



AVIT
AARUPADAI VEEDU INSTITUTE OF TECHNOLOGY



VINAYAKA MISSION'S
RESEARCH FOUNDATION
(Deemed to be University under section 3 of the UGC Act 1956)



Accredited by NAAC

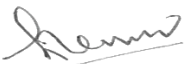


Approved by AICTE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DIGITAL IMAGE PROCESSING LAB MANUAL

Program/ Branch : B. E., / ECE
Year / Semester : III/ V
Academic Year : 2021 – 2022 (Odd Semester)
Regulation : R 2017


HOD/ ECE

17ECCC87		DIGITAL IMAGE PROCESSING LAB								Category		L	T	P	Credit
										CC		0	0	4	2
PREAMBLE															
To understand and implement image processing techniques using open source software															
PRERQUISITE															
Signal Processing															
COURSE OBJECTIVES															
1	To understand image acquisition and storage using a open source software – SCILAB														
2	To study and analyze different image transforms on images														
3	To study, analyze and apply different techniques and algorithms for image enhancement														
4	To study, analyze and apply different techniques and algorithms for image restoration														
5	To study, analyze and apply different techniques and algorithms for image compression														
COURSE OUTCOMES															
On the successful completion of the course, students will be able to															
CO1.Understand the acquisition and storage of different types of images													Understand		
CO2. Understand Analyze and Apply different image transforms and their properties													Apply		
CO3. Apply different Image Smoothing & Sharpening algorithms in time and frequency domain													Apply		
CO4. Apply different algorithms for image restoration													Apply		
CO5. Apply different techniques for image segmentation													Apply		
CO6. Apply different image compression techniques													Apply		
MAPPING WITH PROGRAMME OUTCOMES AND PROGRAMME SPECIFIC OUTCOMES															
COs	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO2	PSO3
CO1	S	M	-	-	M	-	-	-	-	-	-	M	-	-	-
CO2	S	M	M	M	M	-	-	-	-	-	-	M	-	-	-
CO3	S	M	M	M	M	-	-	-	-	-	-	M	-	-	-
CO4	S	M	M	M	M	-	-	-	-	-	-	M	-	-	-
CO5	S	M	M	M	M	-	-	-	-	-	-	M	-	-	-
CO6	S	M	M	M	M	-	-	-	-	-	-	M	-	-	-
S- Strong; M-Medium; L-Low															

List of Experiments

- 1) To acquire an image, store in different formats and display the properties of the images
- 2) To find the discrete Fourier transform of a gray scale image and perform inverse transform to get back the image
- 3) Analyze the rotation and convolution properties of the Fourier transform using any gray scale image
- 4) Find the discrete cosine transform of a given image. Compare discrete Fourier transform and discrete cosine transforms
- 5) Apply histogram equalization for enhancing the given images
- 6) Perform image enhancement, smoothing and sharpening, in spatial domain using different spatial filters and compare the performances
- 7) Perform image enhancement, smoothing and sharpening, in frequency domain using different filters and compare the performances
- 8) Perform noise removal using different spatial filters and compare their performances
- 9) For the given image perform edge detection using different operators and compare the results
- 10) For a given image, compress and decompress using wavelets. Study and compare the efficiency of the scheme with any two schemes

COURSE DESIGNERS

S. No.	Name of the Faculty	Mail ID
1	Mr. P. Subramanian	subramanian@avit.ac.in
2	Mrs.S.Valarmathy	valarmathy@vmkvec.edu.in
3	Mr.R.Ramani	ramani@vmkvec.edu.in

1. IMAGE TYPES

Aim:

To display images of different types along with the information about the images

Theory:

The different image types used in this program are

bmp – bitmap image file

jpeg – joint photographic expert group

png – portable network graphics

tiff – tagged Image File format

The functions used in this program are

subplot divides the current figure into rectangular panes that are numbered rowwise. Each pane contains an axes object which you can manipulate using Axes Properties. Subsequent plots are output to the current pane. `h = subplot(m,n,p)` or `subplot(mnp)` breaks the figure window into an m-by-n matrix of small axes, selects the pth axes object for the current plot, and returns the axes handle. The axes are counted along the top row of the figure window, then the second row, etc.

imshow – The function `imshow(filename)` displays the image stored in the graphics file `filename`. The file must contain an image that can be read by ***imread*** or `dicomread`. `imshow` calls `imread` or `dicomread` to read the image from the file, but does not store the image data in the MATLAB workspace. If the file contains multiple images, `imshow` displays the first image in the file. The file must be in the current directory or on the MATLAB path.

impixelinfo – The function `impixelinfo` creates a Pixel Information tool in the current figure. The Pixel Information tool displays information about the pixel in an image that the pointer is positioned over. The tool can display pixel information for all the images in a figure.

imageinfo – The function `imageinfo` creates an Image Information tool associated with the image in the current figure. The tool displays information about the basic attributes of the target image in a separate figure.

title – The function `title('string')` outputs the string at the top and in the center of the current axes.

