
          hist(7,ivecnum) = hist(7,ivecnum)+1

        else if(mm.eq.8)then
          call vquantizer(tvector,b3v2,ivecnum)
          hist(8,ivecnum) = hist(8,ivecnum)+1

        else if(mm.eq.9)then
          call vquantizer(tvector,b3v3,ivecnum)
          hist(9,ivecnum) = hist(9,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

c____________________________________________________________

      subroutine vquantizer(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

c___________________________________________________

     subroutine readvq

        real b1v1(512,16),b1v2(512,16),b1v3(512,16)
        real b2v1(512,16),b2v2(512,16),b2v3(512,16)
        real b3v1(512,16),b3v2(512,16),b3v3(512,16)
        common /vqcodebook/ b1v1,b1v2,b1v3,b2v1,b2v2,b2v3,b3v1,b3v2
     &                      b3v3

     open (15,file='../quantizer.dir/b1v1.12')
        do 10 i=1,512
          read(15,*) (b1v1(i,j),j=1,16)
10   continue
        close(15)

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

     open (17,file='../quantizer.dir/b1v3.56')
        do 12 i=1,512
          read(17,*) (b1v3(i,j),j=1,16)
12   continue
        close(17)


     open (18,file='../quantizer.dir/b2v1.12')
        do 13 i=1,512
          read(18,*) (b2v1(i,j),j=1,16)
13   continue
```

```
      close(18)

     open (19,file='../quantizer.dir/b2v2.34')
        do 14 i=1,512
          read(19,*) (b2v2(i,j),j=1,16)
14   continue
        close(19)


     open (20,file='../quantizer.dir/b2v3.56')
        do 15 i=1,512
          read(20,*) (b2v3(i,j),j=1,16)
15   continue
        close(20)

     open (21,file='../quantizer.dir/b3v1.12')
        do 16 i=1,512
          read(21,*) (b3v1(i,j),j=1,16)
16   continue
        close(21)


     open (22,file='../quantizer.dir/b3v2.34')
        do 17 i=1,512
          read(22,*) (b3v2(i,j),j=1,16)
17   continue
        close(22)

     open (23,file='../quantizer.dir/b3v3.56')
        do 18 i=1,512
          read(23,*) (b3v3(i,j),j=1,16)
18   continue
        close(23)

        return
     end
c_____________________________________________________________________

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

c      open(1,file='ali.100',access='direct',form=
c    &  'unformatted',recl=512*512)
c
c      open(1,file='/images/cindy40',access='direct',form=
c    &  'unformatted',recl=512)

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

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
c_______________________________________________________________

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

         open(22,file='mono.in80',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
c_______________________________________________________________

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

         open(21,file='mono.out80',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(100)
 20     continue

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

     close(21)

     return
     end
c_______________________________________________________________

     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

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

     close(22)

     return
     end

c_______________________________________________________________

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

```
        subroutine vbitrates(ic,bitrate,raw,col)

        common /gtotal/ gtotal
        integer ic(raw,col),raw,col
        real entropy(9),sum(512),pr(512)

        gtotal=0
        bitrate=0
     do 10 m=1,9

        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

        gtotal=gtotal/3
        write(*,*) 'gtotal = ',gtotal
        write(*,*) 'ventropy = ',(entropy(i),i=1,9)
        write(*,*) 'bitrate = ', bitrate

        return
        end

c_____________________________________________________________

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

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

         return
         end

c_______________________________________________________________

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

```
c
c     Simulation program for 10 band decomposition
c  Name: Hosam Mutlag
c      MAIN
c*****************************************************************

        character*80 input_file
        common /ina/ input_file
        character*80 inputimg
        common /image/ inputimg
        common /bitrate/ rav

        open(50,file='10bdpcm.res',form='formatted')

        do 20 rav=.2,1.3,0.1

c       rav=1.0
        input_file='in24'
        inputimg='ladyp.img'

        call subband10(e1,or1,snr1)

        input_file='in6'
        call subband10(e2,or2,snr2)

        input_file='in8'
        call subband10(e3,or3,snr3)

        write(50,10) rav,snr1,snr2,snr3,e1,or1,e2,or2,e3,or3

 20     continue

        do 30 rav=1.5,5.0,0.5

        input_file='in24'
        inputimg='ladyp.img'

        call subband10(e1,or1,snr1)

        input_file='in6'
        call subband10(e2,or2,snr2)

        input_file='in8'
        call subband10(e3,or3,snr3)

        write(50,10) rav,snr1,snr2,snr3,e1,or1,e2,or2,e3,or3

 30     continue

 10     format(f4.2,3f8.4,6f8.5)
        close(50)
```

```
      stop
      end

c_______________________________________________________
c  This is a 10 band simulation program.
c  it uses dpcm  quantizer for the first band
c  and  laplacian quanizer for the rest. hoffman coading.
c  the program does bit allocation for the 10 bands.
c___________________________________________________
c
     subroutine subband10(xo1,xo2,snr)

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

        real dimg(32,32),cimg(32,32)
        real p(75),p1(75)

        character*80 input_file
        common /ina/ input_file
        character*80 inputimg
        common /image/ inputimg
        common /bitrate/ rav
        common /header/ header

        character*1 header(64)

        real x(10),output,xmean(10),entrpy(10),entrpy1(4),entrpy2(3
        real entrpy3(3)
        real xt(64),xmeant(64)

