NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Electrical and Electronic Engineering

DIGITAL IMAGE COMPRESSION & RESTORATION

Graduation Project
EE- 400

Student: Murad Khader (20042493)

Supervisor: Dr. Ali Serener

Nicosia - 2008

ACKNOWLEDGMENTS

This work would not have been possible without the generous help of God and then the following people as well as their significant contribution to my work.

Dr. Ali SERENER: I would like to sincerely thank for his invaluable supervision, support and encouragement through this work and for introducing me to the world of communications. Also his suggestions and instructions through the under graduate years and for always being kind and helpful to me all these years.

I would like to express my sincere thanks and dedication especially, to my parents and family and gift them this work for their always constant love and supporting, spiritual and financial, in my decisions through the years.

I wish to thank the administration of Near East University for the opportunity to have a successful education.

Finally, I would like to thank my friends specially Tariq Farid & Ahmed Osman for their help in Matlab programming and their helpful ideas.

Murad G.Khader

ABSTRACT

Every digital communication system basically consists of a transmitter, a channel, and a receiver. This system is a means of transporting information from one part to another . The system is described to be digital which means it uses a sequence of symbols from a finite alphabet to represent the information. The transmission of data in digital form allows for the use of a number of powerful signal processing techniques that would otherwise be unavailable.

Digital images are applications for such systems which consist of pixels. Each pixel carries the information of colors at the point it is located at. Pixels all together form the digital image that the human eye can see on the screen of a digital imaging system like a digital camera, computer, or even a digital TV receiver which converts the pixels into analogue waves to be shown on the TV screen.

Time consuming and response to noise are more likely to happen due to the large amount of information the transmitted image contains. For such problems image compression is used.

A digital image transmitted through the channel could be corrupted by various factors called noises, which represent unwanted signals. The clearness of the image depends on how much noise the image is effected by.

Hence, for such a problem a good technique is introduced to get rid of noise totally or reduce the amount of noise; this technique is known as image restoration. Image restoration is used to improve the appearance of the received image that has been degraded (blurred) by using a priori knowledge of the degradation phenomenon.

In this project a graphical user interface (GUI) is created which shows the original image before transmission, degradation of the image using different types of noise and its reconstruction after filtering.

TABLE OF CONTENTS

INTRODUCTION

Traditional communication systems are made up of three major components: the transmitter, the channel and the receiver. The transmitter transmits a signal (or an image) across a noisy channel which introduces distortion to that image. The receiver receives the distorted image and attempts to recover the original image.

The reconstruction process of the corrupted image would be possible that is if a prior knowledge about the degradation model is known. This can be done by different kinds of filtering.

Since the noise behavior presented within an image varies from one to another, then the restoration technique is going to be the one which suits that type of noise, thus amount of noise could be reduced as much as possible.

Images often require a large number of bits to represent them, and if the image needs to be transmitted or stored, it is impractical to do so without somehow reducing the number of bits. The problem of transmitting or storing an image affects all of us daily. TV and fax machines are both examples of image transmission, and digital video players and web pictures are examples of image storage. thus image compression is used to minimize the amount of memory needed to represent an image.

A simple image restoration system is created using graphical user interface. This system can deal with the noise introduced within transmitted image and reduce its amount to a level that makes the image clear enough to be recognized by the human eye or a further image processing machine.

Chapter 1 describes digital images, classification of them, the noise that could result during transmission through the channel, and brief description about the methods of filtering the noise that will be discussed more in the 2^{nd} chapter.

Chapter 2 devoted to image restoration techniques, filtering methods due to corruption through channel and their characteristics , also talks about types of noise.

Chapter 3 will illustrate image compression technique, the benefit of compression. Also describes systems of compression; lossy compression & information preserving compression.

Chapter 4 shows results of a simple image restoration system & simulation of image compression.

CHAPTER ONE

DIGITAL IMAGES

1.1 What is an Image?

An image is a representation of a real scene. It can be classified into two categories; either in black and white or in color form, or in print or digital form. Printed images may have been reproduced either by multiple colors and grayscale or by a single ink source [1].

1.2 What is Digital image?

Digital images consist of pixels. Each pixel carries the information of colors at the point it is located at. Pixels all together forms the digital image that our eyes can see on the screen of a digital imaging system like a digital camera, computer, or even a digital TV receiver which converts these pixels into analogue waves to be shown on the TV screen [1].

1.3 Classification of Images

1.3.1 RGB Images

In colored images each pixel has three color vectors one for green, one for red, and one for blue. Each of those color vectors has a value between 0 and 255 representing the concentration of the color within the pixel. Those three-color vectors within the pixel are mixed to give different colors that human eyes can see like (white, red, black, violet, brown, ect.). We can control the brightness of a colored image by controlling the value of the color vectors of the image pixels. If we decrease all pixel values together the colors in the image are going to be decreased which means that the image will be come darker while if we increase the color vectors of the pixels of an image, the image will be come brighter. We can also control the contrast, which is the degree of color mixing between pixels of an image.

Figure 1.1 shows how color vectors in each pixels are represented. The squares are pixels, all together forming the digital image that a human eye can see. This pixels image (8x8 pixels) is a simple digital image while in reality digital images consist of much more than this number of pixels. For example in mobile cellular phone cameras number of pixels might reach 3 mega pixels and even more. As mentioned before each

color vector in the pixel of a colored image between 0 and 255. That means we have 256 possible values and each color vector can be represented in 8-bit binary form data. In order to represent one pixel of a colored image we need 24 bits to represent the three color vectors of the image.

To understand more how colored image is displayed we can imagine three images: red scale image, blue scale image, and a green scale image. Placed on each other to perform a color mixing, giving the RGB-image.

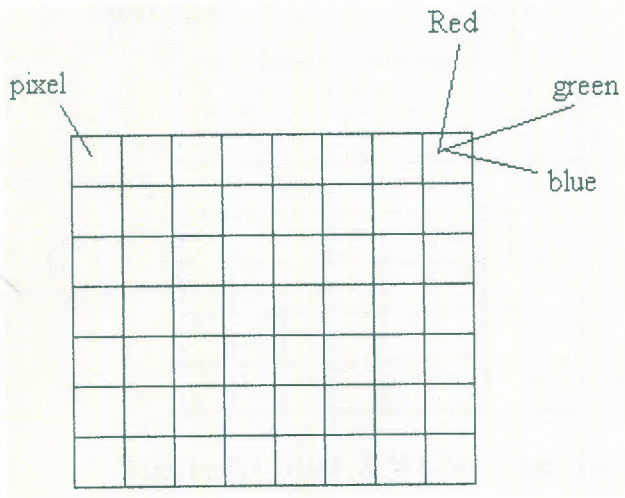


Figure 1.1 Pixels in an image

1.3.2 Gray Level Images

Gray scale images had only one color vector. The value of this color vector represents the level of the gray color in the pixel that varies between 0 for very dark gray (black) and 255 which is very bright gray (white).

As long as we have only one color vector represented with 8-bits of data in each pixel then we can represent each pixel with an 8-bits data representing the gray color value in it. We can increase the brightness of a grayscale image by increasing all the pixel values together, which will increase the color level in all the pixels uniformly. Increasing the contract of a grayscale image will increase the color mixing between the pixels [1].

1.3.3 Black and White Images

Black and white images, also called binary images, are the first digital images found. Each pixel is formed of pure black color or pure white color. The pixel value 0 represents pure black and represent pure white. Each pixel in the black and white image can have two color possibilities, black or white, 0 or 1. Therefore we only need 1-bit data to represent a pixel in a black and white images.

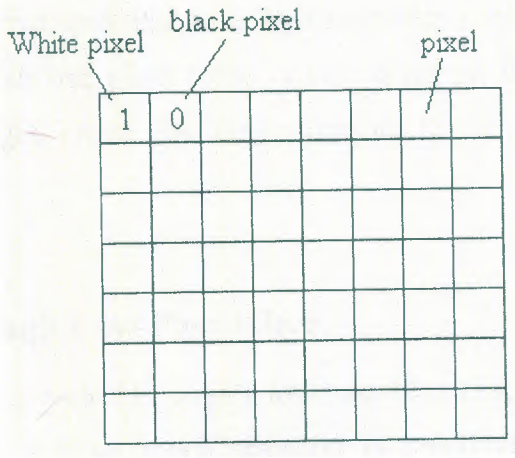


Figure 1.2 Black & White image pixels.

1.4 Edges and Noise in Digital Images

Consider sequentially plotting the gray values of the pixels of an image on the (y-axis) against pixel locations on the (x-axis) as the image is scanned. The result will be a discrete time plot of varying amplitudes showing the intensity of light at each pixel. Lets say that we are at the sixth row and looking at pixels from 1 to 86 at a 256 pixels image (16x16). The intensities of the pixels number 82 and 88 are so different from pixels around them and maybe considered to be noise.

In general, noise is information that does not belong to the surrounding environment. The intensities of pixels 86 and 90 are also different from the neighborhood pixels and may indicate a transition between the object and the background; thus these pixels can be constructed as representing the edges of the object.

1.5 Image Filtering and Noise Reduction

Although we are discussing a discrete signal, this signal may be transformed into a large number of sines and cosines with different amplitudes and frequencies that if added together, will construct a signal. Slowly changing signals (such as small changes between succeeding pixel gray values) will require few sines and cosines in order to be constructed., and thus have low frequency content. On the other hand quickly varying signals (such as large differences between pixel gray levels) will require many more frequencies to be reconstructed, and thus have high frequency content. Both noises and edges are instances in which one pixel value is very different from the neighboring ones, thus noises and edges create the larger frequencies of a typical frequency spectrum.

1.5.1 Image Through Low-Pass Filter

If a large frequency is passed through a low pass filter (a filter that allows lower frequencies to go through without much attention in amplitude, but that severely attenuates the amplitudes of the higher frequencies in the signal) the filter will reduce the influence of all high frequencies, including the noises and edges. This means that, although a low pass filter will reduce noise, it will also reduce the clarity of an image by attenuating the edges, thus softening the image throughout.

1.5.2 Image Through High-Pass Filter

A high-pass filter will increase the apparent effect of higher frequencies by severely attenuating the low-frequency amplitudes. In such cases, noises and edges will be left alone, but slowly changing areas will disappear from the image.

1.5.3 Neighborhood Averaging

Neighborhood averaging can be used to reduce noise in an image, but it also reduces the sharpness of the image. Consider the (3x3) mask shown in figure 1.3 together with it corresponding values, as a portion of an imaginary image with its gray levels indicated. As it can be seen, all the pixels but one are at gray value of 20. The

pixel with a gray level of 100 may be considered to be noise, since it is different from the pixels around it. Applying the mask over the corner of the image with a normalizing value of 9 (the sum of all values in the mask), yields :

$$X = (20\times1 + 20\times1 + 20\times1 + 20\times1 + 100\times1 + 20\times1 + 20\times1 + 20\times1 + 20\times1)/9 = 29$$

20	20	20	20	
20	100	20	20	
20	20	20	20	

1	1	1
1	1	1
1	1	1

Figure 1.3 Neighborhood averaging mask.

