CSE1902 Industrial Internship

A REPORT

submitted by

Vandana.M (19BCE1763)

in partial fulfilment for the award

of

B. Tech. Computer Science and Engineering

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

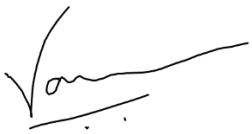
DECEMBER 2022

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled **"Value Added Course - MATLAB Digital Image Processing from Ground Up"** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai 600127 in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology – Computer Science and Engineering** is a record of bonafide work carried out by me**.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.



Signature
Vandana.M (19BCE1763)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled "**Value Added Course - MATLAB Digital Image Processing from Ground Up**" is prepared and submitted by **Vandana.M** **(Register No: 19BCE1763)**. It has been found satisfactory in terms of scope, quality, and presentation as partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology – Computer Science and Engineering** in Vellore Institute of Technology, Chennai, India.

Examined by:

Examiner I **Examiner II**

CERTIFICATE OF MERIT

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

Certificate of Participation

This is to certify that Dr./Prof./Mr./Ms. ……VANDANA MATHI……of ……Vellore Institute of Technology……has participated in the VAP titled **"MATLAB Digital Image Processing from Ground Up"** organised by School of Computer Science and Engineering (SCOPE) from 5th to 20th March 2022 at Vellore Institute of Technology, Chennai.

Dr. Geetha S	Dr. Jani Anbarasi L	Dr. Ganesan R
Coordinator	Coordinator	Dean - SCOPE

VIT – Recognised as Institution of Eminence (IoE) by Government of India
VIT - A place to learn; A chance to grow

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude to my project supervisors, Dr.Nithyanandam P, Head of the Department (HoD), B.Tech Computer Science and Engineering, SCOPE VIT Chennai ; Dr.Ganesan R, Dean of the School of Computer Science & Engineering, VIT Chennai ; Dr.Parvathi R, Associate Dean (Academics) of the School of Computer Science and Engineering, VIT Chennai; Dr.Geetha S, Associate Dean (Research) of the School of Computer Science & Engineering, VIT Chennai for giving me the opportunity to do this internship and providing invaluable guidance throughout this period. Their dynamism, vision, sincerity, and motivation has deeply inspired me. They have taught me the methodology to carry out the projects and to present the research works as clearly as possible. It was a great privilege and honour to work and study under their guidance. I am extremely grateful for what they offered me. I also thank all the staff at Vellore Institute of Technology, Chennai for their kindness and cooperation. Finally, my thanks go to all the people who have supported me to complete the internship directly or indirectly.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation	Expansion
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics

ABSTRACT

Digital Image Processing is rapidly growing in the domain of computer science and computer vision. With applications ranging from medical imaging to nosiness analytics, the need for critical thinking, innovation and optimal processing speed in this domain is increasingly crucial. MATLAB is a tool that helps us perform various operations on images and analyse various attributes simultaneously. The advanced libraries provide us a means to manipulate, enhance and modify images. By learning the skill of digital image processing, one can deal with large datasets and effectively clean, process and engineer data before building models.In this course, I aimed to learn the basics of Digital Image Processing using MATLAB and perform hands-on experiments along the way. In this course, I have employed the skills learnt in the course and programmed an image processing tool using MATLAB GUI that performs multiple image processing functions like conversion to black and white, image complement, edge detection, flip, rotation, etc which is included in the final chapter of the report. This tool can help a user to efficiently edit and process their images based on their need. Hence, I aimed to learning the fundamentals of image processing and hope to transfer the knowledge to other platforms in the future.

INTRODUCTION

Image processing is the process of converting a physical image to a digital representation and then conducting operations on it to extract valuable information. When implementing specific specified signal processing algorithms, the image processing system normally treats all images as 2D signals. There are five main types of image processing, and they are as follows.

1) Visualization - Look for objects in the image that aren't visible.
2) Recognition - Identifying or detecting things in an image is referred to as recognition.
3) Sharpening and restoration - From the original image, create an upgraded image.
4) Pattern recognition - Recognize the many patterns that surround the things in the image.
5) Retrieval - Search and browse through a big library of digital photos that are comparable to the original image.

Performing image processing has many applications and the possibilities are growing every day. In medical imaging, it can assist surgeons and other professionals to visualise and understand the patient's scenario with more clarity and better plan their procedures. In industrial applications, we use image processing to identify defective parts in manufacturing lines, self-driving cars, and satellite mapping for geographical data analytics. In security, we use image processing to detect faces, proctor students during examinations and many more. The possibilities of image processing are endless and only limited by our imagination. This is my motivation behind learning this course. The tremendous acceleration of computer vision in open-source projects has resulted from the emergence of deep learning technologies, which has only raised the demand for image processing tools. Every year, the need for experts with crucial skills in deep learning technologies rises at a rapid rate. For image processing, analysis, visualisation, and method development, Image Processing Toolbox in MATLAB includes a comprehensive range of reference-standard algorithms and workflow tools. Picture segmentation, enhancement, noise reduction, geometric transformations, image registration, and 3D image processing are all things we can do. We can automate common image processing workflows with Image Processing Toolbox apps. We can segment picture data interactively, evaluate image registration algorithms, and handle big data sets in batches. Explore photos, 3D volumes, and movies with visualisation features and apps that allow us to change contrast, construct histograms, and edit regions of interest (ROIs).

In this course, I have employed the skills learnt in this Value Added Course and programmed an Image Processing Tool using MATLAB GUI that can perform both basic and advanced image processing functions like black and white conversion, complement of an image, image flipping and rotation, edge detection and so on. The specifications of this tool will be discussed further in this report.

CHAPTER 1: Fundamentals of Image Processing

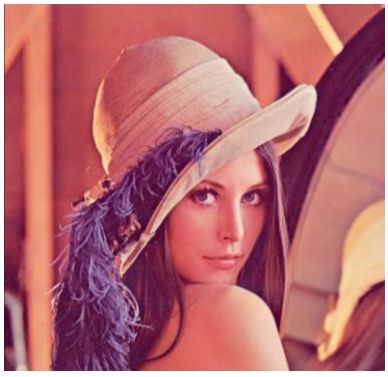
In this lab session, several operations were used. They are explored in the following sections.

Sections:

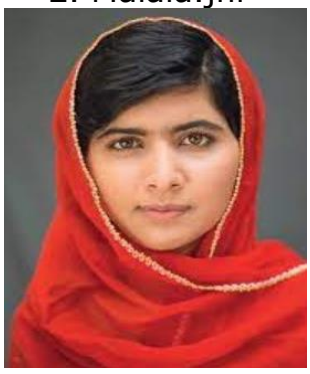
- Read Images
- Convery to Binary
- Convert to Grayscale
- Convert uint8 to uint16
- Convert uint16 to unit8
- Complement of an image
- Montage of images
- Write an image
- Addition of images

Input Images:

1. Lena.png



2. Malala.jfif



1. Reading an Image

```
%Reading an Image
I=imread('malala.jfif');
whos I
imshow(I);
```

```
Name        Size                Bytes  Class    Attributes

I         259x194x3            150738  uint8
```

2. Binary Image

```
% Binary
Bw= im2bw(I,0.4)
whos Bw
imshow(Bw);
```

```
Name        Size                Bytes  Class      Attributes

Bw        259x194                50246  logical
```

3. Greyscale Image

```
%Greyscale
Gray= rgb2gray(I)
imshow(Gray);
```

4. Conversion of uint8 to uint16

```
Six= im2uint16(I);
whos Six
```

```
  Name        Size                Bytes  Class     Attributes

  Six       259x194x3            301476  uint16
```

5. Conversion of uint16 to uint8

```
Eight= im2uint8(I);
whos Eight
```

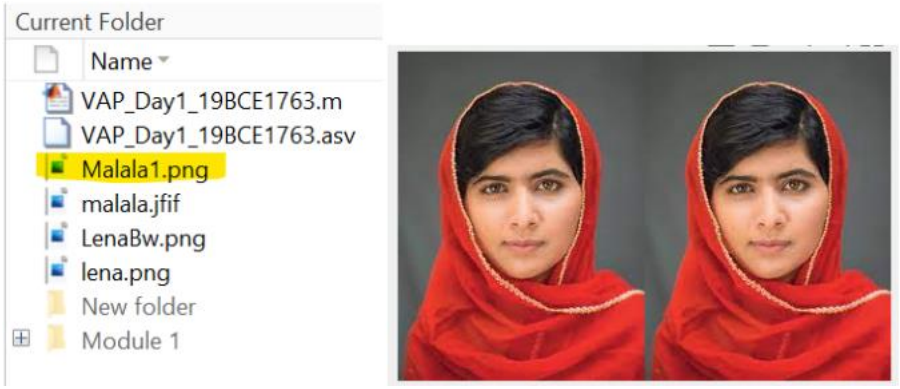
```
  Name        Size                Bytes  Class     Attributes

  Eight     259x194x3            150738  uint8
```

6. Complement of an Image

```
%Complement
comp= imcomplement(I)
imshow(comp);
imshowpair(I, comp, 'montage');
```

7. Writing an Image

```
%Writing an image
imwrite(I,'Malala1.png')
W= imread("Malala1.png")
imshowpair(I,W, 'montage')
```



8. Montage

```
% Read a color image and convert to binary and write the binary image
% and display with color image
L= imread("Lena.png")
lbw= im2bw(L);
imwrite(I,'LenaBw.png')
imshowpair(L,lbw,'montage')
```

9. Addition of Images

```
%1. Addition
P= imread('cameraman.tif')
Q= imread('rice.png')
add= imadd(P,Q)
imshow(add)
```

CHAPTER 2: Image Augmentation and Operation

In this lab session, several functions and operations were used which are explored in the following sections.

Sections:

- Subtraction of Images
- Resize of Images
- Absolute Difference of Images
- Multiplication of Images
- Division of Images
- Crop
- Rotate
- Translate

1. Subtraction of Images

```
%Subtract Image
I= imread('rice.png');
J= imread('cameraman.tif');
H= imsubtract(I,J);
imshow(H)
```

2. Image Resize

```
whos H
RR=imresize(H,[220 220]);
whos RR
```

```
Name        Size            Bytes  Class    Attributes

H         256x256           65536  uint8

Name        Size            Bytes  Class    Attributes

RR        220x220           48400  uint8
```

3. Absolute Difference of an Image

```
ad= imabsdiff(J,I);
imshowpair(ad, H, 'montage')
```

4. Multiplication of Images

```
%Multiplying with constant value
Malala= imread('malala.jfif')
P= immultiply(Malala,0.5);
imshowpair(Malala,P,'montage')
```

5. Division of Images

```
%Dividing with constant value
divim= imdivide(Malala,0.5);
imshowpair(Malala,divim,'montage')
```

6. Crop an Image

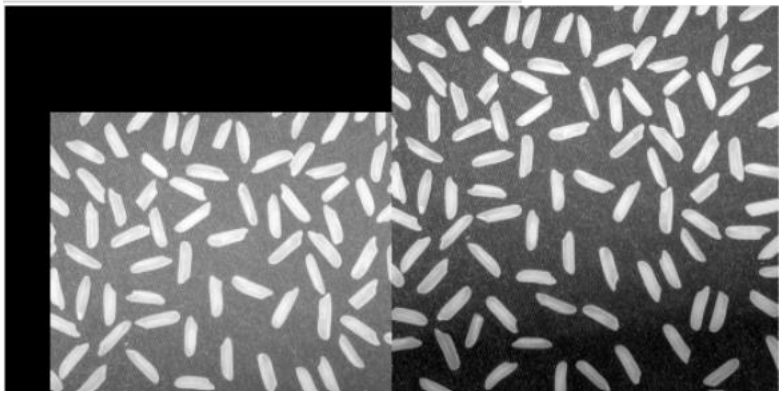
```
%Crop Image
I2=imcrop(P,[45 72 109 142]);
imshowpair(P, I2, 'montage')
```

7. Rotate Image

```
%Rotate
rot= imrotate(I3,-20)
imshowpair(rot, I3, 'montage')
```

8. Translate Image

```
%Translate
tran= imtranslate(I, [30,70])
imshowpair(tran, I, 'montage')
```



CHAPTER 3: Histogram Equalizer, Linear and Non-Linear Filtering

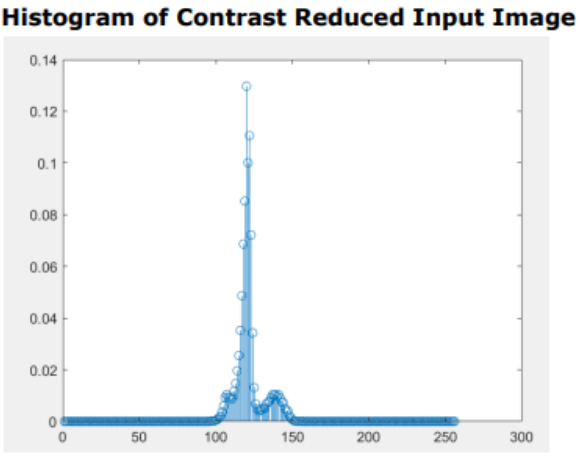
In this lab session, several functions and operations were used which are explored in the following sections.