MATLAB Program

```
clc;
clear all;
close all;

subplot(2,2,1), imshow('cameraman.tif'),title('cameraman.tif');
subplot(2,2,2), imshow('peppers.png'),title('peppers.png');
subplot(2,2,3), imshow('baby.bmp'),title('baby.bmp');
subplot(2,2,4), imshow('oldman.jpg'),title('oldman.png');
impixelinfo;

imageinfo('cameraman.tif');
imageinfo('peppers.png');
imageinfo('baby.bmp');
imageinfo('oldman.jpg');
```

OUTPUT

Image Information for peppers.png

Filename	C:\Program Files\MATLAB\R2009b\toolbox\images\indemos\peppers.png
FileModDate	16-Dec-2002 06:10:58
FileSize	287677
Format	png
FormatVersion	[]
Width	512
Height	384
BitDepth	24
ColorType	truecolor
FormatSignature	[137 80 78 71 13 10 26 10]
Colormap	[]
Histogram	[]
InterlaceType	none
Transparency	none
SimpleTransparencyData	[]
BackgroundColor	[]
RenderingIntent	[]
Chromaticities	[]
Gamma	[]
XResolution	[]
YResolution	[]
ResolutionUnit	[]
XOffset	[]
YOffset	[]
OffsetUnit	[]
SignificantBits	[]
ImageModTime	16 Jul 2002 16:46:41 +0000
Title	[]
Author	[]
Description	Zesty peppers
Copyright	Copyright The MathWorks, Inc.
CreationTime	[]
Software	[]

Disclaimer	[]
Warning	[]
Source	[]
Comment	[]
OtherText	[]

Image information for camearman.tif

Filename	C:\Program Files\MATLAB\R2009b\toolbox\images\imdemos\cameraman. tif
FileModDate	04-Dec-2000 13:57:54
FileSize	65240
Format	tif
FormatVersion	[]
Width	256
Height	256
BitDepth	8
ColorType	grayscale
FormatSignature	[77 77 42 0]
ByteOrder	little-endian
NewSubFileType	0
BitsPerSample	8
Compression	PackBits
PhotometricInterpretatio n	BlackIsZero
StripOffsets	[8x1 double]
SamplesPerPixel	1
RowsPerStrip	32
StripByteCounts	[8x1 double]
XResolution	72
YResolution	72
ResolutionUnit	None
Colormap	[]
PlanarConfiguration	Chunky
TileWidth	[]

TileLength	[]
TileOffsets	[]
TileByteCounts	[]
Orientation	1
FillOrder	1
GrayResponseUnit	0.0100
MaxSampleValue	255
MinSampleValue	0
Thresholding	1
Offset	64872
ImageDescription	This image is distributed by The MathWorks, Inc. with permission from the Massachusetts Institute of Technology.

Image information for baby.bmp

Filename	E:\ baby.bmp
FileModDate	03-Mar-2006 04:05:44
FileSize	687054
Format	Bmp
FormatVersion	Version 3 (Microsoft Windows 3.x)
Width	500
Height	458
BitDepth	24
ColorType	truecolor
FormatSignature	BM
NumColormapEntries	0
Colormap	[]
RedMask	[]
GreenMask	[]
BlueMask	[]
ImageDataOffset	54
BitmapHeaderSize	40
NumPlanes	1
CompressionType	None
BitmapSize	687000
HorzResolution	0
VertResolution	0
NumColorsUsed	0
NumImportantColors	0

Image information for oldman.jpg

Filename	E:\oldman.jpg
FileModDate	28-Jun-2014 13:25:04
FileSize	43635
Format	Jpg
FormatVersion	"
Width	526
Height	526
BitDepth	24
ColorType	truecolor
FormatSignature	"
NumberOfSamples	3
CodingMethod	Huffman
CodingProcess	Progressive
Comment	{ '*' }

Result:

Images of different types have been displayed and studied.

2. IMAGE TRANSFORM - FOURIER TRANSFORM

Aim:

The program is to

- a) Find the Fourier Transform of an image and also the inverse fourier transform.
- b) Verify the rotation property of Fourier Transform
- c) Verify the convolution property of the Fourier Transform

Theory:

The fourier transform, developed by Jean Baptiste Joesph Fourier, is widely used in the field of image processing. An image is a spatially varying function. One way to analyse spatial variations is to decompose an image into a set of orthogonal functions, one such being the fourier functions. A fourier transform is used to transform an intensity image into the domain of spatial frequency.

The program first creates an image, finds its fourier transform and displays it. The inverse fourier transform is found to obtain and display the input image.

The rotation property of Fourier transform states that if an image is rotated by a certain angle, its fourier transform is also rotated by the same angle. The program rotates the input image by an angle of 45 degrees. The fourier transform of the rotated image is found and displayed.

The convolution property of the Fourier transform states that the convolution of two images in time domain is equivalent to multiplication of the fourier transforms of the individual images. Two images are created. The convolution of the two images is carried out and displayed. The fourier transforms of the individual images are found, multiplied and inverse fourier transform is applied to the product. We can observe that the output image is the same as obtained through convolution.