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

1	2	1
2	4	2
1	2	1

Figure 1.4 (5x5) & (3x3) Gaussian averaging filters.

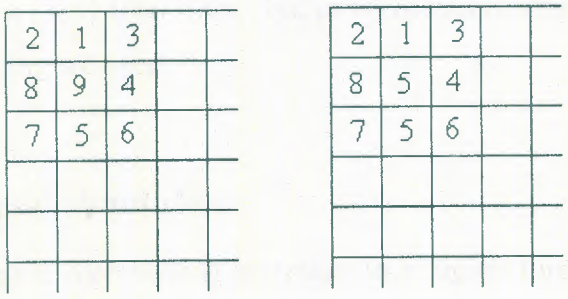
As a result of applying the mask on the indicated corner, the pixel with the value 100 changes to 29 and the large difference between the noisy pixels and the surrounding pixels (100 vs. 20) becomes much smaller (29 vs. 20), thus reducing the noise. With this characteristic, the mask acts as a low pass filter. This averaging low-pass filter will also reduce the sharpness of edges, making the resulting image softer and less focused. We can also use (5x5) filters, which give better results, but require a bit more processing. There are other averaging filters, such as the Gaussian averaging filter (also called the mild isotropic low-pass filter), which is shown in Figure 1.4. This filter will similarly improve an image, but with slightly different results.

1.5.4 Frequency Domain

When the Fourier transform of an image is calculated, the frequency spectrum might show a clear frequency of the noise, which in many cases can be selective eliminated by proper filtering.

1.5.5 Median Filters

One of the main problems in using neighborhood averaging is that, along with removing noises, the filter will blur edges. A variation of this technique is to use a median filter, in which the value of the pixel is replaced by the median of the values of the pixels in a mask around the given pixel, stored in ascending order. A median is the value such that half of the values in the set are below and half are above the median. Since, unlike an average, the median is independent of the value of any single pixel in the set. The median filter will be much stronger in eliminating spike-like noises without burring the object or decreasing the overall sharpness of the image. Consider the image in figure 1.5(a). The gray values, in ascending order, are 1,2,3,4,5,6,7,8 and 9. The middle value is 5, resulting in the image (b) in figure 1.5. Observe that the image has become grainy because the pixel sets with similar values appear longer (as in 5 and 5).



(a) (b)

Figure 1.5 Application of a median filter.

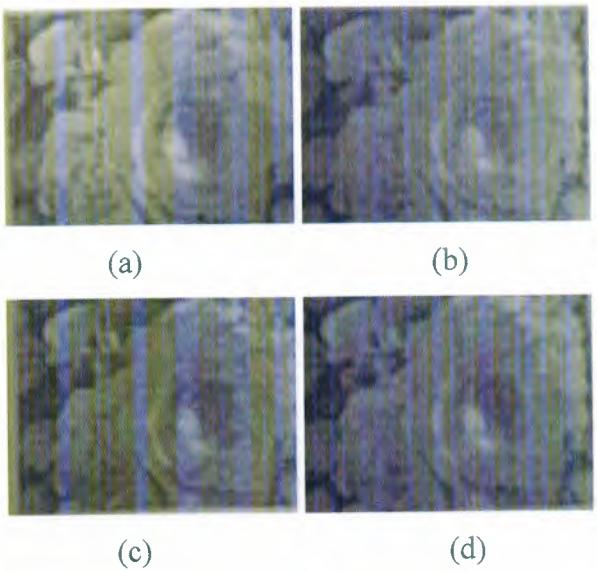


Figure 1.6 (a) original image. (b) image corrupted with a random Gaussian noise. (c) image improved by a (3x3) median filter. (d) same image improved by (7x7) median filter.

In figure 1.6 we can see how that the effect of a (7x7) filter on a corrupted image is better than the effect of a (3x3) filter mask, but as we mentioned before the bigger the mask we use more processing we need.

1.6 Image Sampling and Resolution

Sampling an image is representing an image in a digital form (pixels), as each pixel of the image represents a part of it. As we mentioned before the pixel is the smallest piece of a digital image.

The resolution of an image is the number of samples forming it. If we sample an image of an object at a rate of 64-kilo pixels (having 65536 pixels forming the image), and we sample an image of the same object at a rate of 256 pixels (256 pixels forming the image), the image sampled at 64-kilo pixel resolution is going to be more clear than the one sampled at 256 pixels resolution (Figure 1.7).

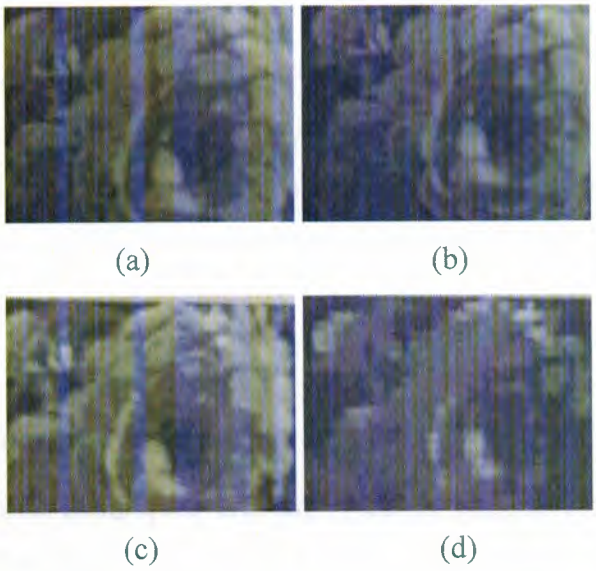


Figure 1.7 Effect of different sampling rates on an image. At (a) 432 x 576 pixels, at (b) 108x144 pixel, at (c) 54x72 pixels, (d) 27x36 pixels. As the resolution decreases, the clarity of the image diminishes accordingly.

1.7 Storage of Digital Images

After the image is converted to its digital form by any imaging device (scanner, digital camera, etc) the image can be stored in a hard disk or any other storage device in different formats. Each has different properties and advantages. Some compression methods discard too much information in order to have small file size. This type of compression causes a lot of information loss, so when we attend to view the image again, we are not going to have an image with the same quality of the original. Since loss less compression does not discard much information, we are not going to have much reduction in the image size, but at the same time we will keep the quality of the image. Some image file formats do not do any kind of compression, like. Raw files (uncooked images). They just discard some information related to the printer and some other applications.

1.8 Image Restoration

Image restoration methods are used to improve the appearance of an image by application of restoration process that uses a mathematical model for image degradation. The modeling of the degradation process differentiates restoration from enhancement where no such model is required.

Examples of the types of degradation considered include blurring caused by motion or atmospheric disturbance, geometric distortion caused by imperfect lenses, superimposed interference patterns caused by mechanical systems, and noise from electronic sources. It is assumed that the degradation model is known, or can be estimated, the degradation process is modeled and its inverse is applied to restore the original image.

1.9 Image Enhancement

Image enhancement techniques are employed to emphasize, sharpen, and/or smooth image features for display and analysis. Image enhancement is the process of applying these techniques to facilitate the development of solution to a computer imaging problem, consequently, the enhancement methods are application specific and are often developed empirically. Enhancement methods operate in the spatial domain manipulating the pixel data, or in the frequency domain by modifying the spectral components, some enhancement algorithms use both.

The type of technique include point operations, where each pixel is modified according to a particular equation that is not dependent on other pixel values, mask operations where each pixel is modified according to the values in a small neighborhood (sub image), or global operations where all the pixel values in the image are taken into consideration. Spatial domain processes include all three types, but frequently domain operations, by nature of the frequency (and sequence) transform are global operations, and can become mask operations by performing the transform on small image blocks instead of the entire image.

Examples of image enhancement techniques include edge detection, modifying the brightness and/or the sharpness of the image, increasing or decreasing the image contrast and many other possible processes.

CHAPTER TWO

IMAGE RESTORATION TECHNIQUES

2.1 Overview

The principal objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application.

Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term spatial domain refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. Frequency domain processing techniques are based on modifying the Fourier transform of an image [2].

Similar in image enhancement, the ultimate goal of restoration techniques is to improve an image in some predefined sense. Although there are areas of overlap, image enhancement is largely a subjective process, while image restoration is for the most part an objective process. Restoration attempts to reconstruct or recover an image that has been degraded by using a priori knowledge of the degradation phenomenon. Thus restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image [3].

The material developed in this chapter is strictly introductory, considering the restoration problem only from the point where a degraded, digital image is given. Some restoration techniques are best formulated in the spatial domain, while others are better suited for the frequency domain. For example, spatial processing is applicable when the only degradation is additive noise. On the other hand, degradations such as image blur are difficult to approach in the spatial domain using small masks. In this case, frequency domain filters based on various criteria of optimality are the approaches of choice [3].

Since the only degradation considered in this report is the additive white Gaussian noise, this chapter focuses only on spatial domain filtering processes.

2.2 Background

As indicated previously, the term spatial domain refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on these pixels. Spatial domain processes will be denoted by the expression

$$g(x,y)=T[f(x,y)] \tag{2.1}$$

where f(x, y) is the input image, g(x, y) is the processed image, and T is an operator on f, defined over some neighborhood of (x, y). In addition, T can operate on a set of input images, such as performing the pixel-by-pixel sum of K images for noise reduction. The principal approach in defining a neighborhood about a point (x, y) is to use a square or rectangular sub image area centered at (x, y), as Figure 2.1 shows.

The center of the sub image is moved from pixel to pixel starting at the top left corner. The operator T is applied at each location (x, y) to yield the output, g, at that location. The process utilizes only the pixels in the area of the image spanned by the neighborhood. Although other neighborhood shapes, such as approximations to a circle, sometimes are used, square and rectangular arrays are by far the most predominant because of their ease of implementation [2].

