

DESIGN OF TWO-DIMENSIONAL FINITE IMPULSE RESPONSE
DIGITAL FILTERS : A SOFTWARE IMPLEMENTATION

by


Berhane Wolde-Gabriel

A Thesis submitted to the University of Nairobi for the Partial Fulfilment of the
Requirements for the Degree of Master of Science in Electrical Engineering.

May 1993

DECLARATION

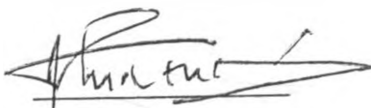
This is my original work and has not been presented for a degree in any other University.



Berhane Wolde-Gabriel

Date 28/10/93

This thesis has been submitted for examination with my approval as University Supervisor.



Prof. S.H. Mneney

Date 29/10/93

TABLE OF CONTENTS

	PAGE
LIST OF TABLES	vi
LIST OF FIGURES	vii
ACKNOWLEDGEMENT	vii
ABSTRACT	ix
1. INTRODUCTION	1
1.1. PROBLEM DEFINITION	4
1.2. SCOPE OF WORK	5
1.3. SUMMARY OF SIGNIFICANT RESULTS	7
2. A REVIEW OF FILTERING THEORY	10
2.1. ANALOG FILTERS	10
2.1.1. Butterworth Lowpass Filters	11
2.1.2. Chebyshev Lowpass Filters	12
2.1.3. Frequency Transformations	14
2.2. DIGITAL FILTERS	14
2.2.1. Digital Filter Representation	15
2.2.2. Classification of Digital Filters	17
2.2.3. Advantages of Digital Filters	19
2.3. TWO-DIMENSIONAL DIGITAL FILTERS	20
2.3.1. 2-D Digital Filter Representation	20
2.3.2. Classification of 2-D Digital Filters	23
3. DESIGN AND REALIZATION OF DIGITAL FILTERS	24
3.1. 1-D FIR DIGITAL FILTER DESIGN	24
3.1.1. Design of FIR Filters Using Windows	25
3.1.2. FIR Filter Design Using Frequency Sampling	26
3.1.3. Filter Design Using Equiripple Approximation	27

3.2. 1-D IIR DIGITAL FILTER DESIGN	29
3.2.1. Filter Design From Analog Filters	30
3.2.1.1. Impulse Invariant Design	30
3.2.1.2. Bilinear Transformation	32
3.2.1.3. Frequency Transformations	33
3.2.2. Time-Domain Design of IIR Filters	33
3.2.3. Optimization Methods	34
3.3. 2-D FIR DIGITAL FILTER DESIGN	35
3.3.1. FIR Filter Design Using Windows	36
3.3.2. FIR Filter Design Using Transformations	38
3.3.3. Optimal 2-D FIR Filter Design	39
3.4. 2-D IIR DIGITAL FILTER DESIGN	40
3.4.1. Space-Domain Design Techniques	41
3.4.2. Frequency Transformations	42
3.5. IMPLEMENTATION OF DIGITAL FILTERS	44
3.5.1. FIR Filter Implementations	44
3.5.2. IIR Filter Implementations	46
4. SOFTWARE IMPLEMENTATION OF 2-D FIR DIGITAL FILTERS	48
4.1. FIR FILTER DESIGN USING THE KAISER WINDOW	48
4.1.1. The Design Procedure	49
4.1.2. The Flow Chart For Filter Design	54
4.1.3. Main Subroutines in the Design Program	56
4.1.3.1. The Window	56
4.1.3.2. The Impulse Response	58
4.1.3.3. The Frequency Response	62
4.1.3.4. The Three-Dimensional Graph	64
4.2. APPLICATION TO IMAGE PROCESSING	67
4.2.1 Flow Chart For Image Processing	73

4.3. PROBLEMS ENCOUNTERED	74
5. RESULTS AND CONCLUSIONS	79
5.1. RESULTS	79
5.1.1. Design of Filters	79
5.1.2. Image Processing	93
5.2. CONCLUSIONS AND RECOMMENDATIONS	96
5.2.1. Design of Filters	96
5.2.2. Image Processing	98
6. REFERENCES	100
7. APPENDIX - PROGRAM LISTING	104

LIST OF TABLES	PAGE
Table 5.1 The Window Function	80
Table 5.2 The Ideal Impulse Response (Lowpass Filter)	82
Table 5.3 The Weighted Impulse Response (Lowpass Filter)	83
Table 5.4 Characteristics of a Designed Lowpass Filter	84
Table 5.5 The Impulse Response (Highpass Filter)	86
Table 5.6 Characteristics of a Designed Highpass Filter	87
Table 5.7 Characteristics of Redesigned Highpass Filter	87
Table 5.8 Impulse Response of Redesigned Highpass Filter	87

LIST OF FIGURES	PAGE
Fig. 4.1 The Three Axes	65
Fig. 5.1 The Kaiser Window Function	81
Fig. 5.2 The Ideal Impulse Response (Lowpass Filter)	82
Fig. 5.3 The Weighted Impulse Response (Lowpass Filter)	83
Fig. 5.4 The Frequency Response (Lowpass Filter)	84
Fig. 5.5 The Impulse Response (Highpass Filter)	86
Fig. 5.6 The Frequency Response (Redesigned Highpass Filter)	88
Fig. 5.7 Impulse Response (Bandpass Filter)	89
Fig. 5.8 Frequency Response (Bandpass Filter)	90
Fig. 5.9 Impulse Response (Bandstop Filter)	91
Fig. 5.10 Frequency Response (Bandstop Filter)	92
Fig. 5.11 Image of a Boy (Original)	93
Fig. 5.12 Image of the Boy (Corrupted With Impulse Noise)	94
Fig. 5.13 Image of the Boy After Filtering Noise-Corrupted Image With a Lowpass Filter	94
Fig. 5.14 Image of the Boy After Filtering the Original Uncorrupted Image With a Highpass Filter	95
Fig. 5.15 Image of the Boy After Filtering the Original Uncorrupted Image With a Highpass Filter of Narrower Transition Width	95

ACKNOWLEDGEMENT

I am highly indebted to Prof. S.H. Mneney, my Supervisor, for his persistent guidance and encouragement throughout the period of the work. His suggestions have been instrumental in shaping the project work into its present form.

Dr. G.S.O. Odhiambo has also been helpful in modifying a subroutine written by C. Ohlsen and G. Stoker that creates shades of gray levels to suit my work. I am grateful to him.

My special regards go to Mr B.K. Chomba and Mr M.N. Otieno, senior technicians at the microprocessor labs, who were very cooperative in assisting me acquaint myself with the personal computers.

I would like to extend my thanks to the German Technical cooperation (GTZ) for having sponsored my studies. My sincere appreciation should also extend to the staff of the German Academic Exchange Service (DAAD), Regional Office for AFRICA, for being highly cooperative in all respects of our communication.

And last, but by no means least, I would like to record my gratefulness to the Awassa College of Agriculture, Addis Ababa University, for making the necessary arrangements for my studies and allowing me study leave.

ABSTRACT

A Turbo-Pascal program is written to design circularly-symmetric, 2-Dimensional, finite impulse response (FIR) digital filters using the Kaiser window method.

The filter specifications are given in the frequency domain and comprise the passband limit(s), the transition width(s) and the ripple. Given these specifications, the first approximations of the window order and window parameter (α) are computed using expressions given by T.S. Speake and R.M. Mersereau [20].

The operation of inverse discrete Fourier transform (IDFT) is used to obtain the ideal impulse response. The window function and the ideal impulse response are then multiplied point by point to get the actual impulse response. The DFT operation is then performed to get the Fourier transform at discrete points in space.

The filter characteristics of the designed filter are made available in tabular forms once the frequency response is determined. This enables the user to compare them with the supplied specifications. If the user is satisfied with the designed filter, then he/she can have the filter impulse response in the form of tables or 3-dimensional graphs. If, on the other hand, the filter specifications are not met and the user wants to

redesign the filter, he/she can supply new values for the window order and window parameter (α). Increasing the window parameter (α) has the effect of reducing the ripple but widening the transition width. With the new values of the window order and parameter (α), the filter is redesigned. The process can be repeated as many times as one wishes until the user is satisfied with the design. The 3-dimensional graphs can be printed on paper if the user so wishes.

For most lowpass filter applications, a redesign of the filter is not required as the first round design produces filters that meet the specifications. For other types, however, redesigns are often required especially if the window order is low.

Coded data for 65x65 images with 32 gray levels are used as input data for the image processing application. An image is displayed on the screen with 16 gray levels (actual gray levels divided by 2).

The image is then corrupted by impulse noise with 10% probability and displayed on the screen.

The effect of the different standard filters on both the original image and the image corrupted with noise can then be studied by designing the required filters and filtering the images with these filters. To effect all these, one only needs to respond

appropriately to prompts from the computer.

The blurring effects of the lowpass filter can best be studied by applying it to an image corrupted with impulse noise while the edge sharpening effect of highpass filters can best be seen when applied to the original uncorrupted image.

1. INTRODUCTION

The advent of the high speed digital computer and its wide availability for research and development work has made it possible for the digital signal processing to emerge as a major discipline [1]. Many of the earlier works on one-dimensional (1-D) digital signal processing theory were modeled to simulate the analog systems theory. Later, with the fast development in the digital technology, it was found that not only could digital systems simulate analog systems very well but also do much more. In fact, many of the methods in common use today do not have analog counterparts [2].

Signals that are inherently two-dimensional (2-D) cannot, in most cases, effectively be handled with one-dimensional (1-D) digital signal processing theory. Hence the need for the development of the 2-D digital signal processing theory - a theory which has no equivalent in the analog world [2]. 2-D digital signal processing deals with the representation of 2-D signals, such as pictures, with 2-D arrays of numbers and the processing of these arrays.

2-D digital filters have found wide applications within the wider context of 2-D digital signal processing. These take the form of one or more of the following objectives [3] :

- 1) enhancement of the image to make it more acceptable to the

human eye,

- 2) removal of the effects of some degradation mechanism, and
- 3) separation of features for easier identification or measurement by machine or human.

Linear 2-D digital filtering can be used in the enhancement of medical images [4,5]. For example, a highpass 2-D digital filter may be employed to reduce spatial low frequency components in an x-ray image thus making features with high frequency components, such as fracture, easier to identify.

Another area of application of 2-D digital filters is in remote sensing. Here many images and maps are collected by platforms aboard aircrafts and satellites through different sensors. These images and maps, in general, need to be processed to improve their quality and to extract the useful information. 2-D lowpass filters may thus be applied to smooth sharp transitions such as those caused by the presence of impulse noise, while 2-D highpass filters can be used to extract the information contained at the edges and boundaries.

Still more effective for the removal of noise are nonlinear filters such as the median filters [4,5], the generalized mean filters [6] and the signal adaptive median (SAM) filters [7].

The median filter has both good noise cleansing and edge preserving properties. A class of filters that combines linear

filters and the median operation specifically designed for edge detection is proposed by Y. Neuvo et al. [8].

In the African context, the remote sensing applications are particularly appealing because they are useful in predicting the weather, monitoring earth resources and cartography. The levels of a lake, for example, may be monitored by studying its boundaries. And if the image taken from a satellite does not clearly show the boundaries, edge detecting digital filters may be applied to highlight the boundaries.

For the extraction of features, shape representation and description of images, P. Maragos et al. [9] have used mathematical morphology. And G.I. Verenza et al. [10] have used knowledge based systems for image processing and interpretation, especially in applications to the medical field.

Some of the filters mentioned above are nonlinear although many important filtering operations, such as spatial frequency filtering, are linear. Linear shift invariant (LSI) digital filters are easy to design and analyze, yet they are powerful enough to solve many practical problems [2]. These class of digital filters can be classified as infinite impulse response (IIR) or finite impulse response (FIR).

In designing IIR filters, stability is a very important constraint. Although this is a problem in both the 1-D and 2-D

digital filters, it is much more difficult to understand in the 2-D case [2,11].

The design of 2-D IIR filters can be accomplished through optimization techniques [12]-[15], by applying spectral transformations on prototype 2-D digital filters [3,16] or by bilinearly transforming a cascaded number of elementary analog filters with different rotations and different 1-D prototypes [17,18].

The design of 2-D FIR digital filters either uses transformation techniques such as the McClellan transformations [19] or are direct extensions of their 1-D counterparts like the windowing techniques [2,20,21] and the optimal design methods [22,23]. Of these, the windowing method is the easiest to apply and it is the most general, for it can be used to find filters of any order with any magnitude and phase characteristic [16].

This work is concerned with the development of a computer program that designs 2-D, circularly symmetric FIR digital filters using the windowing techniques and apply these for the enhancement of digital images.

1.1 PROBLEM DEFINITION

Digital filters can be implemented by either using a dedicated hardware or by programming a general purpose computer [24].

Before the implementation, however, the impulse response of the filter that meets the required specifications has to be determined, that is, the filter has to be designed. Both the design of the filter and the implementation of the filtering process involve a lot of computations. So, even if the filtering is ultimately to be implemented with a dedicated hardware, the filter characteristics can be studied and the necessary modifications be made using software methods before the final filter that meets the filter specifications is designed and implemented using dedicated hardware.

The department of Electrical and Electronic Engineering, University of Nairobi, is currently in possession of software packages for the design of 1-D finite impulse response (FIR) and infinite impulse response (IIR) filters. It does not however have the same for the design of 2-D digital filters. Nor are they readily available, at least in the local markets. In the absence of such a package, it is unthinkable to design 2-D digital filters. This remained a stumbling block to the implementation of any process that uses 2-D digital filters for its realization. This project aims at partially solving this problem.

1.2 SCOPE OF WORK

As the time available for the work cannot allow otherwise, a particular filter type and design technique was considered for

this work. Of the two filter types, i.e. FIR and IIR, the FIR filter was chosen and the design method used is the Kaiser window method. The reasons for choosing the FIR filter for the filter type and the Kaiser window for the design method are given in section (4.1).

The program is designed in such a way that the user provides the kind of filter (lowpass, highpass, bandpass, or bandstop) and the necessary specifications, in response to prompts from the computer. Then the filter is designed and the impulse response (or filter coefficients) are put in the form of tables and, if required, in perspective plot (3-D graph). Upon the computation of the frequency response of the designed filter, characteristics like the cutoff frequency, the transition bandwidth(s), the ripples in the stopband(s) and passband(s) are calculated. Also, the order of the window used and the parameter α used in the computation of the window are made available. This enables the user to see whether the supplied filter specifications are met or not. If they are not met, and the user wants to redesign the filter then he/she has the liberty to supply a new window order and parameter α to be used in the redesign. This can be repeated until such a time when the user is completely satisfied with the result.

Then the frequency response is displayed on a 3-D graph and finally the impulse response of the latest designed filter is

given in the form of a table.

Hardcopies of all the 3-D graphs can be obtained if desired. One only needs to respond in the affirmative when asked whether a hardcopy is required and supply the number of copies when asked for the number of copies.

In the image processing application, some image codes are included with the program and one is asked which image he/she wants to process. The image is then displayed on the screen with 16 gray levels. The user can then corrupt the image with impulse noise by calling a subroutine written for that purpose and view the resulting image. The image corrupted with noise, or the original uncorrupted one, can then be processed with a filter of type and specification of the user's choice.

The original image, the image corrupted with noise and the one processed with a filter can be printed by using the Graphics/PrintScreen capability of the computer.

1.3 SUMMARY OF SIGNIFICANT RESULTS

Given the type of filter to be designed and the specifications, the Kaiser window function is first computed. The expression for the order of the window, the parameter α used to compute the window as given by T.C. Speake and R.M. Mersereau [20] are taken as the first approximate values. The ideal impulse response is

obtained by first sampling the given ideal frequency response and then performing the inverse discrete Fourier transform operation on it. The final impulse response is obtained by multiplying the ideal impulse response and the window function point by point. This completes the first round design work.

But a test has to be conducted to see whether the designed filter meets the given specifications. For that, a discrete Fourier transform operation is performed on the impulse response. The filter characteristics can then be compared with the given specifications. Tests revealed that the specifications are not always met, especially with those of low order filters. So, the user may be required to supply new values for the window order and parameter α if he/she feels the design is not satisfactory and the filter will automatically be redesigned with these new values.

In the image processing applications, the blurring effect of the lowpass filter and the sharpening effect of the highpass filter can be demonstrated. The lowpass blurring effect can best be seen by applying it on an image corrupted with impulse noise. The noise is suppressed although, along with it, edges of the image proper are also somewhat blurred.

The effect of the highpass filter can be seen on the original image. It can be seen that if an image is filtered with a highpass

filter, the transition between high gray levels and low gray levels is sharpened or, if the transition bandwidth is reduced, only the edges are highlighted while the rest is blackened.

The filter design and filtering operations take longer as the filter order is increased. For this reason, the filter order to be designed is limited to a maximum value of 65x65. Filter order gets higher as the transition bandwidth of the filter gets narrower and/or the ripples get smaller.

2. A REVIEW OF FILTERING THEORY

Filtering of signals is a process by which inaccuracies caused by the presence of unwanted signals, or noise, are minimized [25]. Filters were originally viewed as circuits or systems with frequency selective behavior. Thus lowpass, highpass, bandpass and bandstop filters are merely filters that allow selected frequency ranges to pass with little or no attenuation while rejecting frequencies outside these ranges.

Filters can be realized using analog circuits or digital techniques. This chapter gives a review of some of the most common filters.

2.1 ANALOG FILTERS

Although the shift is now increasingly towards the use of digital filters, analog filters are also widely in use. This is so because of the speed, cost and size factors which favor the use of analog components in some applications [26].

Some of the most common applications of analog filters include [27] :

- . selection of the band of frequencies (audio or video) to modulate the radio frequency carrier of the transmitter,
- . selection of the band of frequencies containing the desired signal before amplification at the receiver end,

- . further selection after translating the variable input frequency spectrum to a fixed intermediate frequency,
- . rejection of the two sidebands and the carrier at the transmitter,
- . separation of various channels transmitted by frequency multiplexing methods,
- . band limiting at the transmitter end and smoothing at the receiver end to recover signals transmitted by time multiplexing methods,
- . and many more including as frequency multipliers, spectrum analyzers, matched filters and equalization filters.

Also, a number of useful image processing options in the form of thresholding, level slicing, contouring false colour enhancement are quite possible with analog filters [28].

There are a number of types of lowpass filters, but probably the most commonly used are the Butterworth and the Chebyshev filters [29].

2.1.1 Butterworth (Maximally flat) lowpass filter

The amplitude response of a Butterworth lowpass filter is given by

$$|H(j\omega)| = 1 / \sqrt{1 + (\omega/\omega_c)^{2n}} \quad (2.1)$$

where $n = 1, 2, 3, \dots$
 n = order of the filter,
 ω_c = cutoff frequency.

As n increases the response more nearly approximates that of the ideal filter. The Butterworth filter has excellent amplitude characteristics near $\omega=0$ but its characteristics near the cutoff and the attenuation in the stopband are relatively poor.

2.1.2 Chebyshev (Equiripple) lowpass Filter

Its amplitude response is given by

$$|H(j\omega)| = 1 / \sqrt{1 + \epsilon^2 C_n^2(\omega/\omega_c)} \quad (2.2)$$

where

$n = 1, 2, 3, \dots$
 n = order of the filter,
 n = number of half cycles in the stopband,
 ϵ = constant that determines the ripple height, and
 $C_n(x)$ is the n^{th} -order Chebyshev polynomial given by

$$C_n(x) = \begin{cases} \cos(n \cos^{-1} x) & , |x| \leq 1. \\ \cosh(n \cosh^{-1} x) & , |x| \geq 1. \end{cases} \quad (2.3)$$

Each polynomial oscillates between -1 and $+1$ in the interval $-1 \leq x \leq +1$ and increases in magnitude outside this range. The

Chebyshev polynomial can be found from the recursive formula

$$C_{n+1}(x) = 2xC_n(x) - C_{n-1}(x) . \quad (2.4)$$

With $C_0(x)=1$ and $C_1(x)=x$, other orders may be determined from the recursive formula.

If A_{\max} is defined as the maximum passband deviation, then ϵ can be expressed in terms of A_{\max} as [30]

$$\epsilon = \sqrt{10^{0.1A_{\max}} - 1} . \quad (2.5)$$

In applications where passband ripples are undesirable, the Butterworth filter is preferable to the Chebyshev filter.

However, for a fixed n and a given allowable deviation in the stopband, the Chebyshev filter is the best of all all-pole filters in that it has the smallest transition interval from the passband to some specified attenuation in the stopband. For example, the attenuation in decibels of the Chebyshev amplitude response is approximately $3(n-1)+20\log \epsilon$ below that of the Butterworth filter in the stopband [29].

The phase response of the Butterworth filter is more linear over the passband than that of the Chebyshev. In fact it is generally true that the better the amplitude response of a filter, the poorer its phase response is and vice-versa [29].

2.1.3 Frequency Transformations

Highpass, bandpass and bandstop filters may be obtained from a prototype lowpass filter by replacing the Laplace transform operator s in the transfer function of the prototype by an appropriate expression.

To transform the prototype lowpass filter to a highpass filter the s in the transfer function of the prototype lowpass filter is replaced by $1/s$.

In a bandpass filter B is defined as the bandwidth of the frequencies passed with center frequency ω_0 . The bandpass transfer function is then obtained by replacing the s in the prototype lowpass filter by $(\omega_0^2 + s^2)/Bs$.

In the band-reject filter B is defined as the bandwidth of the rejected frequencies centered at ω_0 . Its transfer function is obtained from that of a prototype lowpass filter by replacing the S in the prototype by $Bs/(s^2 + \omega_0^2)$.

2.2 DIGITAL FILTERS

Discrete time signals are defined only for discrete values of time, i.e. time is quantized while the term digital implies that both time and amplitude are quantized, In digital systems therefore signals are represented as a sequence of numbers which

take only a finite set of values.

A sequence of numbers x , in which the n^{th} number in the sequence is denoted $x(n)$ is formally written as

$$x = \{x(n)\}, \quad -\infty < n < \infty.$$

2.2.1 Digital Filter Representation

A digital system operates on an input sequence $x(n)$ to produce an output sequence $y(n)$. Linear system implies the principle of superposition applies, i.e. if $y_1(n)$ and $y_2(n)$ are the responses when $x_1(n)$ and $x_2(n)$, respectively are the inputs, then $ay_1(n) + by_2(n)$ is the response when $ax_1(n) + bx_2(n)$ is the input.

Shift-invariant systems are characterized by the property that if $y(n)$ is the response to $x(n)$ then $y(n-k)$ is the response to $x(n-k)$.

Thus, the input-output relationship of linear shift-invariant (LSI) digital systems is given by

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) h(n-k), \quad (2.6)$$

where $y(n)$ = output sequence,

$x(n)$ = input sequence,

$h(n)$ = impulse response of system .

Equation (2.6) above is referred to as the linear convolution sum and is denoted by

$$y(n) = x(n) * h(n). \quad (2.7a)$$

It can be easily shown, by substitution of variables, that

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) = h(n) * x(n), \quad (2.7)$$

and so convolution obeys the commutative law.

An LSI system is said to be stable if every bounded input produces a bounded output (BIBO). A necessary and sufficient condition on the impulse response for BIBO stability is

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty. \quad (2.8)$$

Digital filters are linear shift-invariant systems described by

$$y(n) = \sum_{k=0}^M a(k)x(n-k) - \sum_{k=1}^L b(k)y(n-k), \quad (2.9)$$

where $a(k)$ and $b(k)$ are the filter coefficients.

The z -transform $X(z)$ of a sequence $x(n)$ is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}$$

where z is a complex variable. It can be shown that the z -transform of a shifted sequence $x(n-n_0)$ is $z^{-n_0}X(z)$. The

z -transform is also a linear operation. Thus, taking the z -transform of Eqn.(2.9) and rearranging, we get

$$Y(z) = \frac{\sum_{k=0}^M a(k) z^{-k}}{1 + \sum_{k=1}^L b(k) z^{-k}} \cdot X(z). \quad (2.10)$$

Defining

$$H(z) = \frac{\sum_{k=0}^M a(k) z^{-k}}{1 + \sum_{k=1}^L b(k) z^{-k}}$$

we can write

$$Y(z) = H(z) X(z). \quad (2.11)$$

$H(z)$, the ratio of the z -transform of the output $Y(z)$ to the input $X(z)$, is referred to as the transfer function of the filter.

If $z = e^{j\omega}$ in the above equations, the z -transform reduces to the Fourier transform and the transfer function $H(z)$ becomes the frequency response of the filter. $z = e^{j\omega}$ is the unit circle on the z -plane.

2.2.2 Classification of Digital Filters

A filter is classified either as a finite impulse response (FIR) or infinite impulse response (IIR) filter depending on the

duration of the impulse response.

An FIR filter has a finite number of nonzero terms in its impulse response sequence $h(n)$ while an IIR filter has an infinite number of nonzero terms.

If $h(n) \neq 0$ for $n \leq 0$, the system is said to be causal or physically realizable. A noncausal system will have a nonzero response even before the input is applied and, therefore, it is not realizable.

Digital filters can also be classified as recursive or non-recursive for purposes of realization. When the $b(k)$'s in the transfer function are not all zero the calculation of $y(n)$ in the difference equation requires the values of some outputs that have already been calculated. Such filters are called recursive filters. Otherwise the filter becomes nonrecursive. In nonrecursive filters no previous values are required to calculate the present output, $y(n)$.

For nonrecursive filters, the transfer function reduces to

$$H(z) = \sum_{k=0}^M a(k) z^{-k}, \quad (2.12)$$

and the impulse response becomes identical to the $a(k)$ coefficients, i.e.

$$h(n) = \begin{cases} a(n) , & 0 \leq n \leq M . \\ 0 & , \text{ otherwise } . \end{cases}$$

For this reason, a nonrecursive filter is also a finite impulse response (FIR) filter, and a recursive filter is an infinite impulse response (IIR) filter [1,31].

However, in general both FIR and IIR filters can be implemented by either recursive or nonrecursive techniques [31].

2.2.3 Advantages of Digital Filters

Digital filters offer important advantages over their analog counterparts. Some of the advantages are [32]:

- i) They are highly accurate. The inaccuracies of digital filters, which are due to the rounding errors in the computer arithmetic, can be made as small as required. The background noise produced in analog circuits however cannot be so easily controlled and analog components cannot easily be made to a tolerance of less than about one per cent.
- ii) They permit a high degree of flexibility. For example, their characteristics can be changed by merely reading in from memory a new set of filter coefficients or, at most, by re-writing a section of the program.
- iii) They are free from drift. A computer program is not altered by variations like supply voltage and ambient temperature.