        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

        integer iq(64),iqt(64)
        real pr(-64:64),prb(128)
        integer prnum(128)
        character*80 codbook(128)
        character*1 hc(77000)
c__________________________________________________________

        real inimg(256,256)

        real a1(128,128),a2(128,128),a3(128,128),a4(128,128)

        real b1(64,64),b2(64,64),b3(64,64),b4(64,64)

        real c1(32,32),c2(32,32),c3(32,32),c4(32,32)
```

```
      real e1(128,128)
      real d1(64,64)

      real qc1(32,32),qc2(32,32),qc3(32,32),qc4(32,32)
      real qb2(64,64),qb3(64,64),qb4(64,64)
      real qa2(128,128),qa3(128,128),qa4(128,128)

      integer ic1(4,32,32),icr1(4,32,32)
      integer ic2(3,64,64),icr2(3,64,64)
      integer ic3(3,128,128),icr3(3,128,128)

      real outimg(256,256)
c_____________________________________________________________

c     call initial

      call readf

      call readimage(inimg,raw,col)

      write(*,*) 'subband analysis'

      call analysis256(inimg,a1,a2,a3,a4)

      call analysis128(a1,b1,b2,b3,b4)

      call analysis64(b1,c1,c2,c3,c4)

c_______________________________________________________

       write(*,*) 'calculating the variances and means'

      call segma(c1,32,32,x(1),xmean(1))
      call segma(c2,32,32,x(2),xmean(2))
      call segma(c3,32,32,x(3),xmean(3))
      call segma(c4,32,32,x(4),xmean(4))
      call segma(b2,64,64,x(5),xmean(5))
      call segma(b3,64,64,x(6),xmean(6))
      call segma(b4,64,64,x(7),xmean(7))
      call segma(a2,128,128,x(8),xmean(8))
      call segma(a3,128,128,x(9),xmean(9))
      call segma(a4,128,128,x(10),xmean(10))
c
      call dpcm1(c1,dimg,32,32)
      call segma(dimg,32,32,var2,rmean)
      write(*,*) x(1),xmean(1)
      x(1)=var2
      xmean(1)=rmean
      write(*,*) var2,rmean
c
c_____________________________________________________________
```

```
c convert the 10band to 64 band for bit allocation

        do 10 i=1,4
          xt(i)=x(i)
          xmeant(i)=xmean(i)
 10   continue

      do 15 j=1,3
         do 20 i=(j*4)+1,j*4+4
           xt(i)=x(j+4)
           xmeant(i)=xmean(j+4)
 20   continue
 15      continue

      do 11 j=1,3
         do 21 i=(j*16)+1,j*16+16
           xt(i)=x(j+7)
           xmeant(i)=xmean(j+7)
 21   continue
 11      continue

c        write(*,*) (x(i),i=1,10)
c        write(*,*) (xt(i),i=1,64)
c
         call entropy(iqt,xt,xmeant)
c

         iq(1)=iqt(1)
         iq(2)=iqt(2)
         iq(3)=iqt(3)
         iq(4)=iqt(4)
         iq(5)=iqt(5)
         iq(6)=iqt(9)
         iq(7)=iqt(13)
         iq(8)=iqt(17)
         iq(9)=iqt(33)
         iq(10)=iqt(49)

      write(*,*) (iq(i),i=1,10)
c        write(*,*) (iqt(i),i=1,64)
c_______________________________________________________________

      write(*,*) 'quantizing'

         nm=iq(1)
         call readq(nm,p,pl)
         if(nm.ne.iq(1))then
           write(*,*) 'error in reading quantizer data'
           stop
         end if
```

```
      call qdpcm(c1,ic1,nm,var2,rmean,32,32,p,pl)

c     call unifquan(c1,1,ic1,iq(1),xminx,step,4,32,32)
c     call quantizer(c1,1,ic1,iq(1),xmean(1),x(1),4,32,32)
      call quantizer(c2,2,ic1,iq(2),xmean(2),x(2),4,32,32)
      call quantizer(c3,3,ic1,iq(3),xmean(3),x(3),4,32,32)
      call quantizer(c4,4,ic1,iq(4),xmean(4),x(4),4,32,32)

      call quantizer(b2,1,ic2,iq(5),xmean(5),x(5),3,64,64)
      call quantizer(b3,2,ic2,iq(6),xmean(6),x(6),3,64,64)
      call quantizer(b4,3,ic2,iq(7),xmean(7),x(7),3,64,64)

      call quantizer(a2,1,ic3,iq(8),xmean(8),x(8),3,128,128)
      call quantizer(a3,2,ic3,iq(9),xmean(9),x(9),3,128,128)
      call quantizer(a4,3,ic3,iq(10),xmean(10),x(10),3,128,128)

c_____________________________________________________________________

      sum=0
      length=0

      do 25 i=1,4
       if(iq(i).gt.0.1) then

       call bitrates(ic1,i,entrpy1,pr,4,32,32)
c
       call convert(pr,prb,prnum,ns)
c
       call sort(prb,prnum,ns)

       call hcbook(prb,codbook,ns)

       call codehc(ic1,i,codbook,prnum,ns,hc,l1,4,32,32)
c
       call decodehc(icr1,i,codbook,prnum,ns,hc,l1,4,32,32)

       do 103 i1=1,32
       do 103 j1=1,32
         icr=ic1(i,i1,j1)-icr1(i,i1,j1)
c      write(*,*) icr
       if(icr.ne.0) then
         write(*,*) 'ERROR2 IN CHANNEL CODING',i,i1,j1,icr
         stop
       endif