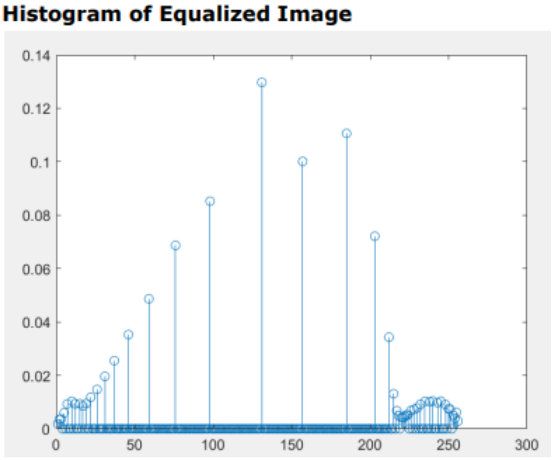
Sections:

- Histogram Equalizer
- Linear Filtering
- Non-linear Filtering

1. Histogram Equalizer

```
clc; clear all; close all;
a=imread('malala_lowc.jpg');
a=rgb2gray(a);
%a=a(1:50,1:50);
figure(1); imshow(uint8(a));
title('Input Image');
[m,n]=size(a);
h=zeros(1,256);
for i=1:m
   for j=1:n
     h(a(i,j)+1)=h(a(i,j)+1)+1;
   end
end
hn=h/(m*n);
figure(2); stem(hn);
c(1)=hn(1);
for i=2:256
   c(i)=c(i-1)+hn(i);
end
g=round(c*255);
he=zeros(m,n);
for i=1:m
   for j=1:n
       he(i,j)=g(a(i,j)+1);
   end
end
figure(3); imshow(uint8(he));
title('Histogram Equalized Image');
h=zeros(1,256);
for i=1:m
   for j=1:n
     h(he(i,j)+1)=h(he(i,j)+1)+1;
   end
end
hn=h/(m*n);
figure(4); stem(hn);
```





2. Linear Filtering

```
clc; clear all; close all;
x=imread('leo.jpg');
x=imnoise(x,'salt & pepper',0.05);
x=rgb2gray(x);
x=double(x);
[m,n]=size(x);
x=imresize(x,[50,50]);
figure(1);
subplot(2,2,1); imshow(uint8(x));
title('Input Image');
%%% Kernels
K1 = 1/9 *[1 1 1;1 1 1;1 1 1]
K1=double(K1);
K2 = 1/16 *[1 2 1;2 4 2;1 2 1]
K2=double(K2);
K3 = [-1 -1 -1;-1 8 -1;-1 -1 -1]
K3=double(K3);
[m,n]=size(x);
y1=zeros(size(x));
y2=zeros(size(x));
y3=zeros(size(x));
```

```
for i=2:m-1
for j=2:n-1
y1(i,j)=sum(sum((x(i-1:i+1,j-1:j+1).*K1)));
y2(i,j)=sum(sum((x(i-1:i+1,j-1:j+1).*K2)));
y3(i,j)=sum(sum((x(i-1:i+1,j-1:j+1).*K3)));
end
end
subplot(2,2,2); imshow(uint8(y1))
title('Mean Filtered Image');
subplot(2,2,3); imshow(uint8(y2))
title('Gaussian Filtered Image');
subplot(2,2,4); imshow(uint8(y3));
title('Laplacian Filtered Image');
```

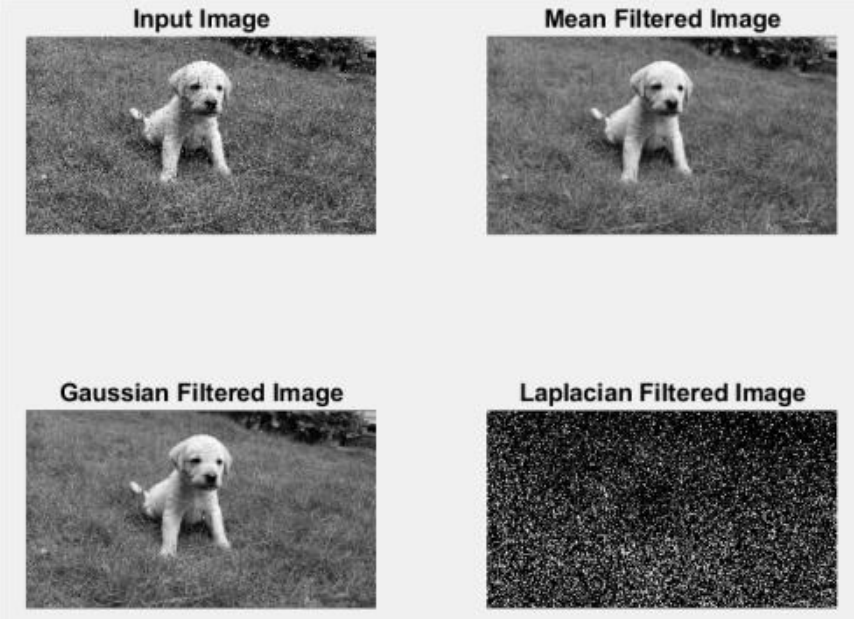
Input Image

```
K1 =

  Columns 1 through 2

    0.1111    0.1111
    0.1111    0.1111
    0.1111    0.1111

  Column 3

    0.1111
    0.1111
    0.1111
```

```
                        Column 3

                          0.0625
                          0.1250
                          0.0625
```

```
K2 =

  Columns 1 through 2

    0.0625    0.1250
    0.1250    0.2500
    0.0625    0.1250
```

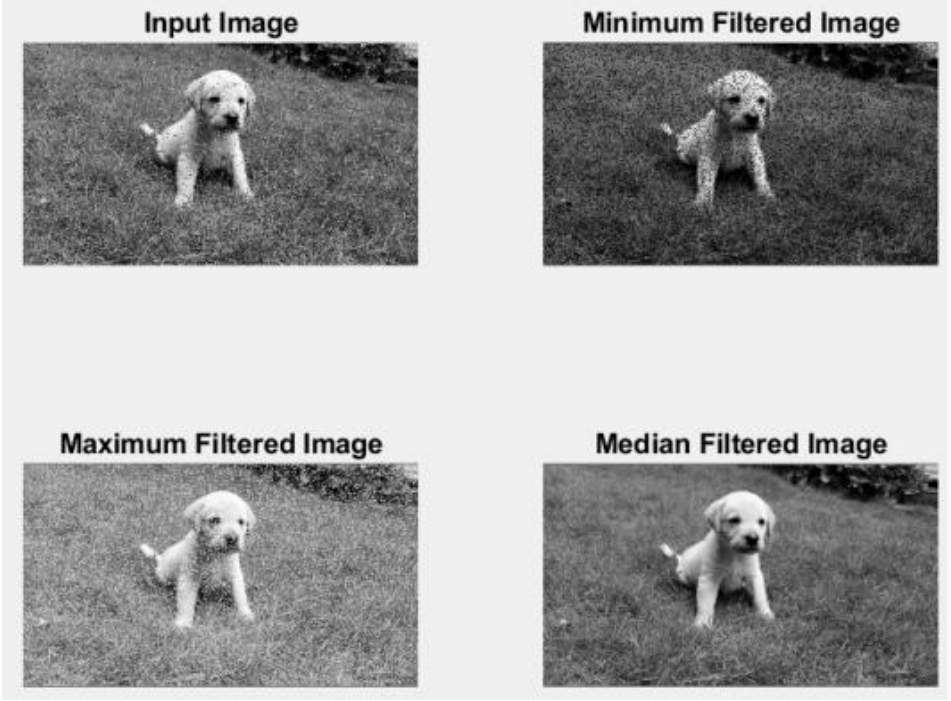
```
K3 =

    -1    -1    -1
    -1     8    -1
    -1    -1    -1
```



3. Non-Linear Filtering

```
clc; clear all; close all;
x=imread('leo.jpg');
x=imnoise(x,'salt & pepper',0.05);
x=rgb2gray(x);
x=double(x);
%x=imresize(x,[50,50]);
[m,n]=size(x);
figure(1);
subplot(2,2,1); imshow(uint8(x));
title('Input Image');
y1=zeros(size(x));
y2=zeros(size(x));
y3=zeros(size(x));
for i=2:m-1
for j=2:n-1
y1(i,j)=min(min(x(i-1:i+1,j-1:j+1)));
y2(i,j)=max(max(x(i-1:i+1,j-1:j+1)));
y3(i,j)=median(median(x(i-1:i+1,j-1:j+1)));
end
end
subplot(2,2,2); imshow(uint8(y1))
title('Minimum Filtered Image');
subplot(2,2,3); imshow(uint8(y2))
title('Maximum Filtered Image');
subplot(2,2,4); imshow(uint8(y3));
title('Median Filtered Image');
```

Input Image


Input Image
Minimum Filtered Image
Maximum Filtered Image
Median Filtered Image

CHAPTER 4: Morphological Operations and High Boost Filtering

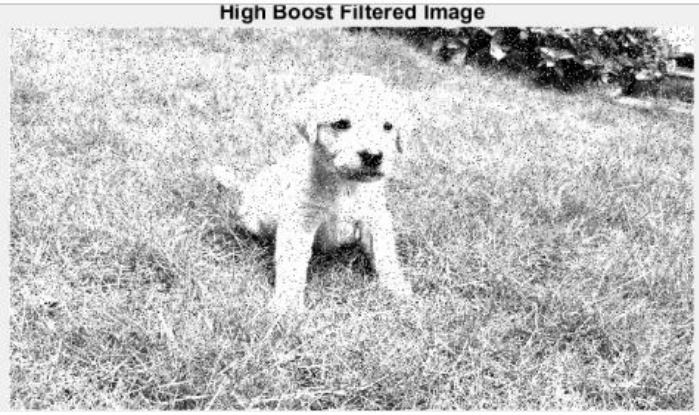
In this lab session, several functions and operations were used which are explored in the following sections.

Sections:

- High Boost Filtering
- Morphological Operations on Images

1. High Boost Filtering

```
clc; clear all; close all;
x=imread('leo.jpg');
x=imnoise(x,'salt & pepper',0.05);
x=rgb2gray(x);
x=double(x);
[m,n]=size(x);
%x=imresize(x,[50,50]);
figure(1);
subplot(2,2,1); imshow(uint8(x));
title('Input Image');
%%% Kernels
K1 = 1/9 *[1 1 1;1 1 1;1 1 1]
K1=double(K1);
[m,n]=size(x);
y1=zeros(size(x));
for i=2:m-1
for j=2:n-1
y1(i,j)=sum(sum((x(i-1:i+1,j-1:j+1).*K1)));
end
end
%High Boost Filtering
A=3;
HB= A*x-y1;
figure; imshow(uint8(HB))
title('High Boost Filtered Image');
```



2. Morphological Operations on Images

```
clc; clear all; close all;
I=imread('sil.jpg');
figure; imshow(uint8(I)); title('Input image')
B=im2bw(I,0.5);
figure; imshow(B); title('Binary image')
S=getnhood(strel('disk',5,0));
Bdil=imdilate(B,S);
figure; imshow(Bdil); title('Dilated image')
Bero=imerode(B,S);
figure; imshow(Bero); title('Eroded image')
figure; imshow(B-Bero); title('Boundary extracted image')
figure; imshow(Bdil-B); title('Boundary extracted image 2')
figure; imshow(Bdil-Bero); title('Boundary extracted image 3')
```

CHAPTER 5: Stenography and Planes of an Image

In this lab session, several functions and operations were used which are explored in the following sections.