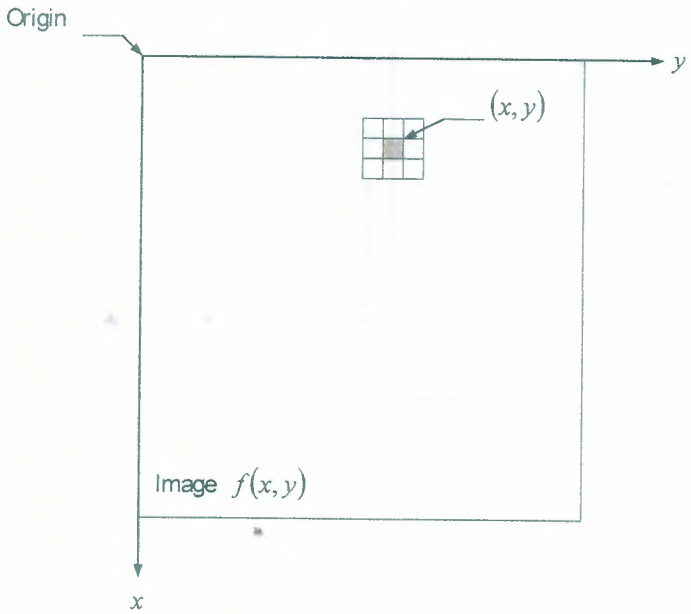


Figure 2.1 A 3×3 Neighborhood About a Point (x, y) in an Image.

The simplest form of T is when the neighborhood is of size 1×1 (that is, a single pixel). In this case, g depends only on the value of f at (x, y), and T becomes a gray-level (also called an intensity or mapping) transformation function of the form

$$s=T(r) \tag{2.2}$$

where, for simplicity in notation, r and s are variables denoting, respectively, the gray level of f(x, y) and g(x, y) at any point (x, y). For example, if T(r) has the form shown

in Figure 2.2(a), the effect of this transformation would be to produce an image of higher contrast than the original by darkening the levels below m and brightening the levels above m in the original image.

In this technique, known as contrast stretching, the values of r below m are compressed by the transformation function into a narrow range of s, toward black. The opposite effect takes place for values of r above m. In the limiting case shown in Figure 2.2(b), T(r) produces a two-level (binary) image. A mapping of this form is called a threshold function. Some fairly simple, yet powerful, processing approaches can be formulated with gray-level transformations. Because enhancement at any point in an image depends only on the gray level at that point, techniques in this category often are referred to as point processing.

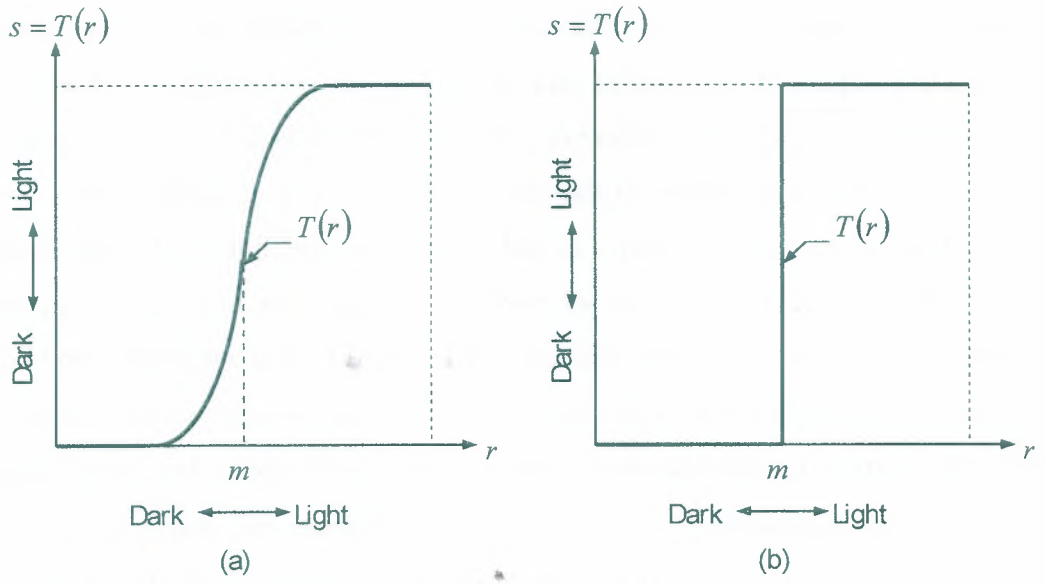


Figure 2.2 Gray level Transformation Functions for Contrast Enhancement.

Larger neighborhoods allow considerably more flexibility. The general approach is to use a function of the values of f in a predefined neighborhood of (x, y) to determine the value of g at (x, y). One of the principal approaches in this formulation is based on the use of so-called masks (also referred to as filters, kernels, templates, or windows). Basically, a mask is a small (say, 3×3) 2-D array, such as the one shown in Figure 2.1, in which the values of the mask coefficients determine the nature of the process, such as

image sharpening. Enhancement techniques based on this type of approach often are referred to as mask processing or filtering [2].

2.3 Order-Statistics Filters

Order-statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.

The best-known example in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median). Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.

The median, ξ, of a set of values is such that half the values in the set are less than or equal to ξ, and half are greater than or equal to ξ. In order to perform median filtering at a point in an image, we first sort the values of the pixel in question and its neighbors, determine their median, and assign this value to that pixel. For example, in a 3×3 neighborhood the median is the 5th largest value, in a 5×5 neighborhood the 13th largest value, and so on. When several values in a neighborhood are the same, all equal values are grouped. For example, suppose that a 3×3 neighborhood has values (10, 20, 20, 20, 15, 20, 20, 25, 100). These values are sorted as (10, 15, 20, 20, 20, 20, 20, 25, 100), which results in a median of 20. Thus, the principal function of median filters is to force points with distinct gray levels to be more like their neighbors. In fact, isolated clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than $n^2/2$ (one-half the filter area), are eliminated by an $n\times n$ median filter. In this case "eliminated" means forced to the median intensity of the neighbors. Larger clusters are affected considerably less [3].

Although the median filter is by far the most useful order-statistics filter in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but ranking lends itself to many other

possibilities. For example, using the 100th percentile results in the so-called max filter, which is useful in finding the brightest points in an image. The response of a 3×3 max filter is given by $R=\max\{z_k|k=1,2,\ldots,9\}$. The 0th percentile filter is the min filter, used for the opposite purpose. Median, max, and mean filters are considered in more detail in section 2.6.1 [2].

2.4 A Model of the Image Degradation/Restoration Process

As Figure 2.4 shows, the degradation process is modeled in this chapter as a degradation function that, together with an additive noise term, operates on an input image f(x,y) to produce a degraded image g(x,y). Given g(x,y), some knowledge about the degradation function H, and some knowledge about the additive noise term q(x, y), the objective of restoration is to obtain an estimate f(x, y) of the original image. We want the estimate to be as close as possible to the original input image and, in general the more we know about H and η, the closer g(x,y) will be to f(x,y). The approach used throughout most of this chapter is based on various types of image restoration filters.

If H is a linear, position-invariant process, then the degraded image is given in the spatial domain by

$$g(x,y)=h(x,y)*f(x,y)+\eta(x,y) \tag{2.3}$$

where h(x, y) is the spatial representation of the degradation function and the symbol * indicates convolution. The convolution in the spatial domain is equal to multiplication in the frequency domain, so the model in equation (2.3) might be written in an equivalent frequency domain representation:

$$G(u,v)=H(u,v)F(u,v)+N(u,v) \tag{2.4}$$

where the terms in capital letters are the Fourier transforms of the corresponding terms in Equation (2.3).

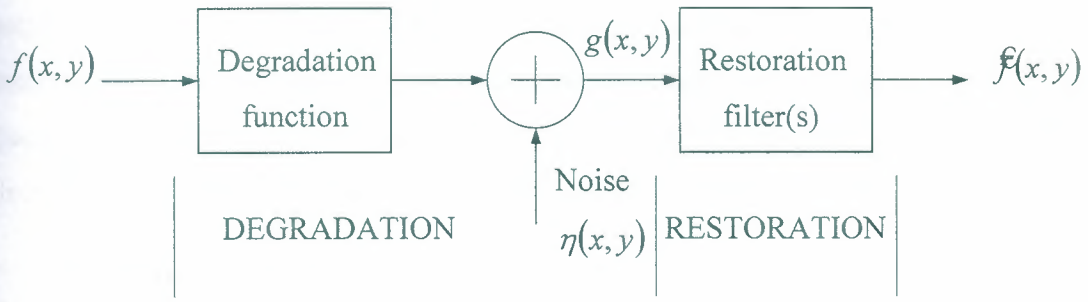


Figure 2.4 A model of the Image Degradation/ Restoration Process.

In the following two sections, we assume that H is the identity operator, and we deal only with degradations due to noise [2].

2.5 Noise effect on digital communication systems

2.5.1 What is noise?

During transmission in communication channels if noise power is high, it might cause changes inn transmitted information that ends up at the information sink as different information then the ones supposed to be received. For example if binary sequence is being sent through the physical channel, it will be received in the following way:

If $F(t) \geq 0$; then transmitted symbol is A

If $F(t) < 0$; the transmitted symbol is B

Here if we let $F(t) = -0.3$, then the transmitted symbol is B, but what happens if we have noise added to that signal is that if $n(t) = 0.4$ the result will be $n(t) + F(t) = 0.1$ which gives as another symbol of the alphabet which is A. In order for this to happen the noise power should be high enough to effect the transmitted information so we have a choice of transmitting with a high power. We also have filtering, but that wouldn't be so efficient for some types of noise.

The term noise is used customarily to designate unwanted waves that tend to disturb the transmission and processing of signals in communication systems. As we know we have different types of noise sources like atmospheric noise, galactic noise or man-made noise those ones are external noise types but we also have another type that

is the internal type of noise that arises form spontaneous fluctuations of current or voltage in electrical circuit. This type of noise represents a basic limitation on the transmission or detection of signals in communication systems involving the use of electronic devices. The two most common examples of spontaneous fluctuations in electrical circuits are shot noise and thermal noise.

2.5.2 Shot Noise

Shot noise arises in electronic devices such as diodes and transistors because of the discrete nature of the current flow in the devices for example in a photo detector circuit a current pulse is generated every time an electron is emitted by the cathode due to incident times denoted by τ as $-\infty < \tau < \infty$. It is assumed that the random emissions of electrons have been going on so for long time. Thus the total current flowing through the photo detector may be modeled as infinite sum of current pulses.

2.5.3 Thermal Noise

Thermal noise is the name given to the electrical noise arising from the random motions of electrons in the conductor. It is also known that the thermal noise is Gaussian distributed with zero mean.

2.5.4 White Noise

The noise analysis of communication systems is customarily based on an idealized form of noise called white noise, the power spectral density of which is independent of the operating frequency. The adjective white is used in the sense that white light contains equal amounts of all frequencies within the visible band of electromagnetic radiation. We express the power spectral density of white noise, with a simple function denoted by $\omega(t)$, as $S(f) = N/2$. The parameter N (the dimensions of N are in watts per hertz), is usually referred to the input stage of the receiver of a communication system, and may be expressed as $N=kT$, where k is Boltzmann's constant and T is the equivalent noise temperature of the receiver. The white Gaussian noise represents the ultimate in randomness; it has an infinite average power and not physically realizable.

