

**POORNIMA INSTITUTE OF ENGINEERING &
TECHNOLOGY, JAIPUR****Department of Computer Science and Engineering****Lab Manual****Digital Image Processing Lab**

<6CS4-21>



Branch	CS	Name of Lab	DIP Lab
Session	2019-20	Subject Code	6CS4-21
Year	3rd Year	Faculty	
Semester	6th Semester	Lab Assistant	

Document No.	PIET/CS/CS/2019/IV/LAB/DIP/ 6CS4-21	Created By	
Version		Verified By	
Authorized By (HOD)	Mr. Deepak Moud		

LAB RULES

DO'S	DON'TS
Be regular to the lab.	Do not come late to the lab.
Follow proper dress code.	Do not throw the connecting wires on the floor.
Maintain Silence.	Do not operate μ p/IC trainer kits unnecessarily.
Know the theory behind the experiment before coming to the lab.	Don't bring any external material inside the LAB.
Arrange the chairs/stools and equipment properly before leaving the lab.	Do not panic if you don't get the output.
Avoid unnecessary talking while doing the experiment.	Don't carry any LAB equipment outside the lab.
Keep the Table clean.	Do not try to repair or tamper lab equipment.

TABLE OF CONTENTS

S.No.	Contents	Page No.
1	Lab Rules	2
2	Instructions	4
3	Zero Lab	4
4	Syllabus	5
5	Marks Scheme	7
6	Lab Plan	8
7	Introduction to the subject	9
8	Lab objective	10
9	Procedure	11
10	List of lab exercises	12
11	Experiments	13-21
12	Resources	22

INSTRUCTIONS

Before entering in the lab

All the students are supposed to prepare the theory regarding the next experiment.

Students are supposed to bring the practical file and the lab copy.

Previous programs should be written in the practical file.

All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

Adhere to experimental schedule as instructed by the lab in-charge.

Get the previously executed program signed by the instructor.

Get the output of the current program checked by the instructor in the lab copy.

Each student should work on his/her assigned computer at each turn of the lab.

Take responsibility of valuable accessories.

Concentrate on the assigned practical and do not play games

If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

Syllabus

List of Experiments

1. Point-to-point transformation. This laboratory experiment provides for thresholding an image and the evaluation of its histogram. Histogram equalization. This experiment illustrates the relationship among the intensities (gray levels) of an image and its histogram.
2. Geometric transformations. This experiment shows image rotation, scaling, and translation. Two-dimensional Fourier transform.
3. Linear filtering using convolution. Highly selective filters.
4. Ideal filters in the frequency domain. Non Linear filtering using convolutional masks. Edge detection. This experiment enables students to understand the concept of edge detectors and their operation in noisy images.
5. Morphological operations: This experiment is intended so students can appreciate the effect of morphological operations using a small structuring element on simple binary images. The operations that can be performed are erosion, dilation, opening, closing, open-close, close-open.

MARKS SCHEME**RTU Marks Scheme**

Maximum Marks Allocation		
Sessional	End-Term	Total
30	20	50

Marks Division

Mid Term I & II		
Practical	Viva	Total
15	5	20
Attendance & Performance		
Performance	Attendance	Total
15	5	20
End-Term Practical		
Practical	Viva	Total
15	5	20

Internal Assessment System

Total Marks – 10

Attendance	Discipline	Performance	Record	Viva	Total
2	2	2	2	2	10

LAB PLAN

Total number of experiment: - 10

Experiment Number	Turns	Scheduled Day
EXP-1	1	Week -1
EXP-2	1	Week -2
EXP-3	1	Week -3
EXP-4	1	Week -4
EXP-5	1	Week -5
EXP-6	1	Week -6
EXP-7	1	Week -7
EXP-8	1	Week -8
EXP-9	1	Week -9
EXP-10	1	Week -10

Distribution of lab hours

Attendance	05 minutes
Explanation of concept	15 minutes
Explanation of experiment	30 minutes
Performance of experiment	60 minutes
Viva / Quiz / Queries / Evaluation of Records	10 minutes
Total	120 minutes

Software required MATLAB

LIST OF EXPERIMENTS 2019-20		
DIGITAL IMAGE PROCESSING LAB(8CS7)		
Experiment No.	Detail of Experiment	Functions used
1	<ol style="list-style-type: none"> 1. Introduction to OCTAVE 2. Extracting image properties 3. Reading, Writing and displaying an image 	<ol style="list-style-type: none"> 1. Iminfo() 2. Info.FileName 3. Info.FileModDate 4. Info.FileSize 5. Info.Width 6. Info.Height 7. imread() 8. imwrite() 9. imshow() 10. sublots
2	<ol style="list-style-type: none"> 1. Separating different channels from images 2. Conversion of images from one format to another 3. Extracting of bit planes in a gray scale image 	<ol style="list-style-type: none"> 1. img(:, :, 1), img(:, :, 2), img(:, :, 3) 2. ind2gray() 3. gray2ind() 4. rgb2gray() 5. gray2rgb() 6. rgb2ind() 7. ind2rbg()
3	<ol style="list-style-type: none"> 1. Resizing the images using program and inbuilt functions 2. Arithmetic operations on an image 	<ol style="list-style-type: none"> 1. imresize() 2. imadd() 3. imsubtract() 4. immul() 5. imdiv() 6. imcomplement() 7. OR operation 8. XOR operation 9. AND operation