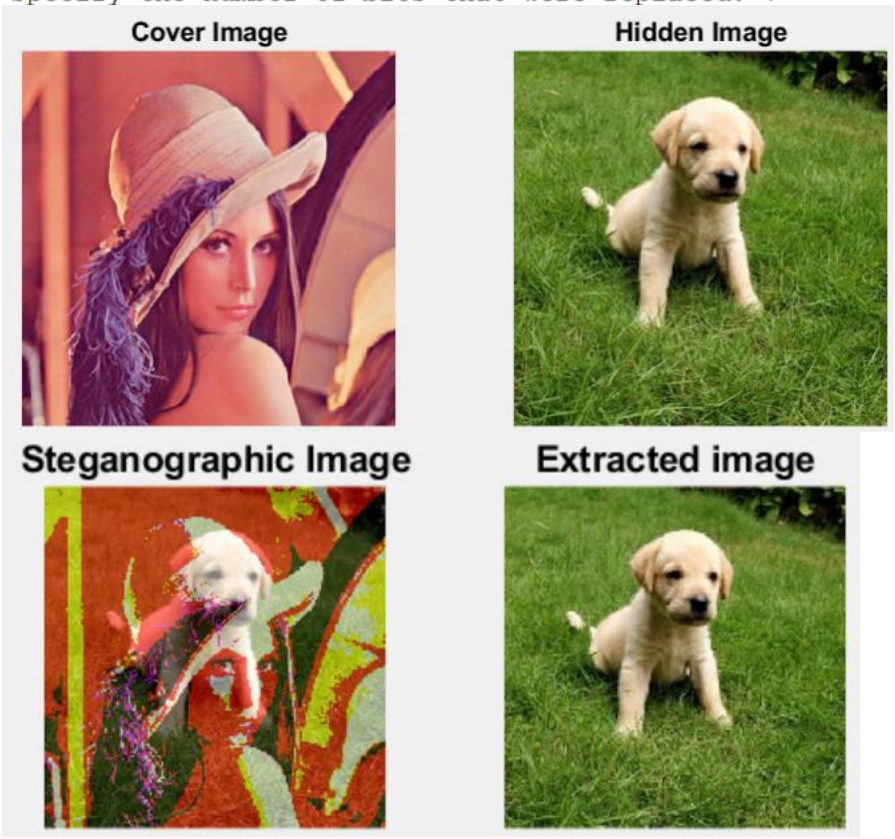
Sections:

- Stenography
- Visual Cryptography
- Bit Plane Slicing
- Bit Planes- Image Reconstruction

1. Stenography

```
Cover= input('Enter cover image name: ');
Message= input('Enter hidden image name: ');
Image= imread(Cover);
Hidden= imread(Message);
n= input('Specify number of bits to replace: ');
if n>=8||n<=0
      disp('Error: n must be between 1 and 7!');
else
      figure;
      subplot(1,2,1), imshow(Image)
      title('Cover Image')
      subplot(1,2,2), imshow(Hidden)
      title('Hidden Image')
      Steganog= uint8(bitor(bitand(Image, bitcmpOld(2^n -1,8)),
bitshift(Hidden, n-8)));
      disp("Extraction of hidden data")
      n= input('Specify the number of bits that were replaced: ');
      Extracted= uint8(bitand(255, bitshift(Steganog, 8-n)));
      figure
      subplot(1,3,1), imshow(Steganog)
      title('Steganographic Image')
      subplot(1,3,2), imshow(Extracted)
      title('Extracted image')
end
function [output] = bitcmpOld(x,N)
      if nargin<2
            output=bitcmp(x);
      else
            maxN=2^N-1;
            fmt ='uint8';
            out1= eval(['bitcmp(',fmt,'(x)',',''',fmt,''')']);
            out2= eval(['bitcmp(',fmt,'(maxN)',',''',fmt,''')']);
            output= out1-out2;
      end
end
```

```
>> VAP_Day5_19BCE1763_steganography
Enter cover image name: 'lenar.jpg'
Enter hidden image name: 'leor.jpg'
Specify number of bits to replace: 7
Extraction of hidden data
Specify the number of bits that were replaced: 7
```



2. Visual Cryptography

mainVis.m

```
inImg= imread('leo.jpg');
isBinaryImage= all(inImg(:)==0|inImg(:)==1)
if isBinaryImage==0
     inImg=im2bw(inImg)
end
figure; imshow(inImg); title('Secret Image');
[share1, share2, share12] = VisCrypt(inImg);
figure;imshow(share1); title('Share 1');
figure;imshow(share2); title('Share 2');
figure;imshow(share12); title('Overlapping Share 1 & 2');
imwrite(share1, 'Share1.bmp');
imwrite(share2, 'Share2.bmp');
imwrite(share12, 'Overlapped.bmp');
```

VisCrypt.m

```
function[share1, share2, share12]= VisCrypt(inImg)
s=size(inImg);
share1= zeros(s(1), (2*s(2)));
share2= zeros(s(1), (2*s(2)));
disp('White Pixel Processing...');
s1a=[1 0];
s1b=[1 0];
[x y]= find(inImg==1);
len= length(x);
for i=1:len
     a=x(i); b=y(i);
     pixShare= generateShare(s1a, s1b);
     share1((a), (2*b-1):(2*b))=pixShare(1,1:2);
     share2((a), (2*b-1):(2*b))=pixShare(2,1:2);
end
disp('Black Pixel Processing..');
s0a=[1 0];
s0b=[0 1];
[x y]= find(inImg==0);
len=length(x);
for i=1:len
     a=x(i); b=y(i);
     pixShare= generateShare(s0a, s0b);
     share1((a), (2*b-1):(2*b))=pixShare(1,1:2);
     share2((a), (2*b-1):(2*b))=pixShare(2,1:2);
end
share12= bitor(share1, share2);
share12= ~share12;
disp('Share Generation Completed.');
```

generateShare.m

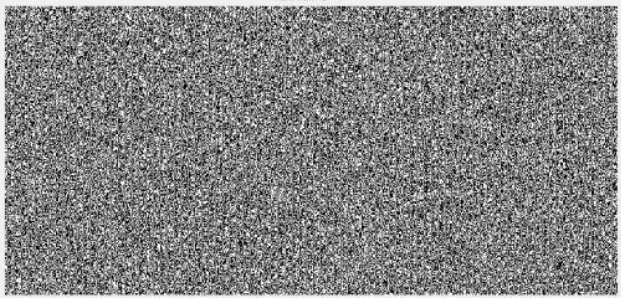
```
function out= generateShare(a,b)
a1=a(1);
a2=a(2);
b1=b(1);
b2=b(2);
in=[a
   b];
out= zeros(size(in));
randNumber= floor(1.9*rand(1));
if(randNumber==0)
     out=in;
elseif(randNumber==1)
     a(1)=a2;
     a(2)=a1;
     b(1)=b2;
     b(2)=b1;
     out=[a
          b];
end
```

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

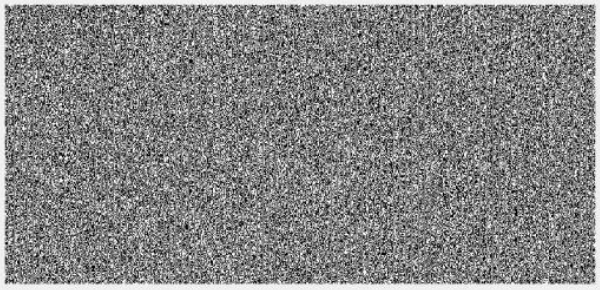
```
White Pixel Processing...
Black Pixel Processing..
Share Generation Completed.
```

Secret Image

Share 1



Share 2

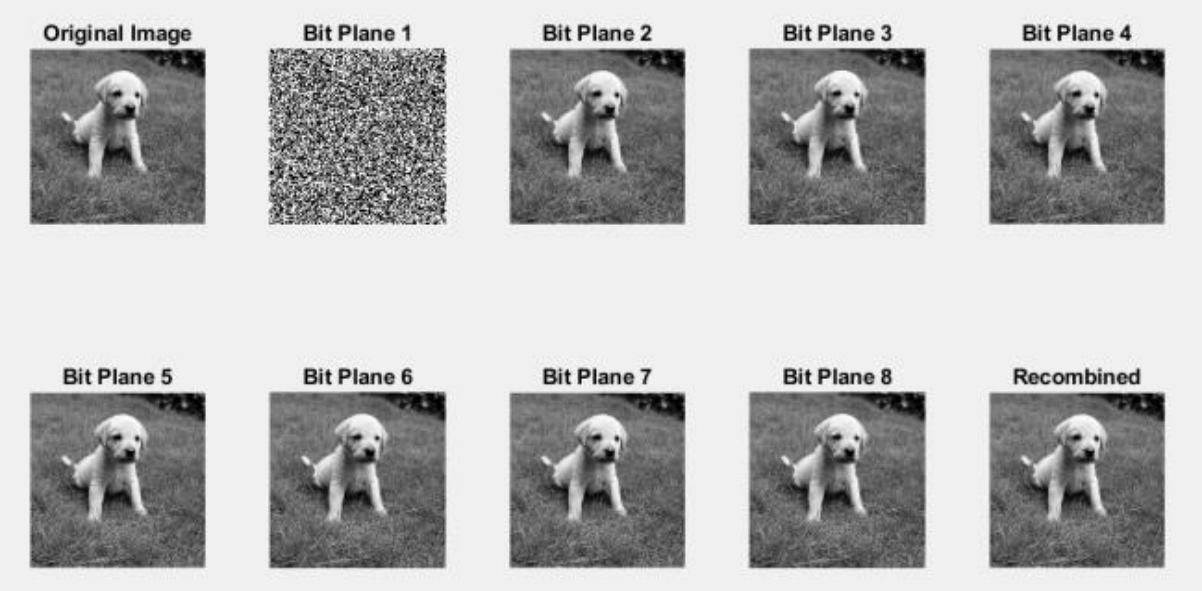


Overlapping Share 1 & 2

3. Bit-Plane Slicing

```
clc;
clear all;
close all;
c= rgb2gray(imread('leor.jpg'));
whos c
cd= double(c);
whos cd
c1= mod(cd,2);
c2= mod(floor(cd/2), 2);
c3= mod(floor(cd/4), 2);
c4= mod(floor(cd/8), 2);
c5= mod(floor(cd/16), 2);
c6= mod(floor(cd/32), 2);
c7= mod(floor(cd/64), 2);
c8= mod(floor(cd/128), 2);
cc=(2*(2*(2*(2*(2*(2*(2*c8+c7)+c6)+c5)+c4)+c3)+c2)+c1);
c5to8= (2*(2*(2*(2*(2*(2*(2*c8+c7)+c6)+c5)))));
c7to8= (2*(2*(2*(2*(2*(2*(2*c8+c7)))))));
subplot(2,5,1);
imshow(c);
title('Original Image');
subplot(2,5,2);
imshow(c1);
title('Bit Plane 1');
subplot(2,5,3);
imshow(c);
title('Bit Plane 2');
subplot(2,5,4);
imshow(c);
title('Bit Plane 3');
subplot(2,5,5);
imshow(c);
title('Bit Plane 4');
subplot(2,5,6);
imshow(c);
title('Bit Plane 5');
subplot(2,5,7);
imshow(c);
title('Bit Plane 6');
subplot(2,5,8);
imshow(c);
title('Bit Plane 7');
subplot(2,5,9);
imshow(c);
title('Bit Plane 8');
subplot(2,5,10);
imshow(c);
title('Recombined');
figure
```

```
subplot(1,3,1), imshow(uint8(cc)), title('Original Image');
subplot(1,3,2), imshow(uint8(c5to8)), title('Reconstructed image with
bit plane 5 to 8');
subplot(1,3,3), imshow(uint8(c7to8)), title('Reconstructed image with
bit plane 7 and 8');
```




```
Name        Size              Bytes  Class    Attributes

c         300x300             90000  uint8

Name        Size              Bytes  Class    Attributes

cd        300x300            720000  double
```

4. Bit Plane – Image Reconstruction

```
clc;
clear all;
close all;
A= rgb2gray(imread('leor.jpg'));
figure
imshow(A); title('Original Image')
B=bitget(A,1);
```

```
figure,
subplot(2,2,1);imshow(logical(B));title('Bit Plane 1');
B= bitget(A,2);
subplot(2,2,2);imshow(logical(B));title('Bit Plane 2');
B= bitget(A,3);
subplot(2,2,3);imshow(logical(B));title('Bit Plane 3');
B= bitget(A,4);
subplot(2,2,4);imshow(logical(B));title('Bit Plane 4');
B= bitget(A,5);
figure,
subplot(2,2,1);imshow(logical(B));title('Bit Plane 5');
B= bitget(A,6);
subplot(2,2,2);imshow(logical(B));title('Bit Plane 6');
B= bitget(A,7);
subplot(2,2,3);imshow(logical(B));title('Bit Plane 7');
B= bitget(A,8);
subplot(2,2,4);imshow(logical(B));title('Bit Plane 8');
B= zeros(size(A));
B=bitset(B,7,bitget(A,7));
B=bitset(B,8,bitget(A,8));
B=uint8(B);
figure, imshow(B); title('Image reconstruction by combining 8 bit
plane and 7 bit plane')
B= zeros(size(A));
B=bitset(B,7,bitget(A,7));
B=bitset(B,8,bitget(A,8));
B=bitset(B,6,bitget(A,6));
B=bitset(B,5,bitget(A,5));
B=uint8(B);
figure, imshow(B); title('Image reconstruction by combining 8,6,5,7
bit planes')
```