2.5.5 Noise Models

The principal sources of noise in digital images arise, during image acquisition (digitization) and/or transmission. The performance of imaging sensors is affected by a variety of factors, such as environmental conditions during image acquisition, and by the quality of the sensing elements themselves. For instance, in acquiring images with an camera, light levels and sensor temperature are major factors affecting the amount of noise in the resulting image. Images are corrupted during transmission principally due to interference in the channel used for transmission. For example, an image transmitted using a wireless network might be corrupted as a result of lightning or other atmospheric disturbance [3].

2.5.5.1 Spatial and Frequency Properties of Noise

Relevant to this discussion are parameters that define the spatial characteristics of noise, and whether the noise is correlated with the image, Frequency properties refer to the frequency content of noise in the Fourier sense (i.e., as opposed to the electromagnetic spectrum). For example, when the Fourier spectrum of noise is constant, the noise usually is called white noise. This terminology is a carry over from the physical properties of white light, which contains nearly all frequencies in the visible spectrum in equal proportions. It is not difficult to show that the Fourier spectrum of a function containing all frequencies in equal proportions is a constant. With the exception of spatially periodic noise, noise is independent of spatial coordinates, and that it is uncorrelated with respect to the image itself (that is, there is no correlation between pixel values and the values of noise components) [3].

2.5.5.2 Gaussian Noise

Gaussian noise is a very good approximation of noise that occurs in many practical cases. Probability density of the random variable is given by the Gaussian curve. Based on the assumptions in the previous section, the spatial noise descriptor with which it shall be concerned is the statistical behavior of the gray-level values in the noise component of the model in Figure 2.4. These may be considered random variables, characterized by a probability density function (PDF).

Because of its mathematical tractability in both the spatial and frequency domains, Gaussian (also called normal) noise models are used frequently in practice. In

fact, this tractability is so convenient that it often results in Gaussian models being used in situations in which they are marginally applicable at best. Figure 2.5 shows the Gaussian probability density function.

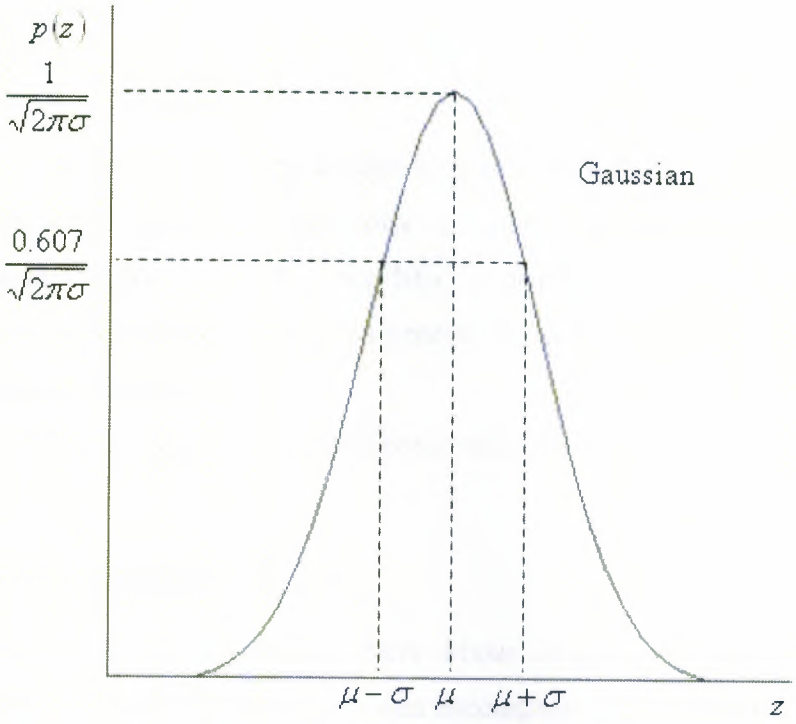


Figure 2.5 Gaussian Probability Density Function.

The PDF of a Gaussian random variable, z is given by

$$p(z)=\frac{1}{\sqrt{2\pi\sigma}}e^{-(z-\mu)^2/2\sigma^2} \tag{2.5}$$

where z represents gray level, μ is the mean of average value of z, and σ is its standard deviation. The standard deviation squared, σ^2, is called the variance of z. A plot of this function is shown in Figure 2.5. When z is described by equation (2.5), approximately 70% of its values will be in the range $[(\mu-\sigma),(\mu+\sigma)]$, and about 95% will be in the range $[(\mu-2\sigma),(\mu+2\sigma)]$ [2].

2.5.5.3 Impulse (salt-and-pepper) noise

The PDF of (bipolar) impulse noise is given by

$$P(z)=\begin{cases}Pa & \text{for } z=a\\ Pb & \text{for } z=b\\ 0 & \text{Otherwise}\end{cases} \tag{2.6}$$

2.6 Restoration in the Presence of Noise Only-Spatial Filtering

When the only degradation present in an image is noise, Equations (2.3) and (2.4) become

$$g(x,y) = f(x,y) + \eta(x,y) \tag{2.7}$$

and

$$G(u,v) = F(u,v) + N(u,v). \tag{2.8}$$

The noise terms are unknown, so subtracting them from g(x, y) or G(u, v) is not a realistic option. In the case of periodic noise, it usually is possible to estimate N(u, v) from the spectrum of G(u, v). In this case N(u, v) can be subtracted from G(u, v) to obtain an estimate of the original image. In general, however, this type of knowledge is the exception, rather than the rule.

Spatial filtering is the method of choice in situations when only additive noise is present.

2.6.1 Order-Statistics Filters

Order-statistics filters are spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter. The response of the filter at any point is determined by the ranking result [3].

2.6.1.1 Median Filter

The best-known order-statistics filter is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel

$$\hat{f}(x,y) = \underset{(s,t) \in S_{xy}}{\text{median}}\{g(s,t)\}. \tag{2.9}$$

The original value of the pixel is included in the computation of the median. Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of both bipolar and unipolar impulse noise [3]. In fact, the median filter yields excellent results for images corrupted by this type of noise.

2.6.1.2 Max and Min Filters

Although the median filter is by far the order-statistics filter most used in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but the reader will recall from basic statistics that ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called max filter, given by

$$\hat{f}(x,y) = \max_{(s,t) \in S_{xy}} \{g(s,t)\}. \tag{2.10}$$

This filter is useful for finding the brightest points in an image. Also, because pepper noise has very low values, it is reduced by this filter as a result of the max selection process in the sub image area S_{xy}.

The 0th percentile filter is the min filter

$$\hat{f}(x,y) = \min_{(s,t) \in S_{xy}} \{g(s,t)\}. \tag{2.11}$$

This filter is useful for finding the darkest points in an image. Also, it reduces salt noise as a result of the min operation [3].

2.6.1.3 Midpoint Filter

The midpoint filter simply computes the midpoint between the maximum and minimum values in the area encompassed by the filter

$$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t) \in S_{xy}} \{g(s,t)\} + \min_{(s,t) \in S_{xy}} \{g(s,t)\}\right]. \tag{2.12}$$

Note that this filter combines order statistics and averaging. This filter works best for randomly distributed noise, like Gaussian or uniform noise [3].

CHPTER THREE

IMAGE COMPRESSION

3.1 Preview

Methods of compressing the data prior to storage and/or transmission are of significant practical and commercial interest. Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image.

At some later time, the compressed image is decompressed to reconstruct the original image or an approximation of it. The initial focus of research efforts in this field was on the development of analog methods for reducing video transmission bandwidth, a process called bandwidth compression [3].

International image compression standards, the field has undergone significant growth through the practical application of the theoretic work that began in the 1940s, when C. E. Shannon and others, first formulated the probabilistic view of information and its representation, transmission, and compression.

Image compression plays a major role in many important and diverse applications, including Tele-video conferencing, remote sensing (the use of satellite imagery for weather and other earth-resource applications), document and medical imaging, facsimile transmission (FAX), and the control of remotely piloted vehicles in military, space, and hazardous waste management applications.

In short, an ever-expanding number of applications depend on the efficient manipulation, storage, and transmission of binary, gray-scale, and color images.

Compression techniques fall into two broad categories: information preserving (Lossless) compression and lossy compression. These methods allow an image to be compressed and decompressed without losing information.

3.2 Fundamentals

The term "data compression" refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between "data" and "information". In fact, data are the means by which information is conveyed or built. Various amounts of data may be used to represent the same amount of information.

That is, it contains data (or words) that either provide no relevant information or simply restate that which is already known. It is thus said to contain data redundancy. Equation (3.1) shows the compression ratio where n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information :

$$C_R = \frac{n_1}{n_2} \tag{3.1}$$

A practical compression ratio, such as 10 (or 10:1), means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second (or compressed) data set. The corresponding redundancy of 0.9 implies that 90% of the data in the first data set is redundant.

3.3 Coding Redundancy

In digital image compression, three basic data redundancies can be identified and exploited: coding redundancy, interpixel redundancy, and psychovisual redundancy. Data compression is achieved when one or more of these redundancies are reduced or eliminated [3].

The gray-level histogram of an image can provide a great deal of insight into the construction of codes to reduce the amount of data used to represent it. In general, coding redundancy is present when the codes assigned to a set of events (such as gray-level values) have not been selected to take full advantage of the probabilities of the events.

A closely related objective measurement is the mean-square signal-to-noise ratio of the compressed-decompressed image, shown in Table 3.1 If f(x,v) is considered to be the sum of the original image g(x,y) and a noise signal e(x,y), the mean-square signal-to-noise ratio of the output image, denoted SNR, is

$$SNR_{rms} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[f(x,y)-g(x,y)]^2} \tag{3.2}$$

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

Table 3.1 Rating scale of the Television Allocations Study Organization.

3.4 Image Compression Models

There are some techniques for reducing or compressing the amount of data required to represent an image. However, these techniques typically are combined to form practical image compression systems. In this section, we examine the overall characteristics of such a system and develop a general model to represent it.

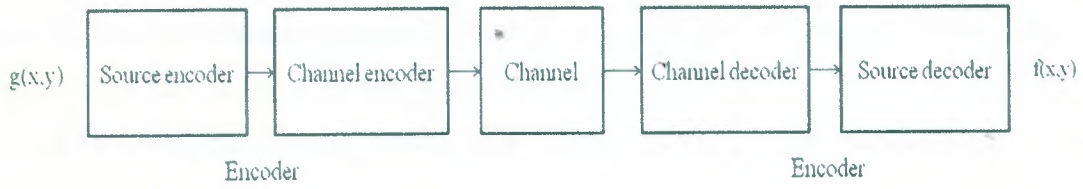


Figure 3.1 A general compression system model.