4	1. Lograthmic Operator (to reduce contrast of brighter regions) 2. Exponential raise(to enhance contrast of brighter regions) 3. Histogram equalization using program and inbuilt functions	1. histeq()
5	Applying low pass filter and high pass filter on images in spatial domain	1. fspecial() 2. imfilter()
6	Noise application on images and its removal	1. imnoise() 2. filter2()
7	Applying low pass filter and high pass filter on images in frequency domain	1. fft2() 2. fftshift()
8	1 Edge detection 2. Image compression 3. Morphological operator	1. edge() 2. imdilate() 3. imerode()
9	1. Image segmentation using threshold value 2. Labelling of objects in an image using segmentation	
10	Small project allocation	

EXPERIMENT 1

Lab Objective:

1. Introduction to OCTAVE

2. Extracting image properties

3. Reading, Writing and displaying an image

About MATLAB:

Extracting image properties:

A digital image differs from a photo in that the values are all discrete. • Usually they take on only integer values. • A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it. These dots are called picture elements, or more simply pixels.

Types of Digital Images

- Binary: Each pixel is just black or white. Since there are only two possible values for each pixel (0,1), we only need one bit per pixel.
- Grayscale: Each pixel is a shade of gray, normally from 0 (black) to 255 (white). This range means that each pixel can be represented by eight bits, or exactly one byte. Other greyscale ranges are used, but generally they are a power of 2.
- True Color, or RGB: Each pixel has a particular color; that color is described by the amount of red, green and blue in it. If each of these components has a range 0–255, this gives a total of 256³ different possible colors. Such an image is a “stack” of three matrices; representing the red, green and blue values for each pixel. This means that for every pixel there correspond 3 values.

Following functions can be used to extract image properties

1. `iminfo()`
2. `Info.Filename`
3. `Info.FileModDate`
4. `Info.FileSize`
5. `Info.Width`
6. `Info.Height`

Reading, Writing and displaying an image

Syntax

`A = imread(filename,fmt)`

Description

`A = imread(filename,fmt)` reads a greyscale or color image from the file specified by the string `filename`, where the string `fmt` specifies the format of the file. If the file is not in the current directory or in a directory in the MATLAB path, specify the full pathname of the location on your system. Display An Image To display image, use the `imshow` function. Syntax `imshow(A)` Description `imshow(A)` displays the image stored in array `A`.

Writing Image :

Syntax

`imwrite(A,filename,fmt)`

Example: `a=imread('pout.tif');`

`imwrite(a,gray(256),'b.bmp');`

`imshow('b.bmp')% imshow is used to display image`

How to get no. of rows and columns of image :

Function `size` gives the rows and columns dimension of image

`[r,c]=size(a)`

EXPERIMENT 2

Lab Objective:

- 1. Separating different channels from images**
- 2. Conversion of images from one format to another**
- 3. Extracting of bit planes in a gray scale image**

Separating different channels from images:

Red Channel-`img(:,:,1)`

Green Channel-`img(:,:,2)`

Blue Channel- `img(:,:,3)`

Conversion of images from one format to another

1. `ind2gray()`

2. `gray2ind()`

3. `rgb2gray()`

4. `gray2rgb()`

5. `rgb2ind()`

6. `ind2rbg()`

Extracting of bit planes in a gray scale image

We are given a Grayscale Image. Since pixel value of grayscale image lies between 0 -255, so its information is contained using 8 bit. So, we can divide those image into 8 planes (8 Binary Image). Binary image are those images whose pixel value can be either 0 or 1.

So, Our task is to extract each bit planes of original image to make 8 binary image

Let particular pixel of grayscale image has value 212. So, its binary value will be 11010100. So, its 1st bit is 0, 2nd is 0, 3rd is 1, 4th is 0, 5th is 1, 6th is 0, 7th is 1, 8th is 1. In this manner, we will take these 8 bit of all pixel and will draw 8 binary image. We have to do this to all the pixels and generate new images.

Below is the implementation of above idea in matlab.

% clearing the output screen

clc;

% reading image's pixel in c

c = imread('cameraman.tif');

% storing image information in cd

cd = double(c);

% extracting all bit one by one

% from 1st to 8th in variable

% from c1 to c8 respectively

c1 = mod(cd, 2);

c2 = mod(floor(cd/2), 2);

c3 = mod(floor(cd/4), 2);

c4 = mod(floor(cd/8), 2);

c5 = mod(floor(cd/16), 2);

c6 = mod(floor(cd/32), 2);

c7 = mod(floor(cd/64), 2);

c8 = mod(floor(cd/128), 2);

% combining image again to form equivalent to original grayscale image

cc = (2 * (2 * (2 * (2 * (2 * (2 * (2 * c8 + c7) + c6) + c5) + c4) + c3) + c2) + c1);

% plotting original image in first subplot

```
subplot(2, 5, 1);  
  
imshow(c);  
  
title('Original Image');  
  
  
% plotting binary image having extracted bit from 1st to 8th  
  
% in subplot from 2nd to 9th  
  
subplot(2, 5, 2);  
  
imshow(c1);  
  
title('Bit Plane 1');  
  
subplot(2, 5, 3);  
  
imshow(c2);  
  
title('Bit Plane 2');  
  
subplot(2, 5, 4);  
  
imshow(c3);  
  
title('Bit Plane 3');  
  
subplot(2, 5, 5);  
  
imshow(c4);  
  
title('Bit Plane 4');  
  
subplot(2, 5, 6);  
  
imshow(c5);  
  
title('Bit Plane 5');  
  
subplot(2, 5, 7);  
  
imshow(c6);  
  
title('Bit Plane 6');  
  
subplot(2, 5, 8);
```

```
imshow(c7);  
title('Bit Plane 7');  
subplot(2, 5, 9);  
imshow(c8);  
title('Bit Plane 8');  
  
% plotting recombined image in 10th subplot  
subplot(2, 5, 10);  
imshow(uint8(cc));  
title('Recombined Image');
```