- iv) Freedom from the constraints of time is achieved when a waveform is in a digital memory .

2.3 TWO-DIMENSIONAL DIGITAL FILTERS

Two-dimensional(2-D) digital filters have found wide application in areas that inherently involve 2-D signals. Examples of 2-D signals include television images, reconnaissance photographs, medical x-ray images, radar and sonar arrays and seismic data.

A 2-D discrete signal is a function defined over the set of ordered pairs of integers.

Thus

$$x = \{x(n_1, n_2), \quad -\infty < n_1, n_2 < \infty \}$$

is a discrete signal defined over the ordered pairs of integers (n_1, n_2) .

A single element from the sequence is referred to as a sample. Thus $x(n_1, n_2)$ represents the sample of the sequence x at the point (n_1, n_2) .

2.3.1 2-D Digital Filter Representation

The input-output relationship of a 2-D digital linear shift-invariant (LSI) system is given by the 2-D convolution sum :

$$y(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) h(n_1 - k_1, n_2 - k_2), \quad (2.13)$$

where

$y(n_1, n_2)$ = 2-D output sequence,

$x(n_1, n_2)$ = 2-D input sequence,

$h(n_1, n_2)$ = impulse response of LSI system .

By a simple substitution of variables, the above equation can be shown to be equivalent to

$$y(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, k_2) x(n_1 - k_1, n_2 - k_2). \quad (2.14)$$

Using the double asterisk (**) to denote the 2-D convolution, the input-output relation of a 2-D digital LSI system is written as

$$y(n_1, n_2) = x(n_1, n_2) ** h(n_1, n_2) = h(n_1, n_2) ** x(n_1, n_2).$$

A necessary and sufficient condition for an LSI system to be BIBO stable is that its impulse response should be absolutely summable, i.e.

$$\sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |h(n_1, n_2)| < \infty. \quad (2.15)$$

Linear shift-invariant 2-D digital filters can be described by a constant coefficient, linear difference equation relating the output of the filter to the input. For a first quadrant filter, the input signal $x(n_1, n_2)$ and the output signal $y(n_1, n_2)$ are

related by

$$\begin{aligned}
 y(n_1, n_2) &= \sum_{l_1=0}^{L_1-1} \sum_{l_2=0}^{L_2-1} a(l_1, l_2) x(n_1-l_1, n_2-l_2) \\
 &= \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} b(k_1, k_2) y(n_1-k_1, n_2-k_2), \quad (2.16) \\
 &\quad (k_1, k_2) \neq (0, 0)
 \end{aligned}$$

The 2-D z-transform of a discrete array $x(n_1, n_2)$ is given by

$$X(z_1, z_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) z_1^{-n_1} z_2^{-n_2},$$

where z_1 and z_2 are complex variables.

Using the linearity and shifting properties of the z-transform and taking the z-transform of Eqn.(2.16) above, the transfer function can be shown to be

$$H(z_1, z_2) = \frac{\sum_{l_1=0}^{L_1-1} \sum_{l_2=0}^{L_2-1} a(l_1, l_2) z_1^{-l_1} z_2^{-l_2}}{\sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} b(k_1, k_2) z_1^{-k_1} z_2^{-k_2}} \triangleq \frac{A_z(z_1, z_2)}{B_z(z_1, z_2)} \quad (2.17)$$

where $a(l_1, l_2)$ and $b(k_1, k_2)$ are constants and $b(0,0)=1$.

With the transfer function $H(z_1, z_2)$ defined above, the z-transform of the output and input are related by

$$Y(z_1, z_2) = X(z_1, z_2) H(z_1, z_2). \quad (2.18)$$

Letting $z_1 = e^{j\omega_1}$ and $z_2 = e^{j\omega_2}$, the z-transform reduces to the Fourier transform and the transfer function $H(z_1, z_2)$ becomes the frequency response of the filter $H(\omega_1, \omega_2)$. The surface in the z-domain described by $z_1 = e^{j\omega_1}$, $z_2 = e^{j\omega_2}$ is known as the 2-D unit surface or the unit bicircle.

2.3.2 Classification of 2-D Digital Filters

If $b(k_1, k_2) = 0$ for all $(k_1, k_2) \neq (0, 0)$ the digital filter described by the difference equation (2.16) reduces to

$$y(n_1, n_2) = \sum_{l_1=0}^{L_1-1} \sum_{l_2=0}^{L_2-1} a(l_1, l_2) x(n_1-l_1, n_2-l_2) \quad (2.19)$$

Such a filter has only a finite number of nonzero terms in its impulse response, which turns out to be the same as the $a(l_1, l_2)$ coefficients, and is therefore called an FIR filter. It is also a nonrecursive filter as the value of $y(n_1, n_2)$ does not depend on previous output values.

On the other hand, if any of the $b(k_1, k_2)$ coefficients, apart from $b(0, 0)$ which is defined to be 1, are not equal to zero then the filter becomes a recursive filter and at the same time an IIR filter.

3. DESIGN AND REALIZATION OF DIGITAL FILTERS

Both 1-D and 2-D digital filters are classified as IIR or FIR filters. For each type of filter there are various design techniques which have got their own merits and demerits.

Given the filter specifications, the design of a filter entails the determination of the filter coefficients so that it meets the design specifications.

Realization of digital filters refers to the way the filter performs its intended purposes, i.e. filtering of signals. The realization techniques are also dependent on the filter type and, to some extent, on the design technique used.

3.1 1-D FIR DIGITAL FILTER DESIGN

1-D FIR Filters have transfer functions of the form

$$H(z) = \sum_{k=0}^{K-1} h(k) z^{-k}. \quad (3.1)$$

The design of 1-D FIR filters is, therefore, the task of determining the coefficients $\{h(k)\}$ so that it meets certain performance specifications. Usually, the specifications are in the frequency domain.

3.1.1 Design of FIR Filters Using Windows

Since $H(e^{j\omega})$, the frequency response of any digital filter, is periodic in frequency, it can be expanded in a Fourier series. The resulting series is of the form

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h(n) e^{-j\omega n}, \quad (3.2)$$

where

$$h(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} d\omega.$$

However, the filter impulse response $h(n)$ is not realizable as it is infinite in duration and is not causal (it begins at $-\infty$). A weighting sequence $w(n)$, called the window, is thus used to modify the Fourier coefficients $h(n)$ in Eqn.(3.2) to control the convergence of the series.

To produce an FIR approximation to $H(e^{j\omega})$, the sequence $\hat{h}(n) = h(n)w(n)$ is formed. $\hat{H}(e^{j\omega})$, the Fourier transform of $\hat{h}(n)$, can be shown to be given by [26]

$$\hat{H}(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\theta}) w(e^{j(\omega-\theta)}) d\theta, \quad (3.3)$$

where ω and θ are frequencies in radians.

Thus, $\hat{H}(e^{j\omega})$ is a periodic continuous convolution of the ideal frequency response with the Fourier transform of the window.

From Eqn. (3.3) above it can be seen that if $W(e^{j\omega})$ is narrow, $\hat{H}(e^{j\omega})$ will be nearly the same as $H(e^{j\omega})$. Therefore, the window is required to have $w(n)$ as short as possible in duration to minimize computations in the implementation of the filter, while having $W(e^{j\omega})$ as narrow as possible in frequency so as to faithfully reproduce the desired frequency response. These are conflicting requirements [26]. One has thus to choose a window with orders such that the required frequency response is not seriously compromised and, at the same time, the computation time is not too long.

Some of the commonly used windows include: the Bartlett, the Hanning, the Hamming, the Blackman and the Kaiser window functions [26,34].

3.1.2 Design of FIR Filters Using Frequency Sampling Method

The main idea here is that a desired frequency response can be approximated by sampling it at N evenly-spaced points and then obtaining an interpolated frequency response that passes through the frequency samples.

For filters with reasonably smooth frequency response, the interpolation error is generally small. In the case of band-selective filters, where the desired frequency response changes radically across bands, the frequency samples which occur in the

transition bands are set as variables whose values are chosen by an optimization algorithm which minimizes some function of the approximation error of the filter [26,33].

Frequency sampling designs are particularly attractive for narrow-band frequency-selective filters where only a few of the samples of the frequency response are nonzero [26].

3.1.3 FIR Filter Design Using Equiripple Approximation Methods

This design method is concerned with zero-phase FIR filters. The frequency response of such filters is given by

$$H(e^{j\omega}) = \sum_{n=-M}^M h(n)e^{-j\omega n}. \quad (3.4)$$

The impulse response sequence has a duration of $N=2M+1$ and for zero phase $h(n)$ must be equal to $h(-n)$. Because of symmetry, the above equation can be written as

$$H(e^{j\omega}) = h(0) + \sum_{n=1}^M 2h(n)\cos(\omega n).$$

Also, since $\cos(n\omega)$ can be expressed as a sum of powers of $\cos(\omega)$, the above can be written as

$$H(e^{j\omega}) = \sum_{k=0}^M a_k (\cos \omega)^k, \quad (3.5)$$

where the a'_x 's are constant coefficients which are related to $h(k)$.

Consider a lowpass filter whose passband range is $0 \leq |\omega| \leq \omega_p$ and stopband range is $\omega_s \leq |\omega| \leq \pi$. Suppose the maximum error in the passband is δ_1 and that in the stopband is δ_2 . It is not possible to specify each of the parameters $\delta_1, \delta_2, \omega_p, \omega_s$ and M . An approach developed by Herman and Schussler is to fix δ_1, δ_2 and M , and let ω_s and ω_p be variables [24,26].

Since Eqn.(3.5) above is an M^{th} order trigonometric polynomial, there can be at most $M-1$ local maxima and minima in the interval $0 < \omega < \pi$.

Differentiating Eq.(3.5) above with respect to ω , we get

$$\frac{dH(e^{j\omega})}{d\omega} = -\sin\omega \left[\sum_{k=1}^M k a_k (\cos\omega)^{k-1} \right]. \quad (3.6)$$

Thus it can be seen from Eqn. (3.6) that $H(e^{j\omega})$ will either have a maximum or a minimum at $\omega=0$ and $\omega=\pi$, and hence there will be at most $(M+1)$ local extrema in the interval $0 \leq \omega \leq \pi$. In general, there will be N_p extrema in the passband and N_s extrema in the stopband, and so we have

$$N_p + N_s = M+1. \quad (3.7)$$

We can write $2M$ equations relating the $M+1$ filter coefficients and the $M-1$ frequencies at which extrema occur. These equations are, however, nonlinear and are solved by iterative process [24,26].

3.2. 1-D IIR DIGITAL FILTER DESIGN

IIR filters promise a potential reduction in computation compared to FIR filters when performing comparable filtering operation. By feeding back output samples, we can use a filter with fewer coefficients (hence less computation) to implement a desired operation. On the other hand, IIR filters pose some potentially significant implementation and stabilization problems not encountered with FIR filters [2].

An IIR filter has a transfer function of the form

$$H(z) = \frac{\sum_{k=0}^M a(k)z^{-k}}{1 + \sum_{k=1}^L b(k)z^{-k}} \quad (3.8)$$

The design of an IIR filter thus centers around finding the filter coefficients, $a(k)$'s and $b(k)$'s, of the transfer function such that the filter satisfies some given performance specification.

Some of the techniques commonly used to design IIR filters are briefly discussed below.

3.2.1 Filter Design From Analog Filters

This technique involves designing an appropriate continuous-time filter and then transforming the resulting filter into a digital one. This method is most useful for designing standard filters such as lowpass, highpass, bandpass and bandstop filters [31].

3.2.1.1 Impulse Invariant Design Method

The impulse response of the digital filter is chosen as equally-spaced samples of the impulse response of the analog filter, $h_a(t)$, that is

$$h(n) = h_a(nT),$$

where T is the sampling period.

It can be shown [26] that the z -transform of $h(n)$ is related to the Laplace transform of $h_a(t)$ by the equation

$$H(z) \Big|_{z=e^{sT}} = \frac{1}{T} \sum_{k=-\infty}^{\infty} H_a\left(s + j\frac{2\pi k}{T}\right).$$

If the system function of the analog filter has N simple poles, then it can be written as

$$H(s) = \sum_{k=1}^N \frac{A_k}{s + s_k},$$

with a corresponding impulse response of

$$h(t) = \sum_{k=1}^N A_k e^{-s_k t},$$

and the impulse response of the digital filter becomes

$$h(n) = h_a(nT) = \sum_{k=1}^N A_k e^{-s_k nT}.$$

The system transfer function of the digital filter $H(z)$ is thus given by

$$H(z) = \sum_{k=1}^N \frac{A_k}{1 - e^{-s_k T} z^{-1}}. \quad (3.9)$$

Then the transform pairs become

$$\sum_{k=1}^N \frac{A_k}{s + s_k} \longleftrightarrow \sum_{k=1}^N \frac{A_k}{1 - e^{-s_k T} z^{-1}}.$$

Hence $H(z)$ may be obtained by first expressing $H(s)$ as a sum of partial fractions and then applying the transform pair relation given above.

For the frequency response of an analog filter and the equivalent digital filter obtained by the impulse invariant transformation to correspond, the analog filter must be band limited to the range $-\pi/T \leq \Omega \leq \pi/T$, where Ω is the analog frequency. A guard filter, i.e.

a suitable lowpass filter, is thus required to guarantee that the analog filter is suitably band limited prior to transformation.

The digital filter obtained using the impulse invariant method is stable if the analog filter was stable. According to Bozic [31], however, there exists a possibility of overflow in the computer program because of the variation of the filter gain with sampling frequency, say at $z=1$ or $\omega=0$. It may therefore be necessary to compensate for it.

3.2.1.2 Bilinear Transformation

Unlike the impulse-invariant design method, the design based on bilinear transformation does not require the partial fraction expansion of $G(s)$. It is just a matter of substituting a function of z for each Laplace operator s appearing in $G(s)$. The function that substitutes the operator s is given by

$$s = \frac{2}{T} \left[\frac{1-z^{-1}}{1+z^{-1}} \right]. \quad (3.10)$$

This transformation is recognized as the bilinear transformation. As a result of the transformation, the imaginary axis in the s -plane maps onto the unit circle in the z -plane, the left-half of the s -plane maps onto the inside of the unit circle and the right-half of the s -plane maps onto the outside of the unit circle.

Stable analog filters are mapped onto stable digital filters by the bilinear transformation.

3.2.1.3 Frequency Transformations

The idea here is to first design a digital lowpass filter and then use algebraic transformations to design the required frequency selective digital filter. This procedure can be applied regardless of the design procedure used to obtain the digital lowpass filter. Transformation functions from lowpass to lowpass, highpass, bandpass, and bandstop filters are readily available [4,26,33].

3.2.2 Time-Domain Design of IIR Filters

In this method a filter is designed such that its impulse response approximates a desired impulse response.

With the z-transform of the filter given by

$$H(z) = \frac{\sum_{k=0}^{M-1} b(k) z^{-k}}{1 + \sum_{k=0}^{N-1} b(k) z^{-k}} = \sum_{k=0}^{\infty} h(k) z^{-k},$$

it is required that the filter impulse response $h(k)$ approximate a desired $g(k)$, over the range $0 \leq k \leq P-1$.

Under a wide variety of conditions, Burrus and Parks, and Brophy and Salazar, among others, have shown that it is possible to find a set of $a(k), b(k)$, such that

$$\langle \epsilon \rangle = \sum_{k=0}^{P-1} [g(k) - h(k)]^2 w(k) \quad (3.11)$$

is minimized over all possible choices of $a(k), b(k)$ where $w(k)$ is a positive weighting function on the error sequence. Since $h(k)$ is a nonlinear function of the filter parameters ($\{a(k)\}, \{b(k)\}$), generally the minimization of ϵ can only be obtained using iterative techniques [33].

3.2.3 Optimization Methods of Designing IIR Filters

Here a mathematical optimization procedure is used to determine the filter coefficients that minimize either the squared error or the p-error. These errors are defined as follows.

Let $H(e^{j\omega})$ and $H_d(e^{j\omega})$ be the actual and required frequency responses, respectively, and let $\{\omega_i, i=1,2,\dots,M\}$ be the discrete set of frequencies at which the error between the actual and desired responses is evaluated. Then, the squared error at these frequencies is given by

$$E_2 = \sum_{i=1}^M \left[|H(e^{j\omega_i})| - |H_d(e^{j\omega_i})| \right]^2, \quad (3.12)$$

and the p-error is defined as a discrete approximation to either

$$E_p = \int_0^\pi W(\omega) \left[|H(e^{j\omega})| - |H_d(e^{j\omega})| \right]^p d\omega, \quad (3.13)$$

or

$$E_p = \int_0^\pi W(\omega) \left[\tau(\omega) - \tau_d(\omega) \right]^p d\omega, \quad (3.14)$$

where the group delay τ is defined as

$$\tau(\omega) = \frac{-d}{d\omega} \left\{ \arg[H(e^{j\omega})] \right\},$$

and $W(\omega)$ is a weighting function. Several Optimization design procedures are available for use to solve for the coefficients [26].

3.3 2-D FIR DIGITAL FILTER DESIGN

These filters share many characteristics with their 1-D counterparts. They are, for example, always stable as their impulse responses are always summable. They can also have real frequency response functions - zero-phase filters. Consequently, the design algorithms for 2-D FIR filters are closely related to 1-D algorithms.

A 2-D FIR filter has input-output relations given by

$$y(n_1, n_2) = \sum_{k_1} \sum_{k_2} h(k_1, k_2) x(n_1 - k_1, n_2 - k_2), \quad (3.15)$$

where k_1, k_2 , included in the sum, are finite in extent and are referred to as the region of support R .

The filter design is thus concerned with the selection of impulse response coefficients $h(k_1, k_2)$, consistent with a given region of support R , which yields a filter that approximates, in some sense, either an ideal impulse response or an ideal frequency response.

3.3.1 2-D FIR Filter Design Using Windows

2-D FIR filter design using windows is a straightforward extension of the 1-D technique. The filter impulse response $h(n_1, n_2)$ is determined as the product of the ideal impulse response $i(n_1, n_2)$ and another array $w(n_1, n_2)$, the window function. Thus,

$$h(n_1, n_2) = i(n_1, n_2)w(n_1, n_2). \quad (3.16)$$

The ideal impulse response $i(n_1, n_2)$ is generally presumed to have infinite support. By confining the support of $w(n_1, n_2)$ to R , however, $h(n_1, n_2)$ is also confined to R . Hence, this technique produces a filter with the required support.

Since h is the product of i and w , the frequency responses $H(\omega_1, \omega_2)$ and $I(\omega_1, \omega_2)$ are related by the convolution sum

$$H(\omega_1, \omega_2) = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} I(\Omega_1, \Omega_2) W(\omega_1 - \Omega_1, \omega_2 - \Omega_2) d\Omega_1 d\Omega_2, \quad (3.17)$$

where $W(\omega_1, \omega_2)$ is the Fourier transform of the window function.

The window function chosen is required to have a region of support R and its frequency response $W(\omega_1, \omega_2)$ should approximate an impulse function if $H(\omega_1, \omega_2)$ is to approximate $I(\omega_1, \omega_2)$. Also, for zero-phase filters the window should satisfy the zero-phase relation,

$$w(n_1, n_2) = w^*(-n_1, -n_2).$$

1-D windows are often used as a basis for generating 2-D windows. Two methods are available for converting 1-D windows to 2-D ones, depending on whether the region R is rectangular or circular [2,26].

For rectangular R , the window is formed as an outer product of two 1-D windows, that is

$$w_R(n_1, n_2) = w_1(n_1)w_2(n_2). \quad (3.18)$$

For a circular R , the window is formed by sampling a circularly rotated 1-D continuous window function, that is

$$w_C(n_1, n_2) = w(\sqrt{n_1^2 + n_2^2}). \quad (3.19)$$

The resulting 2-D windows have nearly circular regions of support. Among the popular 1-D windows that are used to form 2-D windows are the rectangular, the Hanning and the Kaiser windows.

3.3.2 2-D FIR Filter Design Using Transformations

In this method 1-D zero-phase FIR filter is transformed into 2-D zero-phase filter by means of a substitution of variables. The design of high-order 2-D filter is thus decoupled into the design of high-order 1-D filters and a low-order transformation [21]. This method, however, can only be used in the design of zero-phase filters.

The frequency response of a zero-phase 1-D FIR filter can be shown to be [2,19,21]

$$H(\omega) = \sum_{n=0}^N a(n) T_n[\cos \omega],$$

where $T_n(x)$ is the n^{th} Chebyshev polynomial.

By making the substitution

$$F(\omega_1, \omega_2) \longrightarrow \cos \omega,$$

$F(\omega_1, \omega_2)$ being the transformation function, the 2-D frequency response becomes

$$H(\omega_1, \omega_2) = \sum_{n=0}^N a(n) T_n[F(\omega_1, \omega_2)]. \quad (3.20)$$

For $H(\omega_1, \omega_2)$ to correspond to the frequency response of an FIR filter, it can be shown that $F(\omega_1, \omega_2)$ itself must be the frequency response of a 2-D FIR filter; but it can be of low order.

The simplest choice for $F(\omega_1, \omega_2)$ is for it to be the frequency response of a 3x3 filter. In that case

$$F(\omega_1, \omega_2) = A + B\cos(\omega_1) + C\cos(\omega_2) + D\cos(\omega_1 - \omega_2) + E\cos(\omega_1 + \omega_2),$$

where A, B, C, D and E are free parameters.

Since $T_n[x]$ is a polynomial of degree n in x , it follows that $H(\omega_1, \omega_2)$ is a polynomial of degree N in F . The filter with frequency response $H(\omega_1, \omega_2)$ can therefore be realized by cascade and parallel combinations of identical networks each with frequency response $F(\omega_1, \omega_2)$.

3.3.3 Optimal 2-D FIR Filter Design

If $I(\omega_1, \omega_2)$ is the frequency response of the required filter and $H(\omega_1, \omega_2)$ is the frequency response of the filter that we design, then we define the error between these two as

$$E(\omega_1, \omega_2) = H(\omega_1, \omega_2) - I(\omega_1, \omega_2).$$

Optimization techniques are then used to determine the filter coefficients that minimize certain functions of this error, such as the L_2 -norm

$$E_2 = \frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |E(\omega_1, \omega_2)|^2 d\omega_1 d\omega_2, \quad (3.21)$$

the L_p -norm

$$E_p = \left[\frac{1}{4\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} |E(\omega_1, \omega_2)|^p d\omega_1 d\omega_2 \right]^{1/p}, \quad (3.22)$$

and the Chebyshev (L_∞) norm

$$E_\infty = \max_{(\omega_1, \omega_2)} |E(\omega_1, \omega_2)|. \quad (3.23)$$

3.4 2-D IIR FILTER DESIGN

Although 2-D difference equations represent a generalization of 1-D difference equations, they are considerably more complex and are, in fact, quite different. A number of important issues associated with 2-D difference equations, such as the direction of recursion and the ordering relation, are not issues in the 1-D case. Other issues, such as stability, although present in the 1-D case, are far more difficult to analyze for 2-D systems [2].

With the transfer function of 2-D IIR filter given by

$$H(z_1, z_2) = \frac{\sum_{l_1=0}^{L_1-1} \sum_{l_2=0}^{L_2-1} a(l_1, l_2) z_1^{-l_1} z_2^{-l_2}}{\sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} b(k_1, k_2) z_1^{-k_1} z_2^{-k_2}} \triangleq \frac{A_z(z_1, z_2)}{B_z(z_1, z_2)}, \quad (3-24)$$

the design of a 2-D IIR filter is a process of finding the

coefficients, $a(n_1, n_2)$ and $b(k_1, k_2)$, such that the filter satisfies certain performance specifications. Some of these methods are briefly discussed below.

3.4.1 Space-Domain Design Techniques

Suppose $d(n_1, n_2)$ is the required output signal to a given input signal $x(n_1, n_2)$ and $y(n_1, n_2)$ is the actual output signal. The mean-squared error given by

$$e_2 \triangleq \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} [y(n_1, n_2) - d(n_1, n_2)]^2,$$

is required to be minimized and the filter coefficients which give rise to this minimized mean-squared error are the required filter coefficients.

In most derivations of the design algorithm, it is assumed that $a(n_1, n_2)$, $b(n_1, n_2)$, $x(n_1, n_2)$, $y(n_1, n_2)$ and $d(n_1, n_2)$ have their support confined to the first quadrant. The summation must also have finite limits for computational tractability [2].

The error e_2 can be minimized in theory by setting its derivatives with respect to the parameters $\{a(n_1, n_2), b(n_1, n_2)\}$ equal to zero. It is generally not possible to solve analytically for the coefficient values which minimize e_2 . Algorithmic methods, such as the Shank's method, the Descent methods and the Iterative Prefiltering methods, are thus employed [2].

3.4.2 Frequency Transformations

Given a transfer function of a 2-D digital IIR filter in the form of a rational function and assuming that the transfer function is in the first quadrant and stable, the problem becomes one of finding a 2-D-to-2-D transformation characterized by the mapping functions

$$z_1^{-1} = F_1(z_1, z_2), \text{ and}$$

$$z_2^{-1} = F_2(z_1, z_2).$$

These transformations are required to :

- i) produce stable first quadrant transfer function from stable first quadrant transfer functions,
- ii) map real rational functions into real rational functions, and
- iii) preserve some important basic characteristics of the amplitude response (such as ripple magnitude in pass and stopband regions) while altering other characteristics (such as cutoff frequencies or the number and shape of pass and stopband regions).

From the above conditions, it follows that F_1 and F_2 must be all-pass transfer functions and the general form of the spectral transformation for first quadrant filters is [2,3,16]

$$G_i(z_1, z_2) = \frac{\sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f_i(n_1, n_2) z_1^{-n_1} z_2^{-n_2}}{z_1^{-N_1} z_2^{-N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} f_i(n_1, n_2) z_1^{n_1} z_2^{n_2}} \quad (3.25)$$

where $i=1,2$.

The usual practice is to design a prototype lowpass, digital filter and then use algebraic transformations to design other filters. Some transformation functions are readily available for that purpose [3,16].

But, because of the limited number of parameters, it is apparent that there are not sufficient degrees of freedom to specify a transformation to produce any arbitrary frequency response from a given one. For example, it is not possible to obtain a circularly symmetric highpass filter from a circularly symmetric lowpass filter [3].

