



Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

LABORATORY MANUAL

Course No. : CSE4228

Course Title: Digital Image Processing Lab

For the students of 4th Year, 2nd semester of
B.Sc. in Computer Science and Engineering program

TABLE OF CONTENTS

COURSE OUTCOMES	1
PREFERRED PROGRAMMING LANGUAGE/TOOLS	1
TEXT/ REFERENCE BOOKS	1
ADMINISTRATIVE POLICY OF THE LABORATORY	1
LIST OF SESSIONS	
SESSION 1	2
Introduction to Digital Image Processing and MATLAB	2
SESSION 2	8
Digital Image Fundamentals	8
SESSION 3	12
1. Intensity Transformations and Spatial filter	12
2. Filtering in spatial domain	17
3. Noise filtering	18
4. Template matching	19
5. Convolution	20
SESSION 4	21
Image Histogram	21
SESSION 5	23
Morphological Image Processing	23
SESSION 6	25
Image Derivatives and Edge Detection	25
MID TERM EXAMINATION	29
FINAL TERM EXAMINATION	29

COURSE OUTCOMES

- Understand the fundamental concepts of digital image processing, different algorithms and their application in image processing
- Apply different types of MATLAB functions and implement the algorithms, apply different kind of filters for edge detection and noise canceling
- Analyze real world problems and solving them with the help of image processing techniques, explain inherent logic behind the algorithms

This lab complements the *Digital Image Processing (CSE 4227)* course.

PREFERRED PROGRAMMING LANGUAGE/TOOLS

MATLAB

TEXT/ REFERENCE BOOKS

1. R.C. Gonzalez, R.E. Woods, and S.L. Eddins, *Digital Image Processing Using MATLAB (2nd Edition)*, 2009.
2. Scott E Umbaugh, *Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools, Second Edition*.

ADMINISTRATIVE POLICY OF THE LABORATORY

- Students must be performed class assessment tasks individually without help of others.
- Viva for each program will be taken and considered as a performance.
- Plagiarism is strictly forbidden and will be dealt with punishment.
- House Rules
 - No food or beverages are allowed in the lab.
 - Do not insert Flash drives, CD's or any other media into the computers.
 - Use of the Internet connectivity is restricted to web based mail accounts for data transfer and the coursewebsite.
- Work Submission and grading
 - Preliminary hard copy reports are to be submitted in the corresponding lab.
 - Final hard copy reports are to be submitted in the meeting following the lab.
 - Late submission will be awarded with penalty points.
 - Grading will be given according to completeness, clarity and quality of explanations. Since m-files will be supplied, higher emphasis will be given to understanding of the demonstrated principles.

Session 1

Introduction to Digital Image Processing and MATLAB

OBJECTIVES:

The objective of this lab session is

- Introduction of MATLAB
 - MATLAB workspace, Command Window, Variable panel
 - Working with variables, vectors and function
 - Working with library functions.
- Image processing with MATLAB
 - How to read an image in Matlab.
 - How to show an image in Matlab.
 - How to access Image Pixels in Matlab.
 - How to write Image in Matlab.
 - Mirror Image generation.
 - Flipped Image generation.

WHAT IS MATLAB?:

- MATLAB = Matrix Laboratory
- “MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++ and Fortran.”(www.mathworks.com)
- MATLAB is an interactive, interpreted language that is designed for fast numerical matrix calculations

THE MATLAB ENVIRONMENT:

The Figure 1.1 shows the MATLAB window components:

Workspace Displays all the defined variables

Command Window To execute commands in the MATLAB environment

Command History Displays record of the commands used

File Editor Window Define your functions

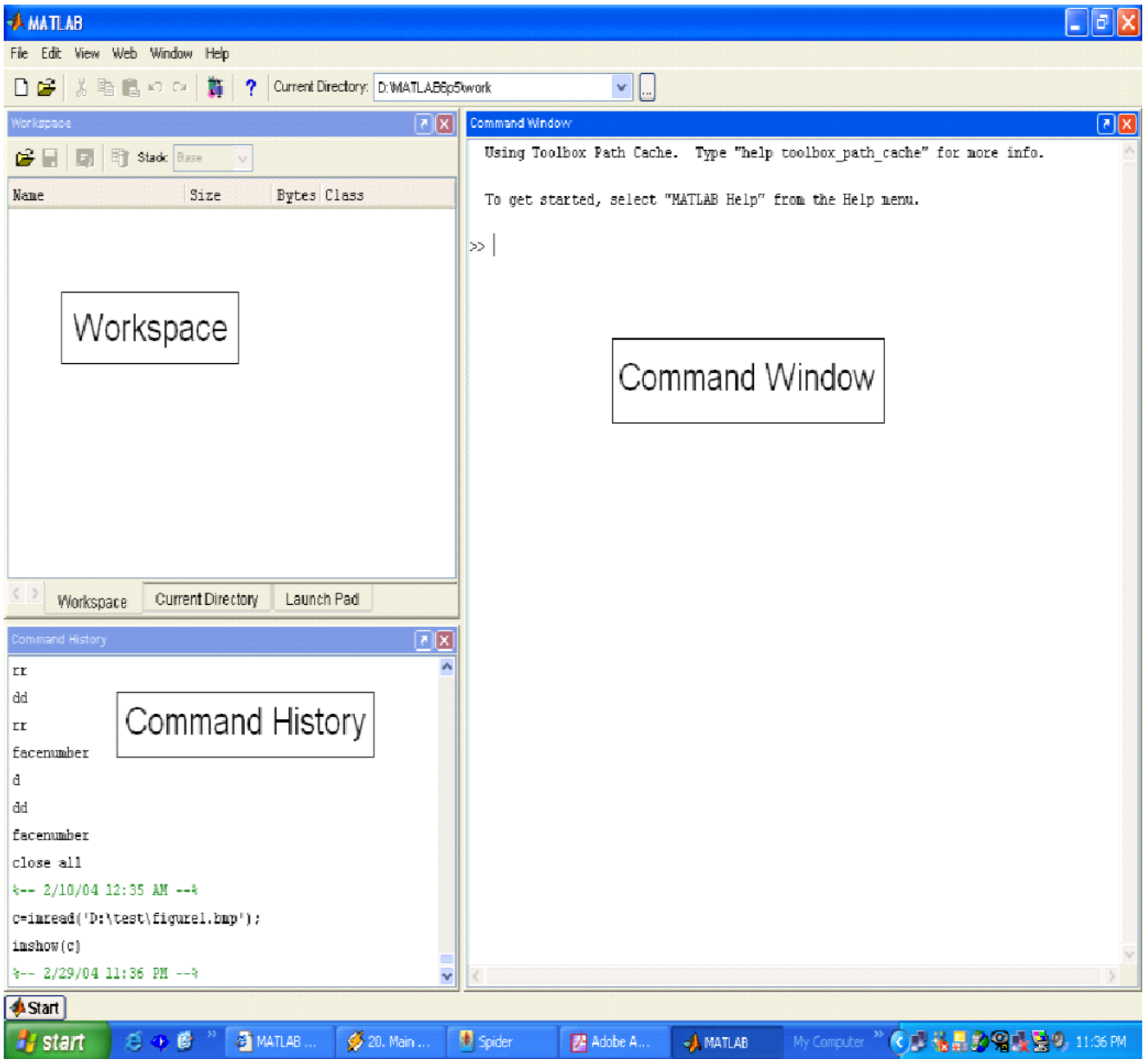
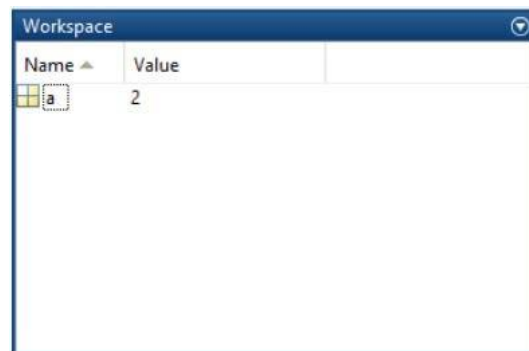
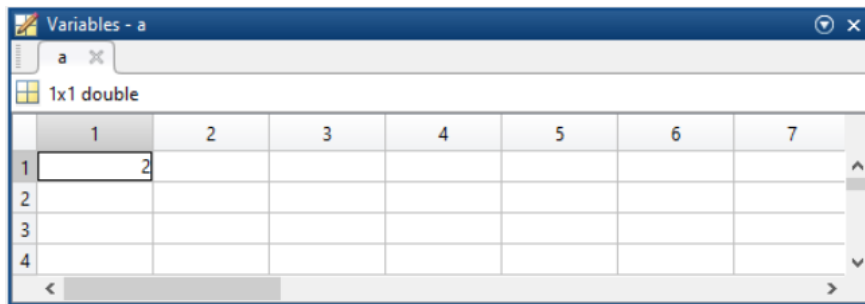


Figure 1.1: MATLAB Environment



(a)

Figure 1.2: Workspace



(b)

Figure 1.3: Variables

WORKSPACE AND VARIABLES: :

Let's start with variable, $a = 2$. After entering, the next line will display the variable with the assigned value.

$$a = 2$$

And you can see that, the variable is appeared in the workplace (as in Figure 1.2). If you double click on that variable icon, a panel named 'Variables' will popup (as in Figure 1.3).

MATLAB stores everything as matrix. You can note the indication of 1×1 double in the variable panel. It mean a is a 1 by 1 matrix (eventually a matrix with only one element), which is double type. By default any numerical data is double type. There are other data types as well such as `uint8`(unsigned integer 8 bit), `char` (character), `logical` (boolean).

WORKING WITH VECTOR AND MATRIX:

In MATLAB, the matrix element indices start from the top left corner. As we traverse right, we go through each column. And as we traverse down, we go through each row.

How to build a matrix?

$$A = [1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9];$$

Creates matrix A of size 3×3

Special matrices `zeros(n,m)`, `ones(n,m)`, `rand()`

Basic Operations on Matrices All operators in MATLAB are defined on matrices: `+`, `-`, `*`, `/`, `^`, `sqrt`, `sin`, `cos`, etc.

Element-wise operators are defined with a preceding dot: `.*`, `./`, `.^`

Matrix Functions `size(A)` – size vector, `sum(A)` – columns sums vector, `sum(sum(A))` – sum of all the elements.

Accessing elements of a matrix To access we use the following format of the command: `Matrix(which_row, which_column)`.

Assigning elements of a matrix

$$M(2, 2) = 99$$

$$M =$$

```

1  2  3  4
5 99  7  8
9  1  2  3

```

Question 1. Now, can you generate a matrix with random numbers greater than or equal to 1 using one single line of command?

You can apply element wise operations. To perform that, we need to put a dot(.) in front of the operator. For example, the following command will do element wise multiplication, that means 1st element of A will be multiplied with the 1st element of B, 2nd of A will be multiplied with 2nd element of B, and so on.

$$C = A . * B$$

$$C =$$

```

9  16  21
24 25  24
21 16  9

```

Question 2. Now, can you square every element of the matrix A using one single line of command?

Question 3. Also, can you square each elements of both A and B, add the squared elements of A with the squared elements B and store them in the matrix C? (Again, using one single line of command)

HOW TO DISPLAY AN IMAGE? :

To display image, use the `imshow` function.

Syntax `imshow(A)`

Description `imshow(A)` displays the image stored in array A.