 103   continue

       length=length+l1
       entrpy(i)=entrpy1(i)
       sum=sum+entrpy1(i)
```

```fortran
        endif
 25   continue

        sum=sum/4.0

c_______________________________________________________________________

        do 35 i=1,3
         if(iq(i+4).gt.0.1) then

         call bitrates(ic2,i,entrpy2,pr,3,64,64)
         call convert(pr,prb,prnum,ns)
         call sort(prb,prnum,ns)
         call hcbook(prb,codbook,ns)
         call codehc(ic2,i,codbook,prnum,ns,hc,l1,3,64,64)
c
         call decodehc(icr2,i,codbook,prnum,ns,hc,l1,3,64,64)
c
         do 104 i1=1,64
         do 104 j1=1,64
           icr=ic2(i,i1,j1)-icr2(i,i1,j1)
c        write(*,*) icr
         if(icr.ne.0) then
           write(*,*) 'ERROR2 IN CHANNEL CODING',i,i1,j1,icr
           stop
         endif

 104     continue

         length=length+l1
         entrpy(i+4)=entrpy2(i)
         sum=sum+entrpy2(i)

         endif
 35   continue

        sum=sum/4.0

c_______________________________________________________________________

        do 45 i=1,3
         if(iq(i+7).gt.0.1) then

         call bitrates(ic3,i,entrpy3,pr,3,128,128)
         call convert(pr,prb,prnum,ns)
         call sort(prb,prnum,ns)
         call hcbook(prb,codbook,ns)
         call codehc(ic3,i,codbook,prnum,ns,hc,l1,3,128,128)
c
         call decodehc(icr3,i,codbook,prnum,ns,hc,l1,3,128,128)
c
```

```
        do 105 i1=1,128
        do 105 j1=1,128
          icr=ic3(i,i1,j1)-icr3(i,i1,j1)
c       write(*,*) icr
        if(icr.ne.0) then
          write(*,*) 'ERROR2 IN CHANNEL CODING',i,i1,j1,icr
          stop
        endif

 105    continue

        length=length+l1

         entrpy(i+7)=entrpy3(i)
         sum=sum+entrpy3(i)
        endif
 45   continue

       write(*,*) 'entropy=',sum/4.0
       xo1=sum/4.0
       xo2=real(length)/(256.0**2)

c________________________________________________________________

      write(*,*) 'DEquantizing'

       call dequantizer(cimg,1,icr1,iq(1),rmean,var2,4,32,32)
       call dpcdecoder(cimg,qc1,32,32)
       call m_error(qc1,c1,32,32,cerror)
       write(*,*) 'c1 error',cerror

c      call unifdequan(qc1,1,icr1,iq(1),xminx,step,4,32,32)
c      call dequantizer(qc1,1,icr1,iq(1),xmean(1),x(1),4,32,32)
       call dequantizer(qc2,2,icr1,iq(2),xmean(2),x(2),4,32,32)
       call dequantizer(qc3,3,icr1,iq(3),xmean(3),x(3),4,32,32)
       call dequantizer(qc4,4,icr1,iq(4),xmean(4),x(4),4,32,32)

       call dequantizer(qb2,1,icr2,iq(5),xmean(5),x(5),3,64,64)
       call dequantizer(qb3,2,icr2,iq(6),xmean(6),x(6),3,64,64)
       call dequantizer(qb4,3,icr2,iq(7),xmean(7),x(7),3,64,64)

       call dequantizer(qa2,1,icr3,iq(8),xmean(8),x(8),3,128,128)
       call dequantizer(qa3,2,icr3,iq(9),xmean(9),x(9),3,128,128)
       call dequantizer(qa4,3,icr3,iq(10),xmean(10),x(10),3,128,12

c________________________________________________________________

      write(*,*) 'synthesis'

       call synthesis64(qc1,qc2,qc3,qc4,d1)
       call synthesis128(d1,qb2,qb3,qb4,e1)
```

```
      call synthesis256(e1,qa2,qa3,qa4,outimg)
c____________________________________________________________

      do 131 i=1,256
       do 131 j=1,256
        if(outimg(i,j).gt.255) then
        outimg(i,j) =255
        else if(outimg(i,j).lt.0) then
        outimg(i,j)=0
        endif
 131  continue


      call m_error(inimg,outimg,col,raw,output)

c     call writeimg(outimg,256,256,'out.img')

c     write(*,*) 'error',output
      snr=10*alog10(255**2/output)
      write(*,*) 'SNR =',snr

c________________________________________________________

      return
      end

c__________________________________________________________
c    SUBROUTINE READ
C    THIS SUBROUTINE READ THE DATA OF FILTERS COEFFICEINTS
C

      subroutine readf
c      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
c     write(*,*) 'reading filter coefficeints'
      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
c___________________________________________________________
c       subroutine initial
c       This subprogram initialize the main program
c
        subroutine initial
        character*80 input_file,inputimg
        real rav
        common /ina/ input_file
        common /image/ inputimg
        common /bitrate/ rav

c       write(*,*) 'initialization'
        write (*,1)
     write (*,3)
        read (5,4) input_file
        write(*,*) ' Enter the name of the input image'
        read(*,*) inputimg
        write(*,*) ' Enter the bit rate'
        read(*,*) rav


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

        return
        end


c___________________________________________________________
        subroutine openf
```

```
      character*80 input_file
      common/ina/ input_file
      open (11,file='../filters.dir/'//input_file,status='old')
      return
      end

c______________________________________________________________________

      subroutine analysis256(inimg,llband,lhband,hlband,hhband)