Transformation functions do also exist that map 1-D IIR prototype filters into 2-D IIR filters [2,16].

Rotated filters may also be obtained by rotating the transfer function of a 1-D continuous filter into a continuous 2-D filter and subsequently transforming the 2-D continuous filter into a 2-D digital filter bilinearly [17,18].

3.5 IMPLEMENTATION OF DIGITAL FILTERS

Digital filters can be implemented as special purpose hardware units or as a program which is run on a general purpose computer. If digital filters are required to operate in the real time, then the hardware implementation is the solution [1,24,33]. As for the algorithms used to implement digital filters, they depend on the type of the filter [2,26,33]. But, in general, an implementation method is not exclusively used to design a particular type of filter. Thus, FIR filters can be realized recursively (a domain of IIR filters) and IIR filters may be implemented by the direct convolution or discrete Fourier transform (DFT) methods as shown by B. Gold and K.L. Jordan [34].

3.5.1 FIR Filter Realizations

There are mainly two algorithms that help implement FIR filters, namely, the direct convolution and the DFT implementation.

1) Direct convolution - It has been shown in Chapter 2 that the input-output relation of FIR filters are of the form

$$y(n) = \sum_k h(k)x(n-k), \quad \text{1-D case}$$

$$y(n_1, n_2) = \sum_{k_1} \sum_{k_2} h(k_1, k_2)x(n_1-k_1, n_2-k_2), \quad \text{2-D case}$$

where the limits of summation are finite and outside these limits

the impulse response is zero.

Therefore, if all the input samples are available and the filter coefficients are known, then the convolution sums given above are used to implement the filtering operation. This method of implementation has the advantage that there is little arithmetic quantization error [35].

2) Discrete Fourier Transform (DFT) Implementation - This method is preferred to the direct convolution method for high-order filters because fast Fourier transform algorithms can be employed to efficiently perform the DFT operations. The steps followed in this method are :

i) The DFTs of the input and impulse response arrays are computed using FFT algorithms. The duration (1-D case) or the region of support (2-D case) has to be extended with sample values of zero to get the linear convolution by this method. Otherwise, we end up with circular convolution.

ii) These DFTs are multiplied point by point.

iii) The inverse discrete Fourier transform (IDFT) of the product is taken, once again, using FFT algorithms. The result is the linear convolution of the input and impulse response arrays - the output.

The DFT is generally evaluated either by means of a row-column decomposition of the DFT sum thus dividing the multidimensional

DFT computation into the computation of a number of 1-D DFTs [2,35] or by means of the vector-radix algorithm [2,36,37]. This latter approach needs 25% fewer complex multiplications than the former [2]. Mersereau and Speake [38] have also generalized the 1-D FFT algorithm to the multidimensional case.

DFT implementations of FIR filters are efficient with respect to speed but prodigal with respect to storage [2].

3.5.2 IIR Filter Realizations

Direct, cascade and parallel, and iterative implementation techniques are some of the methods used to realize the IIR filters [1,2,24,26,33,34].

a) Direct form implementation - This is a method by which the filter is implemented by rearranging the difference equation to express the output samples in terms of the input samples and previously computed output samples.

b) Cascade and Parallel implementation - Here an IIR filter is constructed from a cascade or parallel interconnections of simpler IIR filters.

c) Iterative implementations - This method is primarily developed to deal with 2-D filters with impulse responses that are not recursively computable and, unlike the 1-D case, cannot be

factored.

By making an educated guess at the output, Dudgeon and Mersereau [2] have shown that a better approximation for the output is computed as per the following equation

$$y_i(n_1, n_2) = a(n_1, n_2) ** x(n_1, n_2) + c(n_1, n_2) ** y_{i-1}(n_1, n_2), \quad (3.26)$$

or in the frequency domain

$$Y_i(\omega_1, \omega_2) = A(\omega_1, \omega_2) X(\omega_1, \omega_2) + C(\omega_1, \omega_2) Y_{i-1}(\omega_1, \omega_2), \quad (3.27)$$

where
$$Y_i(\omega_1, \omega_2) = \frac{A(\omega_1, \omega_2)}{B(\omega_1, \omega_2)} X(\omega_1, \omega_2),$$

$$C(\omega_1, \omega_2) = 1 - B(\omega_1, \omega_2),$$

$$y_i(n_1, n_2) = i^{\text{th}} \text{ approximation to the output signal } y(n_1, n_2),$$

and $x(n_1, n_2)$ = the input signal.

The above approximation converges if $|C(\omega_1, \omega_2)| < 1$.

4. SOFTWARE IMPLEMENTATION OF 2-D FIR DIGITAL FILTERS

In this chapter, the overall design procedure and the turbo pascal program written to implement this procedure will be discussed. The most important constituent subroutines will also be elaborated. Discussed in this chapter will also be the subroutines that result in the filtering of digital images with any of the standard filters. Finally, the flow charts for the design program and the filtering operations are given.

4.1 FIR FILTER DESIGN USING THE KAISER WINDOW METHOD

The FIR filter will be considered in this work because it is always stable and so the design algorithm does not need to include a test for stability. This leads to a considerable reduction in the complexity of the software to be designed. Moreover, FIR filters have linear phase characteristics and hence phase distortion in the output response is avoided.

The window method was chosen to design the 2-D FIR digital filter because it is simple and straightforward to use and generally produces a satisfactory design (although it may not always give designs of the lowest order). And of the different windows, the Kaiser window was chosen for the following reasons:

i) Unlike in the spectrum analysis problem where the Fourier

transform of the window $W(\omega_1, \omega_2)$ itself is of paramount importance, in the filter design problem it is the properties of the convolution of $W(\omega_1, \omega_2)$ with the step function that is of concern. The major properties in the frequency domain of a window used in the design of filters are the width of the "main" lobe of $W(\omega_1, \omega_2)$ and the relative sidelobe amplitude [39]. And the Kaiser windows are nearly optimum in the sense of having the largest energy in the main lobe for a given side lobe amplitude [26].

ii) A parameter (α) in the expression for the Kaiser window is used to adjust the trade off between main lobe width and side lobe ripple. The presence of this parameter makes the Kaiser window particularly flexible and versatile.

4.1.1 The Design Procedure

The design procedure starts by making prompts as to what filter type (lowpass, bandpass, highpass or bandstop) the user wants to design and their corresponding specifications to which the user responds by typing the filter type and its specifications. The specifications are in the frequency domain and include the sampling frequency, the passband limit(s), the transition width (all in kHz) and the ripples in the passband(s) and stopband(s).

By convention, the independent variable of the mathematical representation of a signal is taken to be time although it may

not actually represent time. It is because of this convention then that the reciprocal is considered frequency and its units given in kHz.

The normalized passband limit and transition bandwidths are then obtained by dividing the corresponding analog quantities by the sampling frequency [1]. Thus, if the sampling frequency is F_s , the analog passband limit is f_p and the analog transition bandwidth is f_t , then the corresponding normalized quantities are given by

$$\text{Normalized Passband Limit (PB)} = f_p / F_s, \quad (4.1)$$

$$\text{Normalized Transition bandwidth (TB)} = f_t / F_s. \quad (4.2)$$

The normalized quantities are thus dependent on the ratio of their respective analog quantities to the sampling frequency. The exact analog values do not have any bearing on the design procedure as long as the ratios remain the same. The procedure followed to determine the window function is based on the work of T.C. Speake and R.M. Mersereau [20]. The procedure is outlined below.

Given the passband limit (PB), the transition bandwidth (TB), the ripple in the passband (δ_p) and the ripple in the stopband (δ_s), the following quantities are computed.

$$ATT = -20 \log \sqrt{\delta_p \delta_s}, \quad (4.3)$$

$$N = (ATT - 7.00) / (13.68TB), \quad (4.4)$$

$$R = (N - 1) / 2, \quad (4.5)$$

$$\alpha = \begin{cases} 0.56(ATT - 20.2)^{0.4} + 0.083(ATT - 20.2), & 20.2 < ATT < 60. \\ 0, & ATT < 20.2. \end{cases} \quad (4.6)$$

where $ATT =$ Attenuation,

δ_p = Geometric mean of the ripples in the pass- and stop-bands in dB,

$N =$ The order of the circular window,

$R =$ Radius of the circular window, and

$\alpha =$ an adjustable window parameter that specifies a frequency domain trade off between main lobe width and side lobe ripple.

The circular window function is thus determined as :

$$w(m, n) = \begin{cases} \frac{I_0(\alpha \sqrt{1 - (m^2 + n^2) / R^2})}{I_0(\alpha)}, & \sqrt{m^2 + n^2} \leq R. \\ 0, & \sqrt{m^2 + n^2} > R. \end{cases} \quad (4.7)$$

where $I_0(x)$ is the modified Bessel function of the first kind of zeroth order and is given by [40]

$$I_0(x) = \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m} m! \Gamma(m+1)} = \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m} (m!)^2} \quad (4.8)$$

and $I_0(0)=1$, by definition.

The window used here is circular window as it requires fewer nonzero coefficients and seems to be more predictable [20].

The ideal impulse response, $i(m,n)$, is then approximated by sampling the required frequency response and then performing an inverse discrete Fourier transform (IDFT) operation on it. The aliasing error resulting from such an operation is minimized by making the support of the IDFT much larger than the extent of the region of support of the window R [2,26].

In this design, circularly symmetric filters are considered not only because passband and stopband regions of such filters are easy to specify but also because they can be designed to have zero or linear phase characteristics.

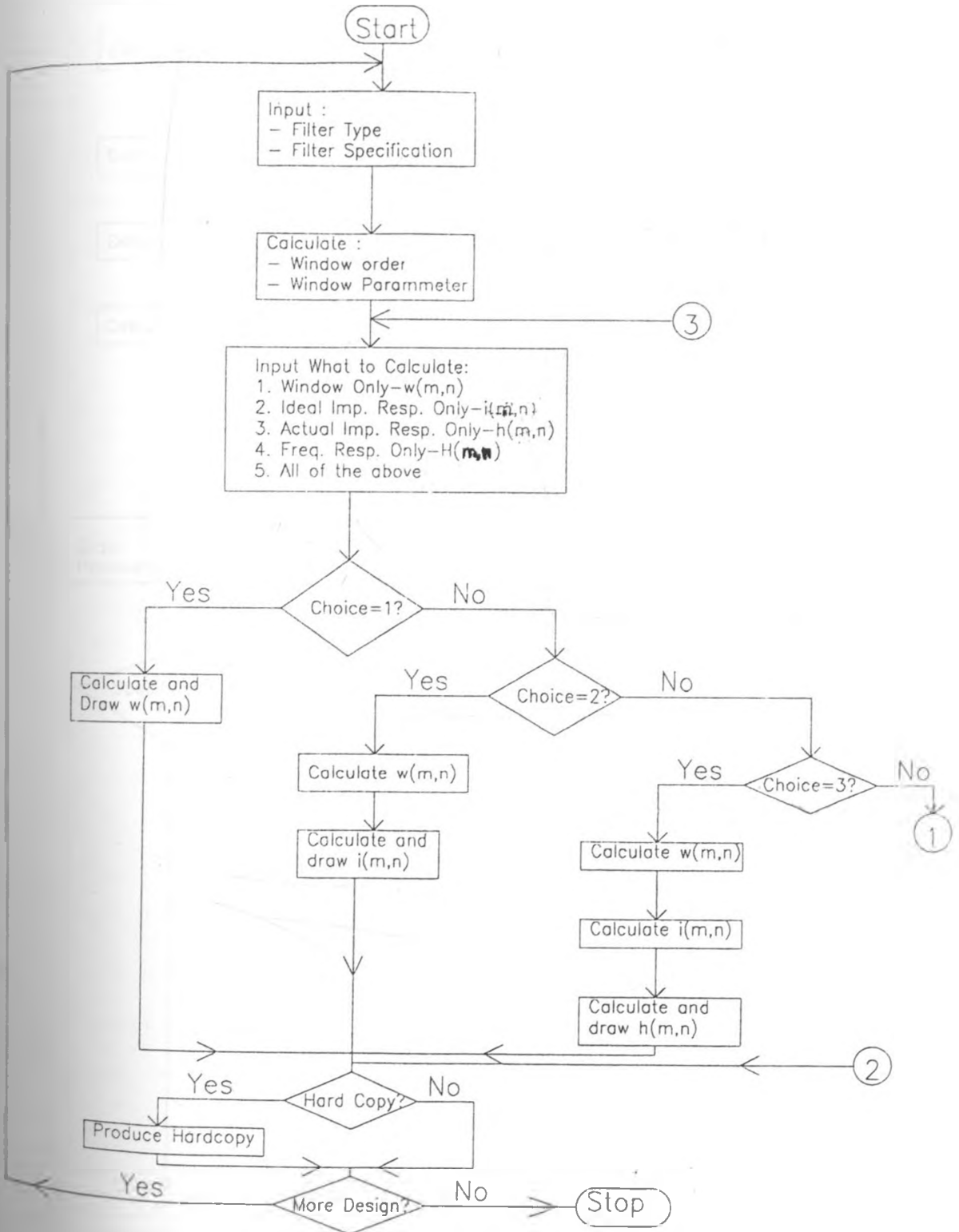
The ideal impulse response is then multiplied point by point with the window function to obtain the actual impulse response of the filter required, i.e.

$$h(m,n) = i(m,n) \cdot w(m,n), \quad (4.9)$$

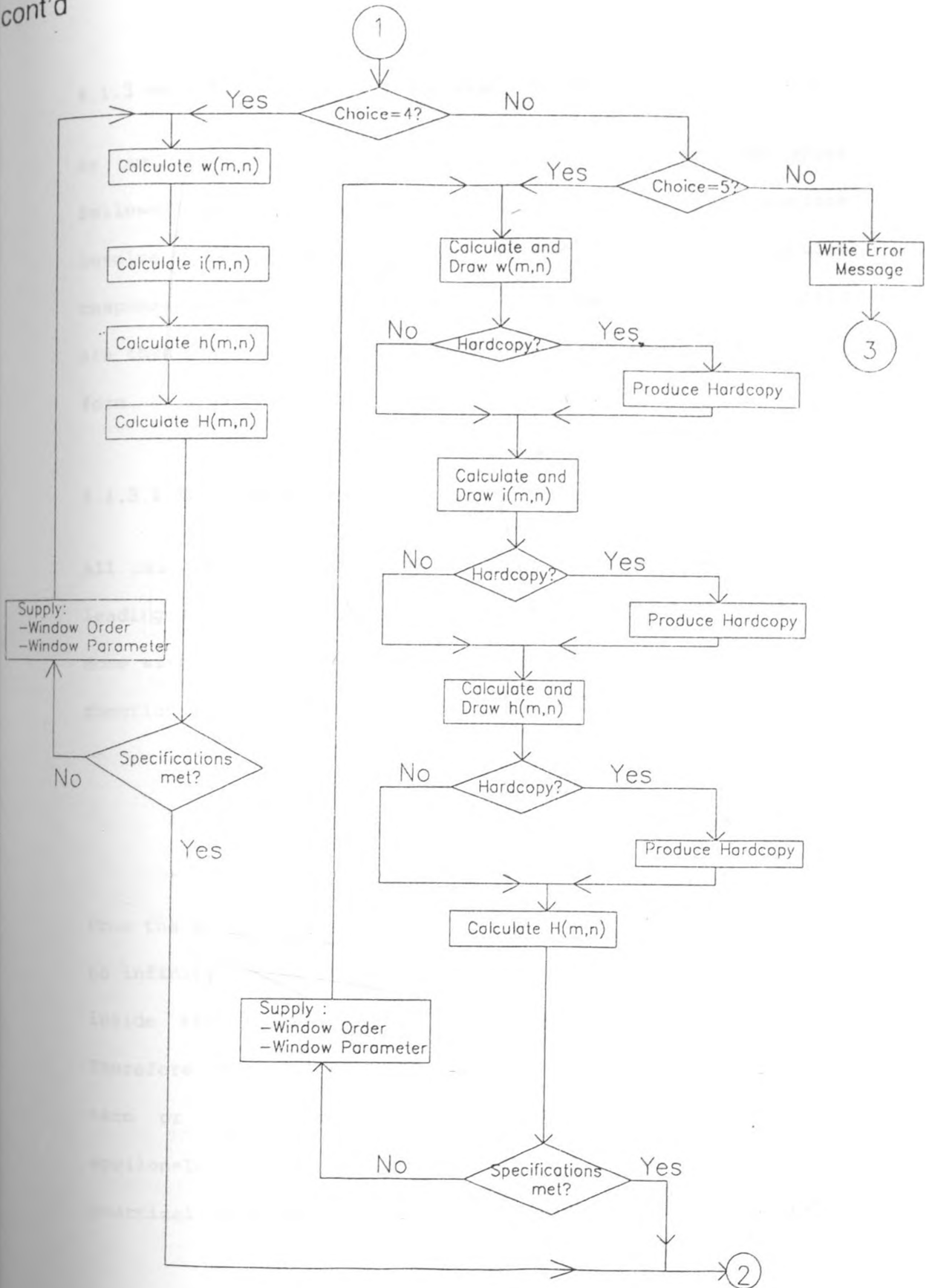
where $h(m,n)$ = the required impulse response,
 $i(m,n)$ = the ideal impulse response, and
 $w(m,n)$ = the window function.

Finally, the DFT is performed on this impulse response to get the frequency response of the filter required.

4.1.2 Flow Chart For The Filter Design



cont'd



4.1.3 Main Subroutines in the Design Program

As outlined in the design procedure section, the major steps followed in designing the filter with given specifications involve the computations of the window function, the impulse response and the frequency response. These computed quantities are then given out in tabular form and, if required, in 3-D graph form.

4.1.3.1 The Window

All calculations, apart from the modified Bessel function (I_0), leading to the window function are straightforward as each is done with one or two statements. But to calculate I_0 , a separate function subroutine is required. The formula for I_0 is [40]

$$I_0(x) = \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m} (m!)^2} = \sum_{m=0}^{\infty} \left[\frac{(x/2)^m}{m!} \right]^2 \quad (4.10)$$

From the above, it is apparent that the range of m extends from 0 to infinity but it can also be seen that, for finite x , the term inside the square bracket gets smaller as m gets larger. Therefore the computation of the sum can be terminated when this term or its square gets below a set minimum value, say $\epsilon = 10^{-8}$. The series converges so fast that, for all practical purposes, the error introduced by leaving out terms

less than this set minimum is insignificant.

The function subroutine is then written as follows

```

begin
  if x=0 then I0=1    { I0(0)=1, by definition.}
  else begin
    x:=x/2;
    m:=1;
    sum:=0;
    term:=1;    { term for m=0.}
    while term>=epsilon do begin { epsilon is set to 10-8 }
      sum:=sum+term;
      y:=power(x,m)/factorial(m);
      term:=y*y;
      m:=m+1;
    end;
    I0:=sum;
  end;
end;
end;

```

In the above, power(x,m) is a previously defined function that raises x to the power of m and factorial(m) is a predefined* function that calculates the factorial of m ,i.e. m!.

Also, in the computation of the window function, as the window is circularly symmetric, only the first quadrant quantities need to be computed using the formula. These are then reflected into the 2nd quadrant as follows:

```

for m:=-RMax to -1 do
  for n:=0 to RMax do
    w[m,n]:=w[-m,n];

```

where RMax is the maximum possible radius of the window function (equal to 32 in the program).

The 1st and 2nd quadrants are in turn reflected into 3rd and 4th quadrants as follows

```

for m:=-RMax to RMax do
  for n:=-RMax to -1 do
    w[m,n]:=w[m,-n];

```

4.1.3.2 The Impulse Response

The inverse DFT relation is given by

$$i(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} I(k_1, k_2) \exp\left(j \frac{2\pi}{N_1} n_1 k_1 + j \frac{2\pi}{N_2} n_2 k_2\right), \quad (4.11)$$

where $I(k_1, k_2)$ is the sample of the specified frequency response and N_1, N_2 define the region of support of the inverse DFT.

For $N_1 = N_2 = N_m$ and a zero-phase filter, let $R_m = (N_m - 1)/2$. The inverse DFT is then written as

$$i(n_1, n_2) = \frac{1}{N_m^2} \sum_{k_1=-R_m}^{R_m} \sum_{k_2=-R_m}^{R_m} I(k_1, k_2) \exp\left(j \frac{2\pi}{N_m} n_1 k_1 + j \frac{2\pi}{N_m} n_2 k_2\right).$$

Letting $r_1 = \frac{2\pi}{N_m} n_1$ and $r_2 = \frac{2\pi}{N_m} n_2$, the above reduces to

$$i(n_1, n_2) = \frac{1}{N_m^2} \sum_{k_1=-R_m}^{R_m} \sum_{k_2=-R_m}^{R_m} I(k_1, k_2) \exp(jr_1 k_1 + jr_2 k_2).$$

Expanding the above we get

$$\begin{aligned}
 i(n_1, n_2) = \frac{1}{N_m^2} & \left\{ I(0, 0) + \sum_{k_1=1}^{R_m} \sum_{k_2=1}^{R_m} \left[I(k_1, k_2) \exp(jr_1 k_1) \exp(jr_2 k_2) \right. \right. \\
 & + I(-k_1, k_2) \exp(-jr_1 k_1) \exp(jr_2 k_2) \\
 & + I(-k_1, -k_2) \exp(-jr_1 k_1) \exp(-jr_2 k_2) \\
 & \left. \left. + I(k_1, -k_2) \exp(jr_1 k_1) \exp(-jr_2 k_2) \right] \right. \\
 & + \sum_{k_1=1}^{R_m} \left[I(k_1, 0) \exp(jr_1 k_1) + I(-k_1, 0) \exp(-jr_1 k_1) \right] \\
 & \left. + \sum_{k_2=1}^{R_m} \left[I(0, k_2) \exp(jr_2 k_2) + I(0, -k_2) \exp(-jr_2 k_2) \right] \right\} .
 \end{aligned}$$

Using the symmetric relation

$$I(k_1, k_2) = I(\pm k_1, \pm k_2)$$

in the expression for the IDFT, we get

$$\begin{aligned}
 i(n_1, n_2) = \frac{1}{N_m^2} & \left\{ I(0, 0) + \sum_{k_1=1}^{R_m} \sum_{k_2=1}^{R_m} I(k_1, k_2) \left[\exp(jr_1 k_1) \left(\exp(jr_2 k_2) \right. \right. \right. \\
 & \left. \left. + \exp(-jr_2 k_2) \right) + \exp(-jr_1 k_1) \left(\exp(jr_2 k_2) + \exp(-jr_2 k_2) \right) \right] \\
 & + \sum_{k_1=1}^{R_m} I(k_1, 0) \left[\exp(jr_1 k_1) + \exp(-jr_1 k_1) \right] \\
 & \left. + \sum_{k_2=1}^{R_m} I(0, k_2) \left[\exp(jr_2 k_2) + \exp(-jr_2 k_2) \right] \right\} .
 \end{aligned}$$

Rearranging and simplifying, we get

$$i(n_1, n_2) = \frac{1}{N_m^2} \left\{ I(0, 0) + \sum_{k_1=1}^{R_m} \sum_{k_2=1}^{R_m} 4I(k_1, k_2) \cos(r_1 k_1) \cos(r_2 k_2) \right. \\ \left. + \sum_{k_1=1}^{R_m} 2I(k_1, 0) \cos(r_1 k_1) + \sum_{k_2=1}^{R_m} 2I(0, k_2) \cos(r_2 k_2) \right\}.$$

The above can be generalized to

$$i(n_1, n_2) = \frac{1}{N_m^2} \sum_{k_1=0}^{R_m} \sum_{k_2=0}^{R_m} \left[A \cdot I(k_1, k_2) \cos(r_1 k_1) \cos(r_2 k_2) \right], \quad (4.12)$$

where

$$A = \begin{cases} 1 & \text{if } k_1 = k_2 = 0, \\ 4 & \text{if } (k_1 \neq 0, k_2 \neq 0), \\ 2 & \text{if } (k_1 = 0, k_2 \neq 0) \text{ or } (k_1 \neq 0, k_2 = 0), \end{cases}$$

and

$$I(k_1, k_2) = \begin{cases} 1, & \text{in the passband.} \\ 0, & \text{in the stopband.} \end{cases}$$

In the program, the expression inside the square bracket (Eqn. 4.12) is called "term" and thus the expression for the ideal impulse response becomes

$$i(n_1, n_2) = \frac{1}{N^2} \sum_{k_1=0}^R \sum_{k_2=0}^R \text{Term},$$

where

$$\text{Term} = \begin{cases} I(k_1, k_2) & , \text{ if } k_1 = k_2 = 0 . \\ 4I(k_1, k_2) \cos(r_1 k_1) \cos(r_2 k_2) & , \text{ if } (k_1, k_2 \neq 0) . \\ 2I(k_1, k_2) \cos(r_1 k_1) \cos(r_2 k_2) & , \text{ if } (k_1 = 0, k_2 \neq 0) \text{ or} \\ & (k_1 \neq 0, k_2 = 0) . \end{cases}$$

A considerable part of the program goes into determining the values of k_1 and k_2 for which the filter is in the stopband, or passband. Different types of filters have different ranges. For example, for a lowpass filter, Term in the expression for $i(n_x, n_y)$ is determined as follows.

```

if (k1=0) and (k2=0) then term:=1           { At origin. }
else if (k1=0) or (k2=0) and (R<=NCutOff) { passband and }
    then term:=2*cos(r1*k1)*cos(r2*k2)     { on axes. }
else if (R<=NCutOff)                       { passband but }
    then term:=4*cos(r1*k1)*cos(r2*k2)     { not on axes. }
else term:=0;                               { Stopband. }

```

where NCutOff is the normalized cutoff frequency.

To make use of symmetry in the impulse response computation, only quantities in the first quadrant are computed which are then reflected to the second quadrant. Reflecting the first and second quadrant quantities into the third and fourth quadrants give the total result.

Also, since quantities outside the range of support of the window function are anyway to be zero (after multiplication with the window), they need not be computed using the expression for

$i(n_1, n_2)$. They are instead assigned zero values.

Finally, the ideal impulse response is multiplied point by point with the window function as follows :

```

for n1:=-RMax to RMax do
for n2:=-RMax to RMax do
  h[n1,n2]:=i[n1,n2]*w[n1,n2];

```

4.1.3.3 The Frequency Response

Once the impulse response is determined, its frequency response can be computed from the DFT relation given by

$$H(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} h(n_1, n_2) \exp(-j\frac{2\pi}{N_1} n_1 k_1 - j\frac{2\pi}{N_2} n_2 k_2), \quad (4.13)$$

where $h(n_1, n_2)$ is the actual impulse response.

