COMSATS University Islamabad, Lahore Campus
Department of Computer Engineering

CPE415 –Digital Image Processing
Lab Manual for Spring 2026 & Onwards

Lab Resource Person

Engr. Abu Bakar Talha

Theory Resource Person

Dr. Ikram Ullah Khosa

Name: _____ Registration Number :_____

Program: _____Batch: _____

Semester _____

Revision History

Sr.No.	Update	Date	Performed by
1	Lab Manual Preparation	Aug-15	Engr M. Hassan Aslam
2	Lab Manual Review	Aug-15	Dr Ikram Ullah Khosa
3	Layout modifications	Aug-15	Engr M. Hassan Aslam
4	Lab Manual Review	Sep-15	Dr Ikram Ullah Khosa
5	Layout modifications	Sep-15	Engr M. Hassan Aslam
6	Review and Update	Aug-16	Engr Rizwan Asif Engr M. Hassan Aslam
7	Lab Manual Modification according to OBE Requirements.	Feb-18	Engr Assad Ali
8	Lab Manual Review	Feb-18	Dr Ikram Ullah Khosa
9	Lab Manual Modification	Feb-2020	Moazzam Ali Sahi
10	Lab Manual Update	Sep-2021	Moazzam Ali Sahi
11	Lab Manual Update	Feb-2022	Moazzam Ali Sahi and Assad Ali
12	Lab Manual Update	Sep-2023	Dr. M Usman Iqbal
13	Lab Manual Update	Sep-2024	Dr. M Usman Iqbal
14	Lab Manual Modification	Feb-2025	Moazzam Ali Sahi

Preface

First and foremost, I am deeply grateful to ALLAH Almighty, whose blessings and mercy have enabled me to complete this lab manual.

Digital Image Processing (DIP) is a rapidly evolving field with applications spanning diverse domains, including astrophysics, neuroscience, medical imaging, computer vision, and beyond. As the demand for skilled professionals in this area continues to grow, it is essential to equip students with a strong foundation in both the theoretical and practical aspects of image processing.

This lab manual is designed to provide students with hands-on experience and a deeper understanding of the core concepts and techniques in Digital Image Processing. Through a series of carefully crafted experiments and exercises, students will explore various image transformations, enhancements, and analysis methods. By the end of this course, students will gain proficiency in key areas such as image enhancement in spatial and frequency domains, image restoration, color image processing, and image segmentation.

The manual is structured to cover the following topics:

- **Introduction to Image Processing:** Image representation and processing devices.
- **Image Fundamentals:** Visual perception, sampling, and quantization.
- **Image Enhancement:** Histogram modification, smoothing, and sharpening techniques.
- **Image Filtering:** Spatial domain filtering, frequency domain filtering (Fourier transform, homomorphic filtering, etc.).
- **Color Image Processing:** Techniques for processing and analyzing color images.
- **Image Restoration:** Algebraic approaches, inverse filtering, and geometric transformations.
- **Image Segmentation:** Methods for partitioning images into meaningful regions.

I would like to extend my heartfelt gratitude to all those who have supported and guided me throughout the creation of this manual. Your encouragement and insights have been invaluable.

It is my sincere hope that this lab manual will serve as a valuable resource for students, helping them develop the skills and expertise needed to excel in the field of Digital Image Processing. Embrace the spirit of "learning by doing" and enjoy the journey of discovery and innovation!

Moazzam Ali Sahi [February 2025]
Lecturer
Department of Computer Engineering,
COMSATS University Islamabad, Lahore Campus.

Books

Textbooks

Digital Image Processing/3E, R.C. Gonzales, R.E. Woods, Addison-Wesley, 2009.

Reference Books

Digital Image Processing, Kenneth R. Castleman, Prentice Hall, 1996
Digital Image Processing and Applications, I. Pitas, John Wiley, 2000
Digital Image Processing/3E, William K. Pratt, John Wiley, 2001

Reference Books for Lab Manual

Digital Image processing using MATLAB, R.C. Gonzales, R.E. Woods, Addison-Wesley

Learning Outcomes

Theory CLOs

1. Analyze the processes of image enhancement and restoration in spatial and frequency domain such as histogram manipulation, contrast enhancement, smoothing, sharpening, and noise removal with a focus on practical applications in image processing. (C4 – PLO1)
2. Evaluate image segmentation, description and representation techniques including edge detection, morphological processing, and feature extraction with a focus on practical applications in image processing. (C5 – PLO2)

Lab CLOs

1. Manipulate image processing techniques using software tools demonstrating proficiency in tasks such as enhancement, restoration, segmentation and feature extraction. (P5 – PLO5)
2. Collaborate in a team environment to implement advanced image processing algorithms while adhering to project deadlines and communication standards. (A5 – PLO10)

CLOs – PLOs Mapping

CLO \ PLO	PLO1	PLO2	PLO5	PLO10	Cognitive Domain	Affective Domain	Psychomotor Domain
CLO1	x				C4		
CLO2		x			C5		
Lab CLO1			x				P5
Lab CLO2				x		A5	

Lab CLOs – Lab Experiment Mapping

CLO \ Lab	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5	Lab 6	Lab 7	Lab 8	Lab 9	Lab 10	Lab 11	Lab 12
Lab CLO1	P3	P3	P4	P4	P5	P5	P5	P5	P5	P5	P5	P5

Grading Policy

Lab Assignments: i. Lab Assignment 1 Marks (Lab marks from Labs 1-3) ii. Lab Assignment 2 Marks (Lab marks from Labs 4-6) iii. Lab Assignment 3 Marks (Lab marks from Labs 7-9) iv. Lab Assignment 4 Marks (Lab marks from Labs 10-12)		25%
Lab Mid Term = 0.5 × (Lab Mid Term exam) + 0.5× (average of lab evaluation of Lab 1-6)		25%
Lab Terminal		50%
✓ CEA	0.5*(Complex Engineering Problem) +0.375*(average of lab evaluation of Lab 7-12) + 0.125*(average of lab evaluation of Lab 1-6)	
☐ OEL	0.1*(Open Ended Lab) +0.4*(Lab Terminal Exam) +0.375 *(average of lab evaluation of Lab 7-12) + 0.125*(average of lab evaluation of Lab 1-6)	
☐ Project Based	0.2*(Lab Project) +0.3*(Lab Terminal Exam) +0.375*(average of lab evaluation of Lab 7-12) + 0.125*(average of lab evaluation of Lab 1-6)	
☐ Conventional	0.5*(Lab Terminal Exam) +0.375*(average of lab evaluation of Lab 7-12) + 0.125*(average of lab evaluation of Lab 1-6)	
	Total (lab)	**100 %**

Final marks	**Theory marks × 0.75 + Lab marks ×0.25**

Software Recourses

MATLAB

Lab Instructions

- All labs comprise two parts: Pre-Lab and In-Lab Exercises
- The students should complete and demonstrate each lab task separately for stepwise evaluation (please ensure that course instructor/lab engineer has signed each step after ascertaining its functional verification)
- Only those tasks that completed during the allocated lab time will be credited to the students. Students are however encouraged to practice on their own in spare time for enhancing their skills.

Lab Report Instructions

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

- Lab objectives
- MATLAB codes
- Results (graphs/tables) duly commented and discussed.
- Critical analysis/Discussion
- Conclusion

Safety Instructions

All students must read and understand the information in this document with regard to laboratory safety and emergency procedures prior to the first laboratory session. **Your personal laboratory safety depends mostly on YOU**. Effort has been made to address situations that may pose a hazard in the lab but the information and instructions provided cannot be considered all-inclusive.

The danger of injury or death from electrical shock, fire, or explosion is present while conducting experiments in this laboratory. To work safely, it is important that you understand the prudent practices necessary to minimize the risks and what to do if there is an accident.

Avoid contact with conductors in energized electrical circuits. Do not touch someone who is being shocked while still in contact with the electrical conductor or you may also be electrocuted. Instead, press the Emergency Disconnect (red button located near the door to the laboratory). This shuts off all power, except the lights.

Make sure your hands are dry. The resistance of dry, unbroken skin is relatively high and thus reduces the risk of shock. Skin that is broken, wet, or damp with sweat has a low resistance. When working with an energized circuit, work with only your right hand, keeping your left hand away from all conductive material. This reduces the likelihood of an accident that results in current passing through your heart.

Be cautious of rings, watches, and necklaces. Skin beneath a ring or watch is damp, lowering the skin resistance. Shoes covering the feet are much safer than sandals.

If the victim isn't breathing, find someone certified in CPR. Be quick! Some of the staff in the Department Office are certified in CPR. If the victim is unconscious or needs an ambulance, contact the Department Office for help or call 1122. If able, the victim should go to the Student Health Services for examination and treatment.

Transistors and other components can become extremely hot and cause severe burns if touched. If resistors or other components on your proto-board catch fire, turn off the power supply and notify the instructor. If electronic instruments catch fire, press the Emergency Disconnect (red button). These small electrical fires extinguish quickly after the power is shut off. Avoid using fire extinguishers on electronic instruments

Table of Contents

1 Follow instructions to display different image types and reproduce their details using MATLAB

1.1 Objectives

Understand MATLAB user interface and learn basic MATLAB commands for matrices and image operations.

1.2 Pre-Lab

Learning to used MATLAB interface, MATLAB Help, vectors, and matrices in MATLAB.

1.2.1 MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar **mathematical** notation. Typical uses include the following:

- Math and computation
- Algorithm development
- Data acquisition
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building.

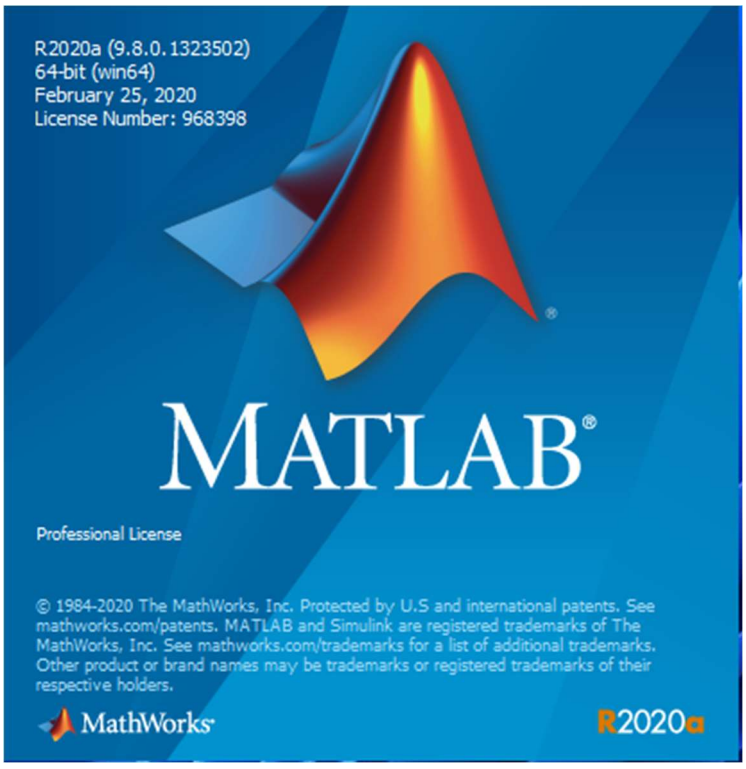
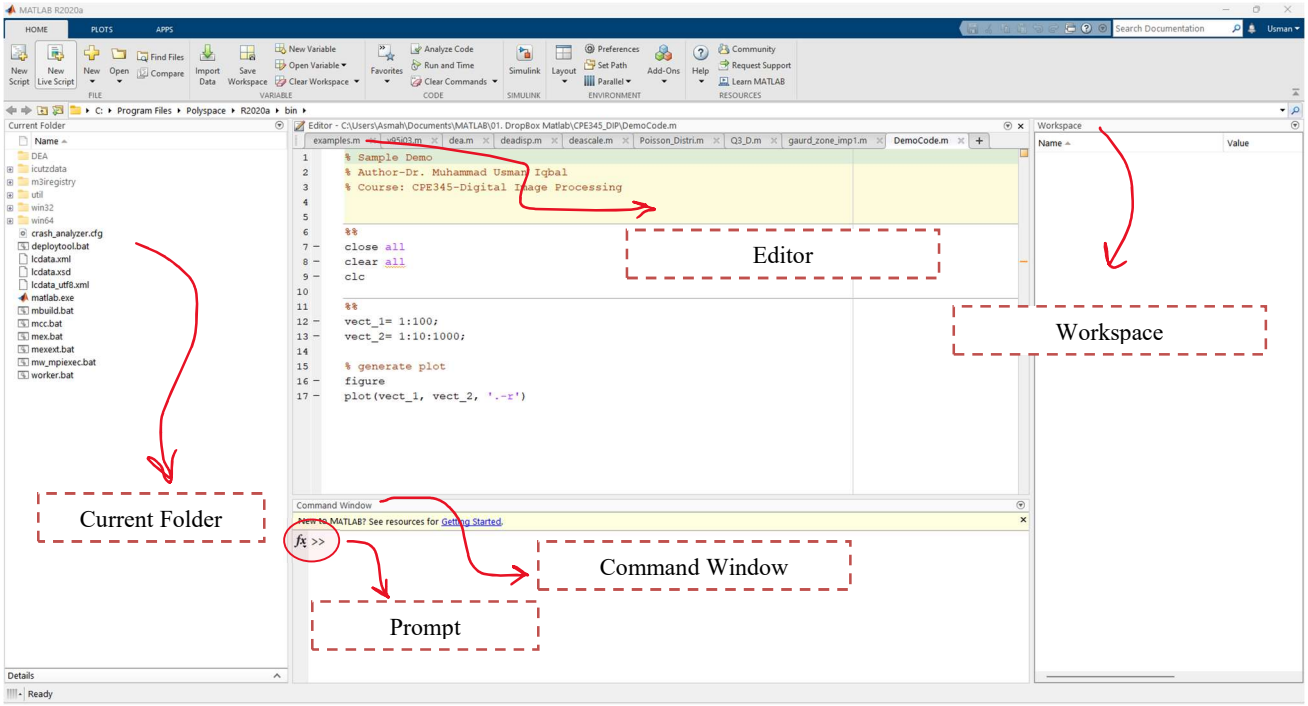


Figure 1.1: Use a version of MATLAB 2018 or above.

1.2.2 MATLAB User Interface



Pre-Lab Task 1

Practice the tools listed in Table 1.

Table 1.1: List of MATLAB Desktop Tools

Tool	Description
Array Editor	View and edit array contents.
Command History Window	View a log of statements entered in the **Command Window**; search for previously executed statements, copy them, and re-execute them.
Command Window	Run MATLAB statements.
Current Folder Browser	View and manipulate files in the current folder.
Current Folder Field	Shows the path leading to the current folder.
Editors	Editor/Debugger and Live Editor (explained in the text).
Figure Windows	Display, modify, annotate, and print MATLAB graphics.
File Comparisons	View detailed differences between two files.
Help Browser	View and search product documentation.
Profiler	Measure execution time of MATLAB functions and lines; count how many times code lines are executed.
Start Button	Run product tools and access product documentation.
Workspace Browser	View and modify contents of the workspace.

1.2.3 How to use MATLAB help

Online help is available from the MATLAB prompt (a double arrow), both generally (listing all available commands). In the text that follows, any line that starts with two greater than signs (>>) is used to denote the MATLAB command line (also known as prompt). This is where you enter your commands.