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

      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 cfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)

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

```
        call cfilter(coff1,hband,hlband,nx,ny,ltap1,mtap1)

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

        return
        end
c_____________________________________________________________________


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

        integer raw,col
        parameter(raw=128,col=128)

        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 cfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)

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

```
      call cfilter(coff1,hband,hlband,nx,ny,ltap1,mtap1)

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

      return
      end
c_____________________________________________________________________


      subroutine analysis64(inimg,llband,lhband,hlband,hhband)

      integer raw,col
      parameter(raw=64,col=64)

      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 cfilter(coff1,lband,llband,nx,ny,ltap1,mtap1)

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

      call cfilter(coff1,hband,hlband,nx,ny,ltap1,mtap1)
```

```
      call cfilter(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
c____________________________________________________________

     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)

      integer raw,col
      parameter(raw=256,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)+hho(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))
c           if(outimg(i,j).gt.255)outimg(i,j)=255
c           if(outimg(i,j).lt.0)outimg(i,j)=0
   20    continue

        return
        end


c________________________________________________________

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

        integer raw,col
        parameter(raw=128,col=128)

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)+hho(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))
c         if(outimg(i,j).gt.255)outimg(i,j)=255
c         if(outimg(i,j).lt.0)outimg(i,j)=0
   20  continue

      return
      end


c_______________________________________________________________
c________________________________________________________
```

```
      subroutine synthesis64(llband,lhband,hlband,hhband,outimg)

      integer raw,col
      parameter(raw=64,col=64)

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)+hho(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))
c           if(outimg(i,j).gt.255)outimg(i,j)=255
c           if(outimg(i,j).lt.0)outimg(i,j)=0
   20    continue

        return
        end


c_______________________________________________________________


      subroutine readimage(pic,nx,ny)
      real pic(nx,ny)
      character*1 pim(65600),header(64)
      integer ilady(256,256)
      character*80 inputimg
      common /image/ inputimg

        open(1,file='/images/'//inputimg,access='direct',
     + form='unformatted',recl=65600)
        write(*,*) 'reading the inputimage','/images/'//inputimg
        read(1,rec=1) (pim(j),j=1,65600)
        read(1,rec=1) (header(j),j=1,64)
        close (1)
        do 1000 i=1,nx
         k=(i-1)*ny
        l=k+65
        ll=l+255
        mm=0
        do 1001 m=1,ll
        mm=mm+1
        ilady(i,mm)=ichar(pim(m))
        if(ilady(i,mm).lt.0) ilady(i,mm)=256+ilady(i,mm)
        pic(i,mm)=float(ilady(i,mm))
1001    continue
```

```
1000     continue
      return
      end


c_____________________________________________________________

      subroutine segma(image,raw,col,output,rmean)

      integer raw,col
      real image(raw,col),output
      double precision s1,s2

      s1=0.0
      s2=0.0

      do 10 i=1,raw
         do 10 j=1,col
          s1=s1+image(i,j)
          s2=s2+(image(i,j))**2
 10   continue

      rmean= s1/(raw*col)
      output=s2/(raw*col)-(rmean)**2

      return
      end
c_________________________________________________________

      subroutine m_error(image1,image2,raw,col,output)

      integer raw,col
      real image1(raw,col),image2(raw,col),output

      s=0.0

      do 10 i=1,raw
         do 10 j=1,col
          s=s+(image1(i,j)-image2(i,j))**2
 10   continue

      output=s/(raw*col)

      return
      end
c_____________________________________________________________

      subroutine write(pic,nx,ny)
      real pic(nx,ny)
```

```
      character*1 pim(65600),header(64)
      common /header/ header

        open(1,file='image.img',access='direct',
     + form='unformatted',recl=65600)
      do 10 i=1,64
      pim(i)=header(i)
 10   continue
      do 20 i=1,nx
        do 20 j=1,ny
          ip=int(pic(i,j))
          if(ip.gt.128) ip=ip-256
           mm=64+j+(i-1)*ny
           pim(mm)= char(ip)
 20   continue
      write(1,rec=1) (pim(j),j=1,65600)
      close (1)
      return
      end

c_______________________________________________________________

      subroutine writeimg(pic,nx,ny,name)
      real pic(nx,ny)
      character*1 pim(65600)
      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_______________________________________________________