For $N_1=N_2=N_m$ and zero-phase filters, following the steps used in the ideal impulse response case, it can be shown that:

$$H(k_1, k_2) = \sum_{n_1=0}^{R_m} \sum_{n_2=0}^{R_m} A \cdot h(n_1, n_2) \cos(r_1 n_1) \cos(r_2 n_2), \quad (4.14)$$

where $R_m = (N_m - 1) / 2$,

$$r_1 = (2\pi k_1) / N_m,$$

$$r_2 = (2\pi k_2) / N_m.$$

The expression for the frequency response can alternatively be written as

$$H(k_1, k_2) = \sum_{n_1=0}^{R_m} \sum_{n_2=0}^{R_m} \text{Term}, \quad (4.15)$$

where

$$\text{Term} = \begin{cases} h(n_1, n_2), & \text{if } n_1 = n_2 = 0. \\ 4h(n_1, n_2) \cos(r_1 n_1) \cos(r_2 n_2), & \text{if } (n_1 \neq 0, n_2 \neq 0). \\ 2h(n_1, n_2) \cos(r_1 n_1) \cos(r_2 n_2), & \text{if } (n_1 = 0, n_2 \neq 0) \text{ or} \\ & (n_1 \neq 0, n_2 = 0). \end{cases}$$

As the region of support of the impulse response (or window order) is much smaller than that of the IDFT, computation time is saved if the range of summation is confined to the radius of the window (R) instead of the radius of support of the IDFT (R_m). The form of expression for the frequency response determination used in the program is, therefore, as follows

$$H(k_1, k_2) = \sum_{n_1=0}^R \sum_{n_2=0}^R \text{Term}. \quad (4.16)$$

Only frequency responses in the first quadrant (axes inclusive) are computed using the above equation. Second, third and fourth quadrant quantities are obtained by first reflecting the first quadrant quantities into the second and then reflecting second and first quadrant quantities into the third and fourth quadrants.

4.1.3.4 The Three-Dimensional Graph

The calculated filter coefficients and discrete frequency response values, as they stand, may not mean any thing to someone who looks at them; they are merely data. When they are plotted on 3-D graphs, however, they reveal all the necessary information that they represent. For that purpose, a 3-D graph subroutine has been written and is included in the program.

To draw a 3-D graph on a 2-D screen, transformations are used. In the screen, the top left corner has coordinates of (0,0) while the coordinates of the bottom right corner depend on the screen resolution. The Turbo Pascal functions GetMaxX and GetMaxY give the screen x and y resolutions, respectively. If we let XM and YM, respectively, be the x and y resolutions, the transformation expressions used in this work are given by :

$$x[m,n] = XM/2 - (n-1) * (XM / (4*64)) + (m-1) * (XM / (8*64))$$

$$y[m,n] = YM/1.4 - (n-1) * (YM / (8*64)) - (m-1) * (YM / (4*64)) - k * z[m,n]$$

where $x[m,n]$ and $y[m,n]$ are the screen x- and y-coordinates of a point with world coordinates of $(m,n,z[m,n])$,

64 is one less than the maximum value that the indices m and n can take - 65 in this case,

k is a factor that adjusts the resolution of the graph and is inversely proportional to $z[m,n]$, and

* is the multiplication operation.

Using these transformations, axes x_1 , x_2 and z (Fig. 4.1) are defined on the screen to correspond to the m and n indices and the dependent variable z , respectively, in the real world.

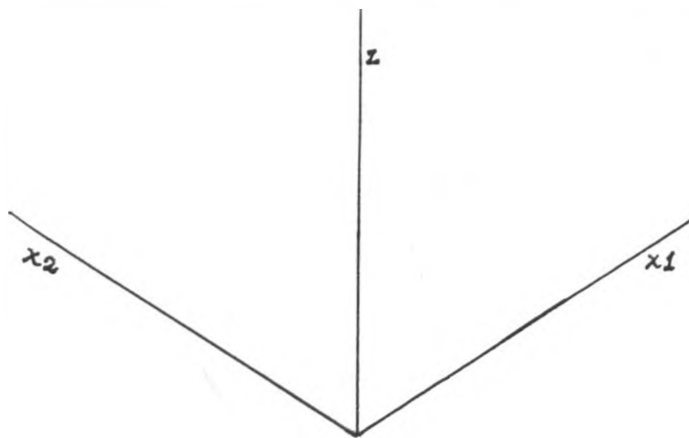


Fig.(4.1) The Three Axes

The 3-D graph is thus plotted as follows.

With the first index held at 1, the second index is varied from 1 to 65. This is the same as moving from the origin to the other end along line x_2 in Fig.1. During the movement, points with (x,y) coordinates corresponding to numbers with adjacent indices are joined with straight lines.

Next, the first index is incremented by 1 and the above procedure is repeated until the first index becomes equal to 65.

But, while repeating the procedure care should be taken that crossing lines are not drawn as these tend to mess up the whole picture. Thus, lines that may cross those in front of them are made invisible.

One method by which this crossing is reduced to a reasonably acceptable level, and the one used in the program, is to draw visible lines using the Turbo-Pascal function `LineTo` when it joins a point to a second one only when the second point has y-coordinates greater than, or less than, all the y-coordinates of points that come before it (i.e. points corresponding to numbers with smaller indices) but have the same x-coordinates as itself. Otherwise, the lines are made invisible by using the function `MoveTo` instead of `LineTo`.

The whole procedure above is repeated but with x_1 replaced by x_2 and vice-versa.

For all its power in handling graphics, Turbo-pascal lacks the ability to transfer a graphics screen information to a parallel printer. There is no single high-level routine for printing a graphics image, but it does provide several routines that can be used to accomplish this task. C.Ohlsen and G.Stoker [41] have written a unit to perform this task. This unit is included with the program.

A program with graphics routines works only when the machine that runs it includes a Turbo-Pascal(TP) compiler that contains Borland Graphics Interface (.BGI) device driver and Character (.CHR) font files and the path to the driver is indicated. But, all machines may not have the .BGI drivers and the .CHR fonts, or even the TP compiler.

One solution, suggested by M.Yester [42] and used in this work, is to include .BGI and .CHR files within the program itself. The procedure used in this program to do it is the so called external procedure method where .BGI and .CHR files are converted into object files with the BINOBJ.EXE program and then linked into the program.

In the program, the EGAVGA.BGI driver and LITT.CHR font are used and hence they are the only ones which are converted into object files and finally linked with the program.

4.2 APPLICATION TO IMAGE PROCESSING

One of the many applications of 2-D digital filters is in image processing which includes image enhancement. Image enhancement can be viewed as [5] selective emphasis and/or suppression of information in the images, with the aim of increasing its usefulness.

In this work, the enhancement effect of different standard digital filters on images of size 65x65 and 32 gray levels will be demonstrated. The procedure is as follows :

A coded data representing the gray levels of a 65x65 image is read from a file. The codes comprise characters ranging from 0 to 9 and A to V. These are then decoded to integers ranging from 0 to 31, the gray levels being proportional to the magnitudes of the integers. An integer array, IB, of size 65x65 is thus formed. The reading in of the image code and its decoding are handled by a subroutine called ''InputAndDecode''.

Then the images are displayed on the screen with 16 gray levels. The 32 gray levels in the data are reduced into 16 by dividing the given gray level by 2 and then rounding to the nearest integer. The technique of creating gray scales on the screen is taken from the works of Ohlsen and Stoker [41], with some modifications added later.

So far, the image seen is the original image without being subjected to corruption with noise or being processed with a filter. The next task would be to see the effect of introducing noise. Thus, positive impulse noise is put into the image by way of making 10% of the image elements take a gray value of 31, the maximum gray level possible. This is done with a subroutine called ''CorruptWithNoise''. Impulse noise falls on the high

side of the frequency spectrum and can be successfully filtered out by a lowpass filter. The new corrupted array replaces the original array. The new array is then displayed to see the effect of introducing noise.

Finally, both the corrupted and uncorrupted images are filtered with different types of filters to see the effect of each filter on the image.

The process of filtering can either be done in the frequency or spatial domain. Filtering in the spatial domain, which is the one used in this work, involves the process of convolving the impulse response of the filter with the array of gray levels of the image.

This convolution process is mathematically expressed as

$$y(n_1, n_2) = \sum_{k_1=0}^{N_m-1} \sum_{k_2=0}^{N_m-1} h(k_1, k_2) \cdot IB(n_1 - k_1, n_2 - k_2), \quad (4.17)$$

or, if the indices can also take negative values, as

$$y(n_1, n_2) = \sum_{k_1=-R_m}^{R_m} \sum_{k_2=-R_m}^{R_m} h(k_1, k_2) \cdot IB(n_1 - k_1, n_2 - k_2), \quad (4.18)$$

where $N_m \times N_m$ = region of support of the IDFT,

$R_m = (N_m - 1) / 2$ is the radius of region of support of IDFT,

$IB(n_1, n_2)$ = the array of gray levels of the image.

Since the result of convolution must fit into a 65x65 array, there is no need for computing numbers with indices outside the range $-32 \leq k_1, k_2 \leq 32$.

Also, since the impulse response outside the region of support is zero, there is no need for computing a convolution sum term contributed from impulse response outside the region of support.

Thus, the limits of summation are given by

$$-R \leq k_1, k_2 \leq R,$$

where R is the radius of the circular region of support of the impulse response.

The convolution sum then becomes

$$y(n_1, n_2) = \sum_{k_1=-R}^R \sum_{k_2=-R}^R h(k_1, k_2) \cdot IB(n_1 - k_1, n_2 - k_2). \quad (4.19)$$

Furthermore, only $(n_1 - k_1)$ and $(n_2 - k_2)$ values in the range -32 to 32 need to be included as array IB is defined only for indices within this region.

The number of multiplications involved in the convolution process can further be reduced if the circular symmetric property of the impulse response is taken into consideration.

For a circularly symmetric filter, we have

$$h(k_1, k_2) = h(\pm k_1, \pm k_2)$$

and thus we have

$$\begin{aligned} y(n_1, n_2) &= \sum_{k_1=-R}^R \left\{ \sum_{k_2=-R}^R h(k_1, k_2) \text{IB}(n_1 - k_1, n_2 - k_2) \right\} \\ &= \sum_{k_1=-R}^R \left\{ h(k_1, 0) \text{IB}(n_1 - k_1, n_2) \right. \\ &\quad \left. + \sum_{k_2=1}^R h(k_1, k_2) \left[\text{IB}(n_1 - k_1, n_2 - k_2) + \text{IB}(n_1 - k_1, n_2 + k_2) \right] \right\} \\ &= \sum_{k_1=-R}^R h(k_1, 0) \text{IB}(n_1 - k_1, n_2) + \sum_{k_1=-R}^R \sum_{k_2=1}^R h(k_1, k_2) \left[\right. \\ &\quad \left. \text{IB}(n_1 - k_1, n_2 - k_2) + \text{IB}(n_1 - k_1, n_2 + k_2) \right] \\ &= h(0, 0) \text{IB}(n_1, n_2) + \sum_{k_1=1}^R h(k_1, 0) \left[\text{IB}(n_1 - k_1, n_2) + \text{IB}(n_1 + k_1, n_2) \right] \\ &\quad + \sum_{k_2=1}^R \left\{ \sum_{k_1=-R}^R h(k_1, k_2) \left[\text{IB}(n_1 - k_1, n_2 - k_2) + \text{IB}(n_1 - k_1, n_2 + k_2) \right] \right\}. \end{aligned}$$

Rearranging and simplifying, we get

$$\begin{aligned} y(n_1, n_2) &= h(0, 0) \text{IB}(n_1, n_2) \\ &\quad + \sum_{k_1=1}^R h(k_1, 0) \left[\text{IB}(n_1 - k_1, n_2) + \text{IB}(n_1 + k_1, n_2) \right] \end{aligned}$$

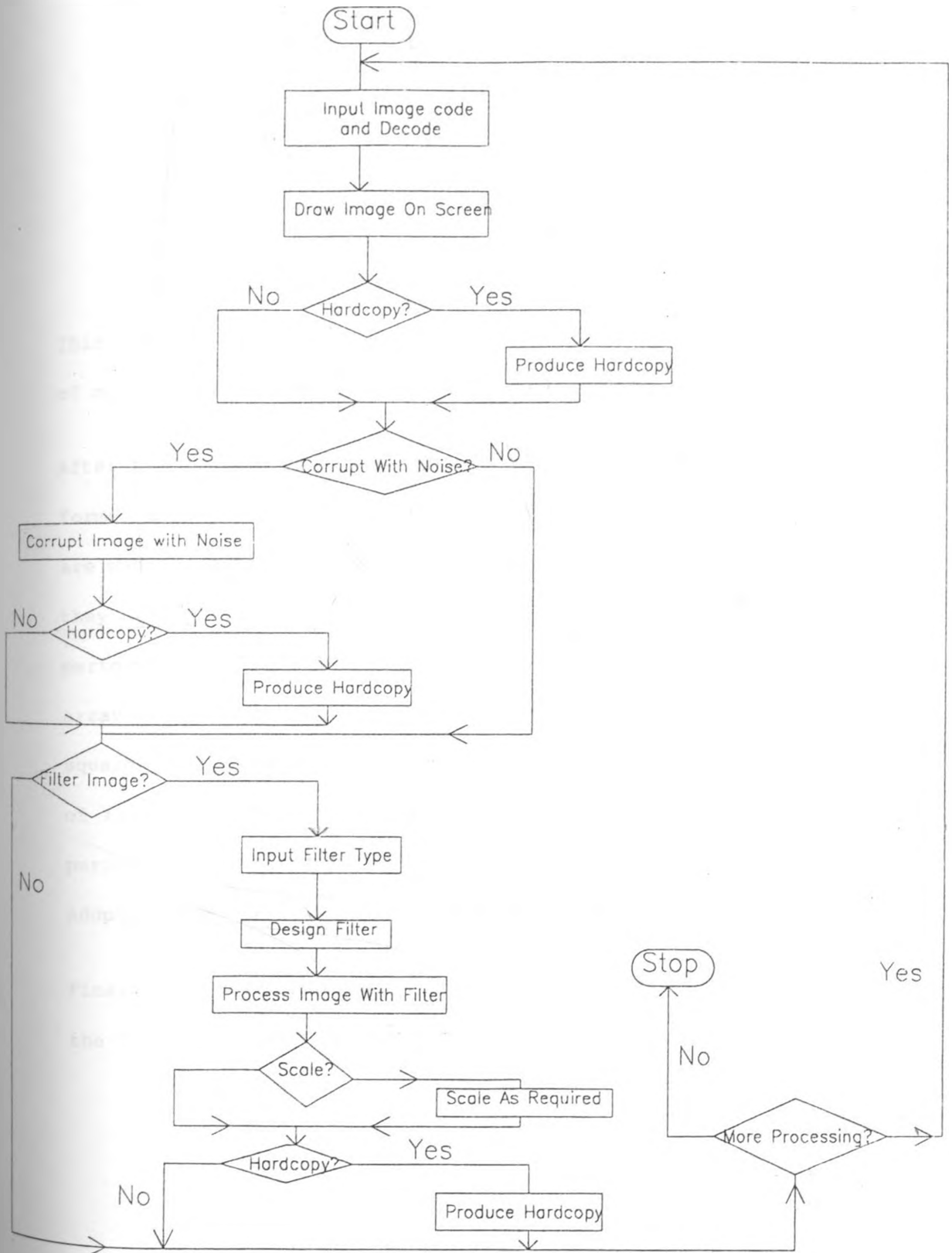
$$\begin{aligned}
& + \sum_{k_2=1}^R h(0, k_2) \left[IB(n_1, n_2 - k_2) + IB(n_1, n_2 + k_2) \right] \\
& + \sum_{k_1=1}^R \sum_{k_2=1}^R h(k_1, k_2) \left[IB(n_1 - k_1, n_2 - k_2) + IB(n_1 + k_1, n_2 - k_2) \right. \\
& \left. + IB(n_1 - k_1, n_2 + k_2) + IB(n_1 + k_1, n_2 + k_2) \right]. \tag{4.20}
\end{aligned}$$

This last expression for the convolution sum involves less number of multiplications and it is used in the program.

After the process of filtering, the resulting array replaces the former array IB. A process of scaling in which the gray levels are made to extend over the full range of the gray scale (in case they occupy only a portion of the whole allowable range) is then performed on this new array in order to produce a new scaled array IB. Any one of four different scaling rules (i.e. linear, square-root, logarithmic or absorption) that determine the set of break points in their own unique way are employed for this purpose. The scaling subroutine is directly adopted from the work of J.L Blankenship [4].

Finally the processed image is displayed on the screen by calling the "ImageOnScreen" subroutine.

4.2.1 Flow Chart For The Image Processing



$$\begin{aligned}
& + \sum_{k_2=1}^R h(0, k_2) \left[IB(n_1, n_2 - k_2) + IB(n_1, n_2 + k_2) \right] \\
& + \sum_{k_1=1}^R \sum_{k_2=1}^R h(k_1, k_2) \left[IB(n_1 - k_1, n_2 - k_2) + IB(n_1 + k_1, n_2 - k_2) \right. \\
& \left. + IB(n_1 - k_1, n_2 + k_2) + IB(n_1 + k_1, n_2 + k_2) \right]. \tag{4.20}
\end{aligned}$$

This last expression for the convolution sum involves less number of multiplications and it is used in the program.

After the process of filtering, the resulting array replaces the former array IB. A process of scaling in which the gray levels are made to extend over the full range of the gray scale (in case they occupy only a portion of the whole allowable range) is then performed on this new array in order to produce a new scaled array IB. Any one of four different scaling rules (i.e. linear, square-root, logarithmic or absorption) that determine the set of break points in their own unique way are employed for this purpose. The scaling subroutine is directly adopted from the work of J.L Blankenship [4].

Finally the processed image is displayed on the screen by calling the "ImageOnScreen" subroutine.

4.3 PROBLEMS ENCOUNTERED

A closed form expression for the ideal impulse response of a circularly symmetric lowpass filter is given by [20]

$$i(m, n) = \frac{f_c J_1(2\pi f_c \sqrt{m^2 + n^2})}{\sqrt{m^2 + n^2}}, \quad (4.21)$$

where f_c = the normalized cutoff frequency,

$J_1(x)$ = the first order Bessel function of the first kind
and is given by [40]

$$J_1(x) = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m+1}}{2^{2m-1} m! \Gamma(m+2)} = \sum_{m=-\infty}^{\infty} \frac{(-1)^m (x/2)^{2m+1}}{m! (m+1)!}. \quad (4.22)$$

Initially, the above formula was used to compute the ideal impulse response of a lowpass filter and no problem was encountered. But then the term for which the Bessel function was being computed, i.e. $x = 2\pi f_c \sqrt{m^2 + n^2}$, was small. When, at a later stage, the formula was used to compute $i(m, n)$ for filters of higher order and thus larger x values, problems started to arise. The computed data had no particular pattern. When they were plotted on 3-D graphs, the whole screen was filled with something reminiscent of noise.

As it turned out, the problem was the limit on the number of significant digits and the range of values the computer can

handle. The limit was seriously felt in the Bessel function computation.

The Bessel function was first computed by defining the functions that calculate the power, i.e. $(x/2)^{2m+1}$, and the factorial, i.e. $m!(m+1)!$, as real and then as extended data types. Real data types can take values in the range 2.9^{-39} to 1.7^{38} and have 11 to 12 significant digits while extended data types can take values in the range 1.9^{-4951} to 1.1^{4932} and have 19 to 20 number of significant digits [41].

The Bessel function computed from the two ways was compared with standard Bessel function tables [43]. The result was that defining the said functions as extended data types produced no significant deviations throughout the range of definition of Bessel function in the standard tables, i.e. 0 to 31.49. On the other hand, defining the functions as real data types gave rise to significant deviations for $x > 15$ and overflow errors for $x > 22$.

The deviation could be attributed to the limit on the number of significant digits while the overflow error is due to the limit on the range of numbers that can be handled in the real data types. In fact, for $x > 32$, even if we could not get a standard table to compare with, the haphazard manner in which the calculated Bessel function varies indicates that even the extended data types do not provide enough number of significant

digits to be useful in computing the ideal impulse response from the closed form expression.

It was after this problem was encountered then that another method, i.e. the sampling of the ideal frequency response and then performing the IDFT operation on it was considered and ultimately used to calculate the ideal impulse response.

Having designed the lowpass filter, the next step was to extend it to other types of filters. The frequency transformation technique was considered. But this method was to be dropped for the following reasons :

- 1) It is not possible to obtain circularly symmetric highpass, bandpass or bandstop filters from circularly symmetric lowpass filters [3,16].
- 2) If a frequency transformation is applied to a nonrecursive prototype, a recursive filter design will generally result and the advantages of linear phase characteristics will be lost [32].

In fact, the method of sampling the ideal frequency response and performing IDFT operations on it proved to be equally applicable to the design of the lowpass and the other types of filters. Of course it has its own problems in the form of aliasing errors. These errors are reduced by making the extent of support of the IDFT much larger than the extent of R (the radius of the window

function).

But again this has its own problems - speed. By having the region of support of IDFT very large, we end up with having very large number of sampling points of the ideal frequency response to match the region of support of the IDFT and consequently enlarged number of computations.

An obvious solution to the problem was to use the fast Fourier transform (FFT) [2,37,38]. FFT is particularly efficient if the region of support of the IDFT ($N \times N$) is very large.

But as N grows larger, the number of complex words of storage (N^2) required soon becomes greater than the capacity of the primary memory of the microcomputer. This necessitates the use of secondary storage devices to store the data. The data are then accessed one section at a time, resulting in input-output difficulties that severely affect the efficiency of the algorithm [2]. These, coupled with the difficulty of writing the program to realize the 2-D FFT algorithm within the available time for the work, forced the author to abandon the use of the FFT to compute the IDFT and DFT. A problem was also encountered in the image processing application in converting the array of integers that represent the gray levels into pictorial representations. An attempt was made to translate a Fortran program written by J.L. Blankenship [4] in which the different gray levels were

represented by overprinting up to 5 characters on a line printer into an equivalent Turbo-Pascal program. This however proved to be not possible at least in the text mode. In the graphics mode, it was possible to do it on the screen although with somewhat not very distinct gray levels. However, it had the advantage that a variety of combinations of character size and fonts and, vertical and horizontal spacings of the characters could be produced and the one that gives the best look chosen.

Later, it was seen that C. Ohlsen and G. Stroker [41] had made use of routines that interface BIOS to the BGI to create different gray levels in the form of shades. Their work was adapted and some modifications made to it with very good results. 16 distinct gray levels were possible to produce with this new technique which gave rise to much better looking images than the ones produced by over striking characters.

5. RESULTS AND CONCLUSIONS

In this chapter, some results obtained by running the programme and the conclusions derived from them are presented. Recommendations for future improvement of the programme are also included.

5.1. RESULTS

The software developed was used to design some of the standard filters, i.e. lowpass, highpass, bandpass and bandstop filters. Also some images were filtered with lowpass filters and highpass filters. Some of the results obtained are reproduced here.

5.1.1. Design of Filters

The window function used to design a filter, the ideal impulse response and the actual impulse response (i.e. the ideal impulse response weighted by the window) of a representative lowpass filter designed with the software, are shown both in tabular (Tables 5.1-5.3) and graphical (Figs. 5.1-5.4) forms. As the frequency response is continuous, only its graphical representation is shown (Fig. 5.4).

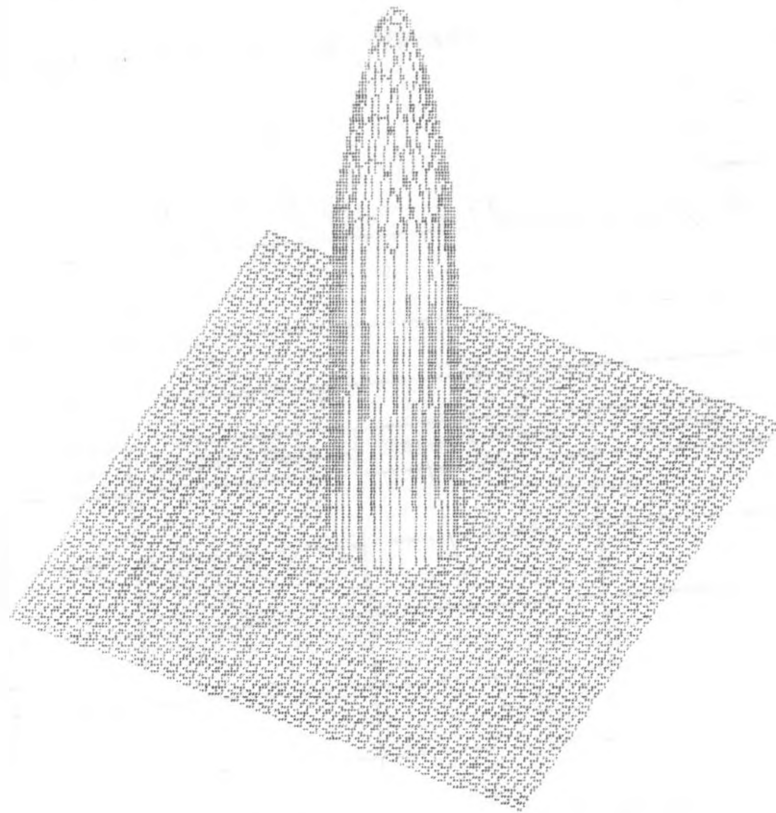
The filter characteristics that appear with the 3-D graphs of the window function and the impulse responses are those supplied by

the user of the programme but after being normalized (Figs.5.1, 5.2, 5.3). The ones that appear with the 3-D graph of the frequency response, however, are the actual characteristics of the designed filter, Fig.(5.4). The complete characteristics of the designed filter is shown in Table (5.4).

After the characteristics are made available, the user is asked whether a redesign is necessary, and if the answer is yes he/she needs to supply new values for the window order and the window parameter. In this example, the filter is not required to be redesigned.

$m \backslash n$	0	1	2	3	4	5	6	7
0	1.000	0.990	0.959	0.910	0.843	0.762	0.668	0.566
1	0.990	0.980	0.949	0.900	0.834	0.753	0.660	0.000
2	0.959	0.949	0.920	0.871	0.806	0.727	0.636	0.000
3	0.910	0.900	0.871	0.825	0.762	0.685	0.596	0.000
4	0.843	0.834	0.806	0.762	0.701	0.628	0.000	0.000
5	0.762	0.753	0.727	0.685	0.628	0.000	0.000	0.000
6	0.668	0.660	0.636	0.596	0.000	0.000	0.000	0.000
7	0.566	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 5.1 The Window Function.



