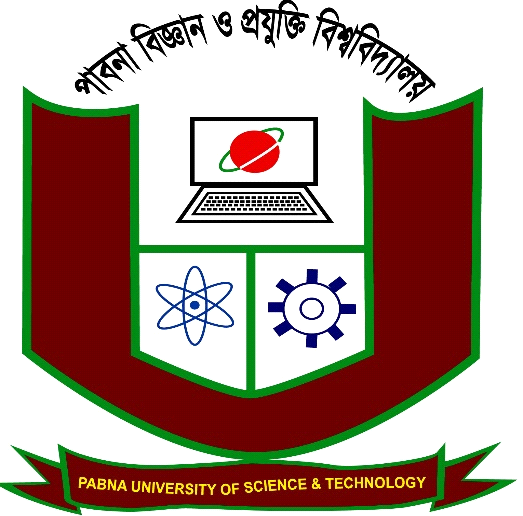
**Pabna University of Science and Technology**



Faculty of **Engineering and Technology**

Department of **Information and Communication Engineering**

**Lab Report**

*Course Code: ICE-3208*

*Course Title: Digital Image and Speech Processing Sessional*

*Submitted by:* *Submitted to:*

Md. Shihabul Hasan Shihab Md. Anwar Hossain

ID: 170627 Associate Professor

Session: 2016-17 Department of ICE

Department of ICE Pabna University of

Pabna University of Science & Technology

Science & Technology

**INDEX**

| Exp. No. | Problem Description | Page No. |
| --- | --- | --- |
|  | Write a MATLAB program for (i) negative (ii) Log (iii) power-law transformation of a gray level image. | 03 |
|  | Write a MATLAB program for (i) showing histogram (ii) contrast stretching (iii) histogram equalization of a gray level image. | 08 |
|  | Write a MATLAB program for (i) high pass and lowpass filter (ii) average filter (iii) median, max and min filter of a gray level image. | 18 |
|  | Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian lowpass and highpass filter of a gray level image in frequency domain. | 24 |
|  | Write a MATLAB program for (i) Laplacian (ii) homomorphic filter of a gray level image in frequency domain. | 34 |
|  | Write a MATLAB program for (i) arithmetic and geometric (ii) harmonic and contraharmonic (iii) midpoint and alpha-trimmed mean filter of a gray level image. | 38 |
|  | Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian bandreject and bandpass filter of a gray level image. | 46 |
|  | Write a MATLAB program for Wiener filter of a gray level image. | 57 |
|  | Write a MATLAB program for separating RGB and HSI components of a color image. | 59 |
|  | Write a MATLAB program for smoothing and sharpening of a color image. | 62 |
|  | Write a MATLAB program for (i) Haar Transform (ii) one dimensional Discrete Wavelet Transform (iii) Discrete Cosine Transform of an image. | 66 |
|  | Write a MATLAB program for edge detection using Sobel, Canny, Prewitt, Roberts, log, zerocross filter. | 73 |
|  | Write a MATLAB program to implement a LPF (FIR) with cutoff 8KHz and to denoise audio. | 76 |
|  | Write a MATLAB program to implement Echo of audio signal. | 80 |
|  | Write a MATLAB program to record and save single and double channel audio. | 82 |
|  | Write a MATLAB program to implement Text to Speech signal. | 84 |

Problem No: 1

Problem Title: Write a MATLAB program for (i) negative (ii) Log (iii) power-law transformation of a gray level image.

(i). Negative transformation of a gray level image:

The negative of an image with intensity levels in the range [0, L-1] is obtained by using the negative transformation which is given by the expression,

s=L-1-r

Reversing the intensity levels of an image in this manner produces the equivalent of a photographic negative. This type of processing particularly suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.

Matlab source code:

clc;

clear all;

close all;

i1=imread('fruits.jpg');

i2=rgb2gray(i1);

figure(1)

subplot(121);

imshow(i2)

title('Original image')

b=255-i2;

subplot(122);

imshow(b);

title('Negative image')

Input and Output:



(ii). Log transformation of a gray level image:

The formula for Logarithmic transformation,

s = c log(r + 1)

Here, s and r are the pixel values for input and output image. And c is constant. In the formula, we can see that 1 is added to each pixel value this is because if pixel intensity is zero in the image then log(0) is infinity so, to have minimum value one is added.

When log transformation is done dark pixels are expanded as compared to higher pixel values. In log transformation higher pixels are compresses.

Matlab source code:

clc;

clear all;

close all;

im=imread('fruits.jpg');

img=rgb2gray(im);

figure(1);

subplot(1,2,1)

imshow(img)

title('Input image.');

img2=im2double(img);

c=input('Input constant c:');

trans\_img = c\*log(1+img2);

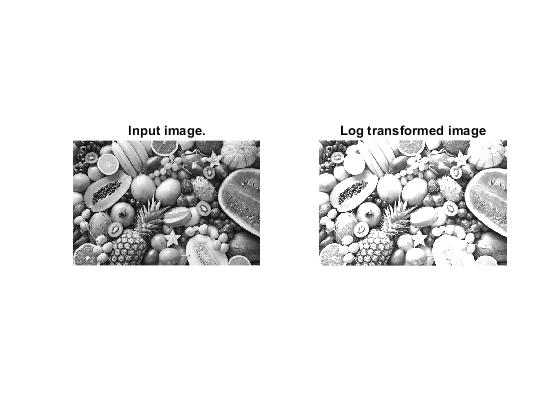
subplot(1,2,2)

imshow(trans\_img)

title('Log transformed image');

Input and Output:

Input constant c: 2



(iii). Power-law transformation of a gray level image:

Power Law Transformation is of two types of transformation nth power transformation and nth root transformation.

Formula:

s = cr ^ γ

Here, γ is gamma, by which this transformation is known as gamma transformation.

All display devices have their own gamma correction. That is why images are displayed at different intensity.

These transformations are used for enhancing images.

Matlab source code:

clc;

im=imread('fruits.jpg');

img=rgb2gray(im);

figure(1);

subplot(121);

imshow(img);

title('Input image.');

img2=im2double(img);

c=input('Input constant c:');

gamma=input('Input constant gamma:');

trans\_img = c\*(img2.^gamma);

subplot(122);

imshow(trans\_img);

title('Power-law transformation')

Input and Output:

Input constant c: 2

Input constant gamma: 5



Problem No: 2

Problem Title: Write a MATLAB program for (i) showing histogram (ii) contrast stretching (iii) histogram equalization of a gray level image.

(i). Showing histogram of a gray level image:

A histogram is a graph. A graph that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set.

A Histogram has two axis the x axis and the y axis. The x axis contains event whose frequency you have to count. The y axis contains frequency. The different heights of bar shows different frequency of occurrence of data.

Matlab source code:

clc;

clear all;

close all;

im=imread('fruits.jpg');

img=rgb2gray(im);

figure(1);

subplot(2,1,1);

imshow(img);

title('Source image')

subplot(2,1,2);

imhist(img);

title('Histogram');

Input and Output:



(ii). Contrast stretching of a gray level image:

Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by `stretching' the range of intensity values it contains to span a desired range of values, e.g. the the full range of pixel values that the image type concerned allows. It differs from the more sophisticated histogram equalization in that it can only apply a linear scaling function to the image pixel values. As a result the ‘enhancement’ is less harsh.

Matlab source code:

clc;

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(1)(top\_left).tif');

figure(1);

subplot(441);

imshow(i1);

title('Source image')

subplot(442);

imhist(i1);

title('Source image Histogram')

cs\_i1=imadjust(i1);

subplot(443);

imshow(cs\_i1);

title('Contrast stretched image');

subplot(444);

imhist(cs\_i1);

title('Contrast stretched image Histogram');

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(2)(2nd\_from\_top).tif');

subplot(445);

imshow(i1);

title('Source image')

subplot(446);

imhist(i1);

title('Source image Histogram')

cs\_i1=imadjust(i1);

subplot(447);

imshow(cs\_i1);

title('Contrast stretched image');

subplot(448);

imhist(cs\_i1);

title('Contrast stretched image Histogram');

i3=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(3)(third\_from\_top).tif');

subplot(449);

imshow(i3);

title('Source image')

subplot(4,4,10);

imhist(i3);

title('Source image Histogram')

cs\_i3=imadjust(i3);

subplot(4,4,11);

imshow(cs\_i3);

title('Contrast stretched image');

subplot(4,4,12);

imhist(cs\_i3);

title('Contrast stretched image Histogram');

i4=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(4)(bottom\_left).tif');

subplot(4,4,13);

imshow(i4);

title('Source image')

subplot(4,4,14);

imhist(i4);

title('Source image Histogram')

cs\_i4=imadjust(i4);

subplot(4,4,15);

imshow(cs\_i4);

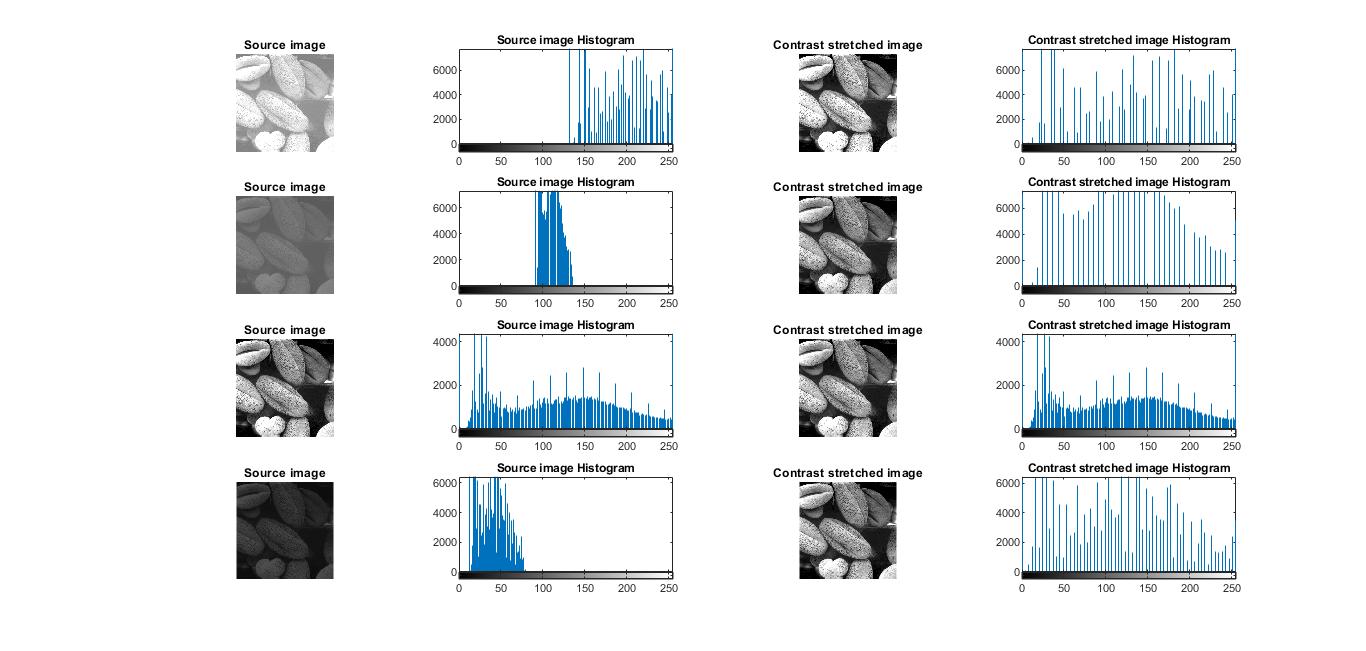
title('Contrast stretched image');

subplot(4,4,16);

imhist(cs\_i4);

title('Contrast stretched image Histogram');

Input and Output:



(iii). Histogram equalization of a gray level image:

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.