As Figure 3.1 shows, a compression system consists of two separate blocks : an encoder and a decoder. As input image f(x,y) is fed into the encoder ,which creates a set of symbols from the input dat. After transmission over the channel, the encoded representation is fed to the decoder, where a reconstructed output image f(x,y) is generated. In general the output image may or may not be an exact replica of the

original image. If it is, the system is error free or information preserving; if not ,some level of distortion is present in the reconstructed image [3].

Both the encoder and decoder shown in Figure 3.1 consist of two relatively independent functions or sub blocks. The encoder is made up of a source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. As would be expected, the decoder includes a channel decoder followed by a source decoder.

If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

3.4.1 The Source Encoder and Decoder

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application and associated fidelity requirements dictate the best encoding approach to use in any given situation. Normally, the approach can be modeled by a series of three independent operations.

As Figure 3.2(a) shows, each operation is designed to reduce one of the three redundancies described mentioned before. Figure 3.2(b) depicts the corresponding source decoder.

In the first stage of the source encoding process, the mapper transforms the input data into a (usually non-visual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.

The terms encoder and decoder reflect the influence of information theory on the field of image compression.

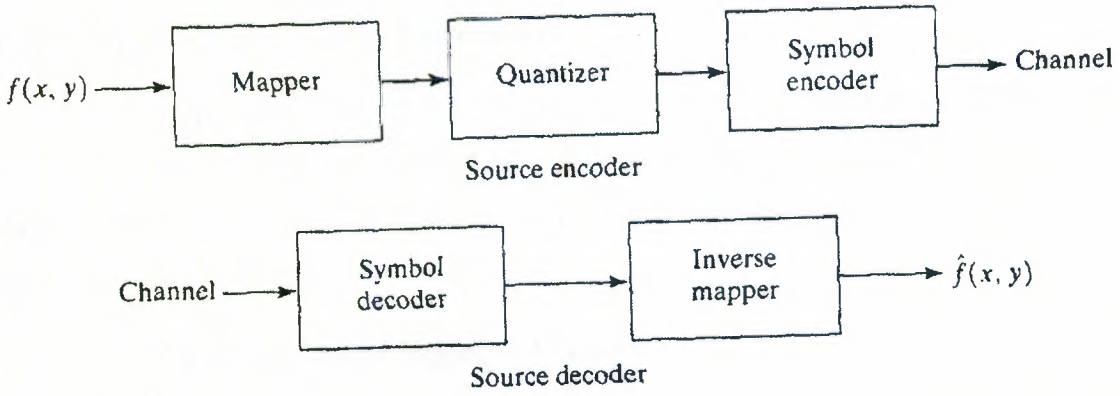


Figure 3.2 a) Source Encoder b) Source Decoder

The second stage, or quantizer block in Figure 3.2(a), reduces the accuracy of the mapper's output in accordance with some pre-established fidelity criterion. This stage reduces the psychovisual redundancies of the input image. Which is an irreversible operation. Thus it must be omitted when error-free compression is desired.

In the third and final stage of the source encoding process, the symbol coder creates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding process. In most cases, a variable-length code is used to represent the mapped and quantized data set.

It assigns the shortest code words to the most frequently occurring output values and thus reduces coding redundancy. The operation, of course, is reversible. Upon completion of the symbol coding step, the input image has been processed to remove the redundancy.

Figure 3.2(a) shows the source encoding process as three successive operations, but all three operations are not necessarily included in every compression system. The source decoder shown in Figure 3.2(b) contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in Figure 3.2(b).

3.5 Fundamental Coding Theorems

In this section, we add a communication system to the model and examine three basic theorems regarding the coding or representation of information. As Figure 3.3 shows, the communication system is inserted between the source and the user and consists of an encoder and decoder [3].

3.5.1 The Noiseless Coding Theorem

When both the information channel and communication system are error free, the principal function of the communication system is to represent the source as compactly as possible. Under these circumstances, the noiseless coding theorem, also called Shannon's first theorem ,which has been come up in 1948, defines the minimum average code word length per source symbol that can be

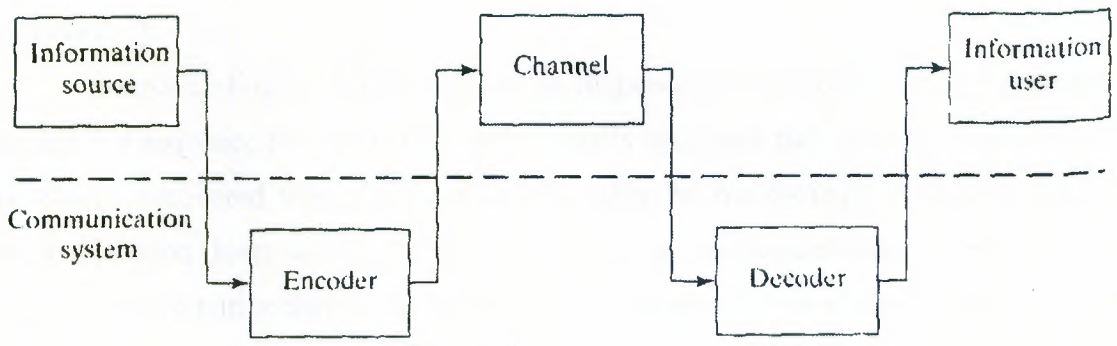


Figure 3.3 A communication system model

3.5.2 The Noisy Coding Theorem

This theorem establishes fundamental limits on error-free communication over both reliable and unreliable channels.

Shannon's second theorem also called the noisy coding theorem, tells us that for any $R < C$, where C is the capacity of the zero-memory channel, then the probability of error can be made arbitrarily small so long as the coded message rate is less than the capacity of the channel [3].

3.5.3 The Source Coding Theorem

In information theory, Shannon's source coding theorem (or noiseless coding theorem) establishes the limits to possible data compression, and the operational meaning of the Shannon entropy.

The source coding theorem shows that (in the limit, as the length of a stream of i.i.d. data tends to infinity) it is impossible to compress the data such that the code rate (average number of bits per symbol) is less than the Shannon entropy of the source, without it being virtually certain that information will be lost. However it is possible to get the code rate arbitrarily close to the Shannon entropy, with negligible probability of loss.

The source coding theorem for symbol codes places an upper and a lower bound on the minimal possible expected length of codewords as a function of the entropy of the input word (which is viewed as a random variable) and of the size of the target alphabet [3].

Source coding is a mapping from (a sequence of) symbols from an information source to a sequence of alphabet symbols (usually bits) such that the source symbols can be exactly recovered from the binary bits (lossless source coding) or recovered within some distortion (lossy source coding). This is the concept behind data compression.

In information theory, the source coding theorem informally states that: " N i.i.d. random variables each with entropy H(X) can be compressed into more than NH(X) bits with negligible risk of information loss, as N tends to infinity; but conversely, if they are compressed into fewer than NH(X) bits it is virtually certain that information will be lost ".

3.6 Error-Free Compression

In numerous applications error-free compression is the only acceptable means of data reduction. One such application is the archival of medical or business documents, where lossy compression usually is prohibited for legal reasons. Another is the processing of satellite imagery, where both the use and cost of collecting the data makes any loss undesirable. Yet another is digital radiography, where the loss of information can compromise diagnostic accuracy.

In these and other cases, the need for error-free compression is motivated by the intended use or nature of the images under consideration. In this section, we focus on the principal error-free compression strategies currently in use.

They normally provide compression ratios of 2 to 10. Moreover, they are equally applicable to both binary and gray-scale images. As indicated in the previous section error-free compression techniques generally are composed of two relatively independent operations: A) devising an alternative representation of the image in which its interpixel redundancies are reduced; and B) coding the representation to eliminate coding redundancies. These steps correspond to the mapping and symbol coding operations of the source coding model discussed in connection with Figure 3.2.

3.6.1 Variable-Length Coding

The simplest approach to error-free image compression is to reduce only coding redundancy. Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. To do so requires construction of a variable-length code that assigns the shortest possible code words to the most probable gray levels. Here, we examine several optimal and near optimal techniques for constructing such a code.

These techniques are formulated in the language of information theory. In practice, the source symbols may be either the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run lengths, and so on).

3.6.1.1 Huffman coding

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

Huffman encoding, based on the frequency of occurrence of a symbol in the file that is being compressed. The Huffman algorithm is based on statistical coding, which means that the probability of a symbol has a direct bearing on the length of its representation. The more probable the occurrence of a symbol is, the shorter will be its

bit-size representation. In any file, certain characters are used more than others. Using binary representation, the number of bits required to represent each character depends upon the number of characters that have to be represented. Using one bit we can represent two characters, i.e., 0 represents the first character and 1 represents the second character.

Using two bits we can represent four characters, and so on. Unlike ASCII code, which is a fixed-length code using seven bits per character, Huffman compression is a variable-length coding system that assigns smaller codes for more frequently used characters and larger codes for less frequently used characters in order to reduce the size of files being compressed and transferred.

For example, in a file with the following data:

XXXXXXYYYYZZ

The frequency of "X" is 6, the frequency of "Y" is 4, and the frequency of "Z" is equal two. If each character is represented using a fixed-length code of two bits, then the number of bits required to store this file would be 24, i.e., (2 x 6) + (2x 4) + (2x 2) = 24. If the above data were compressed using Huffman compression, the more frequently occurring numbers would be represented by smaller bits, such as:

X by the code 0 (1 bit)
Y by the code 10 (2 bits)
Z by the code 11 (2 bits)

Therefore the size of the file becomes 18, i.e., (1x 6) + (2 x 4) + (2 x 2) = 18. In the above example, more frequently occurring characters are assigned smaller codes, resulting in a smaller number of bits in the final compressed file. Huffman compression was named after its discoverer, David Huffman.

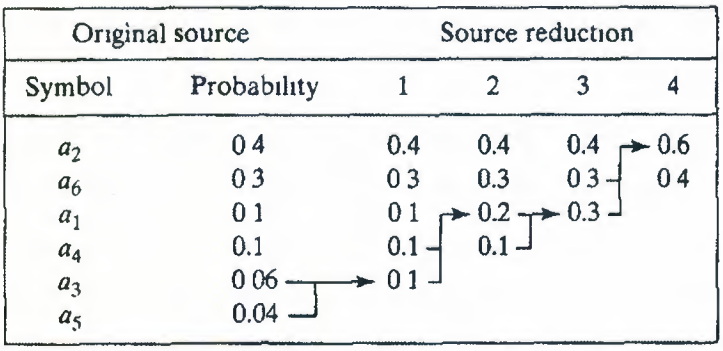


Figure 3.4 Huffman source reductions.