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

```
c  quantizer subroutine
c for odd number of levels (3 to 35),0, 64 and 128 levels

        subroutine quantizer(a,nb,c,n,mean,var2,level,raw,col)
c a:real image
c c:integer quantized image
c n:quantizer order
c nb:band number
c mean: image mean
c var: image variance
c level,raw,col: output array dimension

        integer raw,col,level
        real a(raw,col),p(75),pl(75),mean,var
        integer c(level,raw,col)
c
        if(n.eq.0)then
          do 4 i=1,raw
          do 4 j=1,col
            c(nb,i,j)=0
 4          continue
        go to 44
        endif
c
        m=n
        call readq(n,p,pl)
        if(n.ne.m)then
        write(*,*) 'error in reading quantizer data'
        stop
        end if

         nn = n/2
        var=sqrt(var2)

        do 10 i=1,raw
          do 10 j=1,col
              xx=(a(i,j)-mean)

            do 20 k=nn,1,-1
                px=p(k)*var
              if(xx.gt.px)then
                c(nb,i,j)=k
                go to 100
              else if(xx.lt.(-px))then
                c(nb,i,j)=-k
                go to 100
              endif
 20     continue
        c(nb,i,j)=0
 100    continue
 10     continue
```

```
 44     continue
        return
        end
c_____________________________________________________
c
c  dequantizer subroutine
c for odd number of levels (3 to 35),0, 64 and 128 levels

        subroutine dequantizer(b,nb,ic,n,mean,var2,level,raw,col)
c b:real quantized image
c ic:integer quantized image 3D
c n:quantizer order
c nb:band number
c mean: image mean
c var: image variance
        integer raw,col,level
        real b(raw,col),p(75),pl(75),mean,var
        integer ic(level,raw,col)
c
        if(n.eq.0)then
          do 4 i=1,raw
          do 4 j=1,col
            b(i,j)=mean
 4        continue
        go to 44
        endif
c
        m=n
        call readq(n,p,pl)
        if(n.ne.m)then
        write(*,*) 'error in reading quantizer data'
        stop
        end if

        var=sqrt(var2)

        do 10 i=1,raw
          do 10 j=1,col

           k=ic(nb,i,j)
           if(k.gt.0)then
             b(i,j)=pl(k)*var+mean
           else if(k.lt.0)then
             k=-k
          b(i,j)=-pl(k)*var+mean
           else if(k.eq.0)then
             b(i,j)=mean
        endif

 10     continue
 44     continue
```

```
        return
        end
c_____________________________________________________________
c
c  quantizer subroutine
c for odd number of levels (3 to 35),0, 64 and 128 levels

        subroutine quant(a,b,c,n,mean,var,raw,col)
c a:real image
c b:real quantized image
c c:integer quantized image
c n:quantizer order
c mean: image mean
c var: image variance
        integer raw,col
        real a(raw,col),b(raw,col),p(75),pl(75),mean,var
        integer c(raw,col)
c
        if(n.eq.0)then
          do 4 i=1,raw
          do 4 j=1,col
            b(i,j)=mean
            c(i,j)=0
 4        continue
        go to 44
        endif
c
        m=n
        call readq(n,p,pl)
        if(n.ne.m)then
        write(*,*) 'error in reading quantizer data'
        stop
        end if

         nn = n/2
        var=sqrt(var)

        do 10 i=1,raw
          do 10 j=1,col
              xx=(a(i,j)-mean)

            do 20 k=nn,1,-1
                px=p(k)*var
              if(xx.gt.px)then
                b(i,j)=(pl(k)*var)+mean
                c(i,j)=k
                go to 100
              else if(xx.lt.(-px))then
                b(i,j)=(-pl(k)*var)+mean
                c(i,j)=-k
                go to 100
```

```
                endif
 20       continue
          b(i,j)=0+mean
          c(i,j)=0
 100      continue
 10       continue
 44       continue
          return
          end
c_____________________________________________________________

c read subroutine which
c read quantizer data

          subroutine readq(n,p,pl)
          real p(75),pl(75)
         open(32,file='../quantizer.dir/quant2.dat'
     &        ,access='direct',form='unformatted',recl=1000)

          nn=(n)/2
          if(n.le.35.and.n.ge.3)then
             number=n/2+1
          else if(n.eq.64) then
             number=20
          else if(n.eq.128) then
             number=21
          else
             write(*,*) 'the order of the filter is rong'
             STOP
          endif

          read(32,rec=number) n,(p(i),i=1,nn),(pl(i),i=1,nn)
          close(32)

          return
          end

c_____________________________________________________________
c
c
c This subroutine allocate the bitrate for each subband
c by finding the suitable quantizers
c Number of band is 64
c

          subroutine entropy(iq,var,mean)
          parameter(nn=64)
          real var(nn)
          real mean(nn),br(nn)
          integer iq(nn)
          double precision xx
```

```
        logical ql(nn)
        common /bitrate/ temprav
        prav=temprav
c
c  check for negative br and cancell the corespondant band
c
        do 10 i=1,nn
          ql(i)=.true.
 10   continue

 100    continue
        call gain(var,ql,xx,nn,ncount)
        rav=prav*nn/real(ncount)
        do 30 i=nn,1,-1
           if(ql(i)) then
             x1=var(i)/xx
             br(i)=rav+0.5*xlog2(x1)
             if(br(i).lt.0) then
               br(i)=0
               ql(i)=.false.
               iq(i)=0
               go to 100
             endif
           endif
 30     continue
c
c optimize the bit rates
c
 200    continue
c       write(*,*) 'rav=',rav
        call gain(var,ql,xx,nn,ncount)
c       write(*,*) xx
        do 40 i=nn,1,-1
          if(ql(i)) then
            x1=var(i)/xx
            br(i)=rav+0.5*xlog2(x1)
            in=int(2.0**br(i))
            iq(i)=(in+1)/2*2-1
            if(iq(i).lt.3) then
                iq(i)=0
            else if(iq(i).ge.128) then
                iq(i)=128
            else if(iq(i).ge.64) then
                iq(i)=64
            else if(iq(i).ge.35) then
                iq(i)=35
            endif
            ql(i)=.false.

            if(ncount.gt.2)then