Histogram equalization is used for equalizing all the pixel values of an image. Transformation is done in such a way that uniform flattened histogram is produced. Histogram equalization increases the dynamic range of pixel values and makes an equal count of pixels at each level which produces a flat histogram with high contrast image. While stretching histogram, the shape of histogram remains the same whereas in Histogram equalization, the shape of histogram changes and it generates only one image.

Matlab source code:

clc;

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(1)(top\_left).tif');

figure(1);

subplot(441);

imshow(i1);

title('Source image')

subplot(442);

imhist(i1);

title('Source image Histogram')

cs\_i1=histeq(i1);

subplot(443);

imshow(cs\_i1);

title('Histogram equalized image');

subplot(444);

imhist(cs\_i1);

title('Histogram equalized image Histogram');

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(2)(2nd\_from\_top).tif');

subplot(445);

imshow(i1);

title('Source image')

subplot(446);

imhist(i1);

title('Source image Histogram')

cs\_i1=histeq(i1);

subplot(447);

imshow(cs\_i1);

title('Histogram equalized image');

subplot(448);

imhist(cs\_i1);

title('Histogram equalized image Histogram');

i3=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(3)(third\_from\_top).tif');

subplot(449);

imshow(i3);

title('Source image')

subplot(4,4,10);

imhist(i3);

title('Source image Histogram')

cs\_i3=histeq(i3);

subplot(4,4,11);

imshow(cs\_i3);

title('Histogram equalized image');

subplot(4,4,12);

imhist(cs\_i3);

title('Histogram equalized image Histogram');

i4=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0316(4)(bottom\_left).tif');

subplot(4,4,13);

imshow(i4);

title('Source image')

subplot(4,4,14);

imhist(i4);

title('Source image Histogram')

cs\_i4=histeq(i4);

subplot(4,4,15);

imshow(cs\_i4);

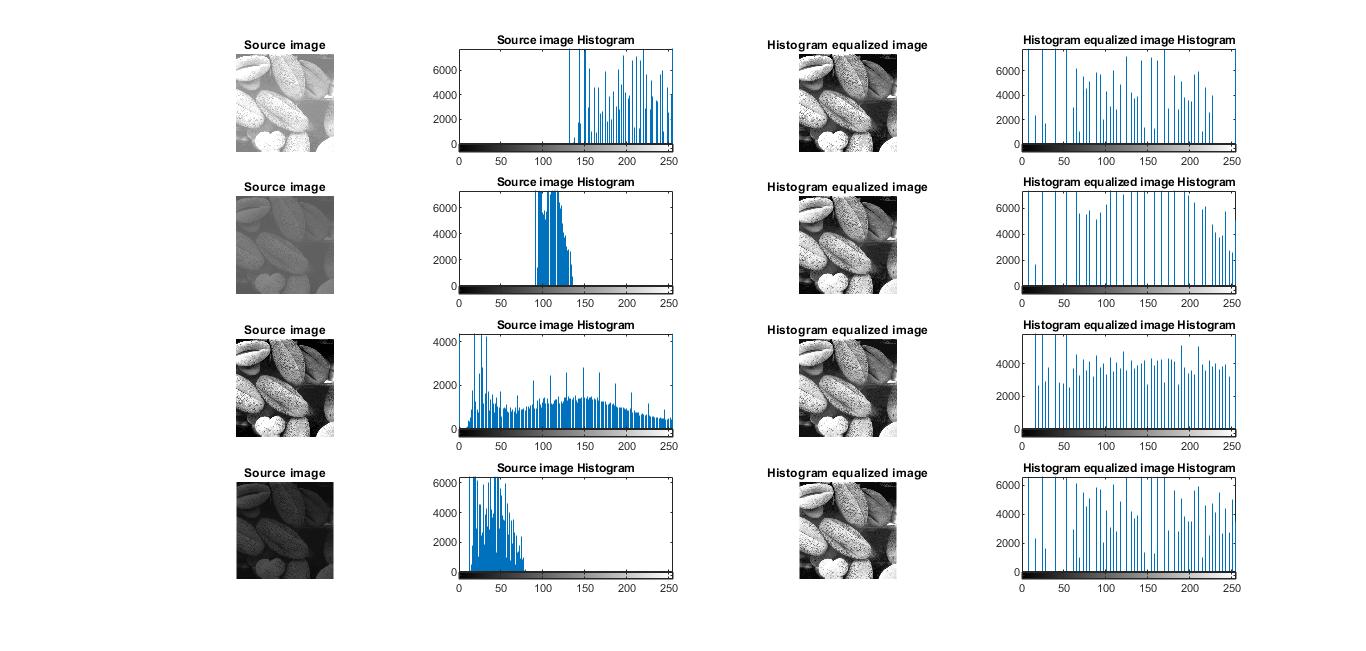
title('Histogram equalized image');

subplot(4,4,16);

imhist(cs\_i4);

title('Histogram equalized image Histogram');

Input and Output:



Problem No: 3

Problem Title: Write a MATLAB program for (i) high pass and lowpass filter (ii) average filter (iii) median, max and min filter of a gray level image.

(i). Highpass and Lowpass filter of a gray level image:

Lowpass filter (smoothing): A low pass filter is used to pass low-frequency signals. The strength of the signal is reduced and frequencies which are passed is higher than the cut-off frequency. The amount of strength reduced for each frequency depends on the design of the filter. Smoothing is low pass operation in the frequency domain.

Highpass filters (sharpening): A high pass filter is the basis for most sharpening methods. An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness. A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information.

Matlab source code:

clc;

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0333(a)(test\_pattern\_blurring\_orig).tif');

figure(1);

subplot(131);

imshow(i1);

title('Source Image');

% Lowpass filter

LowKernel = [ 1 1 1; 1 -10 1; 1 1 1 ];

lpfi = conv2(LowKernel,i1);

subplot(132)

imshow(lpfi);

title('Low pass filtered image');

% Highpass filter

HighKernel = [ -1 -1 -1; -1 10 -1; -1 -1 -1 ];

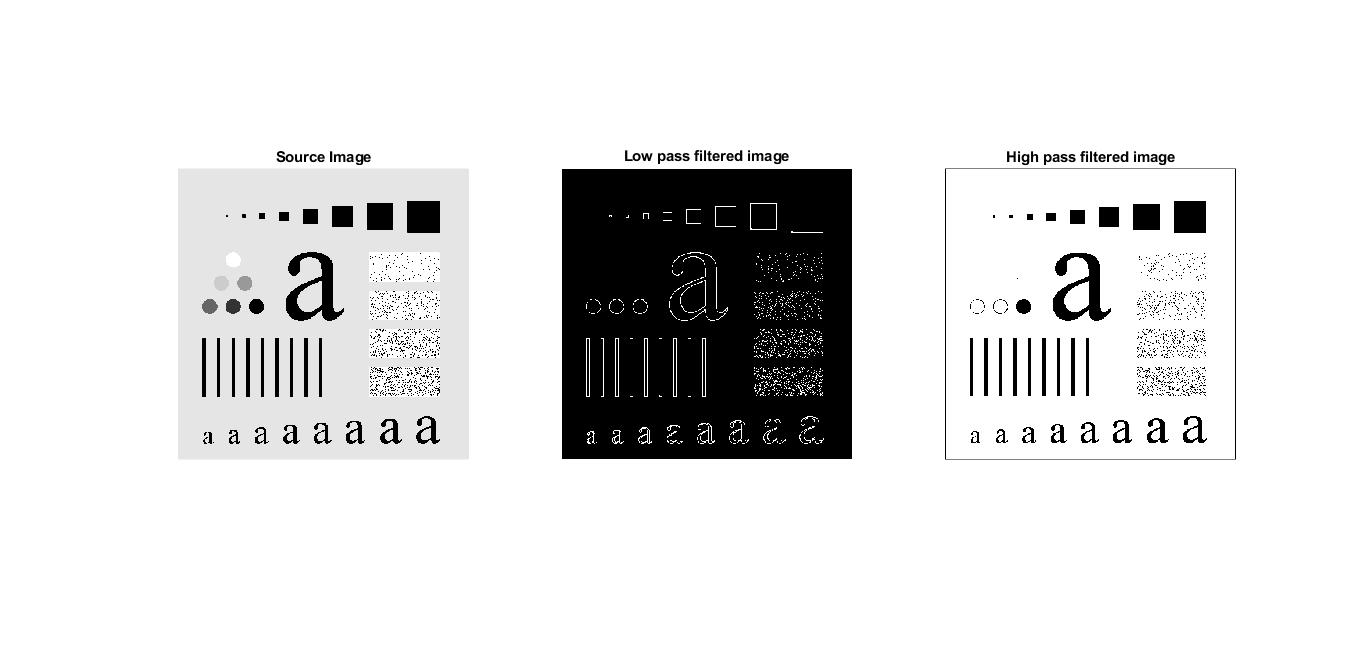
hpfi = conv2(HighKernel, i1);

subplot(133);

imshow(hpfi);

title('High pass filtered image');

Input and Output:



(ii). Average filter of a gray level image:

Average (or mean) filtering is a method of ‘smoothing’ images by

reducing the amount of intensity variation between neighbouring

pixels.

The average filter works by moving through the image pixel by pixel,

replacing each value with the average value of neighbouring pixels,

including itself.

