İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DCT İLE RESİM KODLAMA UYGULAMALARI

YÜKSEK LİSANS TEZİ

Müh. Vadi DİPÇİN

Tezin Enstitüye Verildiği Tarih: 27 Mayıs 1996

Tezin Savunulduğu Tarih: 14 Haziran 1996

Tez Danışmanı: Doç. Dr. Melih PAZARCI

Diğer Jüri Üyeleri: Prof. Dr. Osman PALAMUTÇUOĞULLARI

Prof. Dr. Erdal PANAYIRCI

HAZİRAN 1996

ÖNSÖZ

Bu tez çalışması sırasında bana büyük desteği olan tez danışmanım sayın Doç.Dr. Melih Pazarcı'ya, çalışmalarımda yardımcı olan Ayla Çevik ve Araştırma Görevlisi Tuna Tarım'a, ayrıca bana her konuda destek olan aileme teşekkürü bir borç bilirim.

İÇİNDEKİLER

ÖZET

Günümüzde, günlük yaşamı etkileyen en önemli gelişmelerin başında sayısal teknoloji gelmektedir. Bu teknolojinin sağladığı en önemli olanaklardan biri de sayısal televizyondur. Çok yoğun çalışmaların yürütüldüğü bu alanda en önemli sorunlardan birisi hiç şüphesiz sayısal resim bilgisinin büyüklüğüdür. Gerek depolanması, gerekse işlenmesi ve yayınlanması aşamalarında sistem performansları ve fiyatları açısından, sayısal resim bilgisinin sıkıştırılması bir gerekliliktir.

Bunun sağlanması amacıyla yapılan çalışmalar yaygın olarak kullanılan iki standardın ortaya çıkmasına yol açmıştır. Hareketsiz resimler için JPEG, hareketli görüntüler için ise MPEG standartları ticari uygulamalarda yer almaktadır. Bununla birlikte daha iyi kodlama yöntemleri üzerindeki çalışmalar da devam etmektedir. Bu tezde hareketsiz resimler için alternatif kodlama yöntemleri üzerinde durulmuştur. Bütün yöntemler JPEG ve MPEG'in kullandığı 8*8 Ayrık Kosinüs Dönüşümü (Discrete Cosine Transform: DCT) üzerinde kurulmuştur.

Başlangıç bölümünde resim kodlaması ile ilgili kavramlar ve yöntemler genel olarak tanıtılmıştır. Üçüncü bölümde resim bilgisine kaskad DCT uygulanmasına dayanan bir yöntem incelenmiştir. Dördüncü bölümde çeşitli yöntemlere dayanan vektör kuantalama uygulamalarına yer verilmiştir. Beşinci bölümde, resmin, dönüşüm uygulanmadan önce küçük benzer parçalara ayrılması yoluyla kodlanması gerçekleştirilmiştir. Altıncı bölümde verilen uygulamada JPEG standarında kullanılan lineer skalar kuantalama yerine düzgün olmayan skalar kuantalama ve uygun bir akış uzunluklu kodlama kullanılmıştır.

İlk üç yöntemin istenen performansı verememesiyle birlikte, düzgün olmayan skalar kuantalama uygulaması kayda değer bir verimlilik göstermiştir ve JPEG standardına göre daha avantajlı bir kodlama yöntemi elde edilmiştir. Bu kodlama yöntemi kolaylıkla JPEG ve MPEG'e uyarlanarak bu standartların daha verimli bir şekilde çalışmalarına imkan verebilir.

SUMMARY

IMAGE CODING APPLICATIONS USING DCT

The very rapid development in digital technology in last the 30 years brought many new aspects to our life. Digital systems got cheaper and more sophisticated. This made many things possible which were thought to be just imagination.

One step of this development is Digital TV. Such systems will provide higher quality for pictures using narrow band widths, more compatibility etc. for broadcasting. However, there is an important problem about the size of digital images. A picture of 720*576 luminance resolution and 360*576 chrominance resolution (for each chrominance component) is about 830kB (CCIR 601, 4:2:2 standard). A motion picture using such frames will require about 10GB/min. Not only the transmission, but also the storage of such a big amount information is not practical and very expensive.

This yields the need of compression of digital picture information. A very well known standard for still images is JPEG (Joint Photographic Experts Group). The world wide used standard for motion pictures is MPEG (Motion Pictures Experts Group). MPEG-2 is now being used for digital broadcasting of analog TV signals such as PAL or NTSC.

This thesis proposes a new coding system for still images. All coding systems obtained and tested, depend on the 8*8 DCT (Discrete Cosine Transform) which is also used by both JPEG and MPEG. The definition of N*N forward and inverse DCT is,

$$f(u,v)=\frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \quad (1)$$

$$f(x,y)=\frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}C(u)C(v)f(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \quad (2)$$

where $C(u),C(v)=\frac{1}{\sqrt{2}}$ for u,v=0 and C(u),C(v)=1 for u,v $\neq$0

In image coding systems, usually a quantization is applied after the transform. There are two main kinds of quantization:

1- Scalar Quantization.
2- Vector Quantization.

When applying scalar quantization, each transform coefficient is processed separately. They are either divided by certain divisors, or they are set to some certain quantized values. If division is used, the quotients are coded and stored. During the decoding process, the quotients are multiplied with the divisors used for division and the transform coefficients are obtained. The critical point is that the quotients are rounded to integers before coding. Therefore, the transform coefficients obtained after the multiplication are not equal to the original values. Thus, the picture obtained using the coded data is not exactly the same as the original one and an error occurs. This kind of quantization is **Linear Scalar Quantization.**

The second method of scalar quantization is the **Nonuniform Scalar Quantization.** In this method, certain quantization values are determined according to the probable value set of the coefficients. For all coefficients, the quantization values are scanned and the closest value is chosen. Statistically, most of the coefficients have small values. Therefore, small values are quantized more precisely. For each quantization value, the index number is coded and stored. Lloyd-Max iteration can be used to optimize the codebooks.

In vector quantization, coefficients are coded in groups. The grouping may change according to the application. The size of vectors and the codebook determine the quality of the coded picture. LBG algorithm (Linde-Buzo-Gray) can be used to optimize the codebooks. For each vector, the whole codebook is scanned and the closest code is chosen. MSE (Mean Squared Error) measure is used for comparison.

All these quantization methods are lossy and cause error in the pictures. The amount of the error is important. It may cause important artifacts on the picture. To avoid this, the human visual system is to be studied. The human eye can not see all the high frequency patterns. Therefore, quantization systems are mostly designed so that they will remove the high frequency information and keep the relatively lower frequency information. Scalar or vector, all quantizers must be determined considering the visual effects on the coded pictures.

After quantization, the last step is the **Lossless Coding.** This includes methods like Huffman Coding or Run Length Coding (RLE). Huffman coding reduces the average codeword length in a message sequence to the value of the **entropy.** Entropy is the amount of average information in the message sequence.

RLE is a different coding system. If many coefficients following each other are equal, then they are counted and the count-value is stored with the repeated value. This may be very efficient if many coefficients following each other are equal.

In the thesis, four methods are studied. The first one is the application of **Cascade DCT.** 8*8 DCT is applied to 256*256 pixelsize 8 bit/pixel (256 gray level) pictures. The DCT coefficients which have values between -1024 and 1023, are then scaled to [0,255]. This is done by adding 1024 to the coefficients and dividing by 8. Then all the DCT coefficients of different 8*8 blocks with the same index are to be grouped so that we obtain 32*32 size blocks (There are 1024 each of 64 different frequency components because picture size is 256*256.).

The result of quantization is that many coefficients go to zero after the process. In order to realize this, some selected frequency components may be set to zero. A **zero setting process** can be defined (zonal coding). The "level" of the process is defined as the number of diagonal lines set to zero from bottom right to top left, beginning from the right bottom corner of the 8*8 matrix. The example is given in Figure 1.

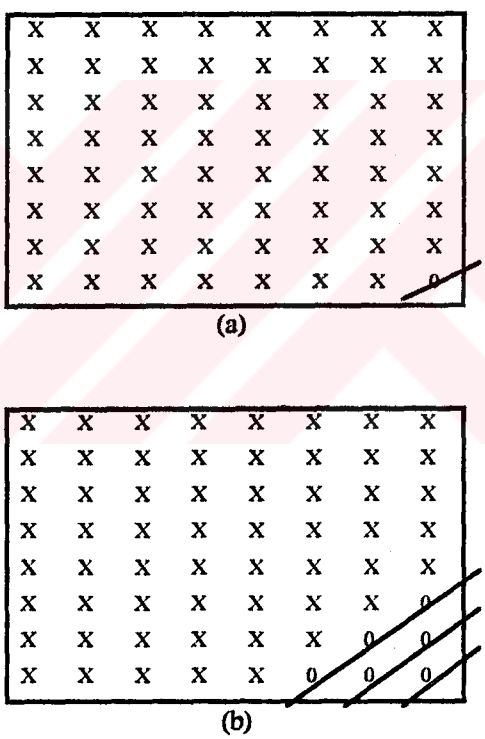


Figure 1: Zero Setting Process. (a) 1.st level. (b) 3.rd level.

After applying the 8.th level zero setting process, the grouping is done and the second DCT is implemented. The main objective of this method is to represent the image data with a set of numbers which gets smaller after each DCT step. Therefore another zero setting process is applied after the second DCT. The second zero setting is done at level 4 and level 8. Then the picture is reconstructed from this data.

The effect of this coding on the picture is interesting. Patterns on the pictures (like the nose holes of Tiffany) are copied onto adjacent blocks. The distance between repetitions is exactly 8 pixels.

The reason of this effect is the zero setting process after the second DCT. Zero setting process is a Low-Pass process which makes adjacent values closer or equal. So, the coefficients of the first DCT are made closer or equal because of the zero setting process applied after the second DCT. Therefore, frequency components of neighbor blocks become similar and the inverse DCT yields similar patterns. This means that the information obtained after the second DCT must be kept at a better resolution than tha first DCT hence this tecnique with Cascade DCT does not bring any advantage for coding systems.

The second studied method is Vector Quantization of DCT coefficients. This is done in many ways, using 2*2, 4*1, 8*1 vectors and with one or two codebooks. For preliminary study, only the DCT coefficients (0,1) (first order vertical frequency component) and (1,0) (first order horizontal frequency component) are quantized. It is confirmed that bigger vector sizes (with equal codebook size) give worse results compared to smaller vector sizes. In the application of quantization with two codebooks, the first codebook is used to quantize the coefficients and the second one is used to quantize the error vectors. This improves the picture quality compared to the application with only one codebook. On the other hand, no generally usable single codebook is obtained. All the quantization processes cause **blocking effects** on the pictures.

The third method divides pictures into smaller and similar parts in the spatial domain. The division ratio of parts may change. For 256*256 pixel picture sizes, 64 is used in order to obtain a good compression quotient. The process is as follows. The pixels of the left top 8*8 block are selected as the left top pixels of 64 small pictures. Pixels of the right neighbor of 8*8 block are selected as the right neighbor pixels of the first ones etc. So, 64 similar but different, offset decimated small copies of the full picture are obtained. This is done in order to obtain higher similarities. The DCT coefficients of similar pictures should be similar and this may result in good compression quotients. However, the similarities are not as high as expected. Since there are 64 similar small pictures, 64 DCT coefficients are relevant and their values are expected to be close. The distribution of their value is about ±100 around the mean and this not enough. Because the similarity is not strong enough, the coefficients are requantized to [0,255] range like in the case of Cascade DCT application so that the distribution becomes more convenient. Then the difference values of the 64 regrouped relevant coefficients from their means are calculated. The differences are Vector Quantized (4*4), and the vector indexes and the mean are stored. The picture reconstructed from the coded data has strong **noise**. Because the picture is divided and neighbor pixels are transformed and quantized separately, there are no blocking effects. The spatial error is like white noise. There are also some **contour** effects. Due to coding noise and counter effects, this method is not recommended for further use.

The last application is the **Nonuniform Scalar Quantization (NSQ)**. This is implemented in such a way that JPEG's coding system is improved. The method first implements 8*8 DCT like JPEG and then reorders the coefficients of each 8*8 block as in JPEG. This order is called the **zig zag scan order**. At this point, JPEG uses linear scalar quantization then Run Length Coding (RLE) and finally Huffman Coding. The method in this thesis uses NSQ, Run Length Coding and Huffman Table.

In JPEG's method, RLE uses some additional bits which show the exact value of coefficients. The number of zeros following each other is determined and the first nonzero coefficient is coded with this number. There are only 4 bits to store the information in the coefficient's value and this is not enough because the quantized values may change between -1024 and 1023. So, the 4 bits are used to store group numbers which show intervals. Additional bits are required in order to represent the exact value of the quantized coefficients in the particular interval.

On the other hand, in NSQ, only index numbers are stored. The maximum codebook size needed to obtain a picture quality as in JPEG's method quality level 50 (IJG implementation), is 25 (the code book size is less than 16 for 90% of frequency components!). Thus, maximum 5 (mostly 4 or less) bits are needed to express the exact value of coefficients. The RLE method constructed for NSQ can supply these 5 bits and, there is no need for the additional bits which can't be compressed using Huffman Code. An additional advantage is that the entropy of the coefficients are smaller compared to linear scalar quantizers because the source alphabet will contain less messages.

The obtained pictures are compared with JPEG pictures, at the quality level around 50. It's very clear that NSQ can compress pictures about 5% better for the same quality when the optimized Huffman Table is used by JPEG. Examples of results are given in Table-1 and Table 2.

Using NSQ, JPEG's standard can be made more efficient and faster because NSQ doesn't need multiplications and divisions. Only subtraction is needed in order to compare the coefficients with the quantization values in the codebook because the closest quantized value is searched for each coefficient. Even the subtraction is not used during the decoding process. On the other hand, NSQ yields certainly more efficient compression than JPEG's linear scalar quantization at the same picture quality. These two advantages are the main reason for further study of the new coding system. JPEG standard can be modified so that it uses NSQ.

Table-1: Comparison of NSQ with regular JPEG. All the JPEG pictures are compressed at quality level 50. The obtained file size is given in the 7.th column. If a custom optimized Huffman table is constructed which requires an additional pass, than the file sizes given in the 8.th column are obtained.

	NSQ			JPEG			
PICTURES	MAE	PSNR (dB)	File Size	MAE	PSNR (dB)	File Size	File Size (Opt.Coding)
Airplane	3.97	32.67	7217	3.67	33.07	7941	7412
Baboon	8.32	27.31	14010	8.24	27.38	16969	16199
Lena	3.74	33.55	6759	3.55	33.50	7687	7137
Ms. America	2.87	36.78	3138	2.51	37.52	3745	3112
Salesman	3.77	34.22	7090	3.61	34.25	7988	7427
Splash	3.20	34.81	5033	3.13	35.43	5791	5058
Susie	2.59	37.37	4172	2.54	37.63	5059	4428
Tiffany	3.70	33.74	5142	3.48	33.83	6251	5637

Table-2: Compression of NSQ with regular JPEG given in percent of compression improvement gain for NSQ.

PICTURE	JPEG with Standard Huffman Table	JPEG with Optimum Huffman Table
Airplane	%9.1	%2.6
Baboon	%17.4	%13.5
Lena	%12.1	%5.3
Ms. America	%16.2	%-0.8
Salesman	%11.2	%4.5
Splash	%13.1	%0.5
Susie	%17.5	%5.8
Tiffany	%17.7	%8.8
MEAN	%16.2	%5.03

The only disadvantage of NSQ is that it's not as flexible as linear quantization in the sense of choosing different quality levels. In order to code pictures at different quality levels, NSQ codebooks must be changed and the RLE method must be modified to accommodate codebooks of different sizes.

There are two possibilities. The first one is to determine a constant quantizer whose quality level and compression quotient are acceptable. This is useful, especially in cases where different quality levels are not required (may be useful for MPEG). The second possibility is to construct different quantizers to provide different quality levels. In this case, RLE implementation details must be determined for each of them, separately as well.

In both cases, NSQ will have its advantage against linear scalar quantization providing better compression.

BÖLÜM 1
GİRİŞ

Özellikle son 30 yılda çok hızlı bir gelişme gösteren sayısal teknoloji, hem pek çok yeniliği günlük yaşantımıza sokmuş hem de yalnızca hayal olduğu sanılan bir çok şeyin gerçekleşmesini mümkün kılmıştır. İlk bilgisayar olan 30 tonluk ENIAC'ı yapan mühendislerin "Geleceğin bilgisayarları mutlaka 1.5 tondan daha hafif olacaktır." şeklindeki açıklamaları bunu anlatan en güzel örnektir.

Sayısal sistemlerin performanslarının hızla artmasının yanı sıra maliyetlerin de aynı hızla düşmesi sonucu, her konuda kullanılmaları mümkün olmuştur. Bunların başında hiç şüphesiz iletişim gelmektedir. Gerek bankacılık sektöründeki gibi çeşitli formatlardaki bilgilerin iletilmesi, gerekse ses ile hareketli görüntü iletimi önem kazanmıştır.

Televizyon yayıncılığında HDTV için hazırlıklar yapılırken, günümüzde analog temele dayanan yayınlar (PAL , NTSC gibi) MPEG-2 standardı ile kodlanarak uydular üzerinden iletilmektedir ki, bu teknoloji halen ülkemizde bile kullanılmaktadır(MPEG: Motion Picture Experts Group). Diğer yandan çok yakın tarihe kadar öngörülememiş bir hızla yayılan INTERNET ağı üzerinden yayıncılık veya çeşitli ülkelerde denemelerine başlanan etkileşimli televizyon (Interactive TV) gibi kavramlar tamamen sayısal teknolojinin gelişmesine bağlıdır.

Bu noktadaki en önemli sorun, ham sayısal resim bilgisinin büyük miktarda veri içermesi ve bu yüzden oldukları gibi yayınlanmalarının mümkün olmamasıdır. Örneğin CCIR 601, 4:2:2 formatındaki 8 bitlik bir görüntü (Y=720*576, Cr=360*576 Cb=360*576) 830kB gibi bir büyüklüktedir. Bu durumda saniyede 25

kare göstermesi gereken bir sistemde 166MB/sn'lik bir performansa ulaşılması gerekir. Bir dakikalık bir görüntü dizisi ise yaklaşık 10GB'lık yer kaplayacaktır. Bu kadar büyük miktarda bilginin bu hızla yayınlanabilmesi bir yana depolanması bile bir sorundur. Bu yüzden sayısal resim bilgisini sıkıştırmak zorunludur.

Yapılan çalışmalarda bu konuda önemli aşamalar kaydedilmiştir. Hareketsiz görüntülerin sıkıştırılmasında JPEG(Joint Photographics Expert Group) standardı oluşturulmuştur ve bu standart yaygın olarak kullanılmaktadır. Diğer yandan hareketli görüntüler için yukarda sözü edilen MPEG [1] standardı da günümüzde kullanılmaktadır. Bilgisayar ortamında hareketli görüntüler için MPEG-1 ve günümüzdeki TV sistemlerinde kullanılmak üzere MPEG-2 standartları oluşturulmuştur. MPEG-4 standardı için çalışmalar devam etmektedir(MPEG-3, MPEG-2 ile birleştirilmiştir.). MPEG yapısı gereği JPEG'in sıkıştırma tekniklerini de içermektedir.

Hareketli görüntü kodlama yöntemlerinin iyileştirilmeleri için, hareketsiz görüntü kodlama yöntemleri üzerinde de çalışılmaktadır. JPEG 8*8 DCT'ye dayanan bir standarttır. Günümüzde alternatif olarak Dalgacık Dönüşümü[2],[3] (Wavelet Transform) ve Fraktal Kodlama gibi yöntemler üzerinde çalışılmaktadır.

Tezde, şu anda en yaygın olarak kullanılan 8*8 DCT'ye dayanan alternatifler incelenmiştir. Dört ayrı bölümde verilen dört ayrı yöntem ile kodlama yapılmış, bunların avantaj ve dezavantajları ortaya konmuştur. Bu uygulamaların tümünde 256*256 boyutlarında 8 bitlik, yani 256 gri ton seviyeli resimler kullanılmıştır. Amaç olarak 8*8 DCT ile elde edilen dönüşüm katsayılarının daha verimli bir şekilde kodlanması hedeflenmiştir. Hata ölçüleri olarak MAE ve PSNR kullanılmıştır. Altıncı bölümde anlatılan son yöntem olan Düzgün Olmayan Skalar Kuantalama ile JPEG standardı hem daha hızlı hem de daha verimli kodlama yapabilecek bir şekilde geliştirilmiştir.

BÖLÜM 2
TEMEL KAVRAMLAR

2.1. RESİM KODLAMA SİSTEMLERİNİN GENEL YAPISI

Bir resmin sayısal sistemlerde işlenmesi için yapılması gereken ilk işlem örneklemedir. Eğer resimler analog sistemlerle kaydedilmişse, bunların sayısal ortama aktarılabilmesi için örneklenmeleri gerekmektedir. Burada örnekleme ile ilgili teorik bilgi verilmemekle beraber, bazı önemli noktalara değinilecektir.

Analog bir ortamda saklanmakta olan bir resim bilgisinin örneklenmesinde kaliteyi belirleyen en önemli faktör çözünürlüktür. Burada çözünürlükle hem resmin yatay ve düşey eksenlerde kaçar piksel ile örneklendiği, hem de renk çözünürlüğü kastedilmektedir. Piksel çözünürlüğü kullanım amacına göre uygun bir şekilde belirlenmelidir. Bu da resmin kullanılacağı veya gösterileceği ortamın işleyebileceği maksimum çözünürlükle sınırlıdır. Bu miktarın üstündeki çözünürlükte örnekleme yapmak bir kalite kazancı sağlamaz. Tabi resmi içeren analog ortamın da istenen kaliteyi sağlayabilmesi gereklidir. Örneğin 35mm'lik filmler HDTV uygulamaları için yeterli kaliteyi sağlarken, 8mm veya 16mm'lik filmler HDTV için uygun kaynaklar değildir.

Renk çözünürlüğünde durum daha basittir. Gözün seçebildiği renk sayısının 24 bitlik resimlerin içerdiği 16.7milyon renkten biraz daha düşük olmasından dolayı, hiçbir uygulamada bunun üstü düşünülmez. 24 bitlik resim denildiğinde kastedilen, her bir pikselin rengi için 24 bit ayrılmış resimdir. Bu 24 bit genellikle 3*8 bit olarak R(kırmızı), G(yeşil), B(mavi) arasında bölüşülür. Böylece her ton (0-255) aralığında

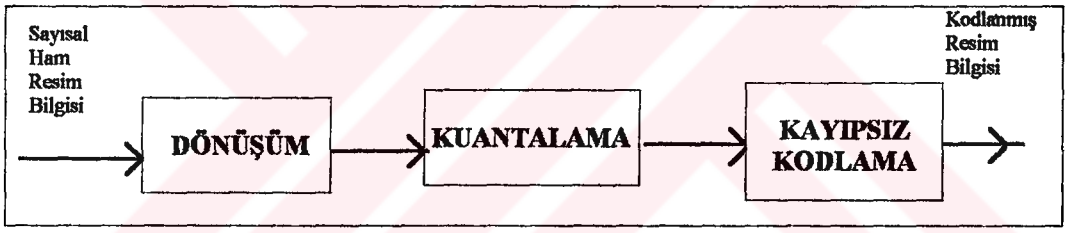
bir değer alır ve bu üç rengin karışımından istenen renk elde edilir. Zaten bilindiği gibi günümüzde renkli monitörlerde renk oluşumu da R,G,B bileşenleri kullanılarak sağlanmaktadır. 24 bitlik resimler gözün renk ayırma sınırının üstünde olduğundan 24 bitlik resimlere **gerçek renkli resim (true color picture)** denir. R,G,B'nin dışında farklı renk koordinat sistemleri de vardır. Bunlarda genellikle parlaklık bilgisi ile 2 ayrı renk bilgisi kullanılır. Böyle koordinatlarda resimde parlaklık bilgisi ile renk bilgisi ayrılmış olur. Bu ayrımın kaynağı klasik TV sistemleridir. Renkli yayın sistemlerinde parlaklık bilgisi ile renk bilgisi ayrı ayrı kodlanmak zorundaydı. Bunun sebebi ise siyah beyaz sistemlere uyumluluk sağlanmasıydı. Siyah beyaz sistemlerle aynı bant genişliği içine renk bilgisinin de sığdırılması için çalışılırken, renk bilgisi için R,G,B koordinatlarının dışındaki bazı koordinat sistemlerinin daha küçük bir bant genişliği gerektirdiği anlaşılınca bu koordinat sistemleri kullanılmaya başlandı. Böylece analog ortamda doğal bir sıkıştırma elde edilmiş oldu. JPEG'de de böyle bir koordinat sistemine geçildikten sonra kodlama yapılır. 24 bitlik bir resim, her piksel için 3 bayt gerektiğinden, içerdiği nokta sayısının 3 katı kadar bayt bilgi içerir. Parlaklık bilgisinin R,G,B bileşenleri cinsinden ifadesi şu şekildedir:

$$Y=0.299R+0.587G+0.114B \qquad (2.1)$$

(2.1)'e göre elde edilen parlaklık bilgisi, resmin siyah beyaza dönüştürülmüş haline ait bilgiyi taşımaktadır. Eğer resim siyah beyaz olarak saklanmak veya işlenmek isteniyorsa, sadece bu bilgiyi kullanmak yeterlidir. Bu durumda resme ait bilgi 8 bite inmiş olur. Yani üç bileşen yerine bir bileşen saklanmış olur. '0' siyahı, '255' ise beyazı gösterir. Gözün görme özellikleri açısından 8 bitlik, yani 256 ayrı gri ton bilgisi içeren bir resim fotoğraf kalitesinde görünür. Çünkü göz parlaklığı 256'dan daha az ayrı tonda algılayabilir. Bu yüzden 8 bitlik siyah beyaz bir resim yeterli kaliteye sahiptir. Tezde kullanılan resimlerin tümü de 8 bitlik siyah beyaz resimlerdir.