MATLAB PROGRAM

```
clc;  
clear all;  
close all;
```

*% to create an image, find and display its Fourier transform & also to find the
inverse fourier transform %*

```
a=zeros(256,256);  
a(110:140,110:140)=1;
```

```
subplot(3,1,1), imshow(a), title('Input Image');  
a1=log(1+abs(fftshift(fft2(a))));  
subplot(3,1,2), imshow(mat2gray(a1)), title('Fourier Transform of the input  
image');
```

```
a1=fft2(a);  
b=ifft2(a1);  
subplot(3,1,3), imshow (b), title('Input image obtained by Inverse fourier  
transform');
```

% To verify the rotation property of the Fourier Transform %

```
a=zeros(256,256);  
a(110:140,110:140)=1;  
figure;  
subplot(2,2,1), imshow(a), title('Input Image');  
a1=log(1+abs(fftshift(fft2(a))));  
subplot(2,2,2), imshow(mat2gray(a1)), title('Fourier Transform of the input  
image');
```

```
c=imrotate(a,45,'bilinear','crop');  
subplot(2,2,3), imshow(c), title('Input image rotated by 45 degrees');
```

```
c1=log(1+abs(fftshift(fft2(c))));  
subplot(2,2,4), imshow(mat2gray(c1)), title('Fourier transform of the rotated  
image');
```

% TO VERIFY THE CONVOLUTION PROPERTY %

```

figure;
a=zeros(256,256);
a(110:140,110:140)=1;
subplot(2,2,1),imshow(a),title('First input image');

b=zeros(256,256);
b(170:200,170:200)=1;
subplot(2,2,2), imshow(b), title('Second input image');

d=conv2(a,b,'same');
subplot(2,2,3), imshow(d),title('Convolution of input images in time domain');

a1=fft2(a);
b1=fft2(b);
e=a1.*b1;
f=fftshift(iff2(e));
subplot(2,2,4), imshow(f), title({'Result of Multiplication of FFTs'; 'of input
images and IFFT'})

```

Result:

Thus the fourier transform and inverse fourier transform has been studied as well as the rotation and convolution properties of the fourier transform.

3. IMAGE TRANSFORM – DISCRETE COSINE TRANSFORM

Aim:

The program is to find the Discrete Cosine Transform of an image and also the inverse discrete cosine transform.

Theory:

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical for compression, since it turns out that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions. In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. The DCTs are generally related to Fourier Series coefficients of a periodically and symmetrically extended sequence whereas DFTs are related to Fourier Series coefficients of a periodically extended sequence. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), whereas in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common.

Result:

Thus the discrete cosine transform and inverse discrete cosine transform have been studied.

MATLAB PROGRAM

```
clc;  
clear all;  
close all;
```

*% to create an image, find and display its discrete cosine transform & also to
find the inverse fourier transform %*

```
I4=[1 1 1 1; 1 1 1 1; 1 1 1 1; 1 1 1 1]  
D4=dctmtx(4)  
output4 = D4*I4*D4  
input4=D4'*output4*D4'
```

% To find the DCT of a given image and the inverse transform %

```
I = imread('cameraman.tif');  
J = dct2(I);  
K = idct2(J);  
imshow(I),title('Input Image');  
figure,imshow(log(abs(J)),[]), colorbar, title('DCT of input image');  
figure,imshow(K,[0 255]), title('Input Image obtained by IDCT');
```

4. IMAGE TRANSFORM – HADAMARD TRANSFORM

Aim:

To generate Hadamard matrix for $N=4$, find the Hadamard transform for a matrix of size 4×4 . To generate a binary figure of size (256×256) and find its Hadamard transform as well as verify the inverse Hadamard.

Theory:

The Hadamard transform is based on basis functions that are simply +1 or -1, instead of the more complex sine and cosine functions used in the Fourier transform. The function Hadamard is used to generate the Hadamard matrix of required size. The program generates a Hadamard matrix of size 4×4 . The Hadamard of the input matrix is found and the inverse also. The program also generates an image of size 256×256 . The Hadamard and the inverse Hadamard are found to verify the program.

MATLAB PROGRAM

```
clc;
clear all;
close all;

A=[1 1 1 1; 1 1 1 1; 1 1 1 1; 1 1 1 1];
disp('The input matrix is'),A
H4=hadamard(4);
disp('The hadamard matrix for N=4 is: '), H4
B = 1/4*H4*A*H4;
disp('The hadamard transform of the input matrix is'),B
A=1/4*H4'*B*H4';
disp('The inverse hadamard is'),A
```

```
A=zeros(256,256);
A(100:150,100:150)=1;
subplot(1,3,1),imshow(A),title('Input Image');
H256=hadamard(256);
B=1/256*H256*A*H256;
subplot(1,3,2),imshow(B),title('hadamard of the input image');
A=1/256*H256'*B*H256';
subplot(1,3,3),imshow(A),title('Input Image obtained by inverse transform');
```

Result:

The Hadamard matrix has been generated and verified for an input matrix of size N=4. The Hadamard of a generated image of size N=256 is generated and verified.

The input matrix is

A =

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

The hadamard matrix for N=4 is:

H4 =

1	1	1	1
1	-1	1	-1
1	1	-1	-1
1	-1	-1	1

The hadamard transform of the input matrix is

B =

4	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

The inverse hadamard is

A =

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

5. HISTOGRAM EQUALISATION

Aim:

To study the different methods of histogram equalization and their effects on enhancing the image. .

