Department of Electronic Engineering
NED University of Engineering & Technology

LABORATORY WORKBOOK

For the Course

DIGITAL IMAGE PROCESSING

(TC-424)

Instructor Name: _____

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Department: _____

LABORATORY WORKBOOK

For the Course

DIGITAL IMAGE PROCESSING

(TC-424)

Prepared By:
Ms. Sundus Ali (Lecturer)

Reviewed By:
Dr. Muhammad Imran Aslam (Associate Professor)

Approved By:
The Board of Studies of Department of Electronic Engineering

CONTENTS

LAB SESSION 01

To study the basic operation on matrices in MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	
Marks Obtained	

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 01

Objective:-

To study and investigate basic operations on matrices in MATLAB

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Theory:-

Entering and quitting MATLAB

To open MATLAB double click on the MATLAB icon. To leave MATLAB, simply type *quit* on command prompt and press enter.

Some basic commands
To check the list of installed toolboxes type
ver
To clear the screen type
clc
To clear the workspace type
clear
To list the current variables type
who
To list the current variables in long form type
Whos

Loading and saving variables
To save the current variables type
save<variable name>
To load the current variables type
load< variable name >
To save all the variables type
save<file name>
To load all the variables type
load< file name >

Creating and manipulating matrices & vectors
To create a vector *v* simply type in:

v = [2 4 7 5]

After pressing "return" the value of *v* will have been echoed back to you.
To suppress echo use semicolon after the command

w = [1 3 8 9];

To create, $z = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$

you can type
z = [1; 1; 0; 0]; or
z = [1

1
0
0];

To enter the matrix,

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

The most obvious way is,
M = [1 2; 3 4] or
M = [[1 3]' [3 4]']

Polynomials
In MATLAB, a polynomial is represented by a vector. To create a polynomial in MATLAB, simply enter each coefficient of the polynomial into the vector in descending order. For instance, let's say you have the following polynomial:
$x^4+3x^3-15x^2-2x+9$
To enter this into MATLAB, just enter it as a vector in the following manner
x = [1 3 -15 -2 9]
You can find the value of a polynomial using the *polyval* function. For example, to find the value of the above polynomial at *x=2*,
Z=polyval([1 3 -15 -2 9], 2)
Or
Z=polyval(x,2)
Finding the roots would be as easy as entering the following command;
roots([1 3 -15 -2 9])
Or
roots(x)

Convolution and De-convolution of Polynomials
The product of two polynomials is found by taking the convolution of their coefficients.
x+2 and **x^2+4x+8**
x = [1 2];
y = [1 4 8];
z = conv(x,y)
z =
1 6 16 16
*deconv is used t*o divide two polynomials. The *deconv* function will return the remainder as well as the result. Let's divide z by y and see if we get x
[xx, R] = deconv(z,y)

<u>Laboratory Task:-</u>

Task#1: Investigate the effect of following commands
(a) c=3 (b) d= 2*c/3 (c) e=c*d^2 (d) f=c-d+e (e) who
(f) whos (g) Save my work (h) clear (i) Load my work

Task#2: Investigate the effect of the following commands:
(a) v(2) (b) sum = v + w (c) diff = v – w (d) vw = [v w] (e) vw(2: 6)
(f) v' (g) v./w (h) v.*w

Task#3: Investigate the effect of the following commands:
(a) z' (b) z*v
(c) [v; w] (d) v*z

(e) [z; v'] (f) z + v'

Task#4: Investigate the effect of the following commands:
(a) N = inv(M) (b) M*N (c) I = eye(2)
(d) M + I (e) M*z(1:2) (f) v(3:4)*M
(g) M(1,1) (h) M(1:2,1:2) (i) M(:,1)
(j) M(2,:)

Task#5: Suppose $x=2t^2+3t+10$ and $y=3t^2+t-7$
(a) Enter the following polynomials in the MATLAB.
(b) Find the value of x at t=3 and value of y at t=2.
(c) Find out the roots of the polynomials.
(d) Multiply both polynomials
(d) Divide x by t+1.

Result:-

The results / output of all five tasks must be attached with this lab.

LAB SESSION 02

To study and investigate Loop Operations on matrices in MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 02

Objective:-

To study and investigate Loop operations on matrices in MATLAB

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Theory:-

During any operation, MATLAB walks through the arrays, element-by-element, and operates the scalar in each array position. This process is performed in a loop. MATLAB provides two basic loops; For loop & While loop

***for* Loop:**

Programs for numerical simulation often involve repeating a set of commands. In MATLAB, we instruct the computer to repeat a block of code a certain number of times, by using a "for loop".

Simple Example would be:

```
for i=1:10
disp(i)
end
```

This code simply repeats code between *for* and *end* statements, *for* i=1,2,...,10. **"disp(i)"** prints out the value of the loop. **"end"** ends the section of code that is being repeated.

Another Example would be:

```
for i=0:5:100
disp(i)
end
```

for i = 0 : 5 : 100
where,
0 refers to lower limit
100 refers to upper limit &
5 refers to an increment b/w upper & lower limit

Nested Loops:

Nesting the loops means,"placing loops in one above another" or "executing one loop in other"
Example:

```
for a=10:10:50
    for b=0:1:10
        disp('this class is boring')
    end
end
```

Relational Operators:

< less than
> greater than
= equal to
<= less than equal to
>= greater than equal to
~= not equal to
& and
| or
~ not

While Loop:

The *while loop* repeats a sequence of commands, as long as some condition is met. We usually do not enter the number of times *while loop* has to be repeated.

Example:

```
n = 10;
while n > 0
disp('this class is boring')
n = n - 1;
end
```

Another example can be:

```
n = 1;
while n > 0
disp('i will work hard to pass in exams')
n = n + 1;
end
```

Laboratory Task:-

1. By using *for* loop,
    1. Make a program to print twenty numbers, starting from your Roll no. and onwards.
    2. Make a program to count backwards from 100 to 0.
    3. Make a table of nine (9) till thirty multiples.
    4. Make a program which squares any 10 numbers in successive order.
    5. Make a program which prints numbers from 1 to 20 every 10 times.

2. By using *while* loop,

Make a 2D grid of XY data points using nested while loops

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 03

To study and investigate Conditional Statements in Matrices using MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 03

Objective:-

To study and investigate Conditional Statements in matrices using MATLAB

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Theory:-

Conditional Statements:
In writing programs, we often need to make decisions based on the values of variables in memory. In order to accomplish it we generally use conditional statements.

***if* Structure:**

if (expression)
(statement)

Example:

```
num=input('press the number 2 key: ')
if (num == 2)
disp('the key pressed is 2')
else
disp('unrecognized key')
end
```

Laboratory Task:-

By using *if else* statement,

Create a number guessing game.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 04

To study and perform basic operations on digital images using MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 04

Objective:-

To study and perform basic operations on digital images using MATLAB

Equipment Required:-

- MATLAB
- Image Processing toolbox

Theory:-

Reading an Image

To import an image from any supported graphics image file format, in any of the supported bit depths, use the *imread* function.