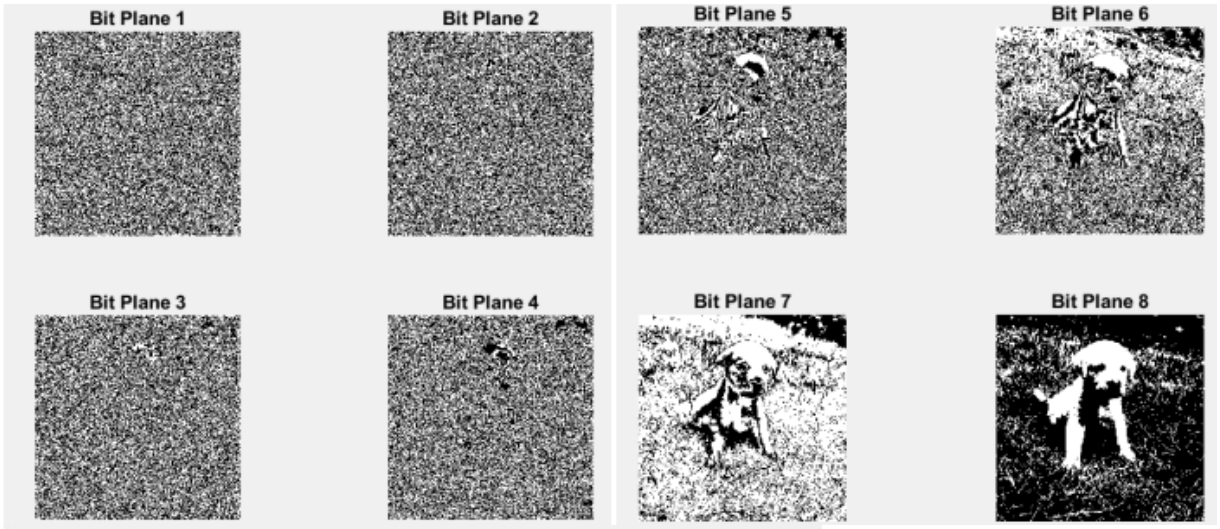


Image reconstruction by combining 8 bit plane and 7 bit plane

Image reconstruction by combining 8,6,5,7 bit planes

CHAPTER 6: Image Noise

In this lab session, several functions and operations were used which are explored in the following sections.

Sections:

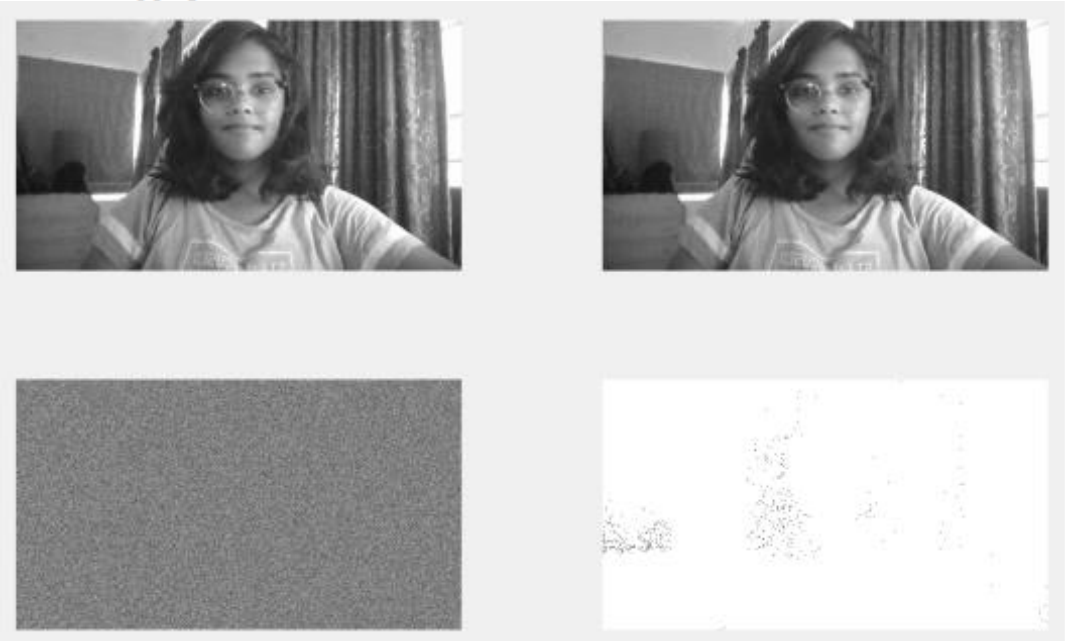
- Gaussian Noise
- Pepper Noise
- Salt Noise
- Speckle Noise
- Uniform Noise
- Rayleigh Noise
- Erlang Noise (Gamma)
- Exponential Noise

```
clear;
clc;
close all;
f = imread('image.png');
f=rgb2gray(f);
figure('Name','Fig 1 (a) Original Image');
imshow(f);
title('Fig 1 (a) Original Image');
[M, N] = size(f);
```

1. Gaussian Noise

```
% Investigate the different mean filters
% Gaussian Noise of 0 mean and variance of 400
gaussianNoise = imnoise2('gaussian', M, N, 0, 20);
applyGaussianNoise = double(f) + gaussianNoise; % Add noise
figure('Name','Fig 1 (b) Gaussian Noise of 0 mean and variance of
400');

imshow(applyGaussianNoise, []);
title('Fig 2 (b) Gaussian Noise of 0 mean and variance of 400');
subplot(2,2,1)
```

```
imshow(f);
subplot(2,2,2)
imshow(f);
subplot(2,2,3)
imshow(gaussianNoise,[]);
subplot(2,2,4)
imshow(applyGaussianNoise);
```

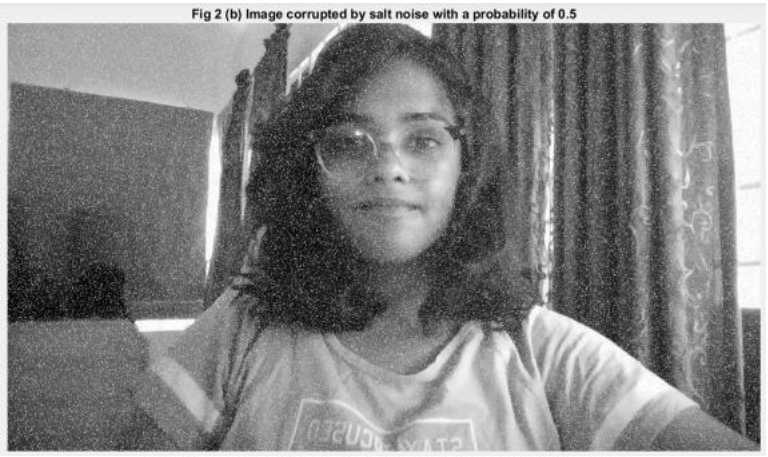


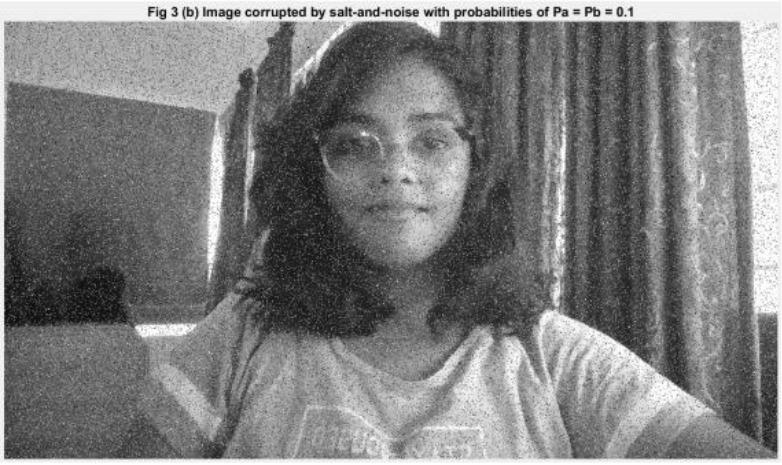
2. Pepper Noise

```
% Image corrupted by pepper noise with a probability of 0.1
pepperNoise = imnoise2('salt & pepper', M, N, 0, 0.1);
applyPepperNoise = f;
applyPepperNoise(pepperNoise == 1) = 0;
figure('Name', 'Fig 2 (a) Image corrupted by pepper noise with a
probability of 0.1');
imshow(applyPepperNoise, []);
title('Fig 2 (a) Image corrupted by pepper noise with a probability
of 0.1');
```

3. Salt Noise

```
% Image corrupted by salt noise with a probability of 0.1
saltNoise = imnoise2('salt & pepper', M, N, 0.1, 0);
applySaltNoise = f;
applySaltNoise(saltNoise == 1) = 225;
figure('Name', 'Fig 2 (b) Image corrupted by salt noise with a
probability of 0.5');
imshow(applySaltNoise, []);
title('Fig 2 (b) Image corrupted by salt noise with a probability of
0.5');
J = imnoise(f,'salt & pepper',0.1);
figure('Name', 'Fig 3 (a) Image corrupted by salt-and-noise with
probabilities of Pa = Pb = 0.1');
imshow(J, []);
title('Fig 3 (b) Image corrupted by salt-and-noise with probabilities
of Pa = Pb = 0.1');
```






```
% adds salt and pepper noise, where d is the noise density. This
affects approximately d*numel(I) pixels.
```

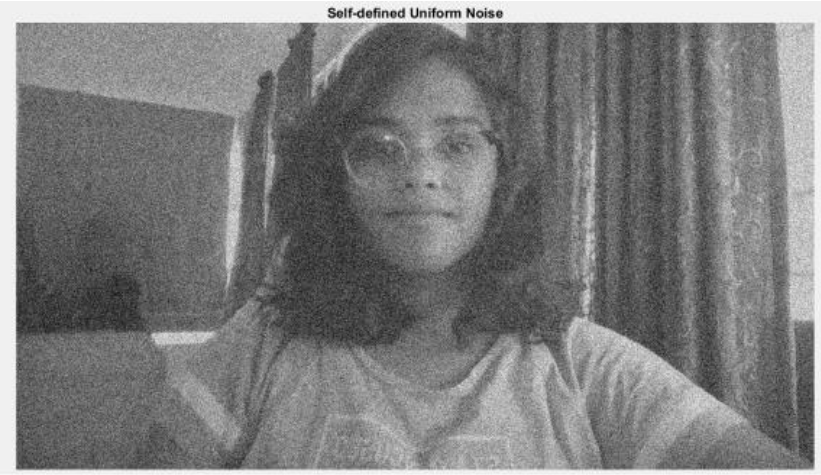
4. Speckle Noise

```
applySpeckle = imnoise(f,'speckle');
figure('Name', 'Fig 5.10 (d) Image corrupted by Speckle Noise J =
```

```
I+n*I ');
imshow(applySpeckle, []);
title('Fig 5.10 (d) Image corrupted by Speckle Noise J = I+n*I ');
```

5. Uniform Noise

```
% adds multiplicative noise using the equation J = I+n*I, where n is
uniformly distributed random noise with mean 0 and variance 0.05.
uniformNoise = imnoise2('uniform', M, N, 0, 200);
applyUniformNoise = double(f)+uniformNoise; % Add noise and convert
to uint 8
figure('Name','Self-defined Uniform Noise');
imshow(applyUniformNoise, []);
title('Self-defined Uniform Noise');
```



6. Rayleigh Noise

```
% Rayleigh Noise
rayleighNoise = imnoise2('rayleigh', M, N, 0, 1000);
applyRayleighNoise = double(f)+rayleighNoise; % Add noise and convert
to uint 8
figure('Name','Self-defined Rayleigh Noise');
imshow(applyRayleighNoise, []);
title('Self-defined Rayleigh Noise');
```

7. Erlang Noise (Gamma)

```
% Erlang (Gamma) Noise
erlangNoise = imnoise2('erlang', M, N, 0.05, 4);
applyErlangNoise = double(f)+erlangNoise; % Add noise and convert to
uint 8
figure('Name','Self-defined Erlang Noise');
imshow(applyErlangNoise, []);
title('Self-defined Erlang(Gamma) Noise');
```

8. Exponential Noise

```
% Exponential Noise
exponentialNoise = imnoise2('exponential', M, N, 0.05);
applyExponentialNoise = double(f)+exponentialNoise; % Add noise and
convert to uint 8
figure('Name','Self-defined Exponential Noise');
imshow(applyExponentialNoise, []);
title('Self-defined Exponential Noise');
```


Self-defined Exponential Noise

CHAPTER 7: Noise Reduction and Filtering

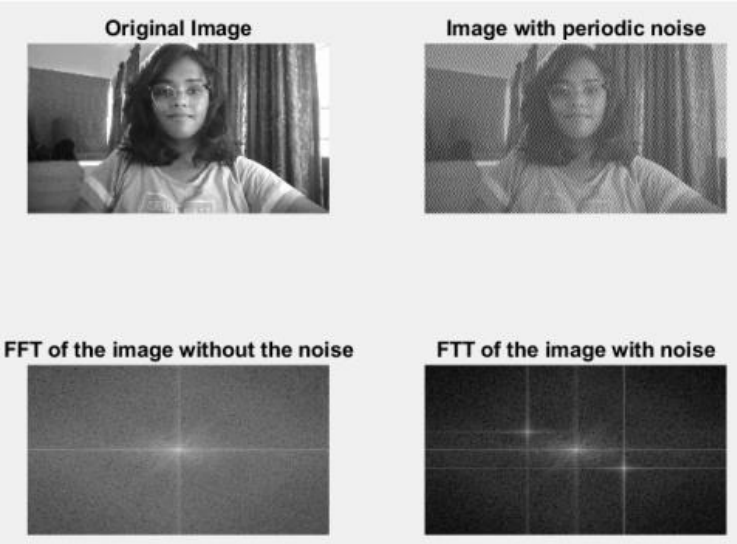
In this lab session, several operations were used which are explored in the following sections.