Matlab source code:

clc;

close all;

clear all;

i1=imread('C:\Users\Shihab\Downloads\Compressed\DIP3E\_CH03\_Original\_Images\Fig0333(a)(test\_pattern\_blurring\_orig).tif');

figure(1);

subplot(231);

imshow(i1);

title('Source Image');

M3=fspecial('average',3);

M9=fspecial('average',9);

M15=fspecial('average',15);

M25=fspecial('average',25);

M35=fspecial('average',35);

J3=imfilter(i1,M3);

J9=imfilter(i1,M9);

J15=imfilter(i1,M15);

J25=imfilter(i1,M25);

J35=imfilter(i1,M35);

subplot (232);

imshow(J3);

title('Filtered by 3X3');

subplot (233);

imshow(J9);

title('Filtered by 9X9');

subplot (234);

imshow(J15);

title('Filtered by 15X15');

subplot (235);

imshow(J25);

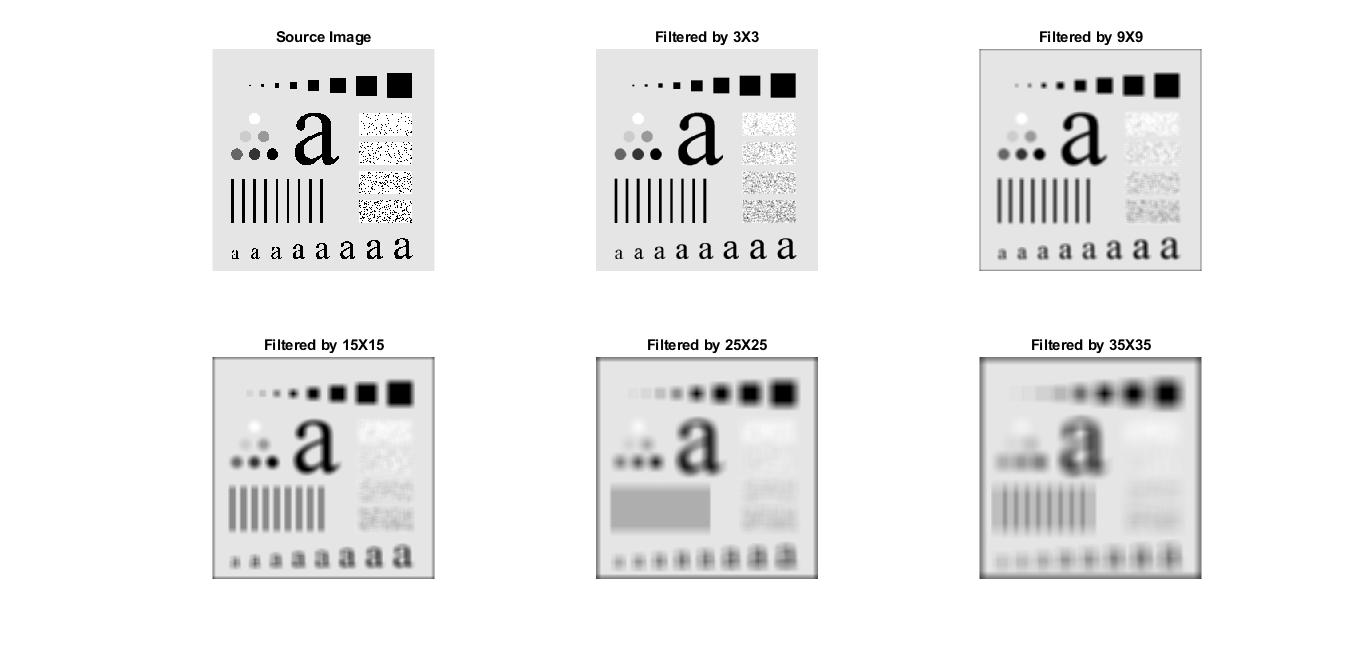
title('Filtered by 25X25');

subplot (236);

imshow(J35);

title('Filtered by 35X35');

Input and Output:



(iii). Median, max and min filter filter of a gray level image:

Median filter: Median filtering is a nonlinear method used to remove noise from images. It is widely used as it is very effective at removing noise while preserving edges. It is particularly effective at removing ‘salt and pepper’ type noise. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. The pattern of neighbours is called the "window", which slides, pixel by pixel over the entire image 2 pixel, over the entire image.

The median is calculated by first sorting all the pixel values from the window into numerical order, and then replacing the pixel being considered with the middle (median) pixel value.

Max Filter: The maximum filter is defined as the maximum of all pixels within a local region of an image. The maximum filter is typically applied to an image to remove negative outlier noise.

Min Filter: The minimum filter is defined as the minimum of all pixels within a local region of an image. The minimum filter is typically applied to an image to remove positive outlier noise.

Matlab source code:

clc;

clear all;

close all;

% Median Filter

img=imread('cameraman.tif');

[r,c]=size(img);

img=im2double(img);

subplot(231);imshow(img);title('Source image');

%%% Adding Salt & Pepper Noise

noisy\_img=imnoise(img,'salt & pepper');

subplot(232);imshow(noisy\_img);title('Salt & Pepper noisy image');

%%% Filtering , window 3\*3

mf\_img=ordfilt2(noisy\_img,5,ones(3,3));

subplot(233);imshow(mf\_img);title('Median filtered image');

% Max & Min Filter

subplot(234);

imshow(img);

title('Source image');

maxf\_img=ordfilt2(noisy\_img,9,ones(3,3));

subplot(235);imshow(maxf\_img);

title('Max filtered image');

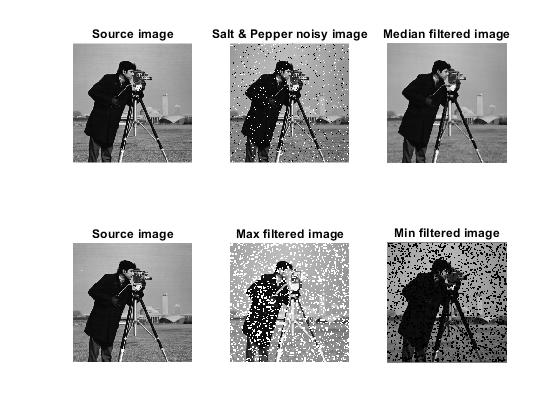
minf\_img=ordfilt2(noisy\_img,1,ones(3,3));

subplot(236);

imshow(minf\_img);

title('Min filtered image');

Input and Output:



Problem No: 4

Problem Title: Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian lowpass and highpass filter of a gray level image in frequency domain.

(i). Ideal lowpass and highpass filter of a gray level image in frequency domain:

Ideal low pass Filters: A 2-D lowpass filter that passed without attenuation all frequencies within a circle of radius D0 from the origin and ‘cuts off’ all frequencies outside this circle is called an ideal low pass filter. It is specified by the function,

H(u,v)=1 if D(u,v)≤D0

H(u,v)=0 if D(u,v)>D0

Where D0 is a positive constant and D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle.

Ideal high pass filters: Ideal high pass filter is used for image sharpening in the frequency domain. Image sharpening is a technique to enhance the fine details and highlight the edges in a digital image. It removes low frequency components from and image and preserves hight frequency components. It can be determined using the following relation,

HHP(u, v) = 1 - HLP(u, v)

Where HHP(u,v) is the transfer function of the highpass filter and HLP(u,v) is the transfer function of the corresponding low pass filter.

H(u,v)=0 if D(u,v)≤D0

H(u,v)=1 if D(u,v)>D0

Where D0 is a positive constant, D(u,v) is the Euclidean distance from any point (u,v) to the origin of the frequency plane.

Matlab source code:

clc;

clear all;

close all;

% Ideal Lowpass Filter(ILPF)

img=imread('cameraman.tif');

[r,c]=size(img);

subplot(231);

imshow(img);

title('Source image');

IMG=fftshift(fft2(img));

%%%Creating filter

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

D=sqrt(u.^2+v.^2);

D0=40;

ILPF=(D<=D0);

subplot(232);

mesh(double(ILPF));

title('ILPF');

ILPF\_IMG=IMG.\*ILPF;

ilpf\_img=ifft2(ILPF\_IMG);

subplot(233);

imshow(mat2gray(abs(ilpf\_img)));

title('ILPF filtered image');

% Ideal Highpass Filter(IHPF)

subplot(234);

imshow(img);

title('Source image');

D0=30;

IHPF=(D>D0);

subplot(235);

mesh(double(IHPF));

title('IHPF');

IHPF\_IMG=IMG.\*IHPF;

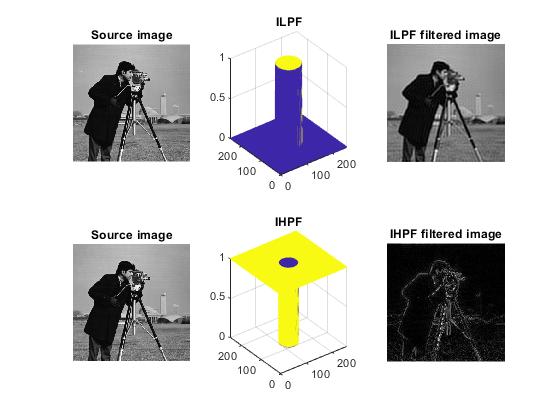
ihpf\_img=ifft2(IHPF\_IMG);

subplot(236);

imshow(mat2gray(abs(ihpf\_img)));

title('IHPF filtered image');

Input and Output:



(ii). Butterworth lowpass and highpass filter of a gray level image in frequency domain:

(ii) Butterworth low pass filters: Butterworth Lowpass Filter (BLPF) is used for image smoothing in the frequency domain.

The transfer function of a butterworth low pass filter of order n, and with cutoff frequency at a distance D0 from the origin is defined as,

H(u, v)= 1/[1+[D(u,v)/ D0]2n]

Butterworth high pass filters: Butterworth Highpass Filter (BHPF) is used for image sharpening in the frequency domain. It can be determined using the relation-

HHP(u,v)=1-HLP(u,v)

where, HHP(u,v) is the transfer function of the highpass filter and HLP(u,v) is the transfer function of the corresponding lowpass filter.

A 2-D Butterworth highpass filter order n and cutoff frequency D0 is defined as,

H(u, v)= 1/[1+[D0/D(u,v)]2n]

Where D0 a positive constant and is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as cutoff frequency and D(u,v) is the euclidean distance between any point (u,v) to the origin of the frequency plane.

Matlab source code:

clc;

clear all;

close all;

% Butterworth Lowpass Filter(BLPF)

img=imread('cameraman.tif');

[r,c]=size(img);

subplot(231), imshow(img), title('Source image');

IMG=fftshift(fft2(img));

%%%Creating filter

[u, v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

D=sqrt(u.^2+v.^2);

D0=15;

n=1;

BLPF = 1./( 1.+ (D./D0).^(2\*n) );

subplot(232), mesh(BLPF), title('BLPF')

BLPF\_IMG=IMG.\*BLPF;

blpf\_img=ifft2(BLPF\_IMG);

subplot(233), imshow(mat2gray(abs(blpf\_img))), title('BLPF filtered image')

% Butterworth Highpass Filter(BHPF)

subplot(234), imshow(img), title('Source image')

D0=15;

n=1;

BHPF=1./( 1.+ (D0./D).^(2\*n) );

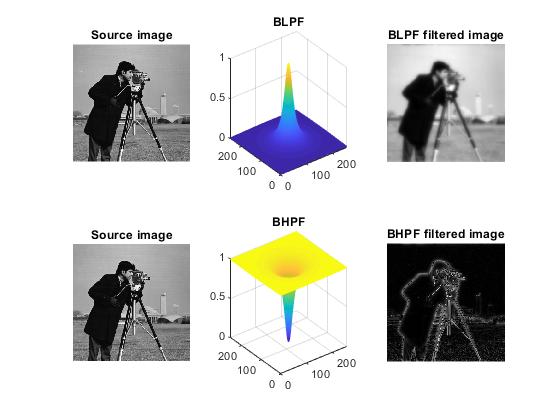
subplot(235), mesh(BHPF), title('BHPF')

BHPF\_IMG=IMG.\*BHPF;

bhpf\_img=ifft2(BHPF\_IMG);

subplot(236), imshow(mat2gray(abs(bhpf\_img))), title('BHPF filtered image')

Input and Output:



(iii). Gaussian lowpass and highpass filter of a gray level image in frequency domain:

Gaussian low pass filters: Gaussian lowpass filter can determinde by the following expression:

H(u,v)=e^(-D^2 (u,v)/2σ^2 )

Where D(u,v) is the distance from the center of the frequency rectangle, σ is a measure of spread about the center. By letting σ=D0 we can express the filter as follows,

H(u,v)=e^(-D^2 (u,v)/2D\_0^2 )

Wher D0 is the cutoff frequency.