HOW TO WRITE AN IMAGE DATA?:

Use `imwrite` to write image to graphics file

Syntax `imwrite(A, filename, fmt)`

Example Program-code 1.1 shows how to write an image file. Figure 1.4 shows the output.

```

1 a=imread('pout.tif');
2 imwrite(a, gray(256), 'b.bmp');
3 imshow('b.bmp')% imshow is used to display image

```

Program 1.1: Writing image to disk

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.

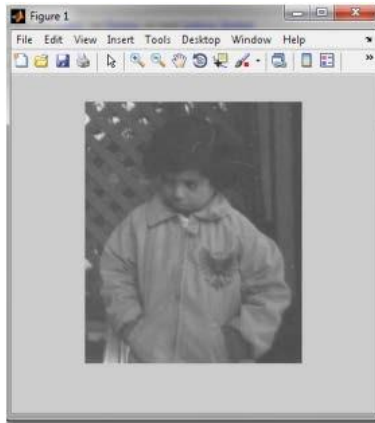


Figure 1.4: Writing image to disk file

MIRROR IMAGE GENERATION:

```

1 % this program produces mirror image of the image passed to it n also
2 % displays both the original and mirror image
3 a=imread('pout.tif');
4 [r,c]=size(a);
5 for i=1:1:r
6     k=1;
7     for j=c:-1:1
8         temp=a(i,k);
9         result(i,k)=a(i,j);
10        result(i,j)=temp;
11        k=k+1;
12    end
13 end
14 subplot(1,2,1),imshow(a)
15 subplot(1,2,2),imshow(result)

```

Program 1.2: Mirror Image Generation



Figure 1.5: Source Image and Mirrored Output

Task 1. Write a MATLAB code that reads a gray scale image and generates the flipped image of original image. Your output should be like the one given in Figure 1.6.

Task 2. Write a MATLAB code that will do the following:



Figure 1.6: Source Image with Desired Output

1. Read any gray scale image.
2. Display that image.
3. Again display the image such that the pixels having intensity values below than 50 will display as black and pixels having intensity values above than 150 will display as white. And the pixels between these will display as it is.

REFERENCES:

- https://www.mathworks.com/help/matlab/learn_matlab/desktop.html
- <https://www.mathworks.com/help/matlab/elementary-math.html>
- https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html
- https://www.mathworks.com/help/matlab/functionlist.html?s_cid=doc_ftr
- <https://www.mathworks.com/help/matlab/2-and-3d-plots.html>

Session 2

Digital Image Fundamentals

OBJECTIVES:

The objective of this lab is to understand

1. The effect of changing the number of gray levels on the quality of images.
2. The effect of changing spatial resolution on the quality of images, using two methods
 - (a) Nearest neighbor interpolation.
 - (b) Bilinear interpolation
3. Basic Relationship of Pixels i.e. Connectivity based on
 - (a) 4-Adjacency
 - (b) 8-Adjacency

CHANGING THE NUMBER OF GRAY LEVELS:

The quality of a gray-level image is significantly affected by its gray-level resolution. Other words, increasing the number of bits per pixel has a great effect in improving the quality of gray level images. This is because that a higher number of gray levels would give a smooth transition along the details of the image and hence improving its quality to the human eye.

EXAMPLE PROGRAM :

```
1 % Changing the Gray Resolution From 256 to 2
2 I = imread('cameraman.tif');
3 K= imfinfo('cameraman.tif');
4 if(K.BitDepth ==24)
5     I=rgb2gray(I);
6 end
7 [r,c] = size(I);
8 I2= uint8(zeros(r,c));
9 for i = 1:r
10     for j=1:c
11         if (I(i,j)>128)
12             I2(i,j) =256;
13         else
14             I2(i,j) =1;
15         end
16     end
17 end
18 figure,
19 subplot(121), imshow(I);
20 subplot(122), imshow(I2);
```

Program 2.1: Changing the number of gray Levels

OUTPUT:



Figure 2.1: Source Image with Desired Output

CHANGING THE SPATIAL RESOLUTION:

Changing the spatial resolution of a digital image, by zooming or shrinking, is an operation of great importance in a wide range of applications (i.e. in digital cameras, biomedical image processing and astronomical images). Simply, zooming and shrinking are the operations of over-sampling and under-sampling a digital image, respectively. Zooming a digital image requires two steps: the creation of new pixel locations, and assignment of gray levels to those new locations. The assignment of gray levels to the new pixel locations is an operation of great challenge. It can be performed using two approaches:

Nearest Neighbor Interpolation each pixel in the zoomed image is assigned the gray level value of its closest pixel in the original image.

Bilinear Interpolation the value of each pixel in the zoomed image is a weighted average of the gray level values of the pixels in the nearest 2-by-2 neighborhood, in the original image.

EXAMPLE PROGRAM :

```
1 I = imread ( 'cameraman . tif ' );
2 K = imfinfo ( 'cameraman . tif ' );
3
4 if (K. BitDepth ==24)
5     I = rgb2gray ( I );
6 end
7 [r , c] = size ( I );
8 I2 ( 1 : r / 2 , 1 : c / 2 ) = I ( 1 : 2 : r , 1 : 2 : c );
9
10 figure ,
11 subplot ( 1 2 1 ) , imshow ( I );
12 subplot ( 1 2 2 ) , imshow ( I2 );
```

Program 2.2: Reducing the Spatial Resolution

Figure 2.2 shows the output of Program-code 2.2

SOME USEFUL MATLAB FUNCTIONS:

- imagesc
- colormap
- imfinfo
- BitDepth
- imread



Figure 2.2: Source Image with Desired Output

- imshow
- rgb2gray
- im2bw
- zeros
- magic

BASIC RELATIONSHIP OF PIXELS:

Basic Relationship of Pixels Connectivity are based on

4-Adjacency A pixel p at (x, y) has 2 horizontal and 2 vertical neighbors:

- $(x + 1, y)$
- $(x - 1, y)$
- $(x, y + 1)$
- $(x, y - 1)$

8-Adjacency A pixel p at (x,y) has 2 horizontal , 2 vertical neighbors and pixels at diagonals:

- $(x + 1, y + 1)$
- $(x + 1, y - 1)$
- $(x - 1, y + 1)$
- $(x - 1, y - 1)$

Task 1. Reducing the Number of Gray Levels in an Image Write a computer program capable of reducing the number of gray levels in a image from 256 to 2, in integer powers of 2. The desired number of gray levels needs to be a variable input to your program.

Task 2. Zooming and Shrinking Images by Nearest Neighbor Write a computer program capable of zooming and shrinking an image by nearest neighbor algorithm. Assume that the desired zoom/shrink factors are integers. You may ignore aliasing effects.

Task 3. Zooming and Shrinking Images by Bilinear Interpolation Write a computer program capable of zooming and shrinking an image by bilinear interpolation. The input to your program is the desired size of the resulting image in the horizontal and vertical direction. You may ignore aliasing effects.

Task 4. Write a program to find the sets of connected component in binary image based on 4-adjacent neighbors.

Task 5. Write a program to find the sets of connected component in a binary image based on 8-adjacent neighbors.

NOTE:

Binary images can be created in mspaint or any other tool of your choice

Session 3

1. Intensity Transformations and Spatial filter

OBJECTIVES:

The objective of this lab is to understand & implement

1. Image enhancement in spatial domain through Gray level Transformation function
2. Linear Transformation
 - Image Negation function
 - Identity function
3. Logarithmic Transformation
4. Power Law Transformation
5. Piece Wise Linear Transformation

BACKGROUND MATERIAL:

Image Enhancement in Spatial Domain -Basic Grey Level Transformations Image enhancement is a very basic image processing task that defines us to have a better subjective judgement over the images. And Image Enhancement in spatial domain (that is, performing operations directly on pixel values) is the very simplistic approach. Enhanced images provide better contrast of the details that images contain. Image enhancement is applied in every field where images are ought to be understood and analysed. For example, Medical Image Analysis, Analysis of images from satellites, etc.

Image enhancement simply means, transforming an image f into image g using T . Where T is the transformation. The values of pixels in images f and g are denoted by r and s , respectively. As said, the pixel values r and s are related by the expression,

$$s = T(r)$$

where T is a transformation that maps a pixel value r into a pixel value s . The results of this transformation are mapped into the grey scale range as we are dealing here only with grey scale digital images. So, the results are mapped back into the range $[0, L - 1]$, where $L = 2^k$, k being the number of bits in the image being considered. So, for instance, for an 8-bit image the range of pixel values will be $[0, 255]$.

There are three basic types of functions (transformations) that are used frequently in image enhancement. They are,

1. Linear
2. Logarithmic
3. Power-Law

The transformation map plot shown in Figure 3.1 depicts various curves that fall into the above three types of enhancement techniques.

The Identity and Negative curves fall under the category of linear functions. Identity curve simply indicates that input image is equal to the output image. The Log and Inverse-Log curves fall under the category of Logarithmic functions and n^{th} root and n^{th} power transformations fall under the category of Power-Law functions.

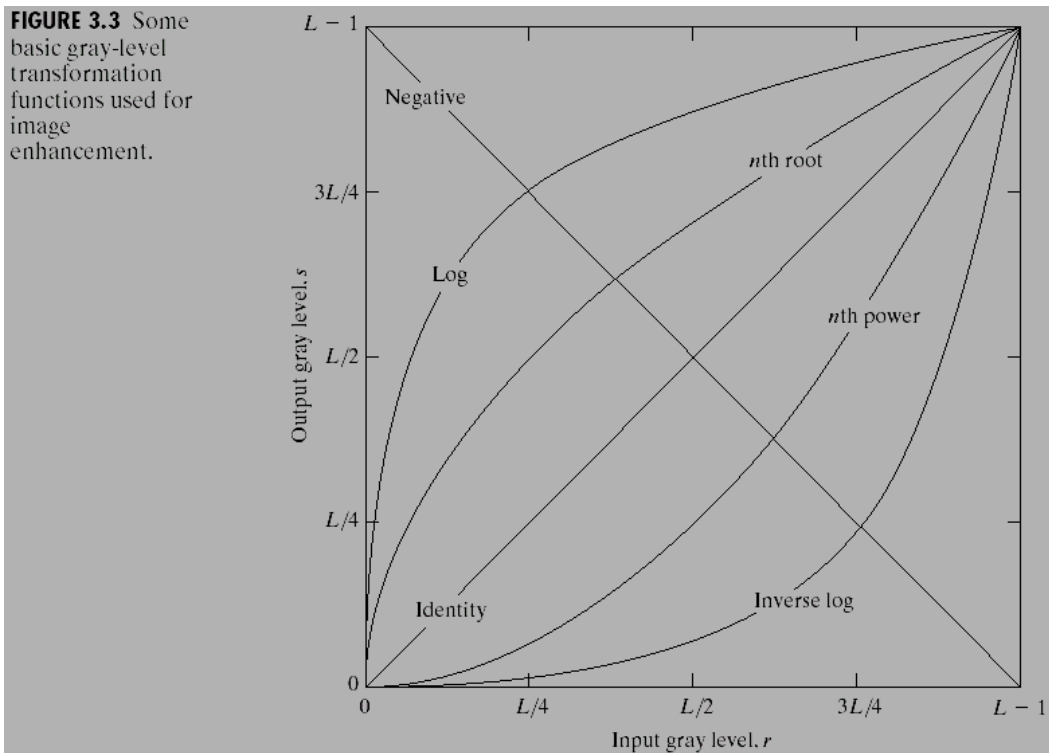


Figure 3.1: Plot of various transformation functions

IMAGE NEGATION FUNCTION:

The negative of an image with grey levels in the range $[0, L - 1]$ is obtained by the negative transformation shown in Figure 3.2, which is given by the expression,

$$s = L - 1 - r$$

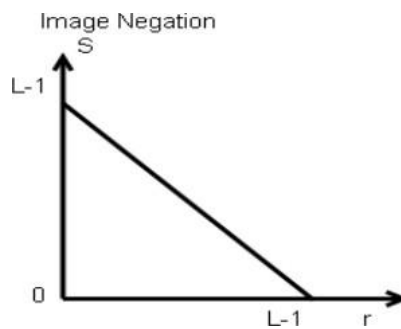


Figure 3.2: Image Negation Transformation

This expression results in reversing of the grey level intensities of the image thereby producing a negative like image. The output of this function can be directly mapped into the grey scale lookup table consisting values from 0 to $L - 1$.