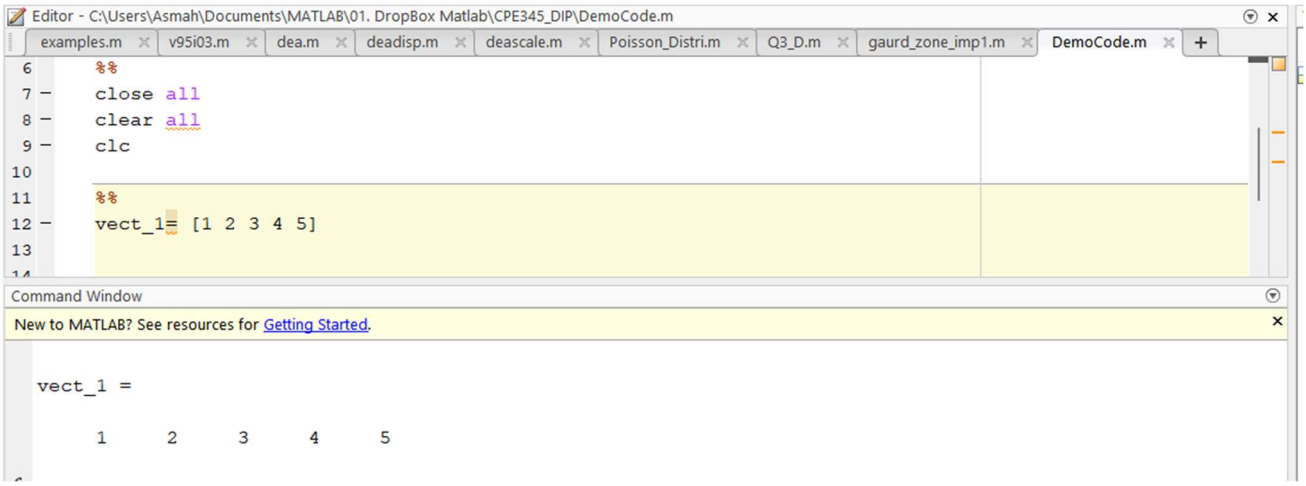
```
>> help
New to MATLAB? See resources for Getting Started.

To view the documentation, open the Help browser.

fx >>
```

1.2.4 Vectors and Matrices in MATLAB

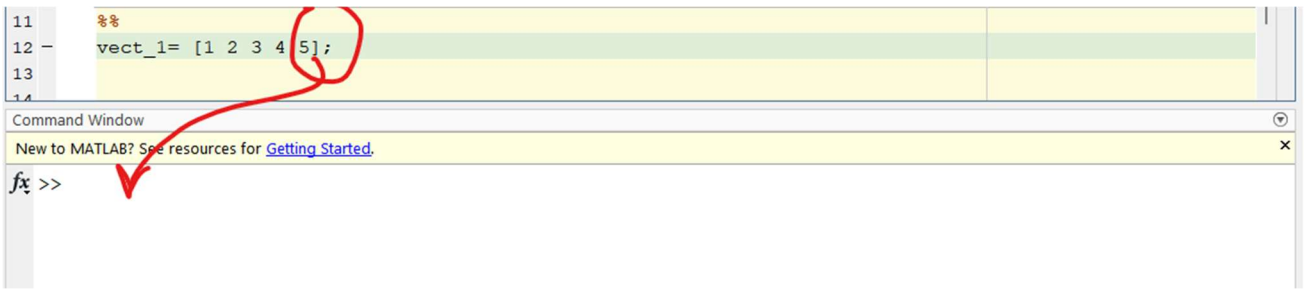
In this section a brief overview of vectors and Matrices generation in MATLAB is described through examples.

1.2.4.1 Defining a Vector

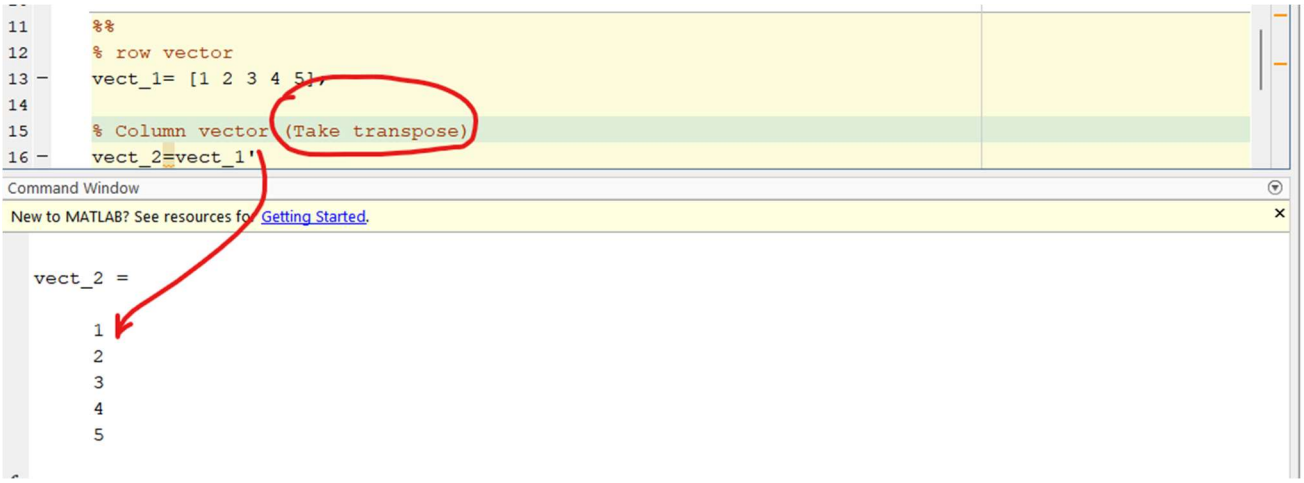
Almost all of MATLAB's basic commands revolve around the use of vectors. A vector is defined by placing a sequence of numbers within square braces:



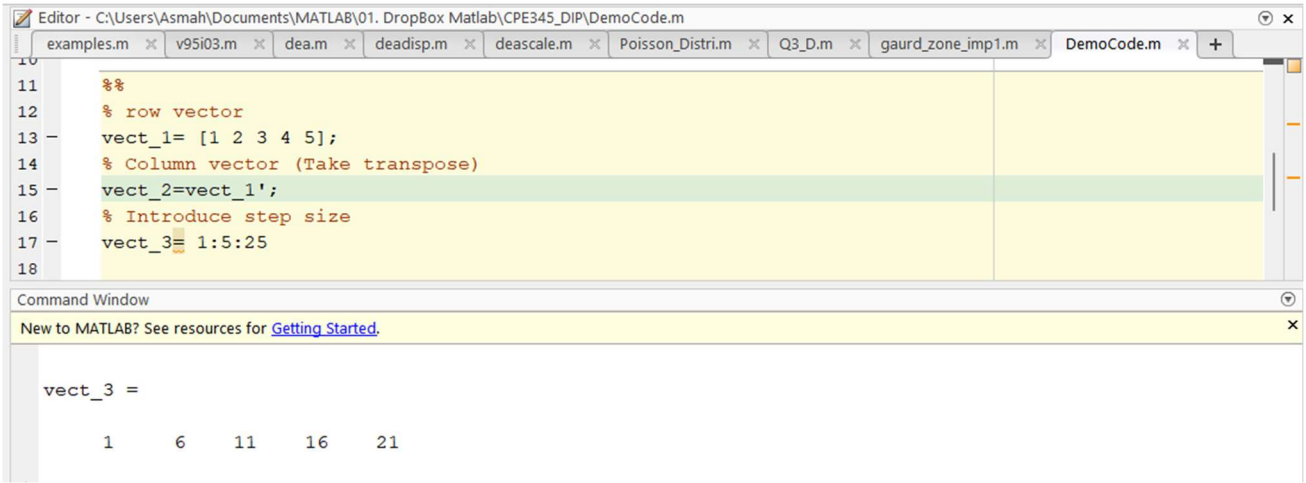
This creates a row vector which has the label "vect_1". Note that MATLAB printed out a copy of the vector after you hit the enter key. If you do not want to print out the result, put a semi-colon at the end of the line:



Notice, though, that this always creates a row vector. If you want to create a column vector you need to take the transpose of a row vector. The transpose is defined using an apostrophe (''' ")



If you wish to use an increment other than one that you have to define the start number, the value of the increment, and the last number. For example, to define a vector that starts with 2 and ends in 4 with steps of .25 you enter the following:



Accessing elements within a vector

You can view individual entries in this vector. For example, to view the first entry just type in the following:

```
>> vect_3(1)

ans =

     1
```

This command prints out entry 1 in the vector. Also notice that a new variable called **ans** has been created. Any time you perform an action that does not include an assignment MATLAB will put the label ans on the result.

1.2.4.2 Defining Matrices

Defining a matrix is similar to defining a **vector**. To define a matrix, you can treat it like a column of row vectors (note that the spaces are required!).

```
20
21 -    mat_1= [1 2 3; 4 5 6; 7 8 9]
22
23
24
```

```
Command Window
New to MATLAB? See resources for Getting Started.

mat_1 =

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

If you have been putting in variables through this and the tutorial on **vectors**, then you probably have a lot of variables defined. If you lose track of what variables you have defined, the `whos` command will let you know all of the variables you have in your work space.

```
>> whos
  Name        Size              Bytes  Class     Attributes

  mat_1       3x3                  72  double
  vect_1      1x5                  40  double
  vect_2      5x1                  40  double
  vect_3      1x5                  40  double

fx >>
```

You can work with different parts of a matrix, just as you can with vectors. Again, you have to be careful to make sure that the operation is legal.

Pre-Lab Task 2

Write and execute .m file with name "Lab_1_PreLab_RegistrationNumber.m" to perform the following tasks:

a. Generate a row vector of the size 1x51.

b. Generate a column vector of the size 1000x1000.

c. Generate two matrices' zeroes and ones of size 1000x1000.

true(M, N)
false(M, N)
magic(M)
rand(M,N)

1.2.4.3 Digital Image Processing Fundamentals

The basic data structure in MATLAB is the array, an ordered set of real or complex elements. This object is naturally suited to the representation of images, real-valued ordered sets of color or intensity data. MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. (Pixel is derived from picture element and usually denotes a single dot on a computer display.)

For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as RGB, require a three-dimensional array, where the first plane in the third dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. This convention makes working with images in MATLAB similar to working with any other type of matrix data and makes the full power of MATLAB available for image processing applications.

1.3 In-Lab Tasks

In-Lab Task 1

Using the MATLAB built-in functions, generate matrices of zeros and ones of the size $M = 100$ *and* $N = 1000$. Use MATLAB function `imshow` to display each image separately in figure window. Your task is to generate an image like this or any other pattern of black and white strips.

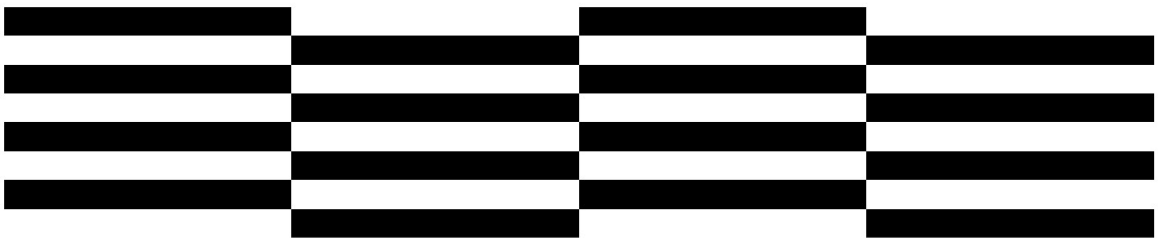


Figure 1.2: Black and White Strips

Use MATLAB Help for the following functions.

zeros
ones
imshow
figure

1.3.1 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.

```
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.

Example:

```
A = imread('chestxray.jpg');
```

This reads the image from the JPEG file "chestxray,jpg" into image array A. Note the use of single quotes (‘ ‘) to delimit the string file name.

Table 1.2: Types of Image Formats supported by MATLAB

Format Name	Description	File Extensions
BMP	Windows Bitmap	.bmp
CUR†	Windows Cursor Resources	.cur
FITS†	Flexible Image Transport System	.fts, .fits
GIF	Graphics Interchange Format	.gif
HDF	Hierarchical Data Format	.hdf
ICO†	Windows Icon Resources	.ico
JPEG	Joint Photographic Experts Group	.jpg, .jpeg
JPEG 2000	Joint Photographic Experts Group	.jp2, .jpf, .jpx, j2c, j2k
PBM	Portable Bitmap	.pbm
PGM	Portable Graymap	.pgm
PNG	Portable Network Graphics	.png
PNM	Portable Any Map	.pnm
RAS	Sun Raster	.ras
TIFF	Tagged Image File Format	.tif, .tiff
XWD	X Window Dump	.xwd

†Supported by `imread`, but not by `imwrite`.

In-Lab Task 2

Using the MATLAB built-in functions, generate matrices of zeros and ones of the size $M = 100\ and\ N = 1000$. Use MATLAB function `imshow` to display each image separately in figure window. Your task is to generate an image like this or any other pattern of black and white strips.

1.3.2 Size of the Image

After reading an image in MATLAB, its size can be determined using the MATLAB function 'size'.

```
% Determine the size of the image
 size(A)

 % assign image size to the variable for future use
 [M N]= size(A)

 % Alternatively, you can use
 s=size(A)
 M=s(1)
 N=s(2)
```

1.3.3 Display an Image

To display images, use the `imshow` function.

```
imshow(X)
```

Example

```
imshow(A);
```

Now, Use the syntax

```
imshow(A,[low high]);
```

It displays as black all values less than or equal to low, and as white all values greater than or equal to high. The values in between are displayed as intermediate intensity values.

Finally, the syntax

imshow(A,[]);

It sets variable low to the minimum value of array A and high to its maximum value. This form of `imshow` is useful for displaying images that have a low dynamic range or that have positive and negative values.

1.3.4 Writing Image Data

To write image to graphics file `imwrite` is used

imwrite(A,filename,fmt)

Example

imwrite(A,'Test.tif');

Function `imwrite` writes the image as a TIFF file because it recognizes the ".tif"extension in the filename.Alternatively, the desired format can be specified explicitly with a third input argument. This syntax is useful when the desired file does not use one oftherecognized file extensions. For example, the following command writes f toa TIFF file called "Test".

imwrite(A,'Test','tif');

How to get no. of rows and columns of image

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

[r,c]=size(A)

r = no of Rows

c = no of columns

Sine and cosine waves plotting

Write a MATLAB code to plot sine and cosine waves with proper titles, labels and annotations, also use different markers and colour for both waves.

Fundamentals of image processing

Write a MATLAB code that perform the following operations.

1. Ask the user to enter the name of the image file.
2. Read the image file.
3. Store the file with a different format and name, the name and format should also be chosen by the user.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

2 Respond to image processing tasks by displaying various image types, tracing image information, and reproducing image and data type conversions using MATLAB

2.1 Objectives

To display various types of images, image information and conversion of images and data types using MATLAB

2.2 In-Lab

Table 2.1Table 2.1: Image Functions for Display Image Information presents the function which can used to explore various type of information from images. Some these functions are also available in the MATLAB image processing APPs.

Table 2.1: Image Functions for Display Image Information

Function	Description
`imtool`	Starts the **Image Viewer app.**
`imageinfo`	**Image Information tool.**
`imcontrast`	**Adjust Contrast tool.**
`imdisplayrange`	**Display Range tool.**
`imdistline`	**Distance tool.**
`impixelinfo`	**Pixel Information tool.**
`impixelinfoval`	**Pixel Information tool** without text label.
`impixelregion`	**Pixel Region tool.**
`immagbox`	**Magnification box** for scroll panel.
`imoverview`	**Overview tool** for image displayed in scroll panel.

2.2.1 Learning `imtool`

```
%% In-Lab
Img= imread('peppers.png');
Img_gray= rgb2gray(Img);
% Practice the following functions
```

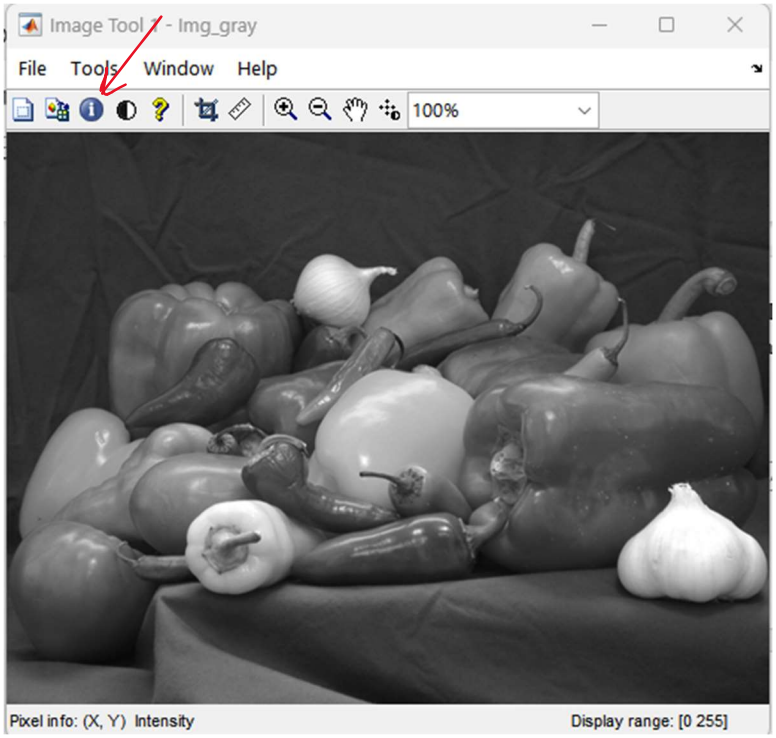
Read two images, one RGB and one Grayscale, in MATLAB which will be utilized through the lab experiment.

According to the above code, RGB image is read into MATLAB using `imread` function and then it is converted into grayscale which will be utilized as the grayscale image.

To avoid plagiarism in your work, you can use any other image and with different names which include your name initial and/or registration numbers.