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 Data

Imwrite write image to graphics file

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

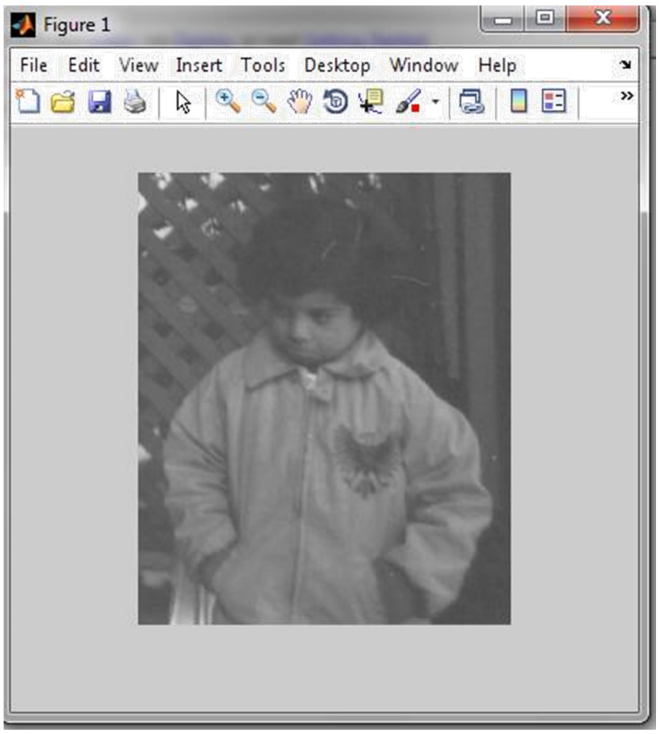


Figure 1 A grey scale image

Accessing the Pixel data

There is a one-to-one correspondence between pixel coordinates and the coordinates MATLAB uses for matrix subscripting. This correspondence makes the relationship between an image's data matrix and the way the image is displayed easy to understand. For example, the data for the pixel in the fifth row, second column is stored in the matrix element (5,2). You use normal MATLAB matrix subscripting to access values of individual pixels. For example, the MATLAB code

A(2,15)

returns the value of the pixel at row 2, column 15 of the image A.

Image Cropping

imcrop displays the image in a figure window and creates an interactive Crop Image tool associated with the image. The image can be a grayscale image, a truecolor image, or a logical array.

Example:

Read image into the workspace.

I = imread('cameraman.tif');

Then, Open Crop Image tool associated with this image. Specify a variable in which to store the cropped image. The example includes the optional return value *rect* in which imcrop returns the four-element position vector of the rectangle you draw.

[J, rect] = imcrop(I);

When you move the cursor over the image, it changes to a cross-hairs  . The Crop Image tool blocks the MATLAB command line until you complete the operation. Using the mouse, draw a rectangle over the portion of the image that you want to crop.

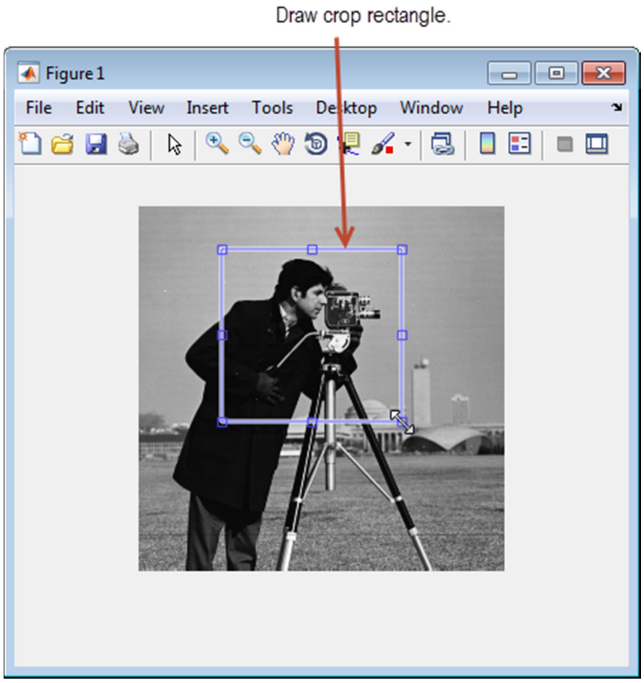


Figure 2. Image before cropping

Perform the crop operation by double-clicking in the crop rectangle or selecting Crop Image on the context menu.

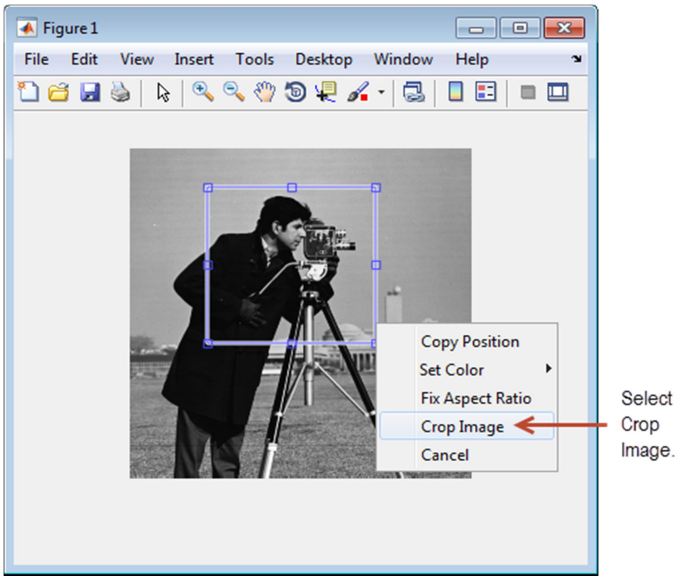


Figure 3. Image being cropped

The Crop Image tool returns the cropped area in the return variable, J. The variable *rect* is the four-element position vector describing the crop rectangle you specified.

Laboratory Tasks:

1. Import and display a grey scale image using MATLAB.
2. Write the same image file on to another graphics file.
3. Also access three different picture elements of the same image and display their values.
4. Perform cropping of the same image using MATLAB

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 05

To study and investigate basic operations on digital images and histogram using MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 05

Objective:-

To study and investigate basic operations on digital images and histogram using MATLAB

Equipment Required:-

- MATLAB
- Image Processing toolbox

Theory:-

Mirror Image Generation:
Horizontally flipping an image is called mirroring an image. This can be done using the following function:

Syntax:

imfliplr(Image)

The resultant image can be allocated to a variable.

Figure 1. Source Image and resultant image after performing horizontal flip

Inverting an Image:

Vertically flipping an image is called inverting an image. This can be done using the following function:

Syntax:

imflip(Image) or imflipud(Image)

The resultant image can be allocated to a variable.

Figure 2. Source Image with resultant image after vertical flip

Negative of an Image:

Image complement or image negative has significance application in the medical field. This is done by subtracting the pixel values in the image from the highest possible pixel value. The resultant image is a negative or complement of the source image.

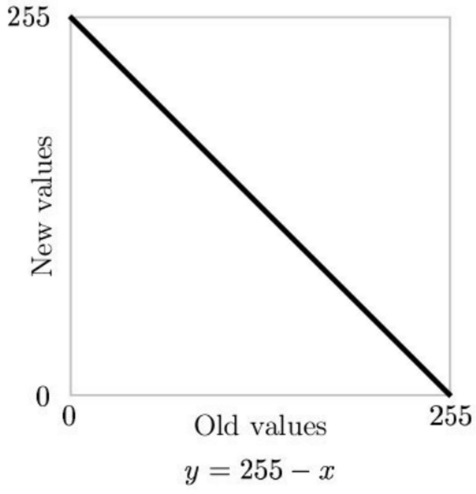


Figure 3. Old and new pixel values (complement)

Use imcomplement function to generate negative of an image

Syntax:

B = imcomplement(A)

Figure 4. Before and after taking complement (negative) of a grey scale image

Figure 5. Before and after taking complement (negative) of an RGB image

<u>Rotating an Image:</u>

Imrotate function used to rotate a grey scale or RGB Image.

Syntax:

B = imrotate(A,angle)

The value of angle is given in degrees and the image rotates in anti-clockwise direction according to the angle.

Figure 6. Before and after applying a +30 degrees rotation to an RGB image

RGB to Grey Scale conversion:

Sometimes it is more suited and convenient to process images in grey scale format rather than in color format. This saves time, memory and other resources (like bandwidth etc). In order to convert an RGB image to grey scale image we use the following function:

Syntax:

I= rgb2gray(B)

Figure 7. Before and after applying RGB to gray scale conversion

Histogram:

The (intensity or brightness) histogram shows how many times a particular grey level (intensity) appears in an image. For example, 0 - black, 255 – white

An image has low contrast when the complete range of possible values is not used. Inspection of the histogram shows this lack of contrast.