ALGORITHM:

1. Read input image
2. Read maximum gray level pixel of input image
3. Replace input image by $(\text{maximum} - \text{input}) = \text{output}$
4. Display output image

Question 1. Explain application of image negation.

THRESHOLDING OF AN IMAGE:

Thresholding is a simple process to separate the interested object from the background. It gives the binary image. The formula for achieving thresholding is as follows

$$s = 0; \text{ if } r \leq t$$

$$s = L - 1; \text{ if } r > t$$

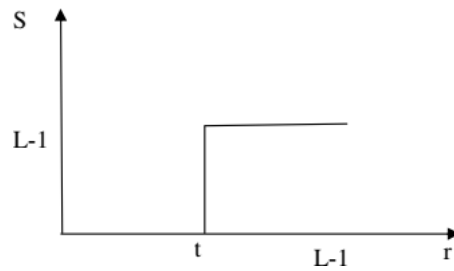


Figure 3.3: Thresholding

ALGORITHM:

1. Read input image
2. Enter thresholding value t
3. If image pixel is less than t replace it by 0.
4. If image pixel is $> t$ replace it by 255
5. Display input image
6. Display threshold image
7. Write input image
8. Write threshold image

CONCLUSION:

Thresholding separate out the object from the background

Question 2. Explain local & global thresholding

Question 3. Discuss some application of thresholding.

LOG TRANSFORMATIONS:

The log transformation curve shown in Figure 3.1 is given by the expression,

$$s = c \log(1 + r)$$

where c is a constant and it is assumed that $r \geq 0$. The shape of the log curve in Figure 3.1 tells that this transformation maps a narrow range of low-level grey scale intensities into a wider range of output values. And similarly maps the wide

range of high-level grey scale intensities into a narrow range of high level output values. The opposite of this applies for inverse-log transform. This transform is used to expand values of dark pixels and compress values of bright pixels.

POWER-LAW TRANSFORMATIONS:

The n^{th} power and n^{th} root curves shown in Figure 3.1 can be given by the expression,

$$s = cr^\gamma$$

This transformation function is also called as gamma correction. For various values of γ different levels of enhancements can be obtained. This technique is quite commonly called as Gamma Correction. If you notice, different display monitors display images at different intensities and clarity. That means, every monitor has built-in gamma correction in it with certain gamma ranges and so a good monitor automatically corrects all the images displayed on it for the best contrast to give user the best experience.

The difference between the log-transformation function and the power-law functions is that using the power-law function a family of possible transformation curves can be obtained just by varying the λ .

POWER-LAW TRANSFORMATION PROGRAM:

```

1 image=imread('pout.tif');
2 figure;
3 imshow(image);
4 image_double=im2double(image);
5 [r c]=size(image_double);
6 cc=input('Enter the value for c==>');
7 ep=input('Enter the value for gamma==>');
8 for i=1:r
9     for j=1:c
10        imout(i,j)=cc*power(image_double(i,j),ep);
11    end
12 end
13 figure,imshow(imout);

```

Program 3.1: Power-law transformation

Enter value $c = 1$ and $\gamma = .2\%$, it will make input image brighter as shown in Figure 3.4

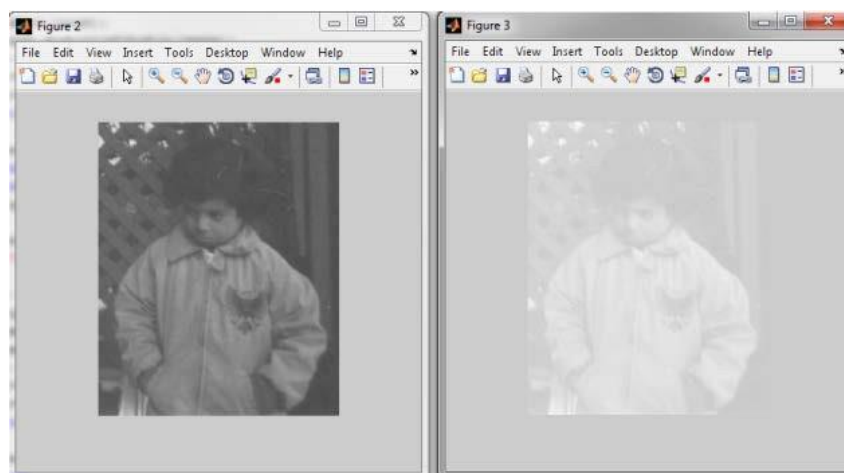


Figure 3.4: Power-Law Transformation with $\gamma < 1$

Enter value $c = 1$ and $\gamma = 5\%$, it will make input image darker as shown in Figure 3.5



Figure 3.5: Power-Law Transformation with $\gamma > 1$

BIT PLANE SLICING:

This transformation involves determining the number of usually significant bits in an image. In case of a 8 bit image each pixel is represented by 8 bits. Plane 0 contains all the lowest order bits in the bytes comprising the pixels in the image & plane 7 contains all the high order bits. The higher order bits contain usually significant data and the other bit planes contribute to more subtle details in the image. Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image.

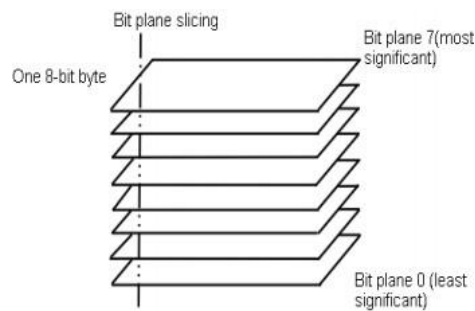


Figure 3.6: Bit plane splicing

ALGORITHM:

1. Read i/p image
2. Use bit and operation to extract each bit
3. Do the step 2 for every pixel.
4. Display the original image and the biplanes formed by bits extracted

Question 4. Explain the importance of bit plane slicing in image enhancement & image compression.

Task 1. Implement negation transform.

Task 2. Implement Logarithmic transform.

Task 3. Implement Piece wise linear transform.

2. Filtering in spatial domain

LOW PASS FILTERING:

Low pass filtering as the name suggests removes the high frequency content from the image. It is used to remove noise present in the image. Mask for the low pass filter is : One important thing to note from the spatial response is that all the coefficients are positive. We could also use 5×5 or 7×7 mask as per our requirement. We place a 3×3 mask on the image . We start from the left hand top corner. We cannot work with the borders and hence are normally left as they are. We then multiply each component of the image with the corresponding value of the mask. Add these values to get the response. Replace the center pixel of the o/p image with these response. We now shift the mask towards the right till we reach the end of the line and then move it downwards. Low-pass filter kernel is:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

ALGORITHM:

1. Read I/p image
2. Ignore the border pixel
3. Apply low pass mask to each and every pixel.
4. Display the o/p image

CONCLUSION:

Low pass filtering makes the image blurred.

Question 5. Explain weighted average filter.

HIGH PASS FILTERING:

High pass filtering as the name suggests removes the low frequency content from the image. It is used to highlight fine detail in an image or to enhance detail that has been blurred. Mask for the high pass filter is:

$$\frac{1}{9} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

One important thing to note from the spatial response is that sum of all the coefficients is zero.

We could also use 5×5 or 7×7 mask as per our requirement. We place a 3×3 mask on the image . We start from the left hand top corner. We cannot work with the borders and hence are normally left as they are. We then multiply each component of the image with the corresponding value of the mask. Add these values to get the response. Replace the center pixel of the o/p image with these response. We now shift the mask towards the right till we reach the end of the line and then move it downwards.

ALGORITHM:

1. Read I/p image
2. Ignore the border pixel
3. Apply high pass mask to each and every pixel.
4. Display the o/p image

CONCLUSION:

High pass filtering makes the image sharpened.

Question 6. Show that high pass = Original – low pass .

MEDIAN FILTERING:

Median filtering is a signal processing technique developed by tukey that is useful for noise suppression in images. Here the input pixel is replaced by the median of the pixels contained in the window around the pixel. The median filter disregards extreme values and does not allow them to influence the selection of a pixel value which is truly representative of the neighborhood.

ALGORITHM:

1. Read i/p image
2. Add salt and pepper noise in the image
3. Use 3×3 window.
4. Arrange the pixels in the window in ascending order.
5. Select the median.
6. Replace the center pixel with the median.
7. Do this process for all pixels.
8. Display the o/p image.

CONCLUSION:

Median filtering works well for impulse noise but performs poor for Gaussian noise.

Question 7. Explain “Median filter removes the impulse noise” with example

Task 4. Write a program to implement Smoothing Spatial filter and note the effects on given images.

Task 5.

Write a program to implement order statistics filters and write down your observations.

3. Noise filtering

OBJECTIVES:

- Learn about different kinds of noises
- Learn about different kind of filters
- Learn about different kinds of noise reduction techniques using filters

DIFFERENT KIND OF NOISE:

Salt and pepper noise Random dark (pepper) and bright (salt) pixels in an image

Gaussian Noise Noises that are distributed with Gaussian probability distribution

DIFFERENT KINDS OF FILTERS:

Max Filter Chooses the maximum pixel value from the window as the center value. Good for reducing pepper noises (why?)

Min Filter Chooses the minimum pixel value from the window as the center value. Good for reducing salt noises (why?)

Median Filter Chooses the median pixel value from the window as the center value. Good for reducing both salt and pepper noises (why?)

Average Filter Chooses the average pixel value from the window as the center value. Good for reducing noises and sharpness caused by random pixels- but reduces the pixel intensities. It is a linear filter. The kernel is:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian Filter Chooses the Gaussian weighted average pixel value from the window as the center value. Good for reducing noises and sharpness caused by random pixels- gives better performance than average filter. It is a linear filter. The kernel is:

$$\begin{bmatrix} 0.0000 & 0.0004 & 0.0017 & 0.0004 & 0.0000 \\ 0.0004 & 0.0275 & 0.1102 & 0.0275 & 0.0004 \\ 0.0017 & 0.1102 & 0.4421 & 0.1102 & 0.0017 \\ 0.0004 & 0.0275 & 0.1102 & 0.0275 & 0.0004 \\ 0.0000 & 0.0004 & 0.0017 & 0.0004 & 0.0000 \end{bmatrix}$$

Task 6. Learn the use of `imnoise()` function of Matlab and introduce noises in an image.

Task 7. Run min, max, median filters on noisy images and notice the output

Task 8. Run average and Gaussian filters on noisy images and notice the differences.