Sayısal ortamda bulunan bir resmi, resim boyutunda bir matrise benzetebiliriz (Resim renkliyse 3 ayrı matris.). Bu formda bir resmin sayısal yöntemlerle kodlanarak sıkıştırılması işlemi temelde üç aşamada gerçekleşir. Bu üç aşamalı sistem Şekil2.1'de gösterilmiştir. Resim önce uygun bir transform ile uzaysal domenden bir

başka domene aktarılır(Herhangi bir dönüşüm uygulamadan doğrudan resim üzerinde kodlama yapan sistemler de vardır, ama genelde bir dönüşüm yapılması sıkıştırılabilirlik açısından yarar sağlar.). Daha sonra kuantalama uygulanır. Kuantalama, dönüşüm sonucu elde edilen bilginin sıkıştırmaya müsait bir hale gelmesi amacıyla uygulanan ve kayıplı olan bir aşamadır. Bu yüzden resim kodlama sistemleri genellikle kayıplı sistemlerdir. Yani görüntü yeniden elde edildiğinde ortaya çıkan resim, orjinal resmin tamamen aynısı değildir. Burada önemli nokta, kodlamadan sonra elde edilen resmin yeterli kaliteye sahip olması ve gözü rahatsız edecek hatalar içermemesidir. Son adım ise kayıpsız kodlamanın yapıldığı adımdır. Entropi kodlaması, Akış Uzunluklu Kodlama (Run Length Coding) gibi yöntemler kullanılır. Bu şekilde kodlama tamamlanır ve kodlanmış resim bilgisi kullanılmak üzere saklanır.



Şekil 2.1: Sayısal bir kuantalayıcının Akış Diyagramı

2.2. DÖNÜŞÜM

Sayısal kodlama için kullanılan ve farklı teorik temellere dayanan bir çok dönüşüm vardır. Sağlaması gereken özelliklere göre katsayıları belirlenen FIR veye IIR filtrelerin dışında sayısal ortam için geliştirilmiş dönüşümler ve analog dönüşümlerin sayısal uygulamaları vardır(Ayrık Fourier Dönüşümü gibi).

Doğal bir resmin komşu pikselleri arasında çok büyük oranda ilişki vardır. Bunun sebebi doğal görüntülerde ani renk veya parlaklık atlamaları olmaması ve bu özelliklerin pikselden piksele yavaş yavaş değişmesi hatta belirli bölgeler içinde

hemen hemen sabit kalmasıdır. Örneğin bir insan portresi düşünüldüğünde, kişinin yüzünün yer aldığı bütün pikseller ten renginde olacak ve komşu piksellere ait R,G,B bileşenleri çok yakın değerli olacaktır. Bu yüzden komşu iki piksele ait R,G,B bileşenleri bir iletişim sisteminde ayrı ayrı gönderildiğinde birbirine çok yakın iki ayrı bilgi gönderilmiş olacaktır. Bu da gereksiz yere sistem performansını düşürecektir. Üstelik resim bilgisinin ne kadar büyük miktarda olduğu düşünülürse bu şekilde gereksiz bilginin gönderilmesi daha da önem kazanır.

İşte resimlere dönüşüm uygulanmasının temel sebebi budur. Dönüşüm ile ulaşılmak istenen temel amaç, yeni domende elde edilen katsayıların mümkün olduğunca ilişkisiz (dekorele) olmasıdır. Böylece saklanacak veye iletilecek her katsayı ayrı bir bilgi parçası taşımış olur.

Bu amaçla kullanılmak üzere geliştirilmiş olan Karhunen-Loewe Dönüşümü (KLT) [2][3], resim için kabul edilen özilişki matrisine dayanarak oluşturulduğundan ilişkisizlendirme işlemini optimum seviyede yapar. Resimlerin istatistik özellikleri için çeşitli modeller oluşturulmuştur. Şunu öncelikle belirtmek gerekir ki, aslında hiç bir model yeterince iyi bir tanımlama oluşturamamaktadır; çünkü resimlerin istatiksel özellikleri değişkendir. KLT, **giriş işaretine bağımlı bir dönüşümdür** ve özellikle hızlı olması gereken kodlama işlemleri için uygun değildir.

Bu aşamada **Ayrık Kosinüs Dönüşümü** (Discrete Cosine Transform, **DCT**) çok güzel bir şekilde devreye girmektedir. Ayrık Fourier Dönüşümü'nün reel bir çeşidi olan DCT, KLT gibi optimum ilişkisizlendirme özelliğine sahip olmasa da, bu işlevi optimuma çok yakın bir seviyede yapmaktadır. Hatta en yaygın olarak kullanıldığı 8*8 boyutlarında KLT ile arasındaki fark son derece küçüktür. Diğer yandan sabit bir yapısı olduğu gibi en büyük avantajı gerçekleştirilmesi için hızlı algoritmaların bulunmasıdır. Bu yüzden günümüzde yaygın standartlar haline gelen JPEG ve MPEG standartları 8*8 DCT'ye dayanmaktadır. İki boyutlu N*N DCT ile Ters DCT'nin tanımları sırasıyla (2.3.a) ve (2.3.b)'de verilmiştir.

$$f(u,v)=\frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad \textbf{(2.3.a)}$$

$$f(x,y)=\frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}C(u)C(v)f(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad \textbf{(2.3.b)}$$

u,v=0 için $C(u),C(v)=\frac{1}{\sqrt{2}}$ ve u,v≠0 için C(u),C(v)=1

Tezdeki uygulamaların tümü de 8*8 DCT'ye dayanmaktadır. KLT, DCT gibi sabit dikdörtgen piksel bloklarını dönüştüren dönüşümlere **Blok Transform** denir. Bu dönüşümler uygulanacağı resim boyutlarına bağlı matrislerle belirlenirler ve dönüşüm ile elde edilen matris de aynı boyutlardadır. x resim matrisini, A transformu belirleyen dönüşüm matrisini ve X de dönüşüm ile elde edilen matrisi göstermek üzere, blok transformlarla yapılan bir dönüşüm genel olarak (2.4)'teki gibi gösterilir.

$$X=A.x \qquad \textbf{(2.4)}$$

8*8 DCT ile elde edilen 8*8'lik katsayı matrisi aslında resmin frekans bileşenlerine ait bilgileri taşımaktadır. Sol üst köşedeki katsayı DC değeridir ve DCT'si hesaplanan 8*8'lik bloğun ortalama parlaklık değerine ait bilgiyi taşır. Hemen sağındaki katsayı (1,0) elemanıdır. Ve 1. mertebe yatay frekans bileşenine ait katsayıdır. DC'nin hemen altındaki (0,1) ise 1.mertebe düşey frekans bileşenini göstermektedir. Diğer tüm katsayılar da matriste durdukları yere göre çeşitli mertebelerden yatay, düşey veya diagonal frekans bileşenlerini belirlerler (Şekil 2.2).

Doğal resimlerde yüksek frekans bileşenleri görece zayıftır. Bu yüzden bu bileşenlere ait katsayılar daha küçük olur. Düşük frekans bileşenleri ise daha büyüktür. Bu yüzden Şekil 2.2'deki DCT çıkış matrisinin eleman değerleri genel olarak sağa ve aşağıya doğru küçülme eğilimi gösterirler. Bu özellikten yararlanmak amacıyla 64 katsayı alternatif bir düzenle dizilir. Söz konusu diziliş tek boyutludur ve temel amacı, katsayıları büyükten küçüğe doğru sıralamaktır. **Zig zag tarama sıralaması** denilen bu dizilişte katsayıların nasıl sıralanacağı Şekil 2.3'te verilmiştir.

DC	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Şekil 2.2: 8*8 DCT çıkış matrisinde her bir katsayının gösterdiği frekans bileşeni.

Sıralama yapıldıktan sonra katsayıların tümünün büyükten küçüğe doğru dizilmiş olacakları kesin değildir. Bununla birlikte, genel eğilim bu yönde gerçekleşmektedir. Ve bu özellik, kodlamada önemli yararlar sağlamaktadır. JPEG standardında da bu sıralama kullanılır.

DC	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Şekil 2.3: Zig zag tarama sırası. DC katsayıdan başlayan tarama, (7,7) katsayısında biter. Aynı dereceden frekans bileşenleri ardarda sıralanırlar.

DCT'nin yanısıra son zamanlarda **Dalgacık Dönüşümü (Wavelet Transform)** üzerinde çalışmalar yoğunlaşmıştır. Dalgacık Dönüşümü, filtre bankaları

oluşturularak resmi istenen şekilde altbantlara ayırır. En önemli özelliği, giriş işaretini sonlu düzlemde istenen çözünürlükte frekans bileşenlerine ayırabilmesidir. Üstelik bunu düzlemdeki sonlu bölgeler için ayrı ayrı yapar*[2]. Birtane ana fonksiyon ile belirlenir. Dönüşüm için kullanılacak diğer fonksiyonlar ana fonksiyonun zamanda ötelenmesi ve ölçeklenmesi (genişletilmesi veya daraltılması) ile belirlenirler. Bu önemli özellikleri nedeniyle Dalgacık Dönüşümü gelecekte adından daha sık bahsettirecektir. Dalgacık Dönüşümü ile ilgili bir kodlama çalışması [4] 'te bulunabilir.

2.3. KUANTALAMA

Kodlanacak dönüşüm katsayıları mutlaka bir kuantalama işleminden geçerler. Kuantalama işlemi resme ait bilginin kısmen kaybedildiği bir aşamadır. Kuantalamadaki amaç, dönüşüm ile elde edilen katsayıların daha verimli bir şekilde kodlanabilecek bir hale getirilmesidir. Resme ait bilginin doğrudan sıkıştırıldığı yöntemler de vardır. Örneğin vektör kuantalama dönüşüm yapılarak ya da yapılmayarak doğrudan resim üzerinde uygulanabilir. Kuantalamada özel olarak dikkat edilmesi gereken konu, kuantalama sonucu resimde oluşacak hataların miktarı ve derecesidir. Bu yüzden insan görme sisteminin özellikleri incelenmelidir. Daha önce belirtildiği gibi, insan gözü yüksek frekansalara karşı daha az duyarlıdır. Bu yüzden DCT gibi, resmi frekans bileşenlerine ayıran bir dönüşümün katsayıları söz konusu olduğunda, yüksek frekans bileşenlerine ait olan katsayılarda daha büyük bir hataya izin verilebilir. Zaten doğal görüntülerde yüksek frekans bileşenleri zayıf olduğundan, etkin bir özellik göstermezler.

Kuantalama genel olarak iki şekilde yapılır:

1- Skalar Kuantalama

2- Vektörel Kuantalama

---

* Fourier Transformunda integralin sınırları eksi ve artı sonsuzdur. Bu durumda işaretin bütün eksen boyunca gösterdiği frekans dağılımı elde edilir. Dalgacık dönüşümü, bunu herhangi bir aralıkta (istenildiği şekilde) gerçekleştirir.

2.3.1 Skalar Kuantalama

Skalar kuantalama, katsayıların tek tek ele alınarak kuantalanmasıyla gerçekleştirilir. Bu işlem temelde iki şekilde olabilir.

Birincisi, katsayıları önceden belirlenen sayılara bölmek yoluyla yapılır. Kodlanmış bilgide **bölüm**ler saklanır. Daha sonra kod çözümünde daha önce **bölen** olarak kullanılan sayılar bu kez **çarpan** olarak kullanılır ve orjinal katsayılar yeniden elde edilir. Fakat hepsi orjinal değerlerinde olmazlar. Çünkü bölme işlemine tamsayı olarak alınan katsayılar yine tam sayı olan **bölen**lere bölündüğünde rasyonel sayılar elde edilir ve bu **bölüm**ler tamsayılara yuvarlanarak saklanır. Bu nokta, kodlama yönteminin kayıplı olduğu noktadır. Yuvarlama hatasından kaynaklanan bu hata, yeniden elde edilen resmin orjinalinden farklı olmasına yol açar. Burada düşünülmesi gereken, hangi katsayının kaça bölüneceğidir. İnsan görme sisteminin özellikleri göz önüne alındığında, yüksek frekans bileşenlerinde daha büyük hatalara izin verilebileceği ortaya çıkar. Bu konuda yapılan çalışmalar sonucu JPEG standardının kullandığı kuantalama matrisi ve elde ediliş koşulları 6.Bölüm'de anlatılmıştır. Bu kuantalama matrisinde her bir DCT katsayısı için bir tane olmak üzere toplam 64 tane **bölen** verilmiştir. Bölme yoluyla yapılan skalar kuantalama örneği Şekil2.6'da verilmiştir.

Orjinal Katsayı	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
Kuantalanmış Değer	-2	-2	-2	-1	-1	-1	-1	0	0	0	1	1	1	1	2	2	2
Yeniden Elde Edilen Katsayı	-8	-8	-8	-4	-4	-4	-4	0	0	0	4	4	4	4	8	8	8

Şekil 2.6: Skalar Kuantalama Örneği. Kuantalayıcı (Bölen) olarak 4 seçilmiştir. Görüldüğü gibi, bölme ve çarpma işlemlerinden sonra, her değer orjinal haliyle elde edilememektedir. Sadece bölenin tam katı olan katsayılar hatasızdır. Bu örnekte, sayılar kuantalamadan sonra tamsayı değerlere yuvarlanmakta ve öyle saklanmaktadırlar. Bazı sistemlerde, yuvarlama yerine bölümün ondalık kısmı atılır ve kuantalanmış değerler öyle saklanır. Bu durumda (-3)'ten (+3)'e kadar tüm sayılar sonuçta sıfır olurlar.

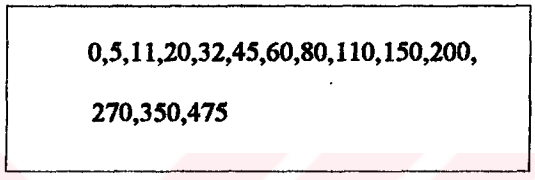
Kuantalamada temel amaç, dönüşümden elde edilen katsayıların kayıpsız kodlama yöntemleriyle sıkıştırmaya daha elverişli hale getirilmesidir. Bir sonraki bölümde incelenecek olan söz konusu sıkıştırma yöntemleri ile sıkıştırma yapmak için giriş bilgisindeki sembollerin istatistik dağılımında bir dengesizlik olması gerekir (Yani bazı sembollerin diğerlerinden daha sık ortaya çıkmaları). Ayrıca sembol sayısının daha az olması da bir avantajdır. Bunlara bağlı bir olumlu özellik ise aynı sembollerin ardarda dizilmeleridir.

Kuantalama yapıldığında, katsayıların oluşturdukları zigzag sıralı dizide bu özellikler ortaya çıkar. Örneğin katsayıları (-1000,1000) aralığında değer alabilen bir katsayı dizisinin değer aralığı, kuantalayıcı olarak 10 kullanıldığı taktirde (-100,100) aralığına düşer. Böylece mesaj dizisindeki sembol sayısı azaltılmış olur. (-1000,1000) aralığındaki 2000 ayrı rakamı belirtmek için 11 bit gerekirken, 200 sembol için 8 bit yeterlidir. Sembol sayısının azaltılması bu yüzden istenir. Şekil 2.6'da görüldüğü gibi, skalar kuantalayıcının giriş çıkış karakteristiği, ardışıl olarak sıralanan sayı gruplarını, aynı değerleri alacak şekilde etkilemektedir. (-5,-4,-3 ve -2'nin -1 olması gibi). Böylece küçültülen değer kümesinde bir sembolden daha fazla sayıda bulunacaktır.

Skalar kuantalamanın JPEG'de de kullanılan en önemli avantajı ise, bir çok katsayının kuantalamadan sonra "0" değerini almasıdır. Özellikle yüksek frekans bileşenleri, doğal resimlerde fazla kuvvetli olmadıklarından, küçüktürler. Göz bu bileşenlerdeki hatalara karşı daha az duyarlı olduğundan, yüksek frekans katsayılarına ait kuantalayıcı sayılar da büyüktür. Bu yüzden özellikle yüksek frekans bileşenlerinin büyük çoğunluğu kuantalamadan sonra sıfır olur. Tabi, bazı düşük frekans bileşenleri de sıfır değerini alırlar. Genel olarak dönüşüm katsayılarında (DCT'de de olduğu gibi) küçük değerler daha sık, büyük değerler ise daha seyrek ortaya çıkmaktadır.

Düzgün Olmayan Skalar Kuantalama (DOSK) (Nonuniform Skalar Quantization) [5] farklı bir yöntemdir. Bu yöntemde bölme yapılmaz. Bunun yerine kuantalanacak katsayıların değer kümesi göz önüne alınarak, bir

kuantalanmış değerler kümesi oluştururulur. Yani hangi katsayı değerinin kaç olarak atanacağı belirlenir. Örneğin (10, 30) aralığındaki tüm katsayılara 20 atanırken (30,90) aralığındaki sayılara 55 atanır(Aralıkların düzgün seçilmesi halinde daha önce anlatılan düzgün skalar kuantalama elde edilir.). Görüldüğü gibi bu sistemde de sembol sayısı azalmakta ve benzerlikler artmaktadır. Ayrıca yine bir çok katsayı sıfıra dönüşebilir. Bunun için örneğin (-10,10) aralığındaki bütün katsayılar sıfır olarak atanabilir. Şekil 2.7'de DOSK için örnek bir kuantalayıcı verilmiştir.



Şekil 2.7: DOSK için örnek bir kuantalayıcı. İşaretin değer kümesi (-600,600) seçilmiştir. Mesaj dizisinde küçük değerli katsayılar daha sık bulunduklarından hata bu bölgede daha küçük tutulur. Büyük değerler için daha büyük hataya izin verilir. Negatif değerler için pozitif olanlarla aynı mutlak değerler kullanılır.

Şekil 2.7'deki örneğe göre [-2,2] aralığındaki sayılar 0 olarak, [3,8] aralığındaki sayılar 5 olarak, [9,15] arası sayılar ise 11 olarak atanacaktır. Burada kural, her katsayının kendine en yakın olan kuantalama değerine atanmasıdır. Kuantalama yapılırken her bir katsayı, kod kitabındaki tüm değerlerle karşılaştırılır ve en yakın olan seçilir.

Şekil 2.7'de verilen kod kitabında toplam 27 sembol vardır(13 pozitif,13 negatif ve sıfır). Kuantalama yapıldıktan sonra bu değerlere verilen endeks numaraları saklanır ve kayıpsız kodlama yöntemleriyle sıkıştırılır (Normal skalar kuantalamadan önemli bir fark buradadır.Saklanan sayılar kuantalama değerleri değil, endeksler.). Dağılımda küçük değerlerin ağır basması için, endeks sıralaması küçükten büyüğe doğru olmalıdır. Hatta negatif değerlerle pozitif değerler içiçe sıralanmalıdır. Örneğin Şekil 2.7'ye göre (0) için endeks numarası yine (0) olarak verilir. (+5) için endeks numarası (1), (-5) için (2), (11) için (3), (-11) için (4) olarak belirlenirse

endeks numaralarının dağılımında da küçük değerler daha sık yer alırlar. Tezdeki DOSK uygulamasında da bu yola baş vurulmuştur.

Anlatılan şekliyle DOSK, normal skalar kuantalamaya göre önemli bir avantaja sahiptir. Bu da gerçek değerler yerine endekslerin saklanmasıdır. Böylece daha küçük bir değer kümesi içindeki sayılar saklanır. Ayrıca çarpma ve bölme işlemlerine ihtiyaç kalmaması sistemin daha hızlı çalışabilmesini sağlar.

DOSK için kullanılacak olan kuantalayıcıyı oluşturmak için, örnek resimlerden alınan istatistiklere dayanan Lloyd-Max Yöntemi[5] kullanılabilir. Söz konusu istatistik bilgi, katsayıların aldıkları değerlerin dağılımlarıdır. Lloyd-Max Yöntemi bu bilgiye dayanarak en az hatayı veren kuantalayıcıyı oluşturur. İteratif bir yöntem olan Lloyd-Max'ın her iterasyonunda hata ya değişmez ya da azalır. İstenen seviyeye ulaşılınca iterasyondan çıkılır ve oluşturulan kuantalayıcı kullanılır. Tablo2.1'de Lloyd-Max Yöntemi'nin uygulanışı adım adım verilmiştir.

Tablo2.1: Lloyd-Max Yöntemi'nin algoritması. Görüldüğü gibi yönteme başlamak için bir başlangıç kod kitabına ve öğretici diziye ihtiyaç vardır. Ayrıca iterasyonun sonlanması için bir eşik değeri belirlenmelidir.

LLOYD-MAX YÖNTEMİ
1- Bir başlangıç kod kitabı ve öğretici dizi seç.
2- Öğretici diziye DOSK uygula
3- Hatayı hesapla. Eşik değerden küçükse iterasyondan çık. Büyükse, kod kitabında aynı kuantalama değerine atanan katsayıların ortalamalarını al ve kod kitabındaki kuantalama değerinin yerine koy,ardından 2'ye dön.

Hata ölçüsü olarak MSE (Mean Squared Error: Ortalama Karesel Hata) kullanılır. N elemandan oluşan bir dizi için MSE'nin ifadesi,

$$\mathrm{MSE} = \frac{\sum_{n=1}^{N}(A_{DOSK} - A_O)^2}{N} \qquad (2.5)$$

şeklindedir. Burada A_{DOSK} DOSK uygulanmış katsayıları, A_O ise orjinal katsayıları göstemektedir.

Öğretici diziler bu tür algoritmaların temel bir öğesidir ve gerekli istatistik bilgiyi oluşturur. Eğer elde edilecek kuantalayıcının evrensel olması isteniyorsa, birbirinden farklı özelliklere sahip, mümkün olduğunca çok resmin kullanılması gerekir. Çünkü bu dizideki katsayılar, başlangıç kod kitabının yerine oluşturulan kod kitabının katsayılarını belirlemektedir.

Lloyd-Max ile bir kod kitabı elde edilmek istendiğinde öncelikle, hangi kod kitabının hangi katsayılar için kullanılacağı belirlenmelidir. Örneğin DCT'de 64 katsayının da aynı kod kitabı ile kuantalanması verimli olmaz. Bunun için katsayıların değerlerinin dağılım fonksiyonları ve aralıkları göz önüne alınarak bir ayrım yapılmalıdır. Hatta her katsayı için ayrı bir kod kitabı da oluşturulabilir ama buna gerek kalmaz. Örneğin zig zag tarama sırasına göre sonlarda yer alan katsayıların büyük çoğunluğu için bir tek kod kitabı yeterlidir. Bunun uygulaması 6.Bölüm'de verilmiştir. Diğer yandan başlangıç kod kitabının nasıl seçileceği de önem taşımaktadır. Yöntemin iyi bir sonuç vermesi için başlangıç kod kitabının da dikkatle belirlenmesi gereklidir. Lloyd-Max sistemi için özellikle başlangıç kod kitabındaki eleman sayısı önemlidir. Yeterli sayıda eleman içermeyen bir başlangıç kod kitabı, beklenen sonucu veremiyebilir.

Diğer önemli bir nokta da Lloyd-Max Yöntemi'nin hata ölçüsü olarak MSE'yi kullanmasıdır. İterasyon sırasında yapılan, başlangıç dizisindeki istatistik bilgilere göre, bu kümeyi temsil eden sayıları en az hata olacak şekilde düzenlemektedir. Yöntemin temelinde gözün görme özellikleri veya kayıplı kodlamada kabul edilebilir

hata oranları ile ilgili hiç bir adım yoktur. Bu yüzden, iterasyondan çıkmak için kullanılacak hata değerini verirken dikkat etmek gerekir. Bir kaç denemeyle istenen kalitede resimleri verecek kuantalayıcı bulunabilir.

2.3.2. Vektör Kuantalama

Vektör Kuantalama (VK), birden çok katsayının birlikte kuantalandığı bir yöntemdir. Skalar kuantalamadan önemli bir farkı, dönüşüm domeni yerine uzaysal domene de uygulanmasıdır (Skalar kuantalama da uzaysal domene uygulanabilir, ama resim kodlamasında bu uygun bir çözüm oluşturmamaktadır.). Yani hiç dönüşüm uygulamadan, doğrudan resim üzerinde VK yapılabilir.