Syntax:

Imhist(I);

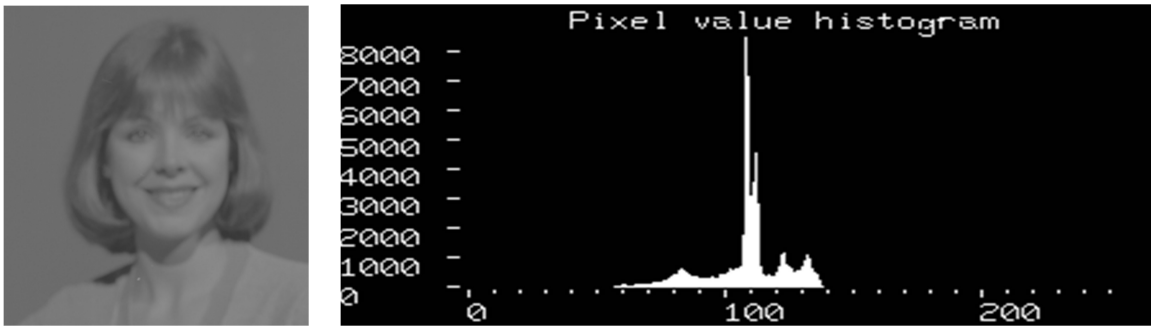


Figure 8. Image histogram of a low contrast grey scale image

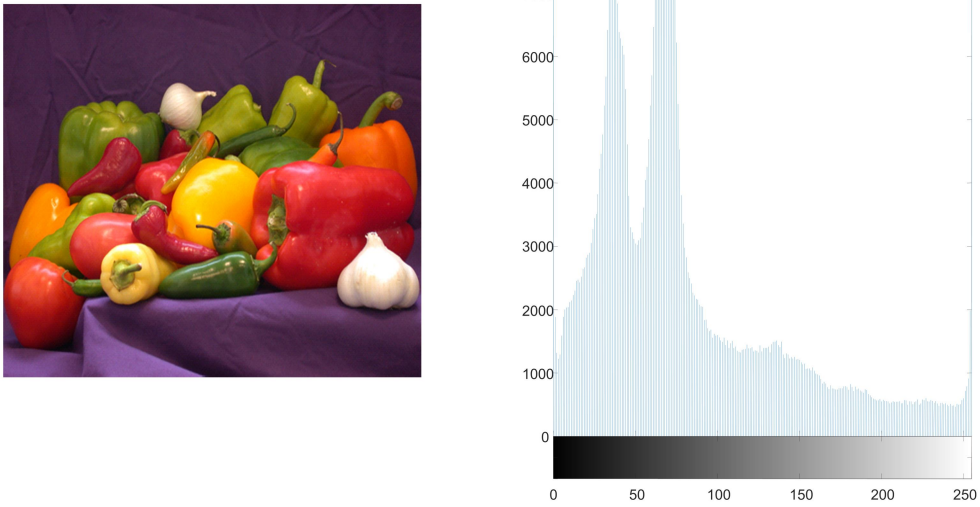


Figure 9. Image histogram of a high contrast RGB image

Laboratory Task:-

Applying the following operations on a grey scale image and an RGB image:

- Mirroring
- Inverting
- Negative
- Rotating at +30, -30, +90,-90 degrees
- Histogram

Also apply RGB to Grey Scale conversion of the RGB image

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 06

To study and perform contrast stretching, Histogram equalization and specification using MATLAB

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 06

Objective:-

To study and perform contrast stretching, Histogram equalization and specification/matching using MATLAB

Equipment Required:-

- MATLAB
- Image Processing toolbox

Theory:-

Histograms:

Given a grayscale image, it's histogram consists of the histogram of its gray levels; that is, a graph indicating the number of times each gray level occurs in the image. We can infer a great deal about the appearance of an image from its histogram. In a dark image, the gray levels would be clustered at the lower end. In a uniformly bright image, the gray levels would be clustered at the upper end. In a well contrasted image, the gray levels would be well spread out over much of the range.

Problem: Given a poorly contrasted image, we would like to enhance its contrast, by spreading out its histogram. There are two ways of doing this.

Histogram Stretching: (Contrast Stretching)

If we have a poorly contrasted image of range [a,b], we can stretch the gray levels in the center of the range out by applying a piecewise linear function. This function has the effect of stretching the gray levels [a,b] to gray levels [c,d], where a<c and d>b, according to the equation:

$$j = \frac{(c-d)}{(b-a)}\,(i-a)+c$$

Here, pixel values less than c are all converted to c, and pixel values greater than d are all converted to d.

Syntax:

imadjust(I,[a,b],[c,d])