4. Template matching

OBJECTIVES:

1. Learn about cosine distance
2. Learn about correlation
3. Use correlation and template matching for finding patterns in a image.

NORMALIZED CROSS CORRELATION:

Finds the similarity between two collection of numbers/ vectors/ images. It helps to find a template within a given image by running a sliding window same as the template and calculating the normalized cross correlation. The normalized cross correlation coefficient between two image windows u, v is defined as:

$$\gamma(u, v) = \frac{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}][t(x-u, y-v) - \bar{t}]}{\sqrt{\sum_{x,y} [f(x, y) - \bar{f}_{u,v}]^2 \sum_{x,y} [t(x-u, y-v) - \bar{t}]^2}}$$

Task 9. Write a function in Matlab which will implement normalized cross correlation between two arrays.

Task 10. Use the idea of normalized cross-correlation two find similar pair of images from a collection of images

5. Convolution

OBJECTIVES:

- Learn about convolution operation
- Use convolution for filtering and edge detection operations

INTUITION:

Convolution denotes the sliding of a function over another function and computing their product sum over the whole interval. In terms of image, it usually involves running a filter/kernel for smoothing/noise reducing/ edge detection/ template matching- as all of them are done using some kernel who slide over the images.

Task 11. Learn the use of Matlab's `im2conv()` function

Task 12. Write a code from scratch which will take an image and a kernel and run convolution

Task 13. Try to perform template matching using your convolution code.

Session 4

Image Histogram

OBJECTIVES:

The objective of this lab is to understand & implement

1. Histogram
2. Histogram Equalization
3. Histogram Matching

HISTOGRAM EQUALIZATION:

Histogram of a digital image with gray levels in range $[0, L - 1]$ is a discrete function $h(k) = n_k$ where $k = k^{th}$ gray level and $n_k =$ no. of pixels of an image having gray level r_k . In histogram there are 3 possibilities as follows,

1. For a dark image the components of histogram on the low (dark) side.
2. For a bright image the component are on high (bright) side &
3. For an image with low contrast they are in the middle of gray side.

Histogram equalization is done to spread their component uniformly over the gray scale as far as possible. This is obtained by function

$$S_k = \frac{\sum_{i=0}^k h_i}{n}; k = 0, 1, 2, \dots, i - 1$$

Thus processed image is obtained by mapping each pixel with level r_k into a corresponding pixel with level s_k in output image. This transformation is called Histogram equalization.

ALGORITHM:

1. Read the input image & its size.
2. Obtain the gray level values of each pixel & divide them by total number of gray level values.
3. Implement the function S_k
4. Plot the equalized histogram and original histogram.
5. Display the original and the new image

CONCLUSION:

Digital histogram enhances image but it does not generate a flat histogram.

Question 1. What information one can get by observing histogram?

HISTOGRAM SPECIFICATION:

Histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram. But it is useful sometimes to be able 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 specification.

$$S_k = T(r_k) = \sum Pr(r_j); k = 0, 1, 2, 3, \dots, L - 1$$
$$V_k = G(z_k) = \sum Pz(z_j); k = 0, 1, 2, 3, \dots, L - 1$$
$$Z_k = G^{-1}(T(r_k)); k = 0, 1, 2, 3, \dots, L - 1$$

Map each pixel with level r_k into a corresponding pixel with level s_k . Obtain the transformation function G from a given histogram $P_z(z)$. For any Z_q this transformation function yields a corresponding value V_q . We would find the corresponding value Z_q from G^{-1} .

ALGORITHM:

1. Obtain the histogram of the given image.
2. Map each level r_k to s_k
3. Obtain the transformation function G from the given $P_z(z)$
4. Calculate z_k for each value of s_k
5. For each pixel in the original image, if the value of that pixel is r_k , map this value to its corresponding level s_k , then map level s_k into the final value z_k
6. Display the modified image and its histogram.

Task 1. Write a program to equalize the histogram.

Task 2. Write a program to implement Histogram Matching Algorithm in Matlab.

Session 5

Morphological Image Processing

OBJECTIVE:

The objective of this lab is to understand:

1. Dilation
2. Erosion
3. Opening
4. Closing

CONCEPTS TO LEARN:

1. Structuring element
2. hit
3. fit
4. miss
5. Set definitions regarding image

NOTE:

Morphological Operations are always done on binary images.

CODE OF MORPHOLOGICAL OPERATIONS:

Program-code 5.1 shows a sample code which takes in input image and shows the binary, dilation, erosion, opening and closing images as in Figure 5.1. The implementation of functions `hitfitmiss()`, `myerosion()`, `mydilation()`, `myopening()`, `myclosing()` are left as exercises.

```
1 R = imread('fprint.jpg');
2 %figure;
3 se = [0 1 0;
4       1 1 1;
5       0 1 0];
6
7
8 subplot(2,3,1);
9 imshow(R);
10 title('input')
11
12 I =im2bw(R);
13 subplot(2,3,2);
14 imshow(I);
15 title('binary')
```

```

16
17 S=myerosion (R, se );
18 subplot(2,3,3);
19 imshow(S)
20 title('erosion')
21
22 S=mydilation(R, se );
23 subplot(2,3,4);
24 imshow(S)
25 title('dilation')
26
27 S=myopening(R, se );
28 subplot(2,3,5);
29 imshow(S)
30 title('opening')
31
32 S=myclosing(R, se );
33 subplot(2,3,6);
34 imshow(S)
35 title('closing')

```

Program 5.1: Morphological Operations

SAMPLE INPUT/OUTPUT:

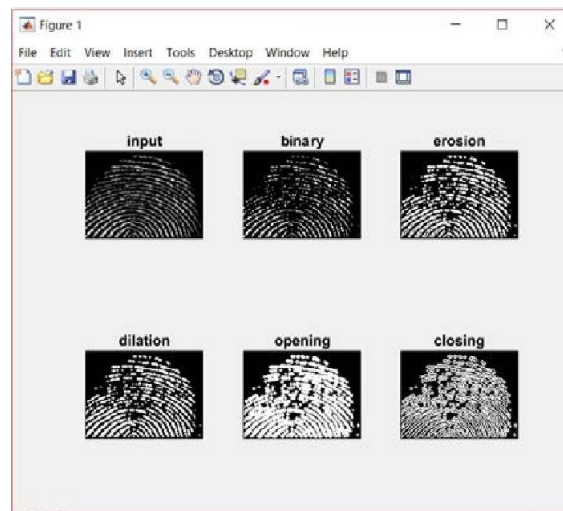


Figure 5.1: Original images and morphologically operated images.

Task 1. Write a program to implement dilation and note the effect on binary images

Task 2. Write a program to implement erosion and note the effect on binary images

Task 3. Write a program to implement opening and note the effect on binary images

$$A \circ B = (A \ominus B) \oplus B$$

Task 4. Write a program to implement closing and note the effect on binary images

$$A \bullet B = (A \oplus B) \ominus B$$

Session 6

Image Derivatives and Edge Detection

OBJECTIVE:

The objective of this lab is to understand & implement

1. To learn about different edge detection operators
2. Use of second derivative for image enhancement: the Laplacian
3. Use of first derivative for image enhancement: the Gradient
4. To implement Image segmentation using edge detection technique.

IMAGE SEGMENTATION:

Image segmentation can be achieved in two ways.

1. Segmentation based on discontinuity of intensity
2. segmentation based on similarities based on intensity edge detection from an important part

An edge can be defined as a set of disconnected pixels that form a boundary between two disjoint regions

EDGE DETECTION MASK:

Edge detection is performed using various masks/operators. An edge detection mask is usually a $n \times n$ matrix which is used as a sliding window on an image and the dot product of the mask and image window is placed on the edge-derived image's corresponding window's center pixel. In Table 6.1, we have shown a 3×3 image pixel window. Different edge detection operators can be applied here.

Z_1	Z_2	Z_3
Z_4	Z_5	Z_6
Z_7	Z_8	Z_9

Table 6.1: A 3×3 image used for mask illustration

ALGORITHM:

1. Construct an empty image with same dimensions as input image
2. Choose an edge detection mask and use it as sliding window over the image
3. With each input image pixel as center, calculate the dot product of the mask and the corresponding image window
4. Place the result in the corresponding image pixel (center) of the empty image
5. At the end, return the new image

ROBERT'S MASK:

Equation: $\nabla f \approx |z_5 - z_9| + |z_6 - z_8|$

Masks along x-axis, y-axis and the sum of masks are:

" # " # " #

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$
 respectively.

PREWITT OPERATOR:

Equation: $\nabla f \approx |z_1 - z_7 + z_2 - z_8 + z_3 - z_9| + |z_3 - z_1 + z_6 - z_4 + z_9 - z_7|$

Masks along vertical axis and horizontal axis are:

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$
 respectively.

SOBEL OPERATOR:

Equation: $\nabla f \approx |z_1 - z_7 + 2z_2 - 2z_8 + z_3 - z_9| + |z_3 - z_1 + 2z_6 - 2z_4 + z_9 - z_7|$

Masks along vertical axis and horizontal axis are:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
 respectively.

LAPLACIAN OPERATOR:

Equation:

$\nabla f \approx |z_9 - z_5| + |z_8 - z_6|$

$\nabla^2 f = [f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1)] - 4f(x, y)$

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & \text{If the center coefficient of the Laplacian Mask is negative} \\ f(x, y) + \nabla^2 f(x, y) & \text{If the center coefficient of the Laplacian Mask is positive} \end{cases}$$

Laplacian Mask Operator:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

CODE OF PREWITT EDGE DETECTION:

Program-code 6.1 shows a sample code of prewitt edge detection and Figure 6.1 shows the output.

```

1 function prewittEdge ()
2
3 S = im2double (rgb2gray ( imread ( 'coins.jpg' ) ));
4
5 %% defining filters
6 hx = [-1 0 1;
7       -1 0 1;
8       -1 0 1];
9

```

```

10 hy = [ 1  1  1;
11       0  0  0;
12      -1 -1 -1];
13
14 %% applying filters
15
16 Rx = imfilter(S, hx);
17 Ry = imfilter(S, hy);
18
19
20 figure;
21 subplot(1,3,1);
22 imshow(S);
23 title('input');
24
25 subplot(1,3,2);
26 imshow(Rx);
27 title('edges along x');
28
29 subplot(1,3,3);
30 imshow(Ry);
31 title('edges along y');
32
33 end

```

Program 6.1: Prewitt Edge Detection

SAMPLE INPUT/OUTPUT:

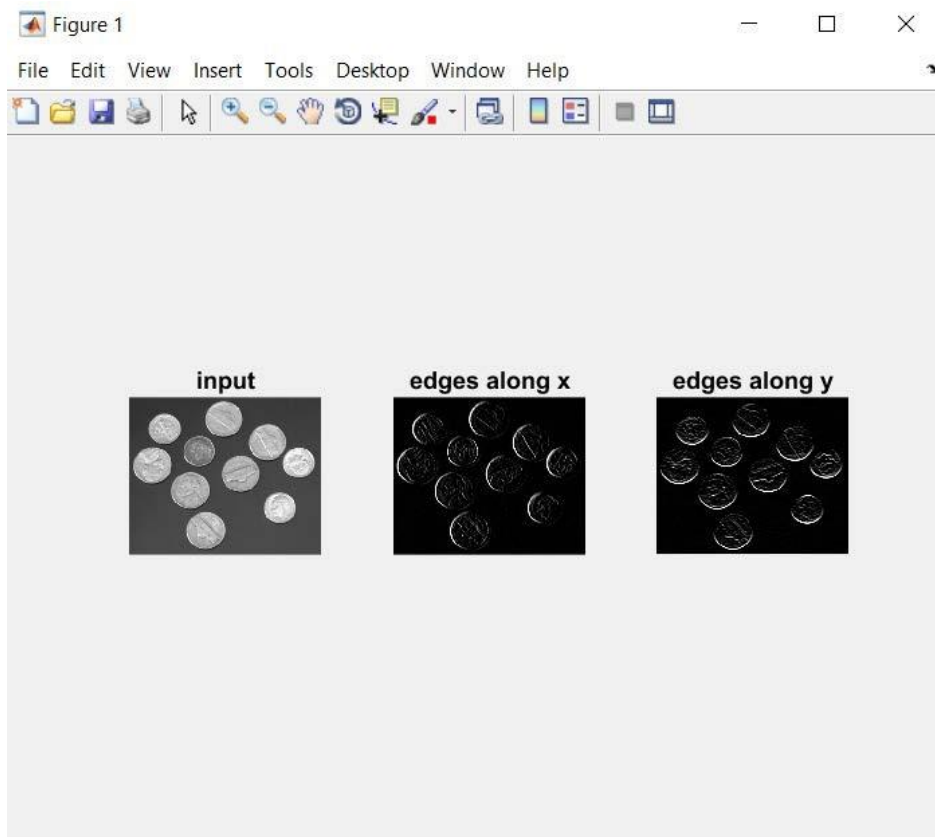


Figure 6.1: Original image and Edge Detected Image using Prewitt Operators in two directions.