Gaussian high pass filters: the transfer function of the gaussian highpass filter with cutoff frequency locus at a distance D0 from the center of the frequency rectangle is given by,

H(u,v)=1-e^(-D^2 (u,v)/2D\_0^2 )

Where D0 a positive constant and is the transition point between H(u, v) = 1 and H(u, v) = 0, so this is termed as cutoff frequency and D(u,v) is the euclidean distance between any point (u,v) to the origin of the frequency plane.

Matlab source code:

clc;

clear all;

close all

% Gaussian Lowpass Filter(GLPF)

img=imread('cameraman.tif');

[r,c]=size(img);

subplot(231), imshow(img), title('Source image')

IMG=fftshift(fft2(img));

%%%Creating filter

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

D=sqrt(u.^2+v.^2);

D0=10;

GLPF = exp (-(D.^2)./(2\*D0^2)) ;

subplot(232), mesh(GLPF), title('GLPF')

GLPF\_IMG=IMG.\*GLPF;

glpf\_img=ifft2(GLPF\_IMG);

subplot(233), imshow(mat2gray(abs(glpf\_img))), title('GLPF filtered image')

% Gaussian Highpass Filter(GHPF)

subplot(234), imshow(img), title('Source image')

D0=10;

GHPF=1 - exp (-(D.^2)./(2\*D0^2)) ;

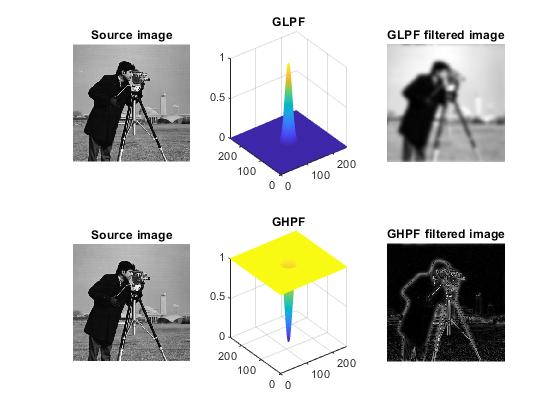
subplot(235), mesh(GHPF), title('GHPF')

GHPF\_IMG=IMG.\*GHPF;

ghpf\_img=ifft2(GHPF\_IMG);

subplot(236), imshow(mat2gray(abs(ghpf\_img))), title('GHPF filtered image')

Input and Output:



Problem No: 5

Problem Title: Write a MATLAB program for (i) Laplacian (ii) homomorphic filter of a gray level image in frequency domain.

(i). Laplacian filter of a gray level image:

A laplacian filter is an edge detector used to compute the second derivatives of an image, measuring the rate at which the first derivatives change. Laplacian can be implemented in the frequency domain using the filter,

H(u,v)= -4π2(u2+v2)

or, with respect to the center of the frequency rectangle, using the filter,

H(u, v)= -4π2[(u-P/2)2+(v-Q/2)2]

=-4π2D2(u, v)

Where D(u, v) is the distance function.

Matlab source code:

clc;

clear all;

close all;

% Laplacian Filter(LF)

img=imread('trees.tif');

[r,c]=size(img);

subplot(131), imshow(img), title('Source image')

IMG=fftshift(fft2(img));

%Dispaly Fourier Transformed Image

IMG1=log(1+abs(IMG));

m=max(IMG1(:));

figure(2), imshow(im2uint8(IMG1/m)), title('Fourier Transformed Image');

%%%Creating filter

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

LF = -4\*pi^2\*(u.^2+v.^2) ;

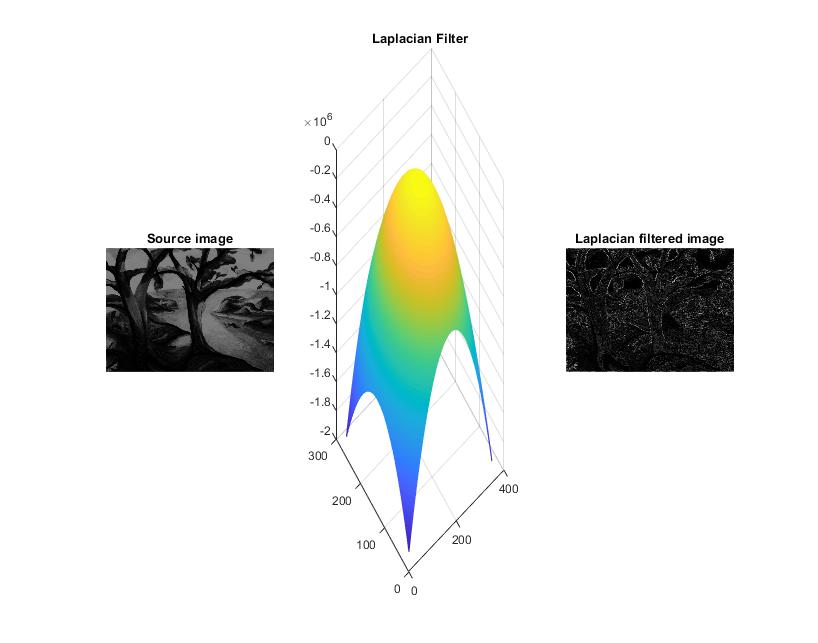
figure(1), subplot(132), mesh(LF), title('Laplacian Filter')

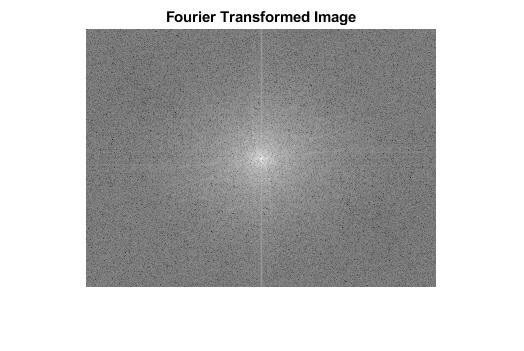
LF\_IMG=IMG.\*LF;

lf\_img=ifft2(LF\_IMG);

subplot(133), imshow(mat2gray(abs(lf\_img))), title('Laplacian filtered image')

Input and Output:





(ii). Homomorphic filter of a gray level image:

Hamomorphic filter is a generalized technique for signal and image processing, involving a nonlinear mapping to a different domain in which linear filter techniques are applied, followed by mapping back to the original domain. It is sometimes used for image enhancement. It is one such technique for removing multiplicative noise that has certain characteristics. It is most commonly used for correcting non-uniform illumination in images. It can be used for improving the appearance of the grayscale image by simultaneous intensity range compression & contrast enhancement.

An image f(x, y) can be represented by its incident light component and reflected light component, and the relationship is as follows

f(x, y) = i(x, y) r (x, y)

The image f(x, y) is produced by the combined action of the illumination field i(x, y) generated by the light source and the reflection coefficient field r(x, y) of the target.

Matlab source code:

clc;

clear all;

close all;

% Homomorphic Filter(HMF)

img=imread('trees.tif');

[r,c]=size(img);

subplot(131), imshow(img), title('Source image')

IMG=fftshift(fft2(img));

%Dispaly Fourier Transformed Image

IMG1=log(1+abs(IMG));

m=max(IMG1(:));

figure(2), imshow(im2uint8(IMG1/m)), title('Fourier Transformed Image');

img=im2double(img);

IMG=fft2(log(img+0.01));

%%%Creating Butterworth High Pass Filter for HMF

%%%Creating filter

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

D=sqrt(u.^2+v.^2);

D0=15;

n=1;

BHPF = 1./( 1.+ (D0./D).^(2\*n) );

figure(1), subplot(132), mesh(BHPF), title('BHPF for HMF')

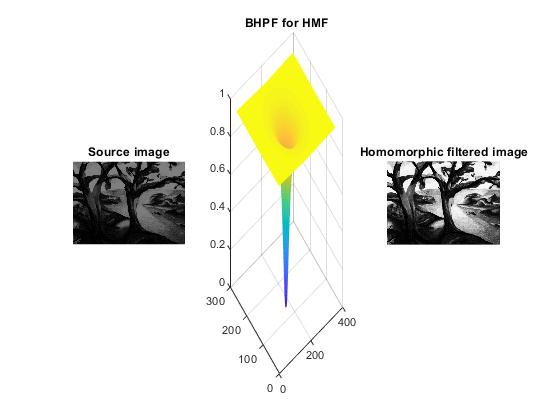
BHPF\_IMG=IMG.\*BHPF;

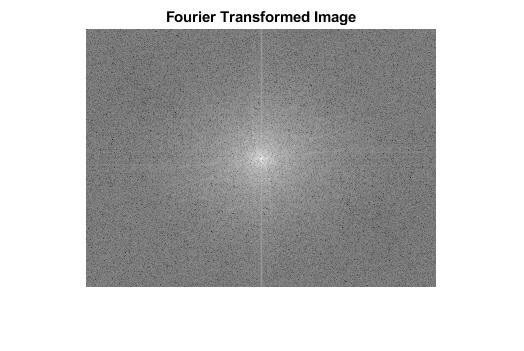
real\_img=real(ifft2(BHPF\_IMG));

exp\_img=exp(real\_img);

subplot(133), imshow(mat2gray(abs(exp\_img))), title('Homomorphic filtered image')

Input and Output:





Problem No: 6

Problem Title: Write a MATLAB program for (i) arithmetic and geometric (ii) harmonic and contraharmonic (iii) midpoint and alpha-trimmed mean filter of a gray level image.

(i). Arithmetic and geometric filter of a gray level image:

Arithmetic Mean filter: This is the simplest of the mean filters. Let Sxy represent the set of coordinates in a rectangular subimage window of size m\*n, centered at point (x,y). The arithmetic mean filter computes the average value of corrupted image g(x,y) in the area defined by Sxy. The value of the restored image f at point (x,y) is simply the arithmetic mean computed using the pixels in the region defined by Sxy.

f’(x,y)=

Geometric Mean filter: An image restored using a geometric mean filter is given by the expression,

f’(x,y)=

Here, each restored pixel is given by the product of the pixels in the sub-image window, raised to the power 1/mn. A geometric mean filter achieves smoothing comparable to the arithmetic mean filter, but it tends to lose less image detail in the process.

Matlab source code:

clc;

clear all;

close all;

% Arithmetic Mean Filter and Geometric Mean Filter

img=imread('cameraman.tif');

[r,c]=size(img);

img=im2double(img);

subplot(221);imshow(img);title('Source image');

%%%Adding Gaussian Noise

noisy\_img=imnoise(img,'gaussian');

subplot(222);imshow(noisy\_img);title('Gaussian noisy image');

%%% 'valid' convolution(3\*3) , so image dimension will be reduced

for i=1:r-2

for j=1:c-2

window = noisy\_img(i:i+2,j:j+2);

amf\_img(i,j)= mean( window(:) );

gmf\_img(i,j)= geomean( window(:) );

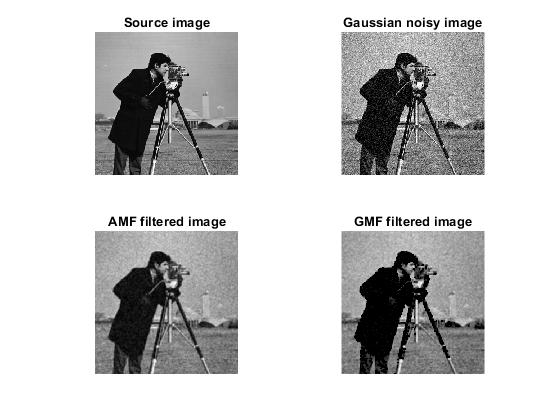
end

end

subplot(223);imshow(amf\_img);title('AMF filtered image');

subplot(224);imshow(gmf\_img);title('GMF filtered image');

Input and Output:



(ii). Harmonic and contraharmonic filter of a gray level image:

Harmonic Filter: The harmonic mean filtering operation is given by the expression

f’(x,y) =

The harmonic mean filter works well for noise, but fails for pepper noise. It does well also with other types of noise like Gaussian noise.