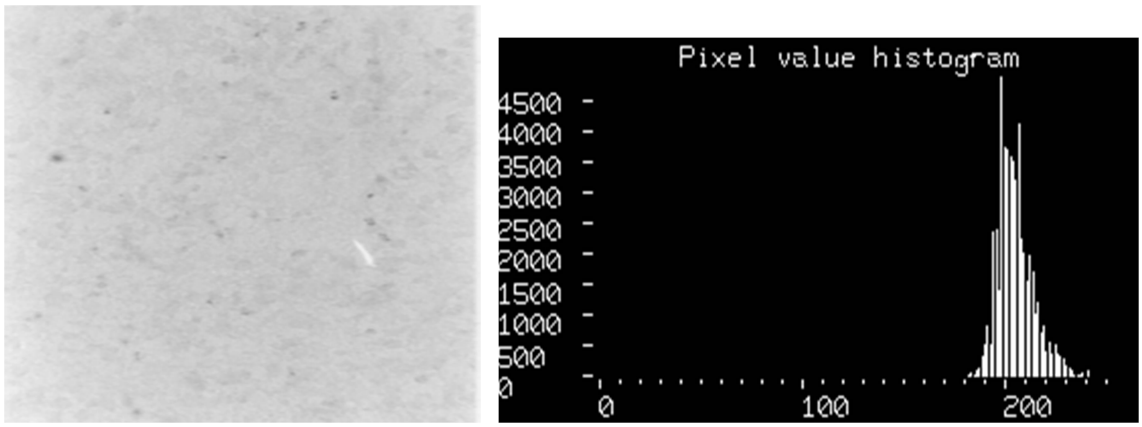


Figure 1. Before Contrast Stretching

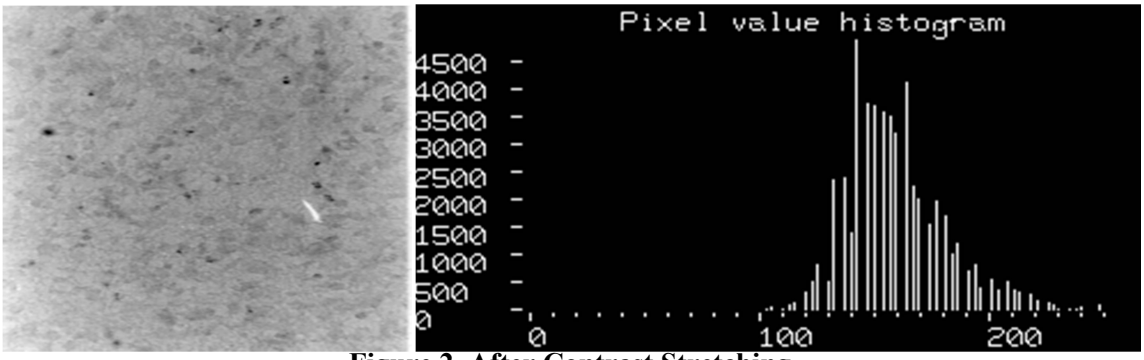


Figure 2. After Contrast Stretching

Figure 3. Another Before and After Contrast Stretching Example

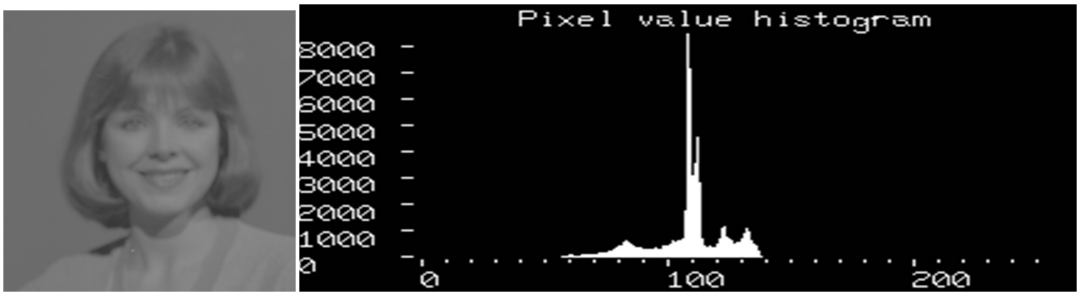


Figure 4. Before Contrast stretching

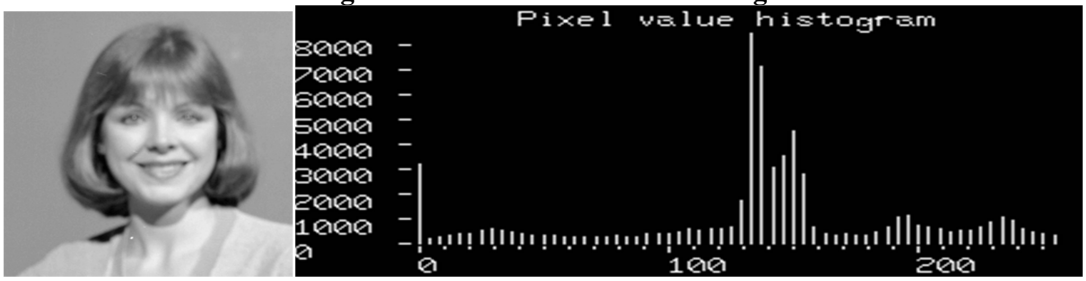


Figure 5. After Contrast Stretching

Histogram Equalization:

Intensity transformation functions based on information extracted from image intensity histograms play a basic role in image processing, in areas such as enhancement, compression, segmentation, and description. The Histogram equalization generates an image whose intensity levels are equally likely, and, in addition, cover the entire range [0, 1]. The net result of this intensity-level equalization process is an image with in-

creased dynamic range, which will tend to have higher contrast. Note that the transformation function is really nothing more than the cumulative distribution function (CDF).

Syntax:

g = histeq(f, nlev)

Where *f* is the input image and *nlev* is the number of intensity levels specified for the output image. If *nlev* is equal to *L* (the total number of possible levels in the input image), then *histeq* implements the transformation function, $T(r_k)$, directly. If *nlev* is less than L, then histeq attempts to distribute the levels so that they will approximate a flat histogram.

Unlike imhist, the default value in histeq is *nlev*=64. For the most part, we use the maximum possible number of levels (generally 256) for nlev because this produces a true implementation of the histogram-equalization method just described.