Sections:

- Image Restoration – Periodic Noise
- Min Max Filtering
- Mean Filtering
- Geometric Mean Filtering

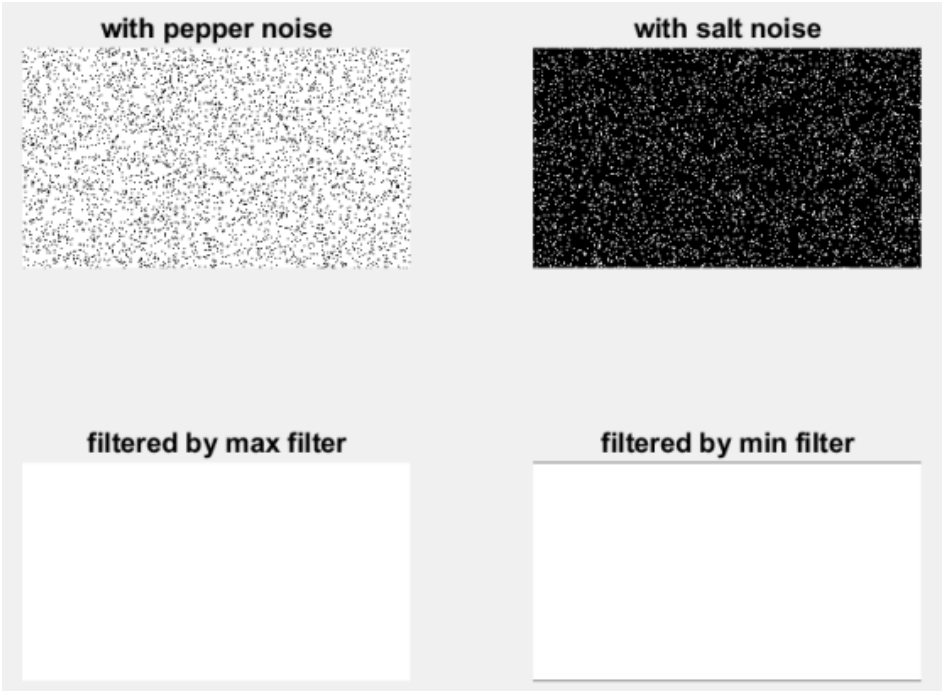
1. Periodic Noise- Image Restoration

```
%% Image restoration - Periodic noise
close all
clear all
clc
% Read image
I = imread('image.png');
I = rgb2gray(I);
[m,n]=size(I);
[x,y] = meshgrid(1:n,1:m);
p = 1+sin(x+y/1.5);
tp = (double(I)/128 + p)/ 4;
figure,
subplot(2,2,1); imshow(I,[]); title('Original Image');
subplot(2,2,2); imshow(tp,[]); title('Image with periodic noise');
If = fftshift(fft2(I));
Ifs = log(1+abs(If));
subplot(2,2,3); imshow(Ifs,[]); title('FFT of the image without the
noise');
tpf = fftshift(fft2(tp));
f = log(1+abs(tpf));
subplot(2,2,4); imshow(f,[]); title('FTT of the image with noise');
```



2. Min Max Filtering

```
clc;
clear all;
close all;
img = double(imread('image.png'));
img = rgb2gray(img);
%synthesize an image with pepper noise and an image with salt noise
noisePattern = rand(size(img));
%pepper noise probability is set as 0.1 in this example
imgPepperNoise = img .* (noisePattern > 0.1);
saltNoise = noisePattern < 0.1;
imgSaltNoise = img .* (1-saltNoise) + saltNoise*255;
figure;
subplot(2,2,1);imshow(imgPepperNoise,[ ]);
title('with pepper noise');
subplot(2,2,2);imshow(imgSaltNoise,[ ]);
title('with salt noise');
%perform max filtering
maxFilteredPepperImg = ordfilt2(imgPepperNoise, 9, ones(3,3));
subplot(2,2,3);imshow(maxFilteredPepperImg,[ ]);
title('filtered by max filter');
%perform min filtering
minFilteredSaltImg = ordfilt2(imgSaltNoise,1,ones(3,3));
subplot(2,2,4);imshow(minFilteredSaltImg,[ ]);
title('filtered by min filter');
```



3. Mean Filtering

```
%% Noise Reduction
clear; clc; close all;
f = im2grey(imread('image.png'));
figure('Name','Fig 1 (a) Original Image');
imshow(f);
title('Fig 1 (a) Original Image');
[M, N,K] = size(f);
% Investigate the different mean filters
% Gaussian Noise of 0 mean and variance of 400
gaussianNoise = imnoise2('gaussian', M, N, 0, 20);
applyGaussianNoise = double(f) + gaussianNoise; % Add noise
figure('Name','Fig 1 (b) Gaussian Noise of 0 mean and variance of
400');
imshow(applyGaussianNoise, []);
title('Fig 1 (b) Gaussian Noise of 0 mean and variance of 400');
% Arithmetic mean filter
ameanResult1 = spfilt(applyGaussianNoise, 'amean1', 3, 3);
figure('Name','Fig 1 (c1) Result of filtering with an arithmetic mean
filter of size 3 x 3');
imshow(ameanResult1, []);
title('Fig 1 (c1) Result of filtering with an arithmetic mean filter
of size 3 x 3');
% Self-defined arithmetic mean filter
ameanResult2 = spfilt(applyGaussianNoise, 'amean2', 3, 3);
figure('Name','Fig 1 (c2) Result of filtering with an self-defined
arithmetic mean filter of size 3 x 3');
imshow(ameanResult2, []);
title('Fig 1 (c2) Result of filtering with an self-defined arithmetic
mean filter of size 3 x 3');
```

Imnoise2.m

```
function R = imnoise2(type,M,N,a,b)
if nargin == 1
   a = 0; b = 1;
   M = 1; N = 1;
elseif nargin == 3
   a = 0; b = 1;
end
%as we need only small letters as the type so...
switch lower(type)
   case 'uniform'
       R = a + (b-a)*rand(M,N);
   case 'gaussian'
       R = a + b*randn(M,N);
   case 'salt & pepper'
       if nargin <= 3
       a = 0.05; b = 0.05;
       end
```

```
% check to make sure that Pa + Pb is not > 1.
        if (a + b) > 1
            error('The sum of the Pa and Pb cannot exeed 1.')
        end
          X = rand(M,N);
          c = find(X<=a);
          R(c) = 1;
          u = a + b;
          c = find(X > a & X <= u);
          R(c) = 1;
      case 'lognormal'
          if nargin<=3
              a = 1; b = 0.25;
          end
          R = a*esp(b*randn(M,N));
      case 'rayleigh'
        R = a + (-b*log(1-rand(M,N))).^0.5;
    case 'exponential'
        if nargin <= 3
            a = 1;
        end
        if a <= 0
            error('the value of a must b positive for exponential
operation')
        end
        k = -1/a
        R = k*log(1 - rand(M,N));
    case 'erlang'
        if nargin <= 3
            a = 2; b = 5;
        end
        if (b ~= round(b)| b <= 0)
            error('Parameter b should b a negative value for erlang')
        end
        k = -1/a;
        R = zeros(M,N);
        for j = 1:b
            R = R + k*log(1 - rand(M,N));
        end
    otherwise
        error('Unknown distribution type.')
end
```

4. Geometric Mean Filter

gmean.m

```
function f = gmean( g, m, n )
f = exp(imfilter(log(g), ones(m, n), 'replicate')).^(1/m/n);
end

% Result of filtering with a geometric mean filter of the same size
gmeanResult = spfilt(applyGaussianNoise, 'gmean', 3, 3);
figure('Name','Fig 1 (d) Result of filtering with a geometric mean
filter of size 3 x 3');
```

```
imshow(gmeanResult, []);
title('Fig 5.7 (d) Result of filtering with a geometric mean filter
of size 3 x 3');
```


```
% Result of filtering with a geometric mean filter of the same size
gmeanResult = spfilt(applyGaussianNoise, 'gmean', 3, 3);
figure('Name','Fig 1 (d) Result of filtering with a geometric mean
filter of size 3 x 3');
imshow(gmeanResult, []);
title('Fig 1 (d) Result of filtering with a geometric mean filter of
size 3 x 3');
```

CHAPTER 8: Noise Reduction

In this lab session, several functions and operations were used which are explored in the following sections.

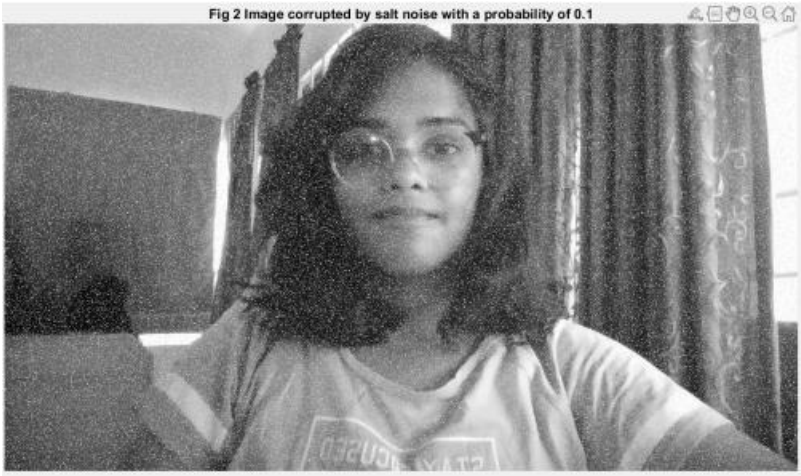
Sections:

- Contraharmonic Filter
- Adaptive Local Filter
- Adaptive Median Filter

1. Contraharmonic Filter

```
clear; clc; close all;
f = im2gray(imread('image.png'));
[M, N] = size(f);
pepperNoise = imnoise2('salt & pepper', M, N, 0, 0.1);
applyPepperNoise = f;
applyPepperNoise(pepperNoise == 1) = 0;
figure('Name', 'Fig 1 Image corrupted by pepper noise with a
probability of 0.1');
imshow(applyPepperNoise, []);
title('Fig 1 Image corrupted by pepper noise with a probability of
0.1');
% Image corrupted by salt noise with a probability of 0.1
saltNoise = imnoise2('salt & pepper', M, N, 0.1, 0);
applySaltNoise = f;
applySaltNoise(saltNoise == 1) = 225;
figure('Name', 'Fig 2 Image corrupted by salt noise with a
probability of 0.1');
imshow(applySaltNoise, []);
title('Fig 2 Image corrupted by salt noise with a probability of
0.1');
% Result of filtering (a) with a 3 x 3 contraharmonic filter of orc
1.5
chmeanPepperNoise = spfilt(applyPepperNoise, 'chmean', 3, 3, 1.5);
figure('Name','Result of filtering (a) with a 3 x 3 contraharmonic
filter of order 1.5');
imshow(chmeanPepperNoise, []);
title('Result of filtering (a) with a 3 x 3 contraharmonic filter c
order 1.5');
% Result of filtering (b) with Q = -1.5
chmeanSaltNoise = spfilt(applySaltNoise, 'chmean', 3, 3, -1.5);
figure('Name','Result of filtering (b) with Q = -1.5');
imshow(chmeanSaltNoise, []);
title('Result of filtering (b) with Q = -1.5');
% Result of filtering Fig. 5.8(a) with a contraharmonic filter of
size 3 x
% 3 and Q = -1.5
```

```
chmeanPepperNoise = spfilt(applyPepperNoise, 'chmean', 3, 3, -1.5);
figure('Name','Fig 5.9 (a) Result of filtering Fig. 5.8(a) with a
contraharmonic filter of size 3 x % 3 and Q = -1.5');
imshow(chmeanPepperNoise, []);
title('Fig 5.9 (a) Result of filtering Fig. 5.8(a) with a
contraharmonic filter of size 3 x % 3 and Q = -1.5');
% Result of filtering Fig. 5.8(b) with a contraharmonic filter of
size 3 x
% 3 and Q = 1.5
chmeanPepperNoise = spfilt(applySaltNoise, 'chmean', 3, 3, 1.5);
figure('Name','Fig 5.9 (b) Result of filtering Fig. 5.8(b) with a
contraharmonic filter of size 3 x % 3 and Q = 1.5');
imshow(chmeanPepperNoise, []);
title('Fig 5.9 (b) Result of filtering Fig. 5.8(b) with a
contraharmonic filter of size 3 x % 3 and Q = 1.5');
```