Contraharmonic Filter: The contraharmonic mean filter yields a restored image based on the expression

f’(x,y)=

Where Q is called the order of the filter. This filter is well suited for reducing or virtually eliminating the effects of salt-and-pepper noise. For positive values of Q, the filter eliminates pepper noise. For negative values of Q it eliminated the salt noise. It cannot do both simultaneously. The contraharmonic filter reduces to the arithmetic mean filter if Q=0 and to the harmonic mean filter if Q = -1.

Matlab source code:

clc;

clear all;

close all;

% Harmonic Mean Filter and Conrtaharmonic Mean Filter

img=imread('cameraman.tif');

[r,c]=size(img);

img=im2double(img);

subplot(221);

imshow(img);

title('Source image');

%%%Adding Gaussian Noise

noisy\_img=imnoise(img,'gaussian');

subplot(222);

imshow(noisy\_img);

title('Gaussian noisy image');

%%% 'valid' convolution(3\*3) , so image dimension will be reduced

Q=1.5;

for i=1:r-2

for j=1:c-2

window = noisy\_img(i:i+2,j:j+2);

hmf\_img(i,j)= harmmean( window(:) );

chmf\_img(i,j)= sum( window(:).^(Q+1) ) ./ sum( window(:).^Q );

end

end

subplot(223);

imshow(hmf\_img);

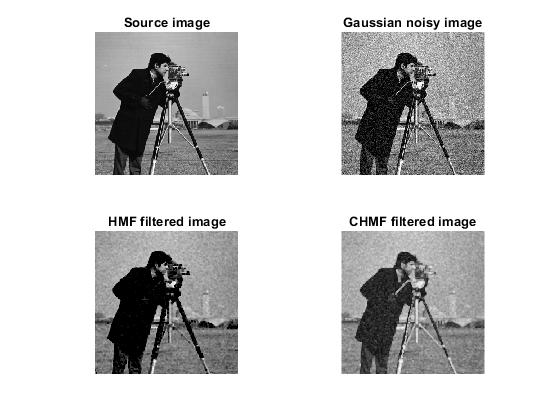
title('HMF filtered image');

subplot(224);

imshow(chmf\_img);

title('CHMF filtered image');

Input and Output:



(iii). Midpoint and alpha-trimmed mean filter of a gray level image:

**Midpoint filter:** the midpoint filter simply computes the midpoint between the maximum and minimum value in the area encompassed by the filter:

f’(x,y)=

This filter combined order statistic and averaging. It works best for randomly distributed noise, like Gaussian or uniform noise.

**Alpha-trimmed filter**: Suppose that we denote the d/2 lowest and the d/2 highest intensity values of g(s,t) in the neighborhood Sxy. Let gr(s,t) represent the remaining mn-d pixels. A filter formed by averaging these remaining pixels is called an alpha-trimmed mean filter.

f’(x,y)=

Where, the value of d can range from 0 to mn-1, when d=0 the alpha trimmed filter reduces to the arithmetic mean filter. If we choose d=mn-1 the filter becomes a median filter. For other values of d the alpha trimmed filter is useful in situations involving multiple types of noise such as combination of salt and pepper and Gaussian noise.

Matlab source code:

clc;

clear all;

close all;

% Midpoint Filter ( MF )

img=imread('cameraman.tif');

[r,c]=size(img);

img=im2double(img);

subplot(221);imshow(img);title('Source image');

%%% Adding Gaussian Noise

noisy\_img=imnoise(img,'Gaussian');

subplot(222);imshow(noisy\_img);title('Gaussian noisy image');

%%% Filtering , window 3\*3

midf\_img= ( ordfilt2(noisy\_img,9,ones(3,3)) + ordfilt2(noisy\_img,1,ones(3,3)) )./2;

subplot(223);imshow(midf\_img);title('Midpoint filtered image');

% Alpha-trimmed Mean Filter ( ATMF )

%%% 'valid' convolution(3\*3) , so image dimension will be reduced

d=25; %percent

for i=1:r-2

for j=1:c-2

window = noisy\_img(i:i+2,j:j+2);

atmf\_img(i,j)= trimmean( window(:),d );

end

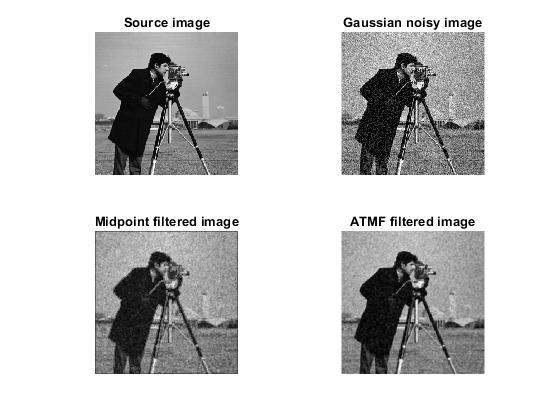
end

subplot(224);

imshow(atmf\_img);

title('ATMF filtered image');

Input and Output:



Problem No: 7

Problem Title: Write a MATLAB program for (i) ideal (ii) Butterworth (iii) Gaussian bandreject and bandpass filter of a gray level image.

(i). Ideal bandreject and bandpass filter of a gray level image:

**Ideal bandreject filter:**  In this type of filter, frequencies outside of the given range are passed without attenuation and frequencies inside of the given range are blocked. This behavior makes ideal band reject filters very sharp.

The centered Fast Fourier Transform (FFT) is filtered by the following function, where DL is the lower bound of the frequency band, DH is the upper bound of the frequency band, and D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle:

***H(u,v)={ 0 if DL≤D(u,v) ≤DH otherwise H(u,v)=1***

**Ideal bandpass filter:** In ideal bandpass filters, frequencies within the given range are passed through without attenuation and frequencies outside of the given range are completely removed. This behavior makes ideal bandpass filters very sharp.

The centered Fast Fourier Transform (FFT) is filtered by the following function, where DL is the lower bound of the frequency band, DH is the upper bound of the frequency band, and D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle:

*H(u,v)={ 1 if DL≤D(u,v) ≤DH otherweise H(u,v)=0*

Matlab source code:

clc;

clear all;

close all;

% Ideal Bandreject Filter(IBRF)

img=imread('cameraman.tif');

[r,c]=size(img);

figure(1);

subplot(332)

imshow(img);

title('Source image');

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

%%%Adding Noise

sin\_noise= 15\*sin( 2\*pi\*1/10\*u + 2\*pi\*1/10\*v);

noisy\_img=double(img)+sin\_noise;

NOISY\_IMG=fftshift(fft2(noisy\_img));

subplot(334);

imshow(noisy\_img,[]);

title('Sinusoidal noisy image')

subplot(337);

imshow(mat2gray(log(1+abs(NOISY\_IMG))));

title('FFT of noisy image');

%%%Creating filter

D=sqrt(u.^2+v.^2);

D0=50;

W=40;

IBRF= ( D<(D0-W/2) | D>(D0+W/2) );

subplot(335);mesh(IBRF);title('IBRF')

IBRF\_IMG=NOISY\_IMG.\*IBRF;

ibrf\_img=ifft2(IBRF\_IMG);

subplot(336);

imshow(mat2gray(abs(ibrf\_img)));

title('IBRF filtered image');

% Ideal Bandpass Filter(IBPF)

IBPF= 1 - IBRF ;

subplot(338);

mesh(IBPF);

title('IBPF')

IBPF\_IMG=NOISY\_IMG.\*IBPF;

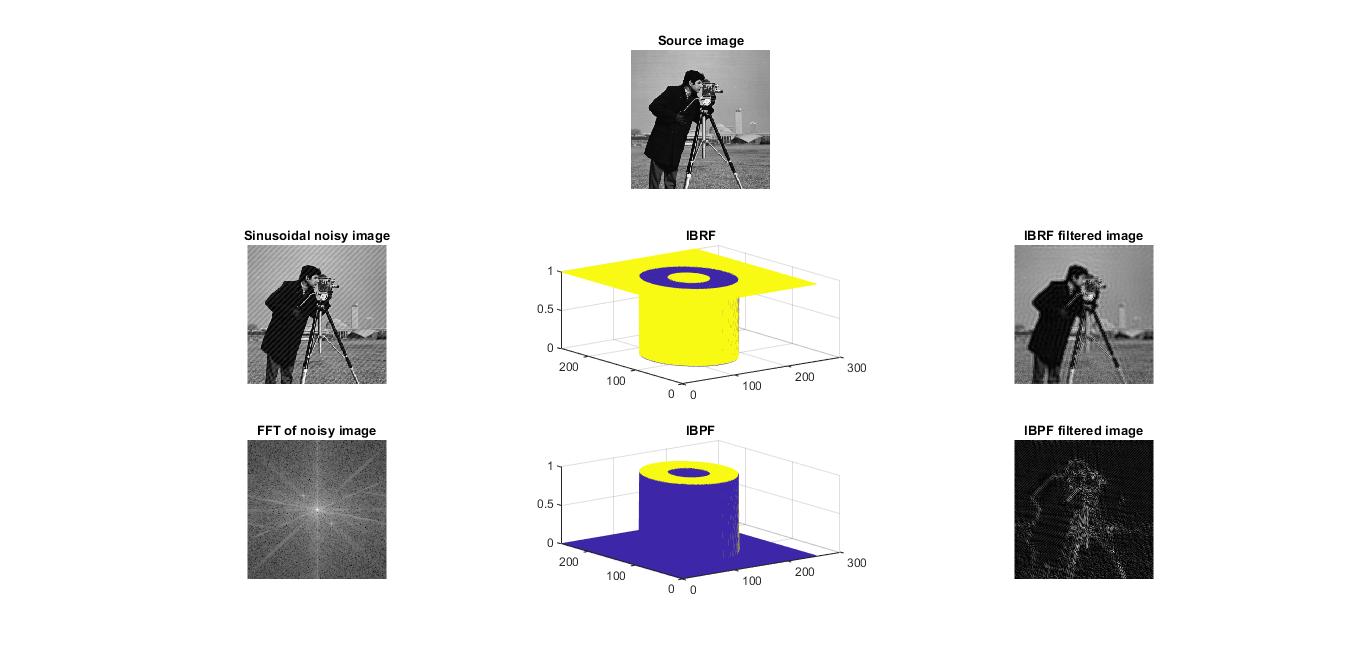
ibpf\_img=ifft2(IBPF\_IMG);

subplot(339);

imshow(mat2gray(abs(ibpf\_img)));

title('IBPF filtered image');

Input and Output:



(ii). Butterworth bandreject and bandpass filter of a gray level image:

**Butterworth bandreject filter:** With a Butterworth band reject filter, frequencies at the center of the frequency band are completely blocked and frequencies at the edge of the band are attenuated by a fraction of the maximum value. The Butterworth filter does not have any sharp discontinuities between passed and filtered frequencies.