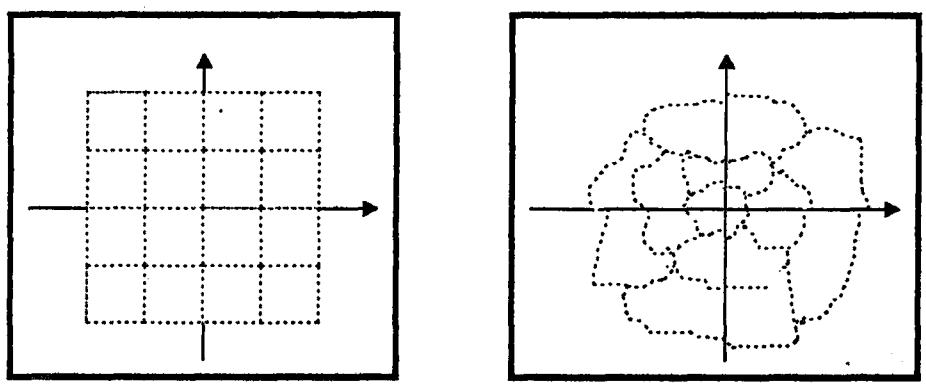
Dönüşümden elde edilen katsayıların VK'ya girmesi için öncelikle belli kıstaslara göre gruplandırılmaları gerekir; çünkü,belirtildiği gibi, bir grup katsayı birlikte kuantalanmaktadır. Bu gruplama için iki temel yol seçilebilir. 8*8 DCT ele alındığında, ilk yol 64 elemanlı katsayı matrisinin belli elemanlarını birleştirmek şeklinde olabilir. Örneğin (1,0),(0,1),(2,0) ve (0,2) bileşenleri bir gruba alınır vb.

Diğer bir yöntem de matris yapısını göz önüne almadan, aynı frekans bileşenlerinin tüm resim için gruplanmasıdır. Farklı bloklara ait tüm (1,0)'ların kendi aralarında gruplar oluşturması gibi. Bu gruplamalar ile oluşturulan vektörler, kodlama için hazırlanan kod kitaplarındaki (vektörlerle aynı boyutlardaki) kod kelimeleri ile karşılaştırılırlar. Vektöre en yakın olan kod kelimesi vektörün yerine atanır ve kod kelimesine ait olan endeks numarası saklanır. Böylece vektörün içerdiği eleman sayısı kadar tamsayı yerine sadece bir tane tamsayı saklanmış olur ve sıkıştırma gerçekleştirilir.

Tabi burada en önemli konu, bu gruplandırmalarla oluşturulacak vektörlerin boyutlarıdır. Vektör boyutu ne kadar büyük olursa, sabit kod kelimesi sayısı için, sıkıştırma oranı da o kadar büyük olur. Buna karşılık kodlanmış bilgiden elde edilecek resmin kalitesi de o kadar düşer. Vektör kodlamada bu dengeyi çok iyi şekilde sağlamak gereklidir ve bu çok zordur. Tezdeki uygulamalarda görülmüştür

ki, endeks numaralarından oluşan dizi kayıpsız kodlama yöntemleriyle çok fazla sıkışmamaktadır. Bu yüzden iyi bir sıkıştırma katsayısı elde etmek için vektör boyutunun büyük tutulması gerekmektedir.

Kaliteyi belirleyen önemli bir faktör de kod kelimelerinin sayısıdır. Özellikle vektör boyu büyük tutulduğunda, resim kalitesinin iyi olması için kod kelimelerinin sayısının fazla olması gerekir. Böylece daha fazla sayıda kombinasyon içerilir. Başka bir deyişle de işaret uzayı daha fazla noktada örneklenmiş olur ve uzaydaki noktalar (yani vektörler) daha az hata ile temsil edilir. Dönüşümlerde, daha önce anlatılan sebeplerden dolayı, küçük değerli katsayılar daha çok olduğundan, işaret uzayı orijine yakın bölgelerde daha çok nokta ile örneklenirken, orijinden uzak noktalarda daha az nokta ile örneklenir. Şekil 2.8.b'de böyle bir örnekleme 2 boyutlu durum için verilmiştir. Şekil 2.8.a'da ise düzgün bir örnekleme gösterilmiştir. İki boyutlu vektörler ele alındığında, her kod kelimesine atanacak olan vektörler, kod kelimesinin etrafında bir alan içinde kalırlar. Çünkü her vektör kendine en yakın kod kelimesine atanmaktadır ve burada MSE ölçü alınmaktadır. MSE ise bu durumda vektörle kod kelimesi arasındaki (vektör boyu kadar boyutlu uzaydaki) mesafe ile doğru orantılı bir büyüklüktür.



Şekil 2.8: VK 'da vektör uzayının bölünmesi. (a)'da düzgün olarak örneklenmiş olan iki boyutlu vektör uzayı gösteriliyor. Her bir karenin merkezinde, o karenin içine düşen vektörlerin atanacağı kod kelimesi bulunmaktadır. (b)'deki örnekte orijine yakın bölge daha sık örneklenmiş. Pratikteki VK uygulamalarında bu yol tutulmalıdır.

VK uygulamalarında kod kelimelerinin oluşumu için, Lloyd-Max Yöntemi'nin geliştirilmesi ile oluşturulan LBG (Linde,Buzo,Gray)Yöntemi [5] kullanılır. LBG için Lloyd-Max'da olduğu gibi bir öğretici dizi ile bir başlangıç kod kitabı gereklidir. Öğretici dizideki vektörler, başlangıç kod kitabındaki kod kelimelerini, kendi istastistik özellikleri çerçevesinde optimize ederler. Bu yüzden seçilecek öğretici dizideki vektörler farklı istatistik özelliklere ait birden çok resmin katsayıları olmalıdır. Bu küme, kodlanacak herhangi bir resmi ne kadar iyi temsil edebilirse, LBG ile elde edilecek olan kod kitabı da o kadar başarılı olur. Tablo 2.2'de LBG algoritmasının adımları verilmiştir.

Tablo2.2: LBG algoritmasının adımları.

LBG ALGORİTMASI
1. Başlangıç kod kitabını ve öğretici diziyi oluştur.
2. Öğretici diziye VK uygula.
3. Hatayı ölç. Eşik değerden küçükse iterasyondan çık. Büyükse, öğretici dizi elemanlarından aynı kod kelimesine atananların ağırlık merkezini(centroid) bul ve söz konusu kod kelimesinin yerine ata; ardından 2'ye dön.

Vektörlerin ağırlık merkezleri (centroid), vektörlerin her bir bileşeni için aritmetik ortalama alınarak bulunur. Vektör sayısı M, boyut sayısı N olmak üzere, M vektörün ağırlık noktası için,

$$c_k = \frac{\sum_{j=1}^{M} v_{jk}}{M} \qquad k=1,2,...,N \qquad (2.6)$$

geçerlidir. Burada c_k ile ağırlık merkezi vektörünün k. bileşeni, v_{jk} ile de j. vektörün k. bileşeni gösterilmektedir.

VK ile ilgili en önemli sorun kod kitabının evrenselliğidir. Birden fazla sayıda katsayı birarada kuantalandığından, bunlar belli varyasyonlar oluşturmaktadırlar ve öğretici küme yetersiz kalırsa, kod kitabı herhangi bir resmi istenen kalitede kuantalayamaz.

Kod kitabının içindeki kod kelimesi sayısı da evrensellik açısından kritik bir konudur. Çünkü daha çok kod kelimesi, daha evrensel bir kod kitabı demektir. Ama kod kelimesi sayısı daha az olursa sıkıştırma oranı yükselir. Ayrıca kodlama süresi ve bellek ihtiyacı da düşer. Diğer yandan daha az kod kelimesi, daha az varyasyon ve daha az örnekleme demek olduğundan kodlanacak resmin kalitesi düşer. Burada yapılması gereken, kod kelimesi sayısı, vektör boyutu ve resim kalitesi arasında uygun bir denge bulmaktır.

2.4. KAYIPSIZ KODLAMA

2.4.1. Enformasyon ve Entropi

Bir mesaj dizisindeki sembollerin taşıdıkları enformasyon miktarı, sembollerin mesaj dizisi içinde bulunma olasılıklarına bağlıdır. Bir sembolün var olma olasılığı ne kadar büyükse, taşıdığı enformasyon o kadar küçüktür. Çünkü bu sembol **beklenen** bir bilgidir. Tam tersine, sembolün gelme olasılığı ne kadar küçükse, o kadar çok bilgi taşır. Bu olguyu açıklamak için tamamı 1 değerli bitlerden oluşan bir mesaj dizisi ele alınsın. Böylece mesaj dizisi, gelme olasılığı %100 yani 1 olan tek bir sembolden oluşmuş olur. Sürekli aynı sembol geldiğinden, gelecekte alınacak mesaj bellidir ve yeni gelen semboller ek bilgi taşımaz. Hatta böyle bir mesaj dizisinin sürekli gönderilmesi bile gerekmez. Burdan çıkan sonuç, olasılığı 1 olan bir sembolün taşıdığı enformasyonun 0 olduğudur.

Enformasyonun diğer bir karakteristik özelliği de, istatistiksel bağımsız iki sembolün toplam enformasyonlarının tek tek enformasyonların toplamı olmasıdır. Çünkü iki sembol tamamen ayrık bilgiler taşımaktadır. Bağımsız iki sembolün birden elde

edilmesi olasılığı ise tek tek olasılıklarının çarpımıdır. Bu yüzden enformasyonu verecek olan fonksiyonun

$$f(p_a * p_b) = f(p_A) + f(p_B) \qquad (2.7)$$

denklemini sağlaması gerekir.

Bütün bu özellikleri taşıyan fonksiyon logaritma fonksiyonudur. Bu yüzden bir mesaj dizisindeki i. sembole ait enformasyon,

$$I = -\log(p_i) \qquad (2.8)$$

ile hesaplanır[6]. Burada p_i, mesaj dizisideki i numaralı sembolün dizi içindeki gerçekleşme olasılığıdır. Sonucun pozitif çıkması için logaritmanın önüne bir eksi işareti konmuştur. Ayrıca genel ifadede logaritmanın önünde bir K çarpanı vardır. Bu çarpan 1 kabul edilir. Logaritmanın tabanı olarak herhangi bir sayı kullanılabilir. Genellikle 2,5 veya 10 kullanılmakla beraber, en önemlisi 2 tabanında olandır. Taban olarak 2 kullanıldığında enformasyonun birimi bit olur. Sonra görüleceği gibi, bu birimle hesaplandığında enformasyon gerçekten de söz konusu bilginin ortalama kaç bitle ifade edilebileceğinin bir ölçüsü olmaktadır. Bu kavramla ilgili olan büyüklük ise **entropi**dir. Entropi bir mesaj dizisinde bulunan tüm sembollerin (bu kümeye **alfabe** de denir.) taşıdığı **ortalama enformasyon**dur. Ortalama enformasyona her sembolün katkısı, gerçekleşme olasılığı ile doğru orantılı olduğundan, her birinin taşıdığı enformasyon, olasılığı ile çarpılır ve tüm değerler toplanır. Entropinin ifadesi,

$$H = -\sum_{i=1}^{M} p_i * \log_2(p_i) \qquad (2.9)$$

şeklindedir. Burada "M" alfabedeki toplam sembol sayısıdır.

Kayıpsız kodlamada temel yaklaşım, bütün sembollerin eşit sayıda bitle gösterilmeleri yerine, daha sık kullanılan sembollere daha kısa bit dizileri, daha seyrek kullanılanlara ise daha uzun bit dizileri karşı düşürmektir. Böylece aynı bilgi daha az bitle ifade edilmiş olur. Mors alfabesi, bunun çok eski bir uygulamasıdır. İngilizce'de en sık kullanılan harf olan "E" harfi en kısa koda sahiptir.

Bu temele dayanan sistemlerde en önemli kavram hiç kuşkusuz **ortalama kod kelimesi uzunluğu**dur. Ortalama kod kelimesi uzunluğu, kodlama sonrasında ortalama olarak her bir sembole kaç bit düştüğünü gösterir. İfadesi,

$$\bar{n} = \sum_{i=1}^{M} n_i p_i \qquad (2.10)$$

Burada n_i, i numaralı sembole verilmiş olan bit dizisinin uzunluğudur. Sonuçta ortalama kod kelimesi uzunluğu ne kadar küçükse, kodlama o kadar verimlidir ve sıkıştırma oranı o kadar yüksektir. Diğer yandan ispatlanmıştır ki [6], ortalama kod kelimesi uzunluğu, alfabedeki sembollerin entropisinden küçük olamaz, en iyi ihtimalle eşit olabilir. Bu yüzden, entropi aslında sembollerin ortalama olarak en az kaç bitle ifade edilip, kodlanabileceğini veren bir büyüklüktür.**Huffman Kodlaması** ise bu şartlar altında optimum kod kuralını oluşturan bir yöntemdir.

2.4.2. Huffman Kodlaması

Huffman Kodlaması, kaynağa ait tüm bilgilerin, yani sembollerin ve olasılıklarının bilindiği durumlarda kullanılabilir. Semboller, olasılıklarına göre büyükten küçüğe doğru dizilirler ve en alttaki ikili birleştirilir. Bu birleştirmede olasılıkları toplanır ve büyükten küçüğe sıralama yeniden yapılır. Sadece iki sembol kalana kadar bu işlem yinelenir ve bu ikiliden birine 1, diğerine 0 verilir. Ardından, tam ters sırada ayrışım yapılır ve her ayrımda yeniden serbest kalan iki sembolden birine 1, diğerine 0 eklenir. Aşağıdaki örnekte bu adımlar görülmektedir.

Mesaj dizisindeki semboller ve olasılıkları şöyle olsun:

1. Sembol : 0.35
2. Sembol : 0.25
3. Sembol : 0.20
4. Sembol : 0.15
5. Sembol : 0.05

Birleştirme uygulanırsa:

1. 0.35		**345.** 0.40		**12.** 0.60	⇒	**0**
2. 0.25	⇒	**1.** 0.35	⇒	**345.** 0.40	⇒	**1**
3. 0.20		**2.** 0.25				
45: 0.20						

Ayrıştırma aşaması:

12. 0.60 ⇒ 0	**345.** 1		**1.** 00		**1.** 00
345. 0.40 ⇒ 1	**1.** 00	⇒	**2.** 01	⇒	**2.** 01
	2. 01		**3.** 10		**3.** 10
			45. 11		**4.** 110
					5. 111

Görüldüğü gibi Huffman Kodlamasıyla her sembol için 3 bit (toplam 5 ayrı sembol olduğundan) kullanılmak yerine, ilk üç sembol için 2 bit kullanılmıştır.

Örnekte verilen kaynağın entropisi,

$H= -0.35\log(0.35) - 0.25\log(0.25) - 0.20\log(0.20) -0.15\log(0.15) -0.05\log(0.05)=2.12$

bulunur. Kodlama sonucu ortaya çıkan ortalama kod kelimesi uzunluğu ise,

$\bar{n} = 0.35*2+0.25*2+0.20*2+0.15*3+0.05*3=2.2$

değerindedir. Görüldüğü gibi sonuç optimum değere çok yakın olmakla birlikte, çok az daha büyük çıkmıştır. Bunun sebebi ise, her bir sembole tamsayı olan bir değerde bit karşı düşürülmesidir. Ondalıklı sayıda bit karşı düşürülmesi mümkün olsaydı, optimum sonuca ulaşılabilirdi.

Huffman Kodlaması, kodlama sistemlerinin sahip olması gereken çok önemli bir koşulu da sağlar. Kodlanmış bilgiden orjinal bilginin elde edilebilmesi için kodun **tek bir şekilde çözülebilmesi** gerekir. Örneğin bir sembole ait bit dizisi (01), diğerine ait olanı (010) olsun. Alınan mesaj dizisi (010101...) şeklinde olursa, kod çözümünde ilk üç bit için tek türlü çözüm mümkün değildir. Çünkü öncelikle ilk iki bitin alınıp, birinci sembolün geldiğine karar vermek mümkün olduğu gibi, ilk üç biti birden ele alıp ikinci sembolün geldiğine karar vermek de mümkündür. Bu sorunun oluşmaması için, hiç bir sembole ait bit dizisinin, diğer bir sembole ait bit dizisinin başlangıç kısmına eşit olmaması gereklidir. Verilen örnekte birinci sembole ait bit dizisi, ikincinin ilk iki bitine eşittir ve sorun çıkarır. Huffman Kodlaması, yapısı gereği bu şarta uymaktadır.

Genel olarak bakıldığında, Huffman Kodlaması'nın verimli olması için iki koşul vardır. Birincisi, sembollerin bir kısmının diğerlerinden daha sık ortaya çıkması; ikincisi ise sembol sayısının mümkün olduğunca az olması. Sembol sayısı ne kadar az olursa, ayrıştırma aşaması o kadar az adımlı olur ve sembollere atanan kodlar o kadar kısa olur.

Bir dizi Huffman Kodlaması ile kodlandığında daha az yer tutacağının garantisi yoktur. Özellikle istatistik dağılım ile ilgili koşulu sağlamayan diziler Huffman Kodlaması ile sıkıştırılamazlar. Hatta bazı diziler daha fazla yer kaplayabilirler. Sıkıştırmanın sağlanması halinde de, bunun miktarı değişkendir. Sıkıştırma oranı tamamen giriş bilgisinin entropisine bağlıdır. Ama uygun diziler için **ortalama olarak** 2 kat sıkıştırma sağlanabildiği söylenebilir.

2.4.3. Akış Uzunluklu Kodlama

Akış Uzunluklu Kodlama (AUK), genel anlamda entropi ve benzeri kavramlarla hiç bir ilişkisi olmayan bir kodlama tarzıdır. Temeli itibarıyla herhangi bir teoriye de dayanmaz. Çok basit bir mantıkla çalışan ve özel bir durumda işe yarayan bir kodlama tarzıdır.

Temeli, mesaj dizisinde bir sembolün pek çok kere ardarda tekrarlanmasıdır. Eğer bu durum geçerliyse, ardarda olan semboller sayılır. Kodlanmış dosyada, önce tekrarlanma adedi, ardından da tekrarlanan sembolün hangisi olduğuna dair bilgi saklanır. Ardından bir sonraki sembolün ardarda kaç defa tekrarlandığı sayılır ve buna ait bilgi kaydedilir. AUK örneği Tablo 2.3'te verilmiştir.

Tablo 2.3: Akış Uzunluklu Kodlama örneği. En alttaki satırda mesaj dizisinin AUK ile kodlanmış hali görülmektedir. Mesaj dizisinde 14 sembol varken, kodlanmış bilgide sadece 10 sembol bulunmaktadır.

AKIŞ UZUNLUKLU KODLAMA															
Mesaj Dizisi	5	5	5	8	8	6	9	9	9	9	9	1	1	1
Adet	3			2		1	5					3		
Sembol	5			8		6	9					1		
KOD	3		5	2	8	1	6	5			9	3		1

Eğer, ardarda tekrarlanan sembol sayısı yeterince fazlaysa, AUK ile çok yüksek oranlarda sıkıştırma elde edilir. Bu oran 2'nin çok üstünde olabilir. Diğer yandan, tekrarlanmalar yeterince sık ve uzun değilse, kodlanmış dizinin boyu orjinalinden daha büyük olabilir. Zaten her kayıpsız kodlama yöntemi için bu durum geçerlidir. Hiç bir kayıpsız sıkıştırma yöntemi, her giriş için daha küçük bir çıkış dizisi oluşturamaz. Dikkat edilirse, AUK'da her sembol dizisi için 2 sembol gelmektedir. Bu durumda sıkıştırmanın gerçekleştirilebilmesi için, sembollerin çoğunun en az 3 defa ardarda sıralanması gerekmektedir.

JPEG'deki kullanımında, AUK sadece ardarda gelen sıfırları sayar. Çünkü diğer değerlerin hiç birisi, sık sık ardarda dizilmezler. Skalar kuantalama sonrasında zig zag tarama sırasına göre birbirini takip eden bir çok sıfır ortaya çıkar.

2.5. RESİM KODLAMADA KULLANILAN HATA ÖLÇÜLERİ

Kodlanmış bilgiden yararlanarak yeniden elde edilen resimler, kodlama kayıplı olduğunda, orjinal resmin tamamen aynı olmadığından, bir hata oluşur. Bu hatayı ölçmek için kullanılan iki temel hata ölçüsü MAE (Mean Absolute Error : Ortalama Mutlak Hata) ve PSNR'dır (Peak Signal to Noise Ratio : Tepe İşaret Gürültü Oranı).

MAE, tüm piksellere ait mutlak hatanın ortalamasıdır. İfadesi,

$$MAE = \frac{\sum_{j=1}^{N} |o_j - k_j|}{N} \qquad (2.11)$$

şeklindedir. Burada N resimdeki piksel sayısı, o_j orjinal resimdeki j. pikselin parlaklığı, k_j ise kodlanmış resimdeki j. pikselin parlaklığıdır.

PSNR ise normal işaret gürültü oranından biraz daha farklıdır. İşaret gürültü oranı, işaretin sahip olduğu güce bağlı bir büyüklüktür. Daha nesnel bir ölçü olması amacıyla, PSNR için işaretin gücü maksimum sayılır. Yani 8 bitlik siyah beyaz bir resim için tüm piksellerin parlaklığı 255 alınır ve bu şekilde işaret gürültü oranı bulunur. Buna göre,

$$PSNR = 10\log \frac{255^2}{\frac{1}{N}\sum_{j=1}^{N}(o_j - k_j)^2} \qquad (2.12)$$

ile hesaplanır. PSNR'ın birimi de dB'dir.

Örneği verilen nesnel hata değerleri, resimlerin kaliteleri hakkında önemli fikir vermekle beraber, herhangi bir yöntem ile elde edilen resimlerin kaliteleri hakkında fikir sahibi olabilmek veya farklı resimleri kalite yönünden karşılaştırmak açısından mutlak ölçü olarak kullanılamazlar. Bu ölçülerin yanı sıra mutlaka öznel bir

değerlendirme de yapılmalıdır ki, bu da resimlere bakılarak ortaya çıkan hataların belirlenmesidir. Nitekim, temel amaç gözü rahatsız etmeyecek kalitede resimler elde etmektir.

Özellikle **farklı yöntemler kullanıldığında** ortaya çıkan MAE değerleri tek başına yöntemleri karşılaştırmak için yetersiz kalır. MAE'si daha büyük olan bir resim, daha az gözle görülür hataya sahip olabilir. Ancak aynı yöntem farklı paramatrelerle uygulanıyorsa, bu durumda daha büyük MAE daha kötü bir resim anlamına gelir.

Örneğin bir resimde bütün parlaklık değerleri 5 arttırılsa, MAE 5 olur. Bu hata değeri tezdeki uygulamalarda elde edilen MAE değerlerinin hemen hepsinden daha büyüktür. Bununla birlikte böyle bir resimde gözle görülür hiç bir hata olmaz. Halbuki MAE değeri 2.5 olan bir resimde ciddi hatalar görülebilir.

BÖLÜM 3
KASKAD DCT

3.1. AMAÇ VE YÖNTEM

DCT'ye dayanan ilk uygulama , ikinci bölümde verilen üç aşamalı görüntü kodlama yönteminin sadece birinci aşamasını ilgilendirmektedir. Bu aşama resim bilgisine uygulanan dönüşümdür. Dönüşümden elde edilen katsayıların amaca en uygun şekilde kuantalanması ve entropi kodlama yöntemlerinin uygulanması aşamasından önce, söz konusu katsayıları daha verimli bir şekilde temsil etmek amaçlanmıştır.

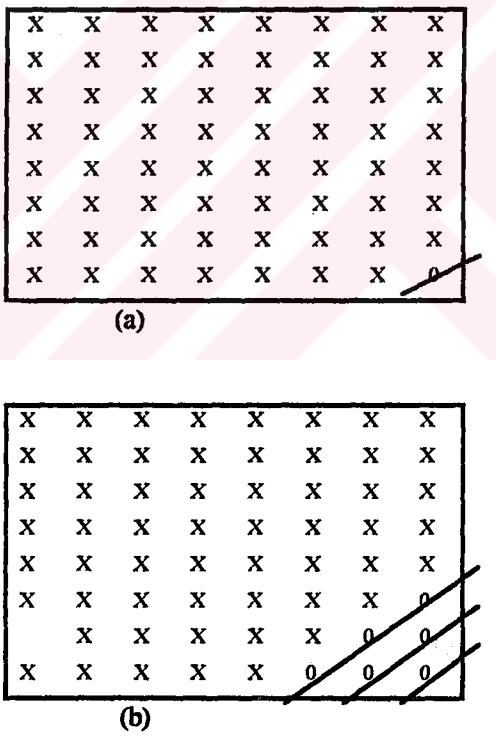
Bunun için seçilen yol , değer kümesi [-1024,1023] olan DCT katsayılarını yeniden [0,255] aralığına öteleyerek ikinci defa DCT uygulanmasıdır.