CONCLUSION:

Prewitt is simpler to implement but sobel gives the better result. Laplacian is more sensitive to noise.

Question 1. Give the difference between first order derivative filter and second order derivative filter.

Question 2. What is compass gradient mask?

Task 1. Write a program to implement “Robert Cross Gradient Operator” and observe the changes on image.

Task 2. Write a program to implement “Sobel Gradient Operator” and observe the changes on image.

Task 3. Write a program to implement “Prewitt Operator” and observe the changes on image.

Task 4. Write a program to implement “Laplacian Operator” and observe the changes on image.

Task 5. Write a program to implement which will take an input image and an edge detection mask. It should return the edge-detected image.

Task 6. Write a program which will take the input image and return the output image as shown in Figure 6.2.

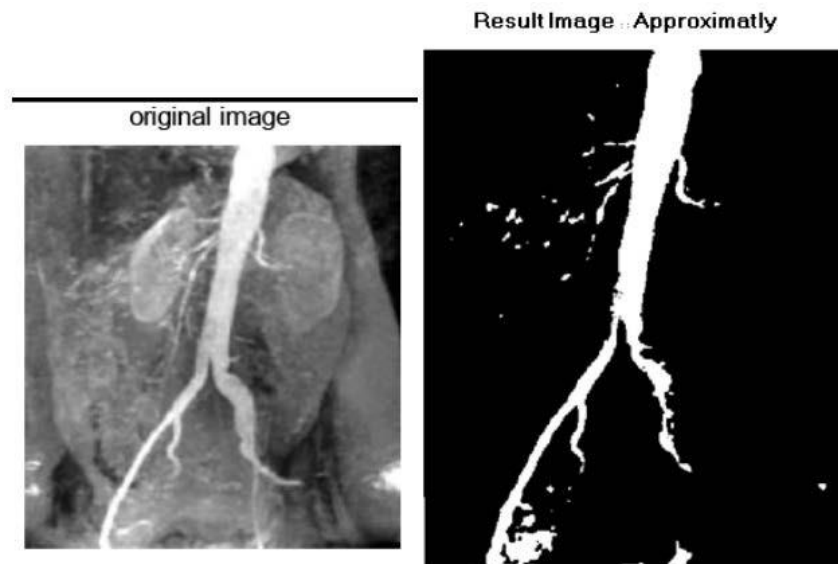


Figure 6.2: Original image and approximated binary edge directed image

MID TERM EXAMINATION

There will be a 40-minutes written mid-term examination. Different types of questions will be included such as MCQ, mathematics, writing code fragments etc.

FINAL TERM EXAMINATION

There will be a one-hour written examination. Different types of questions will be included such as MCQ, mathematics, write a program etc.

Ahsanullah University of Science and Technology
Department of Computer Science and Engineering

Course No: CSE 4228

Course Title: Digital Image Processing Lab

LAB MANUAL – 1

Objective:

The objective of this lab session is to getting familiar with MATLAB. This will cover the following topics –

- 1) MATLAB workspace, Command Window, Variable panel
- 2) Working with variables, vectors and function
- 3) Working with library functions.
- 4) Getting started with MATLAB plotting tools.

Software: MATLAB (any version higher than 2009a)

Pre-requisite: Vector, Matrix, Linear Algebra, Geometry, C, C++.

References:

- [1] https://www.mathworks.com/help/matlab/learn_matlab/desktop.html
 - [2] <https://www.mathworks.com/help/matlab/elementary-math.html>
 - [3] https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html
 - [4] https://www.mathworks.com/help/matlab/functionlist.html?s_cid=doc_ftr
 - [5] <https://www.mathworks.com/help/matlab/2-and-3d-plots.html>
-

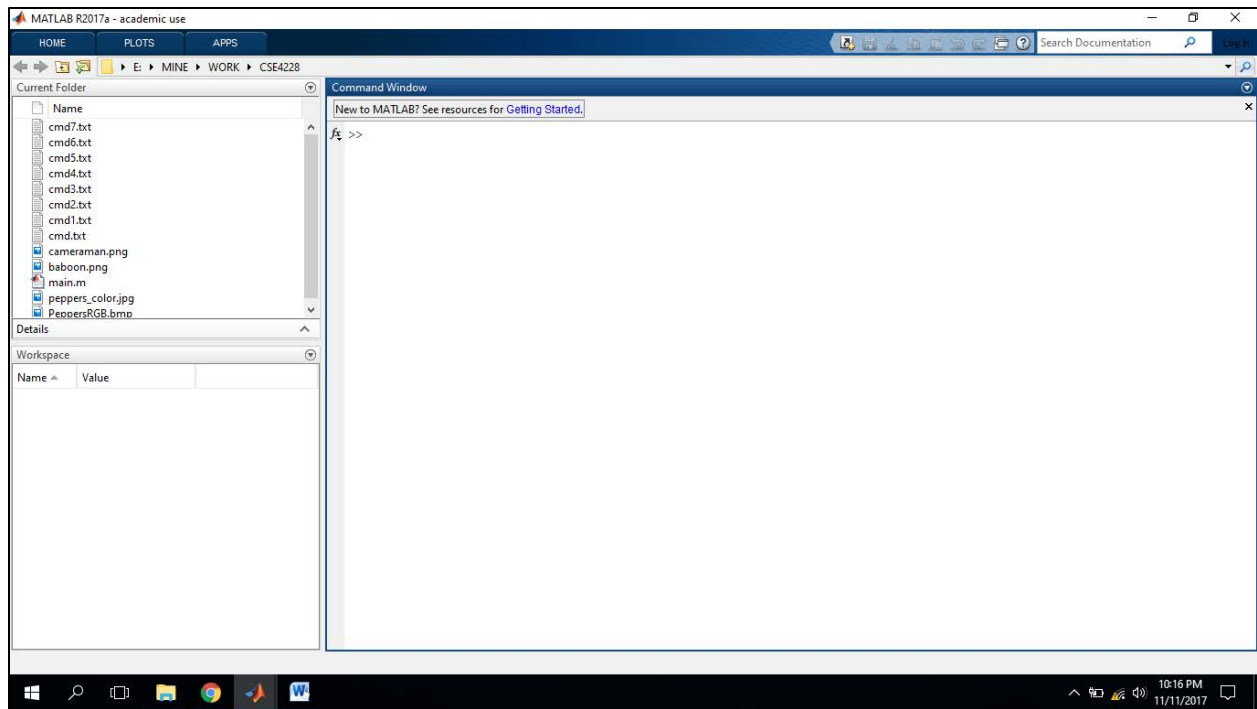


Figure 1

The desktop includes these panels (Figure 1): [1]

- Current Folder — Access your files.
- Command Window — Enter commands at the command line, indicated by the prompt (>>).
- Workspace — Explore data that you create or import from files.

Variables and the workspace:

Let's start with variable.

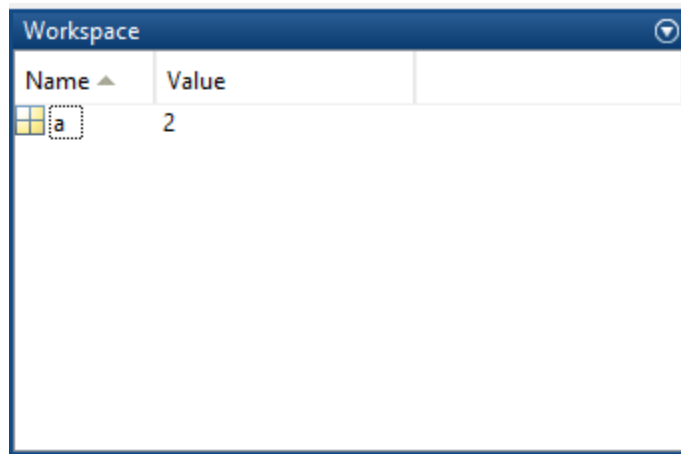
```
a = 2
```

After entering, the immediate next line will display the variable with the assigned value.

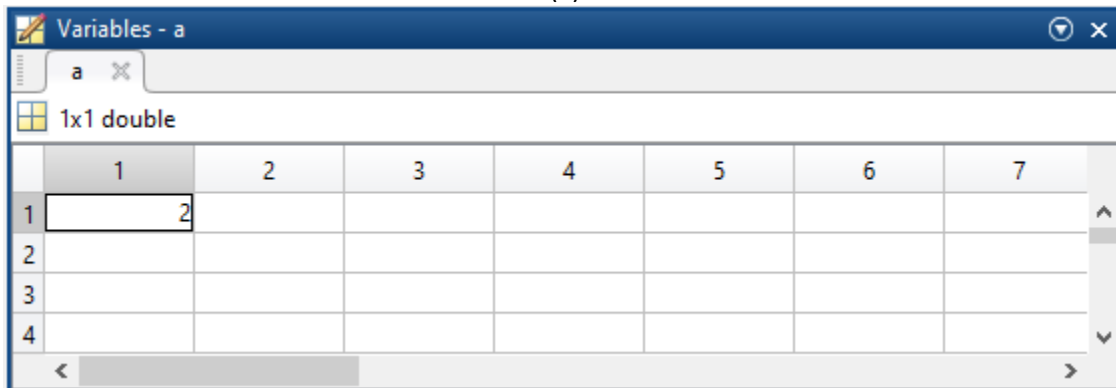
```
a =  
2
```

And you can see that, the variable is appeared in the workplace (Figure 2 a). If you double click on that variable icon, a panel named 'Variables' will popup (Figure 2 b).

MATLAB stores everything as matrix. You can note the indication of 1×1 double in the variable panel. It mean a is a 1 by 1 matrix (eventually a matrix with only one element), which is double type. By default any numerical data is double type. There are other data types as well such as uint8 (unsigned integer 8 bit), char (character), logical (boolean).