The centered FFT is filtered by the following function, where D0 is the center of the frequency band, W is the width of the frequency band, D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle, and n is the dimension of the Butterworth filter:

H(u,v)=1/[1+ {D(u,v)W/(D(u,v)2 – D02)}2n

**Butterworth bandpass filter:** With a Butterworth bandpass filter, frequencies at the center of the frequency band are unattenuated and frequencies at the edge of the band are attenuated by a fraction of the maximum value. The Butterworth filter does not have sharp discontinuities between frequencies that are passed and filtered.

The default for BANDPASS\_FILTER is BUTTERWORTH=1.

The centered FFT is filtered by one of the following functions, where D0 is the center of the frequency band, W is the width of the frequency band, D=D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle, and n is the dimension of the Butterworth filter:

*H(u,v)=1-[1/1+{DW/(D2-D02)}2n]*

Matlab source code:

clc;

clear all;

close all;

% Butterworth Bandreject Filter(BBRF)

img=imread('cameraman.tif');

[r,c]=size(img);

imshow(img);title('Source image');

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

%%%Adding Noise

sin\_noise= 15\*sin( 2\*pi\*1/10\*u + 2\*pi\*1/10\*v);

noisy\_img=double(img)+sin\_noise;

NOISY\_IMG=fftshift(fft2(noisy\_img));

figure(2)

subplot(231);

imshow(noisy\_img,[]);

title('Sinusoidal noisy image');

subplot(234);

imshow(mat2gray(log(1+abs(NOISY\_IMG))));

title('FFT of noisy image');

%%%Creating filter

D=sqrt(u.^2+v.^2);

D0=50;

n=1;

W=20;

BBRF=1./( 1.+ ( (D.\*W) ./ (D.^2-D0.^2) ) .^(2\*n) );

subplot(232);mesh(BBRF);title('BBRF')

BBRF\_IMG=NOISY\_IMG.\*BBRF;

bbrf\_img=ifft2(BBRF\_IMG);

subplot(233);

imshow(mat2gray(abs(bbrf\_img)));

title('BBRF filtered image')

% Butterworth Bandpass Filter(BBPF)

BBPF= 1 - BBRF;

subplot(235);

mesh(BBPF);

title('BBPF')

BBPF\_IMG=NOISY\_IMG.\*BBPF;

bbpf\_img=ifft2(BBPF\_IMG);

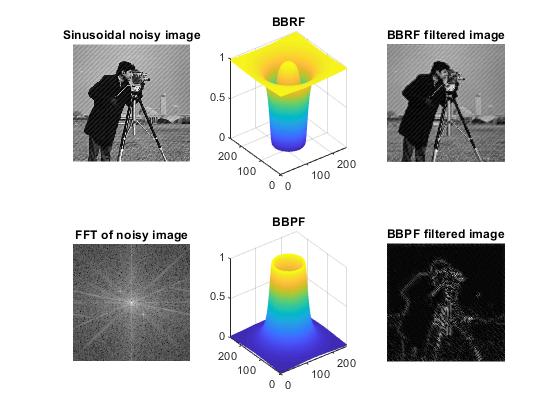
subplot(236);

imshow(mat2gray(abs(bbpf\_img)));

title('BBPF filtered image')

Input and Output:





(iii). Gaussian bandreject and bandpass filter of a gray level image:

**Gaussian bandreject filter:**  In this type of filter, the transition between unfiltered and filtered frequencies is very smooth.

The centered FFT is filtered by the following function, where D0 is the center of the frequency band, W is the width of the frequency band, and D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle:

*H(u,v)=1-e^[{/(D(u,v)2 – D02)/ D(u,v)W }2*]

**Gaussian bandpass filter:** In this type of filter, the transition between unfiltered and filtered frequencies is very smooth.

The centered FFT is filtered by one of the following functions, where D0 is the center of the frequency band, W is the width of the frequency band, and D=D(u,v) is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle.

*H(u,v)=e^[-(D2-D02)/(DW)]2*

Matlab source code:

clc;

clear all;

close all;

% Gaussian Bandreject Filter(GBRF)

img=imread('cameraman.tif');

[r,c]=size(img);

imshow(img);

title('Source image')

[u,v]=meshgrid(-floor(c/2):floor((c-1)/2),-floor(r/2):floor((r-1)/2));

%%%Adding Noise

sin\_noise= 15\*sin( 2\*pi\*1/10\*u + 2\*pi\*1/10\*v);

noisy\_img=double(img)+sin\_noise;

NOISY\_IMG=fftshift(fft2(noisy\_img));

figure(2)

subplot(231);

imshow(noisy\_img,[]);

title('Sinusoidal noisy image');

subplot(234);

imshow(mat2gray(log(1+abs(NOISY\_IMG))));

title('FFT of noisy image');

%%%Creating filter

D=sqrt(u.^2+v.^2);

D0=50;

W=20;

GBRF= 1 - exp ( -(1/2).\* ( ((D.^2)-(D0.^2)) ./ (D.\*W) ).^2 ) ;

subplot(232);mesh(GBRF);title('GBRF');

GBRF\_IMG=NOISY\_IMG.\*GBRF;

gbrf\_img=ifft2(GBRF\_IMG);

subplot(233);

imshow(mat2gray(abs(gbrf\_img)));

title('GBRF filtered image');

% Gaussian Bandpass Filter(GBPF)

GBPF=1 - GBRF;

subplot(235);

mesh(GBPF);

title('GBPF');

GBPF\_IMG=NOISY\_IMG.\*GBPF;

gbpf\_img=ifft2(GBPF\_IMG);

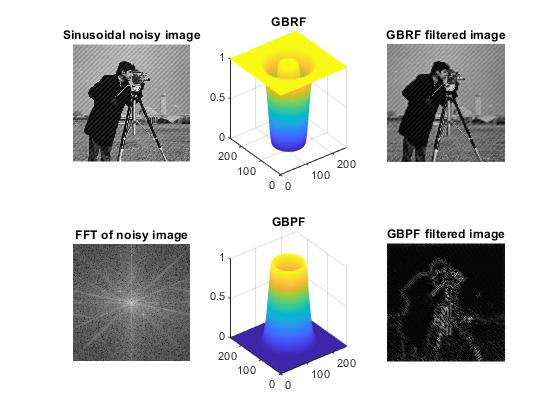
subplot(236);

imshow(mat2gray(abs(gbpf\_img)));

title('GBPF filtered image');

Input and Output:





Problem No: 8

Problem Title: Write a MATLAB program for Wiener filter of a gray level image.

There is an approach that incorporates both the degradation function and statistical characteristics of noise into the restoration process. The method is founded on considering images and noise as random variable, and the objective is to find an estimate f’ of the uncorrupted image f such that the mean square error between them is minimized. This error is measure is given by,

e2=E{(f-f’)2}

It is assumed that the noise and the image are uncorrelated, that one or the other has zero mean, and that the intensity levels in the estimate are a linear function of the levels in the degraded image.

F’(u,v)=

Where we used the fact that the product of a complex quantity with its conjugates is equal to the magnitude of the complex quantity squared. This result is know as the Wiener filter.

Matlab source code:

clc;

clear all;

close all;

% Minimum Mean Square Error Filter( Wiener Filter )

img=imread('cameraman.tif');

subplot(131)

imshow(img)

title('Source image')

%%%Adding Gaussian Noise

noisy\_img=imnoise(img,'gaussian');

subplot(132);

imshow(noisy\_img);

title('Gaussian noisy image');

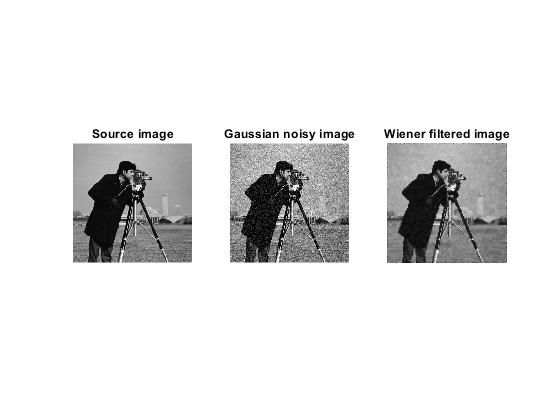
wiener\_img=wiener2(noisy\_img,[5 5]);

subplot(133)

imshow(wiener\_img);

title('Wiener filtered image')

Input and Output:



Problem No: 9

Problem Title: Write a MATLAB program for separating RGB and HSI components of a color image.

In the RGB color model, each color appears in its primary spectral components of red, green and blue. This model is based on Cartesian coordinate system. In RGB model, the gray scale extends from black to white along the line joining these two points. Images represented in RGB color model consist of three component images, one for each primary color. When fed into RGB monitor, these tree images combine on the screen to produce a composite color image. As RGB color image can be viewed as three monochorme intensity images, so we should be able to extract intensity from an RGB image.

HSI, H stands for Hue, S for Saturation and I for Intensity. HSI unlike RGB, separate the intensity (luma) from color information (chroma). For example, face detection is usually done on intensity images. On the other hand, ignoring the intensity can help to get rid of shadows. HSI contains hue and saturation, which are the terms that people use to describe colors. On the other hand, hue and saturation are angles, which can be inconvenient for computing distances in the color space.

Matlab source code:

clc;

clear all;c

lose all;

rgb\_img=imread('lenna.png');

rgb\_img=im2double(rgb\_img);

figure(1)

subplot(221);

imshow(rgb\_img);

title('Original RGB image');

% Separating RGB components

R=rgb\_img; R(:,:,2)=0; R(:,:,3)=0;

G=rgb\_img; G(:,:,1)=0; G(:,:,3)=0;

B=rgb\_img; B(:,:,1)=0; B(:,:,2)=0;

subplot(222);

imshow(R);

title('Red component');

subplot(223);

imshow(G);

title('Green component');

subplot(224);

imshow(B);

title('Blue component');

figure(2)

subplot(221);

imshow(rgb\_img);

title('RGB image');

hsi\_img=rgb2hsv(rgb\_img);

% Separating HSI components

H=hsi\_img; H(:,:,2)=0; H(:,:,3)=0;

S=hsi\_img; S(:,:,1)=0; S(:,:,3)=0;

I=hsi\_img; I(:,:,1)=0; I(:,:,2)=0;

subplot(222);

imshow(H);

title('Hue component');

subplot(223);

imshow(S);

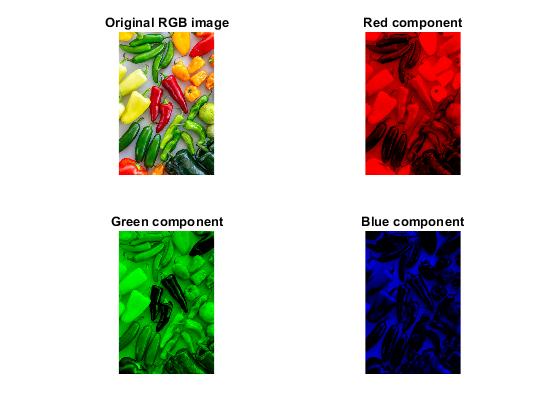
title('Saturation component');

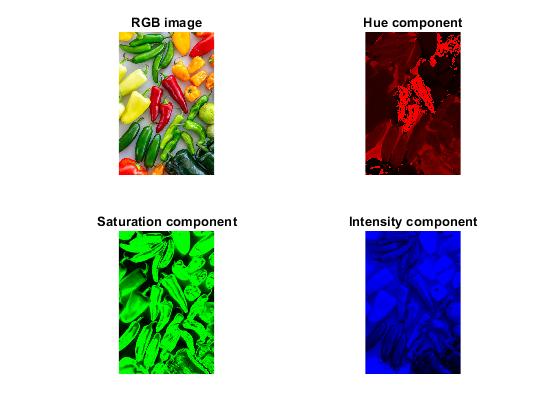
subplot(224);

imshow(I);

title('Intensity component');

Input and Output:





Problem No: 10

Problem Title: Write a MATLAB program for smoothing and sharpening of a color image.