2.2.1.1 Image information Extraction

```
%-------------------------------------------------------------------------
% 1. imtool
% a) Reading a gray image in imtool
Img_1a= imtool(Img_gray);
% check the image information
```



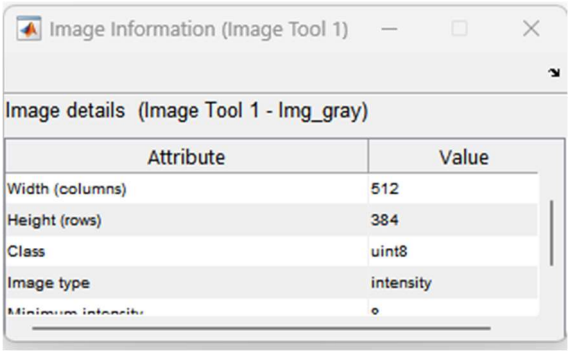


Figure 2.1: Image information is display below the Image in a separate window

Similar information can also be obtained using function `whos` in command window using whos as shown in Figure 2.2:

```
>> whos
  Name            Size                 Bytes  Class                Attributes

  Img           384x512x3             589824  uint8
  Img_1a          1x1                      8  matlab.ui.Figure
  Img_1b          1x1                      0  matlab.ui.Figure
  Img_gray      384x512                196608  uint8
  h               1x1                      0  matlab.ui.Figure
```

Figure 2.2: Image information obtained through 'whos'

2.2.1.2 Image Contrast Adjustment

`Imtool` can be utilized for adjusting the image contrast by sliding the image contrast adjuster over the histogram of the image as shown in the image.

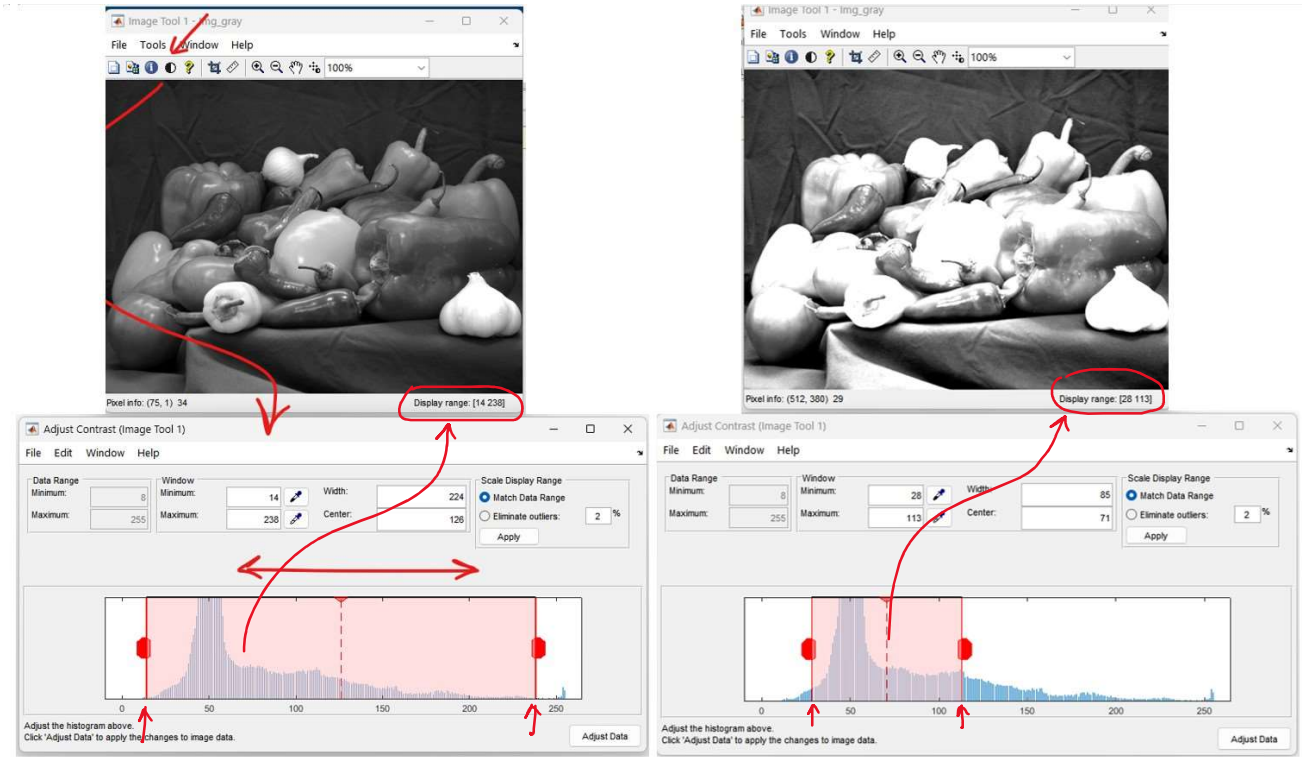


Figure 2.3: Adjusting the contrast of the image using `imtool`

In-Lab Task 1

In addition to performing the above tasks using `imtool`, discuss at least two other applications of `imtool` on your images and display results.

In-Lab Task 2

Display at least two utilizations of `imtool` for RGB image. Briefly explain limitations of `imtool` for RGB images (if any).

2.2.2 Learning imageinfo

This function, `imageinfo`, can be utilized to find detailed information about the image. The information provided by the `imageinfo` is more detailed as compared to `imtool` and `whos`. However, to utilize the `imageinfo`, the handle is to be created first. Follow the following syntax for extracting image information using `imageinfo`.

```
% a) info about gray image
% create a handle
figure
img_info= imshow(Img_gray)% handle is created using name = img_info for
%current image in figure file
info_gray= imageinfo(img_info)
```

```
info_gray =

  Figure (imageinfo) with properties:

      Number: []
        Name: 'Image Info (Figure 1)'
       Color: [0.9400 0.9400 0.9400]
    Position: [617 264 360.7500 157]
       Units: 'pixels'

  Show all properties

                  Alphamap: [1×64 double]
              BeingDeleted: off
                BusyAction: 'queue'
             ButtonDownFcn: ''
                  Children: [3×1 Graphics]
                  Clipping: on
         CloseRequestFcn: 'closereq'
                     Color: [0.9400 0.9400 0.9400]
                  Colormap: [256×3 double]
               ContextMenu: [0×0 GraphicsPlaceholder]
                 CreateFcn: ''
               CurrentAxes: [0×0 GraphicsPlaceholder]
          CurrentCharacter: ''
             CurrentObject: [0×0 GraphicsPlaceholder]
              CurrentPoint: [0 0]
                 DeleteFcn: ''
              DockControls: on
                  FileName: ''
        GraphicsSmoothing: on
         HandleVisibility: 'callback'
            InnerPosition: [617 264 360.7500 157]
            IntegerHandle: off
            Interruptible: on
          InvertHardcopy: on
             KeyPressFcn: ''
            KeyReleaseFcn: ''
                  MenuBar: 'none'
                     Name: 'Image Info (Figure 1)'
                 NextPlot: 'add'
                   Number: []
              NumberTitle: off
            OuterPosition: [609 256 377 223]
```

In-Lab Task 3

Use the handle created for image information extraction for RGB image using `imageinfo` and overwrite at least two properties of image using the object oriented programming. (Like changing color, orientation or turning a property ON/OFF). Display the image information which is overwritten.

In-Lab Task 4

Learn and apply the following functions for gray scale and RGB images without using `imtool`.

imcontrast
imdisplayrange
imdistline
immagbox

The results of In-Lab Task 3 should be like the following image (all above functionalities) without `imtool`.

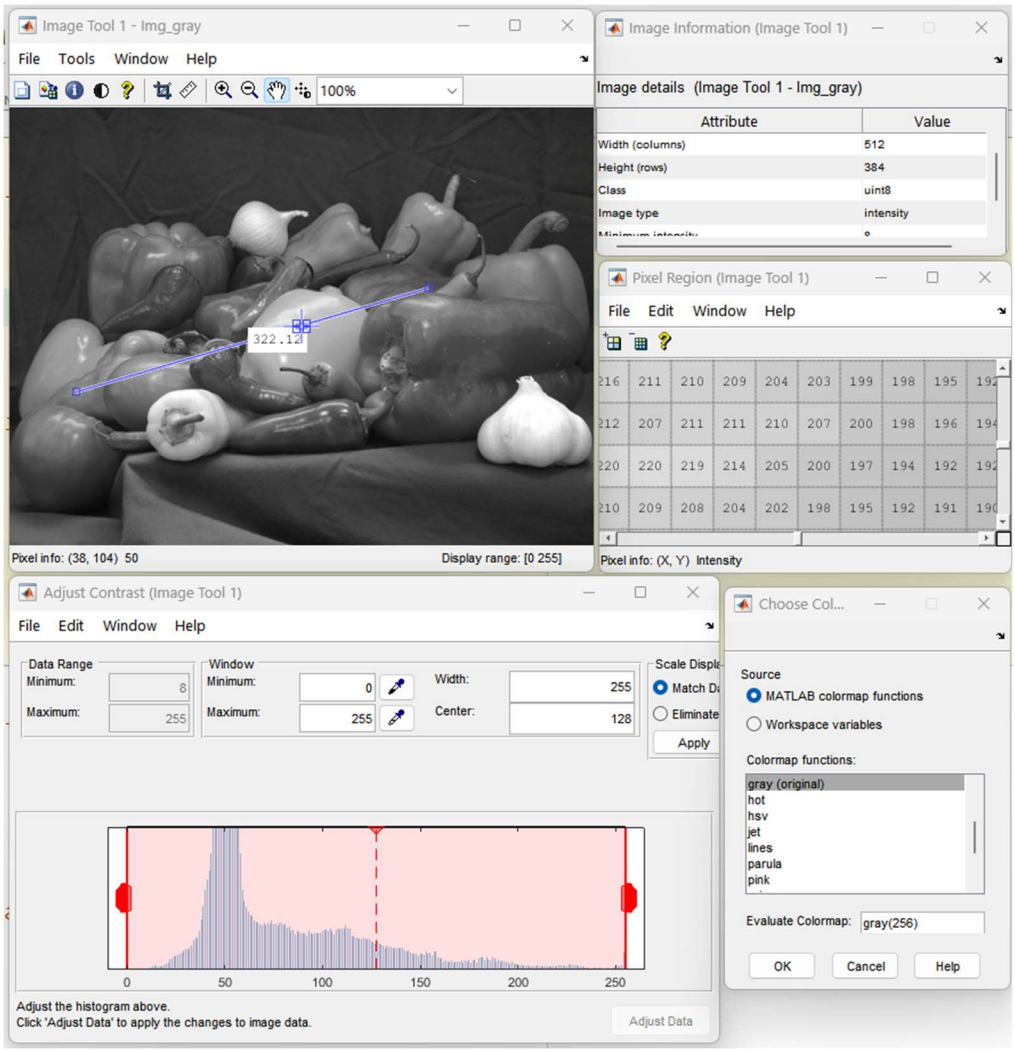


Figure 2.4: Expected In-Lab Task 3 Results

List of Images available in MATLAB

Table 2.2: Image Functions for Display Image Information

```
                autumn.tif                      bag.png
                 blobs.png                    board.tif
            cameraman.tif                     canoe.tif
                 cell.tif                    circbw.tif
              circles.png                   circuit.tif
                coins.png              concordaerial.png
     concordorthophoto.png                    eight.tif
              fabric.png                  football.jpg
              forest.tif                gantrycrane.png
               glass.png                     greens.jpg
             hestain.png                       kids.tif
          liftingbody.png                      logo.tif
                 m83.tif                     mandi.tif
                moon.tif                       mri.tif
            office_1.jpg                    office_2.jpg
            office_3.jpg                    office_4.jpg
            office_5.jpg                    office_6.jpg
               onion.png                     paper1.tif
               pears.png                    peppers.png
             pillsetc.png                       pout.tif
                rice.png                     saturn.png
              shadow.tif                  snowflakes.png
               spine.tif                       tape.png
             testpat1.png                      text.png
                tire.tif                     tissue.png
               trees.tif          westconcordaerial.png
westconcordorthophoto.png
```

2.2.3 Learning `print` function to Save images from MATLAB

Sometimes, it is necessary to save images and plots the way they appear on a MATLAB figure window. The contents of a figure window can be saved in two ways. The first is to use the File pull-down menu in the figure window and then choose "Save As" as shown in the Figure below.

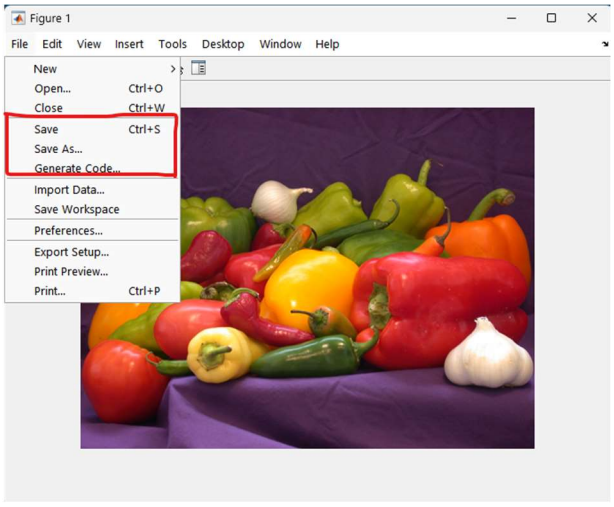


Figure 2.5: Saving Image from the File menu of the Figure.

With this option, the user can select a location, file name, and format. More control over export parameters is obtained by using the print command:

```
%% -print
fig1=figure; % create a figure
imshow(Img)
% print the figure fig1 as the RGB_Image.png
print(fig1,'RGB_Image','-dpng')
% including '-r0', will save the image at the screen resolution
print(fig1,'RGB_Image','-dpng','-r0')
```

In-Lab Task 5

Print an image at the resolution of 300 dots per inch (dpi) in vector graphic format. You should use the MATLAB help to find how to set resolution of the image in print function. You can print your figure in any of the following vector graphic format shown in Table 2.3.

Table 2.3: Image Functions for Display Image Information

Format Specification	Description	File Extension
'pdf'	Full page Portable Document Format (PDF) color	.pdf
'eps'	Encapsulated PostScript (EPS) Level 3 black and white	.eps
'epsc'	Encapsulated PostScript (EPS) Level 3 color	.eps
'eps2'	Encapsulated PostScript (EPS) Level 2 black and white	.eps
'epsc2'	Encapsulated PostScript (EPS) Level 2 color	.eps
'meta'	Enhanced Metafile (Windows only)	.emf
'svg'	SVG (scalable vector graphics)	.svg
'ps'	Full-page PostScript (PS) Level 3 black and white	.ps
'psc'	Full-page PostScript (PS) Level 3 color	.ps
'ps2'	Full-page PostScript (PS) Level 2 black and white	.ps
'psc2'	Full-page PostScript (PS) Level 2 color	.ps

2.2.4 Data Classes

Although we work with integer image coordinates, the values (intensities) of pixels are not restricted to integers. Table 2.4 lists the various data classes supported by MATLAB. The first ten entries in the table are referred to as numeric classes. The next entry is the logical class. The last two entries are used to represent text data, either as character arrays or string arrays. The Toolbox supports a subset of the data classes in Table 2.4**.** Most Toolbox functions support the double, single, uint8, int8, uint16, intl6,

uint32, int32, and logical data classes. Some Toolbox functions, such as those for image display and color-space conversion functions, support a smaller subset: double, single, uint8, uintl6, and logical. To determine the data type support for a specific

function, consult the function's reference page. Classes uint8 and logical are used extensively in image processing—they are the usual classes encountered when reading images with formats such as TIFF or JPEG. These classes use one byte to represent each pixel. Some scientific data sources, such as medical imagery, require more dynamic range than is provided by uint8, so the uintl6 and int16 classes are used often for such data. These classes use two bytes per pixel. The floating-point classes double and single are used for operations requiring floating-point accuracy. Double-precision floating-point uses 8 bytes per array element, whereas single-precision floating-point uses 4 bytes. The class of an image or other MATLAB object is determined using function class, whose syntax is

```
s = class(obj)
```

where s is one of the classes in the first column of Table 2.4.

Table 2.4: Types of Data Classes in MATLAB

Data Class	Description
double	Double-precision, floating-point numbers in the approximate range $\pm 10^{308}$ (8 bytes per element).
single	Single-precision floating-point numbers with values in the approximate range $\pm 10^{38}$ (4 bytes per element).
uint8	Unsigned 8-bit integers in the range [0, 255] (1 byte per element).
uint16	Unsigned 16-bit integers in the range [0, 65535] (2 bytes per element).
uint32	Unsigned 32-bit integers in the range [0, 4294967295] (4 bytes per element).
uint64	Unsigned 64-bit integers—See docs for values. (8 bytes per element).
int8	Signed 8-bit integers in the range [–128, 127] (1 byte per element).
int16	Signed 16-bit integers in the range [–32768, 32767] (2 bytes per element).
int32	Signed 32-bit integers in the range [–2147483648, 2147483647] (4 bytes per element).
int64	Signed 64-bit integers—See docs for values. (8 bytes per element).
logical	Values are 0 or 1 (1 byte per element).
char	Characters (2 bytes per element).
string	Strings (storage depends on the length of the string).

2.2.5 Converting Between Classes

The general syntax for converting between data classes is

```
B = className(A)
```

where className is one of the names in the first column of Table 2.4. For example,

if A is an array of class uint.8, we generate a double-precision array, B, using the command B = double (A). If C is an array of class double in which all values are in the range [0, 255] (but possibly containing fractional values), it can be converted to an uint8 array with the command D = uint8(C).