Kullanılan tüm resimler 256*256 boyutlarında olduğundan ve DCT ikinci bölümde belirtildiği gibi 8*8'lik bloklara uygulandığından , 1024 tane blok oluşmakta ve aynı frekans bileşenine ait (zigzag tarama sıralamasında aynı endekse sahip) her bir katsayıdan 1024 tane elde edilmektedir. Aynı endeksli katsayılar resimde durdukları sıra korunacak şekilde bir araya getirildiklerinde 32*32'lik bloklar oluşturmaktadırlar. Bu bloklar da (64 adet farklı frekans bileşeni olduğundan) 8*8'lik bir düzenlemeyle dizildiklerinde 1.DCT çıkışından elde edilmiş olan katsayılar altbandlara ayrılmış bir şekilde 256*256'lık bir matris oluştururlar. Katsayıların değer kümesi göz önüne alındığında, bu değer kümesini [0,255] aralığına taşımak için katsayıları 1024 ile toplayıp 8'e bölmek gereklidir. Bu şekilde 1.DCT ile elde edilen katsayılar ikinci bir DCT için uygun hale gelmiş olurlar.

Önceki bölümde belirtildiği gibi, doğal resimler az miktarda yüksek frekans bileşenleri içerdiğinden, yüksek frekansa karşılık düşen katsayılar oldukça küçüktür. Bu yüzden kuantalamadan sonra çoğu sıfır değerini alır (Tabii, bu durum vektör

kuantalaması için değil, skalar kuantalama yöntemleri için geçerlidir.). Sıfır değerini almayan katsayılarda da belli bir miktarda hata oluşur. DCT sonrası kuantalama işleminin bu etkilerini görmek için en pratik yöntemlerden biri, seçilen belirli katsayıları sıfır yapmaktır. Böylece sıfır olan katsayıların sebep olacağı hatalar gözlenebilir.

Bu amaçla şu şekilde bir sıfır atama işlemi tanımlanmıştır. Sıfır atama işlemi sağ alt köşeden başlamak üzere çapraz olarak yukarı doğru aşamalarla yapılacaktır. Öyle ki, birinci aşama sağ alt köşedeki elemanın (Şekil 3.1.a), üçüncü aşama sağ alt köşede kalan altı elemanın (Şekil 3.1.b) sıfır yapılmasıdır.



Şekil 3.1 Sıfır Atama örnekleri. (a) 1. aşama. (b) 3.Aşama

Bu sıfır atama işlemi kullanılan iki resme uygulandığında elde edilen sonuçlar göstermiştir ki, sağ alt üçgenin tamamen sıfır yapılması demek olan 8. aşama sıfır atama işleminden sonra bile resimlerde gözle görülür hiç bir hata olmamaktadır

(**NOT:**Doğal görüntülerin çok büyük bir kısmı için bu geçerlidir.). Bu resimler için elde edilen MAE ve PSNR değerleri Tablo 3.1'de verilmiştir.

Tablo 3.1: 8. Aşama sıfır atama işlemi uygulanan resimlerin hata değerleri

	MAE	PSNR (dB)
SALESMAN	2.58	36.12
TIFFANY	2.68	35.02

Böylece, 65536 pikselden oluşan resmin DCT çıkışından elde edilen 65536 DCT katsayısından sadece (28 frekans bileşeni kullanıldığından) 28*1024=28672 tanesi ile resim kaliteli bir şekilde elde edilir.

DCT katsayılarına ikinci bir DCT uygulanmasındaki amaç bu küçülmeyi ikinci bir defa daha gerçekleştirmek ve hatta mümkün olması halinde üçüncü, dördüncü vs. adımlarla ilerletmekti. Sistemin başarılı olması halinde ikinci DCT sadece söz konusu 28768 katsayının oluşturduğu bloklara uygulanabilir ve bu sayıların daha küçük elemanlı bir küme tarafından temsili gerçekleşebilirdi.

Kaskad DCT'nin ikinci bir avantajı daha söz konusu olabilir. DCT katsayıları sayısal olarak ölçeklendikten (1024 ile toplanıp 8'e bölündükten) sonra 128'in etrafında dağılan sayılara dönüşmektedirler. Düşük frekans katsayılarının dağılımı göreceli daha geniş olmakla birlikte, yüksek frekans bileşenlerinin dağılımı çok dar bir bölgede kalmaktadır. Katsayıların büyük çoğunluğu 128±10 gibi bir aralıkta dağılım göstermektedirler. Bu da, yeniden dizilmiş ve ölçeklenmiş katsayılar arasında, orjinal resimdekine göre daha yüksek bir korelasyon olacağını gösterir. Korelasyonun daha yüksek olması ise elde edilen matrisin daha iyi sıkıştırılabilir olduğunu gösterir. Yani ikinci DCT çıkışı aynı şartlarda kuantalandığında, birinciye göre daha avantajlı olmalıdır.(Örneğin daha çok katsayı sıfır olmalıdır.)

3.2. UYGULAMA VE SONUÇLAR

Kaskad DCT yöntemi iki resimde uygulandı. Önceki bölümde belirtildiği gibi ilk DCT 'de 8. aşama sıfir atama işlemi kaliteli bir resim verdiğinden, birinci DCT'den sonra söz konusu sıfir atama işlemi uygulandı. Ardından katsayılar [0,255] aralığına ötelenerek yeniden DCT alındı. Bu aşamada üç ayrı şekilde deneme yapılarak kaskad DCT ve sıfir atama işleminin resim bilgisine etkileri araştırıldı.

a) İkinci DCT katsayıları hiç değiştirilmeden orjinal resme geri dönüldü ve böylece ölçekleme ve ikinci DCT'den kaynaklanan hata belirlendi.

b) İkinci DCT'den elde edilen katsayılara 4. aşama sıfir atama işlemi uygulandı ve resim elde edildi.

c) İkinci DCT katsayılarına 8.aşama sıfir atama işlemi uygulandı ve resim elde edildi.

Bu denemelerden elde edilen resimlerle ilgili MAE ve PSNR değerleri Tablo3.2'de verilmiştir.(Resim isimlerinin yanındaki rakamlardan ilk ikisi 1.DCT sonrası uygulanan sıfir atama işleminin, ikinci ikili ise 2.DCT sonrası uygulanan sıfir atama işleminin kaçıncı aşama olduğunu göstermektedir. 00:Sıfir atama yapılmadı.)

Tablo3.2: Kaskad DCT uygulanmış resimlerin MAE ve PSNR değerleri

	MAE	PSNR(dB)
SALESMAN0800	3.66	33.91
SALESMAN0804	5.83	30.12
SALESMAN0808	10.80	24.77
TIFFANY0800	3.63	33.38
TIFFANY0804	5.32	30.28
TIFFANY0808	8.34	26.46

Görüldüğü gibi ikinci DCT'den sonra sıfir atama işlemi yapılmadan geri dönüldüğünde MAE yaklaşık 1 artamakta ve PSNR ise 2dB kadar düşmektedir. Bu

hatalar her iki DCT hesaplanırken ortaya çıkan yuvarlama hataları ve iki DCT arasındaki ölçekleme sırasında 8'e bölünen katsayılardaki kuantalama hatalarından kaynaklanmaktadır. Söz konusu hata değerleri, resimlerin kalitesinin iyi olduğunu göstermektedir ve resimlerde gerçekten de gözle görülür rahatsız edici hatalar yoktur. Bu da DCT katsayılarına ikinci bir DCT uygulanmasının resim kalitesi açısından bir engel olmadığını ortaya koyar.

Fakat ikinci DCT katsayılarının bir kısmı sıfır yapıldığında durum çok farklıdır. Dördüncü ve sekizinci aşama sıfır atama işlemlerinde resim kalitesi hızla düşmektedir. Üstelik resimdeki gözle görülür hatalar açısından bakıldığında, hata MAE ve PSNR değerlerindeki bozulmanın çağrıştıracağından çok daha büyüktür. Dördüncü aşama sıfır atama işlemi yapıldığında resimde bir **bulanıklaşma** görülmektedir. Daha da ilginci, Tiffany'de burun deliklerinin sayısı beşe çıkmaktadır. İki burun deliğinin tam ortasında bir adet ve yanlarında da birer tane olmak üzere üç adet burun deliği resme eklenmiştir. Benzer şekilde saçlar, parmaklar gibi diğer öğelerin de (hayalet-gölge) benzeri bir şekilde kopyalandığı açıkça görülmektedir. Salesman'de ise örneğin arka plandaki raflardaki kitapların kopya görüntüleri **üst taraflarında** oluşmaktadır.

Bu etki sekizinci aşama sıfır atama işlemi uygulanan resimlerde çok daha güçlü bir şekilde ortaya çıkmakta ve resim kalitesini çok daha fazla düşürmektedir.

Yapılan uygulama sonucu resimdeki şekillerin kopyalanmaları hem ilginç hem de beklenmeyen bir sonuçtur. Bunun sebebini anlamak için yapılan işlemleri gözden geçirmek gerekir. Yapılan sıfır atama işleminin temel etkisi bir alçak geçiren filtreninki gibidir. Sayısal sistemlerde uygulanan alçak geçiren filtrelerin pratikteki etkisi de işaretteki komşu değerleri birbirine yaklaştırmaktır. 1. DCT sonrası uygulanan ölçekleme işleminde sekize bölme yapıldığından katsayılar arasındaki farklar zaten bir miktar düşer. Daha sonra ikinci DCT'ye bir görüntü imiş gibi sokulan bu katsayılar, sıfır atama işlemi de yapıldığından alçak geçiren bir filtreleme işlemine tabi tutulmuş olurlar. Bunun sonucu olarak katsayılar birbirlerine değerce daha da yaklaşırlar ve hatta sıfır atama işleminin seviyesine göre eşitlenebilirler.

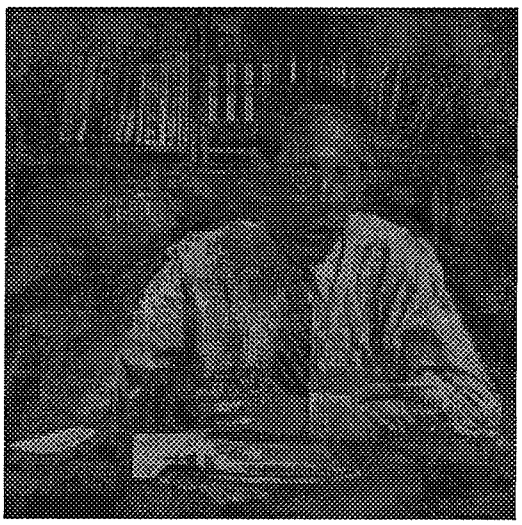
İkinci DCT yapılmadan önce farklı 8*8'lik bloklara ait aynı frekans bileşeni katsayılarının 32*32'lik bloklar oluşturacak şekilde dizildiği ve bunun resimdeki diziliş ile aynı şekilde yapıldığı düşünülürse şu ortaya çıkar. İkinci DCT'ye yanyana, yani komşu piksel olarak giren katsayılar, orjinal resimde komşu olan 8*8'lik blokların aynı frekans bileşenine ait olan katsayılardır. Bunlar yukarda anlatıldığı gibi bir alçak geçiren filtre ile benzeştirilir veya eşitlenirse, komşu 8*8'lik resim bloklarına ait 8*8'lik DCT matrisleri birbirine çok benzeyecektir. Ters DCT alındığında da bu iki bloğun benzer desenler içermesi beklenir. İşte bu yüzden uygulanan sistem resimlerdeki şekillerin blok boyutunun tam katına kopyalanmasına sebep olmaktadır.

Bu savın doğruluğunu test etmek için yapılacak şey, oluşan benzerliklerin arasında kaç piksel mesafe olduğuna bakmaktır. Eğer sav doğruysa bu sayı sekiz olmalıdır, çünkü desenler komşu 8*8'lik bloklarda ortaya çıkmalıdır. Resimlerde bu özellik kontrol edildiğinde aradaki uzaklığın gerçekten de sekiz piksel olduğu görülmüştür.

Sonuç olarak bir resme kaskad DCT uygulanmasının kalite açısından bir sorun oluşturmadığı açıktır. Fakat ikinci seviye DCT katsayılarına kuantalanma uygulamanın mümkün olmadığı ortaya çıkmıştır. Dolayısıyla, ikinci DCT'den sonra elde edilecek katsayıların tamamının hiç bir kayıp olmadan saklanması gerekir ki, bu da bir resim sıkıştırma uygulaması için istenmeyecek verimsiz bir durumdur. Şekil 3.2. ve Şekil 3.3'te, kullanılan ve elde edilen resimlerden örnekler verilmiştir.

Şekil 3.2.a: Tiffany0804. Birinci DCT sonrası 8. aşama, ikinci DCT sonrası 4. aşama sıfır atama işlemi yapılmış resim.

Şekil 3.2.b: Tiffany0808. Her iki DCT sonrası 8. aşama sıfır atama işlemi yapılmış resim.



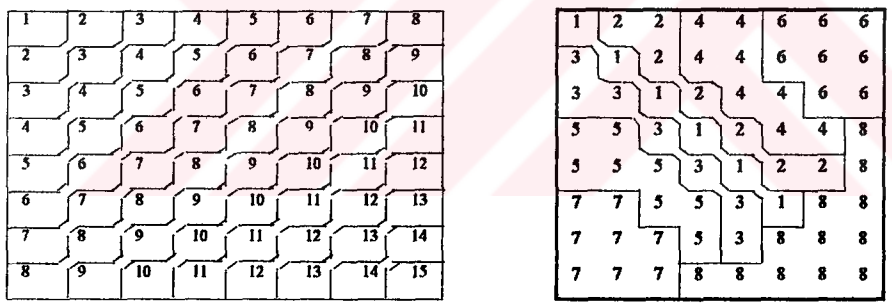
Şekil 3.3.a: Salesman0804. Birinci DCT'den sonra 8.aşama, ikinci DCT'den sonra 4. aşama sıfır atama işlemi yapılmış resim.

Şekil 3.3.b: Salesman0808. Her iki DCT'den sonra 8. aşama sıfır atama işlemi yapılmış resim.

BÖLÜM 4
VEKTÖR KUANTALAMA

4.1 TEMEL AMAÇ VE YÖNTEM

DCT tabanlı vektör kuantalama uygulamalarında genelde seçilen yol vektörlerin, çıkış matrisinde komşu olan katsayıların belli kıstaslara göre gruplanmasıyla oluşturulması şeklindedir. Bu durum gerek 8*8, gerekse 4*4 DCT uygulamaları için geçerlidir. Şekil 4.1'de buna ait iki örnek verilmektedir[5].



(a) (b)

Şekil 4.1. Vektör Kuantalamada DCT katsayılarının vektör oluşturmaları

4.1.a'da frekans derecelerine göre yapılan klasik gruplama (b)'de ise farklı bir yöntem görülmektedir.

Sayısal yöntemlerle kodlanmış görüntülerin yeniden ekrana yansıtılmasında iki temel yöntem kullanılmaktadır. Bunlardan birincisi **ardışıl (sequantial)** yöntemdir. Bu yöntemde resim yukardan aşağıya olmak üzere satır satır ekrana çizilir. Diğer yöntemde ise resmin tamamı ilk aşamada kabaca çizilir. İkinci aşamada ayrıntılar

eklenmeye başlar. Ve resim tüm detayıyla birkaç aşamada elde edilir. Bu yönteme **aşamalı yöntem (progressive)** denir. Yöntem spektral domende veya resim domeninde uygulanabilir. Bu yöntem özellikle veritabanı uygulamalarında kullanılan, ayrıca farklı çözünürlük ve kalitelerde yayın yapmak amacına uygun olan bir kodlama tarzıdır. Eğer DCT çıkışındaki her bir frekans bileşeni aynı DCT matrisindeki diğer katsayılardan tamamen ayrık olarak kodlanırsa, 8*8'lik görüntü bloklarının kaliteleri ve nihai çözünürlük kod çözümü sırasında istenildiği gibi belirlenebilir.

Bu kodlama tarzına uyumlu olması açısından tezdeki vektör kuantalama uygulamalarında her bir frekans bileşeni ayrı olarak ele alınmıştır. 64 bileşenin her biri diğerlerinden bağımsız olarak kendi aralarında gruplanmış ve uygun kod kitapları elde edilmesi ile kod çözümü sırasında sadece istenen bileşenlerin kullanılması hedeflenmiştir.

AC bileşenlerin içinde, resim bilgisi için en önemli olan frekans bileşenleri (1,0) ve (0,1) bileşenleridir. Çünkü bunlarda oluşacak hatalar resmi diğer AC katsayılarda oluşacak hatalara göre daha çok etkilemektedir. Diğer yandan bu katsayıların değer kümeleri hemen hemen aynıdır. Bu yüzden vektör kuantalama uygulamalarına bu iki katsayıdan başlanması ve bu iki katsayı için ortak bir kod kitapçığı bulunması öngörülmüştür. İki katsayı için ortak bir kod kitapçığı kullanılmasına rağmen ayrı kodlanacak olmalarına dikkat edilmelidir. Ortak kod kitabı kullanmaktaki amaç, elde edilecek yöntemle oluşturulması söz konusu olacak bir vektör kuantalayıcı sistemin bellek ihtiyacını azaltmaktır. Her bileşen için ayrı kod kitapçığı kullanılması durumunda, her biri 256 vektör içermesi hedeflenen(her bir vektörün 1 bayt ile gösterilebilmesi için 256 seçilmiştir.) kodlayıcının 64*256 tane vektörü içerecek bir bellek kapasitesine sahip olması gerekecektir ki bu, sistemin maliyeti açısından olumsuz bir faktördür.

Kod kitaplarının oluşturulmasında LBG yöntemi kullanılmıştır. LBG yöntemi için gerekli başlangıç kod kitapları özel bir yolla elde edilmiştir. Bu yol aslında LBG algoritmasının üçüncü adımına benzemektedir. Öğretici kümede bulunan vektörlere

önceden seçilen eşik değerlerinden daha yakın olan komşu vektörler seçilmekte(Vektörler arasındaki uzaklık, vektörlerin sahip olduğu bileşen sayısı kadar boyutlu uzaydaki mesafedir. Örneğin ilk denemede 2*2'lik vektörler kullanılmıştır ve vektörler arası uzaklık 4 boyutlu uzaydaki mesafe olarak hesaplanmıştır.), oluşturulan bu grupların ağırlık merkezleri (centroid) hesaplanmakta ve bir kod kelimesi olarak seçilmektedir. İlk iterasyonda herhangi bir komşuluk grubuna giremeyen vektörler ise daha büyük bir eşikle ikinci iterasyona girmektedirler. Bu işlem gittikçe büyüyen eşik değerleri için tekrarlanmaktadır. Eşik değerlerinin büyüklüklerine göre makul bir iterasyon sayısı seçilmekte ve tüm iterasyonların sonunda hala bir gruba girememiş vektörler de başlangıç kod kitabına katılmaktadır. Bu şekilde elde edilen kod kitapları ise LBG ile optimize edilmektedir.

4.2. UYGULAMA VE SONUÇLAR

4.2.1. Tek Kod Kitabı ile Kodlama

İlk vektör kuantalama uygulamasında (1,0) ve (0,1) frekans bileşenleri, ait oldukları 8*8 DCT bloklarının, 256*256 boyutlarındaki resimde durdukları sırayla 32*32'lik birer matrise dizilmişlerdir. Buradaki diziliş 3.Bölüm'de Kaskad DCT uygulamasındaki ile aynıdır. Bu şekilde sıralanan katsayılara 2*2 boyutlarında vektörler kullanılarak vektör kuantalama uygulanmıştır. 2*2'lik boyutların kullanılmasının sebebi genelde vektör kuantalama uygulamalarında 2*2 veya 4*4 gibi karesel bölgelerin kullanılmasıdır. Sistemin başarılı olması durumunda 4*4'lük vektörlerle kuantalama yapılması da planlanmıştır.

Öğretici dizi olarak 4 resmin ilgili DCT katsayıları kullanılmıştır. Bunlar Tiffany, Lena, Baboon ve Salesman'dir. Bu resimlerin (0,1) ve (1,0) frekans bileşenleri yukarıda anlatılan şekilde bir başlangıç kod kitabı oluşturmak için kullanılmış ve bu kod kitabına daha sonra aynı resimlere ait katsayılarla LBG algoritması uygulanmıştır.

Farklı başlangıç kod kitabı elde etmenin tek yolu, birbirine yeterince yakın seçilecek olan vektörler için kullanılan **yakınlık eşiklerini** değiştirmektir. Her seferinde maksimum 7 eşik değeri kullanılarak çeşitli kod kitapları oluşturulmuştur. Burada temel hedef 256 veya daha az sayıda kod kelimesi içeren ve iyi resim kalitesi sağlayan kod kitaplarının elde edilmesidir. Bu amaçla, uygun eleman sayısını (256) verebilecek bir kod kitabı bulunana kadar, 11 ayrı başlangıç kod kümesi oluşturulmuştur. Örnek olarak iki kod kitabı ile ilgili eşik değerleri ve her eşik değeri için elde edilen kod kelimesi sayısı Tablo 4.1'de verilmiştir. Burada Eşik-1, Eşik-2 şeklinde verilen değerler uygulama sırasına göre, vektörlerin birbirlerine yeterince yakın olup olmadığına karar vermek için kullanılan eşik uzaklık (4 boyutlu uzayda) değerleridir. Her biri için kaçar tane vektör grubu bulunduğu da verilmiştir. **Yalnız vektörler** olarak verilen grup ise, diğer hiç bir vektöre eşik değerlerinden daha yakın olmadığı için hiç bir komşuluk grubuna girememiş vektörleri içermektedir ve bunlar oldukları gibi başlangıç kod kitabına dahil edilmişlerdir.

Tablo 4.1.a: Örnek bir başlangıç kod kitabının eşik değerleri ve kod kelimesi adetleri

EŞİK DEĞERLERİ	**BULUNAN KOD KELİMESİ ADEDİ**
Eşik-1 =15	183
Eşik-2 =25	176
Eşik-3 =35	109
Eşik-4 =50	99
Eşik-5 =75	66
Eşik-6 =100	28
Eşik-7=150	18
Yalnız Vektörler	36
Toplam Vektör Sayısı	**715**

Tablo 4.1.b: Örnek ikinci bir başlangıç kod kitabına ilişkin eşik değerleri ve kod kelimesi sayıları.(b)'de eşikler daha yüksek tutulduğu için, daha uzak vektörler birleştirilmiş ve daha küçük bir kod kitabı elde edilmiştir

EŞİK DEĞERLERİ	**BULUNAN KOD KELİMESİ ADEDİ**
Eşik-1 =60	156
Eşik-2 =100	52
Eşik-3 =150	17
Yalnız vektörler	32
TOPLAM	**257**

Tablo 4.1.b'de verilen kod kitabı, LBG uygulandıktan sonra 255 kod kelimesinden oluşan bir kod kitabına dönüşmüştür. Elde edilen bu yeni kod kitabı kullanılarak öğretici bilgi olarak kullanılan resimlere ve ayrıca farklı iki resme vektör kuantalama uygulanmıştır.Bu resimlere ait MAE ve PSNR değerleri Tablo 4.2.'de verilmiştir. Bu değerler görünüşte uygun değerler olmakla birlikte, resimlerin hepsinde çeşitli seviyelerde kutulanma hataları vardır. Özellikle öğretici dizi içinde olmayan resimler için oluşan kutulanma etkisi çok daha baskındır. Örneğin Susie'de gözü çok rahatsız edecek hatalar vardır. Buna karşılık Salesman'de gözle görülür bir hata yoktur. Lena ve Tiffany'de ise, resimdeki kişilerin tenlerinde yani yumuşak ton değişimi olan bölgelerde kutulanma hatası rahatsız edici etkiler oluşturmaktadır. Bu yüzden, sistemin, bu yöntemle genel bir kullanımının mümkün olamayacağı görülmektedir. Hata değerlerinin çok iyi olması, sadece iki katsayıya kuantalama uygulanmasından kaynaklanmaktadır. Yüksek frekanslar olduğu gibi bırakıldığından resimdeki **görülür hata** sadece 8*8'lik blokların sınırlarındaki gri ton süreksizliğinden ibarettir.

Tablo 4.2.:(1,0) ve (0,1) katsayılarına vektör kuantalama uygulanmış resimlere ait hata değerleri.