As shown in figure above, to form the first source reduction, the bottom two probabilities, 0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

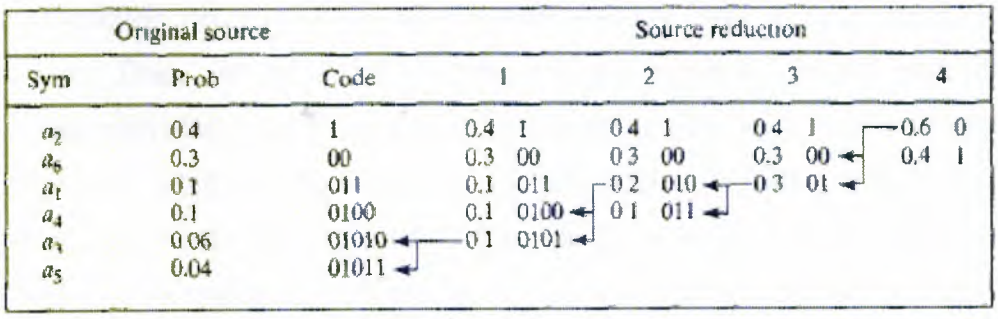


Figure 3.5 Huffman code assignment procedure.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As Figure 3.5 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and 1 would work just as well).

As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to

both of these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish them from each other.

This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far left in Figure 3.5. The average length of this code L_{avg}=(0.4)(1) +(0.3)(2)+ (0.1)(3)+(0.1)(4)+(0.06)(5)+(0.04)(5)=2.2 bits/symbol and the entropy of the source is 2.14 bits/symbol. In accordance with equation (3.3), the resulting Huffman code efficiency is 0.973.

$$\eta = n\frac{H(z)}{L_{avg}} \tag{3.3}$$

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple lookup table manner.

The code itself is an instantaneous uniquely decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous, because each code word in a string of code symbols can be decoded without referencing succeeding symbols.

It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of Figure 3.5, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a_3. The next valid code is 011, which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3a_1a_2a_2a_6$.

3.7 Lossy Compression

Unlike the error-free approaches outlined in the previous section, lossy encoding is based on the concept of compromising the accuracy of the reconstructed image in exchange for increased compression. If the resulting distortion (which may or may not be visually apparent) can be tolerated, the increase in compression can be significant [3].

In fact, many lossy encoding techniques are capable of reproducing recognizable monochrome images from data that have been compressed by more than 100:1 and images that are virtually indistinguishable from the originals at 10: 1 to 50:1. Error-free encoding of monochrome images, however, seldom results in more than a 3:1 reduction in data.

- Lossy Predictive Coding
- Transform Coding
- Wavelet Coding

Lossy image compression is useful in applications such as broadcast television, videoconferencing, and facsimile transmission, in which a certain amount of error is an acceptable trade-off for increased compression performance.

CHAPTER FOUR

RESULTS

4.1 Graphical user interface of MATLAB

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with controls like pushbuttons, list boxes, sliders, menus, and so forth. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button.

The three principal elements required to create a MATLAB Graphical User Interface are :

1. Components such as pushbuttons, labels, edit boxes, and axis. Graphical controls and static elements are created by the function `uicontrol`, and menus are created by the functions `uimenu` and `uicontextmenu`. Axes, which are used to display graphical data, are created by the function `axes`.
2. Figures; are window on the computer screen that the components of GUI can be arranged within. In the past, figures have been created automatically whenever we have plotted data. However, empty figures can be created with the function `figure` and can be used to hold any combination of components.
3. Callbacks, which is a way to perform an action if a user clicks a mouse on a button. Since a mouse click or the key pressed is an event, the MATLAB program must respond to each event if the program is to perform its function.

4.2 Graphical user interface of an image restoration system

figure 4.1 represents the GUI of an image restoration system. In this figure a digital image passes from the transmitter through the noisy channel to the receiver. The following functions shows how the restoration process goes.

- load image : selects the image that is going to be transmitted
- Resize image : resizes the original image to a desired size.
- Convert to Gray : converts the original image into grayscale form.
- Apply noise : blurring the transmitted image by adding noise to it.
- Choose filter : chooses a filtering method from the pop-up menu.

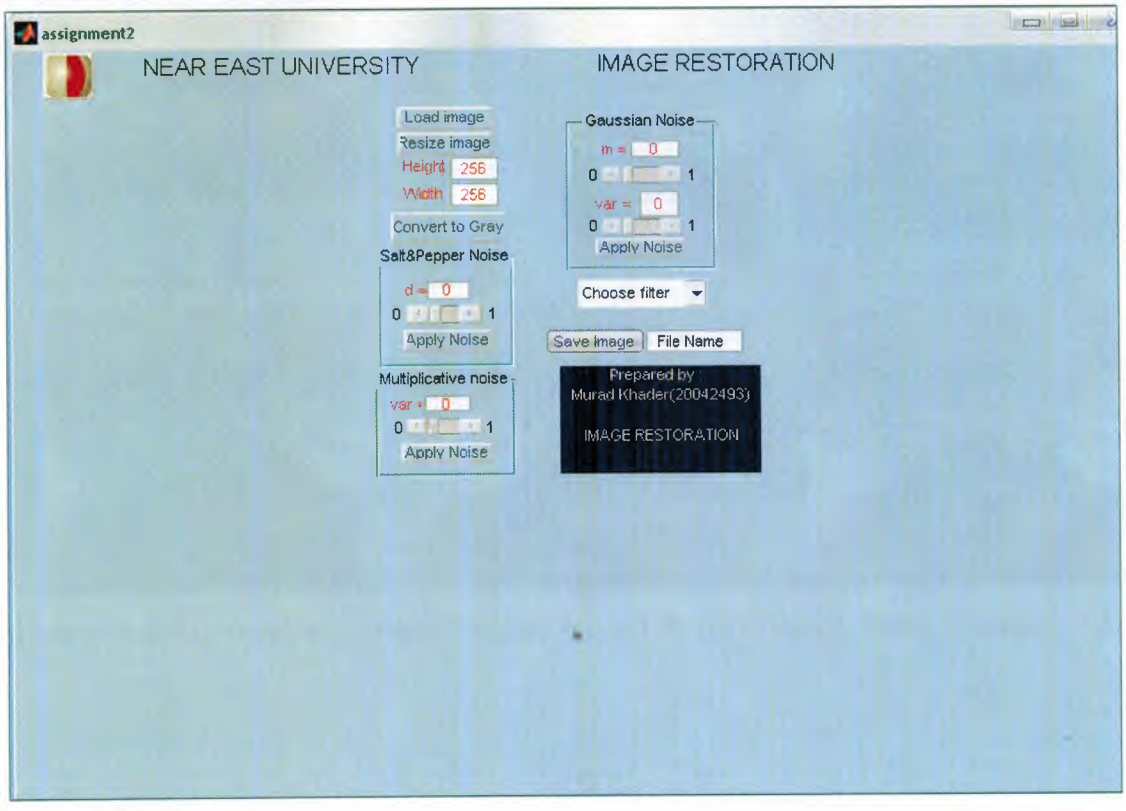


Figure 4.1 GUI of an image restoration system.

In figure 4.2 the image has been resized then transmitted through the noisy channel. the transmitted image is affected by salt & paper noise then it has been filtered using the averaging filter. In figure 4.4 same image which has been affected by the same amount of noise but it was processed using the median filter. Comparing both image by visual inspection the one which filtered using averaging filter, and the one which is filtered using median filter, we could see that the median filter is more efficient than the averaging filter for salt & paper noise.

Figure 4.2 Restoring a corrupted image by salt & paper noise using averaging filter.

(a) (b)

Figure 4.3 a) blurred image with salt & paper noise. b) filtered image using averaging filter.

(a) (b)

Figure 4.3 a) blurred image with salt & paper noise. b) filtered image using median filter.

4.3 Segmentation of image compression

A true color image which has three layers has been compressed using the Huffman coding. Thus, the output image represents the same image but reduced amount of redundant data. Comparing both images; the original image and the compressed image we can notice that the compressed one has less quality than the transmitted one.

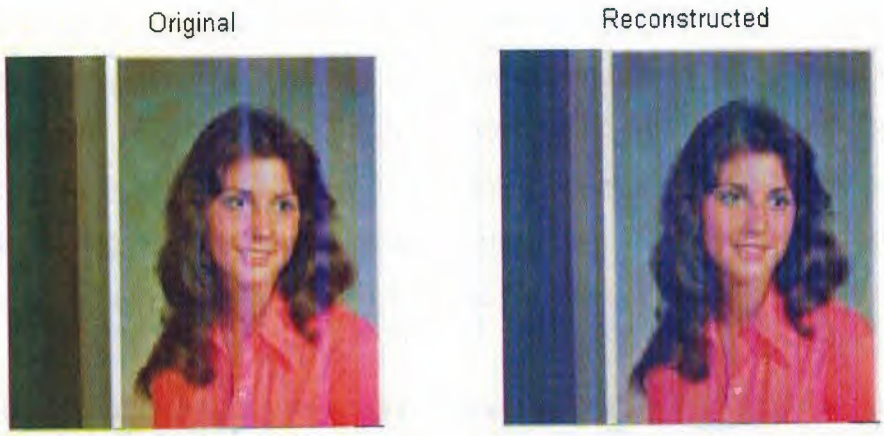


Figure 4.4 shows the original image before compression at the LHS & the reconstructed image after being compressed at the RHS.

The compression rate and ratio indicates how much compression of the original image is done. The original image have better quality than compressed image since it has more redundant data than the compressed one, so better quality but needs more space to store.

CONCLUSION

Where there is a communication system, there will be errors that happens during transmission process which could result in a sever corruption of the signal. Such errors should be overcome in order to receive a signal having less amount of loss.

The image processing techniques; image enhancement, image recognition, image compression, and image restoration, are techniques used to improve the received image after being transmitted and get corrupted through the channel.

Image enhancement emphasizes some features of images or the whole image, so that are apparent (clear) to the human eye or further process by machines. Image restoration could help in processing the corrupted image by using different kinds of filtering, depending on what type of noise the image got affected by. The receiver can apply filtering techniques that would reduce the noise and blurring of edges within the image.

In image compression, the image is processed before transmission by reducing the amount of data or pixels needed to represent that image.

The future research would about making a better filtering method that can understand easily the behavior of noise and can fix almost all the errors with high filtering speed. Also by adding redundant bits, channel coding could be used to obtain better image restoration.

REFERENCES

[1] Niku S.B., Introduction to Robotics, Analysis, Systems, Applications, Prentice hall Inc., Upper saddle River, New Jersey, 2001.

[2] Gonzalez Rafael C., Woods Richard E. and Eddins Steven L. (2002). Digital Image Processing. Second Edition. New Jersey, Prentice-Hall.