Table 2.5: Conversion between the classes

Function	Converts Input to	Valid Input Image Data Classes
im2uint8	uint8	logical, uint8, uint16, int16, single, and double
im2uint16	uint16	logical, uint8, uint16, int16, single, and double
im2int16	int16	logical, uint8, uint16, int16, single, and double
im2double	double	logical, uint8, uint16, int16, single, and double
im2single	single	logical, uint8, uint16, int16, single, and double
mat2gray	double in the full range [0, 1]	logical, uint8, int8, uint16, int16, uint32, int32, single, and double
im2bw	logical	uint8, uint16, int16, single, and double

2.2.6 Types of Images

1. Indexed images
2. Intensity images
3. Binary images
4. RGB images

2.2.6.1 Indexed Image

An indexed image consists of a data matrix, X, and a colormap matrix, map. The data matrix can be of class uint8, uint16, or double. The colormap matrix is an m-by-3 array of class double containing floating-point values in the range [0,1]. Each row of map specifies the red, green, and blue components of a single color. An indexed image uses direct mapping of pixel values to colormap values. The color of each image pixel is determined by using the corresponding value of X as an index into map. The value 1 points to the first row in map, the value 2 points to the second row, and so on.

A colormap is often stored with an indexed image and is automatically loaded with the image when you use the `imread` function. However, you are not limited to using the default colormap--you can use any colormap that you choose. The figure below illustrates the structure of an indexed image. The pixels in the image are represented by integers, which are pointers (indices) to color values stored in the colormap.

The following figure depicts an indexed image.

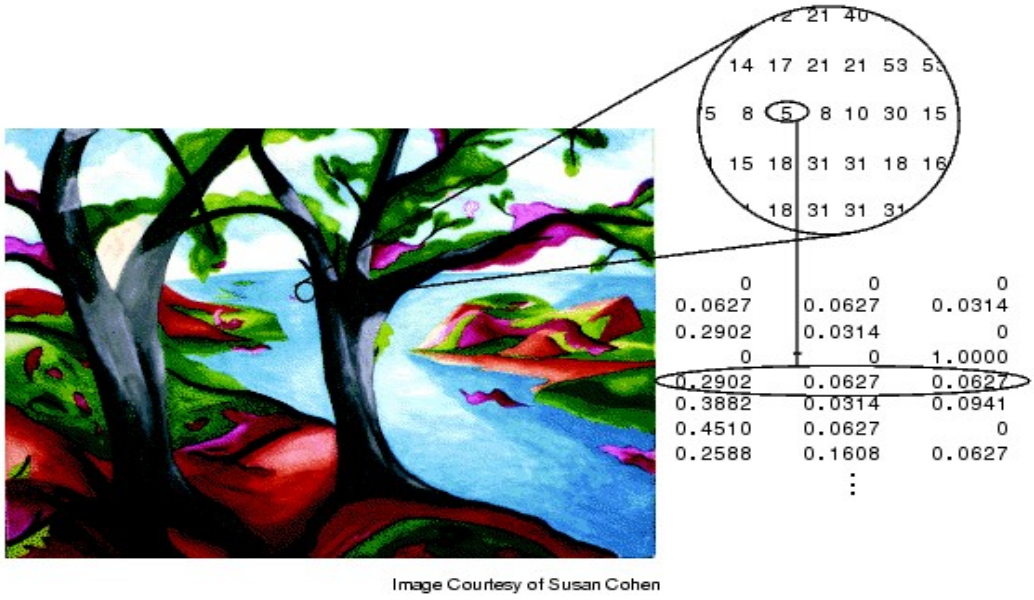


Figure 2.6: Indexed Image

2.2.6.2 Intensity Image

An intensity image is a data matrix, I, whose values represent intensities within some range. MATLAB stores an intensity image as a single matrix, with each element of the matrix corresponding to one image pixel. The matrix can be of class double, uint8, or uint16.While intensity images are rarely saved with a colormap, MATLAB uses a colormap to display them. The elements in the intensity matrix represent various intensities, or gray levels, where the intensity 0 usually represents black and the intensity 1, 255, or 65535 usually represents full intensity, or white. The figure below depicts an intensity image of class double.

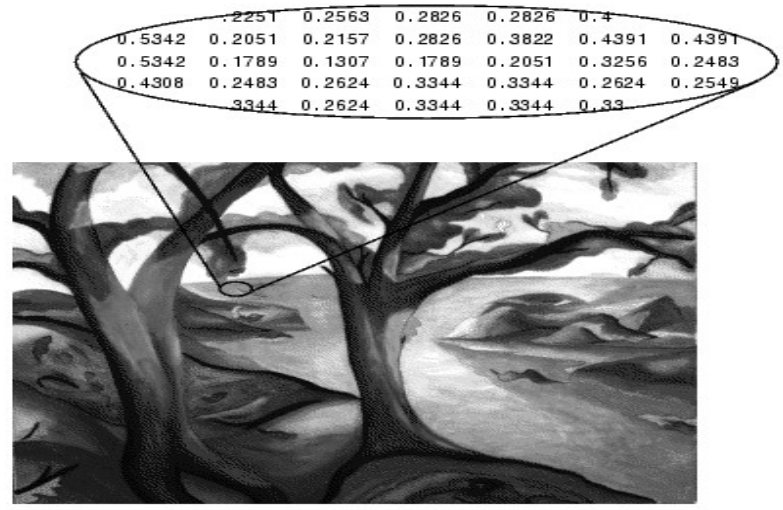


Figure 2.7: Intensity Image

2.2.6.3 Binary Image

In a binary image, each pixel assumes one of only two discrete values. Essentially, these two values correspond to on and off. A binary image is stored as a logical array of 0's (off pixels) and 1's (on pixels). The figure below depicts a binary image.

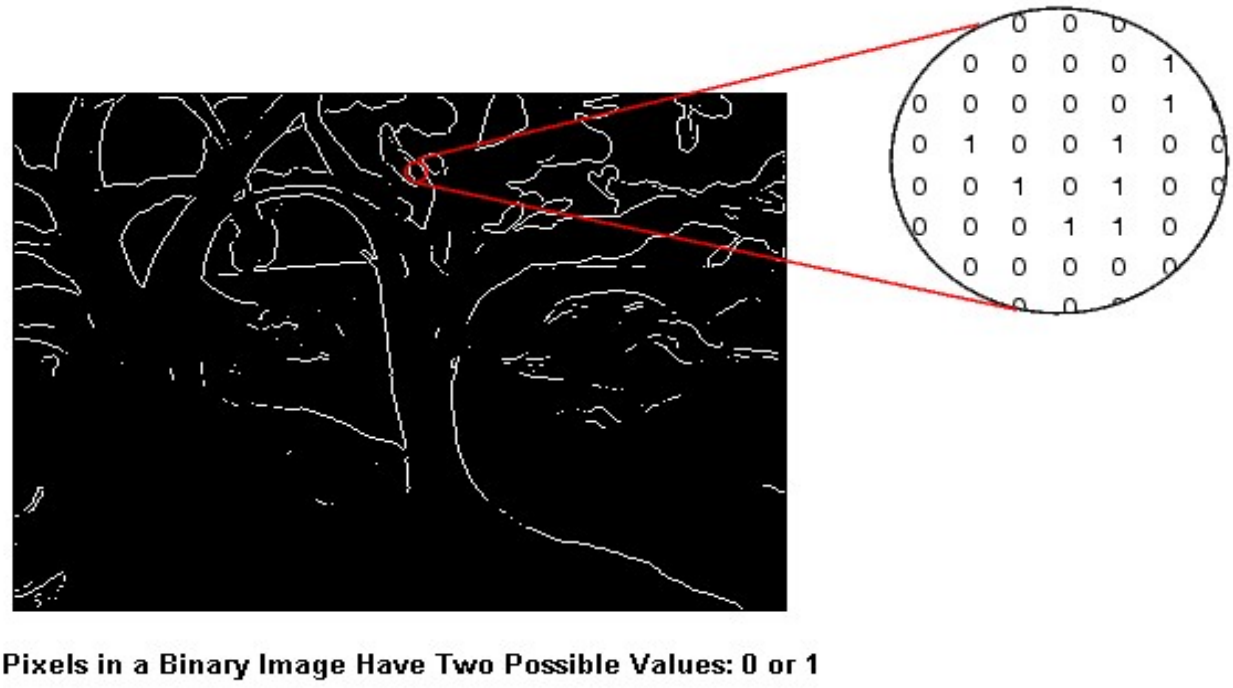


Figure 2.8 Binary Image

2.2.6.4 RGB Image

An RGB image, sometimes referred to as a true-color image, is stored in MATLAB as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel. RGB images do not use a palette. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million colors. The precision with which a real-life image can be replicated has led to the commonly used term true-color image.

An RGB array can be of class double, uint8, or uint16. In an RGB array of class double, each color component is a value between 0 and 1. A pixel whose color components are (0,0,0) is displayed as black, and a pixel whose color components are (1,1,1) is displayed as white. The three-color components for each pixel are stored along the third dimension of the data array.

For example, the red, green, and blue color components of the pixel (10,5) are stored in RGB (10,5,1), RGB (10,5,2), and RGB (10,5,3), respectively. The following figure depicts an RGB image of class double.

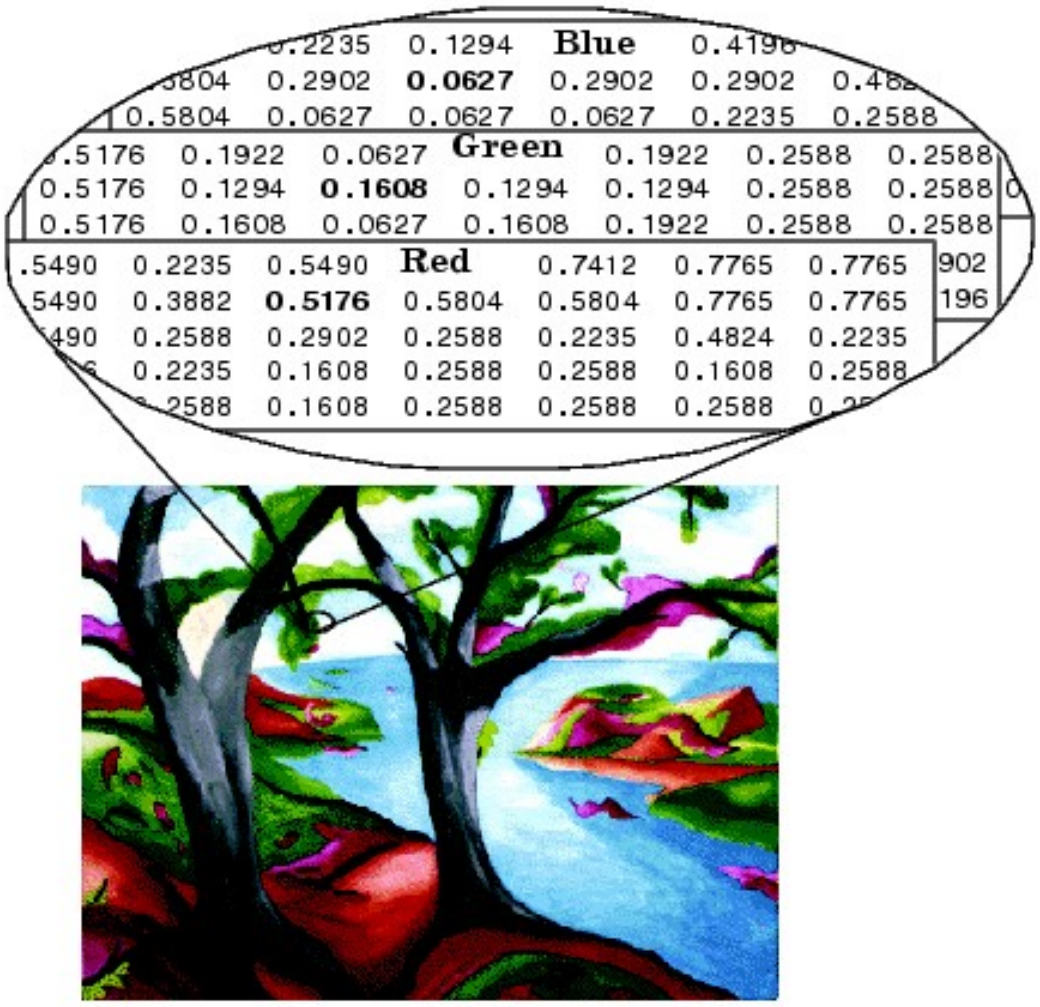


Figure 2.9 RGB Image

2.3 Post-Lab Tasks

Post-Lab Task 1 Thresholding or Gray to Binary Conversion

Threshold is simple concept of setting range of certain value to be a value. The basic purpose of thresholding in image processing is to adjust the pixel value of an image to certain value. Write a MATLAB code to set the pixel value to 0 if the original pixel value is below or equal to 128. Otherwise, if the original pixel value is above 128, the new pixel value will be 255.

Post-Lab Task 2 Generate Mirror Image

There are many ways to generate mirror image, in this task we will generate it using nested loops.

Try implementing the following steps to generate mirror image.

1. Read the image.

2. Start the outer loop from 1 to the total rows of the image.
3. Start the inner loop from 1 to the total columns of the image.
4. In inner loop, write the code to swap the values as shown below.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

Table 1 Input Image

a_{15}	a_{14}	a_{13}	a_{12}	a_{11}
a_{25}	a_{24}	a_{23}	a_{22}	a_{21}
a_{35}	a_{34}	a_{33}	a_{32}	a_{31}
a_{45}	a_{44}	a_{43}	a_{42}	a_{41}
a_{55}	a_{54}	a_{53}	a_{52}	a_{51}

Table 2 Mirror Image

Post-Lab Task 3 Generate Flipped Image

Write a MATLAB code that performs the following tasks without using built-in commands.

1. Reads a colored test image provided by the lab instructor during the lab.
2. Convert it into gray scale image.
3. Generates the flipped image of original image.
4. Store the flipped image with the name provided by the user.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

Table 1 Input Image

a_{51}	a_{52}	a_{53}	a_{54}	a_{55}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}

Table 3 Flipped Image

In Post -Lab tasks you may get help from the following syntax of flow control statements.

1. if and if-else

```
%%
% if syntax
if expression
    statements
end

% if-else syntax
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

Table 2.6: flow control statements in MATLAB

Statement	Description
`if`	`if`, together with `else` and `elseif`, executes a group of statements based on a specified logical condition.
`for`	Executes a group of statements a fixed (specified) number of times.
`while`	Executes a group of statements an indefinite number of times, based on a specified logical condition.
`break`	Terminates execution of a `for` or `while` loop.
`continue`	Passes control to the next iteration of a `for` or `while` loop, skipping any remaining statements in the body of the loop.
`switch`	`switch`, together with `case` and `otherwise`, executes different groups of statements, depending on a specified value or string.
`return`	Causes execution to return to the invoking function.
`try...catch`	Changes flow control if an error is detected during execution.
`parfor`	Parallel `for` loop, used with Parallel Computing Toolbox or MATLAB Coder. See the MATLAB help page on this function for additional details.

```
% for syntax
for index = first:last
    statements
end

% for syntax with increment
for index = first:increment:last
    statements
end

% Nested for loop syntax
for index1 = first1:increment1:last1
    statements1
    for index2 = start2:increment2:last2
        statements2
    end
    additional loop1 statements
end

% While Loop
while expression
    statements
end
% Nested While
while expression1
    statements1
    while expression2
        statements2
    end
    additional loop1 statements
end
```

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

3 Manipulate digital images by applying various intensity transformation techniques using MATLAB

3.1 Objectives

Demonstrate various Intensity Transformation techniques for image processing using MATLAB.

3.2 Pre-Lab

The term **spatial domain** refers to the image plane itself and methods in this category are based on direct manipulation of pixels in an image. In this Lab, we focus attention on a category of spatial domain processing which is called intensity (gray-level) transformations.

In order to carry a consistent theme, most of the examples in this Lab are related to image enhancement. This is a good way to introduce spatial processing because enhancement is highly intuitive and appealing, especially to beginners in the field.

Most of the spatial domain processes discussed in this Lab are denoted by the expression:

$$g(x,y) = T[f(x,y)]$$

where $f(x,y)$ is the input image, $g(x,y)$ is the output (processed) image and T is an operator on/defined over a specified neighbourhood about point (x,y). In addition, T can operate on a set of images, such as performing the addition of K images for noise reduction.

The principal approach for defining spatial neighbourhoods about a point (x_0, y_0) is to use a square or rectangular region cantered at (x_0, y_0), as Fig. 3.1 shows. The regions moved from pixel to pixel starting, say, at the top, left corner and, as it moves, it encompasses different neighbourhoods. Operator T is applied at each location (x,y) to yield the output, g, at that location. Only the pixels in the neighbourhood cantered at (x,y) are used in computing the value of g at (x,y).

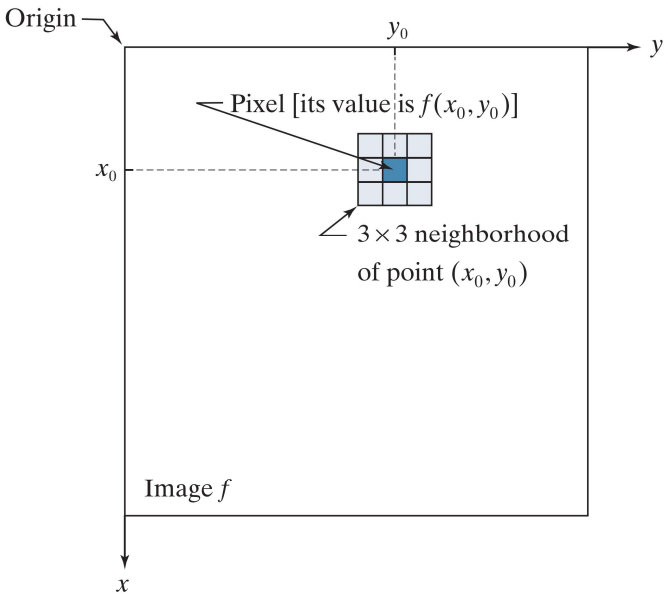


Figure 3.1: A 3 3 × neighbourhood about a point (x_0, y_0), in an image. The neighbourhood is moved from pixel to pixel in the image to generate an output image.

3.2.1 Intensity Transformation Functions

The simplest form of the transformation T is when the neighborhood in Fig. 3.1 is of size 1×1 (a single pixel). In this case, the value of g at (x, y) depends only on the intensity of f at that point and T becomes an intensity transformation function.

$$S = T(r)$$

where r denotes the intensity of f and s the intensity of g, both at the same coordinates (x, y) in the input and output images.

In-Lab Task 1

Use the MATLAB help to understand the basic intensity transformation function `imadjust`, and use the examples in help to produce various image transformation results.