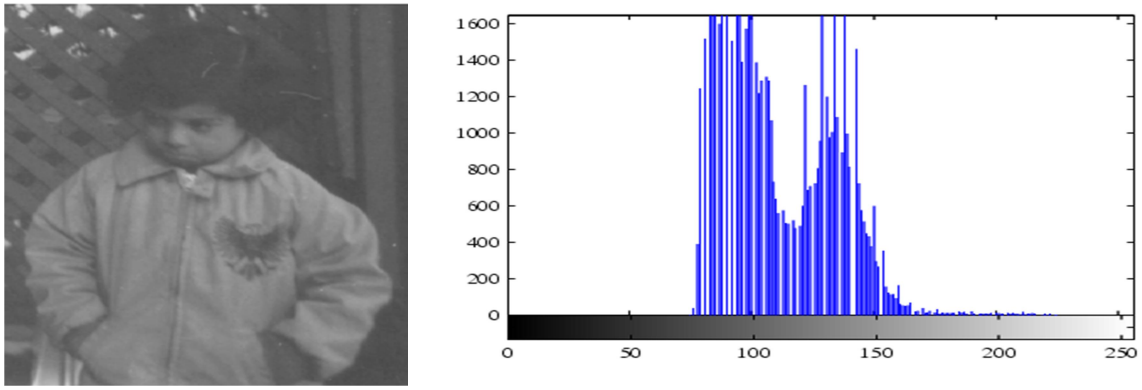


Figure 6. Before Histogram Equalization

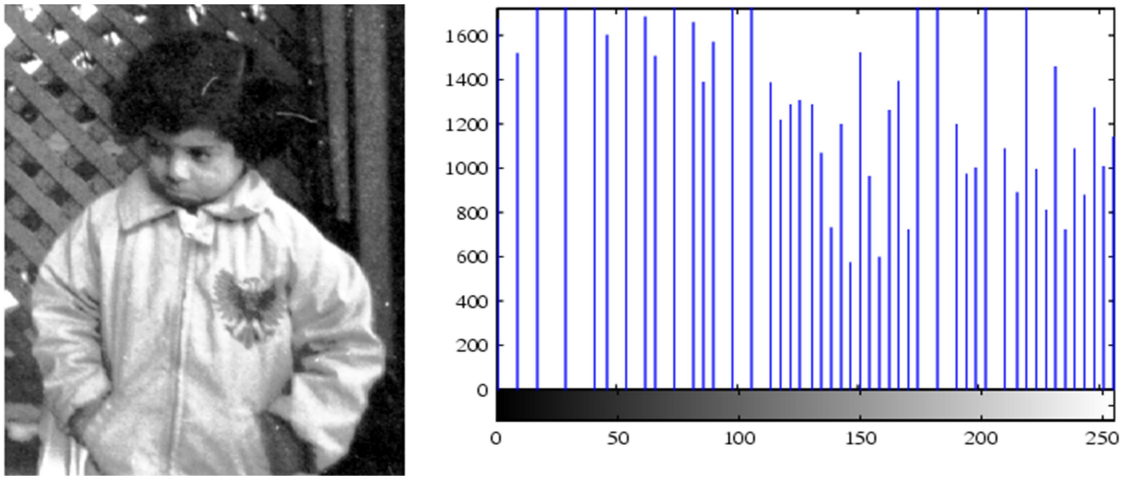


Figure 7. After Histogram Equalization

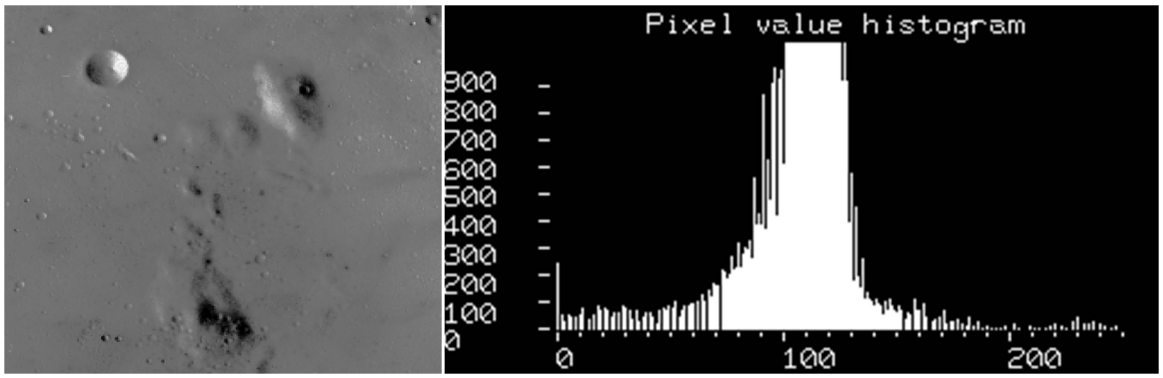


Figure 8. Before Histogram Equalization

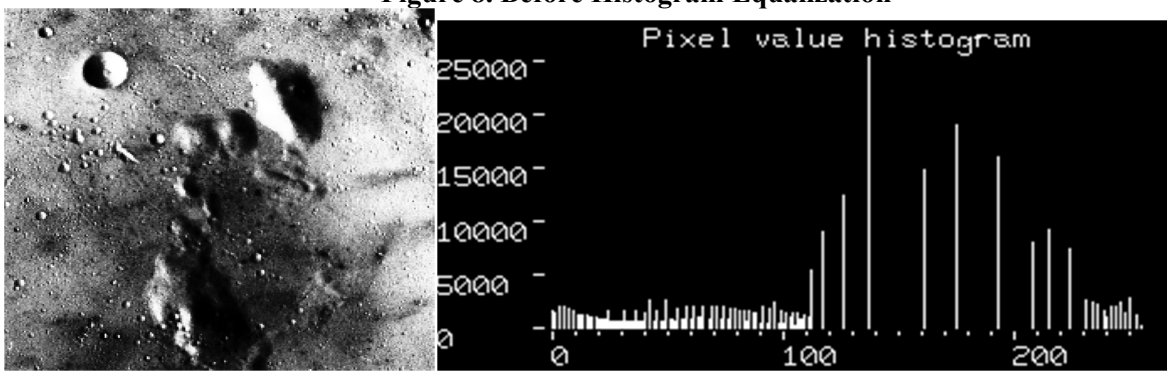


Figure 9. After Histogram Equalization

Histogram Specification:

Histogram equalization produces a transformation function that is adaptive, in the sense that it is based on the histogram of a given image. However, once the transformation function for an image has been computed, it does not change unless the histogram of the image changes. As noted earlier, histogram equalization achieves enhancement by spreading the levels of the input image over a wider range of the intensity scale. We show in this section that this does not always lead to a successful result. In particular, it is useful in some applications to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate a processed image that has a specified histogram is called **histogram matching or histogram specification**. In histogram equalization, the discrete implementation of the preceding method only yields an approximation to the specified histogram.

Syntax:

g = histeq(f, hspec)

Where *f* is the input image, *hspec* is the specified histogram (a row vector of specified values), and *g* is the output image, whose histogram approximates the specified histogram, *hspec*. This vector should contain integer counts corresponding to equally spaced bins. A property of *histeq* is that the histogram of *g* generally better matches *hspec* when length (hspec) is much smaller than the number of intensity levels in *f*. At first glance on image produced by histogram equalization, one might conclude that histogram equalization would be a good approach to enhance the image, so that details in the dark areas become more visible. However, the result in Figure below, obtained using the command

>> f1 = histeq(f, 256);

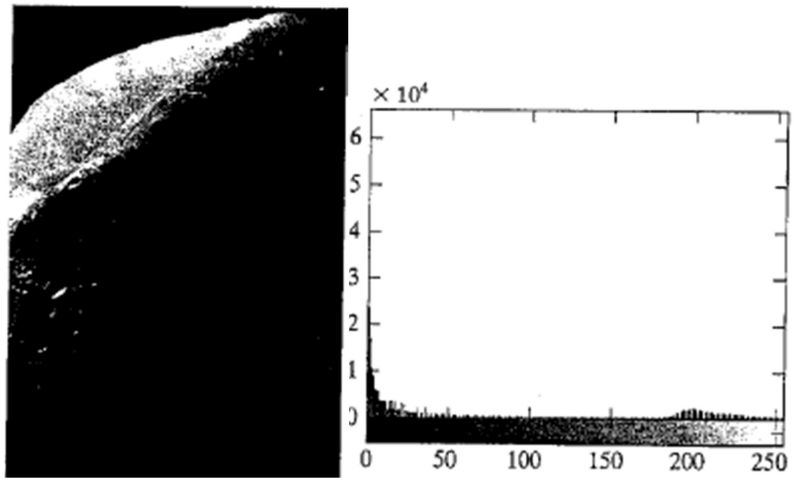


Figure 10. Image and it's histogram before histogram matching

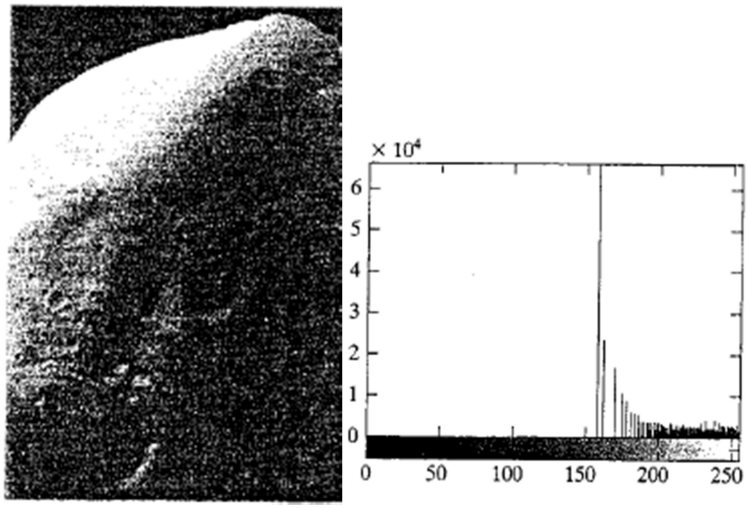


Figure 11. Image and it's histogram after applying histogram matching

shows that histogram equalization in fact did not produce a particularly good result in this case. The reason for this can be seen by studying the histogram of the equalized image, shown in the figure. Here, we see that that the intensity levels have been shifted to the upper one-half of the gray scale, thus giving the image a washed-out appearance. The cause of the shift is the large concentration of dark components at or near 0 in the original histogram. In turn, the cumulative transformation function obtained from this histogram is steep, thus mapping the large concentration of pixels in the low end of the gray scale to the high end of the scale.

One possibility for remedying this situation is to use histogram matching, with the desired histogram having a lesser concentration of components in the low end of the gray scale, and maintaining the general shape of the histogram of the original image.

We note from Fig.2 that the histogram is basically bimodal, with one large mode at the origin, and another, smaller, mode at the high end of the gray scale. These types of histograms can be modeled, for example, by using multimodal Gaussian functions. The M-function described in procedure section of this lab computes a bimodal Gaussian function normalized to unit area, so it can be used as a specified histogram.

Laboratory Tasks:-

1. Apply contrast stretching on any low contrast image
2. Import and save an image and display its image histogram. Also, apply histogram equalization keeping the following values of *nlev*s:
   a. 64 b. 128 c. 256
   For each case plot the resultant histogram and image
3. Perform histogram matching on an image using bi modal Guassian function.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 07

To study and perform spatial domain filtering, on 2D images, smoothening, sharpening and median filters using real-time image

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 07

Objective:-

To study and perform spatial domain filtering, on 2D images, smoothening, sharpening and median filters using real-time image

Equipment Required:-

- Image capturing device (camera, cell phone etc)
- Data transferring cable
- MATLAB
- Image Processing Toolbox

Theory:-

Spatial filtering:

Spatial or neighborhood processing consists of (1) defining a center point,(x,y); (2) performing an operation that involves only the pixels in a predefined neighborhood about that center point (3) letting the result of that operation be the "response" of the process at that point; and (4) repeating the process for every point in the image. The process of moving the center point creates new neighborhoods, one for each pixel in the input image. The two principal terms used to identify this operation are *neighborhood processing* and *spatial filtering,* with the second term being more prevalent. As explained in the following section, if the computations performed on the pixels of the neighborhoods are linear, the operation is called *linear spatial filtering* (the term *spatial convolution* also used); otherwise it is called *non-linear spatial filtering.*

Linear Spatial Filters

The toolbox supports a number of predefined 2-D linear spatial filters, obtained by using function fspecial, which generates a filter mask, w, using the syntax

Syntax:

w=fspecial(type= parameters)

Where type specifies the filter type, and parameters further define the specified filter. The spatial filter we are about to use in this lab is 'laplacian' and its applicable parameters are as described below:

'laplacian' fspecial ('laplacian' , alpha). A 3X3 Laplacian filter whose shape is specified by alpha, a number in the range (0,1]. The default value for alpha is 0.5.