EXPERIMENT 3

Lab Objective:

1. Resizing the images using program and inbuilt functions

2. Arithmetic operations on an image

Resize by scaling

```
I = imread('rice.png');  
J = imresize(I, 0.5);  
figure  
imshow(I)  
title('Original Image')
```

Resize with specified rows and columns

```
% resize by specifying rows
```

```
% and columns
```

```
I2 = imresize(I, [100, 200]);
```

```
%display image size
```

```
size(I2)
```

```
%display the image
```

```
figure, imshow(I2);
```

Image Addition

To add two images or add a constant value to an image, use the `imadd` function. `imadd` is matlab inbuilt function, It adds the value of each pixel in one of the input images with the corresponding pixel in the other input image and returns the sum in the corresponding pixel of the output image.

% img add by constant factor

```
I = rgb2gray(imread('lena.bmp'));
```



```
J = imadd(I,50);

K = imadd(I,90);

L = imadd(I,130);


subplot(2,2,1), imshow(I), title('lena.bmp');

subplot(2,2,2), imshow(J),title('Constant Factor: 50');

subplot(2,2,3), imshow(K),title('Constant Factor: 90');

subplot(2,2,4), imshow(L),title('Constant Factor: 130');

% Addition of Images

close all;

I1=imread('back.bmp');

I2=imresize(I1,[500 500]);

I3 = I2;%background

I1=imread('car.bmp');

I2=imresize(I1,[500 500]);

I4 = I2;%car

J = imadd(I3,I4);

subplot(1,3,1), imshow(I3)

subplot(1,3,2), imshow(I4)

subplot(1,3,3), imshow(J)
```

```
title('Image Addition');  
  
imwrite(J,'output_add_image.png','png');
```



Image Subtraction

imsubtract is inbuilt matlab function to subtract one image from another, or subtract a constant value from an image, use the imsubtract function. imsubtract subtracts each pixel value in one of the input images from the corresponding pixel in the other input image and returns the result in the corresponding pixel in an output image. Image subtraction can be used for more complex image processing.

You can use image subtraction to detect changes in a series of images of the same scene. For this operation the images must be the same size and class.

% image subtraction

```
close all;
```

```
I1=imread('church.bmp');
```

```
I2=imresize(I1,[500 500]);
```

```
I3 = I2;%rgb2gray(I2);
```

```
I1=imread('mask.bmp');  
  
I2=imresize(I1,[500 500]);  
  
I4 = I2;  
  
J = imsubtract(I3,I4);  
  
subplot(1,3,1), imshow(I3)  
  
subplot(1,3,2), imshow(I4)  
  
subplot(1,3,3), imshow(J)  
  
title('image subtraction');  
  
imwrite(J,'output_sub_image.png','png');
```

OUTPUT



Image Multiplication

immultiply is inbuilt matlab function to multiply one image with another, or multiply a constant value from an image, use the immultiply function. immultiply multiplies each pixel value in one of the input images from the corresponding pixel in the other input image and returns the result in the corresponding pixel in an output image. if elements of output_image exceeding the range of the integer type are truncated, and fractional values are rounded. Image multiplication can be used for more complex image processing.

```
% 1.To perform Basic Image Processing Operations
% a. Arithmetic and Logical Operations
clc;
clear all;
close all;
A=imread('cameraman.tif');
subplot(5,3,1)
imshow(A)
title('Image A');
B=imread('rice.png');
subplot(5,3,2)
imshow(B)
title ('Image B');
C=zeros(size(A));
for (x= 100: 200)
    for (y=100: 200)
        C(x,y)=255;
    end
end
subplot(5,3,3)
imshow(C)
title('Image C');
C=uint8(C);
% Logical Operations
lr1=bitand(A,C);
subplot(5,3,4)
imshow(lr1)
title('A and C');
lr2=bitand(B,C);
subplot(5,3,5)
imshow(lr2)
title('B and C');
lr3=bitand(A,B);
subplot(5,3,6)
imshow(lr3)
```

```
title('A and B');
lr4=bitor(A,C);
subplot(5,3,7)
imshow(lr2)
title('A or C');
lr5=bitor(B,C);
subplot(5,3,8)
imshow(lr5)
title('B or C');
lr6=bitor(A,B);
subplot(5,3,9)
imshow(lr6)
title('A or B');
lr7=bitxor(A,C);
subplot(5,3,10)
imshow(lr7)
title('A exor C');
lr8=bitxor(B,C);
subplot(5,3,11)
imshow(lr8)
title('B exor C');
lr9=bitxor(A,B);
subplot(5,3,12)
imshow(lr9)
title('A exor B');
lr10= bitcmp(A);
subplot(5,3,13)
imshow(lr10)
title('Not A');
lr11= bitcmp(B);
subplot(5,3,14)
imshow(lr11)
title('Not B');
lr12= bitcmp(C);
subplot(5,3,15)
imshow(lr12)
title('Not C');
% Arithmetic Operations
lr13=imadd(A,C);
figure,subplot(4,3,1)
imshow(uint8(lr13))
title('A + C');
lr14=imadd(B,C);
subplot(4,3,2)
imshow(uint8(lr14))
title('B + C');
```

```
lr15=imadd(A,B);
subplot(4,3,3)
imshow(uint8(lr15))
title('A + B');
lr16=imsubtract(A,C);
subplot(4,3,4)
imshow(uint8(lr16))
title('A - C');
lr17=imsubtract(B,C);
subplot(4,3,5)
imshow(uint8(lr17))
title('B - C');
lr18=imsubtract(A,B);
subplot(4,3,6)
imshow(uint8(lr18))
title('A - B');
lr19=immultiply(A,C);
subplot(4,3,7)
imshow(uint8(lr19))
title('A * C');
lr20=immultiply(B,C);
subplot(4,3,8)
imshow(uint8(lr20))
title('B * C');
lr21=immultiply(A,B);
subplot(4,3,9)
imshow(uint8(lr21))
title('A * B');
lr22=imdivide(A,C);
subplot(4,3,10)
imshow(uint8(lr22))
title('A/C');
lr23=imdivide(B,C);
subplot(4,3,11)
imshow(uint8(lr23))
title('B/C');
lr24=imdivide(A,B);
subplot(4,3,12)
imshow(uint8(lr24))
title('A/B');
```