	MAE	PSNR(dB)
LENA	**2.11**	**38.53**
TIFFANY	**1.83**	**39.79**
SALESMAN	**1.97**	**39.18**
BABOON	**2.26**	**38.36**
SUSIE	**2.18**	**37.61**
F. MAÇI	**2.72**	**30.50**

4.2.2. İki Kod Kitabı ile Kodlama

Vektör kuantalamasının genel kullanımı ile ilgili esas sorun, elde edilen kod kitaplarının evrensel olmayışlarıdır. Çünkü kod kitaplarının bulunmasında kullanılan öğretici dizinin istatistik yapısı kod kitabına hakim olmakta ve bu da farklı yapıda bir resmin söz konusu kod kitabıyla kodlanmasında kalitenin düşmesine neden olmaktadır. Bu durum, yukarıdaki örnekte de açıkça görülmektedir. Örneğin kod kelimesi sayısı 256 ile sınırlandığında, sadece 256 kombinasyon hatasız temsil edilmiş olur. Bunların dışındaki varyasyonlar en yakın olana atanır. Özellikle (bir çok uygulamada olduğu gibi) 4*4'lük vektörler düşünüldüğünde, 256 kombinasyonun tüm bilgiyi yeterli kalitede temsil etmesi zordur. Kod kitabının büyütülmesi bunun için bir çözümdür. Örneğin 512 kod kelimesi kullanılması durumunda, sistemin hatasız temsil ettiği kombinasyon sayısı 2 kat artacaktır. Bununla beraber, sistemin hafiza ihtiyacı ve kodlama süresi de iki kat artacaktır.

Alternatif bir yöntem iki adet 256 kelimelik kod kitabı oluşturulmasıdır. Öyle ki, her iki kod kitabı da kodlanacak vektörlerin değerlerini temsil edebilecek kod kelimeleri içerirler. Kodlanacak bilgideki her bir vektöre, her bir kod kitabından birer tane kod kelimesi karşı düşürülür. Kod çözümü sırasında ise atanan kod kelimelerinin **ortalamaları** gerçek değer olarak atanır[7]. Böylece sistemin performans kaybı yine iki kat olurken, elde edilebilen kombinasyon sayısı karesel artar ve 65536 olur.

Bu tezde bu yönteme değişik bir yaklaşım yapılmış ve ikinci kod kitabının, gerçek değerleri değil de birinci kodlamadan sonra oluşacak hata vektörlerini temsil etmesi öngörülmüştür. Böylece hem kombinasyon sayısı karesel artacak hem de ikinci kod kitabı çok daha küçük bir hacime sıkışan hata değerlerini temsil etmekle yükümlü olacaktır. Bunun bir avantajı da hata değerlerinin oluşturacakları kümenin, resim bilgisinin oluşturacağı kümeye göre daha evrensel olmasıdır.

Söz konusu uygulamanın gerçekleştirilmesinde 4*1 ve 8*1'lik vektörlerle denemeler yapılmıştır. Vektör boyutlarının bu şekilde seçilmelerinin sebebi kuantalamayla ilgili organizasyonun basitleştirilmesi açısındandır. Bu yöntemin ilkinden daha başarılı olacağı düşüncesiyle, genel bir kullanımda en ideal sistemin elde edilmesi amaçlanmıştır. İlkin, daha önceki uygulamada yapıldığı gibi bir başlangıç kod kitabı elde edilmiş ve LBG ile optimize edilmiştir. Ardından elde edilen kod kitabı kullanılarak fark değerleri için bir kod kitabı bulunmuş ve LBG ile optimize edilmiştir. Öğretici küme olarak 5 resim kullanılmıştır. Bölüm 4.2.1'deki resimlere ek olarak Airplane resmi öğretici kümeye eklenmiştir. Fazladan bir resim kullanılmasının sebebi de bu sistemin daha başarılı olacağı beklentisidir.

Bu kod kitapları oluşturulurken iki kod kitabından birinin 256'dan daha fazla elemana sahip olması bir dezavantaj getirmemektedir. Bunun nedeni, iki kod kitabının toplam kod kelimesi sayısının 512'nin altında kalmasıdır. Böylece toplam iki baytta iki kod kelimesi her zaman için ifade edilebilir. Bu yüzden Tablo 4.3.b'de verilen kod kitapçığında birincil kod kitabının(esas değerleri kodlayan) 328, ikincil kod kitabının(fark değerleri kodlayan) ise 177 olması bir sorun oluşturmamaktadır. Tabii bu durumda 256*256=65536 kombinasyondan daha azı sağlanmaktadır ama elde edilen sonuç LBG'nin ortaya çıkarttığı bir sonuçtur ve kombinasyon sayısı yine de 256'nın çok üzerindedir.

Tablo 4.3.: İki kod kitabı ile klodlama. Eşik değerler daha yüksek olduğundan (b)'de daha az kod kelimesi vardır.

BİRİNCİL KOD KİTABI	
Eşik Değerleri	**Bulunan Kod Kelimesi Sayısı**
Eşik-1 = 8	75
Eşik-2 = 15	112
Eşik-3 = 25	93
Eşik-4 = 35	66
Eşik-5 = 50	56
Eşik-6 = 70	44
Eşik-7 = 90	13
Yalnız Vektörler	82
Toplam	**541**
İKİNCİL KOD KİTABI	
Eşik Değerleri	**BulunanKod Kelimesi Sayısı**
Eşik-1 = 5	127
Eşik-2 = 10	65
Eşik-3 = 15	25
Eşik-4 = 20	9
Eşik-5 = 25	4
Eşik-6 = 30	2
Eşik-7 = 35	1
Yalnız Vektörler	7
Toplam	**240**

(a)

BİRİNCİL KOD KİTABI	**Bulunan Kod Kelimesi Sayısı**
Eşik Değerleri	
Eşik-1 = 30	138
Eşik-2 = 60	93
Eşik-3 = 90	28
Eşik-4 =100	7
Yalnız Vektörler	62
Toplam	**328**
İKİNCİL KOD KİTABI	
Eşik Değerleri	**Bulunan Kod kelimesi Sayısı**
Eşik-1 = 10	117
Eşik-2 = 16	32
Eşik-3 = 24	12
Eşik-4 = 36	8
Eşik-5 = 45	0
Yalnız Vektörler	8
Toplam	**177**

(b)

Tablo 4.3'te verilen başlangıç kod kitapları LBG ile optimize edildikten sonra vektör kuantalama uygulanmıştır. Elde edilen resimlerin MAE ve PSNR değerleri Tablo 4.4'te verilmiştir. Salesman'de her iki durumda da gözle görülür hiç bir hata yoktur. Zaten 40dB'in üstündeki PSNR değerleri de bunu göstermektedir. Buna karşılık Lena'da belirgin kutulanma etkileri vardır ve bunlar sistemin genel amaçlı kullanımını engelleyecek mertebededir.

Tablo 4.4.a: Tablo 4.3.a'daki başlangıç kod kitabının optimize edilmesi ile elde edilen kod kitapları ile kodlama sonucu oluşturulan resimlere ait hata değerleri.

RESİM	MAE	PSNR(dB)
LENA	2.04	34.15
SALESMAN	0.87	43.01
F. MAÇI	1.36	35.47

Tablo 4.4.b: Tablo 4.3.b'deki başlangıç kod kitabının optimize edilmesi ile elde edilen kod kitapları ile kodlama sonucu oluşturulan resimlere ait hata değerleri.

RESİM	MAE	PSNR(dB)
LENA	2.09	34.32
SALESMAN	0.92	42.89
F. MAÇI	1.36	36.10

Görüldüğü gibi daha küçük olan (Tablo 4.3.b) kod kitabından elde edilen resimlerin kalitesi, daha büyük olan kod kitabı için elde edilenler ile hemen hemen aynıdır. Değerler, Salesman için daha olumsuz iken, Lena için MAE olumsuz, PSNR ise daha olumludur (Aradaki farklar çok küçüktür.).

Buradaki önemli diğer bir nokta bu resimlerin bir önceki bölümde elde edilen resimlerle olan karşılaştırmasıdır. Bölüm 4.2.1'de elde edilen resimlerle karşılaştırıldığında Lena'da daha kötü, Salesman'de ise daha iyi sonuç elde edildiği açıktır (Aslında aynı durum Tiffany için de geçerlidir. MAE=43.38 ve MAE=0.76. Görüntü iki istisnai blok hariç mükemmel.). Dolayısıyla iki kod kitabı kullanmanın getirmesi gereken bir avantaj sağlanmıştır. Fakat bu, genel bir uygulama için hala yeterli değildir.

4*1'lik vektörlerle yeterli sonuç alınamaması, 8*1'lik vektörler kullanarak da iyi bir kuantalayıcının elde edilemeyeceğini göstermektedir. Tablo 4.5'te 8*1'lik vektörlerle yapılan vektör kuantalaması uygulamalarında en iyi sonucu veren kuantalayıcıya ait başlangıç kod kitabı ile ilgili bilgiler verilmiştir. Bunların oluşturulmasında tüm şartlar, vektör boyu hariç, 4*1'lik vektörlerle kuantalama uygulamasındakilerle aynıdır.

Gerek birincil gerekse ikincil kod kitapları LBG ile optimize edilmiştir. Optimizasyonun sonunda, birincil kod kitabının 305, ikincil kod kitabının ise 358 kod kelimesi kalmıştır. Bu kod kitapları ile vektör kuantalama uygulanan resimlere ait MAE ve PSNR değerleri Tablo 4.6'da verilmiştir. Lena'da önemli miktarda kutulanma hataları vardır. Buna karşılık 4*1'lik vektörlerle mükemmel sonuç veren Salesman'de de, Lena'daki kadar olmasa da kutulanma hataları oluşmuştur.

Kodlama sırasında oluşacak hatalar açısından en kritik olan (0,1) ve (1,0) frekans bileşenlerine vektör kuantalama uygulamaları buradaki yöntemler çerçevesinde yeterli bir sonuç vermemiştir. Bir sonraki bölümde daha yüksek mertebeden bileşenlerin durumu incelenecektir.

Tablo 4.5: 8*1 boyutlarında VQ uygulaması için elde edilen başlangıç kod kitaplarına ait eşik değerleri ve kod kelimesi sayıları.

BİRİNCİL KOD KİTABI	**Bulunan Kod Kelimesi Sayısı**
Eşik Değerleri	
Eşik-1 = 15	8
Eşik-2 = 30	25
Eşik-3 = 45	30
Eşik-4 = 60	39
Eşik-5 = 80	33
Eşik-6 = 100	25
Eşik-7 = 130	32
Yalnız Vektörler	138
TOPLAM	**330**
İKİNCİL KOD KİTABI	**Bulunan Kod Kelimesi Sayısı**
Eşik Değerleri	
Eşik-1 = 2	5
Eşik-2 = 4	2
Eşik-3 = 6	2
Eşik-4 = 8	7
Eşik-5 = 10	12
Eşik-6 = 15	36
Eşik-7 = 20	40
Yalnız Vektörler	259
TOPLAM	**363**

Tablo 4.6: Tablo 4.5'teki başlangıç kod kitaplarının LBG ile optimizasyonundan elde edilen kuantalayıcının uygulandığı resimler için MAE ve PSNR değerleri.

RESİM	**MAE**	**PSNR(dB)**
LENA	3.77	30.44
SALESMAN	2.11	36.13

4.2.3. Yüksek Mertebeden Bileşenlerin Kuantalanması

Kayıplı kodlamada daha az hataya sebep olan 2. ve 3. mertebe frekans bileşenlerine VQ uygulanması ve yeterli seviyede kalitenin elde edilmesi, (0,1) ve (1,0) bileşenlerinin iyi sonuçlar vermemiş olmasına rağmen, söz konusu bileşenlere 8*1'lik vektörler halinde VQ uygulanarak bir karşılaştırma yapılmıştır. Burada nihai amaç, elde edilen resimlerin kaliteleri olduğundan sadece bunlarla ilgili sayısal değerler verilecektir.

Vektör Kuantalama 1. , 2. ve 3. derece frekans bileşenlerine ayrı ayrı ve birlikte uygulanmıştır. Sadece 3. derece katsayılara yapılan uygulamada bile görsel hatalar mevcuttur ve bunlar sistemin genel kullanımına engel teşkil etmektedirler. Bu durumda, elde edilen hiç bir kod kitabı istenen özellikleri sağlayamamıştır.

Tablo 4.7: LENA'nın 1,2 ve 3. derece DCT katsayılarına VQ uygulanması sonucu elde edilen resimlerin MAE ve PSNR değerleri.

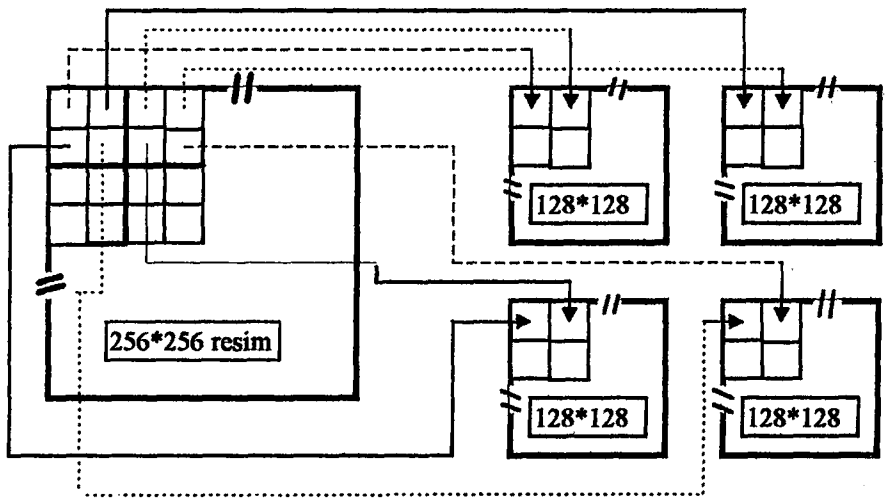
VQ uygulanan frekans bileşenlerinin dereceleri	MAE	PSNR(dB)
3	2.63	34.85
2	2.97	33.92
1	6.19	27.42
3 ve 2	4.14	31.52
3 , 2 ve 1	7.84	26.09

BÖLÜM 5
RESMİN BÖLÜNEREK KODLANMASI

5.1. AMAÇ

Bilgilerin kayıpsız olarak kodlanmasında temel olarak işe yarayan gerçek, mesaj dizilerindeki sembollerin dengesiz bir istatistik dağılıma sahip olmalarıdır. Yani mesaj dizisinde bazı sembollerin daha sık, bazılarının daha seyrek olmalarından yararlanılarak kayıpsız sıkıştırma yöntemleri elde edilir(Huffman Kodlaması gibi). Bu yüzden dönüşüm sonucu elde edilecek katsayıların veya bu katsayıların kuantalanması sonucu elde edilen sayıların yeterince önemli bir kısmının görece dar bir değer kümesinde kalması hedeflenir (JPEG'de çok sayıda sıfırın elde edilmesi gibi.). Hatta eşit değerli katsayıların ardarda gelmesi halinde Akış Uzunluklu Kodlama (Run Length Coding) çok verimli olabilir.

Bu bölümde anlatılan yöntem, mevcut metodların elde ettiğinden daha sık bir şekilde, transform domeninde benzerlikler elde etmeyi amaçlamaktadır. Bunun temeli de, resmi küçük benzer kopyalarına bölerek, bu kopyaların DCT katsayılarının birbirlerine benzeştirilmesidir. Küçük parçaların oluşturulması, Şekil 5.1'de gösterilmiştir. Bu örnekte resim 4 ayrı benzer resme bölünmektedir. 256*256 boyutlarındaki resmin sol üst köşedeki ilk pikseli birinci alt resmin sol üst köşedeki pikseli olarak atanmaktadır. Ona komşu olan 3 piksel de diğer 3 alt resmin sol üst köşesindeki piksel olarak seçilmektedir. Diğer tüm 2*2'lik resim blokları için bu ayrıştırma devam etmektedir.



Şekil 5.1: Resmin 4 küçük benzer parçaya bölünmesi. Şekildeki her küçük kare bir pikseli temsil etmektedir ve piksellerin büyük resimden küçük resimlere atanış şekilleri oklarla gösterilmiştir.

Bu şekilde resim birbirine benzeyen, fakat her biri diğerinden kesinlikle farklı 4 parçaya ayrılmış olur. Burada temel amaç, bu benzer fakat farklı resimlerin transform domeninde sahip olması beklenen benzerliklerden yararlanarak kodlamanın gerçekleştirilmesidir. Örnek olarak 128*128'lik resimlerin sol üst köşesindeki 8*8'lik ilk blok alınsın. Her dört resimde de bu blokların içerdiği desenler birbirine benzediğinden DCT alındığında bulunacak katsayılar da yakın olmalıdır. Ve örneğin her bloktaki DC katsayıları birbirlerine yakın değerler almalıdır. Bu dört DC katsayısı bir şekilde ortak olarak ifade edilebilirse veya biri olduğu gibi saklanıp diğerleriyle olan ilişkisi verimli bir şekilde kodlanabilirse, iyi bir yönteme varılabilir.

Burada şeklen anlatımı kolay olduğu için resmin dörde bölünmesi örneği verilmiştir. Birbirine yakın olması beklenen 4 katsayının yerine sadece bir katsayı konması mümkün olsa bile 4 kat sıkıştırma sağlanmış olur ki, bu yetersizdir. Bunun yerine resmin 16 veya 64'e bölünmesi verimli bir kodlama için gereklidir.

5.2. UYGULAMA VE SONUÇLAR

Öncelikle birbirine yakın olması gereken katsayılar hakkında bir fikir edinilmesi gereklidir. Bunun için resmin 64'e bölündüğü durumda bu katsayılar 8*8'lik matrisler halinde dizilmiş ve gerçekte ne kadar yakın olduklarına bakılmıştır. Başlangıçta umulan, bu 64 sayıdan önemli bir kısmının birbirine eşit olmaları ve farklı olanların da görece dar bir dağılım göstermeleridir.

Fakat ortaya çıkan dağılım beklenene göre oldukça olumsuz özelliklere sahiptir. Katsayıların bir kısmının eşit olması bir yana, dağılım ortalamanın etrafında ± 100 gibi oluşmaktadır. Bu durumda beklenen avantajlar sağlanamamaktadır. Bu dağılımın aralığını küçültmek için ikinci bölümde uygulanan ölçekleme işlemi kullanılmıştır. Yani her katsayı 1024 ile toplanıp 8'e bölünmüştür. Böylece katsayıların değer kümesi $[0,255]$ olarak belirlenir ve dağılım daha dar bir bölgeye sıkışır. Bu şekilde ilk adım olarak skalar kuantalama uygulanmış ve bir miktar bilgi kaybedilmiş olur.

Bu adımdan sonra ortaya çıkan tablo ile en uygun modelin özel bir vektör kuantalaması olduğu sonucuna varılmıştır. Buna göre önce, yakın değerlere sahip ve her alt resimde denk pozisyonda bulunan katsayılardan oluşan 8*8'lik blokların ortalamaları bir baytta saklanır. Sonra blok 4*4'lük 4 adet alt matrise bölünür ve katsayılardan ortalama çıkartılır. Fark değerler ise 4*4'lük bloklar halinde vektör kuantalamasına sokulur.

Vektör kuantalamanın uygulanması için gerekli kod kitapları 4 ayrı resmin katsayılarından yararlanılarak elde edilmiştir. Bunlar Lena, Baboon, Tiffany ve Salesman'dir. Kod kitabı sayısı da dörttür. Birinci kod kitabı DC katsayılar için ayrılmıştır ve dört resme ait tüm katsayıları içermektedir. Yani kodlamada DC değerleri aslında kayıpsız bir şekilde saklanmaktadır. İkinci kod kitabı da (0,1) ve (1,0) katsayıları için kullanılmıştır(zig zag tarama sırasına göre 1 ve 2 numaralı elemanlar.). Üçüncü kod kitabı 3-14 numaralı katsayılar için, dördüncü kod kitabı da

kalan katsayılar için ayrılmıştır. Her kod kitabında 256 vektör vardır ve kod kelimeleri ilgili katsayılardan örnek vektörler alarak seçilmiştir.

Resim bölünerek kodlandığı, kod çözümü sırasında da yeniden komşu pikseller bir araya getirildiği için elde edilecek resimde kutulanma etkisi beklenmemelidir. Yöntemin başarısız olması durumunda, beklenmesi gereken, daha çok resme bir gürültünün eklenmesi şeklinde olmalıdır; çünkü komşu noktalar ayrı ayrı kodlanmaktadır. Gerçekten de, Lena söz konusu sistemle kodlandığında resme gürültü bindirilmişçesine bir etki görülmektedir. Resim adeta karlı bir televizyon yayınından alınmış bir kare gibidir. Ayrıca kimi yerlerde (Örneğin Lena'nın omuzunun kenarında) geçişler keskinleşmiş ve yumuşaklığını kaybetmiştir.Resmin hata değerleri MAE=8.24 ve PSNR=27.10dB'dir.

Resmin bölünmesinden sonra DCT uygulandığı için, yüksek frekanslı DCT katsayıları alışılanın aksine oldukça büyük çıkmaktadır(oldukça sık 100'ün üzerinde değerler almaktadır.). Bu yüzden alışılagelmiş skalar kuantalama uygulanması verimli bir yol değildir. Verimli hale getirmek için daha kaba kuantalama yapılırsa alt resimlerdeki keskin geçişler yok olacak ve asıl resimde yeniden yanyana getirilen pikseller büyük ihtimalle aynı değeri taşıyacaklardır. Bu durumda da çok belirgin kutulanma hataları görünecektir. Hatta yeniden biraraya gelen 8*8'lik komşu piksellerin hepsinin aynı değere gelmiş olması, aslında resmin çözünürlüğünün, dolayısıyla taşıdığı bilginin düşmesi anlamına gelecektir.

Resmin benzer alt parçalara bölünmesinin istatistik kolaylığı sağlamaması ve bu yüzden uygulanan yöntemle iyi bir sıkıştırma oranı elde edebilmek için fazla miktarda bilgi kaybolması, söz konusu yöntemin kullanışlı bir kodlama uygulaması oluşturmasını engellemektedir. Alt resimlerde birbirine karşılık düşen bloklara ait DCT katsayıları yeterince yakın çıkmamıştır. Bu durumda yöntem olumlu bir temel oluşturamamaktadır ve görüntü kodlama için uygun değildir.

BÖLÜM 6

DÜZGÜN OLMAYAN SKALAR KUANTALAMA

6.1. AMAÇ

Vektör kuantalama uygulamalarında görülen temel sorun birden fazla değere ortak olarak bir değer kümesinin atanmasıdır. Örneğin 2*2'lik bir uygulamada, bir vektör için atanacak olan kod kelimesi vektörün ilk üç bileşeni için çok az hata değeri veriyorsa, dördüncü bileşen için çok uygun olmasa da seçilir. Çünkü sonlu sayıda olan kod kelimelerinden en az hata veren seçilmek zorundadır. Bu durum vektör kuantalama uygulamaları gerçeklenirken yapılan bir bilgisayar programı yardımıyla çok açık bir şekilde görülmüştür. Örnek kod kitapları ve katsayı vektörleri için hangi vektörlere hangi kod kelimelerinin atandığını gösteren bu program yardımıyla çok ilginç gözlemler yapılmıştır. Özellikle, istatistik olarak daha az rastlanan ve bu yüzden kod kelimelerinde de daha nadir bulunan büyük değerleri içeren vektörler sorun oluşturmaktadır. Örneğin (100,-100,90,90) gibi bir vektör için (90,-90,85,0) gibi bir kod kelimesi veya (30,30,0,0) gibi bir vektör için (35,40,20,20) gibi bir kod kelimesi atanabilmektedir. Bu hatalardaki karakteristik özellik, ya var olan belirgin bir desenin bloktan tamamen atılması (1. örnek) ya da blokta hiç olmayan desenlerin bloğa eklenmesi (2. örnek) şeklinde ortaya çıkmaktadır. Tabii kod kitabını büyüttükçe bu sorunlar azalmaktadır, fakat bu kez de sıkıştırma oranı küçülmektedir.

Bu sorunların çözümüne bir alternatif pratikte tek boyutlu vektör kuantalama demek olan **Düzgün Olmayan Skalar Kuantalama(DOSK)** yöntemini kullanmaktır. Temel olarak 2.Bölüm'de anlatılan bu yöntem, adeta skalar ve vektörel kuantalama yöntemlerinin arasında kalan melez bir yöntem gibidir. Her bir katsayıya tek bir

karşılık atandığı için skalar bir yöntemdir. Diğer yönden bir kod kitabı vardır ve bu kod kitabındaki katsayılar aslında LBG'nin atası olan Max-Lloyd yöntemiyle oluşturulur. Kod kitabı vektörler yerine sadece skalar büyüklükler içerdiğinden, bunların saklanması için gerekli hafiza miktarı ve atanacak sayıların bulunması için gerekli işlem adedi çok daha azdır. Skalar kuantalama yöntemlerindeki gibi bölme ve çarpma işlemleri de gerektirmez. Uygulamadaki bu kolaylıkları ve vektör kuantalamaya göre sahip olduğu avantajı, DOSK ile uygun bir yöntem aramanın iyi sonuçlar verebileceğini göstermektedir.