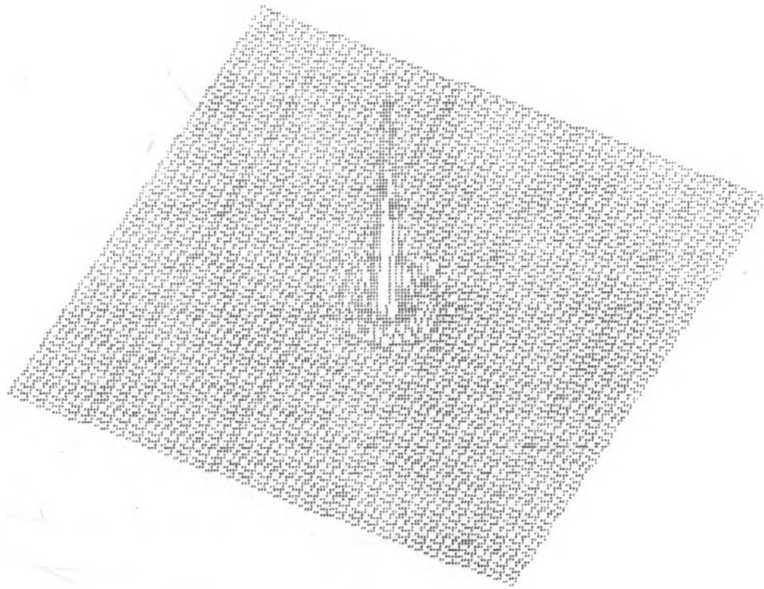
The Window Function.

Filter Type : Lowpass
Transition band width = 0.10
Cutoff Frequency = 0.30
The Ripple = 0.05
The Attenuation (in dB) = 26.02

Fig. 5.1 The Kaiser Window Function.

m \ n \ 	0	1	2	3	4	5	6	7
0	0.283	0.175	0.004-0.033	0.011	0.010-0.011	-0.001		
1	0.175	0.096-0.019-0.027	0.014	0.008-0.011	0.000			
2	0.004-0.019-0.037-0.005	0.018	0.001-0.011	0.003				
3	-0.033-0.027-0.005	0.016	0.010-0.009-0.006	0.000				
4	0.011	0.014	0.018	0.010-0.006-0.010	0.002	0.000		
5	0.010	0.008	0.001-0.009-0.010	0.000	0.000	0.000		
6	-0.011-0.011-0.011-0.006	0.002	0.000	0.000	0.000	0.000		
7	-0.001	0.000	0.003	0.000	0.000	0.000	0.000	0.000

Table 5.2 The Ideal Impulse Response
(Lowpass Filter)



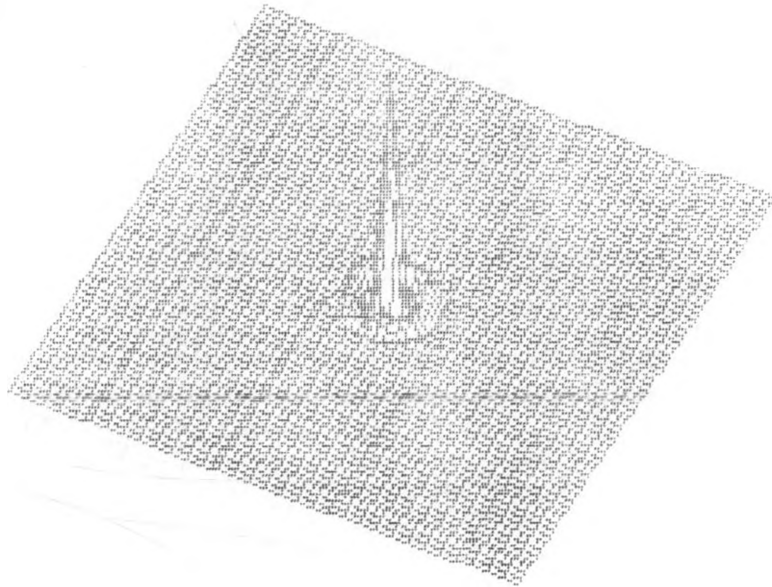
The Ideal Impulse Response

Filter Type : Lowpass
 Transition band width = 0.10
 Cutoff Frequency = 0.30
 The Ripple = 0.05
 The Attenuation (in dB) = 26.02

Fig. 5.2 The Ideal Impulse Response
(Lowpass Filter)

m \ n	0	1	2	3	4	5	6	7
0	0.283	0.173	0.004	-0.030	0.009	0.008	-0.007	-0.001
1	0.173	0.094	-0.018	-0.024	0.012	0.006	-0.007	0.000
2	0.004	-0.018	-0.034	-0.005	0.015	0.000	-0.007	0.000
3	-0.030	-0.024	-0.005	0.014	0.008	-0.006	-0.004	0.000
4	0.009	0.012	0.015	0.008	-0.004	-0.006	0.000	0.000
5	0.008	0.006	0.000	-0.006	-0.006	0.000	0.000	0.000
6	-0.007	-0.007	-0.007	-0.004	0.000	0.000	0.000	0.000
7	-0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 5.3 The Weighted Impulse Response
(Lowpass Filter)



The Impulse Response (Weighted)

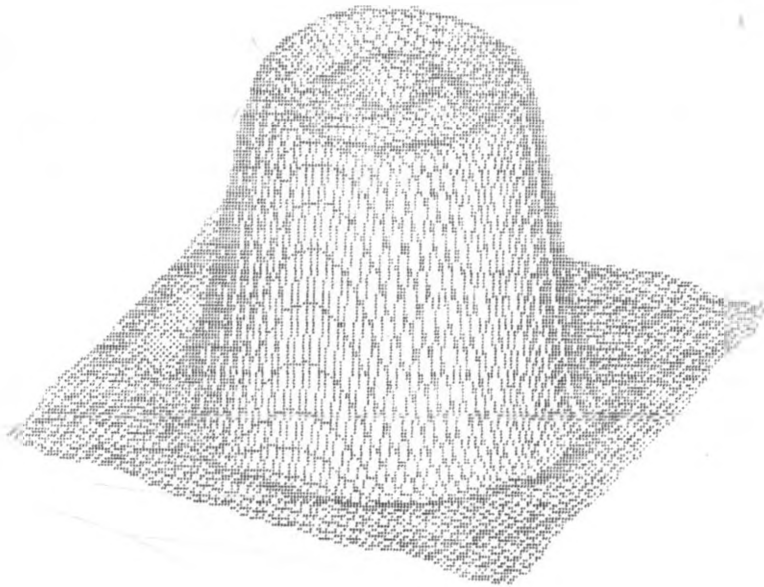
Filter Type : Lowpass
 Transition band width = 0.10
 Cutoff Frequency = 0.30
 The Ripple = 0.05
 The Attenuation (in dB) = 26.02

Ripple in Passband = 0.07
 Ripple in Stopband = 0.04
 Passband Limit = 0.26
 Stopband Limit = 0.35
 Transition Band Width = 0.09
 Cutoff Frequency = 0.28
 Attenuation(dB) = 25.27

The radius of the window used is : 7
 The circular window parameter is : 1.62

Now, do you want to redesign the filter(Y/N)? : n

Table 5.4 Characteristics of
Designed Lowpass Filter.



The Frequency Response

Filter Type : Lowpass
 Transition band width = 0.09
 CutOff Frequency = 0.28
 Ripple in the Passband = 0.07
 Ripple in the Stopband = 0.04
 Attenuation (Geo. mean) = 25.27 dB

Fig. 5.4 Frequency Response of
Designed Lowpass Filter

In the following, the impulse response (after the ideal impulse response is weighted by the window) of a highpass filter is shown both in tabular (Table 5.5) and graphical (Fig. 5.5) form. Then the frequency response is computed and the filter characteristics of the designed filter are given (Table 5.6).

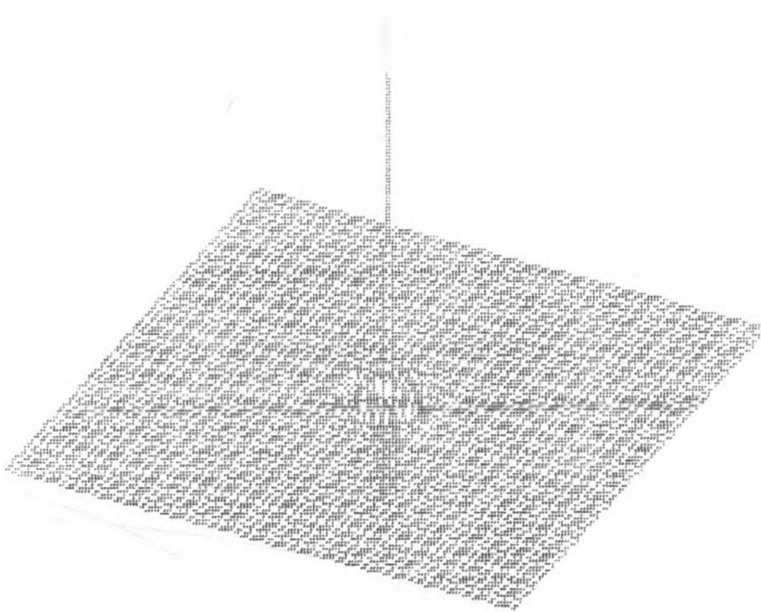
This time the filter is redesigned and new values of the window order and parameter (α) are supplied. The filter characteristics of the redesigned filter are again made available (Table 5.7). The impulse response of the redesigned filter is also shown (Table 5.8). If the user is still not satisfied, he/she can again redesign the filter. For a filter that does not require further redesign, the frequency response of the last designed filter is given in the form of 3-D graph (Fig. 5.6).

The Impulse response (weighted by window) :

m \ n	0	1	2	3	4	5	6	7
0	0.717	-0.173	-0.004	0.030	-0.009	-0.008	0.007	0.001
1	-0.173	0.094	0.018	0.024	-0.012	-0.006	0.007	-0.000
2	-0.004	0.018	0.034	0.005	-0.015	-0.000	0.007	-0.000
3	0.030	0.024	0.005	-0.014	-0.008	0.006	0.004	0.000
4	-0.009	-0.012	-0.015	-0.008	0.004	0.006	-0.000	0.000
5	-0.008	-0.006	-0.000	0.006	0.006	-0.000	0.000	0.000
6	0.007	0.007	0.007	0.004	-0.000	0.000	0.000	0.000
7	0.001	-0.000	-0.000	0.000	0.000	0.000	0.000	0.000

Do you wish to display it on 3-D graph(Y/N)? : y

Table 5.5 The Impulse Reponse
(Highpass Filter)



The Impulse Response (Weighted)

Filter Type : Highpass
 Transition band width = 0.10
 Cutoff Frequency = 0.30
 The Ripple = 0.05
 The Attenuation (in dB) = 26.02

Fig. 5.5 The Impulse Response
(Highpass Filter)

Ripple in Passband = 0.04
 Ripple in Stopband = 0.07
 Passband Limit = 0.35
 Stopband Limit = 0.26
 Transition Band Width = 0.09
 Cutoff Frequency = 0.32
 Attenuation(dB) = 25.27

The radius of the window used is : 7
 The circular window parameter is : 1.62

Now, do you want to redesign the filter(Y/N)? : y

Table 5.6 Characteristics of
 Designed Highpass Filter.

Ripple in Passband = 0.02
 Ripple in Stopband = 0.04
 Passband Limit = 0.36
 Stopband Limit = 0.25
 Transition Band Width = 0.11
 Cutoff Frequency = 0.32
 Attenuation(dB) = 32.16

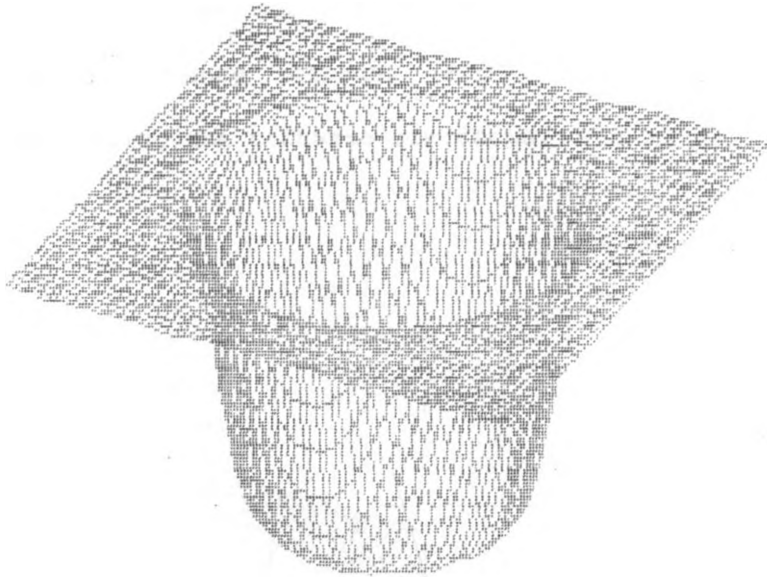
The radius of the window used is : 9
 The circular window parameter is : 3.00

Now, do you want to redesign the filter(Y/N)? : n

Table 5.7 Characteristics of
 Designed Highpass Filter.

m \ n	0	1	2	3	4	5	6	7	8	9
0	0.717	-0.172	-0.003	0.029	-0.008	-0.007	0.006	0.000	-0.002	0.000
1	-0.172	-0.093	0.018	0.023	-0.011	-0.005	0.006	-0.000	-0.002	0.000
2	-0.003	0.018	0.033	0.004	-0.013	-0.000	0.006	-0.001	-0.002	0.000
3	0.029	0.023	0.004	-0.012	-0.007	0.005	0.003	-0.002	-0.001	0.000
4	-0.008	-0.011	-0.013	-0.007	0.004	0.005	-0.001	-0.002	0.001	0.000
5	-0.007	-0.005	-0.000	0.005	0.005	-0.000	-0.003	-0.000	0.000	0.000
6	0.006	0.006	0.006	0.003	-0.001	-0.003	-0.001	0.000	0.000	0.000
7	0.000	-0.000	-0.001	-0.002	-0.002	-0.000	0.000	0.000	0.000	0.000
8	-0.002	-0.002	-0.002	-0.001	0.001	0.000	0.000	0.000	0.000	0.000
9	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 5.8 Impulse Response of
 Redesigned Highpass Filter.

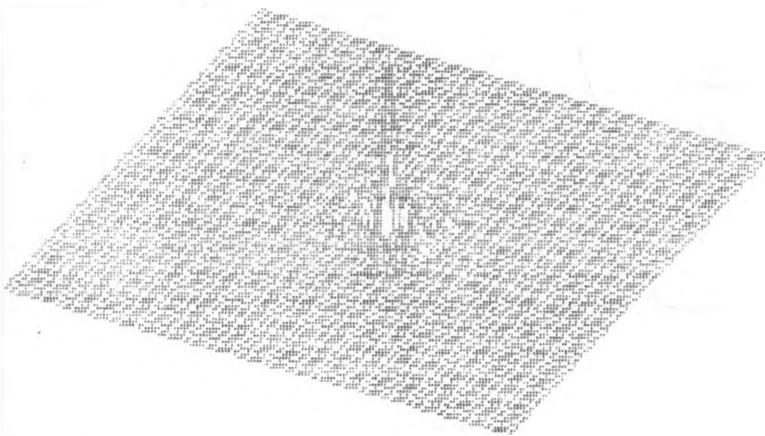


The Frequency Response

Filter Type : Highpass
Transition band width = 0.11
CutOff Frequency = 0.32
Ripple in the Passband = 0.02
Ripple in the Stopband = 0.04
Attenuation (Geo. mean) = 32.16 dB

Fig. 5.6 Frequency Response of Redesigned Highpass Filter.

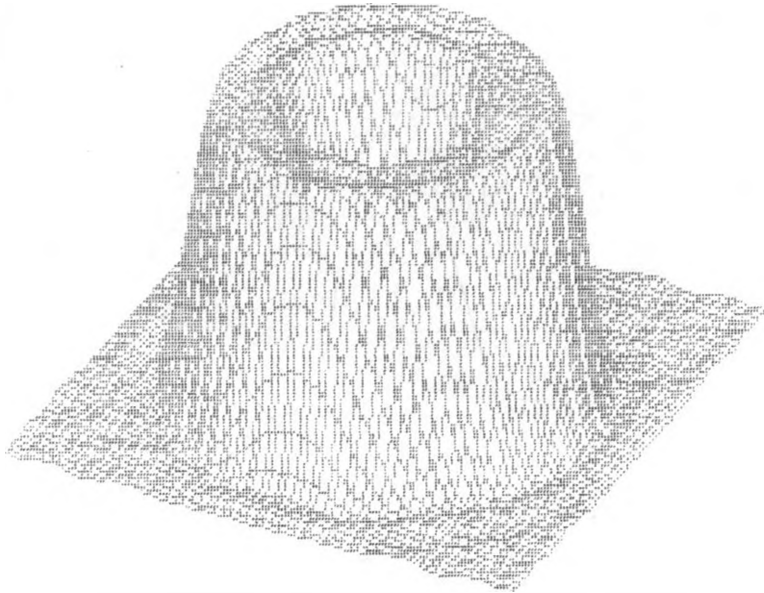
The impulse response (Fig. 5.7) and frequency response (Fig. 5.8) of a bandpass filter are shown next. Also the impulse response (Fig. 5.9) and frequency response (Fig. 5.10) of a bandstop filter designed using the software are shown.



The Impulse Response (Weighted)

Filter Type	= Bandpass
Lower Cutoff	= 0.15
Upper Cutoff	= 0.35
The Ripple	= 0.03
attenuation (dB)	= 30.46
Transition band width	= 0.10

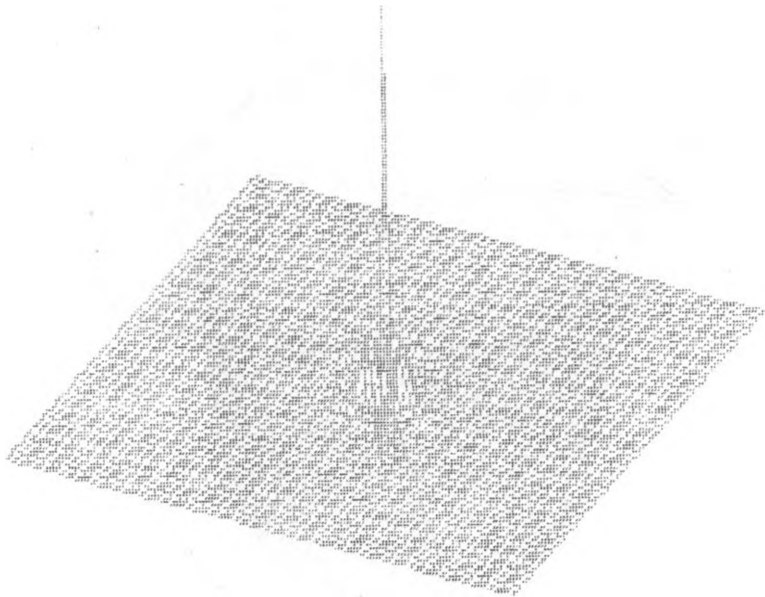
Fig. 5.7 Impulse Response of a Bandpass Filter.



The Frequency Response

Filter Type	=	Bandpass
Lower CutOff	=	0.16
UpperCutOff	=	0.34
Ripple in Lower Stopband	=	0.02
Ripple in Passband	=	0.03
Ripple in upper Stopband	=	0.03
Attenuation (Geo. mean)	=	32.0+ dB
Lower Transition Width	=	0.09

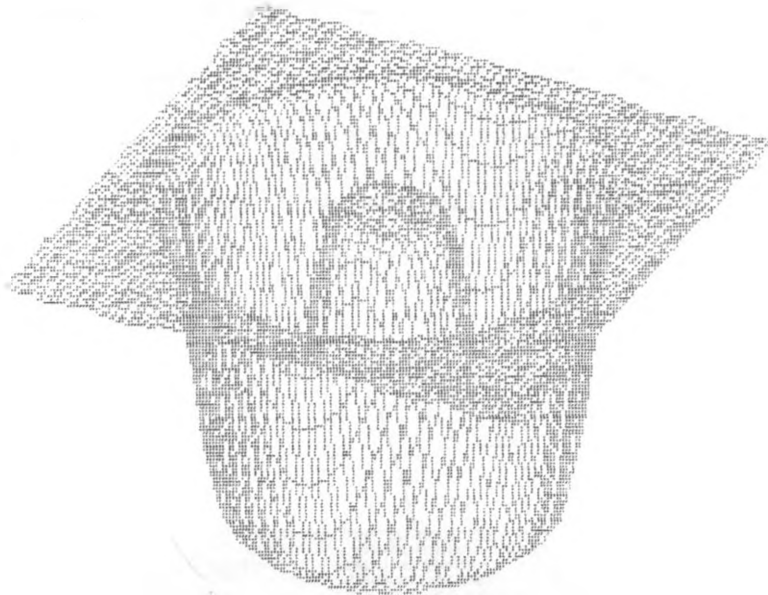
Fig. 5.8 Frequency Response of a Bandpass Filter.



The Impulse Response (Weighted)

Filter Type	=	Bandstop
Lower Cutoff	=	0.15
Upper Cutoff	=	0.35
The Ripple	=	0.03
Attenuation (dB)	=	30.46
Transition band width	=	0.10

Fig. 5.9 Impulse Response of a Bandstop Filter.



The Frequency Response

```

Filter Type           : Bandstop
Lower Cutoff         = 0.14
UpperCutOff          = 0.37
Ripple in lower Passband = 0.02
Ripple in Stopband   = 0.03
Ripple in Upper passband = 0.03
Attenuation (Geo. mean) = 32.04 dB
Lower Transition width = 0.09
  
```

Fig. 5.10 Frequency Response of a Bandstop Filter.

5.1.2 Image Processing

In the diskette containing the program are included codes for 9 different images. But here only one image will be considered.

Fig.(5.11) is an image for a boy before being processed and/or being corrupted with noise. Figs.(5.12)-(5.15) are results obtained after performing some manipulations on the original image.

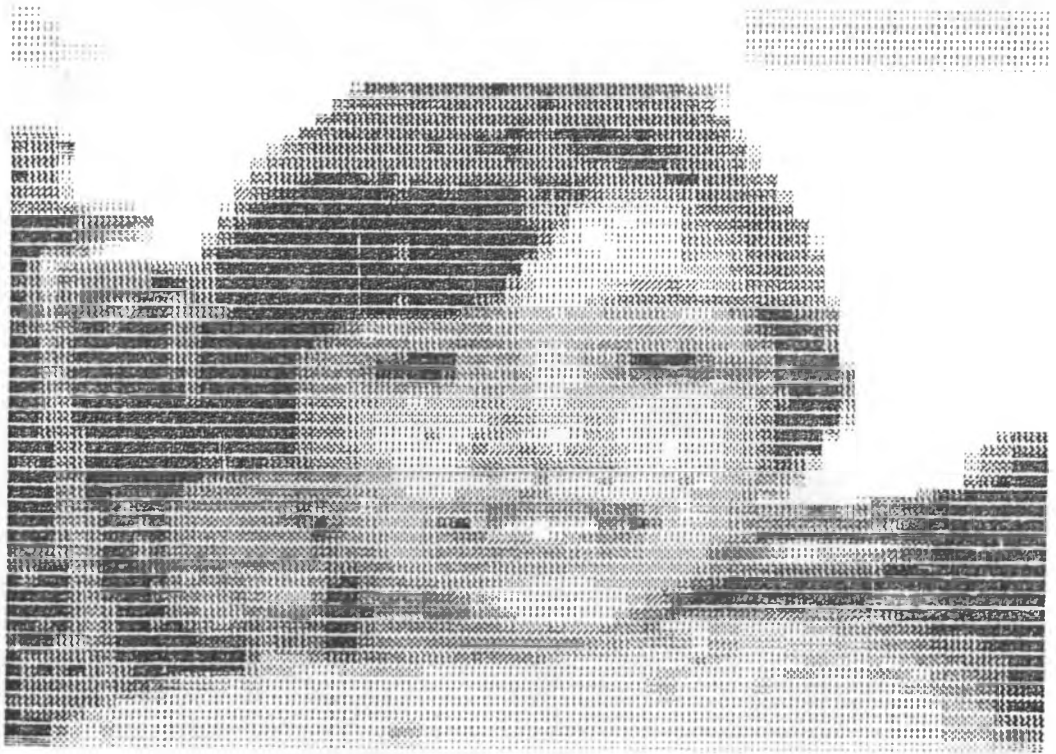


Fig. 5.11 The Original Image of a Boy.

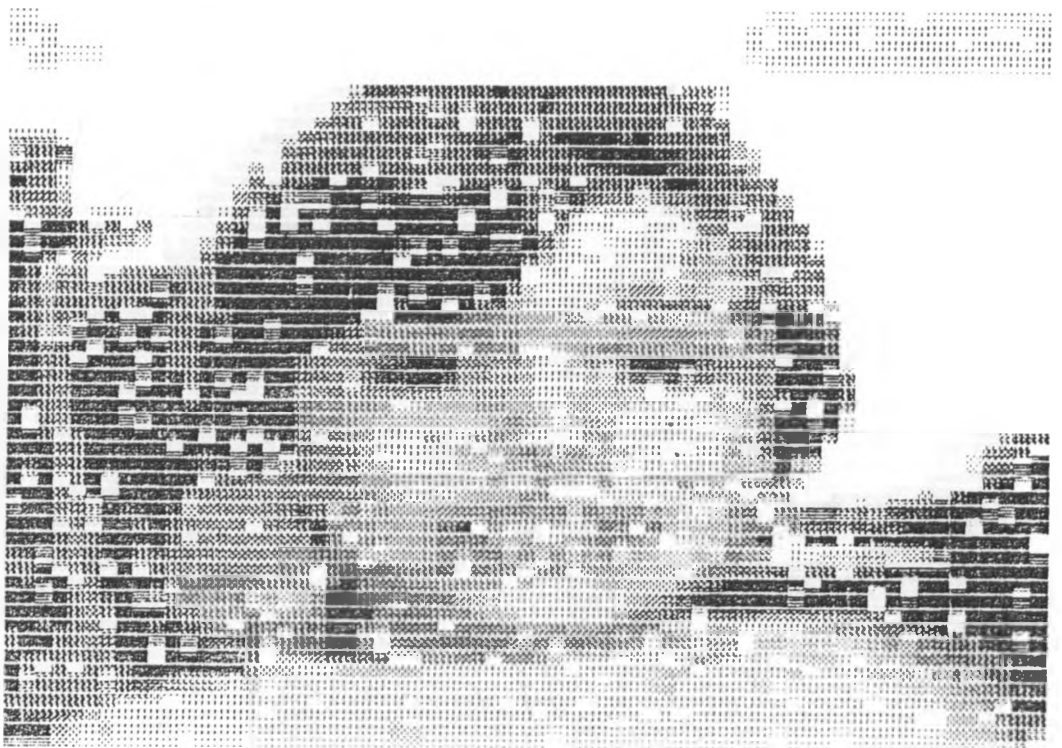


Fig. 5.12 Image of the Boy Corrupted With
Impulse Noise.