               if(iq(i).ge.3) then
```

```
                xiq=real(iq(i))
                xr= xlog2(xiq)
              else
                xr=0
              endif
               xc=real(ncount)
               xrav=rav
               rav=(xrav*xc-xr)/(xc-1.0)
c         write(55,*) br(i),iq(i),xr,var(i),rav
              go to 200
            endif
          endif
 40     continue

c calculate the actuall bit rate
        sum=0.0
        do 50 i=1,nn
c           write(55,*) i,iq(i),var(i),mean(i)
           if(iq(i).gt.1) then
             br(i)=xlog2(real(iq(i)))
           else
             br(i)=0
           endif
           sum=sum+br(i)
 50     continue
        rav=sum/nn
c       write(55,*) 'rav =',rav

        return
        end
c_____________________________________________________________
c
c This subroutine calculates the gain

        subroutine gain(var,ql,xx,nband,ncount)

        real var(nband)
        logical ql(nband)
        double precision xx

        xx=1.0
        count=1.0
        do 10 i=1,nband
          if(ql(i))then
           xx=xx*var(i)
           count=count+1
          endif
 10     continue

        xx=xx**(1.0/count)
        ncount=int(count+.5)
```

```
        return
        end
c_____________________________________________________-

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

        subroutine bitrates(ic,ln,entropy,pr,level,raw,col)

        integer ic(level,raw,col),ln,level,raw,col
        real entropy(level),sum(-64:64),pr(-64:64)


        do 20 n=-64,64
         sum(n)=0.0
 20     continue

        do 10 i=1,raw
        do 10 j=1,col
           k=ic(ln,i,j)
           sum(k)=sum(k)+1
 10     continue
        entropy(ln)=0.0
        total=real(raw*col)
        do 30 n=-64,64
           pr(n)=sum(n)/total
           if(pr(n).gt.0) then
             br=pr(n)*xlog2(1.0/pr(n))
             entropy(ln)=entropy(ln)+br
           endif
 30     continue

c       write(*,*) 'entropy = ',entropy

        return
        end

c_______________________________________________________________

     subroutine convert(pr,prb,prnum,ns)

        real pr(-64:64),prb(128)
        integer prnum(128),ns

        n=0
        do 10 i=-64,64
          if(pr(i).gt.0.0) then
            n=n+1
            prb(n)=pr(i)
            prnum(n)=i
```

```
            endif
 10      continue
         ns=n

            return
         end

c_______________________________________________________________
c
c Program to sort the probability table
c

        subroutine sort(pr,prnum,ns)
        real pr(128),temppr
        integer prnum(128),tempprn
c
c sorting algorithm

        do 10 k=ns-1,1,-1
          do 20 i=1,k
           if(pr(i).lt.pr(i+1)) then
             temppr=pr(i)
             pr(i)=pr(i+1)
             pr(i+1)=temppr
             tempprn=prnum(i)
             prnum(i)=prnum(i+1)
             prnum(i+1)=tempprn
           endif
 20       continue
 10     continue

        return
        end

c___________________________________________________________
c
c subroutine to find the huffman codebook
c

     subroutine hcbook(prb,codbook,ns)

     real pr(128*129/2),prb(128)
        integer pointer(128*129/2)
     character*1 code(128*129/2)
     character*80 codbook(128),tempc,codebook
        logical flag
        do 19 k=1,128
 19        codbook(k)='

        do 50 k=1,ns
           pr(k)=prb(k)
```

```
            pointer(k)=0
            code(k)='x'
 50   continue

         do 51 k=ns+1,128*129/2
           pr(k)=0.0
           pointer(k)=0
           code(k)='x'
 51      continue

           code(ns*(ns+1)/2-1)='1'
           code(ns*(ns+1)/2-2)='0'

         last=0
         do 20 j=1,ns-2
           nn=ns-j
           xx=pr(last+nn)+pr(last+nn+1)
           code(last+nn)='0'
           code(last+nn+1)='1'
           flag=.false.

           do 10 i=last+1,last+nn
             if(flag) then
               pr(i+nn+1)=pr(i-1)
               pointer(i-1)=i+nn+1
             else
               if(xx.gt.pr(i))then
                 flag=.true.
                 pr(i+nn+1)=xx
                 pointer(last+nn)=i+nn+1
                 pointer(last+nn+1)=i+nn+1
               else
                 pr(i+nn+1)=pr(i)
                 pointer(i)=i+nn+1
               endif
             endif
 10        continue
           last=last+nn+1
 20      continue


         do 11 i=1,ns
            k=i
           codbook(i)='
           tempc=''
           do 12 j=1,ns-1
           if(code(k).eq.'0'.or.code(k).eq.'1')then
c            write(*,*) code(k)
             codebook=code(k)//tempc
c            write(*,*) codebook
             tempc=codebook
```

```
          endif
c         write(*,*) 'k=',k
          k=pointer(k)
c         write(*,*) 'k=',k
 12     continue
        codbook(i)=codebook//'    '
 11     continue

c       write(*,*) (pr(i),i=1,14)
c       write(*,*) (pointer(i),i=1,14)
c       write(*,*) (code(i),i=1,14)
c       write(51,*) (codbook(i),i=1,5)

     return
     end


c____________________________________________________________

c subroutine to code an image to hoffman

        subroutine codehc(image,level,codbook,prnum,ns,hc,ll,lev
     &                    ,raw,col)

        integer lev,raw,col
        integer image(lev,raw,col),prnum(128),ns
        character*80 codbook(128),ccode
        character*1 hc(77000)

        do 5 i=1,77000
            hc(i)=' '
 5   continue

        ncount=0

        do 10 i=1,raw
        do 10 j=1,col

          icode=image(level,i,j)

          do 20 k=1,ns
             if(prnum(k).eq.icode)then
               ncode=k
               go to 21
             endif
 20       continue
          write(*,*) 'ERROR IN READING CODBOOK'
          stop
 21       continue
```

```
          length=index(codbook(ncode),' ')-1

          ccode=codbook(ncode)

          do 30 k=1,length
            ncount=ncount+1
            hc(ncount)=ccode(k:k)
 30       continue

 10     continue

        ll=ncount
        do 50 il=1,77000
        if(hc(il).ne.'0'.and.hc(il).ne.'1') go to 60