EXPERIMENT 4

Lab Objective:

- 1. Logarithmic Operator (to reduce contrast of brighter regions)**
- 2. Exponential raise (to enhance contrast of brighter regions)**
- 3. Histogram equalization using program and inbuilt functions**

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. Histogram equalization often produces unrealistic effects in photographs; however it is very useful for scientific images like thermal or x-ray images, often the same class of images to which one would apply false color. Also histogram equalization can produce undesirable effects when applied to images with low color depth. For example, if applied to 8-bit image displayed with 8 bit gray scale it will further reduce color depth (number of unique shades of gray) of the image. Histogram equalization will work the best when applied to images with much higher depth than palette size, like continuous data or 16-bit gray-scale images.

% Program to obtain histogram equalization concept

```
I=imread('trees.tif');  
J=imcomplement(I);  
imhist(J,100); imshow(I);  
title('original');  
figure,imshow(J);  
title('complement');  
I=histeq(I);  
figure,imhist(I,64);  
title('equilized');  
figure,imhist(J,64);  
title('histogram');  
n=numel(I);  
p=imhist(I)/n;
```

```
figure,plot(p);  
title('normalized');  
K=imadjust(I,[0;1],[0.4;1],0.5);  
figure,imshow(K);  
title('adjusted image');  
T=maketform('affine',[.3 0 0;.5 1 0;0 1 1]);  
tformfwd([0,0],T);  
I2=imtransform(I,T);  
figure,imshow(I2);  
title('forward image');
```


EXPERIMENT 5

Lab Objective:

Applying low pass filter and high pass filter on images in spatial domain

Mean Filter

Mean filtering is easy to implement. It is used as a method of smoothing images, reducing the amount of intensity variation between one pixel and the next resulting in reducing noise in images.

The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a $3 \times 3 \times 3$ square kernel is used, as shown below:

$$1/9[1 \ 1 \ 1; 1 \ 1 \ 1; 1 \ 1 \ 1]$$

```
img = imread('hawk.png');  
mf = ones(3,3)/9;
```

Median filter - medfilt2()

```
I = imread('cameraman.tif');  
J = imnoise(I,'salt & pepper',0.02);  
K = medfilt2(J);  
subplot(121);imshow(J);  
subplot(122);imshow(K);
```

fspecial()

```
h = fspecial(type)
```

```
h = fspecial(type, parameters)
```

h = fspecial(type) creates a two-dimensional filter **h** of the specified type. It returns **h** as a correlation kernel, which is the appropriate form to use with **imfilter()**. The **type** is a string having one of these values:

Value	Description
average	Averaging filter
disk	Circular averaging filter (pillbox)
gaussian	Gaussian lowpass filter
laplacian	Laplacian of Gaussian filter
motion	Approximates the linear motion of a camera
prewitt	Prewitt horizontal edge-emphasizing filter
sobel	Sobel horizontal edge-emphasizing filter

```
I = imread('cameraman.tif');
radius = 1;
J1 = fspecial('disk', radius);
K1 = imfilter(I,J1,'replicate');
radius = 10;
J10 = fspecial('disk', radius);
K10 = imfilter(I,J10,'replicate');
subplot(131);imshow(I);title('original');
```

```
subplot(132);imshow(K1);title('disk: radius=1');  
subplot(133);imshow(K10);title('disk: radius=10');
```

EXPERIMENT 6

Lab Objective:

Noise application on images and its removal

Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene. There are several ways that noise can be introduced into an image, depending on how the image is created.

Remove Noise Using an Averaging Filter and a Median Filter

```
I = imread('eight.tif');  
figure  
imshow(I)  
J = imnoise(I,'salt & pepper',0.02);  
figure  
imshow(J)  
Kaverage = filter2(fspecial('average',3),J)/255;  
figure  
imshow(Kaverage)  
Kmedian = medfilt2(J);  
imshowpair(Kaverage,Kmedian,'montage')
```

Remove Noise By Adaptive Filtering

Try This Example

This example shows how to use the `wiener2` function to apply a Wiener filter (a type of linear filter) to an image adaptively. The Wiener filter tailors itself to the local image variance. Where the variance is large, `wiener2` performs little smoothing. Where the variance is small, `wiener2` performs more smoothing.