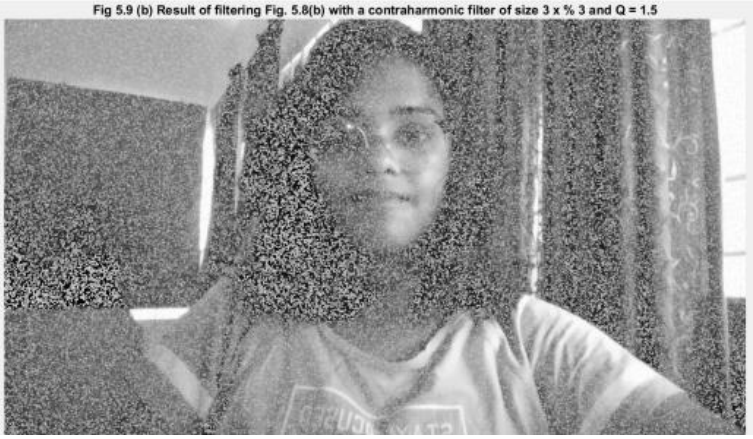


Result of filtering (a) with a 3 x 3 contraharmonic filter of order 1.5

Result of filtering (b) with Q = -1.5

Fig 5.9 (a) Result of filtering Fig. 5.8(a) with a contraharmonic filter of size 3 x % 3 and Q = -1.5

Fig 5.9 (b) Result of filtering Fig. 5.8(b) with a contraharmonic filter of size 3 x % 3 and Q = 1.5



2. Adaptive Local Filter

Imnoise2.m

```
function R = imnoise2(type,M,N,a,b)
if nargin == 1
   a = 0; b = 1;
   M = 1; N = 1;
elseif nargin == 3
   a = 0; b = 1;
end
%as we need only small letters as the type so...
switch lower(type)
   case 'uniform'
       R = a + (b-a)*rand(M,N);
   case 'gaussian'
       R = a + b*randn(M,N);
   case 'salt & pepper'
       if nargin <= 3
       a = 0.05; b = 0.05;
       end
% check to make sure that Pa + Pb is not > 1.
       if (a + b) > 1
           error('The sum of the Pa and Pb cannot exeed 1.')
       end
         X = rand(M,N);
         c = find(X<=a);
         R(c) = 1;
         u = a + b;
         c = find(X > a & X <= u);
         R(c) = 1;
     case 'lognormal'
         if nargin<=3
             a = 1; b = 0.25;
         end
         R = a*esp(b*randn(M,N));
     case 'rayleigh'
       R = a + (-b*log(1-rand(M,N))).^0.5;
   case 'exponential'
       if nargin <= 3
           a = 1;
       end
       if a <= 0
           error('the value of a must b positive for exponential
operation')
       end
       k = -1/a
       R = k*log(1 - rand(M,N));
   case 'erlang'
       if nargin <= 3
           a = 2; b = 5;
       end
       if (b ~= round(b)| b <= 0)
           error('Parameter b should b a negative value for erlang')
       end
       k = -1/a;
       R = zeros(M,N);
       for j = 1:b
           R = R + k*log(1 - rand(M,N));
       end
   otherwise
       error('Unknown distribution type.')
end
```

charmean.m

```
function f = charmean( g, m, n, q )
%CHARMEAN Implements a contraharmonic mean filter
% [g, revertClass] = float(g);
g = double(g);
f = imfilter(g .^ (q+1), ones(m, n), 'replicate');
f = f ./ (imfilter(g .^ q, ones(m, n), 'replicate') + eps);
% f = revertClass(f);
end
```

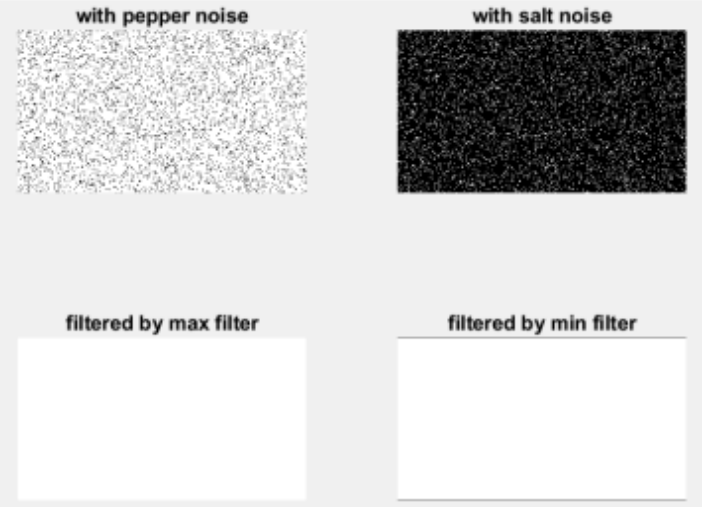
adpmedian.m

```
% Adaptive median filter
function f = adpmedian (g, Smax)
% ADPMEDIAN Parform adaptive median filtering.
% F = ADPMEDIAN(G, SMAX) performs adaptive median filtering of
% image G. The median filter starts at size 3-by-3 and iterates up
% to size SMAX-by-SMAX. SMAX must be an odd integer greater than 1.
% SMAX must be an odd, positive integer greater than 1.
if (Smax <= 1) || (Smax/2 == round(Smax/2)) || (Smax ~= round(Smax))
error ('SMAX must be an odd integer> 1.')
end
% [M, N] = size(g);
% Initial setup.
f = g;
f(:) = 0;
alreadyProcessed = false (size(g));
% Begin filtering.
for k = 3:2:Smax
zmin = ordfilt2(g, 1, ones(k, k),'symmetric');
zmax = ordfilt2(g, k * k, ones(k, k),'symmetric');
zmed = medfilt2(g, [k,k],'symmetric');
processUsingLevelB = (zmed> zmin) & (zmax> zmed) &...
~alreadyProcessed;
zB = (g> zmin) & (zmax> g);
outputZxy = processUsingLevelB & zB;
outputZmed = processUsingLevelB & ~zB;
f (outputZxy) = g(outputZxy);
f (outputZmed) = zmed(outputZmed);
alreadyProcessed = alreadyProcessed | processUsingLevelB;
if all (alreadyProcessed (:))
break;
end
end
% Output zmed for any remaining unprocessed pixels. Note that this
% zmed was computed using a window of size Smax-by-Smax, which is
% the final value of k in the loop.
f (~alreadyProcessed) = zmed (~alreadyProcessed);
end
```

spfilt.m

```
function f = spfilt( g, type, varargin )
switch type
case 'amean1'
w = fspecial('average', [m n]);
f = imfilter(g, w, 'replicate');
case 'amean2'
f = selfDefinedAveFilt(g, m, n);
case 'gmean'
f = gmean(g, m, n);
case 'hmean'
f = harmean(g, m ,n);
case 'chmean'
f = charmean(g, m, n, Q);
case 'median1'
f = medfilt2(g, [m n], 'symmetric');
case 'median2'
f = selfDefinedMedian(g, m, n);
case 'max'
f = imdilate(g, ones(m, n));
case 'min'
f = imerode(g, ones(m, n));
case 'midpoint'
f1 = ordfilt2(g, 1, ones(m, n), 'symmetric');
f2 = ordfilt2(g, m*n, ones(m, n), 'symmetric');
f = imlincomb(0.5, f1, 0.5, f2);
case 'atrimmed'
f = alphatrim(g, m, n, d);
otherwise
error('Unknown filter type')
end
end
```

```
clc;
clear all;
img = im2gray(double(imread('image.png')));
%synthesize an image with pepper noise and an image with salt noise
noisePattern = rand(size(img));
%pepper noise probability is set as 0.1 in this example
imgPepperNoise = img .* (noisePattern > 0.1);
saltNoise = noisePattern < 0.1;
imgSaltNoise = img .* (1-saltNoise) + saltNoise*255;
figure;
subplot(2,2,1);imshow(imgPepperNoise,[ ]);
title('with pepper noise');
subplot(2,2,2);imshow(imgSaltNoise,[ ]);
title('with salt noise');
%perform max filtering
maxFilteredPepperImg = ordfilt2(imgPepperNoise, 9, ones(3,3));
subplot(2,2,3);imshow(maxFilteredPepperImg,[ ]);
title('filtered by max filter');
%perform min filtering
minFilteredSaltImg = ordfilt2(imgSaltNoise,1,ones(3,3));
subplot(2,2,4);imshow(minFilteredSaltImg,[ ]);
title('filtered by min filter');
```




```
clc;
clear all;
close all;
B = imread('image.png');
sz = size(B,1)*size(B,2);
%Add gaussian noise with mean 0 and variance 0.005
B = imnoise(B,'gaussian',0,0.005);
figure,imshow(B); title('Image with gaussian noise');
B = double(B);
%Define the window size mxn
M = 5;
N = 5;
%Pad the matrix with zeros on all sides
C = padarray(B,[floor(M/2),floor(N/2)]);
lvar = zeros([size(B,1) size(B,2)]);
lmean = zeros([size(B,1) size(B,2)]);
temp = zeros([size(B,1) size(B,2)]);
NewImg = zeros([size(B,1) size(B,2)]);
for i = 1:size(C,1)-(M-1)
for j = 1:size(C,2)-(N-1)
temp = C(i:i+(M-1),j:j+(N-1));
tmp = temp(:);
%Find the local mean and local variance for the local region
lmean(i,j) = mean(tmp);
lvar(i,j) = mean(tmp.^2)-mean(tmp).^2;
end
end
%Noise variance and average of the local variance
nvar = sum(lvar(:))/sz;
%If noise_variance > local_variance then
local_variance=noise_variance
lvar = max(lvar,nvar);
%Final_Image = B- (noise variance/local variance)*(B-local_mean);
```

```
NewImg = nvar./lvar;
NewImg = NewImg.*(B-lmean);
NewImg = B-NewImg;
%Convert the image to uint8 format.
NewImg = uint8(NewImg);
figure,imshow(NewImg);title('Restored Image using Adaptive Local
filter');
```