 50  continue
 60     ll2=il-1
        if(ll2.ne.ll) then
         write(*,*) ' length of hc is not correct',ll2
        endif

        return
        end

c__________________________________________________________


c  program to decode the hoffman code
c

        subroutine decodehc(image,level,codbook,prnum,ns,hc,ll,
     &                        lev,raw,col)

        integer lev,col,raw
        integer image(lev,raw,col),icode(128*128),prnum(128),ns
        character*80 codbook(128),ccode
        character*1 hc(77000),code1,code2
        logical flag(128)

        npexel=0
        ncount=0

        do 5 i=1,ns
          flag(i)=.true.
          if(flag(i)) then
c           write(76,*) 'true'
       endif
 5      continue

        do 10 ii=1,ll
          code1=hc(ii)
```

```
            ncount=ncount+1
            do 20 k=1,ns
              if(flag(k)) then
c               write(75,*) 'true'
                ccode=codbook(k)
                code2=ccode(ncount:ncount)
                if(code2.eq.code1) then
                  length=index(ccode,' ')-1
c                 write(74,*) length
                  if(length.eq.ncount) go to 30
                else
                  flag(k)=.false.
                endif
              endif
 20       continue

          go to 10

 30       ncount=0
          do 40 i=1,ns
           flag(i)=.true.
 40       continue

          npexel=npexel+1
c         write(73,*) npexel
          icode(npexel)=prnum(k)

 10       continue

          do 50 i=1,raw
            do 50 j=1,col
            image(level,i,j)=icode((i-1)*col+j)
c           write(72,*) image(level,i,j)
 50       continue

          return
          end

c____________________________________________________________


        subroutine unifquan(pic,ll,image,order,min,step,level,raw,co
        integer raw,col
        integer image(level,raw,col),order
        real pic(raw,col)
        real min,max

c   assume the minimum value and the maximum
        min =pic(1,1)
```

```
      max = pic(1,1)

c   check if all pixels values are between min and max
      do 10 i=1,raw
        do 10 j=1,col
          if(pic(i,j).gt.max) then
            max = pic(i,j)
          else if(pic(i,j).lt.min)then
            min = pic(i,j)
          end if
 10   continue

c   rescale the image and convert it into image array

        step = (max-min)/real(order)
        ixx=order/2

        do 20 i=1,raw
          do 20 j=1,col
            image(ll,i,j) =int((pic(i,j)-min)/step+.5)-ixx

 20     continue

       return
       end

c___________________________________________________________


      subroutine unifdequan(pic,ll,image,order,min,step,level,raw,
      integer raw,col
      integer image(level,raw,col),order
      real pic(raw,col)
      real step,min


c   rescale the image and convert it into pic array

        ixx=order/2

        do 20 i=1,raw
          do 20 j=1,col
           pic(i,j)=real(image(ll,i,j)+ixx)*step+min

 20     continue

       return
       end
c___________________________________________________________
```

```
      subroutine dpcdecoder(e,img,raw,col)

         parameter(ii=32,jj=32)
         integer raw,col
         real e(raw,col),img(raw,col)
         real temp(0:ii+1,0:jj+1)

         do 10 i=0,raw+1
         do 10 j=0,col+1
             temp(i,j)=0
 10   continue

         do 20 i=1,raw
         do 20 j=1,col
            temp(i,j)=e(i,j)+0.25*(temp(i,j-1)+temp(i-1,j-1)+
     &               temp(i-1,j)+temp(i-1,j+1))
            img(i,j)=temp(i,j)
 20   continue

         return
      end

c_________________________________________________________
c
c program for noise less dpcm coding algorithm
c
           subroutine qdpcm(u,ie,iq,var,mean,raw,col,p,pl)

         parameter(ii=32,jj=32)
         integer raw,col
      real u(raw,col),ud(0:ii+1,0:jj+1),udb
         real e,ed
         real p(75),pl(75)
         integer ie(4,raw,col)
         real mean,var

         do 10 i=0,raw+1
         do 10 j=0,raw+1
          ud(i,j)=0.0
 10   continue

           a1=0.25
         a2=0.25
         a3=0.25
         a4=0.25

      do 20 m=1,raw
      do 20 n=1,col
          udb=a1*ud(m-1,n)+a2*ud(m,n-1)+a3*ud(m-1,n-1)+a4*ud(m-1,n+1
          e=u(m,n)-udb
          call quantizer1(e,ed,ie(1,m,n),iq,mean,var,p,pl)
```

```
          ud(m,n)=udb+ed
 20   continue

      return
      end
c_____________________________________________________________
c
c  quantizer subroutine
c for odd number of levels (3 to 35),0, 64 and 128 levels

         subroutine quantizer1(a,b,c,n,mean,var2,p,pl)
c a:real image
c b: real quantized image
c c:integer quantized
c n:quantizer order
c mean: image mean
c var: image variance
c
        real a,p(75),pl(75),mean,var
        integer c
c
        if(n.eq.0)then
            c=0
        go to 44
        endif
c
         nn = n/2
        var=sqrt(var2)

              xx=(a-mean)

            do 20 k=nn,1,-1
                px=p(k)*var
              if(xx.gt.px)then
                b=(pl(k)*var)+mean
                c=k
                go to 100
              else if(xx.lt.(-px))then
                b=(-pl(k)*var)+mean
                c=-k
                  go to 100
              endif
 20     continue
        b=mean
        c=0
 100    continue
 44     continue
        return
        end
c____________________________________________________
c
```

```
c program for dpcm coding algorithm

     subroutine dpcm1(x,y,raw,col)

        parameter(ii=32,jj=32)
        integer raw,col
     real x(raw,col),temp(0:ii+1,0:ii+1),y(raw,col)

        write(*,*) 'dpcm1'
        do 10 i=0,ii+1
        do 10 j=0,jj+1
           if(i.eq.0.or.j.eq.0.or.i.gt.ii.or.j.gt.jj) then
              temp(i,j)=0
           else
              temp(i,j)=x(i,j)
           endif
 10  continue

     do 20 i=1,raw
     do 20 j=1,col
       y(i,j)=temp(i,j)-0.25*(temp(i-1,j)+temp(i,j-1)
     &           +temp(i-1,j-1)+temp(i-1,j+1))
 20  continue

     return
     end
c____________________________________________________________
```