Because the Laplacian is a derivative operator, it sharpens the linage but drives constant are as to zero. Adding the original image back restores the gray-level to nality. Function fspecial('laplacian', alpha) implements a more general Laplacian mask:

$$\begin{matrix} \frac{\alpha}{1+\alpha} & \frac{1-\alpha}{1+\alpha} & \frac{\alpha}{1+\alpha} \\ \frac{1-\alpha}{1+\alpha} & \frac{-4}{1+\alpha} & \frac{1-\alpha}{1+\alpha} \\ \frac{\alpha}{1+\alpha} & \frac{1-\alpha}{1+\alpha} & \frac{\alpha}{1+\alpha} \end{matrix}$$

Which allows fine tuning of enhancement results. Enhancement in this case consists of sharpening the image, while preserving as much of its gray tonality as possible.

Non-Linear Spatial Filtering:

A commonly used tool for generating nonlinear spatial filters in IPT is function ordfilt2, which generates *orderstatisticfilters* (also called *rank filters).* These are non-linear spatial filters whose response is based on ordering (ranking) the pixels contained in an image neighborhood and then replacing the value of the center pixel in the neighborhood with the value determined by the ranking result. Attention is focused in this section on non-linear filters generated by ordfilt2. The best-known order statistic filter in digital image processing is the '**median filter**' which corresponds to the 50^{th} percentile. The toolbox provides a specialized implementation of the 2-D median filter:

Syntax:

g = medfilt2 (f, [m n], padopt)

where the tuple *[m n]* defines a neighborhood of size *m x n* over which the median is computed, and *padopt* specifies one of three possible border padding options: 'zeros' (the default), 'symmetric' in which *f* is extended symmetrically by mirror-reflecting it across its border, and 'indexed', in which *f* is padded with 1s if it is of class double and with 0s otherwise. The default form of this function is

g = medfilt2(f)

which uses a 3 X 3 neighborhood to compute the median, and pads the border of the input with 0s. Median filtering is a useful tool for reducing salt-and-pepper noise in image.

Laboratory Tasks:-

1. After taking a real-time image from image capturing device, apply Laplacian filter on the image when:
   a. Alpha = 0
   b. Filter has -8 at the center

   Generate image plots as results to show the effect of the filter on the output

2. After converting the same image to grey scale, apply median filter on the image by:
   a. Introducing salt and pepper noise in the image
   b. Removing the noise by apply (1) a default median filter (2) median filter with one of the mentioned padding options (refer to the theory discussed above)

   Generate image plots as results to show the effect of the filter on the output

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 08

To study and perform image restoration techniques, inverse filtering and geometric transformation using real-time image

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 08

Objective:-

To study and perform image restoration techniques, inverse filtering and geometric transformation using real-time image

Equipment Required:-

- Image capturing device (camera, cell phone etc)
- Data transferring cable
- MATLAB
- Image Processing Toolbox

Theory:-

Image Restoration:

The objective of restoration is to improve a given image in some predefined sense. Although there are some areas of overlap between image enhancement and restoration, the former is largely a subjective process, while image restoration is for the most part an objective process. Restoration attempts to or recover an image that has been degraded by using some degradation phenomenon. Thus, restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image.

Inverse Filtering:

The simplest approach we can take to restoring a degraded image is to form an estimate of the form

$$F(u,v) = G(u,v)/H(u,v)$$

And then obtain the corresponding estimate of the image by taking the inverse Fourier transform of F(u, v) [where G(u, v) is the Fourier transform of degraded image, H(u,v) is the degrading function]. This approach is appropriately called '**inverse filtering'**.

Wiener filtering is implemented in IPT using function deconvwnr, which has three possible syntax forms. In all these forms, g denotes the degraded and fr is the restored image. The first syntax form,

fr = deconvwnr(g, PSF)

assumes that the noise-to-signal ratio is zero.

Thus, this form of the Wiener filter is the inverse filter mentioned in syntax

Syntax:

Fr = deconvwnr (g, PSF, NSPR)

assumes that the noise-to-signal power ratio is known, either as a constant or array; the function accepts either one. This is the syntax used to implement the parametric Wiener filter, in which case NSPR would be an interactive scalar input. Finally, the syntax

fr = deconvwnr (g, PSF, NACORR, FACORR)

assumes that autocorrelation functions, NACORR and FACORR, of the noise and under-graded image are known.

Geometric Transformation:

Geometric transformations are used frequently to perform image registration, a process that takes two images of the same scene and aligns them so that they can be merged for visualization, or for quantitative comparison. One of the most common form of spatial transformation is affine transformation. This transformation can scale, rotate, translate, or shear a set of points, depending on the values chosen for the elements of T. Table shows how to the values of the elements to achieve different transformations.

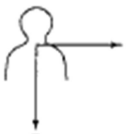
Type	Affine Matrix, T	Coordinate Equations	Diagram
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = z$	
Scaling	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = s_x w$ $y = s_y z$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w\cos\theta - z\sin\theta$ $y = w\sin\theta + z\cos\theta$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w + \alpha z$ $y = z$	
Shear (vertical)	$\begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = \beta w + z$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$	$x = w + \delta_x$ $y = z + \delta_y$	

Table 5.3 (Taken from Chapter 5 of book named "Digital Image Processing using Matlab"by Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins)

IPT represents spatial transformations using a so-called t-form structure. One way to create such a structure is by using function make t-form, whose calling syntax is:

Syntax:

maketform(transform_type, transform_parameters)

Affine transformation matrix in this case has the form

$$T = \begin{matrix} s\cos\theta & s\sin\theta & 0 \\ -s\sin\theta & s\cos\theta & 0 \\ \delta x & \delta y & 1 \end{matrix}$$

IPT function *imtransform* uses inverse mapping instead. An inverse ping procedure scans each output pixel in turn, computes the corresponding location in the input image using $T^{-1}\{(x, y)\}$, and interpolates among nearest input image pixels to determine the output pixel value. Inverse mapping is generally easier to implement than forward mapping.
The basic calling syntax for *imtransform* is

g = imtransform(f, tform, interp)

where *interp* is a string that specifies how input image pixels are interpolated to obtain output pixels; *interp* can be either 'nearest', 'bilinear', 'bicubic'. The *interp* input argument can be omitted, in which case it defaults to 'bilinear'.

Laboratory Task:-

1. After obtaining an RGB image from image capturing device, degrade the image by adding Gaussian noise, apply different deconvolution functions (mentioned in Theory section) in order to restore the original image.

2. Apply geometric transformation on a real-time acquired image using imtransform functions discussed in the Theory section.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 09

Apply Image Compression using Huffman Coding

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 09

Objective:-

Apply Image Compression using Huffman Coding

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Theory:-

Image Compression:

Image compression addresses the problem of reducing the amount of data required to represent a digital image. Compression is achieved by the removal one or more of three basic data redundancies: (1) coding redundancy, which present when less than optimal (i.e., the smallest length) code words are (2) inter-pixel redundancy, which results from correlations between the pixel of an image; and/or (3) psycho-visual redundancy, which is due to data that ignored by the human visual system (i.e., visually nonessential information).

Huffman Coding:

When coding the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run-lengths, and so on), Huffman codes contain the smallest possible number of code symbols (e.g., bits) per source symbol(e.g. gray-level value) subject to the constraint that the source symbols are coded one at a time.
The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, consists of the symbols 0 and 1. As the reduced source symbol with probability 0.5 was generated by combining two symbols in the reduced source to its left. The 0 used to code it is now assigned to both of these symbols and a 0 and 1 are arbitrarily appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source reached.

Laboratory Task:-

Write a MATLAB code for applying Huffman Encoding on a 2D image.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 10

To study and apply image segmentation techniques for point and line detection using real-time image

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 10

Objective:-

To study and apply image segmentation techniques for point and line detection using real-time image

Equipment Required:-

- Image capturing device (camera, cell phone etc)
- Data transferring cable
- MATLAB
- Image Processing Toolbox

Theory:-

Image Segmentation:

Segmentation subdivides an image into its constituent regions or objects. The level to which the subdivision is carried depends on the problem being solved. That is, segmentation should stop when the objects of interest in an application have been isolated.