[3] Gonzalez Rafael C. and Woods Richard E. (2002). Digital Image Processing. Second Edition. New Jersey, Prentice-Hall.

[4] Haykin S., Communication Systems, John Wiley & Sons Inc., Toronto, 1994.

[5] Mamedov F., Telecommunications, Near East University press, Nicosia, 2000.

[6] Proakis J.G., Salehi M., Communication Systems Engineering, Prentice Hall, Upper Saddle River, New Jersey, 2002.

APPENDICES

Appendix I Simulation of Image Compression

```
% "Color.Image.Compression" using Hufman Coding
% Prototype JPEG compression algorithm demostration
% Only colored pictures are considered

clear all
clc
ali=1;
[FileName,PathName] = uigetfile({'*.jpg;*.JPG','JPG Files (*.jpg,
*JPG)';
   '*.tif;*.TIF','TIF Files (*.tif, *.TIF)' ;...
    '*.gif;*.GIF','GIF Files (*.gif, *.GIF)';...
    '*.bmp;*.BMP','BMP Files (*.bmp, *.BMP)';...
    '*.*','All Files(*.*)'},'Select the input file');
if isequal(FileName,0) || isequal(PathName,0)

       warndlg('Error: user pressed cancel. Please, select an image
file.','Load Error');

else
cd (PathName)
c=imread(FileName);
tariq(:,:,1)=imresize(c(:,:,1),[256 256]);
tariq(:,:,2)=imresize(c(:,:,2),[256 256]);
tariq(:,:,3)=imresize(c(:,:,3),[256 256]);
for ali=1:3

    %load p64int.txt;f=p64int; clear p64int;
    x=c(:,:,ali);
    x=imresize(x,[256 256]);
    x=im2double(x);
    x=229*x;
    cd('E:\PROJECT2008\ColorImageCompression')
    save('image.mat', 'x');
    f=x;%(1+128:128+128,1+128:128+128);
    clear x

% level shift by 128
f=f-128;
%pause
drawnow
[mf,nf]=size(f); mb=mf/8; nb=nf/8;
% size of f, # of blocks of f

% Step 1. 2D separable DCT on each 8x8
% blocks
    Ff=blkproc(f,[8 8],'dct');
    % apply DCT to each column of each block of f
    Ff=blkproc(Ff',[8 8],'dct');
    % apply DCT to each row of each block of Ff
    Ff=round(Ff');

% Perceptual scaler quantization
```

```
Q =[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];
% this is the quantization matrix

% Now perform rounding

    Fq=round(blkproc(Ff,[8 8],'divq',Q));
if mb*nb > 1,
   fdc=reshape(Fq(1:8:mf,1:8:nf)',mb*nb,1);
   fdpcm=dpcm(fdc,1);
else
   fdpcm=Fq(1,1);
end
dccof=[];
for i=1:mb*nb,
   dccof=[dccof jdcenc(fdpcm(i))];
end

% Zig-Zag scanning of AC coefficients
z=[1   2   6   7  15  16  28  29
   3   5   8  14  17  27  30  43
   4   9  13  18  26  31  42  44
  10  12  19  25  32  41  45  54
  11  20  24  33  40  46  53  55
  21  23  34  39  47  52  56  61
  22  35  38  48  51  57  60  62
  36  37  49  50  58  59  63  64];
%pause
%echo off
acseq=[];
for i=1:mb
  for j=1:nb
    tmp(z)=Fq(8*(i-1)+1:8*i,8*(j-1)+1:8*j);
    % tmp is 1 by 64
    eobi=max(find(tmp~=0)); %end of block index
                    % eob is labelled with 999
    acseq=[acseq tmp(2:eobi) 999];
  end
end
accof=jacenc(acseq);

disp(['DC coefficient after Huffman coding has '
int2str(length(dccof)) ...
' bits']);
disp(['AC coefficient after Huffman coding has '
int2str(length(accof)) ...
' bits']);

disp(['Compression Rate   '
num2str((length(dccof)+length(accof))/(mb*nb*64)) '   Bits / pixel '])
disp(['Compression Ratio   '
num2str(8/((length(dccof)+length(accof))/(mb*nb*64))) ' : 1'])
```

```
% Decoding side

% Inverse JPEG ,reconstruction of image
acarr=jacdec(accof);
dcarr=jdcdec(dccof);
% figure
% % Assumed that image size is 256 X 256, recostruction begins
 load image.mat % To find MSE, we need to have original image
drawnow
Q =[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];

z=[1   2   6   7  15  16  28  29
   3   5   8  14  17  27  30  43
   4   9  13  18  26  31  42  44
  10  12  19  25  32  41  45  54
  11  20  24  33  40  46  53  55
  21  23  34  39  47  52  56  61
  22  35  38  48  51  57  60  62
  36  37  49  50  58  59  63  64];
z=z(:);
mb=256/8; nb=256/8;  % Number of blocks

Eob=find(acarr==999);
kk=1;ind1=1;n=1;
for ii=1:mb
    for jj=1:nb
        ac=acarr(ind1:Eob(n)-1);
        ind1=Eob(n)+1;
        n=n+1;
        ri(8*(ii-1)+1:8*ii,8*(jj-1)+1:8*jj)=dezz([dcarr(kk) ac
zeros(1,63-length(ac))]);
        kk=kk+1;
    end
end

iFq=round(blkproc(ri,[8 8],'idivq',Q));
iFf=blkproc(iFq,[8 8],'idct2');
iFf=round(iFf+128);
khader(:,:,ali)=mat2gray(iFf);
% Calculate MSE , SNR
MSE=mean(mean((x-iFf).^2))
SNR=10*log10(255^2/MSE)
end

figure
subplot 121
imshow(tariq),title(' Original ')
subplot 122
imshow(khader),title(' Reconstructed ')
end
```