Bibliography

[1] R. E. Crochiere, S.A.Weber, and J.L. Flanagan, "Digital Coding of Speech Subbands,"*Bell Syst. Tech. J.*, vol. 55, pp.1069-1085,1976.

[2] M. Vetterli, "Multi-dimensional Sub-band Coding: some Theory and Algoithms," *Signal Processing*, pp.97-112, 1984

[3] J.W. Woods, S.D. O'Neil, "Subband Coding of Images," *IEEE Trans. ASSP*, vol. ASSP-34, no.5, Oct. 1986

[4] H. Gharavi and A. Tabatabai, "Subband Coding of Digital Image Using Two-Dimensional Quadrature Mirror Filtering," *Proc. SPIE*, vol.707, pp.51-61, Sept. 1986.

[5] P.P. Vaidyanathan, "Quadrature Mirror Filter Banks, M-Band Extensions and Perfect Reconstruction Techniques," *IEEE ASSP Magazine*, pp.4-20, July 1987.

[6] M. Smith, and T.P. Barnwell, "Exact Reconstruction Techniques for Tree-Structured Subband Coders," *IEEE Trans. ASSP*, pp.434-441, 1986.

[7] S.G. Mallat, *"A Theory for Multiresolution Signal Decomposition: The Wavelet Representation,"* MS-CIS-87-22, GRASP Lab.103, Univ. of Pennsylvania, May 1987.

[8] I. Daubechies, "Orthonormal Bases of Compactly Supported Wavelets," *Comm. on Pure and Applied Math.*, vol. XLI, pp.909-996, 1988.

[9] N. Tanabe and N. Farvadin, *"Subband Image Coding Using Entropy-Coded Quantization Over Noisy Channels,"* Computer Science Technical Report Series, University of Maryland, College Park, Maryland, August 1989.

[10] A. N. Akansu, R. Haddad, H Caglar, "The Bionomial QMF-Wavelet Transform for Multiresolution Signal Decomposition," *Submitted to IEEE Trans. ASSP.*

[11] H. Caglar, and A. N. Akansu, "A Generalized, Parametric PR-QMF Design Technique Based on Bernstein Polynomial Approximation," *Submitted to IEEE Trans. IT.*

[12] S.G. Mallat, "Multifrequency Channel Decomposition of Images and Wavelet Models," *IEEE Trans. ASSP*, pp.2091-2110, Dec. 1989.

[13] R.A. Haddad, "A Class of Orthogonal Nonrecursive Binomial Filters," *IEEE Trans. Audio and Electroacoustics,* pp.296-304, Dec. 1971.

[14] R.A. Haddad and A.N. Akansu, "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing," *IEEE Trans. ASSP,* pp. 723-727, March 1991.

[15] Y. Shoham and A Gersho, "Efficient Bit Allocation for an Arbitrary Set of Quantizers," *IEEE ASSP,* pp. 1445-1453, September 1988.

[16] N.S. Jayant and P. Noll, *Digital Coding of Waveforms,* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1984.

[17] Anil K. Jain, *Fundamentals of Digital Image Processing,* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1989.

[18] A. N. Akansu, and Y. Liu, "On Signal Decomposition Techniques," Optical Engineering, to appear in July 1991 issue.

[19] A. N. Akansu, and M. S. Kadur, "Subband Coding of Video with Adaptive Vector Quantization," Proc. ICASSP' 90, pp. 2109-2112.

[20] C. E. Shannon, "A Mathematical Theory of Communication," BSTJ, vol. 27, pp. 379-423, 623-656, 1948.

[21] S. P. Lloyd, "Least Squares Quantization in PCM," Bell Laboratories Technical Note, 1957; reprinted in IEEE Trans., Vol IT-28, no.2, pp. 129-137, March, 1982.

[22] Y. Linde, A. Buzo, and R. M. Gray,"An Algorithm For Vector Quantizer Design," IEEE Trans. on Com., Vol. Com-28, no.1, pp. 84-95, January, 1980.

[23] M. S. Kadur,*Adaptive Subband Video Coding with Motion Compensation,* M.S. Thesis, NJIT, May 1989.