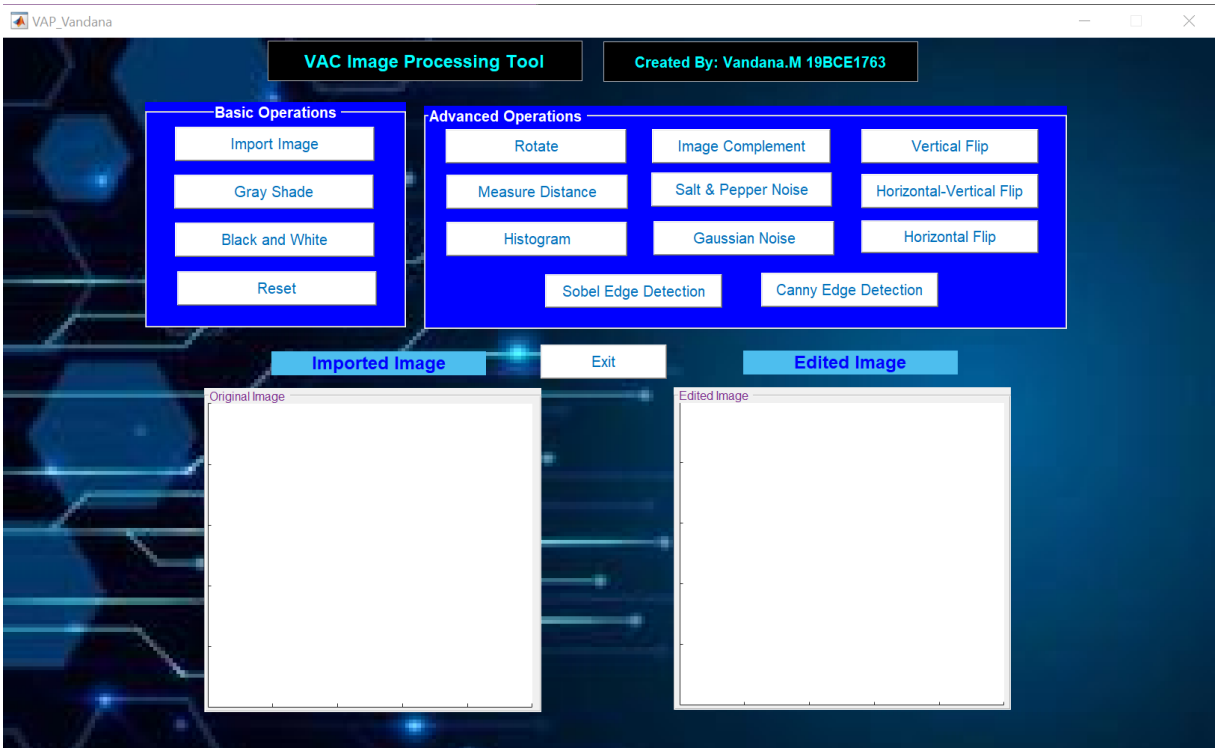
3. Adaptive Median Filter

```
clc;
clear all;
close all;
Gray = im2gray(imread('image.png'));% read image file
%Add various concentrations of salt and pepper noise
N1 = imnoise(Gray,'salt & pepper',0.05);% salt and pepper noise,
noise density 0.05
N2 = imnoise(Gray,'salt & pepper',0.25);% salt and pepper noise,
noise density 0.25
N3 = imnoise(Gray,'salt & pepper',0.45);% salt and pepper noise,
noise density 0.45
N4 = imnoise(Gray,'salt & pepper',0.65);% salt and pepper noise,
noise density 0.65
% Median filter
M1 = medfilt2(N1);
M2 = medfilt2(N2);
M3 = medfilt2(N3);
M4 = medfilt2(N4);
```

```
% Adaptive median filter
f1 = adpmedian(N1,11);
f2 = adpmedian(N2,11);
f3 = adpmedian(N3,11);
f4 = adpmedian(N4,11);
figure
subplot(341);
imshow(N1);
subplot(342);
imshow(N2);
subplot(343);
imshow(N3);
subplot(344);
imshow(N4);
subplot(345);
imshow(M1);
subplot(346);
imshow(M2);
subplot(347);
imshow(M3);
subplot(348);
imshow(M4);
subplot(349);
imshow(f1);
subplot(3,4,10);
imshow(f2);
subplot(3,4,11);
imshow(f3);
subplot(3,4,12);
imshow(f4);
```

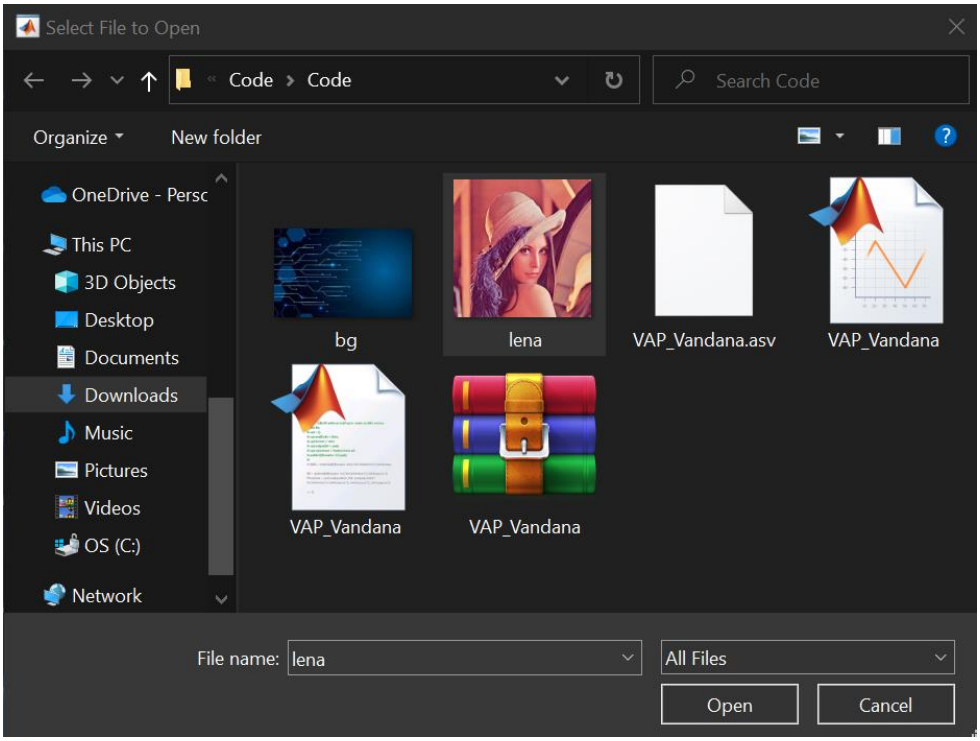
CHAPTER 9: MATLAB GUI & Project

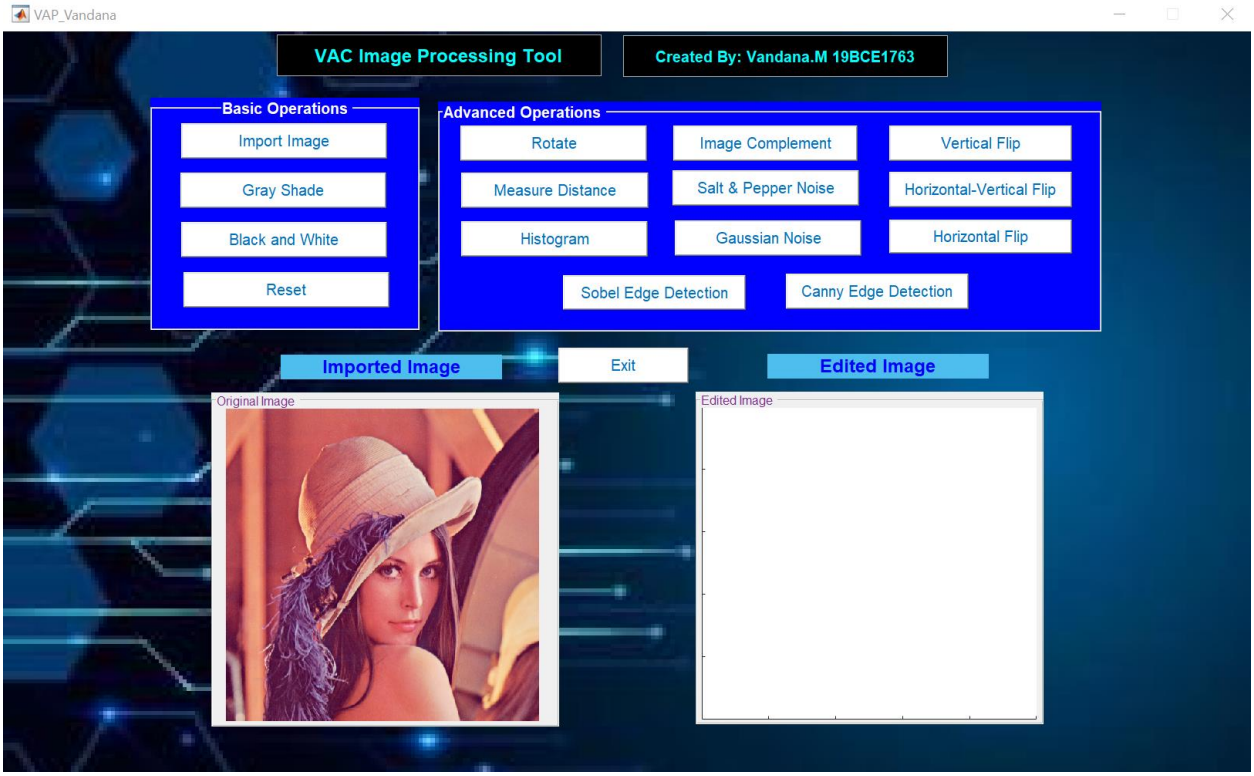
I aimed to create a GUI application using GUIDE framework in MATLAB that can perform various basic functions like black and white, greyscale, complement, rotate, flip and instil different noises in the image and perform edge detection based on user's choices. The objective of this project is to provide an application where learners can understand and experiment with various basic image processing functions, noise and edge detection. The application GUI is shown as follows.



The control panel on the top left named "Basic Operations" contains four buttons namely – "Import Image", "Gray Shade", "Black and White" and "Reset". The panel on the top right named "Advanced Operations" contains buttons that perform the functions "Rotate", "Measure Distance", "Histogram", "Sobel Edge Detection", "Image Complement", "Salt & Pepper Noise", "Gaussian Noise", "Vertical Flip", "Horizontal-Vertical Flip", "Horizontal Flip" and "Canny Edge Detection". The Imported image will be displayed on the axis (white plane) on the bottom left. After any image processing function is applied to the imported image, the edited image is displayed on the axis on the bottom right. The entire application is on a GUIDE framework.

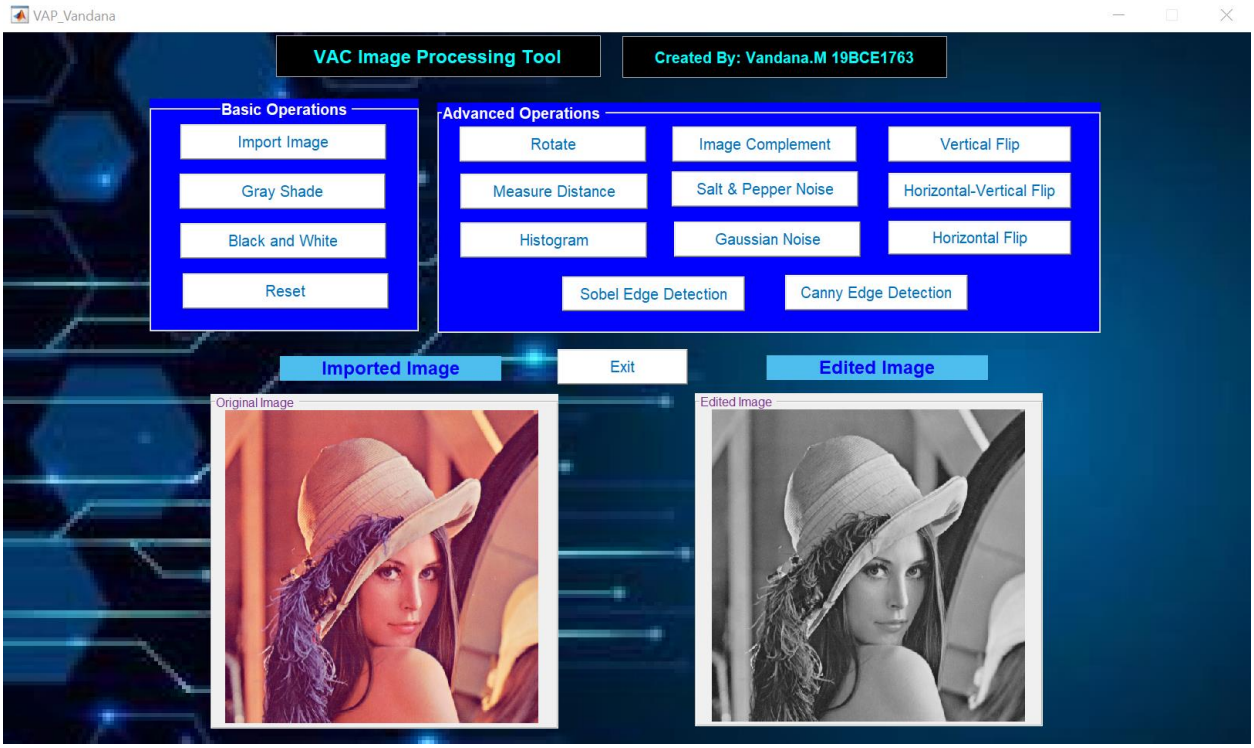
The upload button allows the user to search and upload any image. This is shown below.