DOSK uygulamasına geçmeden önce, uygulamada seçilen yolun ve yapılan tercihlerin daha iyi anlaşılabilmesi için öncelikle JPEG hareketsiz görüntü sıkıştırma yönteminin anlatılmasında fayda vardır. Çünkü JPEG günümüzde en yaygın olarak kullanılan standarttır ve DOSK, yapısı itibariyle, özellikle JPEG sistemini iyileştirmeye müsaittir. Uygulamada da bu amaç doğrultusunda hareket edilmiştir.

6.2. JPEG HAREKETSIZ RESİM SIKIŞTIRMA STANDARDI

JPEG[8] (Joint Photographic Experts Group), hareketsiz görüntüleri sıkıştırmak için bugün en yaygın kullanılan standarttır. Özellikle yaygın olarak kullanılan hareketli görüntü sıkıştırma yöntemi olan MPEG'in (Motion Picture Experts Group) içinde kullanıldığı da düşünülürse önemi daha da iyi anlaşılır.

JPEG'de de dönüşüm için 8*8 DCT kullanılır ve dönüşüm katsayıları 2. Bölüm'de tarif edilen şekilde, zig zag tarama sırasına göre dizilir. Sadece DC katsayılar ayrı kodlanır ve 63 AC katsayı bu tarama sırasına göre işleme tabi tutulur. 64 katsayıya da skalar kuantalama uygulanır. Yani her katsayı önceden belirlenen sayılara bölünür. Bu kuantalayıcı sayılar, resim için seçilen kalite seviyesine göre belirlenir. IJG'nin[9] JPEG kodlama programlarında bu kalite 0-100 arası bir sayıyla belirlenir. 0 en kötü kaliteyi, 100 ise en iyi kaliteyi gösterir. Kullanışlı olan bölge (5,95) arasıdır. 50 ise resmin ortalama kabul edilebilir bir kaliteye sahip olabildiği en düşük seviyedir (Seçilen kalite seviyeleri, istatistik özellikleri farklı resimlerde farklı kaliteye sebep olurlar). Tabii kalite seviyesi düştükçe sıkıştırma oranı artmaktadır.

Kuantalayıcı sayıların tespit edilmesi için özel görsel testler yapılmıştır. Gözün ekran boyunun altı katı mesafeden hiç bir hata göremediği resimleri veren kuantalama tabloları Tablo 6.1'de verilmiştir. Bu tabloların elde edilmesi sırasında 720*576 çözünürlükte parlaklık bilgisi ve 360*576 çözünürlükte renk bilgisi içeren resimler kullanılmıştır. Bu değerlerle kuantalama yapıldığında, profesyonel ekranlarda bir takım hatalar göze çarpmaktadır. Buna karşılık yarı değerleri kullanıldığında elde edilen resim, aslından ayrılamayacak kadar kusursuzdur.

Tablo 6.1.a: Parlaklık bilgisi için kuantalama değerleri
Her kuantalayıcı sayı, kuantalayacağı katsayının DCT matrisindeki yerinde gösterilmiştir.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Tablo 6.1.b: Renk bilgisi için kuantalama değerleri

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	90
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tablo 6.1'te verilen kuantalama değerleri kalite seviyesi 50 için geçerli değerlerdir.

Resim sıkıştırma yöntemlerinin dayandığı temel, gözün yüksek frekanslara karşı duyarlığının düşük oluşudur. Bu yüzden Tablo 6.1'de verilen kuantalayıcı matrislerde yüksek frekanslı dönüşüm katsayıları daha büyük sayılara bölünmektedir. Diğer yandan doğal görüntülerin çoğunda yüksek frekanslı bileşenler zayıftır ve ilgili katsayılar görece küçüktür. Daha küçük katsayıları daha büyük kuantalama sayılarına böldüğümüz için, yüksek frekans katsayılarının büyük bir kısmı kuantalamadan sonra sıfır değerini alır. Tabii alçak frekans bileşenlerinin bir kısmı için de aynı durum söz konusudur. Bu yüzden, JPEG standardında AC katsayılar için bu özellikten yararlanan Akış Uzunluklu Kodlama kullanır. DC'ler için ise DPCM (Differential Pulse Code Modulation) kullanılır. Ardından her iki gruba ait bilgi Huffman Kodlamasından geçirilir.

Gerek DC'ler için DPCM'in uygulanmasıyla elde edilen fark değerlerini, gerekse kuantalanmış AC değerlerini ifade etmek için Tablo 6.2'deki gruplama kullanılır. Buna göre 4 bitlik bir bilgiyle, temsil edilecek katsayının hangi değer kümesinde olduğu gösterilir ve bu küme içindeki tam değerini ifade etmek için **ek bitler** kullanılır.

Tablo 6.2'de SSSS olarak verilen değerler, gösterdikleri gruptaki katsayıların gerçek değerlerini ifade edebilmek için kaç tane ek bit gerektiğini gösterir ve aynı zamanda grubun sıra numarasıdır. Örnek vermek gerekirse -6'yı ifade etmek için SSSS=3 ve ek bitler 001 saklanır.

Akış uzunluklu kodlama yapılırken bu tablodan yararlanılır. 63 AC katsayı için yapılan Akış Uzunluklu Kodlamaya 1.sıradaki (1,0) elemandan itibaren sıfırlar sayılarak başlanır. En fazla 15 taneye gelindiğinde durulur. Çünkü kodlamada her baytın ilk 4 biti ardarda gelen sıfır adedini, ikinci 4 biti ise SSSS numarasını göstermek için kullanılır. Kodlama bu şekilde devam eder. Ardarda 16 tane sıfır, tüm AC katsayılar sıfır ve gelinenden itibaren geri kalan tüm katsayılar sıfır anlamına gelen üç tane özel bayt ayrılmıştır. Bu şekilde elde edilen bilgi Huffman Kodlaması'na tabi tutulur. Ek bitler ise ayrı bir yerde sıralanırlar ve istatistik özellikleri uygun olmadığından Huffman Kodlamasına sokulmazlar.

Tablo 6.2: DC ve AC değerleri için gruplama

DPCM ve AC Değer Bölgeleri		SSSS
0		0
-1	1	1
-3,-2	2,3	2
-7...-4	4...7	3
-15...-8	8...15	4
-31...-16	16...31	5
-63...-32	32...63	6
-127...-64	64...127	7
-255...-128	128...255	8
-511...-256	256...-511	9
-1023...-512	512...1023	10
-2047...-1024	1024...2047	11

Tablo 6.3: Ek bitlerin seçilmesi. Sadece SSSS=3'e kadar olan kısmı verilmiştir. Geri kalanı aynı sistemle devam etmektedir.

SSSS	DPCM Fark Değeri veya AC Katsayı		EK BİTLER
0	0		-
1	-1	1	0,1
2	-3,-2	2,3	00,01,10,11
3	-7...-4	4...7	000,..011,100..111

DC katsayıların da DPCM değerleri Tablo6.2 ve tablo 6.3'teki verilere göre kodlanırlar.AC katsayıların Akış uzunluklu kodlama örneği Tablo 6.4'te verilmiştir.

Tablo 6.4: Akış Uzunluklu Kodlamanın gerçekleştirilmesi. İlk dört katsayı sıfır olduğundan, bunu gösteren sayaç RRRR=4 ve ardından gelen sayı -14 olduğundan -14'ün dahil olduğu grubun numarası SSSS=4. Böylece bu bilgiyi temsil edecek olan bayt 16'lık tabanda (44) değerini alıyor. Ayrıca -14'ün tam değerini ifade etmek için (0001) ek bitleri kullanılıyor. (6-8) arası katsayılar için de aynı sistem geçerli. 9.'dan itibaren tüm katsayılar sıfır olduğundan burada özel olarak ayrılmış bir bayt (Blok sonu) kullanılıyor.

Zigzag indeks	1	2	3	4	5	6	7	8	9	10	11	...	63
AC katsayı	0	0	0	0	-14	0	0	1	0	0	0	...	0
RRRR	4					2			Blok Sonu				
SSSS					4			1			0		
Bayt					44			21			0		
Ek Bitler					0001			1			-		

6.3. DOSK'NIN STANDART JPEG'E GÖRE AVANTAJLARI

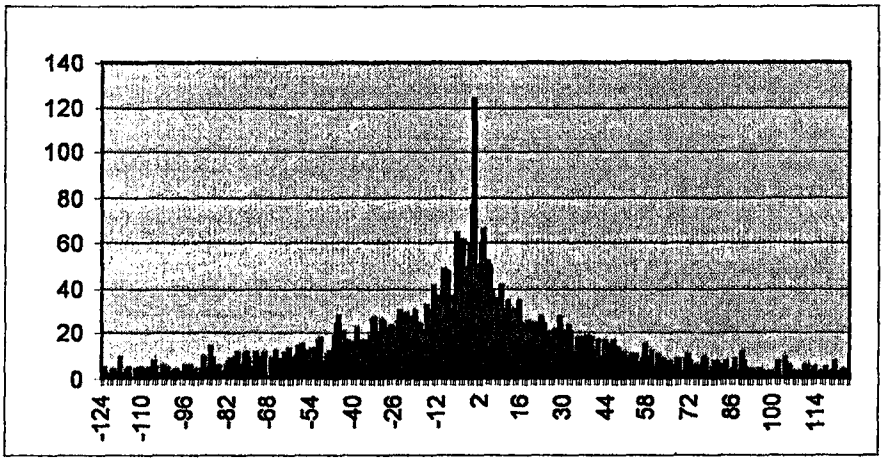
DOSK'da DCT katsayılarını, belirlenen kuantalama sayılarına bölmek yerine, vektör kuantalamadaki gibi bir indeks atandığından Akış Uzunluklu Kodlama'ya giren sıfırdan farklı sayıların değer kümesi daha küçüktür. Örneğin, dört resmin DCT katsayılarından çıkartılan istatistiğe göre, zig zag tarama sırasında 1 numara olan (1,0) elemanının değer kümesi yaklaşık olarak (-500,500) aralığındadır. Bu katsayının Tablo 6.1.a' da verilen kuantalayıcı sayısı ise 11'dir. Dolayısıyla JPEG ile kalite seviyesi 50 seçilerek sıkıştırma yapıldığında bu katsayı (-45,45) gibi bir aralıkta değerler alacaktır. Bu da sıfır hariç 90 farklı değer demektir ve bu bilgi ancak 7 bit ile gösterilebilir. Tabii JPEG'de bu bilgi önce 4 bitlik bir grup numarası ve gerektiği kadar ek bitle sağlanmıştır ve her değer için 7 bit kullanılmaz. Bununla birlikte 1'den 63'e kadar hangi katsayı sıfırdan farklı olursa olsun grup numarası için 4 bit ve gerektiği kadar ek bit mutlaka kullanılmak zorundadır. Sonuçta her sıfırdan farklı katsayı için 5 veya daha fazla bit gereklidir. Üstelik burada ek bitlerin Huffman Kodlaması ile sıkıştırılamadığına da dikkat edilmelidir. Bu yüzden kodlanmış dosyanın boyu açısından ek bitler mertebesinde bir kazanç sağlanır. Ayrıca kullanılan alfabedeki sembol sayısı azaldığından, entropi açısından da bir avantaj sağlanmış olur.

Diğer yandan DOSK'da gerçek değerler yerine indeksler saklandığından temsil edilmesi gereken en büyük sayı, en büyük indeks numarasına eşittir. Daha sonra görüleceği gibi kod kitabında en hassas katsayılar olan (1,0) ve (0,1) için bile 25 ayrı değer yeterli olmaktadır ki, bunlara pozitif ve negatif değerler dahildir. Üstelik daha yüksek frekans katsayıları için bu rakam 3'e kadar inebilmektedir. Bu yüzden DOSK'da ek bitlere hiç ihtiyaç duyulmamaktadır ve bu sıkışmış dosyanın büyüklüğü açısından önemli bir avantaj oluşturmaktadır.

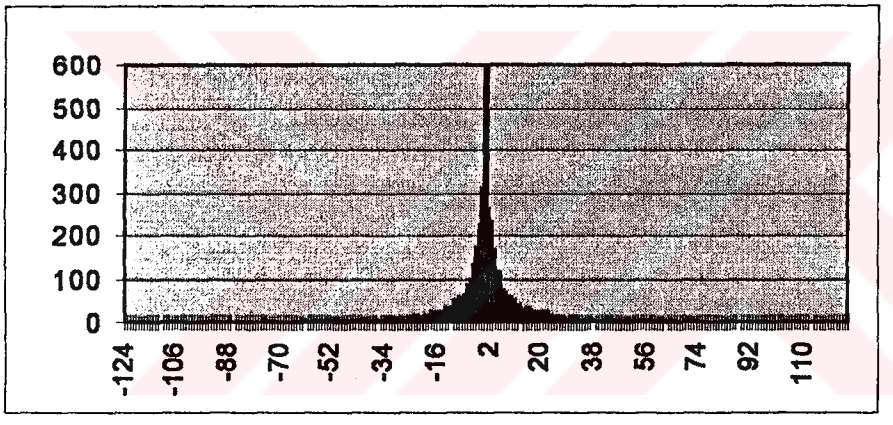
Ayrıca DOSK, normal skalar kuantalama için gerekli olan bölme ve çarpma işlemlerini gerektirmemektedir. Her bir katsayıya karşı düşecek olan değer, vektör kuantalamadaki gibi en yakın olanın, teker teker karşılaştırma yolu ile aranmasıyla bulunabilir. Hatta bu işlemi hızlandırmak için bir **Atama Tablosu (Look Up Table)** da kullanılabilir. Hangi yöntem uygulanırsa uygulansın, kodlama ve kod çözümü JPEG'den daha hızlı olacaktır.

6.4. DOSK UYGULAMASI

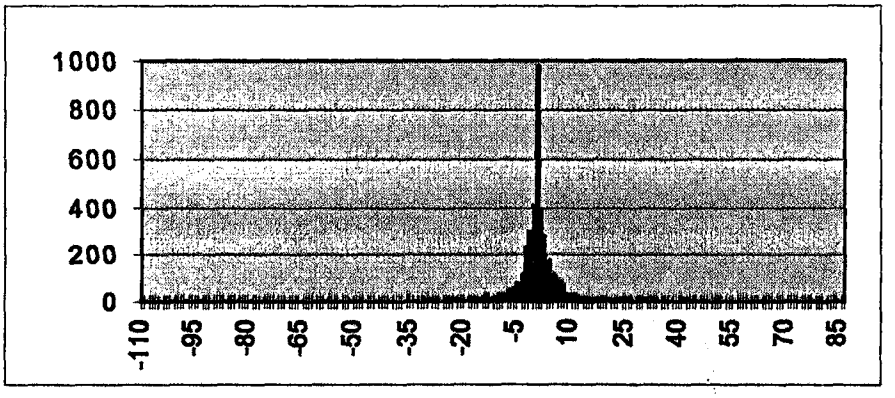
DOSK için kullanılacak kuantalayıcıyı oluşturmak için resimlerin DCT katsayılarına ihtiyaç vardır. Özellikle Max-Lloyd algortiması kullanılacaksa bu zorunludur. Diğer yandan böyle bir kuantalayıcının sahip olması gereken özellikleri yaklaşık olarak tespit edebilmek için de söz konusu istatistik bilgiler gereklidir. Şekil6.1'de Baboon, Salesman, Tiffany ve Lena 'dan elde edilen istatistiki bilgilere dayanarak zig zag tarama sıralamasında 1, 10 ve 63 numaralı katsayıların aldıkları değerlerin dağılımları verilmiştir. Burada hemen dikkat edileceği gibi, yüksek frekans bileşenlerine doğru gittikçe dağılımın değer kümesi hızla daralmakta ve katsayılar daha çok küçük değerlerde yoğunlaşmaktadırlar. Sıfır daima en çok alınan değerdir. Bu noktada şuna dikkati çekmek gerekir. Bu değerler kuantalanmamış, DCT katsayılarıdır. Kuantalama durumunda bu dağılım çok daha dar bir bölgede gerçekleşmektedir. Şekil 6.1.a'da verilen (1,0) elemanına ait olan dağılım gerçekte (-500,500) aralığındadır; fakat gösterilen aralığın dışındaki değerler ya birer ikişer defa alınmakta veya hiç gerçekleşmemektedirler. Şekil 6.1.b'de verilen 10 numaralı katsayı için de gerçek aralık (-154,164)'tür. Diğer yandan Şekil 6.1.c'de verilen aralık dağılımın



(a)



(b)



(c)

Şekil 6.1: Zig zag tarama sırasına göre katsayıların değer dağılımları.(a)'da 1. , (b)'de 10., (c)'de ise 63. katsayı.

tamamını göstermektedir. Sonuçta hepsinde de dağılımın küçük değerler lehine dengesiz olduğu görülmektedir.

DOSK için bir kuantalayıcı tasarlanırken DCT katsayılarını sadece belirli bir hata içinde temsil etmeyi düşünmek yetmez. Gözün görme özelliklerini de hesaba katmak gerekir. Bunun için de ölçü olarak Tablo 6.1.'de verilen kuantalayıcılar kullanılabilir. Genel olarak sadece LBG ile vektör kuantalama yapmanın temel dezavantajı da budur. LBG, katsayılar için hatayı minimize eder, fakat insan görme sisteminin özelliklerini kullanamaz

Bu amaçla birbirine yakın özellikler gösteren katsayılar için ortak kod kitapları hazırlanmış ve bunların içindeki katsayılar Tablo 6.1'deki kuantalayıcılar da göz önünde bulundurularak belirlenmiştir. Küçük değerler için hata, Tablo 6.1'deki kuantalama sayılarına göre biraz küçük tutulmuştur. Farklı denemeler sonucu katsayı değeri büyüdükçe yapılan hatanın Tablo 6.1'in öngördüğünden daha fazla olabileceği görülmüştür. Bu bölümde nihai değerlendirme için kullanılan ve Ek 1'de verilen kuantalayıcı bu esaslara göre elde edilmiştir. Bu şekilde elde edilen kuantalayıcı, gözün görme özelliklerini de hesaba katmış olacağından, resimle ilgili bilgiden atılması sorun olmayacak kısmı atmış olur. Böylece mümkün olduğunca küçük dosya boyuna erişilir.

Diğer yandan Lloyd Max Kuantalayıcısı ile de kod kitapları oluşturulmuştur. Fakat bunlar Tablo 6.1'dekine kıyasla daha düşük hatalara sebep olan kuantalayıcılar ürettiklerinden Tablo 6.1'de verilen ve JPEG sisteminde 50 kalite değerine denk düşen kuantalayıcılarla karşılaştırılmaları doğru olmaz.

Bu kuantalayıcılar 50 seviyesine göre daha kaliteli resimler veren, daha büyük dosyalar oluşturmaktadırlar. Bu yüzden JPEG ile karşılaştırmak için Ek A'da verilen kuantalayıcı kullanılmıştır. Bu kuantalayıcının kullanılmasının nedeni elde edilen resim kalitesinin Tablo 6.1'de verilen kuantalayıcı ile elde edilen resimlerinki ile hemen hemen eşit olmasıdır.

Kullanılan kuantalayıcıda bir tanesi DC'ler için kullanılmak üzere toplam 10 tane kod kitabı vardır. AC katsayılar için olanların gruplanışları ve içerdikleri skalar değerlerin adetleri şu şekildedir:

1. - 2.elemanlar için: 25 adet	3.-5. elemanlar için: 19 adet
6. eleman için: 9 adet	7.-8. elemanlar için: 15 adet
9.-14. elemanlar için: 15 adet	15.-19. elemanlar için: 7 adet
20.-34. elemanlar için: 7 adet	35.-50. elemanlar için: 5 adet
51.-63. elemanlar için: 3 adet	

Burada akla gelebilecek bir soru, seçilen adetlerin, harcadıkları bitlerin tamamını kullanacak şekilde 2'nin en yakın tamsayı üssüne büyütülmelerinin daha iyi bir sonuç verip vermeyeceğidir. Örneğin, ilk iki eleman için 25 yerine 32 kuantalayıcı kullanılabilir. Çünkü bu sayı 25 de seçilse, 32 de seçilse 5 bit kullanacaktır. Fakat yine de tercih, istenen kalitedeki resmi veren mümkün olduğunca küçük değer için yapılmalıdır. Bunun sebebi, sembol sayısının küçük tutulmasının entropiyi olumlu bir şekilde etkileyecek olmasıdır.

6.4.1. DOSK için Akış Uzunluklu Kodlama

DOSK için oluşturulan ve JPEG ile karşılaştırılmak üzere seçilen kod kitaplarının eleman sayıları JPEG'inkinden daha verimli bir Akış Uzunluklu Kodlamayı mümkün kılmaktadır. Bunun sebebi sadece ilk 5 AC katsayının 5 bit, diğerlerinin ise 2-4 bit gerektirmesidir.

Temelde JPEG ile aynı yöntem kullanılabilir. Fakat ilk 5 katsayı için 5 bit ayrılması gerektiğinden, bu katsayılardaki sıfırların sayısı JPEG'deki gibi 4 bite değil 3 bite atanmalıdır. Geri kalan 5 bit ise katsayıya ait kod endeks numarasını taşır. Sıfır sayma işlemi için 3 bit ayrılması, ilk 5 katsayıda en fazla 5 tane sıfır olabileceğinden dolayı olumsuz bir özellik taşımamaktadır.

6. katsayıdan itibaren ise JPEG'in kullandığı gibi 4 bit sıfırların adedini, 4 bit ise katsayıların indekslerini gösterir. Sistem ayrıca bütün AC katsayıların sıfır olup olmadığını en başta kontrol etmekte ve bu doğruysa özel bir karakter kullanmaktadır(FFh). JPEG'de olduğu gibi gelinen AC katsayıdan itibaren geri kalanların sıfır olup olmadığını da kontrol etmekte ve bu durumu ifade etmek için ise (7Fh) karakterini kullanmaktadır.(Aslında bu iki özel karakter yerine bir tek karakter kullanılması mümkündür.Fakat JPEG'de bu yol tercih edilmiştir.)

Görüldüğü gibi Akış Uzunluklu Kodlama bakımından temel fark, ek bitlere ihtiyaç duyulmamasıdır. Huffman kodlamasıyla sıkıştırılamayan ek bitlere gerek kalmaması, dosya boyunu küçültecektir.

DOSK için kodlama örneği Tablo 6.5'te görülmektedir. Burada kodlanan bilgi Tablo 6.4'teki ile aynıdır. Tek farkı zig zag sıralamadaki 5. katsayının (-14) yerine (+15) (indeksler hep pozitif değer alacağından) değerini almasıdır ki, bu kodlama açısından hiç bir şeyi değiştirmemektedir. Tablo 6.4'te, söz konusu bilgi 3bayt + 5 ek bit = 29 bit ile kodlanırken, DOSK ile sadece 3 bayt yeterlidir. Bu da %17.2'lik bir kazançtır.

Akış Uzunluklu Kodlama bu şekilde uygulandıktan sonra Huffman kodlaması ile işlem tamamlanır. DC katsayılar da kuantalandıktan sonra Huffman ile kodlanırlar.

DOSK, anlatıldığı şekliyle 8 ayrı resme uygulanmıştır. Aynı resimler karşılaştırma amacıyla JPEG ile 50 ve civarında kalite seviyelerinde sıkıştırılmış ve sonuçlar gerek kalite açısından gerekse kodlanmış dosyaların boyları açısından karşılaştırılmışlardır. Sonuçlar Tablo 6.6'da verilmiştir.

Tablo 6.5: DOSK ile Tablo 6.4'teki bilginin kodlanması. İlk 5 katsayı için, sıfır adedini göstermek üzere 3 bit, indeks numarasını göstermek üzere 5 bit ayrılıyor. Daha sonraki katsayılarda dörder bit kullanılıyor. Sonuçta ek bitlere hiç gerek kalmadan kodlama tamamlanıyor.

Zigzag indeks	1	2	3	4	5	6	7	8	9	10	11	...	63
AC Katsayı İndeksi	0	0	0	0	15	0	0	1	0	0	0	...	0
Sıfır Sayısı	4					2			Blok Sonu				
Atanacak Bitler	100				01111	0010		0001	01111111				
Bayt (Hex.)	8F					21			7F				

Tablo 6.6: Görüldüğü gibi resimlere ait MAE ve PSNR değerleri DOSK ve JPEG için son derece yakındır. Bununla birlikte DOSK ile elde edilen sıkıştırılmış dosya boyları daha küçüktür. Bunun tek istisnası Ms. Amerika resmidir. Opt. Kodlama sütunu, JPEG'in sıkıştırmakta olduğu resimlerin istatistik özelliklerine göre özel Huffman Kod Tablosu oluşturduğu durumu göstermektedir.Bu durumda Standart Huffman Tablosu ile kodlamaya göre ortalama %10 kazanç sağlanır.