Theory:

The histogram of an image with intensity levels in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$, where r_k is the k th intensity level and n_k is the number of pixels in the image with intensity r_k . A histogram shows us the distribution of grey levels in the image. Histograms are the basis for numerous spatial domain techniques. Histogram manipulation can be used for image enhancement. In histogram equalisation, the pixels are made to occupy the entire range of possible intensity levels and in addition the pixels are made to be distributed uniformly. The net effect will be an image that shows a great deal of gray level detail and has high dynamic range.

The function `imhist` is used to display the histogram of a given image. The figure shows a selection of images and their histograms. Notice the relationships between the images and their histograms. Note that the high contrast image has the most evenly spaced histogram.

Function - ***histeq***

`g=histeq(f, nlev)`

`f` is the input image array

`nlev` is the number of intensity levels specified for the output image. If `nlev` is equal to the total number of possible levels in the input image then `histeq` implements the transformation directly. If `nlev` is less than the total number of possible levels in the input image then `histeq` attempts to distribute the levels so that they will approximate a flat histogram.

We can see that the histogram equalization done here has produced an image with washed out appearance.

The reason can be understood by studying the histogram of the equalized image. The intensity levels have been shifted to the upper one half of the gray scale thereby giving the image the low contrast washed out appearance.

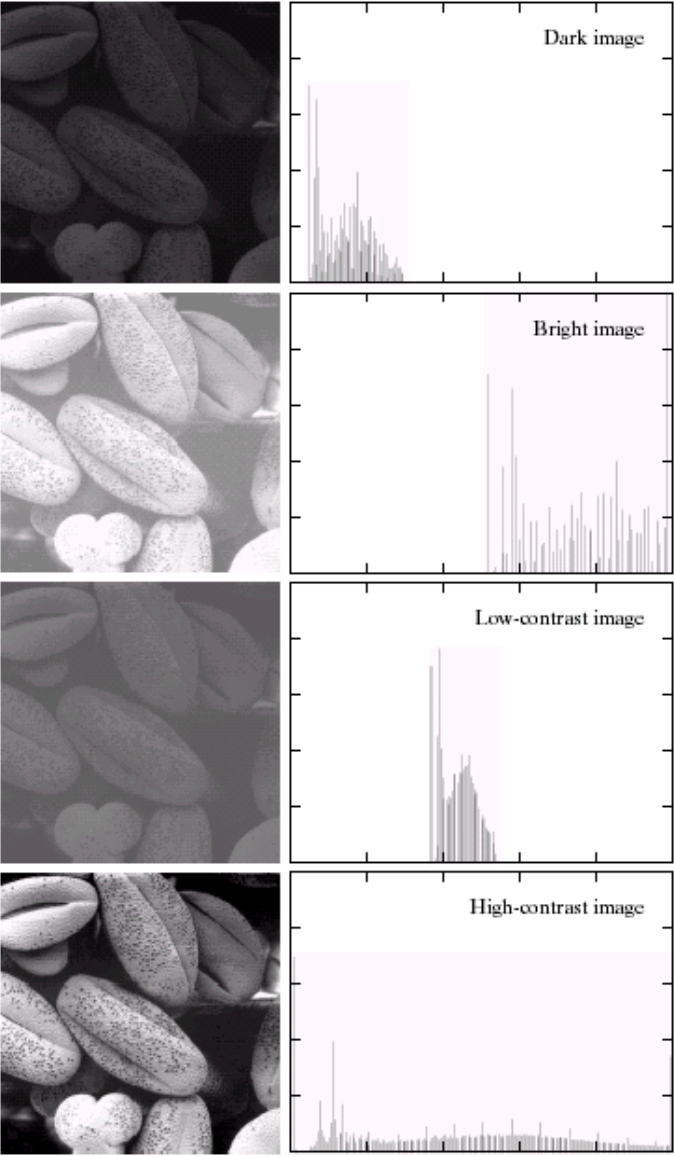
Function `adapthisteq`

enhances the contrast of the grayscale image by transforming the values using contrast-limited adaptive histogram equalization (CLAHE).

CLAHE operates on small regions in the image, called *tiles*, rather than the entire image. Each tile's contrast is enhanced. The neighboring tiles are then

combined using bilinear interpolation to eliminate artificially induced boundaries. The contrast, especially in homogeneous areas, can be limited to avoid amplifying any noise that might be present in the image.

`J = adapthisteq(I,param1,val1,param2,val2...)` specifies any of the additional parameter/value pairs listed in the following table. Parameter names can be abbreviated, and case does not matter.