Fig. 5.13 After Filtering Noise Corrupted Image

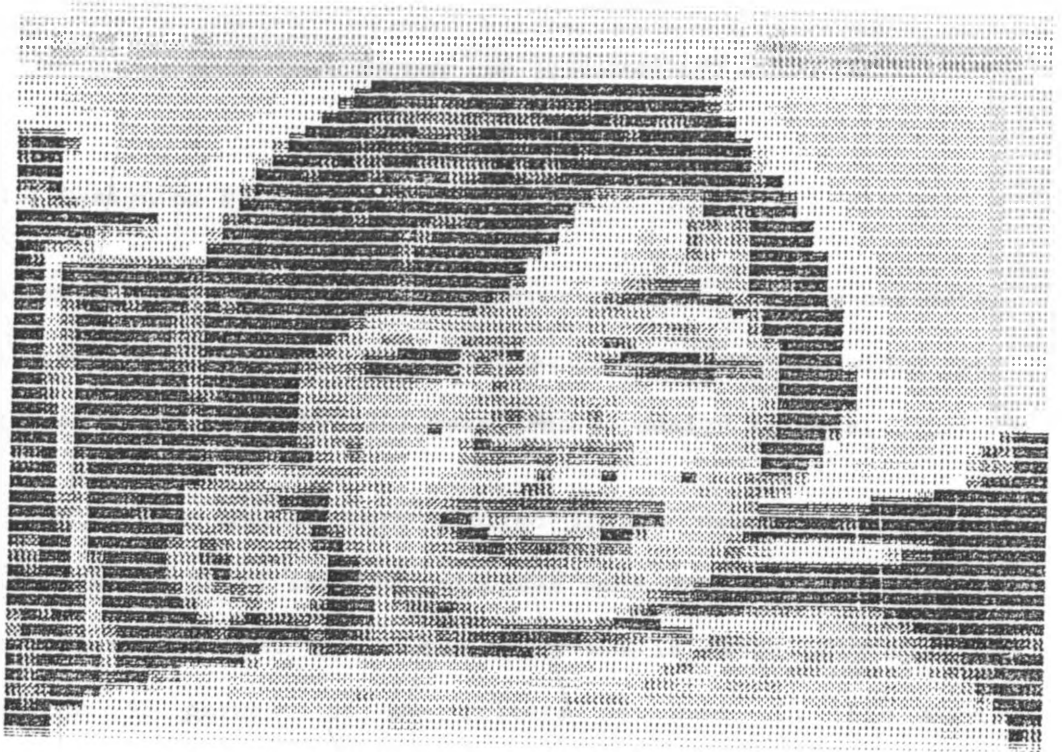
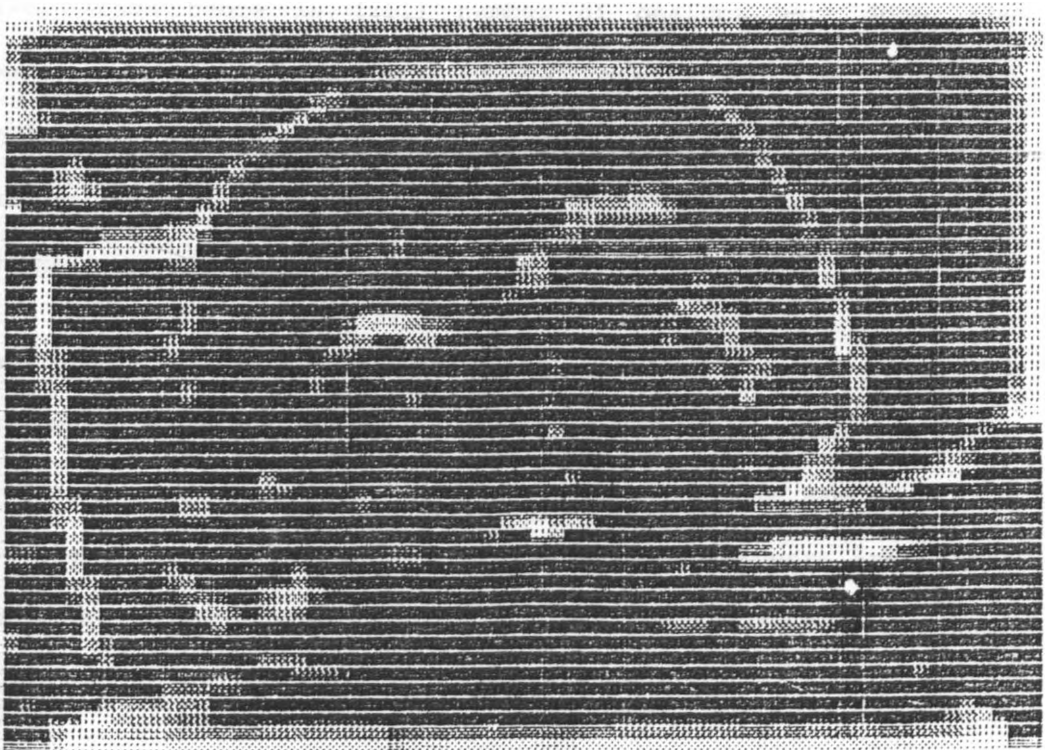


Fig. 5.14 After Filtering the Original Image
With a Highpass Filter.



5.2 CONCLUSIONS AND RECOMMENDATIONS

5.2.1 Design of Filters

The expressions for the minimum filter order and window parameter (α) required to give a particular transition bandwidth and attenuation as given by [20] were used to design filters.

In [20], these values were applied in the design of lowpass filters whose ideal impulse responses were obtained analytically with the result that the filter order was in error only by 2 or less for 95 percent of the filters designed. In this work, although the ideal impulse response of the lowpass filter was found not analytically but by performing the IDFT operation on the sampled values of the given ideal frequency response and much fewer (than those in [20]) number of designs were done, the results agree with the works of T.C. Speake and R.M. Mersereau [20], especially for filters of order greater than 10. However, when it comes to the design of other types of filters, the errors become larger.

Some of the discrepancies that might be observed could partly be attributed to the aliasing error introduced by sampling the ideal frequency response at only 129×129 points. If the sampling size and therefore the region of support of the IDFT were increased, the aliasing error would be reduced. But increasing the region of

support of IDFT would result in reduced speed.

As the FFT techniques were not used to perform the IDFT and the DFT operations, the design procedure is generally slow, and so the developed design program is used to design filters of order not greater 65.

This project work is the first of its kind in the department of Electrical and Electronic Engineering, University of Nairobi. It is therefore bound to be improved from time to time. The following are recommendations along this line:

i) Different filter types with varying filter specifications need to be designed and the characteristics of the designed filter recorded. From the recorded data, relations should be established that give better approximations for the window order and window parameter α than the one taken from [20] so that extra time wasted in redesigning the filter may be saved in the future.

ii) FFTs should be used to compute the IDFTs and the DFTs in the future. This will not only reduce the time required to design a filter of order of up to 65, but may also pave the way for designing filters of order greater than 65. Also, the region of support of the IDFT could be increased, without increasing the computation time, so that the aliasing error introduced by sampling of the ideal frequency response would be minimized.

iii) The software developed is used to design circularly symmetric lowpass, highpass, bandpass (with one passband and two stopband regions) and bandstop (with one stopband and two passband regions) filters. It has to be extended to include filters other than the ones it currently can handle.

5.2.2 Image Processing

The aim of this part of the program is to demonstrate how the different types of filters affect images when they are used to filter the images. This does not mean, however, that these are the best filters available. The fact that lowpass filters have smoothing effect on edges and other sharp transitions in the gray levels of an image and highpass filters have sharpening effect are however clearly demonstrated.

For the lowpass filter applications, the original image is corrupted with impulse noise, and after this corrupted image is filtered with a lowpass filter, the impulse noise and edges are seen to be smoothed - the required result. As for the highpass filter applications, the original image is filtered with the highpass filter and the result is either an image with more clearly defined edges and the rest part of the image less affected, or an image in which the edges are highlighted while the rest part of the image is altogether darkened, depending on

the filter specifications.

The images used for the work are of size 65x65. This is so due to the following :

i) The data available at the time when the program was first written were of that size only. Although, at a later stage, data for images of size 128x128 and 256x256 were made available, a section of the program has to be rewritten to enable the program accept and process these data. Time was just too short to do that.

ii) As the process of filtering used is the direct convolution method, processing of data of such big size would take too long time.

To get better results, both in terms of speed and image look, the following improvements in the program are recommended.

i) Images of size 128x128 or 256x256 should be used instead of the 65x65 currently in use.

ii) Convolution should be performed in the frequency domain. FFT techniques should be used to perform the IDFT and DFT operations.

iii) The gray levels should be increased from their current values of 16.

References

- [1] A. Peled and B. Liu, *Digital Signal Processing : Theory, Design and Implementation*. New York : John Wiley & sons, 1976.
- [2] D.E. Dudgeon and R.M. Mersereau, *Multidimensional Digital Signal Processing*. Englewood Cliffs, NJ : Prentice-Hall, 1984.
- [3] N.A. Pendergrass, S.K. Mitra and E.I. Jury, "Spectral Transformations for Two-Dimensional Digital Filters," *IEEE Trans. Circuits and Systems*, vol. CAS-23, pp. 26-35, Jan. 1976.
- [4] R.C. Gonzalez and P. Wintz, *Digital Image Processing*. Massachusetts : Addison-Wesley, 1987.
- [5] J.L. Lacoume, T.S. Durrani and R. Stora, ed., *Signal Processing, Vol. II*. Amsterdam : Elsevier Science Publishers, 1987.
- [6] A. Kundu, S.K. Mitra, and P.P. Vaidyanathan, "Application of Two-Dimensional Generalized Mean Filtering for Removal of Impulse Noises from Images," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-32, pp. 600-609, June 1984.
- [7] R. Bernstein, "Adaptive Nonlinear Filters for Simultaneous Removal of Different Kinds of Noise in Images," *IEEE Trans. Circuits And Systems*, Vol. CAS-34, pp. 1275-1291, Nov. 1987.
- [8] Y. Neuvo, P. Heinonen, and I. Dfée, "Linear Median Hybrid Edge Detectors," *IEEE Trans. Circuits And Systems*, Vol. CAS-34, pp. 1337-1343, Nov. 1987.
- [9] P. Maragos and R.W. Schafer, "Morphological Systems for Multi-Dimensional Signal Processing," *Proceedings IEEE*, Vol-78, pp. 690-710, Apr. 1990.
- [10] G.I. Vernazza, S.B. Serpico, and S.G. Dellepiane, "A Knowledge-Based Systems for Biomedical Image Processing And Recognition," *IEEE Trans. Circuits And Systems*, Vol. CAS-34, pp. 1399-1416, Nov. 1987.

- [11] N.K. Bose, "Multi-Dimensional Digital Signal Processing : Problems, Progress, and Future Scopes," *Proceedings IEEE*, Vol-78, pp. 590-597, Apr. 1990.
- [12] M.S. Betran, "Approximation of Digital Filters in One and Two Dimensions," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-23, pp. 438-443, Oct. 1975.
- [13] S.A.H. Aly and M.M. Fahmy, "Spatial Domain Design of Two-Dimensional Recursive Digital Filters," *IEEE Trans. Circuits And Systems*, Vol. Cas-27, pp. 892-901, Oct. 1980.
- [14] T. Hinamoto and S. Maekawa, "Spatial Domain Design of a Class of Two-Dimensional Recursive Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-32, pp. 153-162, Feb. 1984.
- [15] J.F. Abramatic, F. Germain, and E. Rosencher, "Design of Two-Dimensional Separable Denominator Recursive Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-27, pp. 445-453, Oct. 1979.
- [16] S. Chakrabarti and S.K. Mitra, "Design of Two-Dimensional Digital Filters via Spectral Transformations," *Proc. IEEE*, vol. 65, pp. 905-914, June 1977.
- [17] J.M. Costa and A.N. Venetsanopoulos, "Design of Circularly Symetric Two-Dimensional Recursive Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-22, pp. 432-443, Dec. 1974.
- [18] S. H. Mneney, A.N. Venetsanopoulos and J.M.Costa, " The Effects of Quantization errors on rotated filters," *IEEE Trans. Circuits and Systems*, vol. CAS-28, pp. 995-1003, Oct.1981.
- [19] R.M. Mersereau, W.F.G. Mecklenbraüker and T.F. Quatieri, JR. "McClellan Transformations for Two-Dimensional Digital Filtering : I - design," *IEEE Trans. Circuits and Systems*, vol. CAS-23, pp. 405-414, July 1976.
- [20] T.C. Speake and R.M. Mersereau, "A Note on the Use of Windows for Two-Dimensional FIR Filter Design," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 125-127, Feb. 1981.

- [21] T.S. Huang ,ed., *Topics in Applied Physics (Vol 42) : Two-Dimensional Digital Signal Processing I*. Berlin : Springer-Verlag, 1981.
- [22] Y.Kamp and J.P. Thiran, "Chebyshev Approximation for Two-Dimensional NonRecursive Digital Filters," *IEEE Trans. Circuits And Systems*, Vol. CAS-22, pp. 208-218, Mar. 1975.
- [23] J.H. Lodge and M.M. Fahmy, "An Efficient l_p Optimization Technique for the Design of Two-Dimensional Linear-Phase FIR Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-28, pp. 308-313, June 1980.
- [24] T.J. Terrell, *Introduction to Digital Filters*. Hong Kong : Macmillan, 1980.
- [25] B.D.O. Anderson and J.B. Moore, *Optimal Filtering*. Englewood Cliffs, NJ : Prentice-Hall, 1979.
- [26] A.V. Oppenheim and R.W. Schaffer, *Digital Signal Processing*. Englewood Cliffs, NJ : Prentice-Hall, 1975.
- [27] D.S. Humpherys, *The Analysis, Design, And Synthesis of Electrical Filters*. Englewood Cliffs, NJ : Prentice-Hall, 1970.
- [28] T.P. McClean and P. Schagen, *Electronic Imaging*. London : Academic Press Inc., 1979.
- [29] D.E. Johnson and J.C. Holburn, *Rapid Practical Design of Active Filters*. New York : John Wiley & Sons, 1975.
- [30] H.J. Blinichikoff and A.I. Zverev, *Filtering in the Time and Frequency Domains*. New york : John Wiley & Sons, 1976.
- [31] S.M. Bozic, *Digital and Kalman Filtering*. London : Edward Arnold, 1986.
- [32] M.H. Ackroyd, *Digital Filters*. London : Butterworth, 1973.
- [33] L.R. Rabiner and B. Golds, *Theory and Applications of Digital Signal Processing*. Englewood Cliffs, NJ : Prentice-Hall, 1975.
- [34] L.R. Rabiner and C.M. Rader, ed., *Digital Signal Processing*. New York : IEEE Press, 1972.

- [35] R.M. Mersereau and D.E. Dudgeon, "Two-Dimensional Digital Filtering," *Proc. IEEE*, vol. 63, pp. 610-623, Apr. 1975.
- [36] G.E. Rivard, "Direct Fast Fourier Transform of Bivariate Functions," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 250-252, June 1977.
- [37] G.L. Anderson, "A Stepwise Approach to Computing the Multidimensional Fast Fourier Transform of Large Arrays," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-28, pp. 280-284, June 1980.
- [38] R.M. Mersereau and T.C. Speake, "A Unified Treatment of Cooley-Tukey Algorithms for the Evaluation of the Multidimensional DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-29, pp. 1011-1017, Oct. 1981.
- [39] J.F. Kaiser and R.W. Schafer, "On the Use of Io-Sinh Window for Spectrum Analysis," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-28, pp. 105-107, Feb. 1980.
- [40] S.R. Wylie and L.C. Barrett, *Advanced Engineering Mathematics*. Singapore : McGraw-Hill Book Company, 1985.
- [41] C. Ohlsen and G. Stoker, *Turbo Pascal: Advanced Techniques*. Carmel : Que Corporation, 1989.
- [42] M. Yester, *Using Turbo Pascal*. Carmel : Que Corporation, 1989.
- [43] R.C. Wheast, ed. , *Handbook of Tables for Mathematics*. Cranwood Parkway, Cleveland, Ohio : The Chemical Rubber Company, 1970.

7. APPENDIX - PROGRAM LISTING

A turbo pascal programme listing is given below that :

1) designs standard (lowpass, highpass, bandpass, bandstop) circularly symmetric 2-dimensional digital filters using the Kaiser widow method.

2) displays, corrupts with impulse noise of 10% probability, and filters with the standard filters, digital images of size 65x65 which are stored in the diskette containing the programme.

The programme can be run on IBM PC or compatible computers with the following hardware and software specifications :

- i) CPU of the 80x86 family,
- ii) co-processor of the 80x87 family,
- iii) memory of size 640K or above,
- iv) PC - DOS version 2.0 or above,
- v) if the compiled programme cannot be obtained, a turbo pascal compiler version 4.0 or above with Borland graphics Interface (.BGI) and Character font (.CHR) files.

The aspect ratio used while in the Graphics mode is 1.00 , the default aspect ratio.

This program may be useful for the following :

1) Demonstration of design and applications of 2-D FIR digital filters on digital image processing for students taking courses on 2-D digital filters and/or image processing.

2) Anyone who wishes to design 2-D FIR digital filters using the Kaiser window and, study filters and see the effects of the different filter types on an image of size 65x65 - corrupted or uncorrupted with noise (impulse).

```
program KFIR_2D_Filter(input,output);
```

```
-----
This program designs a Two-Dimensional Low-Pass, High-Pass,
Band-Pass and Band-Stop Finite Impulse Response (FIR) Filters
using the Kaiser Window Method.
-----
```

```
$M 64000,0,655360)
$IFDEF CPU87 } { To use the 8087 Miroprocessor }
$N+ } { when necessary. }
$ELSE}
{$N-}
$ENDIF}
```

```
uses Dos,Crt, Graph,Printer,{$U A:GraphPRN}GraphPRN; { Links the necessary
} { units. }
```

```
const
epsilon=1.0E-8; { Condition To stop computations of }
{ Bessel Functions. }
MaxOrder = 65; { The Maximum order of the filter. }
RMax = 32;
RMax2 = 64;
```

```
type
Responses = array[-RMax..RMax,-RMax..RMax] of single;
FreqArray = array[0..RMax2,0..RMax2] of real;
RealPntr = ^FreqArray;
```

```
var
GraphDriver,GraphMode,
Height,BottomX,
BottomY,
MaxX,MaxY : integer;
PassbandRipple,
StopbandRipple,
Ripple,Dfpass1,
Dfpass2,Dfpass,
DfTran,Dfstop1,
Dfstop2,DfStop,
ATT,Alpha,NCutOff,
NCutOff1,NCutOff2,
PassBandRipple1,
PassBandRipple2,
StopBandRipple1,
StopBandRipple2,
Dftran1,Dftran2,
NCutOffNew,
NCutOffNew1,
NCutOffNew2 : single;
Width,Radius : byte;,
Nmax,Num,
ScaleCount,
CopyNumber,
MaxSize : byte;
```

```

Hard_Copy,Ans1,
OutOnGraph,
MyChoice,Choice,
FilterType,
Law,Image           : char;
w,i,IB              : Responses;
H                   : RealPntr;
DesignOver          : Boolean;

```

```

procedure LITTfont;      external;    {$L LITT.OBJ}
procedure EGAVGADriver; external;    {$L EGAVGA.OBJ}

```

```

procedure LoadTheFont(ProcedurePointer :pointer);
Begin
  if RegisterBGIfont(ProcedurePointer)<0 then begin
    writeln('Error registering font : ',GraphErrorMsg(GraphResult));
    Halt(1);
  end;
End;  { End of Procedure LoadTheFont. }

```

```

procedure LoadTheDriver(ProcedurePointer :pointer);
Begin
  if RegisterBGIDriver(ProcedurePointer)<0 then begin
    writeln('Error registering driver: ',GraphErrorMsg(GraphResult));
    Halt(1);
  end;
End;  { End of Procedure Load The Driver }

```

```

procedure FrequencyResponseOut;forward;
procedure AllOut;forward;
procedure OutFrequencyOnGraph;forward;

```

```

procedure  InputFilterParameters;
  {-----}
  { This procedure enables the user to input the type of the }
  { filter and the filter specifications of the filter one }
  { wants to design by way of making prompts to which appro- }
  { priate responses are given by the user. }
  {-----}

```

```

var
  SamplingFrequency,
  CurOffFrequency,
  TransitionWidth,
  CurOffFrequency1,
  CurOffFrequency2           : single;

```

```

Begin

```

```

  writeln;
  writeln('Choose the Type of Filter by Printing the Number ');
  writeln('Corresponding to Your Choice...');

```

```

writeln;
writeln('      1. Lowpass Filter. ');
writeln('      2. Highpass Filter. ');
writeln('      3. Bandpass Filter. ');
writeln('      4. Bandstop Filter. ');
writeln;
repeat
  write('Filter Type                : ');
  readln(FilterType);
  writeln;
  Case FilterType of
    '1' : begin      { Lowpass Filters }
      write('Sampling frequency in kHz      : ');
      readln(SamplingFrequency);
      writeln;
      writeln('Now Input Cut Off Frequency ');
      writeln('and Transition Widths... ');
      writeln;
      repeat
        write('Cut Off Frequency in kHz      : ');
        readln(CutOffFrequency);
        write('Transition Width in kHz      : ');
        readln(TransitionWidth);
        NCutOff:=CutOffFrequency/SamplingFrequency;
        Dftran:=TransitionWidth/SamplingFrequency;
        Dfpass:=NCutOff-0.5*Dftran;
        DfStop:=DfTran+DfPass;
        writeln;
        if(Dfstop > 0.5) then begin
          writeln(Chr(7));
          writeln('Too large Cut Off Frequency ');
          writeln(' TRY SMALLER values. ');
          writeln;
          end
        else if (Dfpass < 0) then begin
          writeln(Chr(7));
          writeln('Too small Cutoff Frequency. TRY larger values. ');
          writeln;
          end;
        until ((DfStop <= 0.5) and (Dfpass >= 0));
      end; { end of Case Lowpass Filter Type. }
    '2' : begin      { Highpass Filters }
      write('Sampling Frequency in kHz      : ');
      readln(SamplingFrequency);
      writeln;
      writeln('Now Input Cutoff Frequency and ');
      writeln('Transition width... ');
      writeln;
      repeat

```

```

write('CutOff Frequency in kHz      : ');
readln(CutOffFrequency);
write('Transition Width in kHz      : ');
readln(TransitionWidth);
NCutOff:=CutOffFrequency/SamplingFrequency;
Dftran:=TransitionWidth/SamplingFrequency;
Dfpass:=NCutoff+0.5*Dftran;
Dfstop:=Dfpass-Dftran;
if (Dfpass > 0.5) then begin
    writeln(Chr(7));
    writeln('Too large CutOff Frequency. TRY SMALLER value.');
```

end

```

else if (Dfstop < 0) then begin
    write(Chr(7));
    writeln('Too Small CutOff Frequency. TRY LARGER values.');
```

end;

```

until ((Dfstop >=0) and (Dfpass <= 0.5));
end; { end of highpass filter type. }

'3' : begin          { Bandpass Filters }
write('Sampling Frequency in kHz    : ');
readln(SamplingFrequency);
writeln;
writeln('Now Input the CutOff Frequencies and ');
writeln('the      Transition Width...');
writeln;
repeat
write('Lower CutOff Frequency in kHz : ');
readln(CutOffFrequency1);
write('Upper CutOff Frequency in kHz : ');
readln(CutOffFrequency2);
write('Transition Width in kHz      : ');
readln(TransitionWidth);
NCutOff1:=CutOffFrequency1/SamplingFrequency;
NCutOff2:=CutOffFrequency2/SamplingFrequency;
Dftran:=TransitionWidth/SamplingFrequency;
DfPass1:=NCutOff1+0.5*Dftran;
DfPass2:=NCutOff2-0.5*Dftran;
Dfstop1:=Dfpass1-Dftran;
Dfstop2:=Dfpass2+Dftran;
if (Dfpass1 >= Dfpass2) then begin
    writeln(Chr(7));
    writeln('Too Large Lower Cutoff or Too Small');
```

writeln('Upper Cutoff.TRY again !');

end

```

else if (Dfstop2 > 0.5) then begin
    writeln(Chr(7));
    writeln('Too large Upper CutOff or Transnsion');
```



```

        writeln('Band or both. TRY SMALLER values !');
        writeln;
        end
    else if (Dfstop1 < 0) then begin
        writeln(Chr(7));
        writeln('Too small Lower CutOff or too Large');
        writeln('Transition band or both. TRY again !');
        writeln;
    end;
    until ((Dfpass1 < Dfpass2) and (Dfstop2 <= 0.5)
           and (Dfstop1 >= 0));
end; { end of Bandpass filter type. }

'4' : begin { Bandstop Filters }
    write('Sampling Frequency in kHz      : ');
    readln(SamplingFrequency);
    writeln;
    writeln('Now Input the Cutoff Frequencies and ');
    writeln('the Transition Width...');
    writeln;
    repeat
        write('Lower Cutoff Frequency in kHz : ');
        readln(CutOffFrequency1);
        write('Upper Cutoff Frequency in kHz : ');
        readln(CutOffFrequency2);
        write('Transition Width in kHz      : ');
        readln(TransitionWidth);
        NCutOff1:=CutOffFrequency1/SamplingFrequency;
        NCutOff2:=CutOffFrequency2/SamplingFrequency;
        Dftran:=TransitionWidth/SamplingFrequency;
        Dfpass1:=NCutOff1-0.5*Dftran;
        Dfstop1:=Dfpass1+Dftran;
        Dfpass2:=NCutOff2+0.5*Dftran;
        Dfstop2:=Dfpass2-Dftran;
        if (Dfpass2 > 0.5) then begin
            writeln(Chr(7));
            writeln('Too large Second Passband Limit. ');
            writeln(' TRY SMALLER. ');
            writeln;
            end
        else if (Dfstop2 <= Dfstop1) then begin
            writeln(chr(7));
            writeln('Too narrow Bandstop or too large');
            writeln('Transition Width. TRY AGAIN. ');
            writeln;
        end;
    until ((Dfpass1 < Dfpass2) and ( Dfpass2 < 0.5)
           and ( Dfstop2 > Dfstop1));
end; { end of case Badstop filter type. }