Appendix II Graphical User Interface of An Image Restoration System

```
function varargout = assignment2(varargin)
% ASSIGNMENT2 M-file for assignment2.fig
%      ASSIGNMENT2, by itself, creates a new ASSIGNMENT2 or raises the
existing
%      singleton*.
%
%      H = ASSIGNMENT2 returns the handle to a new ASSIGNMENT2 or the
handle to
%      the existing singleton*.
%
%      ASSIGNMENT2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in ASSIGNMENT2.M with the given input
arguments.
%
%      ASSIGNMENT2('Property','Value',...) creates a new ASSIGNMENT2
or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before assignment2_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to assignment2_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Begin initialization code


gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @assignment2_OpeningFcn, ...
                   'gui_OutputFcn',  @assignment2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code


% --- Executes just before assignment2 is made visible.
function assignment2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to assignment2 (see VARARGIN)

% Choose default command line output for assignment2
handles.output = hObject;

neu1 = imread('neulogo1.jpg');

set(handles.text19,'CData',neu1);


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes assignment2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = assignment2_OutputFcn(hObject, eventdata,
handles)

% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in load_button.
function varargout =load_button_Callback(h,eventdata,handles,varargin)

err=0;


[FileName,PathName] = uigetfile({'*.jpg;*.JPG','JPG Files (*.jpg,
*JPG)';
   '*.tif;*.TIF','TIF Files (*.tif, *.TIF)' ;...
    '*.gif;*.GIF','GIF Files (*.gif, *.GIF)';...
    '*.bmp;*.BMP','BMP Files (*.bmp, *.BMP)';...
    '*.*','All Files(*.*)'},'Select the input file');

if isequal(FileName,0) || isequal(PathName,0)

       warndlg('Error: user pressed cancel. Please, select an image
file.','Load Error');

else
       [pathstr,name,ext,versn] = fileparts(FileName);

       if isequal(ext,'.tif') || isequal(ext,'.jpg') ||
isequal(ext,'.JPG')...
              || isequal(ext,'.TIF') || isequal(ext,'.BMP') ||
isequal(ext,'.GIF') ...
              || isequal(ext,'.bmp') || isequal(ext,'.gif'),
```

```
                cd (PathName)

                handles.FileName=FileName;


                w=imread(FileName);


                handles.data = w;
                guidata(h,handles)

                axes(handles.axes1);

                set(handles.frame1,'Visible','off');

          imshow(w);

          set(handles.uipanel5,'Title',strcat('Original Image File Name
: ', FileName));
          set(handles.uipanel5,'TitlePosition','lefttop');

                    set(handles.information,'string','The Image is
acquired using the function [a=imread(filename)]');


         else

                warndlg('Error: invalid file format. Please select an
image file.','Load Error');

         end

end
% hObject    handle to load_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in res_button.
function varargout =res_button_Callback(h,eventdata,handles,varargin)
axes(handles.axes2);
w = handles.data;
ht = str2double(get(handles.height_ed,'String'));
wd = str2double(get(handles.width_ed,'String'));
b=imresize(w,[ht wd],'nearest');
set(handles.frame2,'Visible','off');
imshow(b);
ht1=get(handles.height_ed,'String');
wd1=get(handles.width_ed,'String');
 set(handles.uipanel6,'Title',strcat('The resized Image ',' Hieght','
',ht1,' Width ','  ',wd1));
  set(handles.uipanel6,'TitlePosition','lefttop');

                    set(handles.information,'string','The Image is
resized using the function b=imresize(a,[hieght width],nearest) where
nearest refer to Nearest-neighbor interpolation');
handles.data1=b;
guidata(h,handles)
% hObject    handle to res_button (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function height_ed_Callback(hObject, eventdata, handles)
% hObject    handle to height_ed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of height_ed as text
%        str2double(get(hObject,'String')) returns contents of
height_ed as a double


% --- Executes during object creation, after setting all properties.
function height_ed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to height_ed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



function width_ed_Callback(hObject, eventdata, handles)
% hObject    handle to width1_ed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of width1_ed as text
%        str2double(get(hObject,'String')) returns contents of
width1_ed as a double


% --- Executes during object creation, after setting all properties.
function width_ed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to width1_ed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



% --- Executes on button press in CGray.
function varargout =CGray_Callback(h,eventdata,handles,varargin)
axes(handles.axes4);
b=handles.data1;
cg=rgb2gray(b);
set(handles.frame3,'Visible','off');
imshow(cg);

set(handles.uipanel7,'Title','Gray scale Image ');
 set(handles.uipanel7,'TitlePosition','lefttop');

                  set(handles.information,'string','The Image is
coverted to gray scale using the function [g=rgb2gray(b)]');
handles.data2=cg;
guidata(h,handles)
% hObject    handle to CGray (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in gaus_button.
function varargout =gaus_button_Callback(h,eventdata,handles,varargin)
axes(handles.axes3);
cg=handles.data2;
m = get(handles.slider1,'Value');
var = get(handles.slider1,'Value');

J = imnoise(cg,'gaussian',m,var);

set(handles.frame4,'Visible','off');

imshow(J);

var1 = get(handles.edit4,'string');
m1 = get(handles.edit3,'string');

 set(handles.uipanel8,'TitlePosition','lefttop');

set(handles.uipanel8,'Title',strcat('The Image after applying Gaussian
noise,','m =',m1,'var =',var1 ));

  set(handles.information,'string','The Gaussian noise is applied to
the image using function  J = imnoise(g,gaussian,m,var) ');
handles.data3=J;
guidata(h,handles)

% hObject    handle to gaus_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on slider movement.
function varargout
=slider1_Callback(hObject,eventdata,handles,varargin)
set(handles.edit3,'string', [get(hObject,'Value');]);
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on slider movement.
function varargout
=slider2_Callback(hObject,eventdata,handles,varargin)
set(handles.edit4,'string', [get(hObject,'Value');]);
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3
as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4
as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on selection change in filter_pop.
function varargout = filter_pop_Callback(h,eventdata,handles,varargin)

axes(handles.axes5);

f=handles.data3;

val = get(h,'Value');
switch val
case 1

case 2
 l=filter2(fspecial('average',3),f)/255;
 set(handles.frame5,'Visible','off');
 imshow(l);

  set(handles.uipanel9,'TitlePosition','lefttop');

 set(handles.uipanel9,'Title','The Image after applying Averaging
filter');

  set(handles.information,'string','The Averaging filter is applied to
the image using function  l=filter2(fspecial(average,3),J)/255 ');
 case 3
  l= medfilt2(f,[3 3]);
  set(handles.frame5,'Visible','off');
  imshow(l);

   set(handles.uipanel9,'TitlePosition','lefttop');

 set(handles.uipanel9,'Title','The Image after applying Median
filter');

  set(handles.information,'string','The Median filter is applied to
the image using function  medfilt2(J,[3 3]) ');
end

handles.data4=l;
guidata(h,handles)

% hObject    handle to filter_pop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns filter_pop contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
filter_pop


% --- Executes during object creation, after setting all properties.
function filter_pop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to filter_pop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on slider movement.

function varargout
=slider3_Callback(hObject,eventdata,handles,varargin)
set(handles.edit6,'string', [get(hObject,'Value');]);
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end

% --- Executes on button press in pushbutton5.
function varargout =pushbutton5_Callback(h,eventdata,handles,varargin)
axes(handles.axes3);
cg=handles.data2;
d = get(handles.slider3,'Value');

J = imnoise(cg ,'salt & pepper', d);

d1 = get(handles.edit6,'string');

set(handles.uipanel8,'Title',strcat('The Image after applying
salt&pepper noise,','d =',d1 ));
 set(handles.uipanel8,'TitlePosition','lefttop');
```

```
  set(handles.information,'string','The salt&pepper noise is applied
to the image using function  J = imnoise(g ,salt & pepper, d) where d
is the noise density');
   set(handles.frame4,'Visible','off');
imshow(J);
handles.data3=J;
guidata(h,handles)


% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6
as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on slider movement.
function varargout
=slider4_Callback(hObject,eventdata,handles,varargin)
set(handles.edit7,'string', [get(hObject,'Value');]);

% hObject    handle to slider4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject,'BackgroundColor',[.9 .9 .9]);
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7
as a double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on button press in pushbutton6.
function varargout =pushbutton6_Callback(h,eventdata,handles,varargin)

axes(handles.axes3);

cg=handles.data2;

var = get(handles.slider4,'Value');

J = imnoise(cg,'speckle',var);

set(handles.frame4,'Visible','off');

imshow(J);
```

```
var1 = get(handles.edit7,'string');

set(handles.uipanel8,'TitlePosition','lefttop');
set(handles.uipanel8,'Title',strcat('The Image after applying
Multiplicative noise,','var =',var1 ));

 set(handles.information,'string','The Multiplicative noise is
pplied to the image using function  J = imnoise(g,speckle,var) ');
andles.data3=J;
uidata(h,handles)
 hObject    handle to pushbutton6 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)



 --- Executes on button press in saveimage.
unction varargout =saveimage_Callback(h,eventdata,handles,varargin)


 l=handles.data4;

 filteredimage = get(handles.edit8 ,'String');

  imwrite( l , filteredimage);

   set(handles.information,'string',strcat('The filtered image is
aved to the defult workpath using function imwrite( l ,
ilteredimage)',',The file name is : ',filteredimage));



 hObject    handle to saveimage (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)


nction edit8_Callback(hObject, eventdata, handles)
 hObject    handle to edit8 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)

 Hints: get(hObject,'String') returns contents of edit8 as text
        str2double(get(hObject,'String')) returns contents of edit8
  a double


--- Executes during object creation, after setting all properties.
nction edit8_CreateFcn(hObject, eventdata, handles)
hObject    handle to edit8 (see GCBO)
eventdata  reserved - to be defined in a future version of MATLAB
handles    empty - handles not created until after all CreateFcns
lled

Hint: edit controls usually have a white background on Windows.
      See ISPC and COMPUTER.
 ispc
  set(hObject,'BackgroundColor','white');
```

```
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
;
end


 --- Executes during object creation, after setting all properties.
function information_CreateFcn(hObject, eventdata, handles)
 hObject    handle to information (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    empty - handles not created until after all CreateFcns
alled


 --- Executes on button press in text19.
unction text19_Callback(hObject, eventdata, handles)
 hObject    handle to text19 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)


 --- Executes on slider movement.
unction slider9_Callback(hObject, eventdata, handles)
 hObject    handle to slider9 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)

 Hints: get(hObject,'Value') returns position of slider
        get(hObject,'Min') and get(hObject,'Max') to determine range
f slider


 --- Executes during object creation, after setting all properties.
unction slider9_CreateFcn(hObject, eventdata, handles)
 hObject    handle to slider9 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    empty - handles not created until after all CreateFcns
alled

 Hint: slider controls usually have a light gray background.
 isequal(get(hObject,'BackgroundColor'),
et(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
d


 --- Executes on slider movement.
nction slider10_Callback(hObject, eventdata, handles)
 hObject    handle to slider10 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)

 Hints: get(hObject,'Value') returns position of slider
        get(hObject,'Min') and get(hObject,'Max') to determine range
 slider
```

```
--- Executes during object creation, after setting all properties.
unction slider10_CreateFcn(hObject, eventdata, handles)
 hObject    handle to slider10 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    empty - handles not created until after all CreateFcns
alled

 Hint: slider controls usually have a light gray background.
 isequal(get(hObject,'BackgroundColor'),
et(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor',[.9 .9 .9]);
d



nction edit16_Callback(hObject, eventdata, handles)
 hObject    handle to edit16 (see GCBO)
 eventdata  reserved - to be defined in a future version of MATLAB
 handles    structure with handles and user data (see GUIDATA)

 Hints: get(hObject,'String') returns contents of edit16 as text
        str2double(get(hObject,'String')) returns contents of edit16
 a double


--- Executes during object creation, after setting all properties.
nction edit16_CreateFcn(hObject, eventdata, handles)
hObject    handle to edit16 (see GCBO)
eventdata  reserved - to be defined in a future version of MATLAB
handles    empty - handles not created until after all CreateFcns
lled

Hint: edit controls usually have a white background on Windows.
      See ISPC and COMPUTER.
 ispc && isequal(get(hObject,'BackgroundColor'),
t(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
d



nction edit17_Callback(hObject, eventdata, handles)
hObject    handle to edit17 (see GCBO)
eventdata  reserved - to be defined in a future version of MATLAB
handles    structure with handles and user data (see GUIDATA)

Hints: get(hObject,'String') returns contents of edit17 as text
       str2double(get(hObject,'String')) returns contents of edit17
 a double


-- Executes during object creation, after setting all properties.
nction edit17_CreateFcn(hObject, eventdata, handles)
Object    handle to edit17 (see GCBO)
eventdata  reserved - to be defined in a future version of MATLAB
handles    empty - handles not created until after all CreateFcns
lled

int: edit controls usually have a white background on Windows.
     See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton16.
function pushbutton16_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on slider movement.
function slider11_Callback(hObject, eventdata, handles)
% hObject    handle to slider11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18
as a double


% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on slider movement.
function slider12_Callback(hObject, eventdata, handles)
% hObject    handle to slider12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function slider12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19
as a double


% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton19.
function pushbutton19_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton20.
function pushbutton20_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20
as a double


% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit21_Callback(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit21 as text
%        str2double(get(hObject,'String')) returns contents of edit21
as a double
```

```
-- Executes during object creation, after setting all properties.
ction edit21_CreateFcn(hObject, eventdata, handles)
Object    handle to edit21 (see GCBO)
ventdata  reserved - to be defined in a future version of MATLAB
andles    empty - handles not created until after all CreateFcns
led

int: edit controls usually have a white background on Windows.
     See ISPC and COMPUTER.
spc && isequal(get(hObject,'BackgroundColor'),
0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');


- Executes on button press in pushbutton21.
tion pushbutton21_Callback(hObject, eventdata, handles)
bject    handle to pushbutton21 (see GCBO)
entdata  reserved - to be defined in a future version of MATLAB
ndles    structure with handles and user data (see GUIDATA)


- Executes on selection change in popupmenu3.
tion popupmenu3_Callback(hObject, eventdata, handles)
bject    handle to popupmenu3 (see GCBO)
ntdata   reserved - to be defined in a future version of MATLAB
ndles    structure with handles and user data (see GUIDATA)

ts: contents = get(hObject,'String') returns popupmenu3 contents
ll array
    contents{get(hObject,'Value')} returns selected item from
menu3


Executes during object creation, after setting all properties.
ion popupmenu3_CreateFcn(hObject, eventdata, handles)
ject    handle to popupmenu3 (see GCBO)
ntdata  reserved - to be defined in a future version of MATLAB
dles    empty - handles not created until after all CreateFcns
d

t: popupmenu controls usually have a white background on Windows.
   See ISPC and COMPUTER.
c && isequal(get(hObject,'BackgroundColor'),
'defaultUicontrolBackgroundColor'))
t(hObject,'BackgroundColor','white');


Executes on button press in pushbutton22.
on pushbutton22_Callback(hObject, eventdata, handles)
ect    handle to pushbutton22 (see GCBO)
tdata  reserved - to be defined in a future version of MATLAB
les    structure with handles and user data (see GUIDATA)


on edit22_Callback(hObject, eventdata, handles)
ect    handle to edit22 (see GCBO)
tdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit22 as text
%        str2double(get(hObject,'String')) returns contents of edit22
as a double


% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function load_button_CreateFcn(hObject, eventdata, handles)
% hObject    handle to load_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
frame1.
function frame1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to frame1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```