MATLAB PROGRAM

```
clc;
clear all;
close all;
f=imread('fig0308(a).tif');
imshow(f);
title('Orginal image');
figure, imhist(f)
ylim('auto')
title('Histogram of the original image');
g=histeq(f,256);
figure,imshow(g)
title('Histogram equalised image');
figure,imhist(g)
ylim('auto')
title('Histogram of the equalised image');
```

```
clc;
clear all;
close all;
f=imread('fig0310(a).tif');
imshow(f);
title('Orginal image');
figure, imhist(f)
ylim('auto')
title('Histogram of the original image');
g=histeq(f,500);
figure,imshow(g)
title('Histogram equalised image');
figure,imhist(g)
ylim('auto')
title('Histogram of the equalised image');
```

```
clc;
clear all;
close all;
f=imread('fig0310(a).tif');
imshow(f);
```

```
title('Orginal image');
figure, imhist(f)
ylim('auto')
title('Histogram of the original image');
g=histeq(f,256);
figure,imshow(g)
title('Histogram equalised image');
figure,imhist(g)
ylim('auto')
title('Histogram of the equalised image');
h=adapthisteq(f, 'Numtiles', [25 25], 'cliplimit', 0.05, 'distribution',
'rayleigh');
figure, imshow(h)
title('contrast limited adaptive histogram equalisation');
figure,imhist(h)
title('histogram of contrast limited adaptive histogram equalised image');
```

6. IMAGE SMOOTHING

Aim:

To smoothen the given image using different spatial filters and study the effects.

Theory:

Image smoothing spatial filters are used for blurring and noise reduction. The output of a smoothing linear filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters are also called as averaging filters. In averaging filters, the value of every pixel in the image is replaced by the average of the gray levels in the neighborhood defined by the filter mask. This process results in an image with reduced sharp transitions in gray levels. However edges which are desirable features of an image are characterized by sharp transitions in gray levels. Hence averaging filters have the undesirable effect that blurs the edges. A major use of averaging filters is in the reduction of irrelevant detail in an image. Irrelevant details refer to the pixel regions that are small with respect to the size of the filter mask. A spatial averaging filter in which all coefficients are equal is sometimes is called a box filter.

We consider an input image of size 500 x 500 pixels as shown. The black squares at the top are of sizes 3, 5, 9, 15, 25, 35, 45 and 55 pixels. Their borders are 25 pixels apart. The letters at the bottom range are of sizes 10, 12, 14, 16, 18, 20, 22 and 24 pixels. The large letter is 60 pixels. The vertical bars are 5 pixels wide, 100 pixels high and separated by 20 pixels. The diameter of the circles is 25 pixels and the circles are 15 pixels apart. The noisy rectangles are of size 50 x 120 pixels.

The filter masks used in the program are of sizes 3, 5, 9, 15 and 35 pixels. For filter mask of size 3 x 3, there is a general slight blurring throughout the image. But details that are approximately the same size as the filter mask are affected considerably more. For example the small letter 'a' and the fine grain noise show significant blurring when compared to the rest of the image. The jagged borders of the characters and gray circles have been smoothed pleasantly. For the 5 x 5 mask, the result is similar with a slight increase in the blurring. For n=9, we can see considerably more blurring. For 15 x 15 and 35 x 35 masks, the results are extreme with respect to the size of the objects.

MATLAB PROGRAM

```
clc;
clear all;
close all;

f=imread('inputimage.tif');
subplot(1,2,1),imshow(f),title('Input Image');

w=1/9*ones(3);
g1=imfilter(f,w);
subplot(1,2,2),imshow(g1),title('Image smoothened by square averaging filter
mask of size 3 x 3');

w=1/25*ones(5);
g2=imfilter(f,w);
figure, subplot(1,2,1),imshow(g2),title('Image smoothened by square
averaging filter mask of size 5 x 5');

w=1/81*ones(9);
g3=imfilter(f,w);
subplot(1,2,2),imshow(g3),title('Image smoothened by square averaging filter
mask of size 9 x 9');

w=1/225*ones(15);
g4=imfilter(f,w);
figure, subplot(1,2,1),imshow(g4),title('Image smoothened by square
averaging filter mask of size 15 x 15');

w=1/1225*ones(35);
g5=imfilter(f,w);
subplot(1,2,2),imshow(g5),title('Image smoothened by square averaging filter
mask of size 35 x 35');
```

Result:

Different spatial filters have been used for image smoothening and the effects have been studied.

7. IMAGE SHARPENING

Aim:

To sharpen the given image using different spatial filters and study the effects.

Theory:

Sharpening spatial filters seek to highlight fine detail by removing blurring from images and highlighting the edges. Sharpening filters are based on spatial differentiation. Differentiation measures the rate of change of a function. It's just the difference between subsequent values and measures the rate of change of the function. The second order derivative simply takes into account the values both before and after the current value. The 2nd derivative is more useful for image enhancement than the 1st derivative due to the stronger response to fine detail and simpler implementation. The first sharpening filter we will look at is the Laplacian filter, one of the simplest sharpening filters. Applying the Laplacian to an image we get a new image that highlights edges and other discontinuities. The result of Laplacian filtering is not an enhanced image. We have to do more work in order to get our final image. We have to subtract the Laplacian result from the original image to generate our final sharpened enhanced image. In the final sharpened image edges and fine detail are much more obvious. The same process can be repeated for Sobel and Prewitt filters also.

Result:

Different spatial filters have been used for image sharpening and the effects have been studied.