Point, Line and Edge Detection:

In this lab, we will discuss techniques for detecting the three basic types of intensity discontinuities in a digital image: points, lines, and edges (we will continue edge detection in next lab). The most common way to look for discontinuities is to run a mask through the image.

Point Detection:

The detection of isolated points embedded in areas of constant or nearly constant intensity in an image is called Point detection. Point detection is implemented in MATLAB using function *imfilter*, with the mask. The important requirements are that the strongest response of a mask must be when the mask is centered on an isolated point, and that the response be 0 in areas of constant intensity.
If T is given, the following command implements the point detection approach just discussed:

>> g = abs(imfilter(double(f), w)) >= T;

where *f* is the input image, *w* is an appropriate point-detection mask, and *g* is the resulting image. Recall that *imfilter* converts its output to the class of the input, so we use double (f) in the filtering operation to prevent premature truncation of values if the input is of class *uint*8, and because the *abs* operation does accept integer data. The output image *g* is of class *logical*; its values are 0 and 1. If T is not given, its value often is chosen based on the filtered in which case the previous command string is broken down into three basic steps
(1) Compute the filtered image, abs (imfilter (double (f) , w)),
(2) find the value for T using the data from the filtered image, and
(3) compare the image against T.

Line Detection:

The next level of complexity is line detection. Consider the masks in figure 1. First mask were moved around an image, it would respond more strongly to the lines (one pixel thick oriented horizontally. With a constant background, maximum response would result when the line passed through the middle of the mask. Similarly, the second mask in fig 1.responds best to lines oriented at+45°; the third mask to detect vertical lines; and the fourth mask to lines in -45°direction. Note that the preferred direction of each mask is weighted with a larger coefficient (i.e. 2) than other possible directions. The values of coefficients of each mask sum to zero, indicating a zero response from the mask in areas of constant intensity.

Horizontal			**+45^0**			**Vertical**			**-45^0**		
-1	-1	-1	-1	-1	2	-1	2	-1	2	-1	-1
2	2	2	-1	2	-1	-1	2	-1	-1	2	-1
-1	-1	-1	2	-1	-1	-1	2	-1	-1	-1	2

Figure 1. Various masks to detect image features

Laboratory Task:-

1. Write a MATLAB code capable of detecting points in an image.
2. Write a MATLAB code capable of detecting horizontal, vertical lines and lines at 45 degrees lines in an image.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 11

To study and apply image segmentation techniques for edge detection using real-time image

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 11

Objective:-

To study and apply image segmentation techniques for edge detection using real-time image

Equipment Required:-

- Image capturing device (camera, cell phone etc)
- Data transferring cable
- MATLAB
- Image Processing Toolbox

Theory:-

Image Segmentation:

Segmentation subdivides an image into its constituent regions or objects .The level to which the subdivision is carried depends on the problem being solved. That is, segmentation should stop when the objects of interest in an application have been isolated. In continuation of previous lab here we will do edge detection.

Edge Detection:

Although point and line detection certainly are important in any discussion image segmentation, edge detection is by far the most common approach detecting meaningful discontinuities in intensity values. Such discontinuities are detected by using first and second-order derivatives. With the preceding discussion as background, the basic idea behind edge detection is to find places in an image where the intensity changes rapidly, using one of two general criteria:

- Find places where the first derivative of the intensity is greater in magnitude than a specified threshold
- Find places where the second derivative of the intensity has a zero

IPT's function **'edge'** provides several derivative estimators based on the criteria just discussed. For some of these estimators, it is possible to specify whether the edge detector is sensitive to horizontal or vertical edges or to both. The general syntax for this function is

Syntax:

[g, t] = edge (f, 'method', parameters)

Where *f* is the input image, *method* is one of the approaches listed in Table 1, and *parameters* are additional parameters . In the output, *g* is a logical array with 1s at the locations where edge points were detected in *f* and 0s elsewhere. Parameter *t* is optional; it gives the threshold used by edge to determine which gradient values are strong enough to be called edge points.

Edge Detector	Basic Properties
Sobel	Finds edges using Sobel approximation to derivatives

Prewitt	Finds edges using Prewitt approximation to derivatives
Roberts	Finds edges using Roberts approximation to derivatives
Laplacian of a Gaussian (LoG)	Finding edges by looking for zero crossings after filtering f(x,y) with Gaussian filter
Zero Crossing	Finding edges by looking for zero crossings after filtering f(x,y) with user specified filter
Canny	Finding edges by looking for local maxima of the gradient of f(x,y).The gradient is calculated using derivative of a Gaussian filter. The method uses two thresholds to detect strong and weak edges. Therefore, this method is more likely to detect true weak edges.

Table 1 Methods of edge detection

Laboratory Task:

Apply different methods (any three) of edge detection mentioned in Table 1 above to detect edges in a gray scale image.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 12

To study and apply image segmentation techniques for region based segmentation

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 12

Objective:-

To study and apply image segmentation techniques for region based segmentation

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Theory:-

Region based Segmentation:

The objective of segmentation is to partition an image into regions. In previous lab we approached this problem by finding boundaries between regions based on discontinuities in intensity levels. In this lab we will discuss segmentation techniques that are based on finding the regions directly.

Segmentation Using the Watershed Transform:

In geography, a watershed is the ridge that divides areas drained by different river systems. A catchment basin is the geographical area draining into a river or reservoir. The watershed transform applies these ideas to gray-scale image processing in a way that can be used to solve a variety of image segmentation problems. Understanding the watershed transform requires that we think of a gray scale image as a topological surface, where the values of f(x, y) are interpreted as heights. We can, for example, visualize the simple image in Fig.1 (a) as the three-dimensional surface in Fig. 1(b). If we imagine rain falling on this surface, it is clear that water would collect in the two areas labeled as catchment basins. Rain falling exactly on the labeled watershed ridgeline would be equally likely to collect in either of the two catchment basins. The watershed transform finds the catchment basins and ridge lines in a gray-scale image. In terms of solving image segmentation problems, the key concept is to change the starting image into another image whose catchment basins are the objects or regions we want to identify.

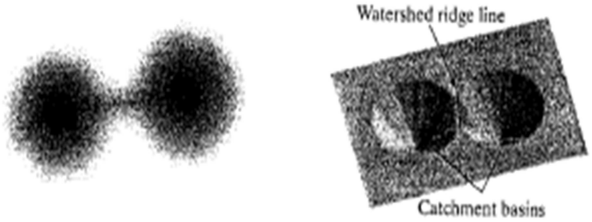


Figure 1 (a) and (b)

Watershed Segmentation Using the Distance Transform

A tool commonly used in conjunction with the watershed transform for segmentation is the distance transform. The distance transform of a binary image is a relatively simple concept: It is the distance from

every pixel to the nearest non-zero-valued pixel. Note that 1-valued pixels have a distance transform value of 0. The distance transform can be computed using IPT function bwdist, whose calling syntax is

D=bwdist(f)

Watershed Segmentation Using Gradients

The gradient magnitudes used often to preprocess a gray-scale image prior to using the watershed transform for segmentation. The gradient magnitude image has high pixel values along object edges, and low pixel values everywhere else. Ideally, then, the watershed transform would result in watershed ridgelines along object edges.

Marker-Controlled Watershed Segmentation

Direct application of the watershed transform to a gradient image usually leads to over segmentation due to noise and other local irregularities of the gradient. The resulting problems can be serious enough to render the result virtually useless. In the context of the present discussion, this means a large number of segmented regions. A practical solution to this problem is to limit the number of allowable regions by incorporating a preprocessing stage designed to bring additional knowledge into the segmentation procedure.