	DOSK			**JPEG**			
RESİMLER	MAE	PSNR (dB)	Dosya Boyu	MAE	PSNR (dB)	Dosya Boyu	Dosya Boyu (Opt.Kodlama)
Airplane	3.97	32.67	7217	3.67	33.07	7941	7412
Baboon	8.32	27.31	14010	8.24	27.38	16969	16199
Lena	3.74	33.55	6759	3.55	33.50	7687	7137
Ms. Amerika	2.87	36.78	3138	2.51	37.52	3745	3112
Salesman	3.77	34.22	7090	3.61	34.25	7988	7427
Splash	3.20	34.81	5033	3.13	35.43	5791	5058
Susie	2.59	37.37	4172	2.54	37.63	5059	4428
Tiffany	3.70	33.74	5142	3.48	33.83	6251	5637

Tablo 6.6'da görüldüğü gibi resimlerin MAE ve PSNR değerleri eşit sayılacak kadar yakındır. Resimler kalite açısından o kadar benzer durumdadırlar ki, oluşan ufak tefek hata desenleri bile JPEG ve DOSK ile kodlanan resimlerde hemen hemen aynıdır. Bir istisna hariç resimlerin tümünde DOSK daha iyi sonuç vermektedir. Sadece Ms. Amerika'nın kodlanmasında, Optimum Huffman Tablosu kullanılırsa, JPEG küçük bir farkla daha avantajlı olmaktadır. Bu resimde de diğerlerinde olduğu gibi hata desenleri benzemekle birlikte, JPEG ile kodlanan resimdeki görülür hata

MAE ve PSNR'dan daha hızlı artmaktadır. Resmin önemli bir kısmı koyu ve düz bir zemin olduğundan, bu kısım MAE ve PSNR'ın büyümesini önlemekte fakat resimdeki şahsın yüzünde hatalar görülmektedir. Örneğin MAE ve PSNR'ın biraz daha iyi görünmesine rağmen, JPEG ile kodlanan resimde gözler, DOSK'dakine göre hissedilir derecede kötüdür.

Sekiz resim için dosya boyu açısından elde edilen kazanç yüzde olarak Tablo 6.7'de verilmiştir.

Tablo 6.7: DOSK ile kodlanan resimlerin JPEG ile kodlananlara göre yüzde olarak küçülme oranları. Negatif değerler büyümeyi göstermektedir

RESİM	Standart Huffman Tablosu ile JPEG	Optimum Huffman Tablosu ile JPEG
Airplane	%9.1	%2.6
Baboon	%17.4	%13.5
Lena	%12.1	%5.3
Ms. Amerika	%16.2	%-0.8
Salesman	%11.2	%4.5
Splash	%13.1	%0.5
Susie	%17.5	%5.8
Tiffany	%17.7	%8.8
ORTALAMA	%16.2	%5.03

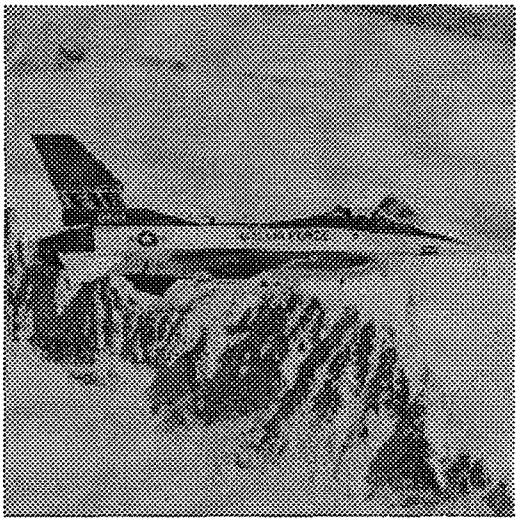
Tablo 6.6 ve 6.7'de örnek verilen resimlerden Tiffany, Lena, Airplane'in DOSK ve JPEG ile kodlanarak elde edilmiş resimleri Şekil 6.2'de verilmiştir.

Şekil 6.2.a: JPEG ile 50 kalite seviyesinde sıkıştırılmış olan Tiffany.

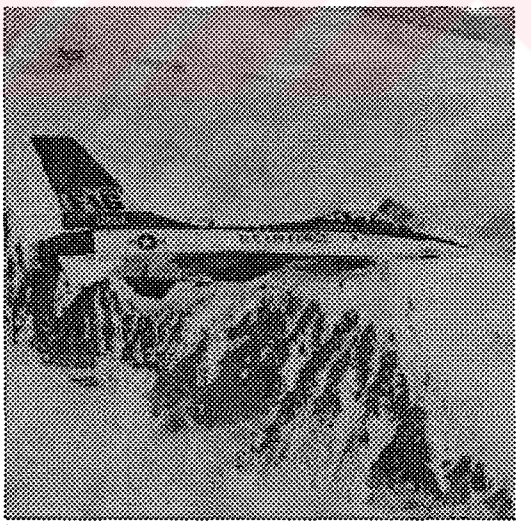
Şekil 6.2.b: DOSK ile kodlanmış Tiffany

Şekil 6.2.c: JPEG ile 50 kalite seviyesinde sıkıştırılmış olan Lena.

Şekil 6.2.d: DOSK ile kodlanmış Lena.



Şekil 6.2.e: JPEG ile 50 kalite seviyesinde sıkıştırılmış Airplane.



Şekil 6.2.f: DOSK ile kodlanmış Airplane.

Görüldüğü gibi resimlerin kaliteleri ile ilgili büyüklükler oldukça yakın tutularak, dosya boyları karşılaştırılmış ve DOSK'nın daha avantajlı olduğu gösterilmiştir. Yine de her resim için kalite değerlerini tam olarak eşitlemek mümkün değildir. Karşılaştırmayı bu dezavantajı ortadan kaldırarak yapabilmek amacıyla bir tanımlama yapılmıştır. Bu tanımlama ile belirlenen kavram **kilobayt başına düşen işaret gürültü oranıdır** ve PSNR'nin dosya boyuna bölünmesi ile elde edilir. Böylece her resim için bir baytın taşıdığı işaret gürültü oranı katkısı hesaplanır ve dosya boyu ile kalite özellikleri açısından kazançları farklı şekilde oluşan 8 resim için DOSK'nın JPEG'e olan üstünlüğünü göstermek üzere kullanılabilir. Bu büyüklük resim boyuna bağlıdır ve sadece eşit büyüklükteki resimler için kullanılabilir. Tablo 6.8'de DOSK ve JPEG ile sıkıştırılmış resimler için söz konusu karşılaştırma yapılmıştır.

Tablo 6.8: 8 resme ait PSNR/Dosya Boyu. Görüldüğü gibi 8 resmin altısında DOSK daha başarılı. Böylece resimler daha ortak tabana sahip bir ölçüyle karşılaştırılmış oluyorlar.

	DOSK	JPEG	
RESİMLER	**PSNR/DosyaBoyu (dB/kbayt)**	**PSNR/DosyaBoyu (dB/kbayt)**	**KAZANÇ(%)**
Airplane	4.53	4.46	1.46
Baboon	1.95	1.69	15.4
Lena	4.96	4.69	5.75
Ms. Amerika	11.7	12.06	-2,96
Salesman	4.83	4.61	4.66
Splash	6.92	7.00	-1.1
Susi	8.96	8.50	5.41
Tiffany	6.56	6.00	9.33
ORTALAMA			**4.74**

6.5. SONUÇ

DOSK, katsayı değerleri yerine endekslerini taşıması ve böylece daha küçük değer kümeli sayıları içermesi, bunun sonucu olarak da JPEG türü bir Akış Uzunluklu Kodlama'da ek bitlere ihtiyaç duymaması ile lineer skalar kuantalama kullanan JPEG'e göre daha üstün sıkıştırma özellikleri sağlayabilmektedir. Diğer yandan Max-Lloyd Kuantalayıcısı ile tezde kullanılandan daha iyi resim kalitesi veren kuantalayıcılar da elde edilmiştir. Bu kuantalayıcılar örenk olarak kullanılan kuantalayıcılara göre daha kaliteli ve daha az sıkıştırılmış resimler oluşturan kuantalayıcılardır. Bunun temel sebebi, Lloyd Max Kuantalayıcısı ile elde edilen kuantalayıcının katsayılarda oluşturduğu hatanın Tablo 6.1'dekinden daha küçük olmasıdır. DOSK hem skalar hem de vektörel kuantalama olmanın avantajlarını sunmaktadır.

Bununla birlikte DOSK söz konusu iki kuantalama yönteminin dezavantajlarını da içermemektedir. Vektör kuantalamadaki gibi çok aykırı değerlerin atanması ihtimali yoktur. İster Max-Lloyd Yöntemi ile kod kitabı elde edilişinde, ister kuantalama yapılırken tüm katsayılar ayrı ayrı ele alındığında bu sorun ortadan kalkmaktadır. Vektör kuantalamaya göre en önemli üstünlüğü Max-Lloyd Yöntemiyle elde edilecek olan bir kod kitabının evrensel kullanıma müsait olmasıdır. Örneğin sadece 4 resimle oluşturulacak bir kod kitabı, bu 4 resmin dışındaki tüm resimleri de başarıyla kodlayabilir. Vektör kuantalamada LBG kullanılarak elde edilen kod kitabı, vektörler birer sayı grubu içerdiğinden, kod kitabı yeterince büyük değilse, evrensel olamamaktadır.

Kuantalama değerleri değişmeyen tek bir kuantalayıcı kullanıldığından, doğal olarak, her resimde aynı sıkıştırma oranı ve resim kalitesi sağlanamamıştır. Bu her resmin farklı istatistik özellikleri olmasından kaynaklanmaktadır. Örneğin Baboon JPEG'dekine göre çok daha fazla sıkıştırılabilmektedir. Tablo 6.8'de görüldüğü gibi, sonuç genel olarak JPEG'dekinden daha iyidir.

Oluşturulan sistem, basit bir şekilde, kullanılmakta olan JPEG sistemine uyarlanabilir ve daha hızlı, daha verimli bir kodlama elde edilmiş olur. Bu arada uygun bir kod kitabının seçilmesi gerekecektir. Ve sistemin gereği bu kod kitabı sabit kalacaktır. Çünkü tanımlanan akış uzunluklu kodlamanın kullanılabilmesi için her bir katsayının endeksi için ayrılan bit sayısı korunmalıdır (Bu sınırlar içinde değişim olabilir. Sınırlar değiştirilirse, akış uzunluklu kodlama da uygun şekilde değiştirilmelidir.). Bu yüzden JPEG'deki gibi farklı kalite ve sıkıştırma oranlarını tercih etmek pek mümkün olmayacaktır. Bu da, eğer bir dezavantaj sayılırsa, DOSK'nın JPEG'e göre tek dezavantajıdır.

BÖLÜM 7
SONUÇLAR VE ÖNERİLER

Tezde 8*8 DCT'ye dayanan hareketsiz resim kodlama yöntemleri incelenmiştir. Bu amaçla dört ayrı yöntem uygulanmıştır. Bunlardan 3. ve 5. Bölümler'de anlatılanlar özgün fikirler olup, daha önce denenmemiş yöntemlerdir.

3. Bölüm'de anlatılan Kaskad DCT uygulamasında, dönüşüm ile elde edilen DCT katsayılarına ikinci bir DCT uygulanmıştır. Birinciden elde edilen katsayıların bir kısmı ikinci DCT'ye, sıfır yapılarak girmiştir. Amaç ikinci DCT katsayılarına da benzer bir işlem yapılarak üçüncü, dördüncü vs. aşamalara geçmektir. Fakat görülmüştür ki, ikinci DCT'yi uygulamanın resim kalitesi açısından bir sakıncası olmamasına rağmen, ikinci DCT katsayılarının bir kısmına sıfır değeri atanması durumunda resimde olumsuz etkiler oluşmaktadır.

Vektör Kuantalama uygulamasında farklı vektör boylarında kodlama yapılmıştır. Elde edilen kod kitaplarının hiç biri evrensel bir kullanım için yeterli olmamıştır. Diğer yandan, bir yerine iki kod kitabı kullanarak resim kalitesinin arttığı gösterilmiştir. Burada temel kazanç, sistem gereksinimlerinin (bellek, kodlama süresi vb.) iki katına çıkmasına karşın, vektör uzayının örneklendiği nokta sayısının karesel artmasıdır.

5. Bölüm'de resimler küçük benzer kopyalarına ayrılarak kodlanmıştır. Böylece benzer 8*8 blokların DCT katsayılarının benzer olması sağlanmak istenmiş ve bu benzerlikten yararlanılmaya çalışılmıştır. Birbirine karşı düşen bloklara ait katsayıların beklendiği kadar yakın çıkmamasına rağmen, elde edilen katsayılara özel bir vektör kuantalaması uygulanmıştır. Resimde komşu olan pikseller ayrılarak DCT alındığından, elde edilen resimlerde kutulanma hatası oluşmamakla birlikte, resme

gürültü binmişçesine bir hata ortaya çıkmıştır. Bu hata da, oluşturulan sistemin bir kodlama yöntemi olarak kullanılmasını engelleyecek mertebededir.

Doğrusal olmayan skalar kuantalama uygulamasında, JPEG standardı temel alınarak bir yöntem geliştirilmiştir. JPEG'in kullandığı lineer skalar kuantalama yerine DOSK uygulanmıştır. Bunun temel avantajı, kayıpsız kodlamaya giren sembol sayısının daha az olmasıdır ve bu yüzden JPEG'de gerekli olan ek bitler kullanılmamaktadır. Ayrıca entropiden de bir miktar kazanç sağlanmaktadır. Gerçekleştirilen uygulamalarda da açıkça görülmüştür ki, aynı kalitedeki resimler için DOSK kullanılarak elde edilen sistem daha küçük kodlanmış bilgi dizileri oluşturmaktadır.

DOSK, lineer skalar kuantalamaya göre böyle bir avantaja sahipken, vektör kuantalamaya göre de avantajlıdır. İki yöntemin ortasında kalan DOSK'da evrensel bir kod kitabı elde etmek, vektör kuantalamadakinin aksine çok kolaydır. Buna karşılık VK'daki gibi optimizasyon yöntemlerine de açıktır.

DOSK, tezde verilen şekliyle veya amaca uygun değişikliklerle, kolaylıka JPEG standardına uyarlanabilir ve böylece JPEG daha verimli bir sistem oluşturur. Aynı şekilde MPEG için de DOSK'dan faydalanılabilir; çünkü MPEG, JPEG'in yöntemlerini de kullanmaktadır. Burada önemli bir nokta da, yapılacak değişikliklerin basit olması ve sistem ihtiyaçlarını da azaltmasıdır. Çünkü DOSK, lineer skalar kuantalamadan daha hızlı bir yöntemdir.

Belirtilmesi gereken son nokta da, DOSK'nın genelde, farklı dönüşümlerle de daha verimli kullanıma açık bir yapıya sahip olmasıdır. Resimleri frekans bileşenlerine ayıran ve DCT'ye benzer bir karakteristikte bilgi kaybına uygun olan her dönüşüm için daha uygun bir seçim olacaktır.

KAYNAKLAR

[1] TELLİOĞLU, ERHAN, MPEG Hareketli Görüntü Sıkıştırma Standardı İTÜ Fen Bilimleri Enstitüsü,Y.Lisans Tezi, Haziran 1995.

[2] AKANSU,A.N., HADDAD,R.A., Multiresolution Signal Decomposition Academic Press Inc., 1992.

[3] MALVAR, H.S., Signal Processing with Lapped Transforms. Artech House, 1992.

[4] DAUBECHIES, I., ANTONINI, M., BARLAUD, M., MATHIEU, P., Image Coding Using Vector Quantization in the Wavelet Transform Domain. IEEE ICASSP 90.

[5] GERSHO, A., GRAY, R.M., Vector Quantization and Signal Compression. Kluwer Academic Publishers, 1992.

[6] ASH, Robert, Information Theory. Interscience Publishers, 1965.

[7] HUH, Y., HWANG, J.J., RAO, K.R., Block Wavelet Transform Coding of Images Using Classified Vector Quantization. IEEE Transactions on Circuits and Systems for Video Technology. Vol 5, No 1, 63-67, Feb. 1995.

[8] PENNEBAKER, W.B., MITCHELL, J.L., JPEG Still Image Data Compression Standard. Van Nostrand Reinhold, 1993.

[9] JPEG FAQ, ftp.uu.net 192.48.96.9 graphics/jpeg.

EK A

Burada, 6. Bölüm'de kullanılan Düzgün Olmayan Kuantalayıcı verilmiştir. Kuantalayıcıların hangi DCT katsayıları için kullanılacakları ise zig zag tarama sıralamasına göre gösterilmiştir. Her bir kuantalama değerinin endeks numarası dizideki sıra numarasına eşittir. İlk değer olan sıfırın endeks numarası sıfırdır.

DC katsayılar için kuantalama değerleri (68 adet):
0,±10,±20,±35,±50,±70,±90,±110,±130,±150,±170,±190,±210,±230,±250,±275,±300, ±330,±360,±390,±420,±450,±480,±510,±540,±570,±600,640,-660,680,-720,720,-780, 760,-840,800,-900,850,-960,900,950,1000.

1-2 nolu katsayılar için kuantalama değerleri (25 tane):
0,±18,±36,±48,±64,±80,±96,±114,±150,±191,±240,±291,±360.

3-5 nolu katsayılar için kuantalama değerleri (19 tane):
0,±21,±42,±63,±84,±100,±125,±161,±200,±251.

6 nolu katsayı için kuantalama değerleri (9 tane):
0,±47,±94,±141,±188.

7-8 nolu katsayılar için kuantalama değerleri (15 tane):
0,±13,±26,±39,±52,±80,±121,±160.

9-14 nolu katsayılar için kuantalama değerleri (15 tane):
0,±16,±32,±48,±64,±90,±131,±170.

15-19 nolu katsayılar için kuantalama değerleri (7 tane):
0,±22,±44,±85.

20-34 nolu katsayılar için kuantalama değerleri (7 tane):
0,±30,±60,±90.

35-50 nolu katsayılar için kuantalama değerleri (5 tane):
0,±51,±100.

51-63 nolu katsayılar için kuantalama değerleri (3 tane): 0,±90.

EK B

Bu bölümde, tez çalışması sırasında yazılan programların kaynak kodları verilmiştir. Yazılan program sayısı ellinin üzerindedir ve bunların hepsi kritik olmadığından, yalnızca önem taşıyanlardan alıntılar aktarılmıştır. Programlar, kolayca incelenebilmeleri amacıyla, oldukları gibi değil, sadece içerdikleri algoritmaların bulunduğu bölümler çerçevesinde sunulmuştur (Örneğin giriş bilgilerinin okunduğu veya kullanıcı arabirimi olan bölümler kısmen çıkartılmıştır.).

1- Bu program iki kod kitabı ile uygulanan Vektör Kuantalama için öğretici dizi oluşturmaktadır. Dört Ayrı resme ait katsayılar kullanılır. Girişteki tanımlamalarda vektör boyutunun 4 olduğu görülmektedir. Zigzag tarama sıralamasına göre 1. elemandan itibaren 2 frekans bileşenine ait katsayılar kullanılacaktır.

```
#define vectornumber 2560
#define vectorsize 4
#define coef1      1
#define coefnr     2


void main()
{

  int tnr,start;
  int threshold[7];


  /**************************
   ** Komşuluk Matrisi          **
   **************************/
   for(i=0;i<vectornumber;i++)   neighbor[i]=0;

  /**************************
   ** Komşu vektörlerin Aranması **
   **************************/
    seqnr=0;
    vnr1=0;
```

```
    for(counter=0;counter<7;counter++){
        thr=threshold[counter];
         for(i=0;i<vectornumber;i++){
            if(neighbor[i]==0){
              seqnr+=1;
              compare();
            }
         }
     }


    /**********************************
     ***  Yalnız Vektörlerin Sayılması    ***
     **********************************/

     alone_vectors=0;
     for(i=0;i<vectornumber;i++) { if(neighbor[i]==0) alone_vectors+=1;}


    /**************************************
     *** Ağırlık Merkezlerinin Hesaplanması ***
     **************************************/

     for(i=1;i<seqnr;i++){
          vnr=0;
          for(j=0;j<vectorsize;j++) ad[j]=0;

          for(j=0;j<vectornumber;j++){
            if(neighbor[j]==i){
              for(k=0;k<vectorsize;k++) ad[k]+=coefmat[j][k];
              vnr+=1;
            }
          }
          for(k=0;k<vectorsize;k++) centmat[i][k]=ad[k]/vnr;
     }


}// main'in sonu



compare()
{
  int c1,c2,other_neighbors=0;
  double distance,d,dif;
```

```
for(c1=0;c1<vectornumber;c1++){
   if(c1==i) continue;            //Kendisiyle karşılaştırılmamalı
   if(neighbor[c1]==0){           //c1 nolu vektör daha önce komşu seçilmemişse
        distance=0;
        for(c2=0;c2<vectorsize;c2++){
          dif=(double)(coefmat[c1][c2]-coefmat[i][c2]);
          distance+=(dif*dif);
        }

        d=sqrt(distance);
        if(d<thr) {neighbor[c1]=seqnr; other_neighbors+=1; } //c1'i, i nolu vektörün
                                                              komşusu olarak ata.
   }
  }

  if(other_neighbors>0) neighbor[i]=seqnr; // eğer i nolu vektörün komşuları varsa,
                                            gurba i ' yi de kat.
  else seqnr-=1;
}// compare isimli alt programın sonu.
```