Read the image into the workspace.

```
RGB = imread('saturn.png');  
Convert the image from truecolor to grayscale.
```

```
I = rgb2gray(RGB);
```

Add Gaussian noise to the image

```
J = imnoise(I,'gaussian',0,0.025);
```

Display the noisy image. Because the image is quite large, display only a portion of the image.

```
imshow(J(600:1000,1:600));
```

```
title('Portion of the Image with Added Gaussian Noise');
```

Remove the noise using the `wiener2` function.

```
K = wiener2(J,[5 5]);
```

Display the processed image. Because the image is quite large, display only a portion of the image.

```
figure
```

```
imshow(K(600:1000,1:600));
```

```
title('Portion of the Image with Noise Removed by Wiener Filter');
```

EXPERIMENT 7

Lab Objective:

Applying low pass filter and high pass filter on images in frequency domain

Frequency Domain Specific Filters

See section 14.5.3 in your textbook, and Chapter 4 and Section 5.4 in Digital Image Processing Using MATLAB

As you have already seen, based on the property that multiplying the FFT of two functions from the spatial domain produces the convolution of those functions, you can use Fourier transforms as a fast convolution on large images. Note that on small images it is faster to work in the spatial domain.

However, you can also create filters directly in the frequency domain. There are three commonly discussed filters in the frequency domain:

- Lowpass filters, sometimes known as smoothing filters
- Highpass filters, sometimes known as sharpening filters
- Notch filters, sometimes known as band-stop filters

3.1 Lowpass Filters

Lowpass filters:

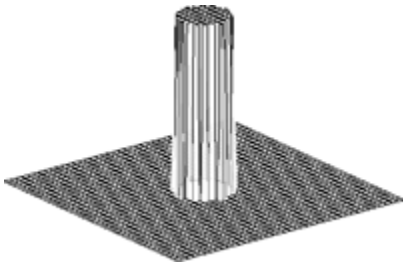
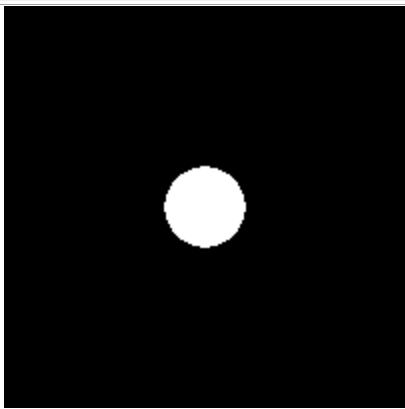
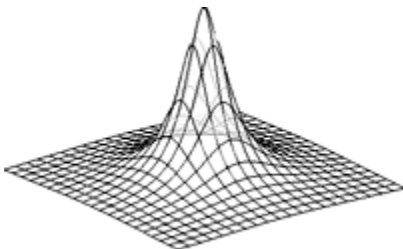

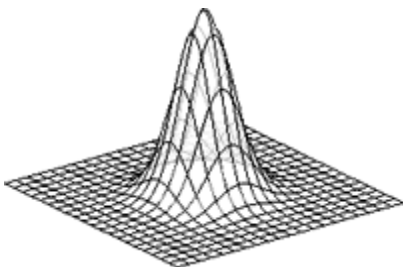
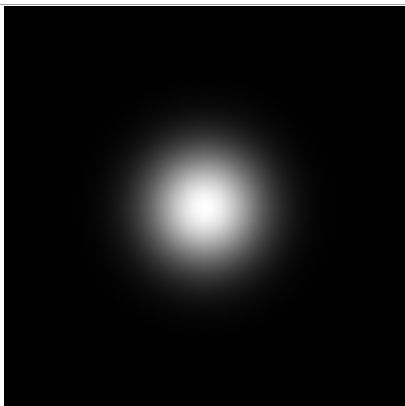
- create a blurred (or smoothed) image
- attenuate the high frequencies and leave the low frequencies of the Fourier transform relatively unchanged

Three main lowpass filters are discussed in *Digital Image Processing Using MATLAB*:

1. ideal lowpass filter (ILPF)
2. Butterworth lowpass filter (BLPF)
3. Gaussian lowpass filter (GLPF)

The corresponding formulas and visual representations of these filters are shown in the table below. In the formulae, D_0 is a specified nonnegative number. $D(u,v)$ is the distance from point (u,v) to the center of the filter.

Lowpass Filter	Mesh	Image

<p>Ideal:</p> $H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases}$		
<p>Butterworth:</p> $H(u, v) = \frac{1}{1 + [D(u, v) / D_0]^{2n}}$		
<p>Gaussian:</p> $H(u, v) = e^{-D^2(u, v) / 2D_0^2}$		

To view the MATLAB calls that were used to create the images in the above table, click [on this link](#).

The following is the result of applying a Gaussian lowpass filter on an image.