Color image smoothing: is the technique intended for removing possible image perturbations without losing image information. White Gaussian noise is the most common factor which can significantly affect visual quality of images; each pixel of the image will be changed from its original value by some small amount that follows a Gaussian distribution.

Color image sharpening: this is a technique whose purpose is the improvement of the image visual appearance and highlight or recover certain details of the image for conducting a suitable analysis by a human or a machine.

Matlab source code:

clc;

clear all;

close all;

rgb\_img=imread('peppers.jpg');

rgb\_img=im2double(rgb\_img);

figure(1)

subplot(221);imshow(rgb\_img);title('Original RGB image');

% Creating filter

avg\_filter=fspecial('average',[5 5]);

laplacian\_filter=fspecial('laplacian',0.2);

% Filtering RGB components independently

R=rgb\_img(:,:,1); avg\_R=imfilter(R,avg\_filter);

lap\_R=imfilter(R,laplacian\_filter);

G=rgb\_img(:,:,2); avg\_G=imfilter(G,avg\_filter);

lap\_G=imfilter(G,laplacian\_filter);

B=rgb\_img(:,:,3); avg\_B=imfilter(B,avg\_filter);

lap\_B=imfilter(B,laplacian\_filter);

% Combining 3 channels after filtering

avg\_rgb\_img=cat(3,avg\_R,avg\_G,avg\_B);

lap\_rgb\_img=cat(3,lap\_R,lap\_G,lap\_B);

%%%% Converting to HSI from RGB

hsi\_img=rgb2hsv(rgb\_img);

% Filtering intensity component only

H=hsi\_img(:,:,1);

S=hsi\_img(:,:,2);

I=hsi\_img(:,:,3); avg\_I=imfilter(I,avg\_filter);

lap\_I=imfilter(I,laplacian\_filter);

% Combining 3 channels after filtering

avg\_hsi\_img=cat(3,H,S,avg\_I);

lap\_hsi\_img=cat(3,H,S,lap\_I);

% Converting to RGB from HSI

avg\_rgb\_img\_from\_hsi=hsv2rgb(avg\_hsi\_img);

lap\_rgb\_img\_from\_hsi=hsv2rgb(lap\_hsi\_img);

% Calculate difference

diff\_avg\_img=avg\_rgb\_img-avg\_rgb\_img\_from\_hsi;

diff\_lap\_img=lap\_rgb\_img-lap\_rgb\_img\_from\_hsi;

% Displaying

subplot(222);

imshow(avg\_rgb\_img);

title('Smoothing using RGB model');

subplot(223);

imshow(avg\_rgb\_img\_from\_hsi);

title('Smoothing using HSI model');

subplot(224);

imshow(diff\_avg\_img);

title('Difference');

figure(2)

subplot(221);

imshow(rgb\_img);

title('Original RGB image');

subplot(222);

imshow(lap\_rgb\_img);

title('Sharpening using RGB model');

subplot(223);

imshow(lap\_rgb\_img\_from\_hsi);

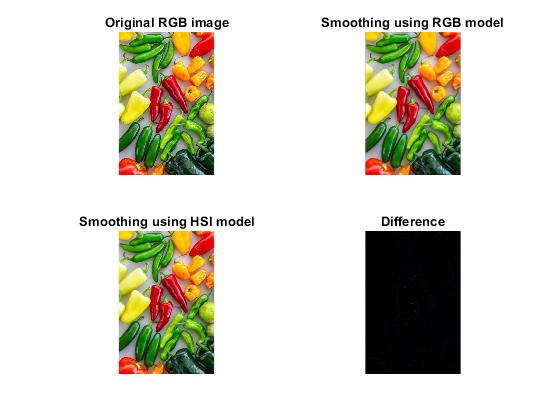
title('Sharpening using HSI model');

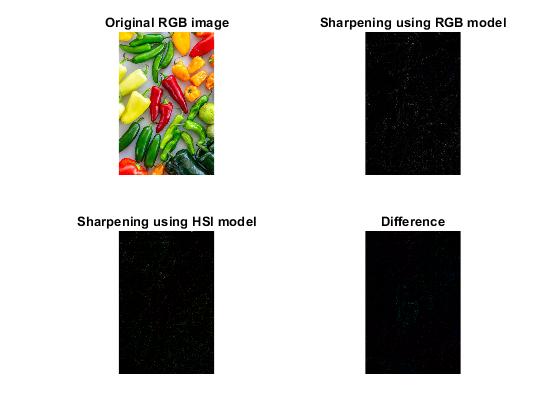
subplot(224);

imshow(diff\_lap\_img);

title('Difference');

Input and Output:





Problem No: 11

Problem Title: Write a MATLAB program for (i) Haar Transform (ii) one dimensional Discrete Wavelet Transform (iii) Discrete Cosine Transform of an image.

(i). Haar Transform of an image:

The Haar transform is the simplest of the wavelet transforms. This transform cross-multiple a function against the Haar wavelet with various shifts and stretches, like the fourier transform cross-mulitple a function against a sine wave with two phased and many stretches.

Haar transform can be expressed in the following matrix form,

*T=HFHT*

Where F is an N\*N image matrix, H is an N\*N Haar transformation matrix and T is the resulting N\*N transform. The transpose is required because H is not symmetric. For the Haar transform, H contains the Haar basis functions hk(z). They are defined over the continuous closed interval z∈[0,1] for k=0,1,2…… N-1 where N=2n. to generate H, we define he integer k such that k=2p+q-1, where 0≤p≤n-1, q=0 or 1 for p=0 and 1≤q≤2p for p≠ 0. The the Haar basis functions are h0(z)=h00(z)=

Matlab source code:

%haartransfrom

clc;

clear all;

close all;

i=imread('peppers.png');

subplot(231); imshow(i); title('original image');

%2D discrete wavelt transformations (single level) using Haar Basic

%functions

[ia1, ih1, iv1, id1]=dwt2(i, 'haar');

%display different coefficients

a1=ia1/255; h1=log10(ih1)\*0.3; v1=log10(iv1)\*0.3; d1=log10(id1)\*0.3;

subplot(232); imshow(real(a1)); title('Approximation');

subplot(233); imshow(abs(h1)); title('Horizontal details');

subplot(234); imshow(abs(v1)); title('Vertical detail');

subplot(235); imshow(abs(d1)); title('diagonal detail');

%combined different coefficients in one image

transformed\_i\_level\_1=[a1 v1 ; h1 d1];

subplot(236); imshow(abs(transformed\_i\_level\_1));

title('level-1 transformation');

%reconstruction from level-1 transformation

rec\_i=idwt2(ia1, ih1, iv1, id1, 'haar');

figure(2);

subplot(211);imshow(rec\_i/255);

title('Reconstruction from level-1 transformation');

%leve-2 transformation

[ia2, ih2, iv2, id2]=dwt2(ia1, 'haar');

%combined different coefficients in one image

a2=ia2/255; h2=log10(ih2)\*0.3; v2=log10(iv2)\*0.3; d2=log10(id2)\*0.3;

tranformed\_i\_level\_2=[[a2 v2 ; h2 d2] v1 ; h1 d1];

subplot(212);

imshow(abs(tranformed\_i\_level\_2));

title('Level-2 transformation');

Input and Output:





(ii). One dimensional Discrete Wavelet Transform of an image:

A discrete wavelet transform (DWT) is any wavelet transform for which the wavelets are discretely sampled. As with other wavelet transforms, a key advantage it has over Fourier transforms is temporal resolution: it captures both frequency and location information (location in time).

Matlab source code:

clc;

close all;

clear all;

X=imread('peppers.png');

figure(1);

subplot(1,2,1);

imshow(X);

title('Original Image');

X= double(X);

i=imresize(X,[512 512]);

subplot(1,2,2);

imshow(i);

title('Resize Image');

sX=size(X);

[LL,LH,HL,HH]= dwt2(X,'db1');

figure(2)

subplot(2,2,1);imshow(LL);title('LL band of image');

subplot(2,2,2);imshow(LH);title('LH band of image');

subplot(2,2,3);imshow(HL);title('HL band of image');

subplot(2,2,4);imshow(HH);title('HH band of image');

Input and Output:





(iii). Discrete Cosine Transform of an image:

A discrete cosine transform expressed a finite sequence of data points in terms of a sum of a cosine functions oscillating at different frequencies.

Discrete cosine transform helps separate the image into parts (or spectral sub-bands) of different importance with respect to the image’s visual quality. The DCT is similar to the discrete Fourier transform, it transforms a signal or image from the spatial domain to the frequency domain.

Matlab source code:

%discrete cosine transformation

clc;

clear all;

close all;

img=imread('autumn.tif');

img=rgb2gray(img);

subplot(131); imshow(img); title('original image');

%2d discrete cosine transformation

dct\_img=dct2(img);

subplot(132);

imshow(log(abs(dct\_img)), []);

colormap(gca, jet);

title('image after dct');

%inverse discrete cosine transormation

dct\_img(abs(dct\_img)<10)=0;

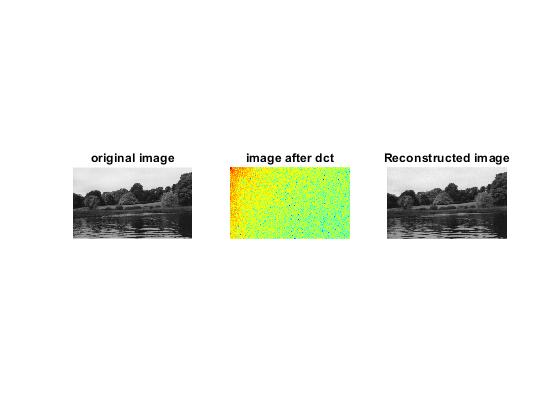
idct\_img=idct2(dct\_img);

subplot(133);

imshow(idct\_img, [0, 255]);

title('Reconstructed image');

Input and Output:



Problem No: 12

Problem Title: Write a MATLAB program for edge detection using Sobel, Canny, Prewitt, Roberts, log, zerocross filter.

**Sobel Filter:** the sobel filter is used for edge detection. It works by calculating the gradient of image intensity at each pixel within the image. It finds the direction of the largest increase from light to dark and the rate of change of that direction. The result show how abruptly or smoothly the image changes it each pixel and therefore how likely it is that pixel represents and edge. The soble filter uses two 3\*3 kernels. One for changes in the horizontal direction and one for changes in the vertical direction.