```

```

else begin
  writeln(Chr(7));
  writeln('INVALID Choice of Filter Type. TRY AGAIN!!');
end;
writeln;
end { End of CASE. }
until FilterType IN ['1'..'4'];
writeln;
writeln('Now Input The Ripple...');
writeln;
repeat
  write('Ripple           : ');
  readln(Ripple);
  writeln;
  if (Ripple < 1.0E-3) then begin
    writeln(Chr(7));
    writeln('Too small ripple. TRY LARGER values. ');
  end
  else Ripple:=Ripple;
until (Ripple > 1.0E-3);
End; { End of Procedure Input Filter Parameters. }

function power(number,index:real):extended: { Raises number to }
var                                           { the power of index. }
  itslog : extended;                         { Extended range will }
Begin                                        { be required in the }
  itslog:=index*ln(number);                  { Bessel function }
  power:=exp(itslog);                         { computation. }
End; { End of Function Power. }power}

function factorial(n:integer):extended: { Gives out the factor- }
var                                           { ial of a given number. }
  fact   : extended;                         { Extended range will }
  number : integer;                          { be required in the }
Begin                                        { computation of Bessel }
  fact:=1;                                   { function. }
  for number:=1 to n do
    fact:=fact*number;
  factorial:=fact;
End; { End of Function factorial. }

function IO(x:real):extended:
{-----}
{ Function IO calculates the modified Bessel function of }
{ the first kind of order zero to be used later in the cal- }
{ culation of window coefficients. }
{-----}
var
  n           : byte;
  term,y,sum  : extended;

```

Begin

```
if x=0 then I0:=1
      else
```

```
begin
```

```
  x:=x/2; { To initialise the sum }
```

```
  n:=1; { and the first term }
```

```
  sum:=0;
```

```
  term:=1;
```

```
  while term>=epsilon do
```

```
  begin
```

```
    sum:=sum+term; { This part computes }
```

```
    y:=power(x,n)/factorial(n); { the actual series }
```

```
    term:=y*y;
```

```
    n:=n+1;
```

```
  end; { end of while. }
```

```
  I0:=sum;
```

```
end ; {end of else. }
```

```
End; { End of function I0-Modified Bessel function of order o }
```

```
procedure SpecificationsAndParameters;
```

```
{-----}
{ This Procedure Outputs the given specifications (after being }
{ normalized) and, calculates and outputs window parameters }
{ such as radius, widow order and the Alpha parameter to be }
{ used later in window calculation. }
{-----}
```

```
var
```

```
diff : real;
```

```
Begin
```

```
ATT:=-20*ln(ripple)/ln(10);
```

```
Radius:=round((ATT-7)/(2*13.68*Dftran));
```

```
Nmax:=2*Radius+1;
```

```
diff:=ATT-20.2;
```

```
if diff>0 then
```

```
  Alpha:=0.56*power(diff,0.4)+0.083*diff
```

```
else Alpha:=0;
```

```
writeln;
```

```
writeln('The Normalized Filter Specifications are : ');
```

```
writeln;
```

```
case FilterType of
```

```
'1'..'2' : writeln('Cutoff Frequency = ',NCutoff:1:2);
```

```
'3'..'4' : begin
```

```
  writeln('Lower Cutoff = ',NCutoff1:1:2);
```

```
  writeln('Upper Cutoff = ',NCutoff2:1:2);
```

```
end;
```

```
end;
```

```
writeln('The ripple = ',ripple:1:2);
```

```
writeln('Attenuation (dB) = ',ATT:1:2);
```

```

writeln;
writeln('The Window Parameters are:');
writeln;
writeln('Radius of Window           = ',Radius);
writeln('Window Order                 = ',Nmax);
writeln('The circular window parameter = ',Alpha:1:2);
End: { End of Procedure WindowParameters}

```

```

procedure Tabulate(Y:Responses);
{-----}
{ This procedure enables to present the window function }
{ and the impulse responses in tabular form.           }
{-----}

```

```

var
  m,n      : byte;
Begin
  writeln('m');
  write(' ');
  for m:=0 to radius do
    write(m:6);
  writeln;
  write(' ');
  for m:=0 to Radius do
    write('-----');
  writeln;
  for m:=0 to radius do begin
    write(m:2,' | ');
    for n:=0 to radius do
      write(Y[m,n]:6:3);
    writeln;
  end;
End: { End of Procedure Tabulate. }

```

```

procedure ThreeDGraph(Y : Responses);
{-----}
{ This procedure enables one to draw the three-dimensional }
{ graph of the impulse responses, windows or the frequency }
{ responses of the Two-Dimensional digital filter.         }
{-----}

```

```

Type
  VertAxis = array[1..MaxOrder,1..MaxOrder] of integer;
var
  k,m,n,MiIndex,MaIndex,i,j      : byte;
  X1,X2,Y1,Y2,Minimum,
  Maximum,m1,n1                   : integer;
  a,b,c,d                         : real;
  x,z                             : VertAxis;
  v                               : array[1..MaxOrder] of integer;
Begin
  BottomX:=MaxX div 2;           { (BottomX,BottomY) is the (x,y)- }

```

```
BottomY:=round(MaxY/1.4); { coordinate of the reference point.}
```

```
for m:=1 to MaxOrder do
for n:=1 to MaxOrder do begin
  m1:=m-(RMax+1);
  n1:=n-(RMax+1);
  a:=((MaxX/4)/(MaxOrder-1))*(n-1);
  b:=((MaxX/8)/(MaxOrder-1))*(m-1);
  c:=((MaxY/8)/(MaxOrder-1))*(n-1);
  d:=((MaxY/4)/(MaxOrder-1))*(m-1);
  x[m,n]:=BottomX-round(a)+round(b);
  z[m,n]:=BottomY-round(c)-round(d)-round(Height*Y[m1,n1]);
end;
```

{ The following part of the program connects points which share
the same coordinate on the 1st axis of the independent variab-
les. Rearside lines that overlap with front lines are invisible. }

```
for m:=1 to MaxOrder do begin
  X1:=x[m,1];X2:=x[m,2];
  Y1:=z[m,1];Y2:=z[m,2];
  MoveTo(X1,Y1);LineTo(X2,Y2);
```

```
for n:=3 to MaxOrder do begin
  X2:=x[m,n];Y2:=z[m,n];
if (m<=4) then LineTO(X2,Y2)
  else
begin
  j:=1;
  while ((m-2*j)>=1) and ((n-j)>=1) do { This part of the }
    begin { procedure checks if }
      v[j]:=z[m-2*j,n-j]; { there exists overlap }
      k:=j; j:=j+1; { and lines only when }
    end; { there is no overlap. }
  MiIndex:=1; MaIndex:=1;
  for i:=2 to k do
    if (v[MiIndex]>v[i]) then MiIndex:=i;
    Minimum:=v[MiIndex];
  for i:=2 to k do
    if (v[MaIndex]<v[i]) then MaIndex:=i;
    Maximum:=v[MaIndex];
    if (Y2>=Maximum) or (Y2<=Minimum) then LineTo(X2,Y2)
      else MoveTo(X2,Y2);
end; { end of else. }
end; { end of inner loop, i.e. n:=3 to MaxOrder. }
end; { end of outer loop, i.e. m:=1 to MaxOrder. }
```

{ The following part of the program connects points which
share the same coordinates on the 2nd axis of the inde-
pendent variables. }

```
for n:=1 to MaxOrder do
```

```

begin
    X1:=x[1,n];Y1:=z[1,n];
    X2:=x[2,n];Y2:=z[2,n];
    MoveTo(X1,Y1);LineTo(X2,Y2);
    for m:=3 to MaxOrder do
    begin
        X2:=x[m,n];Y2:=z[m,n];
        if (n<=2) or (m<=4) then LineTo(X2,Y2)
    else begin
        j:=1;
        while ((m-2*j)>=1) and ((n-j)>=1) do
        begin
            v[j]:=z[m-2*j,n-j];
            k:=j;j:=j+1;
        end;
        MiIndex:=1;MaIndex:=1;
        for i:=2 to k do
            if (v[MiIndex]>v[i]) then MiIndex:=i;
            Minimum:=v[MiIndex];

            for i:=2 to k do
                if (v[MaIndex]<v[i]) then MaIndex:=i;
                Maximum:=v[MaIndex];
        if (Y2>=Maximum) or (Y2<=Minimum) then LineTo(X2,Y2)
            else MoveTo(X2,Y2);

            end; { end of else. }
        end; { end of inner loop, i.e. m:=3 to MaxOrder. }
    end; { end of outer loop, i.e. n:=1 to MaxOrder. }
End; { End of Procedure ThreeDGraph. }

```

```

procedure CalculateWindowCoefficients;
{-----}
{ Computes the window function to be used later to modify }
{ the ideal impulse response. }
{-----}
var
    m,n      : integer;
    y,x      : real;
Begin
    for m:=0 to RMax do
    for n:=0 to RMax do begin
        y:=sqrt(sqr(m)+sqr(n));
        if y > Radius then w[m,n]:=0
        else begin
            x:=sqrt(1-(sqr(y)/sqr(Radius)));
            w[m,n]:=I0(Alpha*x)/I0(Alpha);
        end;
    end;
end; { end of do loop. }

```

```

for m:=-RMAX to -1 do      { Reflects 1st quadrant values }
for n:=0 to RMax do      { to 2nd quadrant. }
  w[m,n]:=w[-m,n];
for m:=-RMax to RMax do  { Reflects 1st and 2nd quadrant }
for n:=-RMax to -1 do   { values to 3rd and 4th quad- }
  w[m,n]:=w[m,-n];      { rant. }
if (MyChoice='2') and ((Choice='1') or (Choice='5'))
then begin
  RestoreCrtMode;
  writeln;
  writeln('The 2-D window coefficients are :');
  writeln;
  Tabulate(w);
  writeln;
  write('Do you wish to display it on 3-D graph(Y/N)? : ');
  readln(OutOnGraph);
end;
End; { End of Procedure CalculateWindowCoefficients. }

```

```

procedure CalculateIdealImpulseResponse;
{-----}
{ The ideal impulse response is computed by sampling the }
{ ideal frequency response and inverse frequency transform- }
{ ing these sampled frequency response values. }
{-----}
var
  w1,w2,m,n,k : integer;
  term,sum,r1,r2,R : real;
Begin
  MaxSize:=129;
  k:=2*RMax;
  for m:=0 to RMax do      { In the first quadrant of }
  for n:=0 to RMax do      { the (m,n)-plane. }
  if round(sqrt(sqrt(m)+sqrt(n))) <= Radius
  then begin
    sum:=0;
    for w1:=0 to k do      { In the first quadrant of }
    for w2:=0 to k do begin { the (w1,w2)-plane. }
      r1:=(m/MaxSize)*2*pi;
      r2:=(n/MaxSize)*2*pi;
      R:=sqrt(sqrt(w1)+sqrt(w2))/MaxSize;
    case FilterType of
      '1' : begin { Lowpass Filter }
        if ((w1=0) and (w2=0)) then term:=1
        else if ((w1=0) or (w2=0)) and (R <= NCutOff)
          then term:=2*cos(w1*r1)*cos(w2*r2)
        else if (R <= NCutOff)
          then term:=4*cos(w1*r1)*cos(w2*r2)
        else term:=0;
      end; { end of case lowpass filter type. }

```

```

'2' : begin      { Highpass Filter }
    if ((w1=0) or (w2=0)) and (R >= NCutOff)
        then term:=2*cos(w1*r1)*cos(w2*r2)
    else if (R > NCutOff)
        then term:=4*cos(w1*r1)*cos(w2*r2)
    else term:=0;
    end; { end of case highpass filter type. }
'3' : begin      { Bandpass Filter }
    if ((w1=0) or (w2=0)) and ((R >= NCutOff1)
                                and (R <= NCutOff2))
        then term:=2*cos(w1*r1)*cos(w2*r2)
    else if ((R >= NCutOff1) and (R <= NCutOff2))
        then term:=4*cos(w1*r1)*cos(w2*r2)
    else term:=0;
    end; { end of case bandpass filter type. }
'4' : begin      { Bandstop Filter }
    if ((w1=0) and (w2=0)) then term:=1
    else if ((w1=0) or (w2=0)) and
        ((R <= NCutOff1) or (R >= NCutOff2))
        then term:=2*cos(w1*r1)*cos(w2*r2)
    else if (R <= NCutOff1) or (R >= NcutOff2)
        then term:=4*cos(w1*r1)*cos(w2*r2)
    else term:=0;
    end; { end of case bandstop filter type. }
end; { end of CASE. }
sum:=sum+term/sqr(MaxSize);
end;
    i[m,n]:=sum;
end
else i[m,n]:=0;
    for m:=-RMax to -1 do
        for n:=0 to RMax do
            i[m,n]:=i[-m,n];
        for m:=-RMax to RMax do
            for n:=-RMax to -1 do
                i[m,n]:=i[m,-n];
            if (MyChoice='2') and ((Choice='2') or (Choice='5'))
                then begin
                    RestoreCrtMode;
                    writeln;
                    writeln('The ideal impulse response is : ');
                    writeln;
                    Tabulate(i);
                    writeln;
                    write('Do you wish to display it on 3-D graph(Y/N)? : ');
                    readln(OutOnGraph);
                end;
        end;
End; { End of Procedure CalculateIdealImpulseResponse. }

```



```

procedure CalculateImpulseResponse;
  {-----}
  { The weighted impulse response is obtained by multiplying }
  { the ideal impulse response with the window function,      }
  { point by point.                                           }
  {-----}
var
  m,n : integer;
Begin
  for m:=-RMax to RMax do
    for n:=-RMax to RMax do
      i[m,n]:=i[m,n]*w[m,n];
    if (MyChoice='2') and ((Choice='3') or (Choice='5'))
      then begin
        RestoreCrtMode;
        writeln;
        writeln('The Impulse response (weighted by window) : ');
        writeln;
        Tabulate(i);
        writeln;
        write('Do you wish to display it on 3-D graph(Y/N)? : ');
        readln(OutOnGraph);
      end;
  End; { End of Procedure CalculateImpulseResponse. }

```

```

procedure CalculateFourierTransform;
  {-----}
  { Computes the discrete Fourier transform.                    }
  {-----}
var
  w1,w2,m,n,k1,k2 : integer;
  term,sum,r1,r2 : real;
begin
  New(H);
  for w1:=0 to 2*RMax do
    for w2:=0 to 2*RMax do begin
      r1:=(w1/MaxSize)*2*pi;
      r2:=(w2/MaxSize)*2*pi;
      sum:=0;
      for m:=0 to Radius do
        for n:=0 to Radius do begin
          if (m=0) and (n=0) then term:=i[m,n]
          else if (m=0) or (n=0) then
            term:=2*i[m,n]*cos(m*r1)*cos(n*r2)
          else term:=4*i[m,n]*cos(m*r1)*cos(n*r2);
            sum:=sum+term;
          end;
          H^[w1,w2]:=sum;
        end;
      end;
    end;
  end;

```

End: { End of Procedure CalculateFourierTransform. }

Procedure InputAndDecode;

```
{-----}
{ Inputs the image code of Monalisa from a separate file }
{ and decodes it into integer values representing the inten- }
{ sity at each point of space of 65x65 points. }
{-----}
```

```
var
  IA          : array[-RMax..RMax,-RMax..RMax] of byte;
  StrChar     : Text;
  Str,Choice  : Char;
  i,j        : integer;
begin
  writeln('Type the Number corresponding to the image');
  writeln('you want to see. ');
  writeln;
  writeln('1. Image for Mona Lisa. ');
  writeln('2. Image for Lincoln. ');
  writeln('3. Image for a Boy. ');
  writeln('4. Image for The Statue of Liberty. ');
  writeln('5. Image for The Planet saturn. ');
  writeln('6. Geometrical figures. ');
  writeln('7. Some Characters. ');
  writeln('8. A Finger Print. ');
  writeln('9. Chromosomes. ');
  writeln;
  Repeat
    write('Now input your Choice      : ');
    readln(Choice);
    writeln;
    case Choice of
      '1' : Assign(StrChar,'A:MONALISA.COD');
      '2' : Assign(StrChar,'A:LINCOLN.COD');
      '3' : Assign(StrChar,'A:BOY.COD');
      '4' : Assign(StrChar,'A:STATUE.COD');
      '5' : Assign(StrChar,'A:SATURN.COD');
      '6' : Assign(StrChar,'A:GEOMETRI.COD');
      '7' : Assign(StrChar,'A:CHARACTE.COD');
      '8' : Assign(StrChar,'A:FINGER.COD');
      '9' : Assign(StrChar,'A:CHROMOSM.COD')
    else begin
      writeln(Chr(7));
      writeln('Invalid Choice. You have only to choose from ');
      writeln('integers in the range (1..9). ');
      writeln;
    end; { end of else. }
  end { end of CASE. }
until Choice in ['1'..'9'];
  Reset(StrChar);
  for i:=-RMax to RMax do begin
```

```

for j:=-RMax to RMax do begin
  read(StrChar,Str);
  if(Str >='0') and (Str <='9') then
    IA[i,j]:=Ord(Str)-Ord('0')
  else IA[i,j]:=Ord(Str)-Ord('A')+10;
end; { end of inner do loop, i.e. j:=-RMax to RMax. }
readln(StrChar);
end; { end of outer do loop. }
Close(StrChar);
for i:=-RMax to RMax do
  for j:=-RMax to RMax do
    IB[i,j]:=IA[i,j];
End; { End of Procedure InputAndDecode. }

```

```

procedure ScaleASRequired:

```

```

{-----}
{ Scales the decoded values according to different scaling }
{ methods so that the lowest value becomes 1 and the highest }
{ value becomes 32. }
{-----}

```

```

var

```

```

  i,j,k           : integer;
  FLEV,KLt,AA,EE,SS,T,
  Range,Imin,Imax : single;
  LEV             : array[1..32] of single;

```

```

Begin

```

```

  Imin:=31; Imax:=0;
  for i:=-RMax to RMax do
  for j:=-RMax to RMax do begin
    if Imin > IB[i,j] then Imin:=IB[i,j];
    if Imax < IB[i,j] then Imax:=IB[i,j];
  end;
  writeln;
  writeln('Choose the Scale from among the following by');
  writeln('typing the corresponding number. ');
  writeln;
  writeln('1. Linear Scale. ');
  writeln('2. Square-root Scale. ');
  writeln('3. Logarithmic Scale. ');
  writeln('4. "Absorption" Scale. ');
  WRITELN('5. Without Scaling. ');
  writeln;
  Repeat
    write(' Your Choice      ');
    readln(Law);
  If Law In ['1'..'4'] then begin
    Case Law of
      '1': begin
        Range :=(Imax-Imin)/32;
        for i:=1 to 32 do begin
          FLEV :=Imin+(i-1)*Range+0.5;

```

```

        LEV[i] :=Int(FLEV);
    end;
end; { end of case linear scale. }
'2': begin
    AA :=(sqrt(Imax)-sqrt(Imin))/32;
    for i:=1 to 32 do begin
        FLEV :=sqr(sqrt(Imin)+(i-1)*AA)+0.5;
        LEV[i] :=Int(FLEV);
    end;
end;{ end of case square-root scale. }
'3': begin
    EE :=(Imax-Imin)/ln(33);
    for i :=1 to 32 do begin
        FLEV :=Imin+EE*ln(i)+0.5;
        LEV[i] :=Int(FLEV);
    end;
end; { end of end of case logarithmic scale. }
'4': begin
    if Imin > 1 then T := Imin
        else T :=1;
    SS :=-ln(Imax/T)/32;
    for i:=1 to 32 do begin
        FLEV :=Imax*exp(SS*(32-i))+0.5;
        LEV[i] :=Int(FLEV);
    end;
end; { end of case absorption scale. }
end; { end of CASE. }
for i:=-RMax to RMax do
for j:=-RMax to RMax do begin
    KLT :=1;
    for k :=1 to 32 do
        if(IB[i,j] >= LEV[k]) then KLT :=k;
    IB[i,j] :=KLT;
    end;
end { end of case law in (1..4). }
else if (Law<>'5') then begin
    writeln(Chr(7));
    writeln('You have just typed ',Law,'. Please Type');
    writeln('One of of these only : 1,2,3,4 or 5.');
```

```

    writeln;
end
until Law in ['1'..'5'];
End; { End of procedure ScaleAsRequired. }
```

```

Procedure CorruptWithNoise;
```

```

var
```

```

    i,j      : integer;
```

```

Begin
```

```

    for i:=-RMax to RMax do
```

```

        for j:=-RMax to RMax do
```

```

    if Random<0.1 then
        IB[i,j]:=32
    End; { End of Procedure CorruptWithNoise. }

procedure ImageOnScreen; { Draws the image on the Screen. }
Const
    DAC = 14;
var
    RedVal,GreenVal,BlueVal    : Integer;
    VideoMode                   : Byte;

Function GetVideoMode : Byte;
{-----}
{ Places a variable directly into the video display data }
{ where the current video mode is stored.                }
{-----}

var
    Crt_Mode : Byte Absolute $40:49;
Begin
    GetVideoMode := Crt_Mode;
End;{ End of function GetVideoMode. }

Procedure GetRGBPalette(I :Word; var R,G,B : Integer);
{-----}
{ This function gets the red, green, and blue values.      }
{-----}

var
    Regs : Registers;
Begin
    Regs.AH := $10; { Select BIOS function 10H.    }
    Regs.AL := $15; { Select suroutine 15H.      }
    Regs.BX := I;   { Select the DAC to query.    }
    Intrl($10,Regs); { Call the BIOS Interrupt.   }
    R := Regs.DH;   { Read the red value into R.  }
    G := Regs.CH;   { Read the green value into G. }
    B := Regs.CL;   { Read the blue value into B. }
End: { End of function GetRGBPalette. }

procedure CreateGrayScale(FirstDAC,NumofDACs : Word);
{-----}
{ This procedure creates the gray scale. }
{-----}

var
    i : integer;
    Regs : Registers;
Begin
    for i:= 0 to 15 do
        begin

```

```

Regs.AH := $10; { Request function 10H.      }
Regs.AL := $10; { Request subfunction 10H.  }
Regs.BX := i;   { Register value to set.    }
Regs.CH := 4*i; { Green value to set.       }
Regs.CL := 4*i; { Blue value to set.        }
Regs.DH := 4*i; { Red value to set.         }
Intr($10,Regs); { Create the gray scale.    }
end;
End; { End of procedure CreateGrayScale. }

procedure ShowGrayScale;{ Displays the 16 gray scales. }
var
  m,n,I,X1,Y1,X2,Y2      : Word;
  XInc,YInc,X0,Y0        : Word;
  P                       : PaletteType;
Begin
  X0 := MaxX div 3;
  Y0 := 0;
  GetPalette(P);
  For I := 0 to 15 do
    SetPalette(I,I);
  XInc := (MaxX-2*X0) div 65;
  YInc := MaxY div 65;
  {SetFillStyle(SolidFill,15);
  Bar(0,0,MaxX,MaxY);}
  for m:=0 to 64 do
  for n:=0 to 64 do
  begin
    X1 := X0+XInc*n;
    Y1 := Y0+YInc*m;
    X2 := X1+XInc;
    Y2 := Y1+YInc;
    if (round(IB[-32+m,-32+n])<=30)
      then I := round(IB[-32+m,-32+n]/2)
      else I := 15;
    SetFillStyle(SolidFill,I);
    Bar(X1,Y1,X2,Y2);
  end;
End; { End of Procedure ShowGrayScale. }

Begin { Begin Procedure ImageOnScreen. }
  GetRGBPalette(DAC,RedVal,GreenVal,BlueVal);
  CreateGrayScale(0,16);
  ShowGrayScale;
  readln;
  CloseGraph;
End; { End of Procedure ImageOnScreen. }

procedure Convolve;
  {-----}
  { This procedure produces the convolution sum of the filter }

```

```

{ impulse response and the image code.
}
{-----}
var
  m1,m2,m3,m4,
  k1,k2,n1,n2,l1,l2          : integer;
  Term0,Term1,Term2,
  Term3,sum0,
  sum1,sum2                  : real;
begin
  for n1:=-RMax to RMax do
  for n2:=-RMax to RMax do
  begin
    sum0:=0; sum1:=0; sum2:=0;
    for k1:=1 to Radius do
    for k2:=1 to Radius do
    begin
      m1:=n1+k1;
      m2:=n1-k1;
      m3:=n2+k2;
      m4:=n2-k2;
      if (m1<=RMax) and (m3<=RMax)
        then Term0:=IB[m1,m3]
        else Term0:=0;
      if (m1<=RMax) and (m4>=-RMax)
        then Term1:=IB[m1,m4]
        else Term1:=0;
      if (m2>=-RMax) and (m4>=-RMax)
        then Term2:=IB[m2,m4]
        else Term2:=0;
      if (m2>=-RMax) and (m3<=RMax)
        then Term3:=IB[m2,m3]
        else Term3:=0;
      Sum0:=sum0+i[k1,k2]*(Term0+Term1+Term2+Term3);
    end;
    for k1:=1 to Radius do
    begin
      m1:=n1+k1;
      m2:=n1-k1;
      if (m1<=RMax) then Term0:=IB[m1,n2]
        else Term0:=0;
      if (m2>=-RMax) then Term1:=IB[m2,n2]
        else Term1:=0;
      Sum1:=sum1+i[k1,0]*(Term0+Term1);
    end;
    for k2:=1 to Radius do
    begin
      m3:=n2+k2;
      m4:=n2-k2;
      if (m3<=RMax) then Term0:=IB[n1,m3]
        else Term0:=0;
      if (m4>=-RMax) then Term1:=IB[n1,m4]