MATLAB PROGRAM

```
clc;
clear all;
close all;

f=imread('fig0413(a).tif');
subplot(2,2,1),imshow(f),title('input image');

w=1/81*ones(9);
g=imfilter(f,w);
subplot(2,2,2),imshow(g),title('Smoothened Image');

w=fspecial('laplacian',0);
g1=imfilter(g,w,'replicate');
i=f-g1;
subplot(2,2,3),imshow(i);title('special laplacian filter');

w=fspecial('prewitt');
g2=imfilter(g,w);
i=f-g2;
figure, imshow(i);title('Prewitt filter');

w=fspecial('sobel');
g3=imfilter(g,w);
i=f-g3;
figure, imshow(i);title('Sobel filter');
```

8. EDGE DETECTION

Aim:

To detect the edges in the given image using different operators.

Theory:

The basic idea behind edge detection is to find places in an image where the intensity changes rapidly, using one of the following criteria

a) Find places where the first derivative of the intensity is greater in magnitude than a specified threshold.

b) Find places where the second derivative of the intensity has a zero crossing

The function `edge` in the MATLAB provides several edge estimations based on the above criteria. Sobel edge detector computes the gradient by using the discrete differences between the rows and columns of a discrete neighborhood where the center pixel is in each row and column is weighed by 2 to provide smoothing. Prewitt edge detector finds the edges using the prewitts approximation to the first order derivatives implemented using prewitt mask. Robert edge detector finds the edges using the Robert approximation to the first order derivatives implemented using Robert cross gradient mask. The lapalcian of Gaussian edge detector finds edges by looking for zero crossings after filtering the input image with a LoG filter. The zero Crossing detector finds the edges by looking for zero crossings after filtering the input image with a specified filter. The canny edge detector finds edges by looking for local maxima of the gradient of the input image. The gradient is calculated using the derivatives of a Gaussian filter. The method uses two thresholds to detect strong and weak edges in the output, and includes the weak edges only if they are connected to the strong edges. Therefore this method is more likely to detect true weak edges. The program uses all the above edge detectors to find the edges of a checkerboard image.

MATLAB PROGRAM

```
clc;
close all;
clear all;

a = checkerboard(15,5,5)>0.5;
subplot(2,3,1);
imshow(a); title('Input Image');

b = edge(a,'sobel',0.1,'both');
subplot(2,3,2);
imshow(b); title('Sobel Horizontal and vertical Edge Detection ');

b = edge(a,'sobel',0.1,'horizontal');
subplot(2,3,3);
imshow(b); title('Sobel Horizontal Edge Detection');

b = edge(a,'sobel',0.1,'vertical');
subplot(2,3,4);
imshow(b); title('Sobel Vertical Edge Detection');

d = edge(a,'prewitt','horizontal',0.1);
subplot(2,3,5);
imshow(d); title('Prewitt Horizontal Edge Detection');

d = edge(a,'prewitt','vertical',0.1);
subplot(2,3,6);
imshow(d); title('Prewitt Vertical Edge Detection');

d = edge(a,'prewitt','both',0.1);
figure, subplot(2,3,1);
imshow(d); title('Prewitt Horizontal and Vertical Edge Detection');

c = edge(a,'roberts','both');
subplot(2,3,2);
imshow(c); title('Roberts Horizontal and Vertical Edge Detection');
```

```
e = edge(a,'canny',.01);  
subplot(2,3,3);  
imshow(e); title('Canny Edge Detection ');
```

```
f = edge(a,'zerocross');  
subplot(2,3,4);  
imshow(f); title('Zero Cross Edge Detection');
```

```
c = edge(a,'log');  
subplot(2,3,5);  
imshow(c); title('Laplacian of Gaussian Edge Detection');
```

Result:

The detection of edges in the given image has been carried out using various edge detectors and verified.

9. NOISE REMOVAL

Aim:

To remove the noise from the given image.

Theory:

The method of choice for reducing noise is the spatial filtering. Different spatial filters like the arithmetic mean filter, geometric mean filter, harmonic mean filter, contraharmonic mean filter, median filter, max filter, min filter, midpoint filter and alpha trimmed mean filter.

The different types of noises that can occur are random noise, salt and pepper noise, salt noise, pepper noise, Gaussian noise etc.,

The type of filter used to remove or reduce noise in a given noisy image depends on the noise present in the image and the image characteristic itself.

In this program we are considering an input image corrupted by 4 different noises namely salt and pepper noise, salt noise, pepper noise and Gaussian noise. For each noise we are considering two filters and verify the performance by calculating the rmse values.

The program can be extended by including all filters for all the types of noises.