Original Image	Fourier Spectrum of Image
-----------------------	----------------------------------

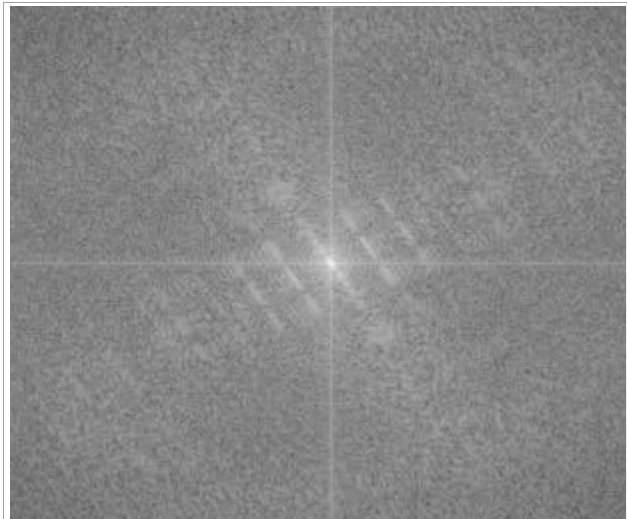
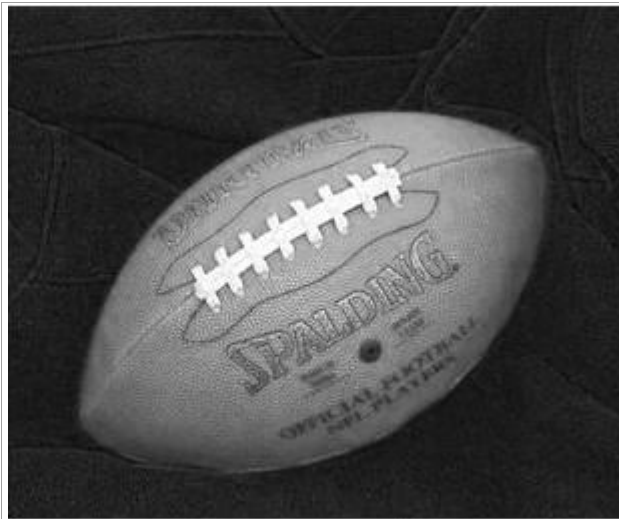
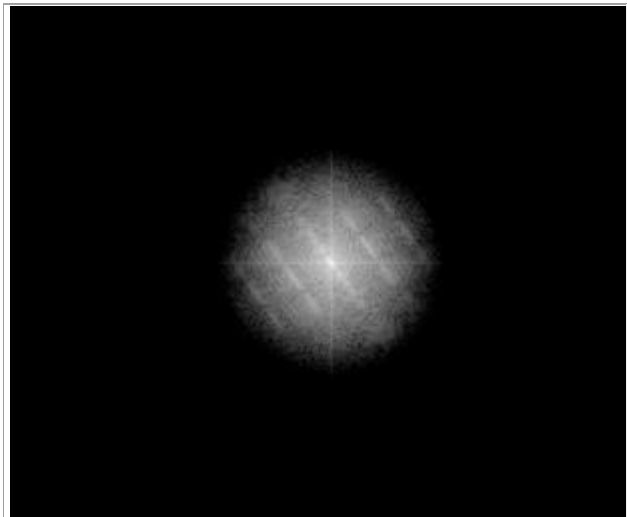


Image with Gaussian lowpass filter



Spectrum of image with Gaussian lowpass filter



The above images were created using three M-files ([paddedsize.m](#), [lpfilter.m](#) and [dftuv.m](#)) and the following MATLAB calls:

```
footBall=imread('football.jpg');  
%Convert to grayscale  
footBall=rgb2gray(footBall);  
imshow(footBall)
```

```
%Determine good padding for Fourier transform
```



```

PQ = paddedsize(size(footBall));

% Create a Gaussian Lowpass filter 5% the width of the Fourier transform
D0 = 0.05*PQ(1);
H = lpfilter('gaussian', PQ(1), PQ(2), D0);

% Calculate the discrete Fourier transform of the image
F=fft2(double(footBall),size(H,1),size(H,2));

% Apply the highpass filter to the Fourier spectrum of the image
LPFS_football = H.*F;

% convert the result to the spacial domain.
LPF_football=real(ifft2(LPFS_football));

% Crop the image to undo padding
LPF_football=LPF_football(1:size(footBall,1), 1:size(footBall,2));

% Display the blurred image
figure, imshow(LPF_football, [])

% Display the Fourier Spectrum
% Move the origin of the transform to the center of the frequency rectangle.
Fc=fftshift(F);
Fcf=fftshift(LPFS_football);
% use abs to compute the magnitude and use log to brighten display
S1=log(1+abs(Fc));
S2=log(1+abs(Fcf));
figure, imshow(S1,[])
figure, imshow(S2,[])

```

3.2 Highpass Filters



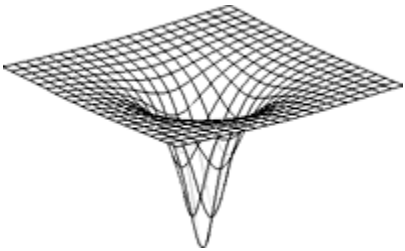
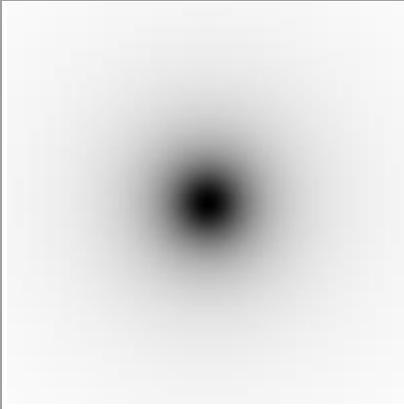
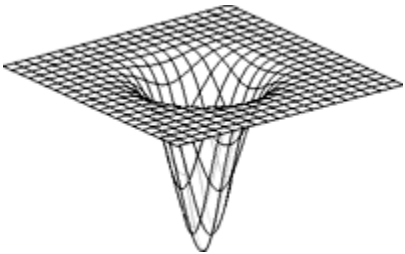
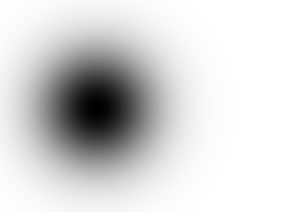
Highpass filters:

- sharpen (or shows the edges of) an image
- attenuate the low frequencies and leave the high frequencies of the Fourier transform relatively unchanged

The highpass filter (H_{hp}) is often represented by its relationship to the lowpass filter (H_{lp}):


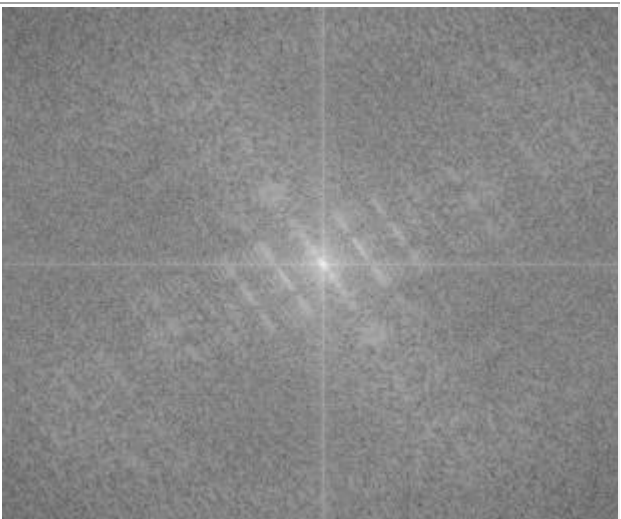

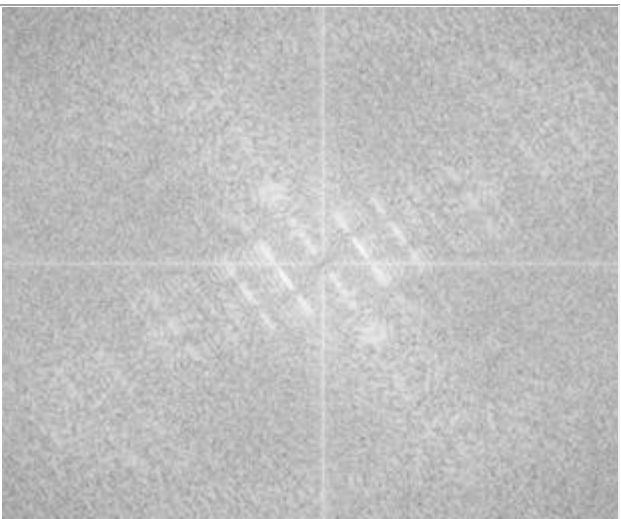
$$H_{hp}(u, v) = 1 - H_{lp}(u, v)$$

Because highpass filters can be created in relationship to lowpass filters, the following table shows the three corresponding highpass filters by their visual representations:

Lowpass Filter	Mesh	Image
Ideal		
Butterworth		
Gaussian		

To view the MATLAB calls that were used to create the images in the above table, click [on this link](#).

The following is the result of applying a Gaussian lowpass filter on an image.

Original Image	Fourier Spectrum of Image
	
Image with Gaussian highpass filter	Spectrum of image with Gaussian highpass filter
	

The above images were created using four M-files ([paddedsized.m](#), [dftuv.m](#), and [hpfilter.m](#)) and the following MATLAB calls

```
footBall=imread('football.jpg');
```

```
%Convert to grayscale  
footBall=rgb2gray(footBall);  
imshow(footBall)
```

```
%Determine good padding for Fourier transform
PQ = paddedsize(size(footBall));

%Create a Gaussian Highpass filter 5% the width of the Fourier transform
D0 = 0.05*PQ(1);
H = hpfilter('gaussian', PQ(1), PQ(2), D0);

% Calculate the discrete Fourier transform of the image
F=fft2(double(footBall),size(H,1),size(H,2));

% Apply the highpass filter to the Fourier spectrum of the image
HPFS_football = H.*F;

% convert the result to the spacial domain.
HPF_football=real(ifft2(HPFS_football));

% Crop the image to undo padding
HPF_football=HPF_football(1:size(footBall,1), 1:size(footBall,2));

%Display the "Sharpened" image
figure, imshow(HPF_football, [])

% Display the Fourier Spectrum
% Move the origin of the transform to the center of the frequency rectangle.
Fc=fftshift(F);
Fcf=fftshift(HPFS_football);
% use abs to compute the magnitude and use log to brighten display
S1=log(1+abs(Fc));
S2=log(1+abs(Fcf));
figure, imshow(S1,[])
figure, imshow(S2,[])
```

EXPERIMENT 8

Lab Objective:

1 Edge detection

2. Image compression

3. Morphological operator

Edge Detection:

The Canny edge detector is an edge detection operator that uses a multistage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works. The Process of Canny edge detection algorithm can be broken down to 5 different steps: 1. Apply Gaussian filter to smooth the image in order to remove the noise 2. Find the intensity gradients of the image 3. Apply non-maximum suppression to get rid of spurious response to edge detection 4. Apply double threshold to determine potential edges 5. Track edges by hypothesis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