2- Bu program, iki kod kitabı ile uygulanan Vektör Kuantalama'nın kullandığı **İkincil Kod Kitabını** optimize eden LBG algoritmasını içermektedir. Önce birincil kod kitabı ile VK uygulanır ve ikinci için iterasyon başlatılır. Main'de her şey alt programlara dallanma yoluyla kontrol edilmiştir. Kullanılan tüm alt programlar da bu bölümde yer almıştır.

```
#define vectorsize   4


void main()
{

  int order_end,order_start;
  int vqerror2;

  vectornumber=(1024/vectorsize)*5*(order_end-order_start+1);
  used_vector=(int *) calloc(vectornumber,sizeof(int)); //info of assigned vectors


 /**** Öğretici dizi ile kod kitaplarının okunması ****/

  read_pr_codebook();
  read_codebook();
  read_trseq(order_start,order_end);
```

```
/*****************************************
 ** Fark kod kitabının (ikincil kod kitabı) LBG ile     **
 ** optimizasyonu için önce birincil VK uygulanmalı  **
 *****************************************/

apply_vq_lbg52(vectorsize);
subtract_new_coef(vectorsize);

/*****************************
 *****************************
 ***                       ***
 ***    LBG'nin kontrolü   ***
 ***                       ***
 *****************************
 *****************************/

//VK uygulanmış katsayılar için yeni matrisin tanımlanması.
newcoefmat=imatrix(0,vectornumber-1,0,vectorsize-1);


do{


/*****************************************
 **** LBG algoritmasının ilk adımı          ****
 **** Vektör Kuantalama                     ****
 *****************************************/
 apply_vq_lbg5(vectorsize);
 assign_new_coef(vectorsize);

 /*****************************
  *** Hatanın hesaplanması      ***
  *****************************/

  vqerror=vq_error();
  vqerror2=(int)(vqerror+.5);

  if(vqerror>threshold){
      clrscr();
      gotoxy(5,5);
      cprintf("The mae is %4.2f and still greater than the treshold!",vqerror);

      used_cw_nr=0;
      for(i=0;i<limit;i++){
         if(usedcw[i]==1) used_cw_nr+=1;
      }
```

```
        cprintf("(The number of used codewords in this codeword is %d )",
used_cw_nr);

        cprintf("Do you want to implement a new iteration?(Y/N)");
        kr=getche();

    /*****************************************
     *** Yeni kod kitabının oluşumu için      ***
     *** ağırlık merkezlerinin hesaplanması   ***
     *****************************************/

     if(kr=='y' || kr=='Y')  find_new_cb();

     if(kr=='n' || kr=='N'){
           gotoxy(5,10);
           cprintf("Do you want to save the result?(Y/N)");
           kr2=getche();
           if(kr2=='Y' | kr2=='y')
           saver();
           exit(1);
         }
    }


    if(vqerror<threshold){
      clrscr();
      gotoxy(5,5);
      cprintf("The mae is %4.2f and this last codebook will be saved!",vqerror);
      used_cw_nr=0;
      for(i=0;i<limit;i++){
           if(usedcw[i]==1) used_cw_nr+=1;
      }
      gotoxy(5,8);
      cprintf("The number of codewords in this codeword is %d",used_cw_nr);
      saver();
      exit(1);
    }


  } while (vqerror>threshold);

  saver();
}//MAIN'in sonu
```

```
/***********************************************
 ***********************************************
 ****                                       ****
 ****          KULLANILAN FONKSİYONLAR      ****
 ****                                       ****
 ***********************************************
 ***********************************************/


read_codebook()
{

   int c1,c2;

   if((finputcb=fopen(in_cb_name,"r"))==NULL){
                textcolor(RED);
                cprintf("\n Inputfile not opened!!!");
                cprintf("\n Press any key to continue...");
                getch();
                exit(1);
   }

   fscanf(finputcb,"%d",&limit);    // Number of codewords in the cb.
   cbmat=imatrix(0,limit-1,0,vectorsize-1);

   clrscr();
   gotoxy(5,5);
   cprintf("Reading the dif codebook");

   for(c1=0;c1<limit;c1++){
     gotoxy(5,5);
     cprintf("%d of %d              ",c1,limit);

     for(c2=0;c2<vectorsize;c2++) fscanf(finputcb,"%d",&cbmat[c1][c2]);
   }
   fclose(finputcb);
}//END OF READ_CODEBOOK

/*********************************************************************/

read_pr_codebook()
{

   int c1,c2;

   if((finputcb2=fopen(in_cb_name2,"r"))==NULL){
                textcolor(RED);
                cprintf("\n Inputfile not opened!!!");
                cprintf("\n Press any key to continue...");
```

```
                getch();
                exit(1);
    }


    clrscr();
    gotoxy(5,5);
    cprintf("Reading the primary codebook");


    fscanf(finputcb2,"%d",&limit2);    // Number of codewords in the cb.
    cbmat2=imatrix(0,limit2-1,0,vectorsize-1);

    for(c1=0;c1<limit2;c1++){
      gotoxy(5,5);
      cprintf("%d of %d                ",c1,limit2);

      for(c2=0;c2<vectorsize;c2++) fscanf(finputcb2,"%d",&cbmat2[c1][c2]);
    }
    fclose(finputcb2);
}//READ_PR_CODEBOOK sonu

/*****************************************************************/


/*Öğretici dizinin okunması */

read_trseq(int coef1,int end)
{

  int **indata;
  int coefnr,j,start;

  coefnr=end-coef1+1;

  /***** Matris tanımlamaları *****/
  indata=imatrix(0,1023,0,63);
  coefmat=imatrix(0,vectornumber-1,0,vectorsize-1);


  /***** Katsayıların okunması *****/

  if((finput=fopen("tif.qs2","r"))==NULL){
                textcolor(RED);
                printf("\n Inputfile not opened!!!");
                printf("\n Press any key to continue...");
                getch();
                exit(1);
  }
```

```
for(i=0;i<1024;i++){
    for(j=0;j<64;j++) fscanf(finput,"%d",&indata[i][j]);
  }

  fclose(finput);

  start=0;
  for(i=coef1;i<coef1+coefnr;i++){
    for(j=0;j<1024;j++)  coefmat[start+j/vectorsize][j%vectorsize]=indata[j][i];
  }

  if((finput=fopen("bab.qs2","r"))==NULL){
                textcolor(RED);
                printf("\n Inputfile not opened!!!");
                printf("\n Press any key to continue...");
                getch();
                exit(1);
            }

  for(i=0;i<1024;i++){
    for(j=0;j<64;j++) fscanf(finput,"%d",&indata[i][j]);
  }

  fclose(finput);

  start+=(1024*coefnr/vectorsize);
  for(i=coef1;i<coef1+coefnr;i++){
    for(j=0;j<1024;j++)  coefmat[start+j/vectorsize][j%vectorsize]=indata[j][i];
  }


  if((finput=fopen("len.qs2","r"))==NULL){
                textcolor(RED);
                printf("\n Inputfile not opened!!!");
                printf("\n Press any key to continue...");
                getch();
                exit(1);
            }

  for(i=0;i<1024;i++){
    for(j=0;j<64;j++) fscanf(finput,"%d",&indata[i][j]);
  }

  fclose(finput);

  start+=(1024*coefnr/vectorsize);
  for(i=coef1;i<coef1+coefnr;i++){
    for(j=0;j<1024;j++)  coefmat[start+j/vectorsize][j%vectorsize]=indata[j][i];
  }
```

```
  if((finput=fopen("sal.qs2","r"))==NULL){
                textcolor(RED);
                printf("\n Inputfile not opened!!!");
                printf("\n Press any key to continue...");
                getch();
                exit(1);
            }

  for(i=0;i<1024;i++){
    for(j=0;j<64;j++) fscanf(finput,"%d",&indata[i][j]);
  }

  fclose(finput);

  start+=(1024*coefnr/vectorsize);
  for(i=coef1;i<coef1+coefnr;i++){
    for(j=0;j<1024;j++)  coefmat[start+j/vectorsize][j%vectorsize]=indata[j][i];
  }


  if((finput=fopen("air.qs2","r"))==NULL){
                textcolor(RED);
                printf("\n Inputfile not opened!!!");
                printf("\n Press any key to continue...");
                getch();
                exit(1);
            }

for(i=0;i<1024;i++){
    for(j=0;j<64;j++) fscanf(finput,"%d",&indata[i][j]);
  }

  fclose(finput);

  start+=(1024*coefnr/vectorsize);
  for(i=coef1;i<coef1+coefnr;i++){
    for(j=0;j<1024;j++)  coefmat[start+j/vectorsize][j%vectorsize]=indata[j][i];
  }

  free_imatrix(indata,0,1023,0,63);

}//READ_TRSEQ  sonu
```

```
// İkincil Vektör Kuantalama uygulayan fonksiyon

apply_vq_lbg5(int vsize)
{
  int vnr,cwnr;
  int error,min_error;


  for(cwnr=0;cwnr<limit;cwnr++) usedcw[cwnr]=0;

  for(vnr=0;vnr<vectornumber;vnr++){
    gotoxy(5,7);
    cprintf("%d of %d                ",vnr,vectornumber);
    min_error=1000;
    for(cwnr=0;cwnr<limit;cwnr++){
      error=error_mae(vnr,cwnr,vsize);
      if(min_error>error) {
        min_error=error;
        used_vector[vnr]=cwnr;usedcw[cwnr ]=1;
       }
    }
  }

}
// APPLY_VQ_LBG5  sonu


// Birincil Vektör Kuantalama uygulayan fonksiyon

 apply_vq_lbg52(int vsize)
{
  int vnr,cwnr;
  int error,min_error;


  for(vnr=0;vnr<vectornumber;vnr++){
    gotoxy(5,6);
    cprintf("%d of %d                ",vnr,vectornumber);

    min_error=1000;
    for(cwnr=0;cwnr<limit2;cwnr++){
        error=error_mae2(vnr,cwnr,vsize);
        if(min_error>error) {min_error=error; used_vector[vnr]=cwnr;}
    }
  }

}
// APPLY_VQ_LBG52 sonu
```

```
// MAE olarak hatayı hesaplayan fonksiyon

int error_mae(int vecnr,int cwnbr,int vs)
{
  int c1;
  int mae=0,err;

  for(c1=0;c1<vs;c1++){
    err=coefmat[vecnr][c1]-cbmat[cwnbr][c1];
    if(err<0) err*=(-1);
    mae+=err;
  }
  return mae;
}// ERROR_MAE  sonu

/*********************************************************************/

// MAE olarak hatayı hesaplayan fonksiyon

int error_mae2(int vecnr,int cwnbr,int vs)
{
  int c1;
  int mae=0,err;

  for(c1=0;c1<vs;c1++){
    err=coefmat[vecnr][c1]-cbmat2[cwnbr][c1];
    if(err<0) err*=(-1);
    mae+=err;
  }
  return mae;
}// ERROR_MAE2  sonu

/*********************************************************************/


// Uygulanan VK'ya göre yeni katsayı değerlerini atayan fonksiyon.

assign_new_coef(int vsize)
{
  int vnr,c;

  for(vnr=0;vnr<vectornumber;vnr++){
    for(c=0;c<vsize;c++) newcoefmat[vnr][c]=cbmat[used_vector[vnr]][c];
  }
}// ASSIGN_NEW_COEF  sonu
```

```
// Uygulanan birincil VK'ya göre fark değerleri bulan fonksiyon.

subtract_new_coef(int vsize)

{
  int vnr,c;

  for(vnr=0;vnr<vectornumber;vnr++){
    for(c=0;c<vsize;c++) coefmat[vnr][c]-=cbmat2[used_vector[vnr]][c];
  }
}// SUBTRACT_NEW_COEF  sonu.

/*****************************************************************/


// VK için hatayı hesaplayan fonksiyon
double vq_error()
{
  int vnr,c;
  double  error1,sum=0;

 for(vnr=0;vnr<vectornumber;vnr++){
    for(c=0;c<vectorsize;c++){
        error1=coefmat[vnr][c]-newcoefmat[vnr][c];
        if(error1<0) error1*=(-1);
        sum+=error1;
    }
  }

  sum/=(double)(vectornumber);
  sum/=(double)(vectorsize);
  return sum;
}// VQ_ERROR  sonu

/*****************************************************************/

//  Oluşturulan kod kitabını kaydeden fonksiyon.

saver()
{
  int j;

  gotoxy(5,5);
  cprintf("Out of do-while loop because error=%4.2lf",vqerror);
  gotoxy(5,6);
  cprintf("Saving the new codebook(only the used vectors) as :");
  puts(out_cb_name);
```

```
  if((foutput=fopen(out_cb_name,"w"))==NULL){
    textcolor(RED);
    printf("\n Outputfile not opened!!!");
    printf("\n Press any key to continue...");
    getch();
    exit(1);
  }

  used_cw_nr=0;
        for(i=0;i<limit;i++){
           if(usedcw[i]==1) used_cw_nr+=1;
  }

  fprintf(foutput,"%d\n",used_cw_nr);

  for(i=0;i<limit;i++){
    if(usedcw[i]==1){
        for(j=0;j<vectorsize;j++)    fprintf(foutput,"%d\n",cbmat[i][j]);
    }
  }
}// SAVER  sonu

/*****************************************************************/

// Yeni kod kitabını oluşturan fonksiyon.

find_new_cb()
{

  int c1,c2,c3,c4;
  int vnr1;
  double sum[vectorsize];


  clrscr();
  for(c1=0;c1<vectornumber;c1++) used_vec[c1]=0;
  for(c1=0;c1<limit;c1++) usedcw[c1]=0;

    for(c1=0;c1<vectornumber;c1++){

        gotoxy(5,5);
        cprintf("%d  of %d      ",c1,vectornumber);

        if(used_vec[c1]==0){
          vnr1=0;
          for(c2=0;c2<vectorsize;c2++) sum[c2]=0.0;
            for(c4=0;c4<vectornumber;c4++){
              if(used_vector[c1]==used_vector[c4]){   // eğer her iki vektöre de aynı
                                                      kod kelimesi atanmışsa.
```

```
            for(c3=0;c3<vectorsize;c3++) sum[c3]+=coefmat[c4][c3];
            vnr1+=1;
            used_vec[c2]=1;
          }
        }
      for(c3=0;c3<vectorsize;c3++) sum[c3]/=(double)(vnr1);
      for(c3=0;c3<vectorsize;c3++)
         cbmat[used_vector[c1]][c3]=(int)(sum[c3]+.5);
      usedcw[used_vector[c1]]=1;
     }
  }
}// FIND_NEW_CB  sonu

/*********************************************************************/
```

3- Bu bölümde verilen program, DOSK için Lloyd-Max yöntemiyle kod kitabı optimizasyonunu gerçekleştiren programdır. Her bir AC katsayı için ayrı bir kod kitabı oluşturulmaktadır.

```
void main(void)
{
  int zz_order,qtzr_limit,min_limit,max_limit,start_nr,stop_nr;
  char oq_name[]="oq01.cb",nq_name[]="nq01.cb",stat_name[]="zz01.dat",yes_no;
  int stat_values[1000],assigned_values[1000];
  int orig_qtzr[64],new_qtzr[64];
  int i,j,OUT,min_error,error,loop_counter,stat_counter;
  double overall_error,error_before,overall_sum;



  /*** Sürece ait sayaç.  Bütün AC katsayılar taranıyor. ***/
  for(zz_order=1;zz_order<64;zz_order+=1){
    textcolor(RED);
    gotoxy(4,4);
    cprintf("ZZ_ORDER=%d  of 63  ",zz_order);

    error_before=5000;

    // Giriş çıkış dosyalarının isimlerinin oluşturulması
    oq_name[2]=48+(zz_order/10);
    oq_name[3]=48+zz_order-10*(zz_order/10);
    nq_name[2]=48+(zz_order/10);
    nq_name[3]=48+zz_order-10*(zz_order/10);
```

```
    stat_name[2]=48+(zz_order/10);
    stat_name[3]=48+zz_order-10*(zz_order/10);

    textcolor(YELLOW);
    gotoxy(5,5);
    cprintf("Opening the files %s and %s and obtaining %s",oq_name, stat_name,
nq_name);


    // Optimize edilecek başlangıç kod kitabının okunması
    if((f_orig_qtzr=fopen(oq_name,"r"))==NULL){
            printf("\n Original quantizer %s can not be opened!!!",oq_name);
            printf("\n Press any key to continue...");
            getch();
            exit(1);
    }
    fscanf(f_orig_qtzr,"%d",&qtzr_limit);
    for(i=0;i<qtzr_limit;i++) fscanf(f_orig_qtzr,"%d",&orig_qtzr[i]);
    fclose(f_orig_qtzr);


    // İstatistik bilginin okunması
    if((f_stat=fopen(stat_name,"r"))==NULL){
            printf("\n Statistical info file  %s can not be opened!!!",stat_name);
            printf("\n Press any key to continue...");
            getch();
            exit(1);
    }
    fscanf(f_stat,"%d",&min_limit);
    fscanf(f_stat,"%d",&max_limit);


    // İstatistik bilginin endeks numaralarına ötelenmesi (+500 öteleme!)
    start_nr=500+min_limit;  stop_nr=500+max_limit;


    for(i=start_nr;i<=stop_nr;i++)  fscanf(f_stat,"%d",&stat_values[i]);
    fclose(f_stat);

    /*********************************
     * Giriş bilgisinin işlenmesi tamamlandı.  *
     * Algoritma başlıyor.                     *
     *********************************/

    OUT=0;
    loop_counter=0;
    do{
          overall_error=0;
```

```
// İlk adım kuantalamanın uygulanması
for(i=start_nr;i<=stop_nr;i++){
  min_error=1000;
  for(j=0;j<qtzr_limit;j++){
       error=i-500-orig_qtzr[j];
       if(error<0) error*=(-1);
       if(error<min_error){min_error=error; assigned_values[i]=j;}
  }
}

cprintf("done");

// İkinci adım hatanın hesaplanması
for(i=start_nr;i<=stop_nr;i++){
  error=i-500-orig_qtzr[assigned_values[i]];
  if(error<0) error*=(-1);
  overall_error+=error*stat_values[i];
}
overall_error/=4096.0; //MAE

gotoxy(5,8);
cprintf("The current overall_error is %5.2lf and the error before was %5.2lf
",overall_error,error_before);

//Yeni bir iterasyon için karar
if(error_before-overall_error<.01) OUT=1;

if(OUT==0){
 if(loop_counter%10==9){
  gotoxy(5,9);
  cprintf("Loop_counter=%d . Shall I go on?(Y/N)",loop_counter);
  yes_no=getch();
  gotoxy(5,9);
  cprintf("                                          ");
  if(yes_no=='N' || yes_no=='n') OUT=1;
 }
}

// Hatanın, bir sonraki tur için, bir önceki hata değeri olarak saklanması
error_before=overall_error;

loop_counter++;

// Üçüncü adım ağırlık merkezlerinin hesaplanması
// ve bunların eski kuantalama değerlerinin yerine atanması.
// Bu eğer IF OUT=0 ise yapılacak.
if(OUT==0){
```

```
            for(i=0;i<qtzr_limit;i++)  new_qtzr[i]=0;
             for(i=0;i<qtzr_limit;i++){
                  stat_counter=0;
                  overall_sum=0.0;
                  //Ağırlık Merkezinin Hesaplanması
                  for(j=start_nr;j<=stop_nr;j++){
                    if(assigned_values[j]==i){
                      overall_sum+=(j-500)*stat_values[j];
                      stat_counter+=stat_values[j];
                    }
                  }
                  if(stat_counter!=0){
                   overall_sum=(double)(overall_sum/stat_counter);
                   new_qtzr[i]=(int)(overall_sum+.5);
                  }
             }
            }


            //Ağırlık Merkezleri belirlendi. EĞER yeni bir iterasyon yapılacaksa
            //kuantalama değerleri olarak atanacaklar.
            if(OUT==0)
              {for(j=0;j<qtzr_limit;j++) orig_qtzr[j]=new_qtzr[j];}

        }while(OUT==0);//ALGORITMADAN ÇIKIŞ


        // Kuantalayıcının kaydedilmesi.


        if((f_new_qtzr=fopen(nq_name,"w"))==NULL){
                printf("\n New quantizer %s can not be opened!!!",oq_name);
                printf("\n Press any key to continue...");
                getch();
                exit(1);
        }
        fprintf(f_orig_qtzr,"%d\n",qtzr_limit);
        for(i=0;i<qtzr_limit;i++) fprintf(f_new_qtzr,"%d\n",new_qtzr[i]);
        fclose(f_new_qtzr);
      }
}
```

4- DOSK kodlaması yapan program. Kuantalama değerleri programın içinde sabit değerlerle tanımlanmıştır. Program çıkışında, DC ve AC katsayıları iki ayrı dosyada kaydetmektedir.

```
void main()
{

  int cb[31],error,merror,c1,c2,zero_flag,zero_flag2;
  int byte_limit[1024],counter,block_nr,byte_nr;



  int dc_coef[68]={ 0,
                    10,-10,20,-20,
                    35,-35,50,-50,70,-70,90,-90,
                    110,-110,130,-130,150,-150,170,-170,
                    190,-190,210,-210,230,-230,250,-250,275,-275,
                    300,-300,330,-330,360,-360,390,-390,420,-420,
                    450,-450,480,-480,510,-510,540,-540,570,-570,600,-600,
                    640,-660,680,-720,720,-780,760,-840,800,-900,850,-960,
                    900,950,1000};


  int limit[64]={ 0,
                  25,25,
                  19,19,19,
                  9,
                  15,15,
                  15,15,15,15,15,15,
                  7,7,7,7,7,
                  7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
                  5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
                  3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3};

  int cb12[25]={ 0,
                 18,-18,36,-36,
                 48,-48,64,-64,-80,80,
                 96,-96,114,-114,
                 150,-150,191,-191,240,-240,291,-291,360,-360};

  int cb35[19]={ 0,
                 21,-21,
                 42,-42,63,-63,
                 84,-84,
                 100,-100,125,-125,161,-161,200,-200,251,-251};
```

```
int cb6[9]={ 0,
             47,-47,94,-94,
             141,-141,188,-188};

int cb78[15]={ 0,
              13,-13,26,-26,
              39,-39,52,-52,80,-80,121,-121,160,-160};

int cb914[15]={ 0,
               16,-16,32,-32,48,-48,
               64,-64,90,-90,131,-131,170,-170};

int cb1519[7]={ 0,
               22,-22,44,-44,85,-85};

int cb2034[7]={ 0,
               30,-30,60,-60,90,-90};

int cb3550[5]={ 0,51,-51,100,-100};

int cb5163[3]={0,90,-90};


byte_limit[0]=1024;


 // Giriş matrisinin tanımlanması
inmat=imatrix(0,63,0,1023);

for(i=0;i<1024;i++){
   for(j=0;j<64;j++){
      fscanf(finput,"%d",&inmat[j][i]);
    }
}


/****************************************
****************************************
****                                ****
****   İŞLEMİN BAŞLANGICI           ****
****                                ****
****************************************
****************************************/


/**** ÇIKIŞ MATRİSİ ****/
outmat=imatrix(0,63,0,1023);
for(i=0;i<64;i++){
  for(j=0;j<1024;j++) outmat[i][j]=0;}
```

```
/************************************
****    DC katsayıların kuantalanması    ****
************************************/

gotoxy(5,5);
cprintf("Processing DC's!                    ");

for(j=0;j<1024;j++){
        dummy=inmat[0][j];
        merror=1000;
        for(k=0;k<68;k++){
          error=dummy-dc_coef[k];
          if(error<0) error*=(-1);
          if(merror>error){merror=error; inmat[0][j]=k;}
        }
 }



/************************************
****    İkinci adım: DOSK'nın tüm       ****
****    AC katsayılara uygulanması      ****
************************************/


   for(i=1;i<=63;i++){

     if(i<=2)       { for (j=0;j<limit[i];j++) cb[j]=cb12[j]; }
     if(i>2 && i<6)  { for (j=0;j<limit[i];j++) cb[j]=cb35[j]; }
     if(i==6)       { for (j=0;j<limit[i];j++) cb[j]=cb6[j];  }
     if(i>6 && i<9)  { for (j=0;j<limit[i];j++) cb[j]=cb78[j]; }
     if(i>8 && i<15) { for (j=0;j<limit[i];j++) cb[j]=cb914[j];}
     if(i>14 && i<20){ for (j=0;j<limit[i];j++) cb[j]=cb1519[j];}
     if(i>19 && i<35){ for (j=0;j<limit[i];j++) cb[j]=cb2034[j];}
     if(i>34 && i<51){ for (j=0;j<limit[i];j++) cb[j]=cb3550[j];}
     if(i>50)       { for (j=0;j<limit[i];j++) cb[j]=cb5163[j];}


     for(j=0;j<1024;j++){
       dummy=inmat[i][j];
       merror=1000;
       for(k=0;k<limit[i];k++){
         error=dummy-cb[k];
         if(error<0) error*=(-1);
         if(merror>error){merror=error; inmat[i][j]=k;}
       }
     }
}
```

```
/*******************************************
 ***  DOSK tamamlandı. inmat matrisi kuantalama  ***
 ***  değerlerinin endekslerini taşıyor.          ***
 *******************************************/



 /*******************************************
  ****  JPEG benzeri Akış Uzunluklu Kodlama  ****
  *******************************************/

  for(j=0;j<1024;j++) outmat[0][j]=inmat[0][j];  //DC'lere ait endeksler
                                                  oldukları gibi çıkışa
                                                  aktarılıyor.

  for(block_nr=0;block_nr<1024;block_nr++){
      byte_nr=1;
      counter=0;
      c1=1;
      zero_flag=1;


   /** 63 elemanlı dizide tüm AC katsayıların sıfır olup olmadığının kontrolü **/
   do{
        if(inmat[c1][block_nr]!=0) zero_flag=0;
        c1++;
       }while(zero_flag==1 && c1<64);

      if(zero_flag==1){
       outmat[1][block_nr]=255; // "tüm AC'ler sıfır" anlamında özel karakter.
       byte_nr++;
      }

   /** 5 bit gerektiren ilk 5 katsayı için Akış Uzunluklu Kodlama **/
      if(zero_flag==0){
           for(i=1;i<5;i++){
               if(inmat[i][block_nr]==0){
                counter+=1;
               }
               else{
                     outmat[byte_nr][block_nr]=counter;
                     outmat[byte_nr][block_nr]<<=5;

      outmat[byte_nr][block_nr]=outmat[byte_nr][block_nr]+inmat[i][block_nr];
                     byte_nr+=1;
                     counter=0;
               }

           }
```

```
        /* Diğer katsayılar için Akış Uzunluklu Kodlama
          for(i=6;i<64;i++){
            c2=i;
            zero_flag2=1;

            do{
              if(inmat[c2][block_nr]!=0) zero_flag2=0;
              c2++;
            }while(zero_flag2==1 && c2<64);



            if(zero_flag2==1){
              outmat[byte_nr][block_nr]=127; // Geri kalan tüm AC'ler sıfır
                                                anlamında özel işaret.
              byte_nr++;
              i=100;
            }

            if(zero_flag2==0){

            if(inmat[i][block_nr]==0 && counter<15){
              counter+=1;
            }
            else{
                  outmat[byte_nr][block_nr]=counter;
                  outmat[byte_nr][block_nr]<<=4;

    outmat[byte_nr][block_nr]=outmat[byte_nr][block_nr]+inmat[i][block_nr];
                  byte_nr+=1;
                  counter=0;
            }
            }// eğer (zero_flag2==0) ise sonlandır.
        }



    }//eğer (zero_flag==0) ise sonlandır.
    byte_limit[block_nr]=byte_nr;

}
```

```
    if((foutput=fopen(outfile,"wb"))==NULL){
            printf("\n Outputfile-2 not opened!!!");
            printf("\n Press any key to continue...");
            getch();
            exit(1);
    }


    for(i=0;i<1024;i++){
        for(j=1;j<byte_limit[i];j++) fputc(outmat[j][i],foutput);
    }

    fclose(foutput);

    if((foutput=fopen("dc.dat","wb"))==NULL){
            printf("\n Outputfile-2 not opened!!!");
            printf("\n Press any key to continue...");
            getch();
            exit(1);
    }


    for(i=0;i<1024;i++) fputc(outmat[0][i],foutput);
    fclose(foutput);
}
```

ÖZGEÇMİŞ

Vadi Dipçin 1971'de İstanbul'da doğdu. 1989'da İstanbul Lisesi'nden mezun olduktan sonra, aynı yıl İstanbul Teknik Üniversitesi Elektronik ve Haberleşme Mühendisliği Bölümü'e girdi. 1994'te lisans öğrenimini tamamladı ve İTÜ Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Programı'nda yüksek lisans çalışmasına başladı. Halen İTÜ Video Eğitim Merkezi'nde araştırma görevlisi olarak çalışmaktadır.