(a)



(b)

Figure 2

Note that, throughout this manual, to make it more readable, we will highlight the commands, and the prompted output will not be highlighted. For example, the following is a command.

```
b = 3
```

And the following is the immediate prompt.

```
b =
```

```
3
```

Let's start with type casting.

```
a = 1
```

```
a =
```

```
1
```

```
d = logical(a)
```

```
d =
```

```
1
```

You can see that, `d` will be stored as `1 x 1 logical` variable in the workspace.

Now, Let's work more with variables and arithmetic operations. Let's play with some simple arithmetic operations. Assume that we have already have `a` and `b` in our workspace.

```
a + b
```

```
ans =
```

```
5
```

Observe that, `ans` is a default variable that stores the result of `a + b`, and it is stored in workspace. If you simply use `ans` as a command, the value inside `ans` will be prompted.

```
ans
```

```
ans =
```

```
5
```

But, we can store the result in a different variable as well.

```
c = a + b
```

```
c =
```

```
5
```

```
c = a * b
```

```
c =
```

```
6
```

```
c = b^3 % power of 3
```

```
c =  
27
```

You can observe that, % is used for commenting in MATLAB.
There are some special variables.

```
pi  
ans =  
3.1416
```

```
c = a*pi  
c =  
6.2832
```

Some library functions:

Some useful library functions for trigonometric operations are shown below.

```
c = cos(30) % take radian as input  
c =  
0.1543
```

```
c = cos(2*pi)  
c =  
1
```

```
c = cosd(30) % take degree as input  
c =  
0.8660
```

```
acosd(c) % inverse cosine (in degree)  
ans =  
30.0000
```

You can also try `sin()`, `tan()` etc.

Some other important functions are provided below.

```
c = mod(10,3) % remainder
```

```
c =
```

```
1
```

```
c = exp(2) % exponential
```

```
c =
```

```
7.3891
```

```
d = ceil(c) % ceiling (there is floor() and round() as well)
```

```
d =
```

```
8
```

```
c = log(10) %natural logarithm
```

```
c =
```

```
2.3026
```

```
c = log10(10) % 10 base logarithm
```

```
c =
```

```
1
```

There is a huge collection of functions dedicated for elementary math. You can find them here [1].

Let's see some other useful command.

- `pwd` : to print the current directory
- `clc` : to clean command prompt history
- `clear` : to delete all the variables from the workspace. You can delete a particular variable from the workspace by the clear command followed by the variable name. For example,

```
clear c
```

Using / not using semicolon:

Semicolons (;) can be used after each command. If used, the output prompt will be suppressed. For example, if you use the following command without a semicolon at the end, the output will be prompted.

```
d = cos(pi)
```

```
d =
```

```
-1
```

But, if you use the following command with a semicolon at the end, the output will not be prompted.

```
d = cos(pi);
```

Working with vector and matrix:

Defining a row vector –

```
v = [1, 2, 3, 4]
```

```
v =
```

```
1      2      3      4
```

You can also omit the commas (,) in between the elements. This will give you the same vector.

```
v = [1 2 3 4]
```

```
v =
```

```
1      2      3      4
```

We are going to follow this approach throughout this manual.

Now, Defining a column vector –

```
v = [1; 2; 3; 4]
```

```
v =
```

```
1  
2  
3  
4
```

Having the idea of row and column vector, now we can easily define a matrix –

```
M = [1 2 3 4; 5 6 7 8; 9 1 2 3]
```

M =

1	2	3	4
5	6	7	8
9	1	2	3

In MATLAB, the matrix element indices start from the top left corner. As we traverse right, we go through each column. And as we traverse down, we go through each row.

	1 st column		4 th column	
	v		v	
1 st row ->	1	2	3	4
	5	6	7	8
3 rd row ->	9	1	2	3

Accessing elements of a matrix:

Now let's see how to access elements from a matrix. To access we use the following format of the command: Matrix (which_row, which_column). For example, Let's assume that we have matrix M in our workspace.

```
m = M(1,1) % accessing 1st row, 1st column from M
```

m =

1

```
m = M(2,3) % accessing 2nd row, 3rd column from M
```

m =

7

You can use `end` to access the last element in a row or column.

```
M(1, end) % 1st row, last column
```

ans =

4

```
M(end, end) % last row, last column
```

```
ans =
```

```
3
```

Assigning elements of a matrix:

```
M(2,2) = 99
```

```
M =
```

```
1     2     3     4
5     99    7     8
9     1     2     3
```

Some library functions to generate matrix:

`eye (m)` : Generating an identity matrix, For example, for $m = 5$

```
I = eye(5) % identity matrix of 5 x 5
```

```
I =
```

```
1     0     0     0     0
0     1     0     0     0
0     0     1     0     0
0     0     0     1     0
0     0     0     0     1
```

`ones (m)` : Generating a matrix that contains only 1 (one). For example, $m = 5$

```
I = ones(5)
```

```
I =
```

```
1     1     1     1     1
1     1     1     1     1
1     1     1     1     1
1     1     1     1     1
1     1     1     1     1
```

You can also generate matrix rather than square shape. You can use `ones (m, n)` to define rows and columns. For example,


```
I = ones(3,5) % 3 rows and 4 columns
```

```
I =
```

```
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
```

Similarly,

```
I = zeros(3,5)
```

```
I =
```

```
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
```

```
I = rand(3,5)
```

```
I =
```

```
    0.8147    0.9134    0.2785    0.9649    0.9572
    0.9058    0.6324    0.5469    0.1576    0.4854
    0.1270    0.0975    0.9575    0.9706    0.8003
```

```
I = ceil(rand(3,2))
```

```
I =
```

```
    1    1
    1    1
    1    1
```

Let's apply matrix operations of two matrices A and B.

```
A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
    1    2    3
    4    5    6
    7    8    9
```

```
B = [9 8 7; 6 5 4; 3 2 1]
```

B =

```
9    8    7
6    5    4
3    2    1
```

C = A + B

C =

```
10   10   10
10   10   10
10   10   10
```

C = A - B

C =

```
-8   -6   -4
-2    0    2
4     6    8
```

C = A * B

C =

```
30   24   18
84   69   54
138  114  90
```

C = 2*A

C =

```
2    4    6
8    10   12
14   16   18
```

C = A^2

C =

```
30   36   42
66   81   96
102  126  150
```

Q.1: Now, can you generate a matrix with random numbers greater than or equal to 1 using one single line of command?

We can apply element wise operations. To perform that, we need to put a dot(.) in front of the operator. For example, the following command will do element wise multiplication, that means 1st element of A will be multiplied with the 1st element of B, 2nd of A will be multiplied with 2nd element of B, and so on.

```
C = A .* B
```

```
C =
```

```
     9     16     21
    24     25     24
    21     16      9
```

Q.2: Now, can you square every element of the matrix A using one single line of command?

Q.3: Also, can you square each elements of both A and B, add the squared elements of A with the squared elements B and store them in the matrix C? (Again, using one single line of command)

Using colon operator:

Let's take the matrix A in previous examples.

```
A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
     1     2     3
     4     5     6
     7     8     9
```

Say, we want to access all the columns at 1st row, that is 1, 2 and 3

```
c1 = A(1, :)
```

```
c1 =
```

```
     1     2     3
```

Moreover,

```
c1 = A(2, :) % 2nd row, all columns
```

```
c1 =
```

```
     4     5     6
```

Similarly, to get all rows at 3rd columns -

```
r1 = A(:, 3)
```

```
r1 =
```

```
3  
6  
9
```

We can use colon operator to convert a matrix into a column vector.

```
v = A(:)
```

```
v =
```

```
1  
4  
7  
2  
5  
8  
3  
6  
9
```

Colon operator is very useful to access a range of elements from a matrix. To give an example, let's take a bigger matrix.

```
R = floor(rand(7)*10)
```

```
R =
```

```
1    5    0    4    1    0    0  
2    9    0    0    6    2    4  
6    2    5    3    2    9    1  
4    7    7    1    6    1    9  
3    7    9    7    6    8    0  
8    3    1    3    7    5    7  
5    5    5    5    4    9    8
```

Say, we want to access 2nd row to 5th row of 1st column.

```
s = R(2:5, 1)
```

```
s =
```

```
2  
6  
4  
3
```

Similarly,

```
s = R(1, 2:5) % column 2 to 5 of row 1
```

```
s =
```

```
5    0    4    1
```

We can crop a region using the same concept. For example, row 2 to 5 and column 3 to 6.

```
S = R(2:5, 3:6)
```

```
S =
```

```
0    0    6    2
5    3    2    9
7    1    6    1
9    7    6    8
```

For better understanding, the elements of S accessed from R are gray-shaded below.

```
1    5    0    4    1    0    0
2    9    0    0    6    2    4
6    2    5    3    2    9    1
4    7    7    1    6    1    9
3    7    9    7    6    8    0
8    3    1    3    7    5    7
5    5    5    5    4    9    8
```

We can assign as well.

```
R(2:5, 3:6) = 99
```

```
R =
```

```
1    5    0    4    1    0    0
2    9    99   99   99   99   4
6    2    99   99   99   99   1
4    7    99   99   99   99   9
3    7    99   99   99   99   0
8    3    1    3    7    5    7
5    5    5    5    4    9    8
```

The colon operator is useful to generate a range of values. Say, we want to create a vector named data with values from 1 to 10.

```
data = 1:10
```

```
data =  
      1      2      3      4      5      6      7      8      9     10
```

```
data = 1:2:10 % increment by 2 steps
```

```
data =  
      1      3      5      7      9
```

Q.4: Can you generate a vector containing 10 values starting from pi to 2*pi with equal steps in between?

Some library function for matrix/ vector:

Transpose of Vector:

```
t = transpose(data)
```

```
t =  
      1  
      3  
      5  
      7  
      9
```

Same thing can be done using a single quote (') operator.

```
t = data'
```

```
t =  
      1  
      3  
      5  
      7  
      9
```

Sorting a vector:

```
t = sort(data)
```

```
t =  
    1    3    5    7    9
```

Many MATLAB functions can be overloaded.

```
t = sort(data, 'descend')  
t =  
    9    7    5    3    1
```

Sum of all the elements of a vector:

```
k = sum(data)  
  
k =  
    25
```

Mean, median:

```
k = mean(data)  
  
k =  
    4.1429
```

```
k = median(data)  
  
k =  
    4
```

Transpose of Matrix:

```
R = floor(rand(5)*10)  
  
R =  
    2    2    3    5    7  
    6    7    4    3    0  
    6    2    6    4    6  
    1    8    8    7    4  
    4    8    6    8    4
```

```
r = R'
```

```
r =
     2     6     6     1     4
     2     7     2     8     8
     3     4     6     8     6
     5     3     4     7     8
     7     0     6     4     4
```

Sorting a Matrix:

```
R = floor(rand(5)*10)
```

```
R =
     8     4     1     8     0
     0     9     8     6     2
     3     1     5     3     1
     2     2     5     5     1
     8     1     1     4     2
```

By default, MATLAB sorts row-wise. That means it treats every column as an individual vector and sort them.

```
s = sort(R)
```

```
s =
     0     1     1     3     0
     2     1     1     4     1
     3     2     5     5     1
     8     4     5     6     2
     8     9     8     8     2
```

However, if you want to perform colum-wise, just pass 2 as parameter.

```
s = sort(R,2)
```

```
s =
     0     1     4     8     8
     0     2     6     8     9
     1     1     3     3     5
     1     2     2     5     5
     1     1     2     4     8
```

This concept works for other functions as well.