**Prewitt Filter:** the prewitt filter is a discrete differentiation filter that computes the gradient estimate of the image intensity function where its output is the equivalent gradient vector or the norm of the gradient vector. It applies convolution on the image with a separable, integer and small valued filter in the horizontal/vertical direction. The prewitt filter (HPrewitt) with a different window size can be represented as follows. At size 3\*3, HPrewitt(3\*3) is given by:

1 1 1

0 0 0

-1 -1 -1

**Canny Filter:** The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. The Gaussian reduces the effect of noise present in the image.

Matlab source code:

%edge detection

clc;

clear all;

close all;

I=imread('coins.png');

figure(1),

subplot(231), imshow(I), title('Original image');

BW1=edge(I, 'sobel');

BW2=edge(I, 'canny');

subplot(232), imshow(BW1), title('Soble filter');

subplot(233), imshow(BW2), title('Canny filter');

I=imread('circuit.tif');

subplot(234), imshow(I), title('Original image');

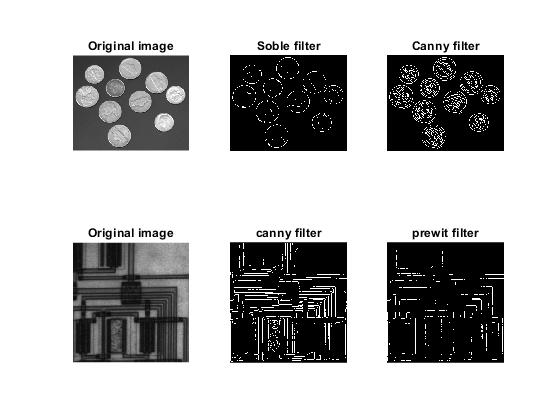
BW1=edge(I, 'canny');

BW2=edge(I, 'prewitt');

subplot(235), imshow(BW1), title('canny filter');

subplot(236), imshow(BW2), title('prewit filter');

Input and Output:



Problem No: 13

Problem Title: Write a MATLAB program to implement a LPF (FIR) with cutoff 8KHz and to denoise audio.

FIR filters: FIR filters are widely used due to the powerful design algorithms that exist for them, their inherent stability when implemented in non-recursive form, the ease with which one can attain linear phase, their simple extensibility to multirate cases, and the ample hardware support that exists for them among other reasons. FIR filters with a variety of characteristics.

Obtaining Low pass FIR Filter Coefficients: To summarize, two functions are presented that return a vector of FIR filter coefficients: firceqrip and firgr. Firceqrip is used when the filter order (equivalently the filter length) is known and fixed. The choice of a filter order of 100 was arbitrary. In general, a larger order results in a better approximation to ideal at the expense of a more costly implementation. Doubling the order roughly reduces the filter's transition width in half (assuming all other parameters remain the same).

Matlab source code:

clc;

clear all;

close all;

[filename, pathname]=uigetfile(".", 'select the input audio');

[x, Fs]=audioread(num2str(filename));

%filter implementation

Fsf=44100; %sampling frequency

Fp=8e3; %passband frequency in Hz

Fst=8.4e3; %stopband frequency in Hz

Ap=1; %passband ripple in db

Ast=95; %stopband attenuation in db

%design the filter

df=designfilt('lowpassfir', 'PassbandFrequency', Fp, 'StopbandFrequency', Fst, 'PassbandRipple', Ap, 'StopbandAttenuation', Ast, 'SampleRate', Fsf);

fvtool(df) %visualize frequency response fvtool(df);

xn=awgn(x,15, 'measured'); %signal corrupted by white gaussian noise

y=filter(df, xn);

subplot(3,1,1), plot(x(1:450)); title('original signal');

subplot(3,1,2), plot(xn(1:450)); title('noisy signal');

subplot(3,1,3), plot(y(1:450)); title('filtered signal');

audiowrite('noisy\_signal.wav',xn,Fs);

audiowrite('filtered\_signal.wav',xn,Fs);

Input and Output:

Input audio:



Output audio:

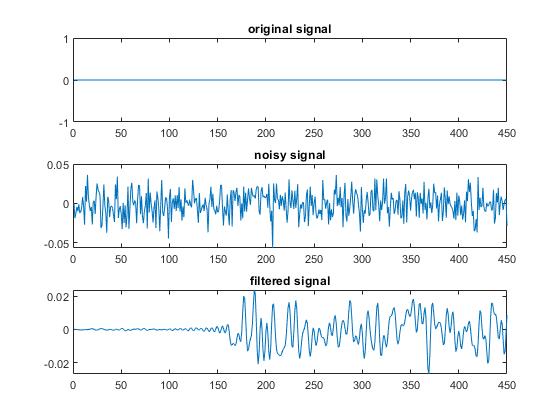
Noisy signal:

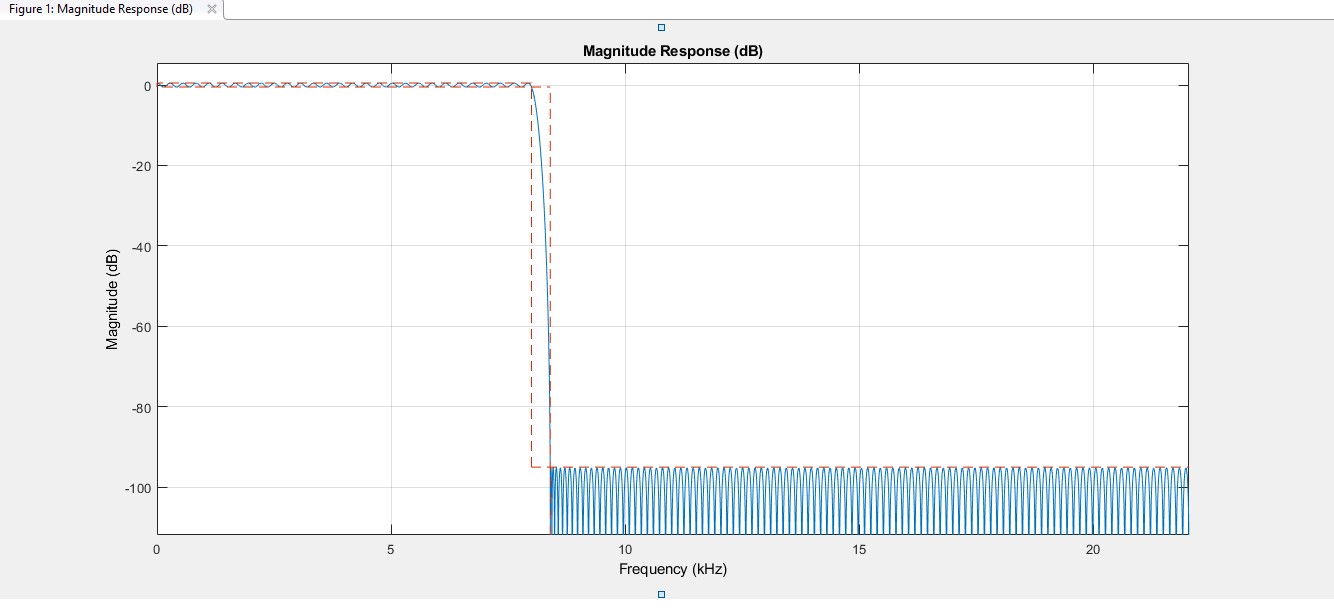


Filtered signal:



Outputs:





Problem No: 14

Problem Title: Write a MATLAB program to implement Echo of audio signal.

In audio signal processing, echo is a reflection of sound that arrives at the listener with a delay after the direct sound. The delay is directly proportional to the distance of the reflecting surface from the sound and the listener.

In this case listeners perceive and audible repetition of a signal after some duration of time. Listeners perceive distinct echoes when the time delay is relatively long. When a time delay is short listeners do not perceive echoes. Instead a single fused sound is perceived.

Matlab source code:

clc;

clear all;

close all;

[filename, pathname]=uigetfile(".", 'select the input audio');

[x, Fs]=audioread(num2str(filename));

n=length(x); %get the length of the audio file

a=0.8; %attenuation factor

d=2000; %delay

y=zeros(n+d, 1); %initialize the output audio file

xn=padarray(x, [d,0], 0, 'Pre');

for i=(d+1):1:n

y(i-d, 1)=x(i)+a\*xn(i-d);

end

audiowrite('echo\_signal.wav',y,Fs);

Input and Output:

Input audio:



Output Audio:



Problem No: 15

Problem Title: Write a MATLAB program to record and save single and double channel audio.

**Record and save audio:**

Record data from an audio input device such as a microphone connected to a system:

1. Create an audiorecorder object.

2. Call the record or recordblocking method, where:

•record returns immediate control to the calling function or the command prompt even as recording proceeds. Specify the length of the recording in seconds, or end the recording with the stop method. The recording is performed asynchronously.

•record blocking retains control until the recording is complete. Specify the length of the recording in seconds. The recording is performed synchronously.

3. Create a numeric array corresponding to the signal data using the getaudiodata method.

**Record microphone input:** Create an audio recorder object named recObj for recording audio input.

Matlab source code:

clc;

clear all;

close all;

Fs=44100;

noc=1;

nob=16;

disp('Start Recording');

recObj=audiorecorder(Fs,nob,noc);

recordblocking(recObj,5);

play(recObj);

myRecording=getaudiodata(recObj);

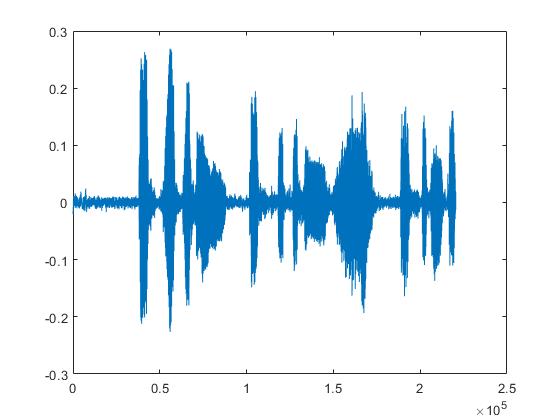
plot(myRecording);

sound(myRecording,Fs);

audiowrite('Recorded\_audio.wav',myRecording,Fs);

Input and Output:

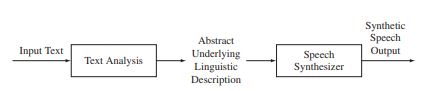




Problem No: 16

Problem Title: Write a MATLAB program to implement Text to Speech signal.

**Text to speech**: In the field of text-to-speech synthesis, or TTS is widely known where the goal of the machine is to convert ordinary text messages into intelligible and natural sounding synthetic speech so as to transmit transformation from a machine to a person by voice.



**Figure**: The block diagram of a basic TTS system

Ideally, the input to the TTS system is arbitrary input text and the output of the TTS system is synthetic speech. There are two fundamental processes that all TTS systems must perform. An analysis of the text must be performed in order to determine the abstract underlying linguistic description of the speech signal. Then the proper sounds corresponding to the text input must be synthesized; i.e., a sampled version of the desired spoken output must be created via the speech synthesizer in a form that can be converted to an acoustic signal by a D-to-A converter.

Matlab source code:

clc;

clear all;

close all;

NET.addAssembly('System.speech');

mySpeaker=System.Speech.Synthesis.SpeechSynthesizer;

mySpeaker.Rate=0;

mySpeaker.Volume=100;

Speak(mySpeaker, 'this is digital Image and Speech Processing');

Input and Output:

Input sentence: 'this is digital Image and Speech Processing'

The programme run successfully and speaks the sentence.