%Program for edge detection algorithm

```
I=imread('coins.png');
```

```
figure,imshow(I) title ('figure 1 original image');
```

```
h=ones(5,5)/25;
```

```
b=imfilter(I,h);
```

```
figure,imshow(b) title ('figure 2 filtered image');
```

```
c=edge(b,'sobel');
```

```
figure,imshow(c) title ('figure 3 edge detected output by sobel operator');
```

```
d=edge(b,'prewitt');
```

```
figure,imshow(d) title ('figure 4 edge detected output by prewitt operator');
```

```
e=edge(b,'robert');
```

```
figure,imshow(e) title ('figure 5 edge detected output by robert operator');
```

```
f=edge(b,'canny'); figure,imshow(f) title ('figure 6 edge detected output by canny operator');
```

Morphological Operations

Erosion (usually represented by \ominus) is one of two fundamental operations (the other being dilation) in morphological image processing from which all other morphological operations are based. It was originally defined for binary images, later being extended to grayscale images, and subsequently to complete lattices. With A and B as two sets in Z^2 (2D integer space), the dilation of A and B is defined as $A(+)B = \{Z|(B \setminus Z) \cap A \neq \emptyset\}$. In the above example, A is the image while B is called a structuring element. In the equation, $(B \setminus Z)$ simply means taking the reflections of B about its origin and shifting it by Z. Hence dilation of A with B is a set of all displacements, Z, such that $(B \setminus Z)$ and A overlap by at least one element. Flipping of B about the origin and then moving it past image A is analogous to the convolution process. In practice flipping of B is not done always. Dilation adds pixels to the boundaries of object in an image. The number of pixels added depends on the size and shape of the structuring element. Based on this definition, dilation can be defined as $A(+)B = \{Z|(B \setminus Z) \cap A \neq \emptyset\}$

% Program for morphological operations:

```
Erosions& Dilation f=imread('coins.png');
```

```
B=[0 1 1;1 1 1;0 1 0];
```

```
f1=imdilate(f,B);
```

```
se=strel('disk',10);
```

```
f2=imerode(f,se);
```

```
figure,imshow(f) title('input image');
```

```
figure,imshow(f1) title('dilated image');
```

```
figure,imshow(f2) title('eroded image');
```

EXPERIMENT 9

Lab Objective:

- 1. Image segmentation using threshold value**
- 2. Labeling of objects in an image using segmentation**

Image segmentation is a commonly used technique in digital image processing and analysis to partition an image into multiple parts or regions, often based on the characteristics of the pixels in the image. Image segmentation could involve separating foreground from background, or clustering regions of pixels based on similarities in color or shape.

Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image.

Matlab Code to Count the no of Objects in a Binary Image

```
clc;  
  
clear;  
  
close all;  
  
% Read the Input Image  
  
InputImage=imread('eight.tif');  
  
% Display the Input Image  
  
subplot(2,2,1);  
  
imshow(InputImage);title('InputImage');
```

Convert the Gray Scale Image to Binary Image

```
% Convery Gray Scale Image to Binary Image
```

```
BinaryImage=im2bw(InputImage);
```

```
%Display Binary Image
```

```
subplot(2,2,2);
```

```
imshow(BinaryImage);
```

Complement the Binary Image

```
% Complement the Binary Image
```

```
ComplementImage=imcomplement(BinaryImage);
```

```
subplot(2,2,3);
```

```
imshow(ComplementImage);
```

Perform morphological operations

```
% Fill in the holes of a binary image
```

```
HolesClearedImage = imfill(ComplementImage,'holes');
```

```
%Display Holes Cleared Image
```

```
subplot(2,2,4);
```

```
imshow(HolesClearedImage);title('HolesClearedImage');
```

Apply labels for each binary connected component


```
%Apply Labels for each connected components in 2-D binary image
```

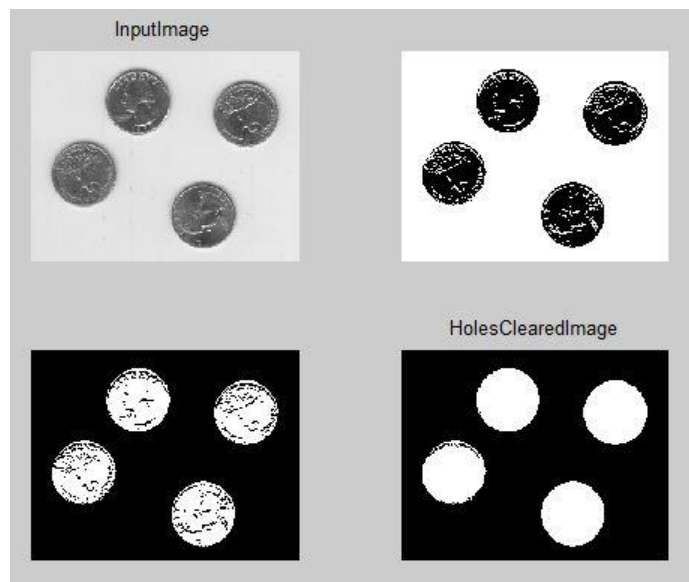
```
[L,Num] = bwlabel(HolesClearedImage)
```

```
%Display each label
```

```
figure;
```

```
for i=1:Num
```

```
subplot(2,2,i);
```



```
imshow(L==i);
```

```
pause(1)
```

```
end
```