Sum of Matrix:

```
k = sum(R) % row-wise
```

```
k =
```

```
    21    17    20    26    6
```

```
k = sum(R,2) % column-wise
```

```
k =
```

```
    21
```

```
    25
```

```
    13
```

```
    15
```

```
    16
```

```
k = sum(R(:)) %sum of all the values.
```

```
k =
```

```
    90
```

Mean of Matrix:

```
k = mean(R) % row-wise
```

```
k =
```

```
    4.2000    3.4000    4.0000    5.2000    1.2000
```

```
k = mean(R,2) % column-wise
```

```
k =
```

```
    4.2000
```

```
    5.0000
```

```
    2.6000
```

```
    3.0000
```

```
    3.2000
```

```
k = mean(R(:)) % mean of all
```

```
k =
```

```
    3.6000
```

More functions can be found here [4].

Plotting:

Assume that, we want to plot five 2-D points. The (x, y) coordinates are $(-5, -2)$, $(6, 4)$, $(8, -3)$, $(9, 5)$ and $(-1, 1)$. Now, let's store this coordinate data in two vectors X and Y, where X contains all x-coordinates and Y contains all the y-coordinates.

```
X = [-5 6 8 9 -1];
```

```
Y = [-2 4 -3 5 1];
```

To plot them, we can simply call the `plot()` function.

```
Plot(X,Y,'*')
```

A window will pop up (Figure 3) showing the axis and the plotted points. Points will be plotted with $(*)$ as we pass this as the third parameter. These are called markers. You can use `\.'`, `\o'`, `\o'`, `\X'`, `\x'` etc. as markers.

Note that, MATLAB plotting axis system follows the conventional Cartesian coordinates (Figure 4).

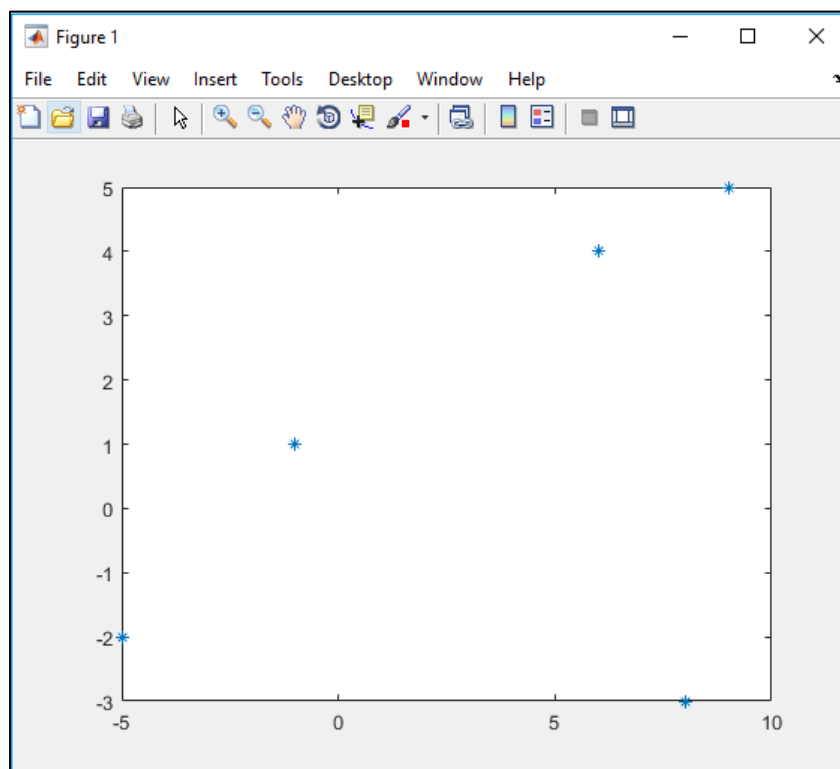


Figure 3

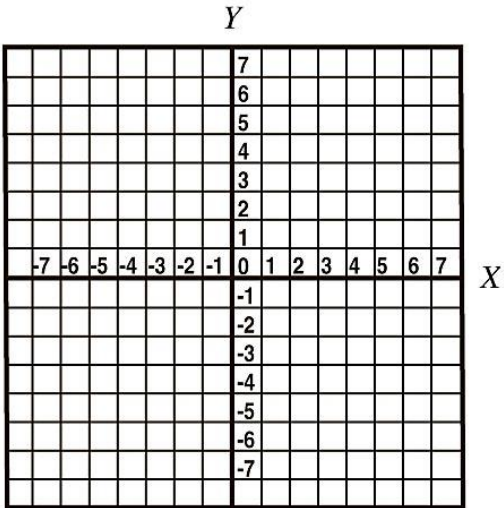


Figure 4

Let's plot $y = x^3$ equation. We define 50 points for x. Then obtain the y from $y = x^3$ equation (Figure 5).

```
x = 1:50;
```

```
y = x.^3;
```

```
plot(x,y, '.')
```

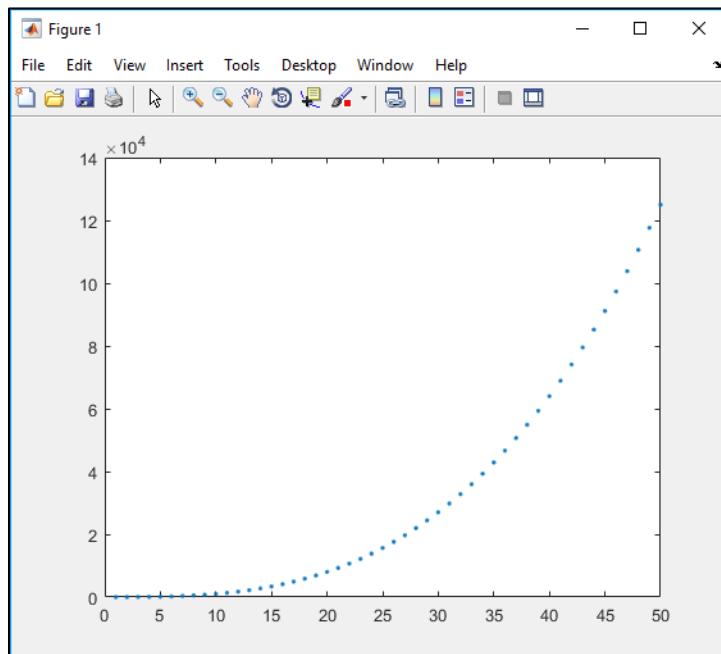


Figure 5

We can connect the points by passing different parameter.

```
plot(x,y,'.-')
```

A hyphen after the dot means to connect the dots with a continuous line (Figure 6).

You can also change the colors of the markers.

```
plot(x,y,'.-r') % r stands for red.
```

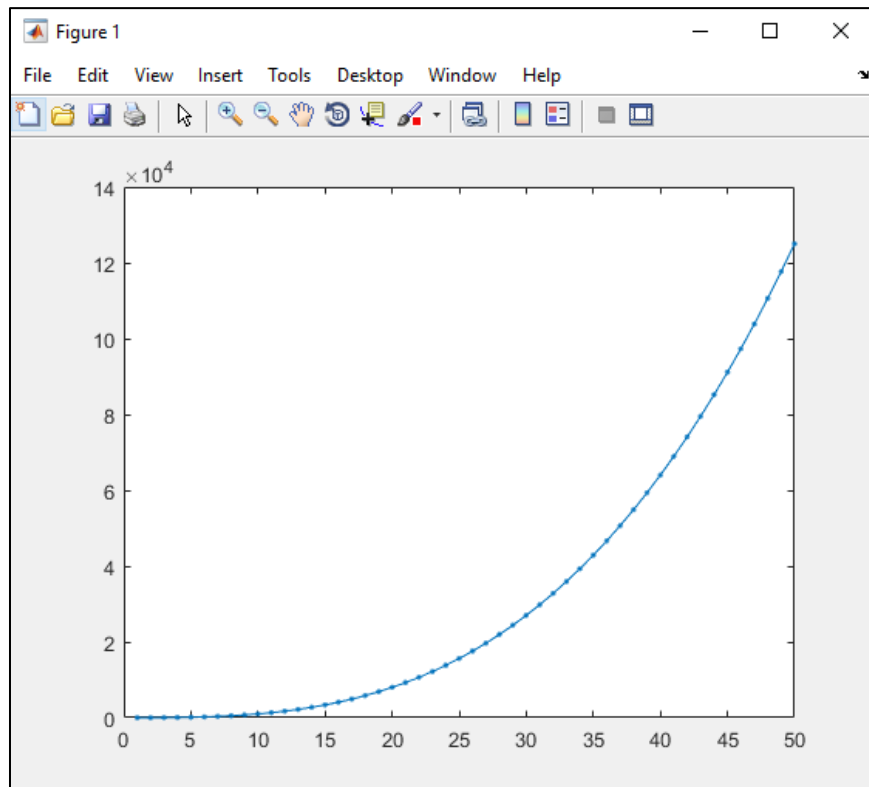


Figure 6

MATLAB allows you to redraw on a current figure. Say, we want to plot $y = (x+10)^3$ along with the $y = x^3$. That means you want to plot on top each other. In that case, we use `hold on` command to let the figure window wait for next plotting.

```
x = 1:50;
```

```
y = x.^3;
```

```
plot(x,y,'.-b'); % plotting the first equation
```

```
hold on; % holding the figure to wait.
```

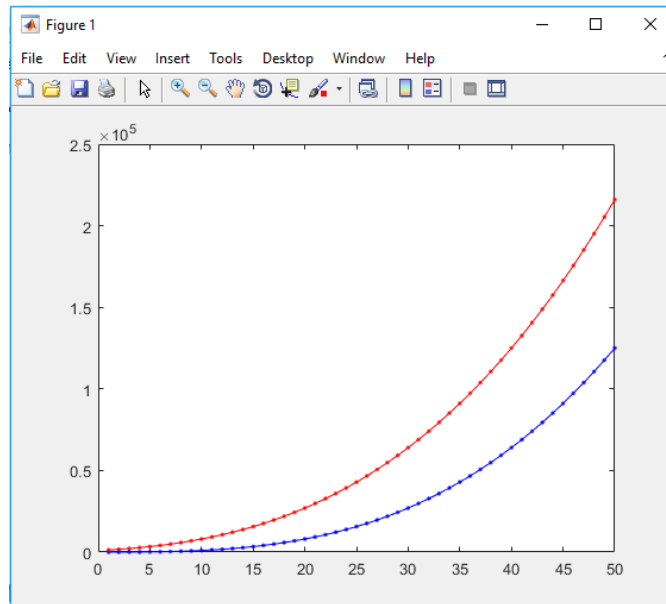
```
y = (x+10).^3; % deriving the y for the second equation
```

```
plot(x,y,'.-r'); % plotting the first equation
```