MATLAB PROGRAM

```
clc;
clear all;
close all;

f=imread('fig0505(a).tif');
g=imnoise(f,'salt & pepper',0.1);
subplot(2,3,1), imshow(g),title('Input image corrupted by salt and pepper
noise');
g1=spfilt(g,'amean',3,3);
subplot(2,3,2),imshow(g1),title('Salt and Pepper Noise removal with
arithmetic mean filter ');
g2=spfilt(g,'median',3,3);
subplot(2,3,3),imshow(g2),title('Salt and Pepper Noise removal with median
filter ');

[M N]=size(f);
R=imnoise2('salt & pepper',M,N,0.1,0);
gp=f;
gp(R==0)=0;
subplot(2,3,4),imshow(gp),title('Image corrupted by Pepper noise');

g3=spfilt(gp,'chmean',3,3,1.5);
subplot(2,3,5),imshow(g3),title('Pepper noise removal by contraharmonic
mean filter');
g4=spfilt(gp,'max',3,3);
subplot(2,3,6),imshow(g4),title('Pepper noise removal by max filter');
[M N]=size(f);
R=imnoise2('salt & pepper',M,N,0,0.1);
gs=f;
gs(R==1)=255;
figure,subplot(2,3,1),imshow(gs),title('Image corrupted by Salt noise');

g5=spfilt(gs,'atrimmed',3,3);
subplot(2,3,2),imshow(g5),title('Salt noise removal by alphas trimmed filter');
g6=spfilt(gs,'min',3,3);
subplot(2,3,3),imshow(g6),title('Salt noise removal by min filter');

gh=imnoise(f,'gaussian');
```

```

subplot(2,3,4),imshow(gh),title('Image corrupted by gaussian noise');

g7=spfilt(gh,'gmean',3,3);
subplot(2,3,5),imshow(g7),title('Gaussian noise removal by geometric mean
filter');
g8=spfilt(gh,'hmean',3,3);
subplot(2,3,6),imshow(g8),title('Gaussian removal by harmonic mean filter');

rmse1=compare(f,g);
disp('The RMSE between the input image and the image with salt and pepper
noise'),rmse1
rmse2=compare(f,g1);
disp('The RMSE after applying arithmetic mean filter for reducing salt and
pepper noise'),rmse2
rmse3=compare(f,g2);
disp('The RMSE after applying median filter for reducing salt and pepper
noise'),rmse3
rmse4=compare(f,gp);
disp('The RMSE between the input image and the image with pepper
noise'),rmse4
rmse5=compare(f,g3);
disp('The RMSE after applying contraharmonic mean filter for reducing
pepper noise'),rmse5
rmse6=compare(f,g4);
disp('The RMSE after applying max filter for reducing pepper noise'),rmse6
rmse7=compare(f,gs);
disp('The RMSE between the input image and the image with salt
noise'),rmse7
rmse8=compare(f,g5);
disp('The RMSE after applying alphatrimmed filter for reducing salt
noise'),rmse8
rmse9=compare(f,g6);
disp('The RMSE after applying min filter for reducing salt noise'),rmse9
rmse10=compare(f,gh);
disp('The RMSE between the input image and the image with gaussian
noise'),rmse10
rmse11=compare(f,g7);
disp('The RMSE after applying geometric mean filter for reducing gaussian
noise'),rmse11

```



```
rmse12=compare(f,g8);  
disp('The RMSE after applying harmonic mean filter for reducing gaussain  
noise'),rmse12
```

Result:

The performance of different filters have been studied for removing noises of different types.

10. INVERSE FILTERING

Aim:

To study the effect of inverse filtering in restoring degraded images.

Theory:

In the restoration of images degraded by noise and motion blur, the simplest approach is to ignore the noise term in the degradation model and form an estimate of the form $F^{\wedge}(u,v) = G(u,v) / H(u,v)$. The corresponding estimate of the image is obtained by taking the inverse fourier transform of $F^{\wedge}(u,v)$. This approach is called the inverse filtering. Taking noise into account, we can express our estimate as $F^{\wedge}(u,v) = F(u,v) + N(u,v)/H(u,v)$. Even if $H(u,v)$ is known exactly, $F(u,v)$ and thereby the input image cannot be recovered exactly because the noise component is a random function whose fourier transform $N(u,v)$ is not known. In addition there usually is the problem of $H(u,v)$ having numerous zeros. Even if the noise term $N(u,v)$ were negligible, dividing it by vanishing values of $H(u,v)$ would dominate restoration estimates. The typical approach when attempting inverse filtering is to form the ratio $F^{\wedge}(u,v) = G(u,v) / H(u,v)$ and then limit the frequency range for obtaining the inverse, to frequencies near the origin. The idea is that zeros in $H(u,v)$ are less likely to occur near the origin because the magnitude of the transform typically is at its highest values in that region. There are numerous variations of this basic theme.

In the program we first use direct inverse filtering to restore the image and observe that the process is dominated by noise. Having an estimations of the noise signal ratio improves the restoration process.

Result:

Image restoration using direct inverse filtering and with the usage of estimated NSR has been compared.

MATLAB PROGRAM

```
clc;
clear all;
close all;

I = im2double(imread('cameraman.tif'));
subplot(2,2,1),imshow(I),title('Input Image');
LEN = 21;
THETA = 11;
PSF = fspecial('motion', LEN, THETA);
blurred = imfilter(I, PSF, 'conv', 'circular');
noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred, 'gaussian', ...
                        noise_mean, noise_var);
subplot(2,2,2),imshow(blurred_noisy),title('Input Image degraded by motion
blur and noise');
estimated_nsr = 0;
wnr2 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
subplot(2,2,3), imshow(wnr2),title('Restoration of Blurred, Noisy Image Using
Ideal Inverse Filter');
estimated_nsr = noise_var / var(I(:));
wnr3 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
subplot(2,2,4),imshow(wnr3), title('Restoration of Blurred, Noisy Image Using
Estimated NSR');
```

11. LOSSLESS COMPRESSION

Aim:

To achieve lossless compression of the given image using MATLAB program

Theory:

Image compression addresses the problem of reducing the amount of data required to represent a digital image. Compression is achieved by removing one or three data redundancies namely coding redundancy, spatial/temporal redundancy and irrelevant information.