```

```

                else Term1:=0;
                sum2:=sum2+i[0,k2]*(Term0+Term1);
            end;
            w[n1,n2]:=i[0,0]*IB[n1,n2]+sum0+sum1+sum2;
        end;
        for n1:=-RMax to Rmax do
            for n2:=-RMax to RMax do
                IB[n1,n2]:=w[n1,n2];
            end;
        end; { End of Procedure Convolve. }

procedure WishHardCopy; { To inquire whether Hardcopy is required.}
Begin
    writeln;
    write('Do you wish to retain hardcopy? (Y/N) : ');
    readln(Hard_Copy);
    if (UpCase(Hard_Copy)='Y') then begin
        write('Number of copies? : ');
        readln(CopyNumber);
    end;
End; { End of Procedure WishHardcopy. }

procedure HardCopyIfNecessary; { Calls the unit Hardcopy.}
Begin
    if (UpCase(Hard_Copy)='Y') then begin
        for Num:=1 to CopyNumber do begin
            HardCopy(0);
            readln;
        end;
    end;
End; { End of Procedure HardcopyIfNecessary. }

procedure CalculateFilterParameters;
{-----}
{ This procedure calculates the filter parameters of the }
{ filter designed and enables the user to redesign the }
{ filter by providing new window order and parameter }
{ until such a time that the user is satisfied. }
{-----}

var
    FMax,FMin,A,
    wCmin,wCMax,
    FMin1,FMin2,
    wCMax1,wCMin1,
    wCMax2,wCMin2,
    FMax1,FMax2,R      : real;
    w1,w2              : integer;
Begin
    Case FilterType of
        '1'..'2' : begin
            FMax := 0.5;

```



```

FMin := 0.5;
wCMin:=0;
wCMax:=0.5;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
  if (H^[w1,w2]>FMax) then FMax:=H^[w1,w2];
  if (H^[w1,w2]<FMin) then FMin:=H^[w1,w2];
end; { end of do loop. }
PassBandRipple:=FMax-1;
StopBandRipple:=-FMin;
if (FilterType='1') then begin
  Dfpass:=0;
  Dfstop:=0.5;
  for w1:=0 to 2*Rmax do
  for w2:=0 to 2*Rmax do begin
    A:=sqrt(sqrt(w1)+sqrt(w2))/(4*RMax);
    if (A>Dfpass) and (H^[w1,w2]>(1-PassBandRipple))
      then Dfpass:=A;
    if (A<Dfstop) and (H^[w1,w2]<StopBandRipple)
      then Dfstop:=A;
    if (A>wCMin) and (H^[w1,w2]>0.707)
      then wCMin:=A;
    if (A<wCMax) and (H^[w1,w2]<0.707)
      then wCMax:=A;
  end; { end of do loop. }
  Dftran:=Dfstop-Dfpass;
end { end of "if..then begin". }
else begin
  Dfpass:=0.5;
  Dfstop:=0;
  for w1:=0 to 2*RMax do
  for w2:=0 to 2*RMax do begin
    A:=sqrt(sqrt(w1)+sqrt(w2))/(4*RMax);
    if (A<Dfpass) and (H^[w1,w2]>(1-PassBandRipple))
      then Dfpass:=A;
    if (A>Dfstop) and (H^[w1,w2]<StopBandRipple)
      then Dfstop:=A;
    if (A>wCMin) and (H^[w1,w2]<0.707)
      then wCmin:=A;
    if (A<wCMax) and (H^[w1,w2]>0.707)
      then wCMax:=A;
  end; { end of do loop. }
  Dftran:=Dfpass-Dfstop;
end; { end of else. }
NCutOffNew:=(wCMax+wCMin)/2;
ATT:=-10*ln(abs(PassBandRipple*StopBandRipple))/ln(10);
end; { end of case "1..2". }
'3' : begin
  FMax:=0.5;
  FMin1:=0.5;

```

```

FMin2:=0.5;
Dfpass1:=0.5;
Dfpass2:=0;
wCMax1:=0.5;
wCMin1:=0;
wCMax2:=0.5;
wCMin2:=0;
Dfstop1:=0;
Dfstop2:=0.5;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do
    if(H^[w1,w2]>FMax) then FMax:=H^[w1,w2];
PassBandRipple:=FMax-1;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
    A:=sqrt(sqr(w1)+sqr(w2))/(4*RMax);
    if (A<Dfpass1) and (H^[w1,w2]>(1-PassBandRipple))
        then Dfpass1:=A;
        if (A>Dfpass2) and (H^[w1,w2]>(1-PassBandRipple))
            then Dfpass2:=A;
        if (A>wCMin2) and (H^[w1,w2]>0.707)
            then wCmin2:=A;
        if (A<wCMax1) and (H^[w1,w2]>0.707)
            then wCMax1:=A;
end; { end of do loop. }
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
    A:=sqrt(sqr(w1)+sqr(w2))/(4*RMax);
    if (A<Dfpass1) and (H^[w1,w2]<FMin1)
        then FMin1:=H^[w1,w2];
        if (A>Dfpass2) and (H^[w1,w2]<FMin2)
            then FMin2:=H^[w1,w2];
end; { end of do loop. }
StopBandRipple1:=-FMin1;
StopBandRipple2:=-Fmin2;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
    A:=sqrt(sqr(w1)+sqr(w2))/(4*RMax);
    if (A<Dfpass1) then begin
        if (A>wCMin1) and (H^[w1,w2]<0.707)
            then wCMin1:=A;
        if (A>Dfstop1) and (H^[w1,w2]<StopBandRipple1)
            then Dfstop1:=A;
    end;
    if (A>Dfpass2) then begin
        if (A<wCMax2) and (H^[w1,w2]<0.707)
            then wCMax2:=A;
        if (A<Dfstop2) and (H^[w1,w2]<StopBandRipple2)
            then Dfstop2:=A;
    end;
end;
end; { end of do loop. }

```

```

NCutOffNew1:=(wCMax1+wCMin1)/2;
NCutOffNew2:=(wCMax2+wCMin2)/2;
Dftran1:=Dfpass1-Dfstop1;
Dftran2:=Dfstop2-Dfpass2;
Dftran:=sqrt(Dftran1*dftran2);
R:= (StopBandRipple1*PassbandRipple*StopBandRipple2);
ATT:=-20*ln(R)/(3*ln(10));
end; { end of case "3".}
'4' : begin
FMax1:=0.5;
FMax2:=0.5;
FMin:=0.5;
Dfstop1:=0.5;
Dfstop2:=0;
wCMax1:=0.5;
wCMax2:=0.5;
wCMin1:=0;
wCMin2:=0;
Dfpass1:=0;
Dfpass2:=0.5;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do
if (H^[w1,w2]<FMin) then FMin:=H^[w1,w2];
StopBandRipple:=-FMin;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
A:=sqrt(sqrt(w1)+sqrt(w2))/(4*RMax);
if (A>Dfstop2) and (H^[w1,w2]<StopBandRipple)
then Dfstop2:=A;
if (A<Dfstop1) and (H^[w1,w2]<StopBandRipple)
then Dfstop1:=A;
if (A<wCMax1) and (H^[w1,w2]<0.707)
then wCMax1:=A;
if (A>wCMin2) and (H^[w1,w2]<0.707)
then wCMin2:=A;
end; { end of do loop. }
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
A:=sqrt(sqrt(w1)+sqrt(w2))/(4*RMax);
if (A<Dfstop1) and (H^[w1,w2]>FMax1)
then FMax1:=H^[w1,w2];
if (A>Dfstop2) and (H^[w1,w2]>FMax2)
then FMax2:=H^[w1,w2];
end; { end of do loop. }
PassBandRipple1:=FMax1-1;
PassBandRipple2:=FMax2-1;
for w1:=0 to 2*RMax do
for w2:=0 to 2*RMax do begin
A:=sqrt(sqrt(w1)+sqrt(w2))/(4*RMax);
if (A<Dfstop1) then begin
if (A>wCMin1) and (H^[w1,w2]>0.707)

```

```

        then wCMin1:=A;
    if (A>Dfpass1) and (H|w1,w2|>(1-PassBandRipple1))
        then Dfpass1:=A;
    end;
    if (A>Dfstop2) then begin
        if (A<wCMax2) and (H|w1,w2|>0.707)
            then wCMax2:=A;
        if (A<Dfpass2) and (H|w1,w2|>(1-PassBandRipple2))
            then Dfpass2:=A;
        end;
    end; { end of do loop. }
    NCutOffNew1:=(wCMax1+wCMin1)/2;
    NCutOffNew2:=(wCMax2+wCMin2)/2;
    Dftran1:=Dfstop1-Dfpass1;
    Dftran2:=Dfpass2-Dfstop2;
    R:=abs(PassBandRipple1*PassBandRipple2*StopBandRipple);
    ATT:=-20*ln(R)/(3*ln(10));
end; { end of case "4". }
end; { end of CASE. }

```

RestoreCrtMode;

writeln('The Filter Designed is Characterised by the following : ');

writeln;

Case FilterType of

'1'..'2' : begin

```

    writeln('Ripple in Passband           = ',PassBandRipple:0:2);
    writeln('Ripple in Stopband           = ',StopBandRipple:0:2);
    writeln('Passband Limit                   = ',Dfpass:0:2);
    writeln('Stopband Limit                     = ',Dfstop:0:2);
    writeln('Transition Band Width              = ',Dftran:0:2);
    writeln('Cutoff Frequency                   = ',NCutOffNew:0:2);
    writeln('Attenuation(dB)                    = ',ATT:0:2);

```

end; { end of case "1..2". }

'3'..'4' : begin

```

    writeln('Lower Stopband Limit             = ',Dfstop1:0:2);
    writeln('Lower Passband Limit               = ',Dfpass1:0:2);
    writeln('Lower Transition Bandwidth          = ',Dftran1:0:2);
    writeln('Lower Cutoff                       = ',NCutOffNew1:0:2);
    writeln('Upper Passband Limit               = ',Dfpass2:0:2);
    writeln('Upper Stopband Limit               = ',Dfstop2:0:2);
    writeln('Upper Transition Bandwidth          = ',Dftran2:0:2);
    writeln('Upper Cutoff                       = ',NCutOffNew2:0:2);
    writeln('Attenuation in dB (Geo. mean)      = ',ATT:0:2);
    if (FilterType='3') then begin
        writeln('Ripple in lower Stopband          = ',StopBandRipple1:0:2);
        writeln('Ripple in Upper Stopband          = ',PassBandRipple:0:2);
        writeln('Ripple in Passband                 = ',StopBandRipple2:0:2);
    end

```

end

else begin

```

    writeln('Ripple in Lower Stopband          = ',PassBandRipple1:0:2);
    writeln('Ripple in Stopband                 = ',StopBandRipple:0:2);

```

```

        writeln('Ripple in Upper Passband           = ',PassBandRipple2:0:2);
    end;
    end: { end of case "3..4". }
end: { end of CASE. }
writeln;
writeln('The radius of the window used is      : ',radius);
writeln('The circular window parameter is      : ',Alpha:1:2);
writeln;
write('Now, do you want to redesign the filter(Y/N)? : ');
readln(Ans1);
writeln;
if (UpCase(Ans1)='Y') then begin
    writeln;
    write('Input a new value of radius (+ve integer) : ');
    readln(radius);
    write('Input a new value for Alpha              : ');
    readln(Alpha);
    if (Choice='4') then FrequencyResponseOut
        else Allout;
end
else begin
    for w1:=0 to RMax do
        for w2:=0 to RMax do
            w[w1,w2]:=H^[2*w1,2*w2];
        for w1:=-RMax to -1 do
            for w2:=0 to RMax do
                w[w1,w2]:=w[-w1,w2];
            for w1:=-RMax to RMax do
                for w2:=-RMax to -1 do
                    w[w1,w2]:=w[w1,-w2];
                OutFrequencyOnGraph;
            end;
        end;
    End: { End of Procedure CalculateFilterParameters. }

procedure ProcessImage;
    {-----}
    { This procedure controls the sequence of events that lead }
    { to the filtered image output. }
    {-----}

var
    Ans1,Ans2          : Char;
    i,j                : integer;
begin
    InputAndDecode;
    ScaleCount:=1;
    ScaleAsRequired;
    GraphDriver := VGA;
    GraphMode := VGAGHi;
    InitGraph(GraphDriver,GraphMode,'');
    MaxX:=GetMaxX;
    MaxY:=GetMaxY;

```

```

ImageOnScreen;
RestoreCrtMode;
write('Corrupt Image with noise? (Y/N) : ');
readln(Ans1);
if (UpCase(Ans1)='Y') then begin
  CorruptWithNoise;
  InitGraph(GraphDriver,GraphMode,'');
  ImageOnScreen;
  RestoreCrtMode;
end;
write('Do you wish to filter the image? (Y/N) : ');
readln(Ans2);
if (UpCase(Ans2)='Y') then begin
  InputFilterParameters;
  SpecificationsAndParameters;
  writeln;
  write('Filter being designed...');

  CalculateWindowCoefficients;
  CalculateIdealImpulseResponse;
  CalculateImpulseResponse;
  writeln;
  writeln(Chr(7));
  write('Filter is now designed. Image is being processed..');
  Convolve;
  writeln(Chr(7));
  for i:=-RMax to RMax do
  for j:=-RMax to RMax do begin
    if (IB[i,j] <=1) then IB[i,j]:=0;
    if (IB[i,j] >=31) then IB[i,j]:=31;
  end; { end of do loop. }
  InitGraph(GraphDriver,GraphMode,'');
  ImageOnScreen;
end;
RestoreCrtMode;
CloseGraph;
End; { End of ProcessImage. }

procedure MessageOut;
{-----}
{ Type of Filter and its Characteristics will be outputted along }
{ with the Three-Dimensional graph of the filter. }
{-----}
var
  x,y1,y2,y3,
  y4,y5,y6,y7,
  y8,y9,yinc      : integer;

function Str5(RealNo : single) : String;
var
  Tens,Ones,Tenths,Hundredths,IntNo      : integer;
begin

```

```

IntNo := Round(100*RealNo);
Tens := ((IntNo div 10) div 10) div 10;
Ones := ((IntNo div 10) div 10) mod 10;
Tenths :=(IntNo div 10) mod 10;
Hundredths :=IntNo mod 10;
if (Tens=0) and (Tenths=0) and (Hundredths=0) then
  Str5:=Chr(32)+Chr(48+Ones)+Chr(32)+Chr(32)+Chr(32)
else if (Tens=0) then
  Str5:=Chr(32)+Chr(48+Ones)+Chr(46)+Chr(48+Tenths)+
    Chr(48+Hundredths)
else if (Tenths=0) and (Hundredths=0) then
  Str5:=Chr(48+Tens)+Chr(48+Ones)+Chr(32)+Chr(32)+Chr(32)
else Str5:=Chr(48+Tens)+Chr(48+Ones)+Chr(46)+Chr(48+Tenths)+
    Chr(48+Hundredths)
end; { end of function Str5. }

```

Begin

```

x:=round(MaxX/3);
y1:=round(Max Y/1.24);
yinc:=round(Max Y/40);
y2:=y1+yinc;
y3:=y2+yinc;
y4:=y3+yinc;
y5:=y4+yinc;
y6:=y5+yinc;
y7:=y6+yinc;
y8:=y7+yinc;
y9:=y8+yinc;
SetTextStyle(2,0,4);
Case FilterType of

  '1': OutTextXY(x,y1,'Filter Type           : Lowpass ');
  '2': OutTextXY(x,y1,'Filter Type           : Highpass');
  '3': OutTextXY(x,y1,'Filter Type           : Bandpass');
  '4': OutTextXY(x,y1,'Filter Type           : Bandstop');
end; { end of CASE. }

```

Case FilterType of

```

'1'..'2' :
begin
  OutTextXY(x,y2,'Transition band width   = '+Str5(Dftran));
  if (DesignOver=False) then begin
    OutTextXY(x,y3,'Cutoff Frequency     = '+Str5(NCutoff));
    OutTextXY(x,y4,'The Ripple           = '+Str5(Ripple));
  end
else begin
  OutTextXY(x,y3,'CutOff Frequency       = '+Str5(NCutoffNew));
  OutTextXY(x,y4,'Ripple in the Passband = '+Str5(PassBandRipple));
  OutTextXY(x,y5,'Ripple in the Stopband = '+Str5(StopBandRipple));
end;

```

```

if (DesignOver=False) then
  OutTextXY(x,y5,'The Attenuation (in dB) = '+Str5(ATT))
  else OutTextXY(x,y6,'Attenuation (Geo. mean) = '+Str5(ATT)+' dB');
end; { end of case "1..2". }

'3'..'4' :
begin
  if (DesignOver=False) then begin
    OutTextXY(x,y2,'Lower Cutoff           = '+Str5(NCutOff1));
    OutTextXY(x,y3,'Upper Cutoff           = '+Str5(NCutOff2));
    OutTextXY(x,y4,'The Ripple              = '+Str5(Ripple));
    OutTextXY(x,y5,'Attenuation ( dB)      = '+Str5(ATT));
    OutTextXY(x,y6,'Transition band width = '+Str5(Dftran));
  end
  else begin
    OutTextXY(x,y2,'Lower Cutoff           = '+Str5(NCutOffNew1));
    OutTextXY(x,y3,'UpperCutOff           = '+Str5(NCutOffNew2));
    if (FilterType='3') then begin
      OutTextXY(x,y4,'Ripple in Lower Stopband = '+Str5(StopBandRipple1));
      OutTextXY(x,y5,'Ripple in Passband      = '+Str5(PassBandRipple));
      OutTextXY(x,y6,'Ripple in upper Stopband = '+Str5(StopBandRipple2));
    end
    else begin
      OutTextXY(x,y4,'Ripple in lower Passband = '+Str5(PassBandRipple1));
      OutTextXY(x,y5,'Ripple in Stopband      = '+Str5(StopBandRipple));
      OutTextXY(x,y6,'Ripple in Upper passband = '+Str5(PassbandRipple2));
    end;
    OutTextXY(x,y7,'Attenuation (Geo. mean) = '+Str5(ATT)+' dB');
    OutTextXY(x,y8,'Lower Transition width  = '+Str5(Dftran1));
    OutTextXY(x,y9,'Upper Transition width   = '+Str5(dftran2));
  end;
end; { end of case "3..4". }
end; { end of CASE. }
End; { End of Procedure MessageOut. }

```

```

procedure WindowOut;

```

```

{-----}
{ This procedure outputs the window functions in the form   }
{ of tables or 3-Dimensional Graph.                         }
{-----}

```

```

Begin

```

```

  SetGraphMode(GraphMode);
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateWindowCoefficients;
  DesignOver:=False;
  if (UpCase(OutOnGraph)='Y') then begin
    WishHardCopy;
    SetGraphMode(GraphMode);
    Height:=MaxY div 3;
    ThreeDGraph(w);
  end;
end;

```



```

    SetTextStyle(2,0,6);
    OutTextXY(round(MaxX/3.2),round(MaxY/1.32),' The Window Function');
    MessageOut;
    readln;
    HardCopyIfNecessary;
    RestoreCrtMode;
    CloseGraph;
  end;
End; { End of Procedure WindowOut. }

```

```

Procedure IdealImpulseOut:
  {-----}
  { This procedure outputs the ideal impulse response in the }
  { form of tables or 3-Dimensional graph. }
  {-----}

```

```

begin
  SetGraphMode(GraphMode);
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateIdealImpulseResponse;
  DesignOver:=False;
  if (UpCase(OutOnGraph)='Y') then begin
    WishHardCopy;
    SetGraphMode(GraphMode);
    Height:=MaxY div 2;
    ThreeDGraph(i);
    SetTextStyle(2,0,6);
    OutTextXY(round(MaxX/3.4),round(MaxY/1.32),'The Ideal Impulse Response');
    MessageOut;
    readln;
    HardCopyIfNecessary;
    RestoreCrtMode;
    CloseGraph;
  end;
End; { End of Procedure IdealImpulseOut. }

```

```

Procedure WeightedImpulseOut:
  {-----}
  { To output the impulse response after the ideal impulse }
  { response is weighted by the Kaiser window. }
  {-----}

```

```

Begin
  SetGraphMode(GraphMode);
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateWindowCoefficients;
  CalculateIdealImpulseResponse;
  DesignOver:=False;
  CalculateImpulseResponse;
  if (UpCase(OutOnGraph)='Y') then begin

```

```

WishHardCopy;
SetGraphMode(GraphMode);
Height:=Max Y div 2;
ThreeDGraph(i);
SetTextStyle(2,0,6);
OutTextXY(round(MaxX/3.6),round(MaxY/1.32),'Weighted Impulse Response');
MessageOut;
readln;
HardCopyIfNecessary;
if (UpCase(Hard_Copy) = 'Y') then
  RestoreCrtMode;
  CloseGraph;
end;
End; { End of Procedure WeightedImpulseOut. }

procedure FrequencyResponseOut;
{-----}
{ To Compute and output the filter parameters of the filter designed.}
{-----}
Begin
  SetGraphMode;
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateWindowCoefficients;
  CalculateIdealImpulseResponse;
  CalculateImpulseResponse;
  CalculateFourierTransform;
  CalculateFilterParameters;
End; { End of Procedure FrequencyResponseOut. }

procedure OutFrequencyOnGraph; { To draw frequency response on 3-D graph. }
Begin;
  WishHardCopy;
  SetGraphMode(GraphMode);
  Height:=Max Y div 4;
  ThreeDGraph(w);
  SetTextStyle(2,0,6);
  OutTextXY(round(MaxX/3.2),round(MaxY/1.32),'The Frequency Response');
  DesignOver:=True;
  MessageOut;
  readln;
  HardCopyIfNecessary;
  RestoreCrtMode;
  writeln;
  writeln('The Impulse response of the Filter just designed is :');
  writeln;
  Tabulate(i);
  readln;
  CloseGraph;
End; { End of Procedure OutFrequencyOnGraph. }

```

```

procedure AllOut; { To output calculated quantities. }
Begin
  SetGraphMode(GraphMode);
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  DesignOver:=False;
  CalculateWindowCoefficients;
  if (UpCase(OutOnGraph)='Y') then begin
    WishHardCopy;
    SetGraphMode(GraphMode);
    Height:=MaxY div 3;
    ThreeDGraph(w);
    SetTextStyle(2,0,6);
    OutTextXY(round(MaxX/3.2),round(MaxY/1.32),'The Window Function. ');
    MessageOut;
    readln;
    HardCopyIfNecessary;
    RestoreCrtMode;
  end;
  SetGraphMode(GraphMode);
  ClearViewPort;
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateIdealImpulseResponse;
  if (UpCase(OutOnGraph)='Y') then begin
    WishHardCopy;
    SetGraphMode(GraphMode);
    Height:=MaxY div 2;
    ThreeDGraph(i);
    SetTextStyle(2,0,6);
    OutTextXY(round(MaxX/3.4),round(MaxY/1.32),'The Ideal Impulse Response ');
    MessageOut;
    readln;
    HardCopyIfNecessary;
    RestoreCrtMode;
  end;
  SetGraphMode(GraphMode);
  ClearViewPort;
  SetTextStyle(0,0,3);
  OutText('WORKING !!');
  CalculateImpulseResponse;
  if (UpCase(OutOnGraph)='Y') then begin
    WishHardCopy;
    SetGraphMode(GraphMode);
    Height:=MaxY div 2;
    ThreeDGraph(i);
    SetTextStyle(2,0,6);
    OutTextXY(round(MaxX/3.6),round(MaxY/1.32),'Weighted Impulse Response ');
    MessageOut;
    readln;
    HardCopyIfNecessary;
  end;
end;

```

```

RestoreCrtMode;
end;
SetGraphMode(GraphMode);
ClearViewPort;
SetTextStyle(0,0,3);
OutText('WORKING !!');
CalculateFouRierTransform;
RestoreCrtMode;
CalculateFilterParameters;
End; { End of Procedura AllOut. }

```

```

procedure FilterDesignOut;

```

```

{-----}
{ This procedure controls the sequence of procedures that result }
{ in the 3-D graph representation of the filter designed.       }
{-----}

```

```

Begin

```

```

InputFilterParameters;
SpecificationsAndParameters;
writeln;
writeln('Press "ENTER" to continue!');
readln;
GRaphDriver:=DEtect;
InitGraph(GraphDriver,GraphMode.'');
MaxX:=GetMaxX;
MaxY:=GetMaxY;
RestoreCrtMode;
writeln('1. Output the Window Coefficients Only. ');
writeln('2. Output the Ideal Impulse Response Only. ');
writeln('3. Output the Weighted Impulse Response Only. ');
writeln('4. Output the Frequency Response Only. ');
writeln('5. Output All the Above. ');
writeln;
writeln('Choose any of the options above by printing the');
writeln('corresponding number. ');
writeln;
Repeat
write(' Chosen Option : ');
readln(Choice);
writeln;
case Choice of
'1' : WindowOut;
'2' : IdealImpulseOut;
'3' : WeightedImpulseOut;
'4' : FrequencyResponseOut;
'5' : AllOut
else begin
writeln(Chr(7));
writeln('NON-EXISTING OPTION. TRY AGAIN. ');
writeln;
end;
end; { end of CASE. }

```

```

        until Choice IN['1'..'5']
End; { End of Procedure FilterDesignOut. }

function finished:boolean;
var
    answer:char;
Begin
    writeln;
    write('Another Design(Y/N)?');
    readln(answer);
    writeln;
    finished:=(answer <> 'Y') and (answer <> 'y')
End: { End of function Finished. }

BEGIN { Main Program }
    LoadTheFont( @LITTfont);
    LoadTheDriver( @EGAVGADriver);
    repeat
        writeln(' Here are two options to choose from. Choose the');
        writeln(' option you want by printing the number');
        writeln(' corresponding to your choice. ');
        writeln;
        writeln('1. Process an image with a filter and see result. ');
        writeln('2. Design Filter and Output its characteristics');
        writeln(' on 3-D graphs. ');
        repeat
            writeln;
            write(' Chosen option                : ');
            readln(MyChoice);
            writeln;
            if (MyChoice='1') then ProcessImage
            else if (MyChoice='2') then FilterDesignOut
            else begin
                write(Chr(7));
                writeln('You have Typed ',MyChoice,'. Please Type 1 or 2. ');
                writeln;
            end
        until MyChoice In ['1','2']
    until finished
END. { ENDof Main program. }

```