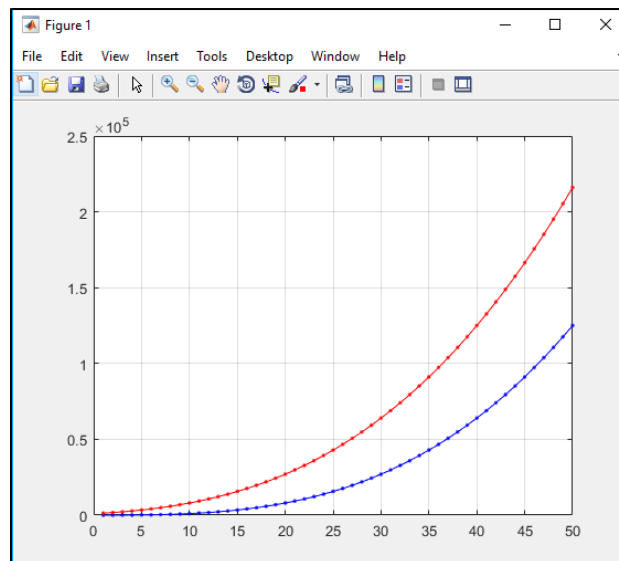
```
hold off; % do not wait any more
```

See output in figure 7a. You can introduce grid lines on the current figure (Figure 7b). To do this, keep the current figure opened and enter the command –

```
grid on;
```



(a)



(b)

Figure 7

Also, you can have multiple figures for multiple plotting. Say, we want the previous two equations to be plotted in two different figure windows. In that case, enter `figure` command to open a new figure (Figure 8).

```
x = 1:50;  
y = x.^3;  
figure; % new window  
plot(x,y,'.-b');  
y = (x+10).^3;  
figure; % new window  
plot(x,y,'.-r');
```

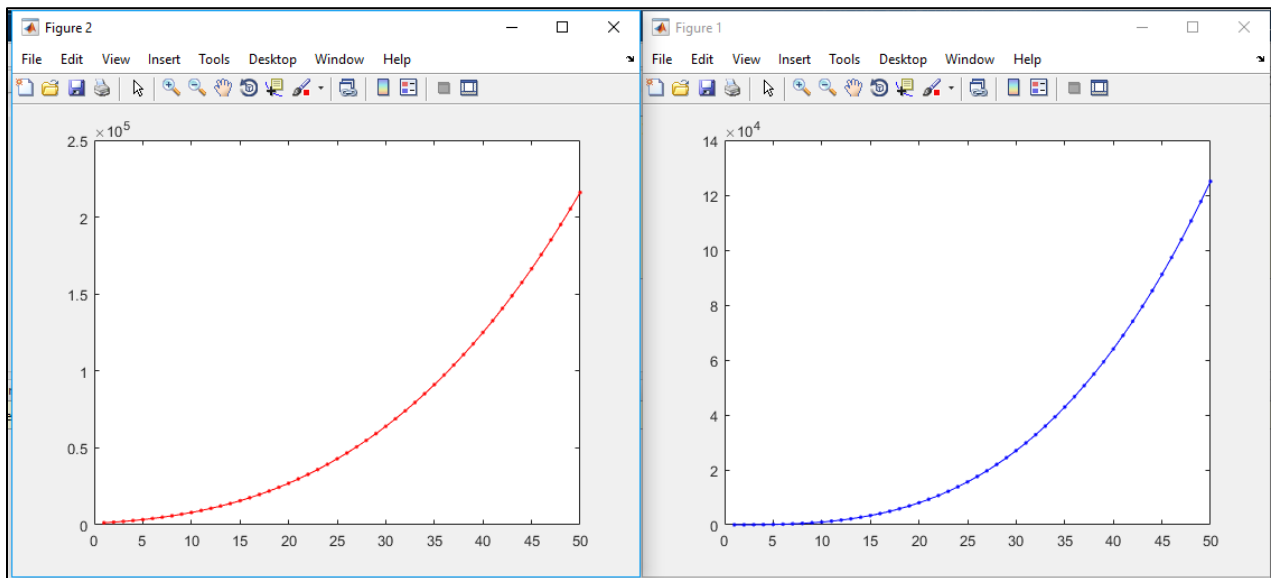


Figure 8

Another useful plotting function is `bar()`. For example,

```
bar(x, y)
```

Figure 9 shows the output.

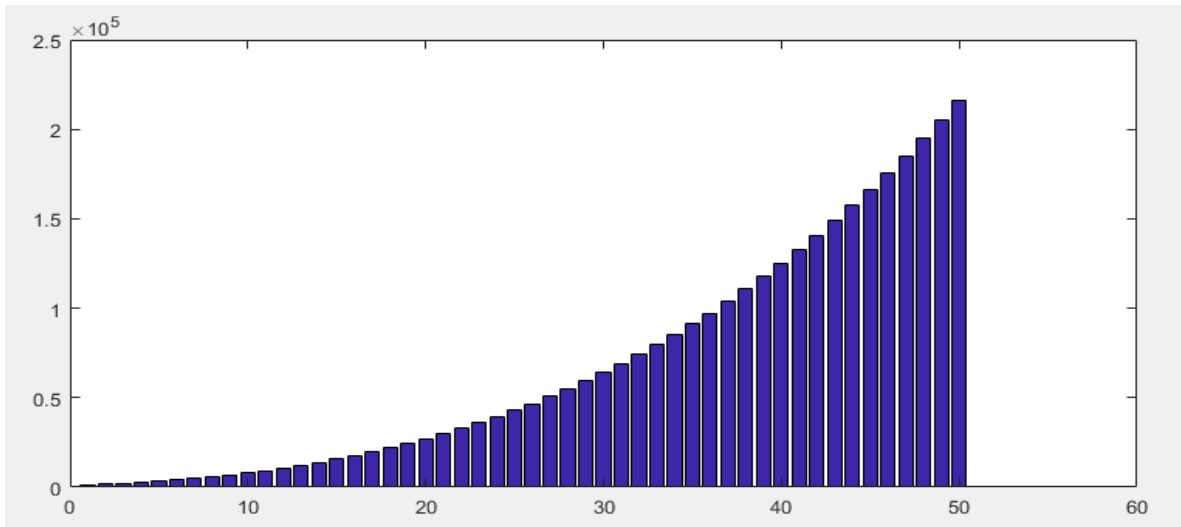


Figure 9

More about MATLAB plotting is available here [5].

[END]

Prepared by-
Mohammad Imrul Jubair

Ahsanullah University of Science and Technology
Department of Computer Science and Engineering

Course No: CSE 4228

Course Title: Digital Image Processing Lab

LAB MANUAL – 2

Objective:

The objective of this lab session is to getting familiar with MATLAB and Image Processing Toolbox. This will cover the following topics –

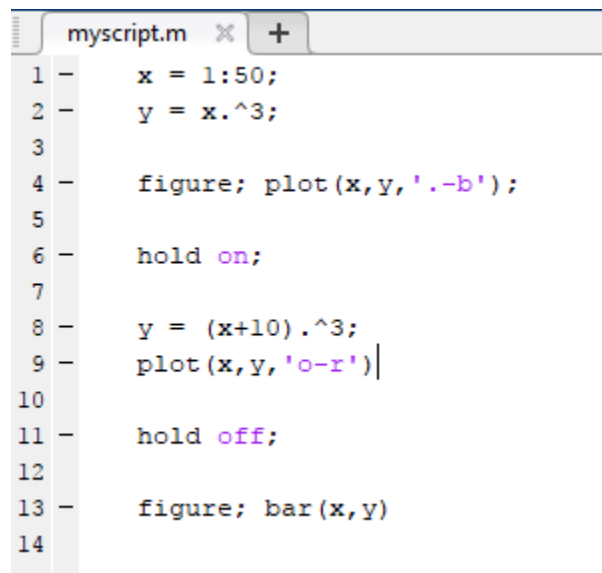
- 1) MATLAB scripts, m-files
- 2) Control statements, functions
- 3) Introduction to Image Processing Toolbox.

Software: MATLAB (any version higher than 2009a)

Pre-requisite: Basic on MATLAB, Fundamental concept of digital images, C/ C++.

Getting started with m-File:

Create a text file with .m extension. Now you can put multiple commands inside of this file and run.



```
myscript.m x +
1 - x = 1:50;
2 - y = x.^3;
3
4 - figure; plot(x,y,'.-b');
5
6 - hold on;
7
8 - y = (x+10).^3;
9 - plot(x,y,'o-r')
10
11 - hold off;
12
13 - figure; bar(x,y)
14
```


Control Statements:

Sample code - 1:

```
A = floor(rand(5,5)*10);  
B = ones(5,5)*9;  
C = A + B;  
[row, col] = size(C); % size() returns the dimension of a matrix  
D = zeros(row, col);  
  
for i = 1:row  
    for j = 1:col  
  
        if i==j  
            D(i,j) = (C(i,j));  
        end  
  
    end  
end  
  
disp(D) %disp() prints a variable in command
```

Sample Code-2:

Sample code -1 can be done easily without control statements.

```
%% Same task is done without loop + if  
A = floor(rand(5,5)*10);  
B = ones(5,5)*9;  
C = A + B;  
[row, col] = size(C);  
I = eye(row, col);  
D = C.*I;  
disp(D)
```

Getting started with image processing toolbox:

It is convenient to keep the m-script and the image you want to work with in the same directory. Let's say we have `cameraman.png` in the same directory.

Now we will go quickly over the basic commands to get started with image processing.

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

`I` will contain the matrix of the image. In workspace, you should see variable `I` with 256 x 256 unit8. That means, the image has 256 rows and 256 columns and every pixel occupies a space of unsigned 8 bit integer. You can double click on the variable and open the Variable panel. Here, every element is a pixel. Observe that every pixel's intensity is in [0 -255] as there are 8 bits assigned.

We can access pixel values just the way we have access matrix elements.

```
pix = I(1,1); % pixel values at (1,1)
```

Now,

```
figure; imshow(I); % it will display the image
```

Now let's dig more. Try to understand the following code.

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

```
figure; imshow(I);
```

```
[row, col] = size(I);
```

```
K = uint8(ones(row, col));
```

```
for i = 1:row
    for j = 1:col
        K(j,i) = I(i,j);
    end
end
```

```
figure; imshow(K);
```

Q.1: By the way, can you do the same without a loop?

Moreover, you can write a matrix as an image formation on to you disk. For example, if we want to save the matrix `K` as an image named `modified.jpg`, we can use the following command.

```
imwrite(K, 'modified.jpg');
```

The image will be written on to the current directory.

There are other commands that can be helpful.

```
imfinfo() : retrieve the image information  
imtool() : open GUI to explore the image
```

R,G,B channels:

Luckily the provide `cameraman.png` is a gray image. But real images are color image. In gray image, every pixel has one value [0 - 255]. In color image, every pixel has 3 values (R, G, B) and every one of these has the range [0 - 255].

Let's work with a color image.

```
I = imread('peppers_color.jpg');
```

Now, in workspace you will see `512 x 512 x 3 uint8`. We know the meaning, only change is – there is a 3rd dimension which stands for R,G,B. In MATLAB, red channel is 1, green channel is 2, and blue channel is 3.

It will be more clear if we access a pixel value. Say, we use the command –

```
r = I(10, 10, 1);  
g = I(10, 10, 2);  
b = I(10, 10, 3);
```

Here, `r` will have the value at the row # 10, column # 10, in color channel # 1 (that means red).

`g` will have the value at the row # 10, column # 10, in color channel # 2 (that means green).

`b` will have the value at the row # 10, column # 10, in color channel # 3 (that means blue).

Q.2: Now, using three lines of code, can you store all the red values, all the green values and all the blue values in three separate matrices say – R, G and B? And plot display them separately?

Hints- the colon operator can be helpful here.

[END]

Prepared by-
Mohammad Imrul Jubair