```
%% In-Lab
Img= imread('peppers.png');
Img=im2double(Img);
Img_gray= rgb2gray(Img);
% Practice the following functions
figure
imshow(Img)
figure
imshow(Img_gray)
%--------------------------------------------------------------------------
% 1. J = imadjust
% Using imadjust by default values
figure
Img_adj= imadjust(Img_gray,[0.3 0.7], []);
imshow(Img_adj)
```

Your task is to explore all the input values of the following syntax and apply.

```
J = imadjust(I,[low_in high_in],[low_out high_out],gamma);
```

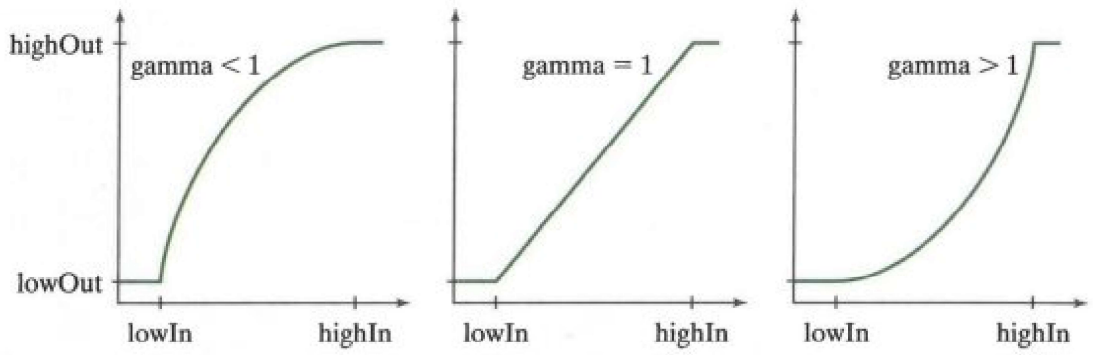


Figure 3.2: Impact of Gamma in `imadjust`

In-Lab Task 2

Use the MATLAB `imadjust` to display and analyse the impact of using different values of gamma. Display images using at least three different values of gamma. Briefly discuss the observations.

In-Lab Task 3

Use the MATLAB help to understand the function `imcomplement`, and use the examples in help to produce various image transformation results. Apply the `imcomplement` function to RGB image, display the results and discuss.

You should compare your input and output image in one figure and label them Input Image and Output Image as shown in Figure 3.3.

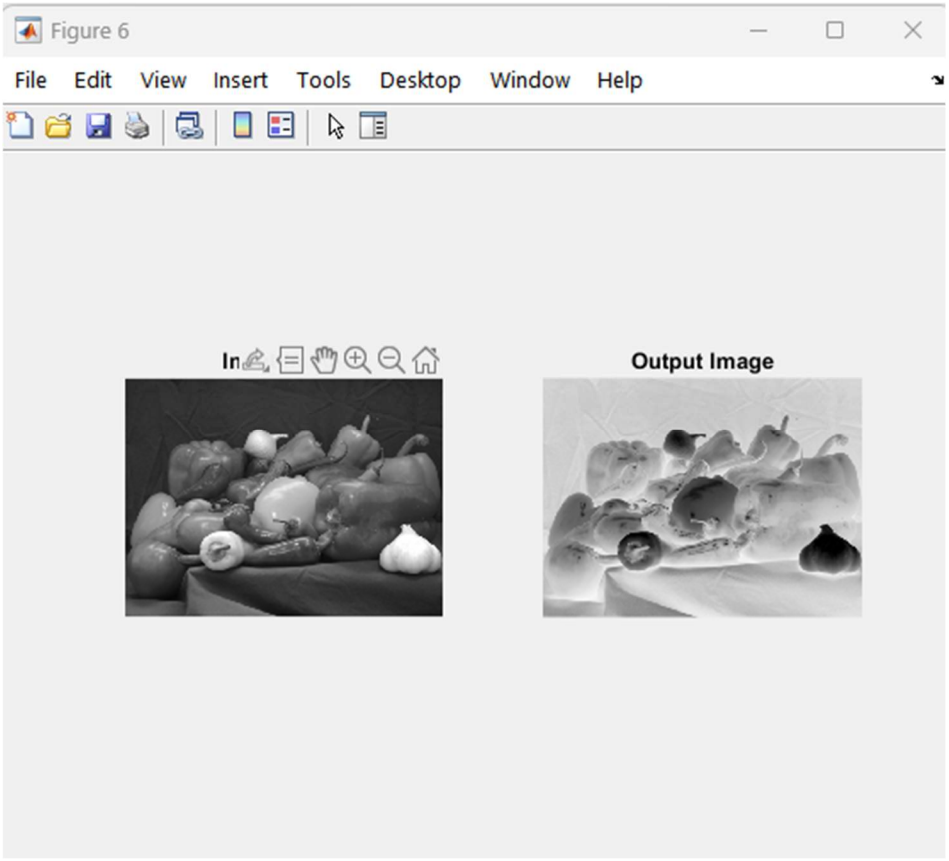


Figure 3.3: Comparison of Input and Output Image after applying `imcomplement` function.

In-Lab Task 4

Use the MATLAB intensity transformation function `imadjust` to create a negative of the image. Compare the output of the task with results obtained in the In-Lan Task2.

3.2.2 Contrast Stretching

It is of interest to be able to use function `imadjust` "automatically," without having to deal with the low and high parameters discussed above. Function `stretchlim` can be used for this purpose; its basic syntax is

```
% 3. LowHigh = stretchlim(f)
LowHigh = stretchlim(Img_gray);
```

```
>> LowHigh

LowHigh =

    0.0941
    0.9207
```

where `LowHigh` is a two-element vector of a lower and upper limit that can be used to achieve contrast stretching.

By default, values in `LowHigh` specify the intensity levels that saturate the bottom and top 1% of all pixel values in f. We use function `stretchlim` with function `imadjustas` follows:

```
% 3. LowHigh = stretchlim(f)
LowHigh = stretchlim(Img_gray);
figure
Img_adj= imadjust(Img_gray,LowHigh, [1 0]);
imshow(Img_adj)
```

3.2.3 Logarithmic and Contrast-Stretching Transformations

Logarithmic and contrast-stretching transformations are tools for dynamic rangemanipulation. Logarithm transformations are implemented using the expression.

$$g = c * log(l + f)$$

where c is a constant and f is floating point. The shape of this transformation is similar to the gamma curve in Fig. 3.2. However, the shape of the gamma curve is variable, whereas the shape of the log function is fixed.

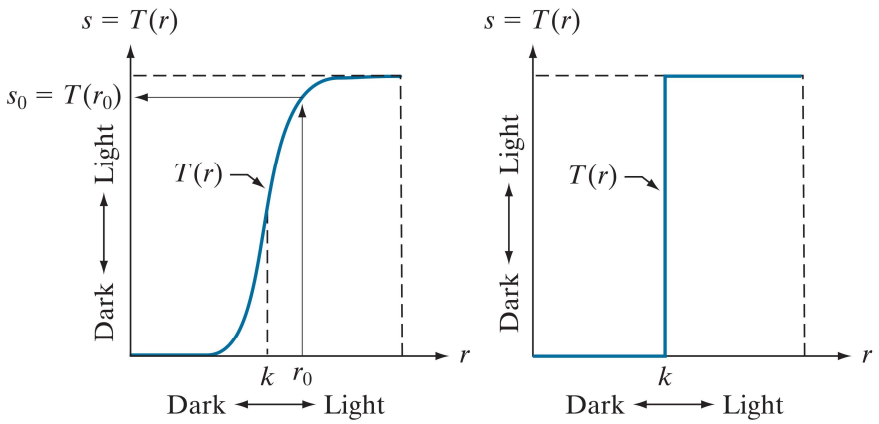


Figure 3.4: Intensity transformation functions. (a) Contrast stretching function (b) Thresholding function

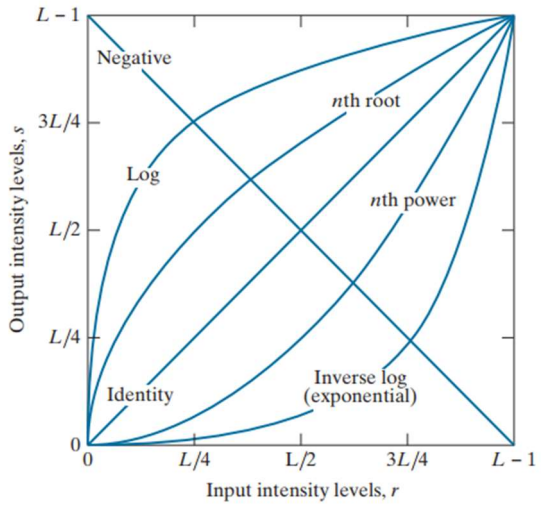


Figure 3.5: Some basic Intensity Transformation Functions

One of the principal uses of the log transformation is to compress dynamic range. By computing the log, a dynamic range on the order of, for example, 10^6, is reduced to approximately 14 $[i.e., loge(10^6) = 13.8]$, which is much more manageable.

A function with the shape in Fig. 3.4(a) is called a contrast-stretching transformation because it expands a narrow range of input intensity values into a wide (stretched) range of output values. The result is an image of higher contrast. In the limiting case shown in Fig. 3.4(b), the output is a binary image. This limiting function, which binarizes an image, is called a thresholding function.

Contrast Stretching Transformation

The function in Fig. 3.4(a) can be expressed as

$$s = T(r) = \frac{1}{1 + \left(\frac{k}{r}\right)^E}$$

where r denotes the intensities of the input image, s denotes the corresponding intensity values in the output image, k is a constant, and E controls the slope of the function.

Log Transformation

The general form of the log transformation in Fig. 3.4 is

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

where c is a constant and it is assumed that r ≥ 0.

Power – Law Transformation

Power-law transformations have the form.

$$s = cr^{\gamma}$$

where *c* and g are positive constants.

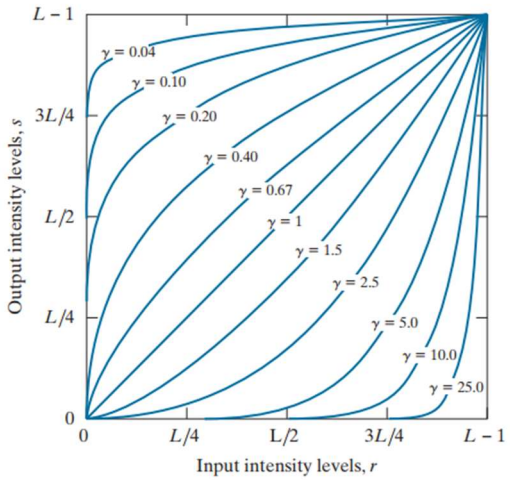


Figure 3.6: Plot of different gamma functions in Power Law

3.3 Post-Lab Tasks

Write a custom function for Intensity Transformation which can be utilized for contrast stretching, log and power law transformation. Test each case of transformation for different values of constant and other variables. Discuss the results.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ Date: _____

4 Manipulate digital images by organizing pixel intensities for histogram equalization and matching using MATLAB

4.1 Objectives

To display image histogram using MATLAB.
To manipulate image histogram for equalization using MATLAB.
To demonstrate image for desired histogram matching using MATLAB

4.2 Pre-Lab

Intensity transformation functions based on information extracted from image, intensity histograms play a central role in image processing, in areas such as enhancement, compression, segmentation, and description. Histogram is a bar-graph used to profile the occurrence of each gray level in the image. Mathematically it is represented as

$$h(r_k) = n_k$$

Where r_k is k[th] grey level and n_k is number of pixels having that Grey level. Histogram can tell us whether image was scanned properly or not. It gives us idea about tonal distribution in the image. Histogram equalization can be applied to improve appearance of the image. Histogram also tells us about objects in the image. Object in an image have similar gray levels so histogram helps us to select threshold value for object detection. Histogram can also be used for image segmentation.

Assume for a moment that intensity levels are continuous quantities normalized to the range [0, 1] and let P_r denote the probability density function (PDF) of the intensity levels in a given image, where the subscript is used for differentiating between the PDFs of the input and output images. Suppose that we perform the following transformation on the input levels to obtain output(processed) intensity levels "S", where w is the dummy variable for integration.

$$s = Tr = \int_0^r Pr(w)dw$$

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

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 in the previous section, 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 an image that has a specified histogram is called histogram matching or histogram specification.

Syntax

```
imhist(I,n) imhist(X,map)
[counts,x] = imhist(...)
```

Now you are required to the use the `imhist()` command and perform histogram formation of any test image, Also briefly explain that what type of information can be extracted from the output.

Read about Histogram, how it is formed, histogram equalization and histogram matching before coming for lab.

4.3 In Lab Tasks

In-Lab Task 1

4.3.1.1 Histogram formation

Write a MATLAB code that show the histogram of the image passed to it, without using the build in command `imhist()`.

```
%% In-Lab
Img= imread('peppers.png');
Img=im2double(Img);
Img_gray= rgb2gray(Img);
% Practice the following functions
figure
imshow(Img)
figure
imshow(Img_gray)
[N M] = size(Img_gray)
```