An approach used to control over segmentation is based on the concept of markers. A marker is a connected component belonging to an image. We like to have a set of internal markers, which are inside each of the objects of interest, as well as a set of external markers, which are contained within the background. These markers are then used to modify the gradient image using a procedure described in last part of code given in procedure. Various methods have been used for computing internal and external markers, many of which involve the linear filtering, nonlinear filtering, and morphological processing described in previous chapters. Which method we choose for a particular application is highly dependent on the specific nature of the images associated with that application.

Laboratory Task:

With the help of MATLAB, apply the above mentioned region based segmentation techniques on an RGB image.

Result:-

The code, results / output of all tasks must be attached with this lab.

LAB SESSION 13

To study and apply image segmentation techniques for region based segmentation

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 13

Objective:-

Video Conference using NetMeeting

Equipment Required:-

- Microsoft NetMeeting software
- Access to the internet

Theory:-

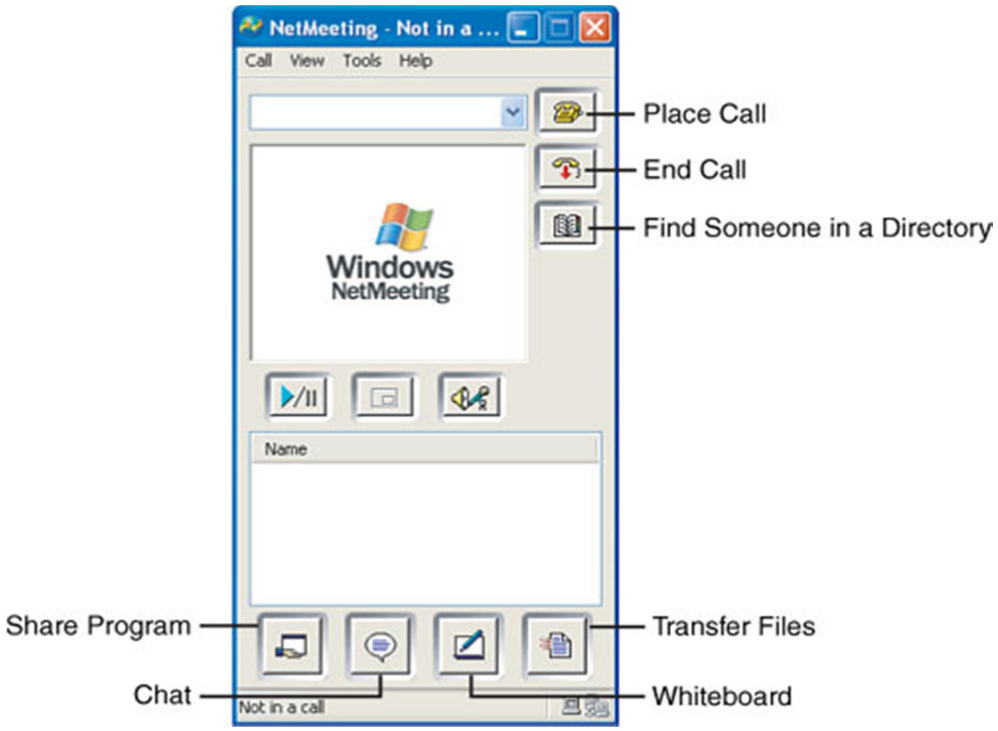
VIDEO CONFERENCING THROUGH MICROSOFT NETMEETING SOFTWARE:

Video conferencing becomes increasingly popular. Imagine that people can have a meeting through Internet without physically getting together! In this lab, we use Microsoft's NetMeeting8 software to have video conferencing over two kinds of network, namely, the Local Area Network and the Wide Area Network.

Download Microsoft's NetMeeting8 software from following website:
http://download.cnet.com/Microsoft-NetMeeting/3001-2381_4-10742128.html

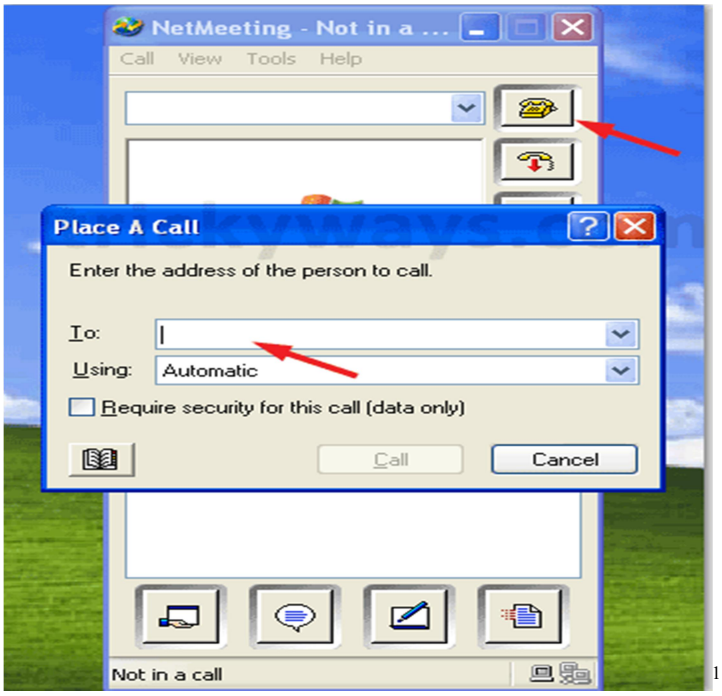
Installing NetMeeting software is very straightforward. A standard Windows setup program asks questions about where to place files and shortcuts, offering reasonable defaults, and then copies everything over.You also choose whether or not to register in the directory.

Following is the GUI of Microsoft's NetMeeting software:



Laboratory Task:

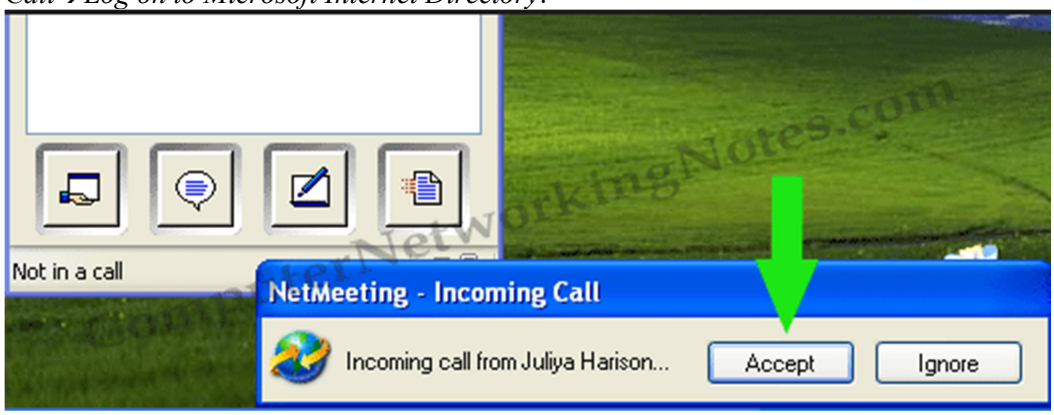
1. Over the Local Area Network (LAN): From *Call→New call*, call your partner's IP address[1] directly.



2. Over the Wide Area Network (WAN): Login to the server with your partner by

[1]

Call→Log on to Microsoft Internet Directory.



3. Compare the results of both video and audio part between LAN and WAN.

Result:-

The results / output of the task must be attached with this lab.

LAB SESSION 14

Open-ended lab: To apply JPEG compression on a gray scale image using DCT

Student Name: _____

Roll Number: _____ **Batch:** _____

Semester: _____ **Year:** _____

Total Marks	Marks Obtained

Remarks (If Any): _____

Instructor Name: _____

Instructor Signature: _____ **Date:** _____

LAB SESSION 14

Objective:-

Open-ended lab: To apply JPEG compression on a gray scale image using DCT

Equipment Required:-

- MATLAB
- Image Processing Toolbox

Laboratory Task:

Apply JPEG compression and display its effect on a gray scale image.

Result:-

The code, results / output of the task must be attached with this lab.