When coding the gray levels of an image, Huffman codes contain the smallest possible number of code symbols (bits) per source symbol subject to the constraint that the source symbols are coded one at a time. Huffman code generation is not in itself compression. To realize the compression that is built into a Huffman code, the symbols for which the code was created must be mapped in accordance with the code generated. The Huffman compression is achieved by using the function `mat2huff`. Huffman encoded images are of little use unless they can be decoded to recreate the original images from which they were derived. The decoder must compute the Huffman code used to encode and then inverse map the encoded data to rebuild the input image. The function `huff2mat` is used for the purpose.

The lossless predictive coding eliminates the interpixel redundancies of closely spaced pixels by extracting and coding only the new information in each pixel. The new information of a pixel is defined as the difference between the actual and the predicted value of that pixel. The functions `mat2lpc` and `lpc2mat` are used to implement the lossless predictive coding and decoding processes.

MATLAB PROGRAM

```
clc;
clear all;
close all;

f=imread('cameraman.tif');
subplot(1,3,1),imshow(f),title('Input Image');
c=mat2huff(f);
g=huff2mat(c);
cr1=imratio(f,c)
subplot(1,3,2),imshow(g, [0 255]),title('Image Decoded after Huffman
Coding');
rmse=compare(f,g)

c=mat2lpc(f);
cr1=imratio(f,c)
g=lpc2mat(c);
subplot(1,3,3),imshow(g, [0 255]),title('Image Decoded after Lossless
Predictive Coding');
rmse=compare(f,g)
```

Result:

Lossless compression has been done using Huffman and Lossless Predictive coding and the results verified.

12. WAVELET CODING

Aim:

To study the wavelet coding and its applications in image compression

Theory:

The wavelet transform has made it easier to compress, transmit and analyse many images. Unlike the fourier transform, whose basis functions are sinusoids, wavelet transforms are based on small waves called wavlets of varying frequency and limited duration. This allows them to provide the equivalent of a musical score for an image, revealing not only what notes (or frequencies) to play but also when to play them. Fourier transforms on the other hand provide only the notes or frequency information and the temporal information is lost in the transformation process.

The two dimensional wavelet transform results in a 2-dimensional scaling function and three directionally sensitive wavelets. The wavelets measure functional variations and intensity variations for images along different directions namely horizontal, vertical and diagonal.

The program does single level wavelet decomposition. The inverse wavelet transform gives the input image. Then the different coefficients namely approximate coefficients, horizontal detail coefficients and diagonal detail coefficients are made zero one at a time and inverse transform is applied in each case to observe the effect. It can be seen that the zeroing of approximate coefficients results in considerable loss of image information and a large RMSE value which indicates that the approximate coefficients are very much essential. The zeroing of horizontal, vertical or diagonal detail coefficients do not result in much loss of information as can be seen by their lower RMSE values.

MATLAB PROGRAM

```
clc
clear all
close all;

f = imread ('figure.tif');
subplot(1,3,1), imshow(f), title('Input Image');

[c,s] = wavefast(f,1,'sym4');
subplot(1,3,2), wavedisplay(c,s,-6), title('One Scael Wavelet Transform');

f1 = waveback(c,s,'sym4');
subplot(1,3,3), imshow(mat2gray(f1)), title('Image obtained thro inverse
wavelet');

[nc,y] = wavecut('a',c,s);
Figure, subplot(2,2,1), wavedisplay(nc,s,-6), title('Approximate coefficients
made zero');
f1=waveback(nc,s,'sym4');
f1=uint8 (f1)
subplot(2,2,2), imshow(mat2gray(f1)),title('Image obtained after zeroing all
approximate coefficients');

[nc,y] = wavecut('h',c,s);
figure, subplot(2,2,3), wavedisplay(nc,s,-6), title('Horizontal detail coefficients
made zero');
f2=waveback(nc,s,'sym4');
subplot(2,2,4), imshow(mat2gray(f2)),title('Image obtained after zeroing all
horizontal detail coefficients');

[nc,y] = wavecut('v',c,s);
figure, subplot(2,2,1), wavedisplay(nc,s,-6), title('Vertical detail coefficients
made zero');
f3=waveback(nc,s,'sym4');
subplot(2,2,2), imshow(mat2gray(f3)),title('Image obtained after zeroing all
vertical detail coefficients');
```

```
[nc,y] = wavecut('d'c,s);  
figure, subplot(2,2,3), wavedisplay(nc,s,-6), title('Diagonal detail coefficients  
made zero');  
f4=waveback(nc,s,'sym4');  
subplot(2,2,4), imshow(mat2gray(f4)),title('Image obtained after zeroing all  
diagonal detail coefficients');  
rmse1=compare(f,f1);  
rmse2=compare(f,f2);  
rmse3=compare(f,f3);  
rmse4=compare(f,f4);
```