```
%% Histogram Bar Plot
figure
h=imhist(Img_gray)
h1 = h(1:10:256);
horz = 1:10:256;
bar(horz, h1)
axis([0 255 0 7200])
set(gca, 'xtick', 0:50:255)
set(gca, 'ytick', 0:2000:7200)
title('Bar Graph - Histogram of Gray Image')
```

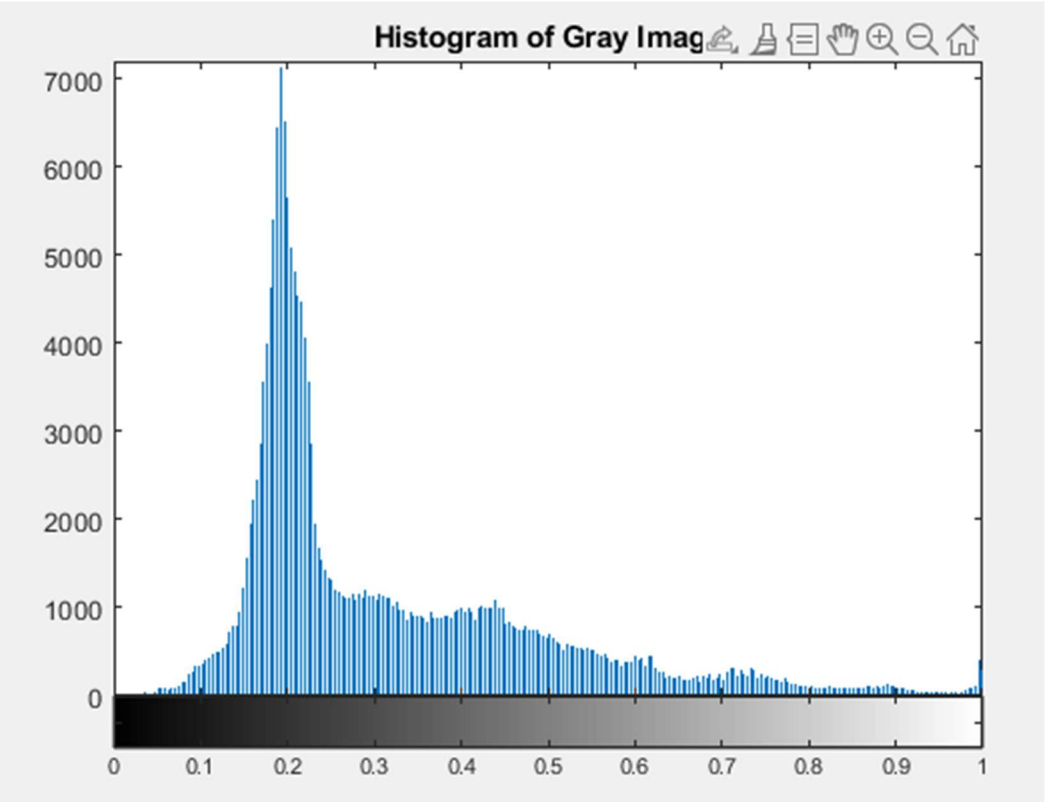


Figure 4.1: Histogram of Gray Image

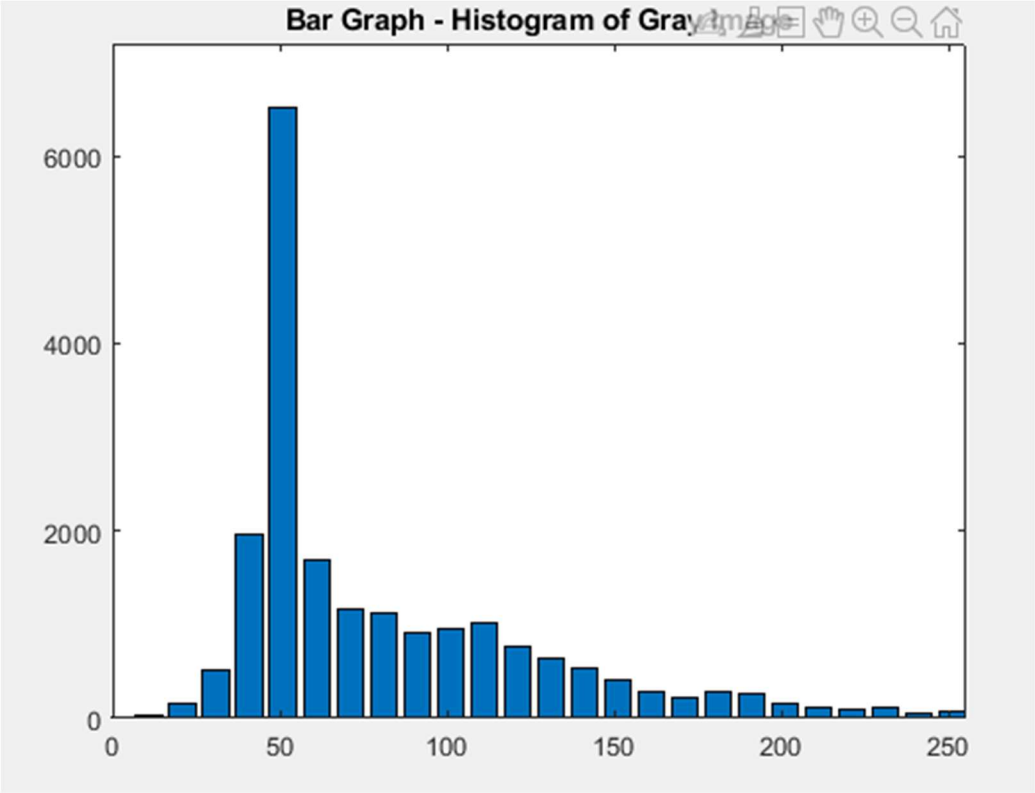


Figure 4.2: Bar Graph of Histogram of Gray Image

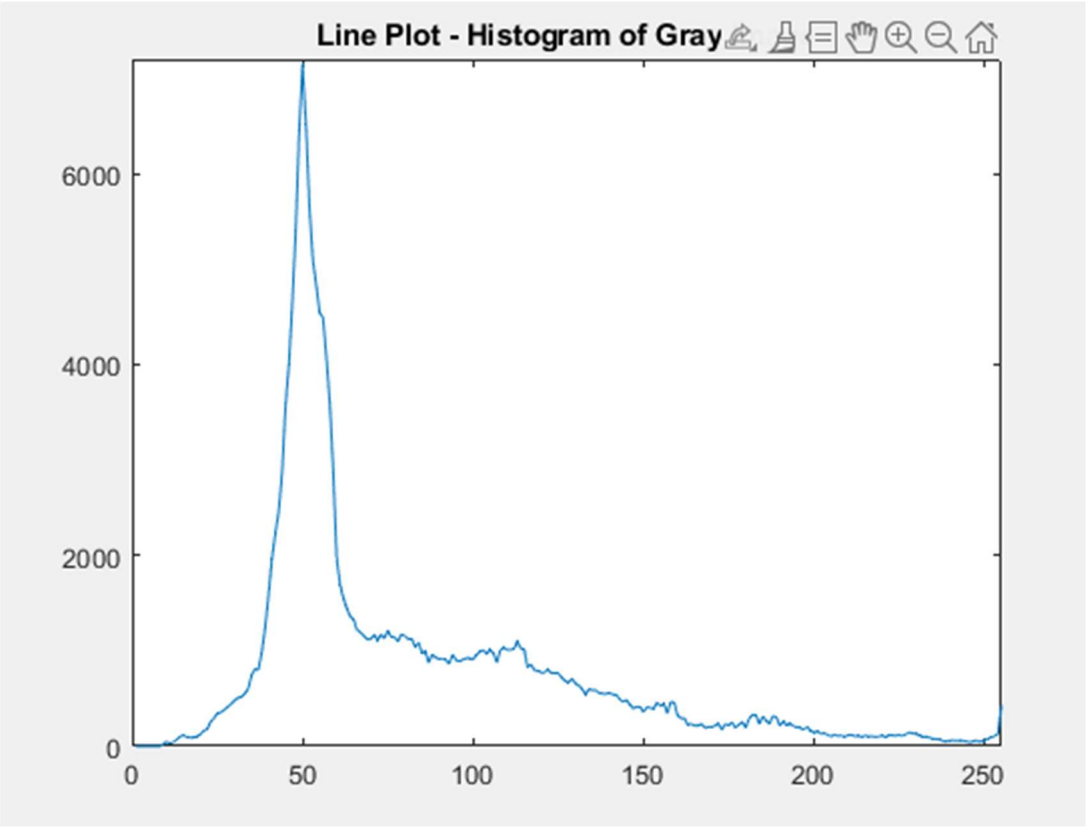


Figure 4.3: Line Plot of Histogram of Gray Image

Table 4.1: Attributes for functions stem and plot

Symbol	Color	Symbol	Line Style	Symbol	Marker
k	Black	–	Solid	+	Plus sign
w	White	– –	Dashed	o	Circle
r	Red	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
b	Blue	none	No line	x	Cross
c	Cyan			s	Square
y	Yellow			d	Diamond
m	Magenta			none	No marker

In-Lab Task 2

4.3.1.2 Histogram Equalization

Write a MATLAB code that apply histogram equalization on the image passed to it, without using the built-in command.

```
%% Histogram Equalization
J = histeq(Img_gray);

% Display the original and enhanced images
figure
subplot(1,2,1)
imshow(Img_gray)
title('Orignal Image')
subplot(1,2,2)
imshow(J)
title('Histogram Equalization Image')
```

Figure 4.4: Comparison of Original and Histogram Equalized Image

4.4 Post-Lab Tasks

Write a MATLAB code that perform the histogram matching of the grayscale image and an intensity transformed grayscale image.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

5 Manipulate the input image by constructing spatial transformations using `imfilter` in MATLAB

5.1 Objectives

To perform spatial transformation to the Input Image using imfilter.

5.2 Pre-Lab Theory

5.2.1 Spatial Filtering

Spatial filtering in image processing involves the manipulation of pixel values within an image based on their spatial relationships. It employs a small matrix, known as a filter or kernel, which is systematically applied to the image by sliding it across each pixel. The filter performs mathematical operations on the pixel values in its local neighborhood, producing a new value for the center pixel. This process, applied throughout the entire image, results in various effects such as smoothing, sharpening, edge detection, or noise reduction. Smoothing filters, like Gaussian or mean filters, reduce noise and blur, while sharpening filters enhance image details. Edge detection filters highlight boundaries. Spatial filtering is fundamental to tasks like noise reduction, feature extraction, and overall image enhancement in fields such as computer vision and digital image processing. The specific choice of filter and its parameters depends on the desired outcome of the image processing application.

5.2.2 Linear Spatial Filtering

Linear spatial filtering in image processing involves convolving an image with a filter or kernel to perform operations like blurring, sharpening, or edge detection. The filter is a small matrix, and each element represents a weight. The convolution operation computes the weighted sum of pixel values within a local neighborhood, producing a new value for the center pixel. For instance, a smoothing filter, like a Gaussian filter, replaces each pixel with the weighted average of its neighbors, reducing noise and creating a blurred effect. In contrast, a sharpening filter, such as a Laplacian filter, enhances edges and details by emphasizing intensity variations. Linear spatial filtering is a key technique for manipulating image characteristics and extracting features, playing a vital role in tasks like image enhancement and preprocessing in computer vision applications.

Convolution and correlation are two closely related operations in linear spatial filtering, both involving the application of a filter (or kernel) to an image. These operations are fundamental in tasks like image processing and computer vision.

Convolution:

Convolution is the most common operation in linear spatial filtering. It involves flipping the filter both horizontally and vertically and then sliding it over the image, computing the sum of element-wise products at each position. The result of convolution is a filtered image, with the filter emphasizing or de-emphasizing certain features based on the weights in the filter. Mathematically, the convolution of an image I with a filter K is denoted as I∗K, and at each position (x,y), it is calculated as:

$$(I * K)(x, y) = \sum_i \sum_j I(x + i, y + j) \cdot K(i, j)$$

Correlation:

Correlation is similar to convolution but without flipping the filter. In correlation, the filter is slid over the image without reversing its values. The result is another way of obtaining a filtered image, but the emphasis on features may be different from convolution.

Mathematically, the correlation of an image I with a filter K is denoted as I⋆K, and at each position. (x,y), it is calculated as:

$$(I \star K)(x,y) = \sum_i \sum_j I(x+i, y+j) \cdot K(i,j)$$

In practice, convolution is more widely used in image processing, and the terms "convolution" and "correlation" are sometimes used interchangeably. The key difference lies in whether or not the filter is flipped during the operation.

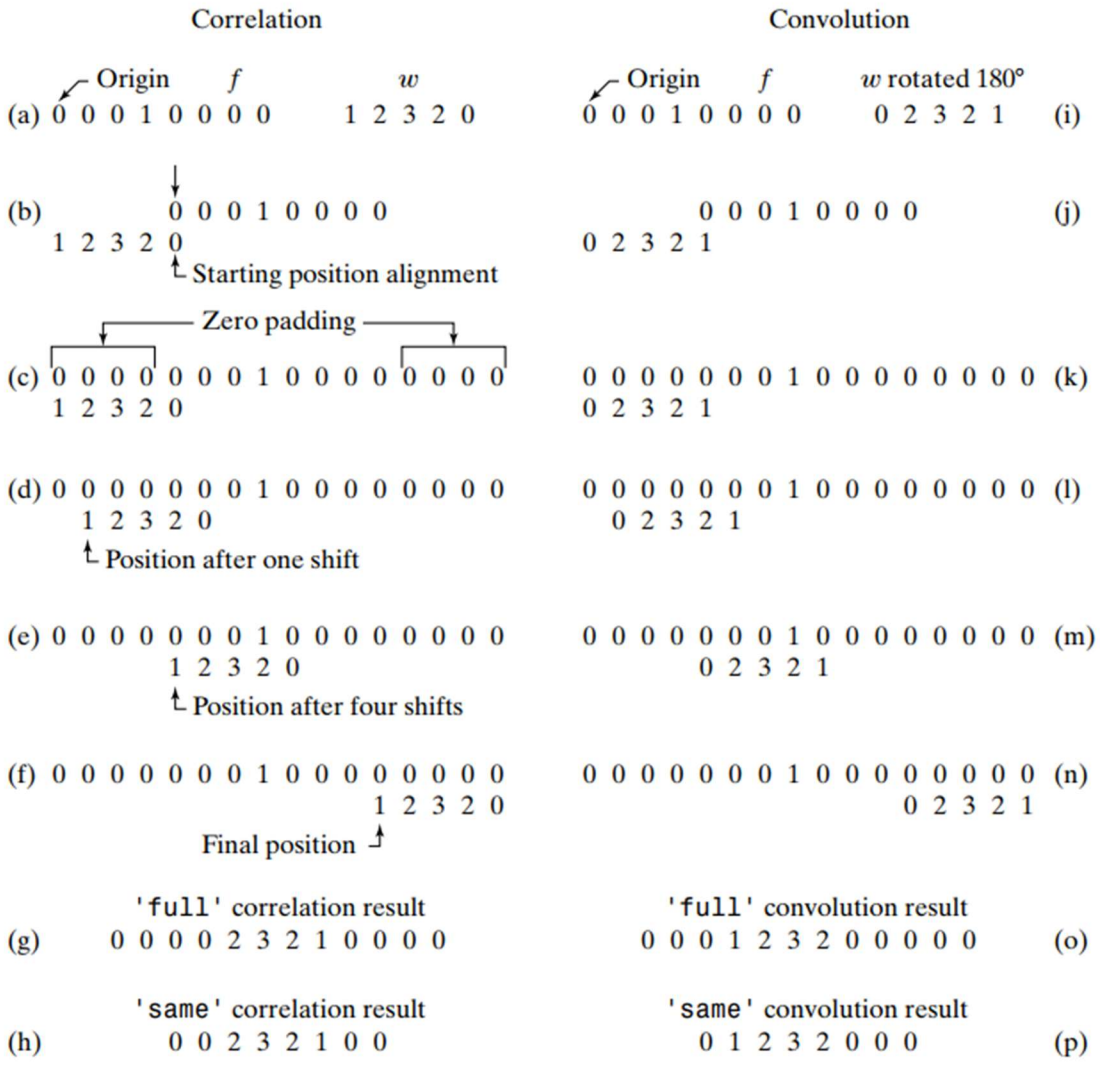


Figure 5.1: 1-D Correlation and Convolution

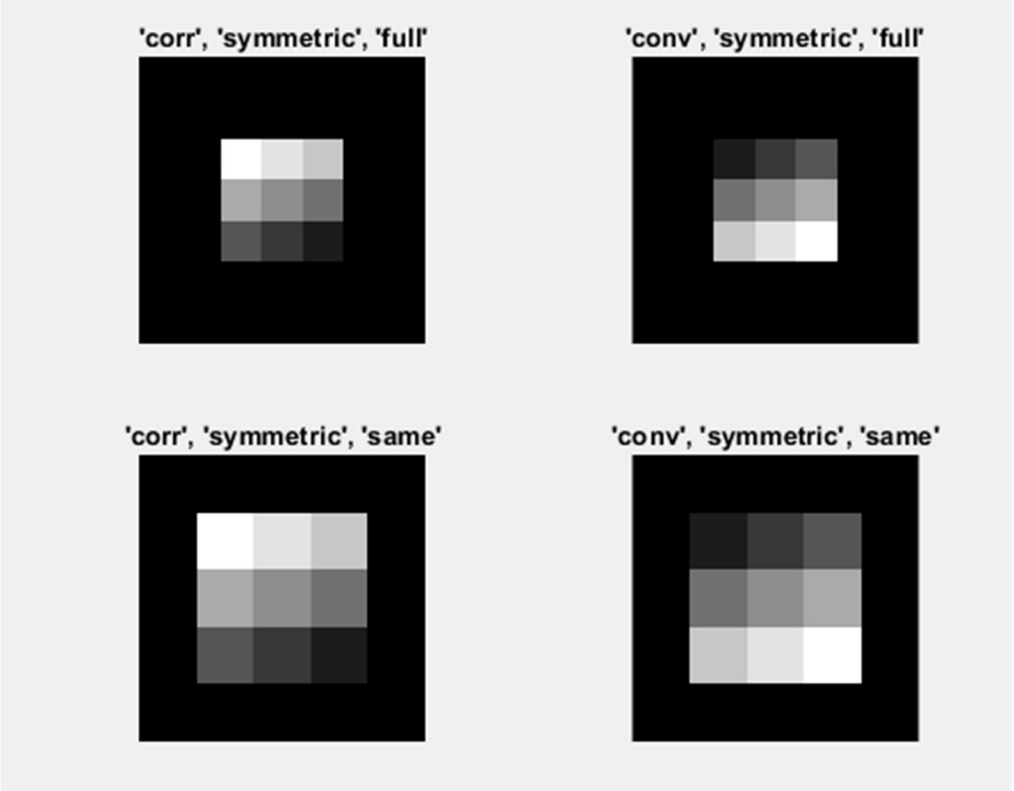
5.3 In Lab Task

In-Lab Task 1

Apply the spatial filtering to the grayscale image using 'imfilter' and explore different options.

```
%% Spatial Filtering
% Generate a Matrix f
f= [ 0 0 0 0 0; 0 0 0 0 0; 0 0 1 0 0; 0 0 0 0 0;0 0 0 0 0]
% Generate a mask w
w= [1 2 3; 4 5 6; 7 8 9]
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'full'")
filt_1= imfilter(f, w, 'corr', 'symmetric', 'full')
% Apply imfilter
disp("Using Options - 'conv', 'symmetric', 'full'")
filt_2= imfilter(f, w, 'conv', 'symmetric', 'full')
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same'")
filt_3= imfilter(f, w, 'corr', 'symmetric', 'same')
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same'")
filt_4= imfilter(f, w, 'conv', 'symmetric', 'same')
```

```
figure
subplot(2,2,1)
imshow(filt_1, [])
title("'corr', 'symmetric', 'full'")
subplot(2,2,2)
imshow(filt_2, [])
title("'conv', 'symmetric', 'full'")
subplot(2,2,3)
imshow(filt_3, [])
title("'corr', 'symmetric', 'same'")
subplot(2,2,4)
imshow(filt_4, [])
title("'conv', 'symmetric', 'same'")
```



5.4 Post-Lab Tasks

Write a MATLAB code that perform the special filtering on grayscale image and show filtered image using the following variations:

Option
Padding Options
numeric scalar, X
`'symmetric'`
`'replicate'`
`'circular'`
Output Size
`'same'`
`'full'`
Correlation and Convolution Options
`'corr'`
`'conv'`

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

6 Assemble and apply linear and non-linear spatial filters to manipulate digital images using MATLAB

6.1 Objectives

- To perform image averaging by applying mean filter using MATLAB
- To perform image averaging by applying weighted average filter using MATLAB
- To perform image filtering by applying median filter using MATLAB

6.2 Pre-Lab Theory

Spatial Filtering is sometimes also known as neighborhood processing. Neighborhood processing is an appropriate name because you define a center point and perform an operation (or apply a filter) to only those pixels in predetermined neighborhood of that center point. The result of the operation is one value, which becomes the value at the center point's location in the modified image. Each point in the image is processed with its neighbors. 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 nonlinear spatial filtering. There are some of the filters mentioned below

1. Averaging filtering
2. Weighted average filtering
3. Median filtering
4. Laplacian filtering
5. Object oriented shape filtering

6.2.1.1 Mean Filter

Mean filtering is also named as Smoothing, Averaging or Box filtering**.** Mean filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of mean filtering is simply to replace each pixel value in an image with the mean ("average") value of its neighbours, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean. Often a 3×3 square kernel is used, as shown below, although larger kernels (e.g. 5×5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel).

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

6.2.1.2 Median Filter

Median filtering is also known as rank filtering. The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.Like the mean filter, the median filter considers each pixel in the image in turn and looks at its nearby neighbours to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighbouring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighbourhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighbourhood under consideration contains an even number of pixels, the average of the two middle pixel values is used).

6.2.1.3 Laplacian Filter

Laplacian filter is also known as Laplacian of Gaussian (LoG). The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single gray-level image as input and produces another gray-level image as output.The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$\partial^2 I \qquad \partial^2 I \; L(x, y) = \partial x_2 + \partial y_2$$

6.3 In Lab Tasks

In-Lab Task 1

```
%% Read an Input Image
Img= imread('peppers.png');
Img=im2double(Img);
Img_gray= rgb2gray(Img);
% Practice the following functions
figure
imshow(Img_gray)
Img= Img_gray;
title ("Input Image")
```

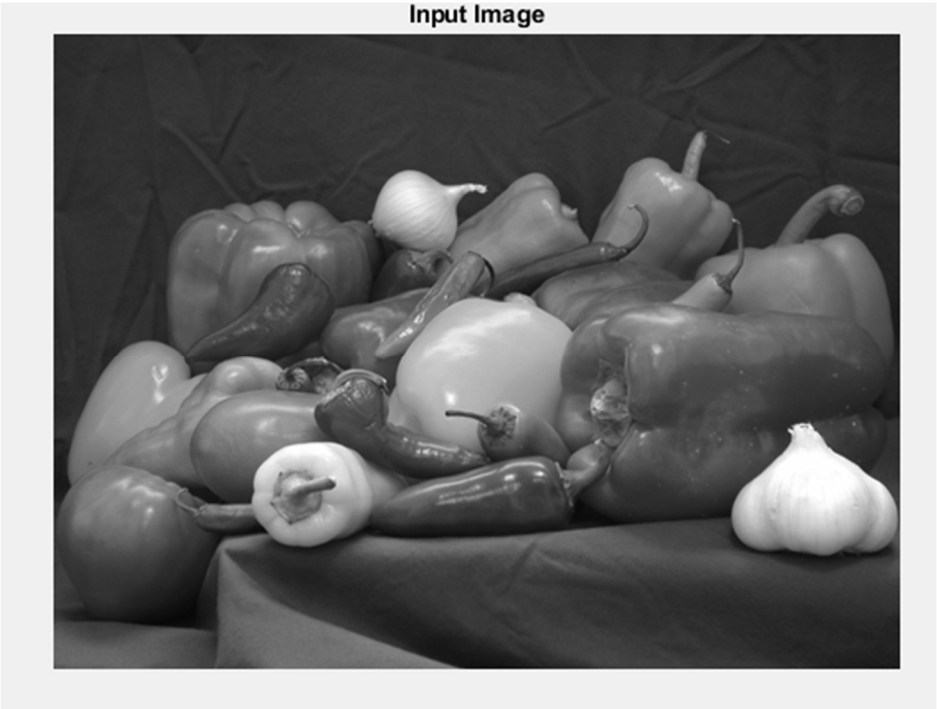


Figure 6.1: Input Image

```
%% Create Filter Using fspecial
% Define the filter size
% h_size= input(' Enter the filter size [m n]...:');
% h_size= [4 4];
% Generate a filter - Averaging
% h= fspecial('average', h_size);
h1= fspecial('average',[4 4]);
h2= fspecial('average',[10 10]);
h3= fspecial('average',[20 20]);
h4= fspecial('average',[50 50]);
```

```
%% Apply imfilter Using Different Options
disp("Using Options - 'corr', 'symmetric', 'full'")
filt_1= imfilter(Img, h1, 'corr', 'symmetric', 'full');
% Apply imfilter
disp("Using Options - 'conv', 'symmetric', 'full'")
filt_2= imfilter(Img, h1, 'conv', 'symmetric', 'full');
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same'")
filt_3= imfilter(Img, h1, 'corr', 'symmetric', 'same');
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same'")
filt_4= imfilter(Img, h1, 'conv', 'symmetric', 'same');
```

```
figure
subplot(2,2,1)
imshow(filt_1, [])
title("'corr', 'symmetric', 'full', Average [4 4] ")
subplot(2,2,2)
imshow(filt_2, [])
title("'conv', 'symmetric', 'full', Average [4 4] ")
subplot(2,2,3)
imshow(filt_3, [])
title("'corr', 'symmetric', 'same', Average [4 4] ")
subplot(2,2,4)
imshow(filt_4, [])
title("'conv', 'symmetric', 'same', Average [4 4] ")
```

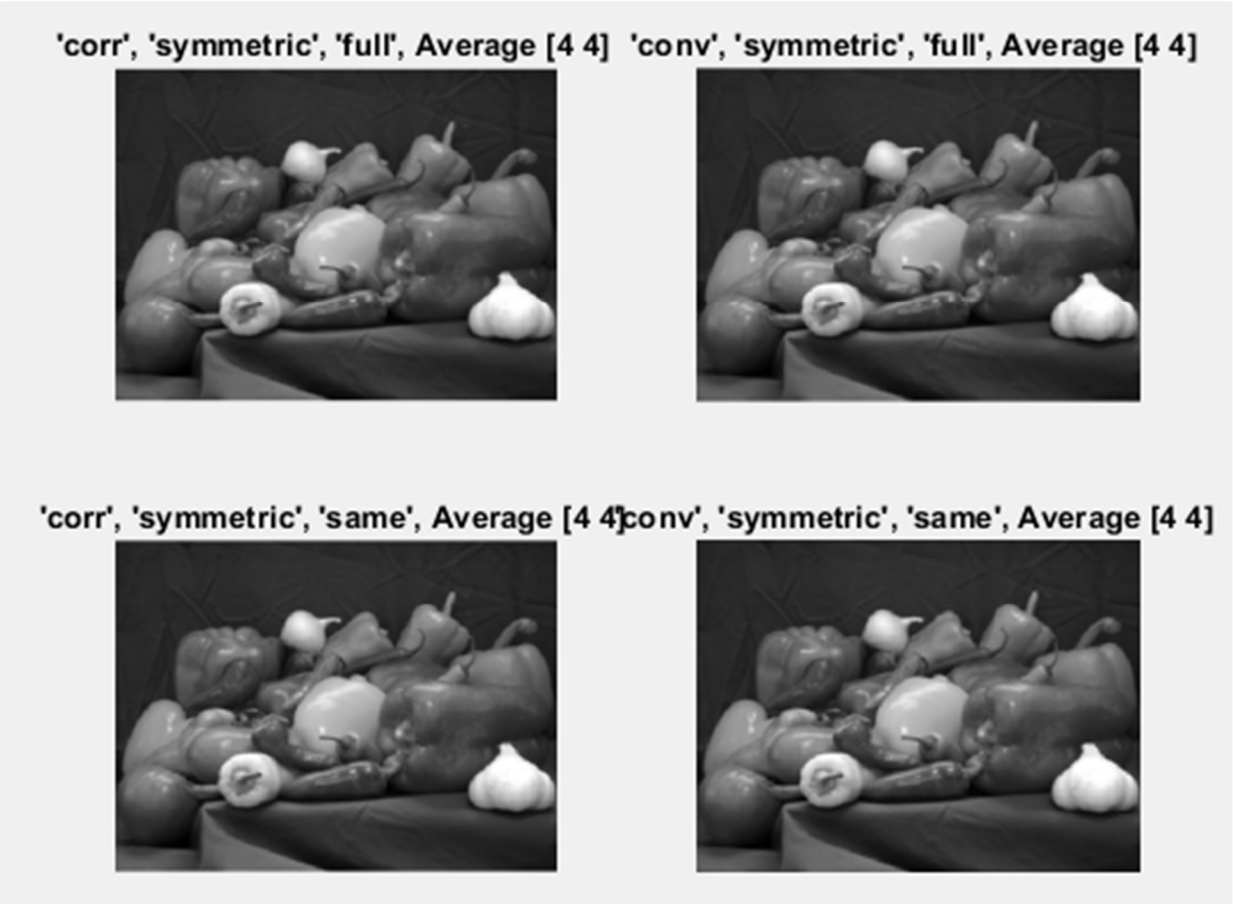


Figure 6.2: Applying Filter Using Different options of Filter

```
%% Impact of Chaging Filter Size - Average
disp("Using Options - 'corr', 'symmetric', 'full' and Filter size [4 4]")
filt_1= imfilter(Img, h1, 'corr', 'symmetric', 'full');
% Apply imfilter
disp("Using Options - 'conv', 'symmetric', 'full'and Filter size [10 10]")
filt_2= imfilter(Img, h2, 'conv', 'symmetric', 'full');
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same' and Filter size [20 20]")
filt_3= imfilter(Img, h3, 'corr', 'symmetric', 'same');
% Apply imfilter
disp("Using Options - 'corr', 'symmetric', 'same' and Filter size [50 50] ")
filt_4= imfilter(Img, h4, 'conv', 'symmetric', 'same');


figure
subplot(2,2,1)
imshow(filt_1, [])
title("Average Filter 4x4")
subplot(2,2,2)
imshow(filt_2, [])
title("Average Filter 10x10")
subplot(2,2,3)
imshow(filt_3, [])
title("Average Filter 20x20")
subplot(2,2,4)
imshow(filt_4, [])
title("Average Filter 50x50")
```

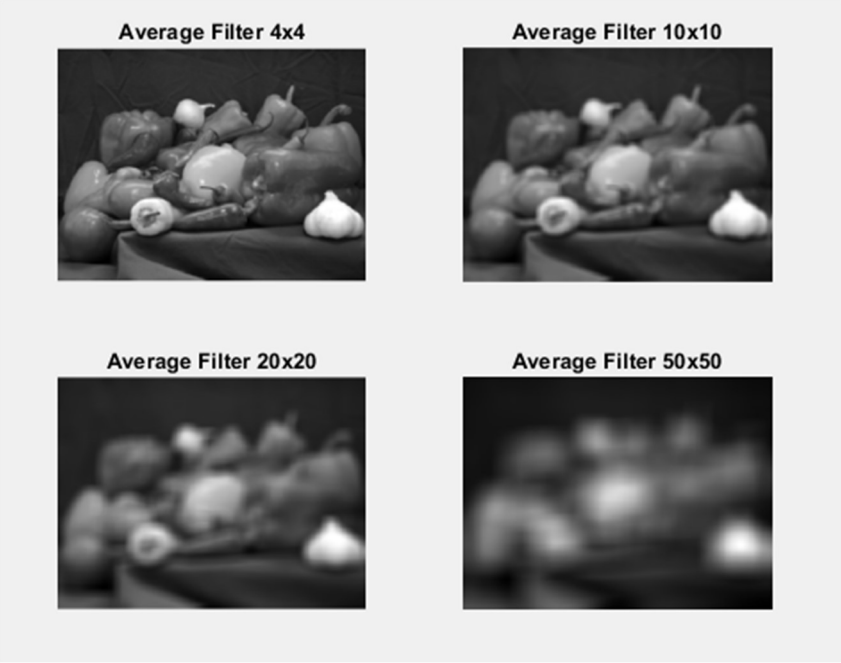


Figure 6.3: Impact of Filter Size

```
%% Creatig different types of filter
h_size = [4 4];
h_disk= fspecial('disk',4);
h_gaussian = fspecial('gaussian',h_size);
h_laplacian= fspecial('laplacian',0.3);
h_log= fspecial('log',h_size);
```

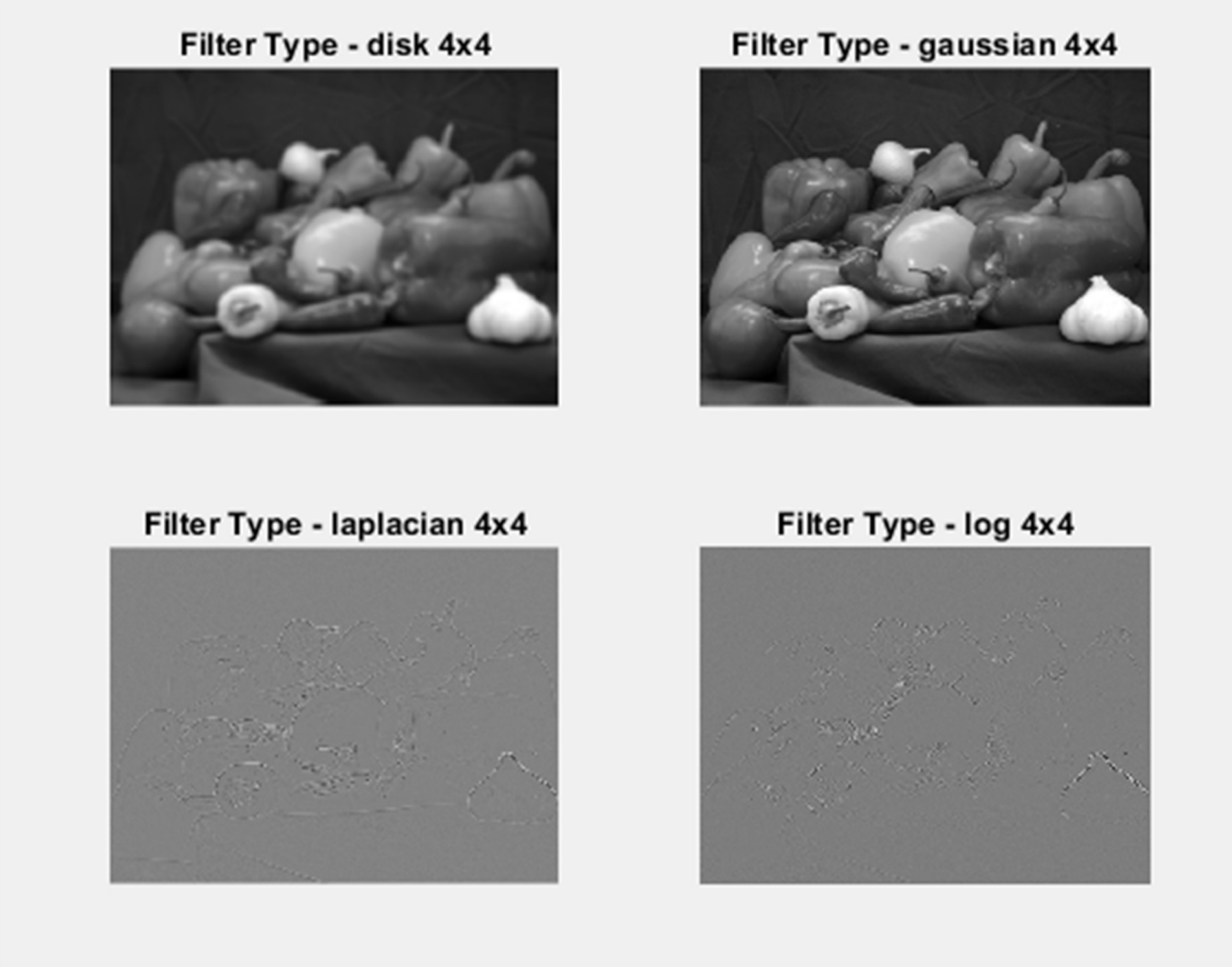


Figure 6.4: Applying Different types of Filters

6.4 Post-Lab Tasks

Write a MATLAB function that ask the user to choose among the different filters to apply on the input image and filter options and values according to the desired filter.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

7 Manipulate and measure frequency components for digital image processing in the frequency domain using MATLAB

7.1 Objectives

- To manipulate digital image between spatial and frequency domain using MATLAB
- To perform magnitude and angle of digital image in frequency domain using MATLAB

7.2 Pre-Lab Theory

The general idea is that the image ($f(x, y)$ of size M x N) will be represented in the frequency domain ($F(u, v)$). The equation for the two-dimensional discrete Fourier transform (DFT) is:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

The concept behind the Fourier transform is that any waveform that can be constructed using a sum of sine and cosine waves of different frequencies. The exponential in the above formula can be expanded into sines and cosines with the variables u and v determining these frequencies.The inverse of the above discrete Fourier transform is given by the following equation:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

Thus, if we have $F(u, v)$, we can obtain the corresponding image ($f(x, y)$) using the inverse, discrete Fourier transform.

MATLAB has three functions to compute the DFT:

- `fft` -for one dimension (useful for audio)
- `fft2` -for two dimensions (useful for images)
- `fftn` -for n dimensions

MATLAB has three functions that compute the inverse DFT:

- `ifft`
- `ifft2`
- `ifftn`

The following convolution theorem shows an interesting relationship between the spatial domain and frequency domain.

$$f(x,y) * h(x,y) \leftrightarrow H(u,v)F(u,v)$$

and, conversely,

$$f(x,y)h(x,y) \leftrightarrow H(u,v) * G(u,v)$$

The symbol "*" indicates convolution of the two functions. The important thing to extract out of this is that the multiplication of two Fourier transforms corresponds to the convolution of the associated functions in the spatial domain.

7.3 Pre-Lab Task

Write a MATLAB code capable of finding the Fourier transform of the image ("`cameraman.tif`"). Required output should be identical to Figure 7-1.

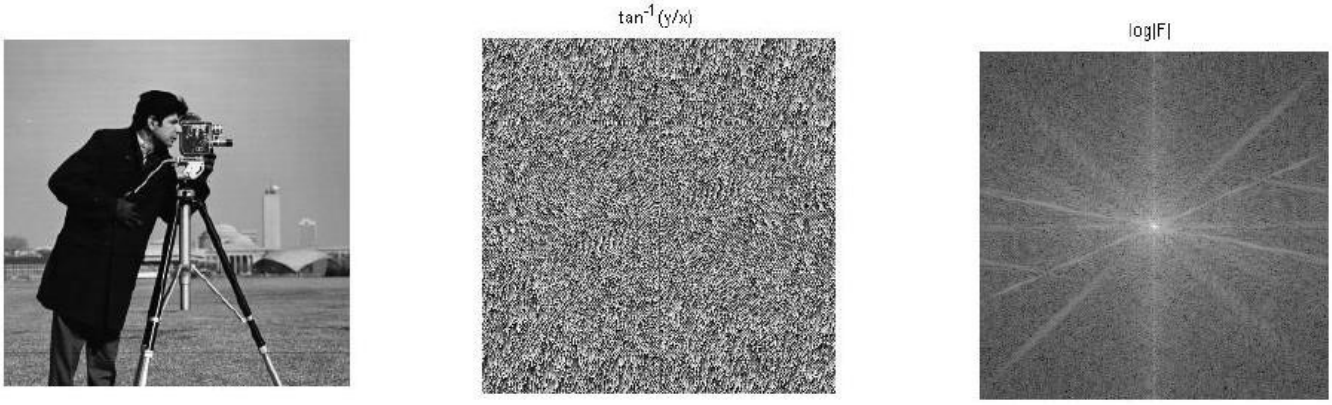


Figure 7-1 Frequency domain analysis

7.4 In-Lab Task

7.4.1 Frequency domain analysis

Write a MATLAB code to perform the following task using mathematical formula

1. Read a grayscale image provided by the lab instructor.
2. Plot magnitude response of the image.
3. Plot phase response of the image.
4. Write a brief analysis of the magnitude and phase response of the provided image.

7.4.2 Fourier Transform of Image

1. Write a MATLAB code to perform the following task using mathematical formula

    a. Read any grayscale image
    b. Apply Fourier transform.

c. Show the results
d. Apply inverse Fourier transform
e. Show the result

7.5 Post Lab Task

Write a MATLAB code to find inverse Fourier transform of an image.

Note: Image after inverse Fourier transform should be entirely identical with the input image.

Attach your results and code here.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

8 Construct frequency domain filters and manipulate images for advanced filtering using MATLAB

8.1 Objectives

- To perform image filtering by applying low-pass filter in frequency domain using MATLAB
- To perform image filtering by applying high-pass filter in frequency domain using MATLAB

8.2 Pre-Lab Theory

8.2.1 Lowpass and HighPass Frequency Domain Filters

Based on the property that multiplying the FFT of two functions from the spatial domain produces the convolution of those functions, you can use Fourier transforms as a fast convolution on large images. As a note, on small images, it is faster to work in the spatial domain. However, you can also create filters directly in the frequency domain. There are two commonly discussed filters in the frequency domain.

1. Lowpass filters, sometimes known as smoothing filters
2. Highpass filters, sometimes known as sharpening filters

8.2.2 Lowpass Frequency Domain Filters

Lowpass filters:

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

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

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

8.2.3 Highpass Frequency Domain Filters

High pass filters:

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

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

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

Because highpass filters can be created in relationship to low pass filters.

8.3 Pre-Lab Task

Read chapter no 3 of the book "Digital Image processing using MATLAB, R.C. Gonzales, R.E. Woods, Addison-Wesley" and try to understand the relationship between filters and images.

8.4 In-Lab Task

8.4.1 Gaussian Filter

Write a MATLAB program to implement the Gaussian filters on the image passed to it.

8.4.2 Butterworth Filter

Write a MATLAB code to implement the Butterworth low and high pass filter on an image provided by the lab instructor, also give your analysis about the image before and after applying the filters.

8.4.3 Ideal Filter

Write a MATLAB code to implement the Ideal low and high pass filter on an image provided by the lab instructor, also give your analysis about the image before and after applying the filters.

8.4.4 High Pass Filter

Write a MATLAB code to implement the Ideal, Butterworth and Gaussian high pass filter on an image provided by the lab instructor and find that which filter most sharpens the edges of the in the provided image, support your answer with proper justifications.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

9 Calibrate noise models and apply restoration techniques for noise reduction in digital images using MATLAB

9.1 Objectives

- To perform noisy images by adding different noises in sample images using MATLAB
- To perform restored images by applying spatial filters using MATLAB

9.2 Pre-Lab Theory

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

The degradation process is modelled as a degradation function that, together with an additive noise term, operates on an input image $f(x, y)$ to produce a degraded image

$$g(x, y) = H\,[f(x, y)] + \eta(x, y)$$

Given $g(x, y)$, some knowledge about the degradation function H, and some knowledge about the additive noise term $\eta(x, y)$, the objective of restoration is to obtain an estimate, $f(x, y)$, of the original image. We want the estimate to be as close as possible to the original input image. In general, the more we know about H and $\eta(x, y)$, the closer $f(x, y)$ will be to $f(x, y)$. If H is a linear, spatially invariant process, it can be shown that the degraded image is given in the spatial domain by

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y)$$

Where $g(x, y)$ is the spatial representation of the degradation function.

9.3 Pre-Lab Task

Write a MATLAB code, capable of restoring and reconstructing a color image provided by the lab instructor. You are allowed to use any technique, but you are required to provide the proper justification for your chosen technique.

9.4 In-Lab Task

9.4.1 Smoothing filter for images

Write a MATLAB code that removes different noise (Gaussian, salt & pepper) from the image. Follow the following steps

1. Read an image
2. Add Gaussian noise to it and save it to another variable
3. Add Salt & pepper noise and save it to another variable
4. Apply different spatial filter techniques to remove the 2 noises separately

To display digital image by adding noise and its restoration using MATLAB

5. Compare the input image and denoised images
6. In your lab report clearly mention which filter works for each noise

9.4.2 Image restoration: Noise removal of different types of added noises

Write a MATLAB code to perform the following tasks.

1. Read a RGB image provided by the instructor.
2. Insert Gaussian noise on 'Green'
3. Insert salt & paper noise on 'Red'
4. Concatenate the image
5. Apply filtering to remove the added noise.
6. Show and compare the noisy and restored image.
7. Briefly explain that which filters are used to remove the noise and why.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

10 Apply colour space conversions by manipulating and organizing digital images using MATLAB

10.1 Objectives

- To perform individual channels color image using MATLAB
- To perform a color image conversion between different color spaces using MATLAB

10.2 Pre-Lab Theory

An image in MATLAB is a matrix $m \times n$ containing colors to be displayed. The colors have to be defined in a color map, which is another matrix. A color map matrix may have any number of rows, but it must have exactly three columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue (that's why it's called an RGB image or matrix). Color intensity can be specified on the interval 0.0 to 1.0 in case of double. Similarly, the range of values is [O, 255] or [O, 65535]for RGB images of class `unit8` or `unit16`, respectively. The number of bits used to represent the pixel values of the component images determines the bit depth of an RGB image. For example, if each component image is an 8-bit image, the corresponding RGB image is said to be 24 bits deep.

10.3 Pre-Lab Task

Write a MATLAB code, capable of separating the RGB layers of any color image. You are also required to show the three layers separately.

10.3.1 Basic color mapping

```
colors1 = [
0 0 0   % First element = black
0 0 1   % blue
0 1 0   % green
0 1 1   % cyan
1 0 0   % red
1 0 1   % purple
1 1 0   % yellow
1 1 1]; % Last element = white
% We prepare the matrix that contains the colors to be
displayed % The list refers to the number of the elements in
the color list w = [1 2 3 4 5 6 7 8];
% We use the 'colormap' function to define the
actual
% palette in our workspace colormap(colors1) % We
use the 'image' instruction to display the matrix
image(w) axis off
```

To display digital image in different color spaces and conversion among them using MATLAB

10.3.2 Separating layers of RGB

```
I =
imread('Image_name');
r = I(:, :, 1); g =
I(:, :, 2); b = I(:,
:, 3); K = cat(3, r,
g, b); figure
subplot(121);
imshow(I);
subplot(122);
imshow(K); figure
subplot(311);
imshow(r);
subplot(312);
imshow(g);
subplot(313);
imshow(b);
```

10.3.3 Brightness changes in RGB Image

```
I =
imread('Image_name');
r = I(:, :, 1); g =
I(:, :, 2); b = I(:,
:, 3); r_new=r*2;
g_new=g*2; b_new=b*2
K = cat(3, r_new, g_new,
b_new); figure subplot(321);
imshow(r); subplot(322);
imshow(r_new); subplot(323);
imshow(g); subplot(324);
imshow(g_new); subplot(325);
imshow(b); subplot(326);
imshow(b_new); figure
subplot(121);
imshow(I)
subplot(1
2);
imshow(K)
;
```

10.3.4 Conversion between different color schemes: RGB to HSI

```
image=imread('Image_name');
imshow(image);
title('ORIGINAL IMAGE')
```

```
image=im2double(image);
r=image(:,:,1);
g=image(:,:,2);
b=image(:,:,3); num=0.5 *
((r-g)+(r-b)); den=sqrt((r-
g).^2 +(r-b).*(g-b));
theta=acos(num./(den+eps));
H=theta;
H(b>g)=2*pi -H(b>g);
H=H/(2*pi);
num=min(min(r,g),b);
den=r+g+b;
den(den==0)=eps;
S=1-3.* num./den;
H(S==0)=0;
I=(r+g+b)/3;
hsiimage=cat(3,H,S,
I); figure;
imshow(hsiimage);
title('HSI IMAGE')
```

10.4 In-Lab Task

10.4.1 Conversion between different color schemes: HIS to RGB and RGB to CMY

Write a MATLAB code to perform the following task without using the built-in commands

1. Convert the HSI image to RGB.
2. Convert the RGB image to CMY.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

11 Construct and apply morphological operations to extract image components using MATLAB

11.1 Objectives

To perform digital image component extraction by applying different morphological operations using MATLAB.

11.2 Pre-Lab Theory

In image processing, we use mathematical morphology to extract image components which are useful in representation and description of region shape such as Boundaries, Skeletons, Convex hull, Thinning, Pruning etc. There are two Fundamental morphological operations.

Dilation

Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1.

Erosion

Erosion removes pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to0.

Opening and closing operations can be done by combination of erosion and dilation in different sequence.

11.3 Pre-Lab Task

Read chapter no 10 of the book "Digital Image processing using MATLAB, R.C. Gonzales, R.E. Woods, Addison-Wesley" and try to understand the concept and usage of morphological operation on images.

11.3.1 Dilation

```
bw =
imread('text.png');
se=[0 1 0; 1 1 1; 0 1
0];
bw_out=imdilate(bw,se);
subplot(1,2,1);
imshow(bw);
subplot(1,2,2);
imshow(bw_out);
```

To display digital image by applying different morphological operations using MATLAB.

11.3.2 Erosion

```
bw =
imread('text.png');
se=ones(6,1);
bw_out=imerode(bw,se);
subplot(1,2,1);
imshow(bw);
subplot(1,2,2);
imshow(bw_out);
```

11.4 In-Lab Task

11.4.1 Morphological operation example

Write a MATLAB code, capable of removing the salt and pepper noise of an image provided by the lab instructor, using different morphological operations.

11.4.2 Closing and Opening using Erosion and Dilation

Write a MATLAB code to implement the closing and opening operation using erosion and dilation on an image provided by the lab instructor.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____

12 Manipulate digital images by applying and comparing edge detection techniques using MATLAB

12.1 Objectives

- To perform edge detection on digital image by applying different gradient operators using MATLAB.
- To perform edge detection on digital image by applying Laplacian operator using MATLAB.
- To perform edge detection on display digital image by applying different edge detectors using MATLAB.

12.2 Pre-Lab Theory

There are many operators which can be used for edge detection, some of them are as fallows.

1. Sobel operator
2. Roberts operator
3. Prewitt operator
4. Laplacian of Gaussian method
5. Zero-cross method

To use these operator, edge() function is used, syntax of `edge()` function is.

Syntax

```
BW = edge(I);
BW = edge(I,'sobel');
BW = edge(I,'prewitt');
BW = edge(I,'roberts');
```

`BW = edge(I)` takes a grayscale or a binary image I as its input, and returns a binary image BW of the same size as I, with 1's where the function finds edges in I and 0's elsewhere.

By default, edge uses the Sobel method to detect edges but the following provides a complete list of all the edge-finding methods supported by this function:

1. The Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
2. The Prewitt method finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
3. The Roberts method finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of I is maximum.
4. The Laplacian of Gaussian method finds edges by looking for zero crossings after filtering I with a Laplacian of Gaussian filter.
5. The zero-cross method finds edges by looking for zero crossings after filtering I with a filter you specify.

6. The Canny method finds edges by looking for local maxima of the gradient of I. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

The parameters you can supply differ depending on the method you specify. If you do not specify a method, edge uses the Sobel method.

12.3 In-Lab Tasks

12.3.1 Edge detection using Sobel, Roberts and Prewitts filters

Write a MATLAB code that detects edges in a n image using Sobel, Roberts and Prewitts filters without using built-in commands and compare the result of each filter with built-in commands Filtering using built-in commands is given below

```
i =
imread('Image_name'); I
= rgb2gray(i);
BW1 = edge(I,'prewitt');
BW2= edge(I,'sobel');
BW3=
edge(I,'roberts');
subplot (2,2,1);
imshow(I);
title('original');
subplot(2,2,2);
imshow(BW1);
title('Prewitt');
subplot(2,2,3);
imshow(BW2);
title('Sobel');
subplot(2,2,4);
imshow(BW3);
title('Roberts');
```

12.3.2 Edge detection using canny filters.

Write a MATLAB code to perform edge detection on an image provided by the lab instructor using canny filter.

12.3.3 Canny versus Sobel Filter

Write a MATLAB code to perform the edge detection on an image provided by the lab instructor, using canny and sobel method. You are also required to compare the result of edge detection using both methods and identify which method shows the best result for the provided image.

12.3.4 Vertical and Horizontal edge detection

Write a MATLAB code to perform the vertical and horizontal edge detection on an image provided by the lab instructor and then add the results of vertical and horizontal edge detection. You are also required to compare the result of vertical and horizontal edge detection with normal edge detection and give your analysis that which edge detection gives better result.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed all assigned tasks independently, strictly followed all health and safety protocols, and presented the results clearly and accurately.	4	
Good	The student completed the assigned tasks with minimal guidance, mostly followed health and safety protocols, and presented the results appropriately.	3	
Average	The student partially completed the assigned tasks, inconsistently followed health and safety protocols, and presented partial or unclear results.	2	
Worst	The student did not complete the assigned tasks, frequently disregarded health and safety protocols, and failed to present results.	1	

Instructor Signature: _____ **Date:** _____