Select File to Open
Code › Code
Search Code
Organize
New folder
OneDrive - Persc
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
OS (C:)
Network
bg
lena
VAP_Vandana.asv
VAP_Vandana
VAP_Vandana
VAP_Vandana
File name:
lena
All Files
Open
Cancel

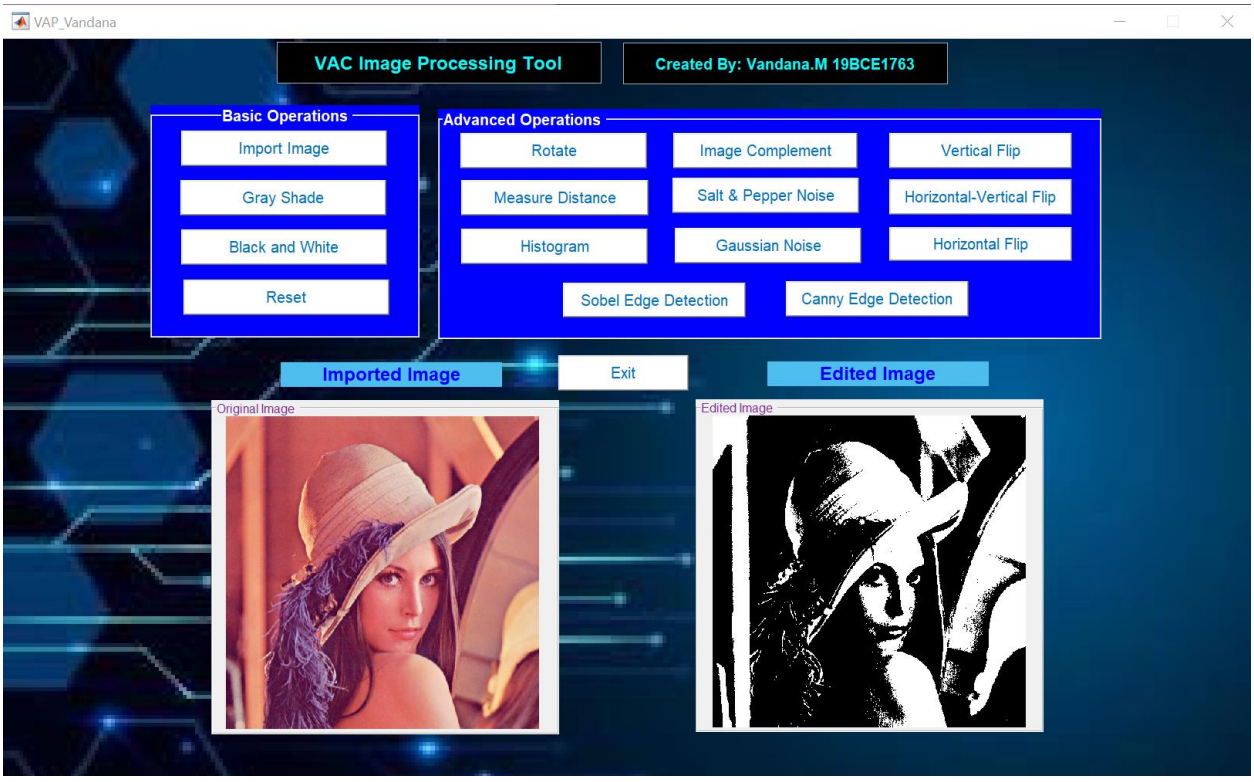

VAP_Vandana
VAC Image Processing Tool
Created By: Vandana.M 19BCE1763
Basic Operations
Import Image
Gray Shade
Black and White
Reset
Advanced Operations
Rotate
Image Complement
Vertical Flip
Measure Distance
Salt & Pepper Noise
Horizontal-Vertical Flip
Histogram
Gaussian Noise
Horizontal Flip
Sobel Edge Detection
Canny Edge Detection
Imported Image
Exit
Edited Image
Original Image
Edited Image

Basic Operations

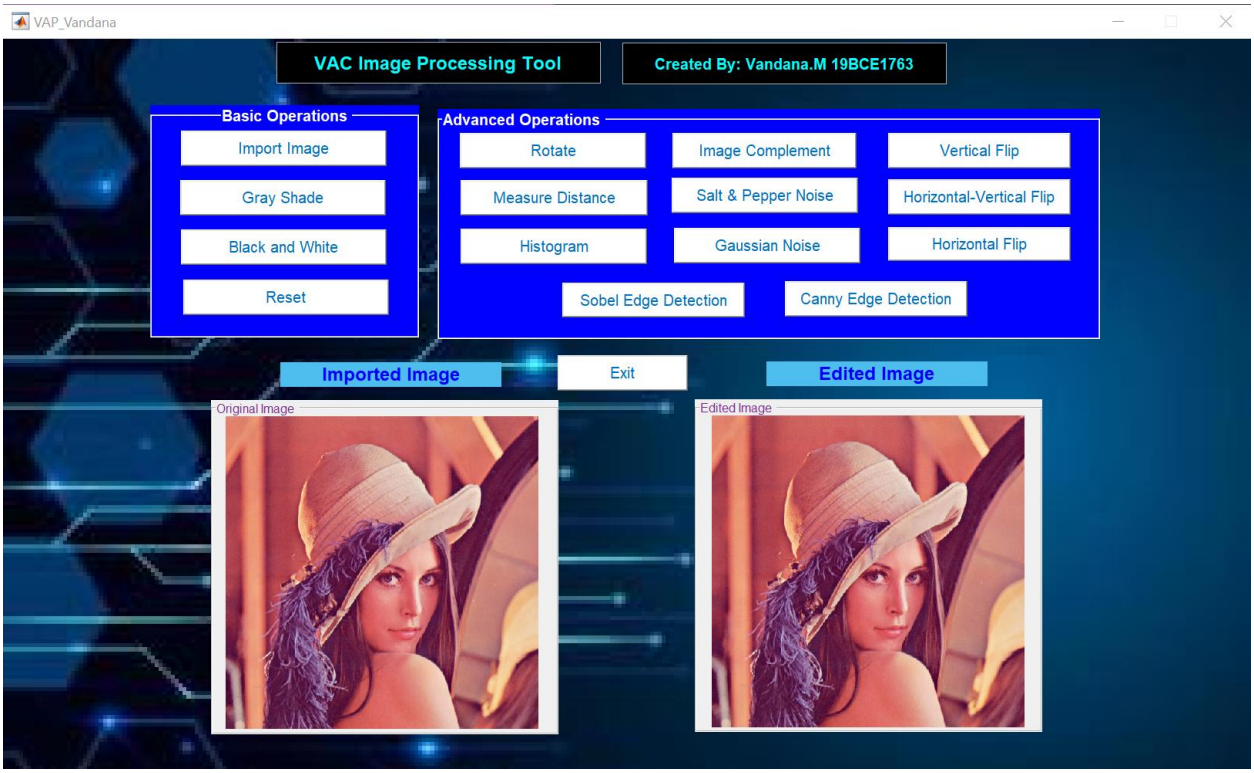
- Gray Shade



- Black and White

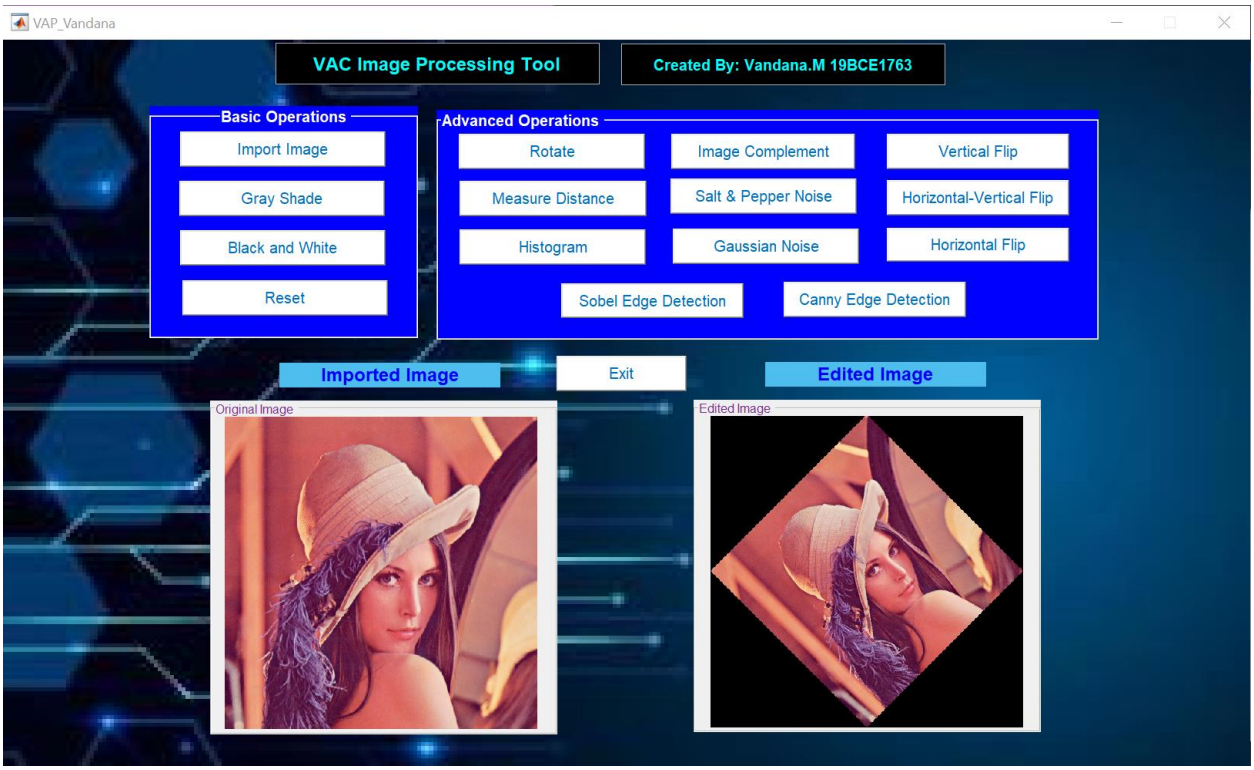


- Reset

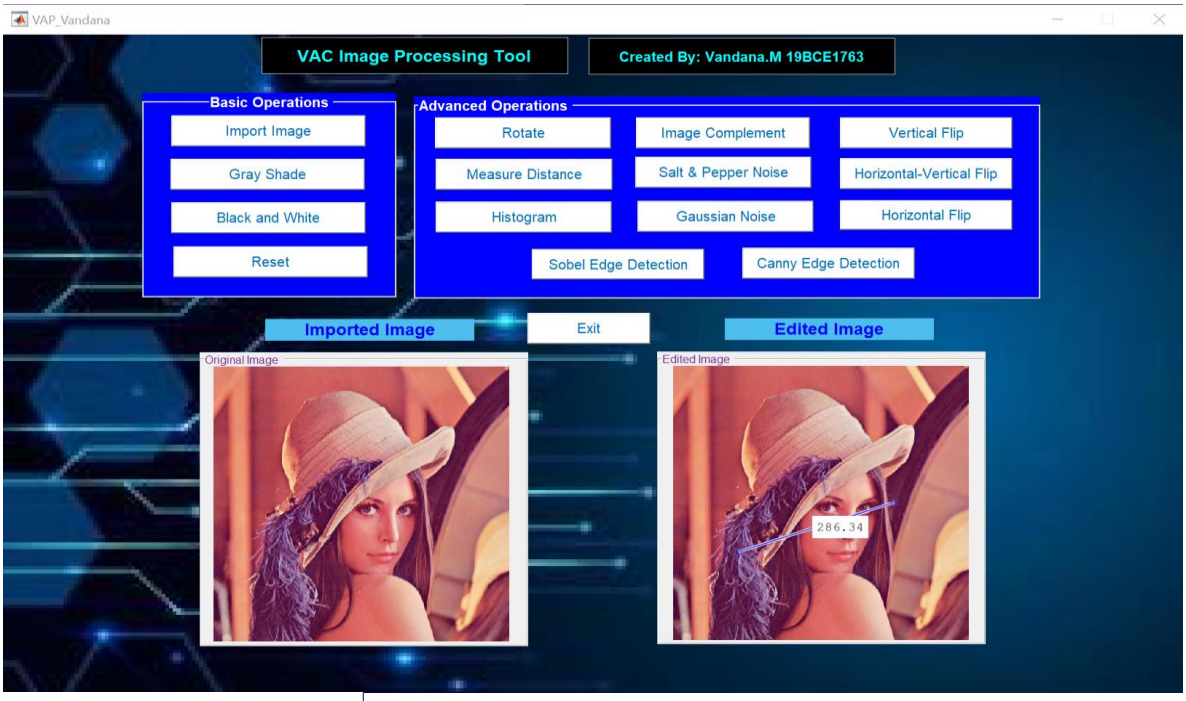


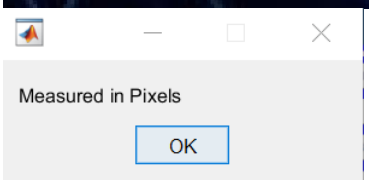
Advanced Operations

- Rotate

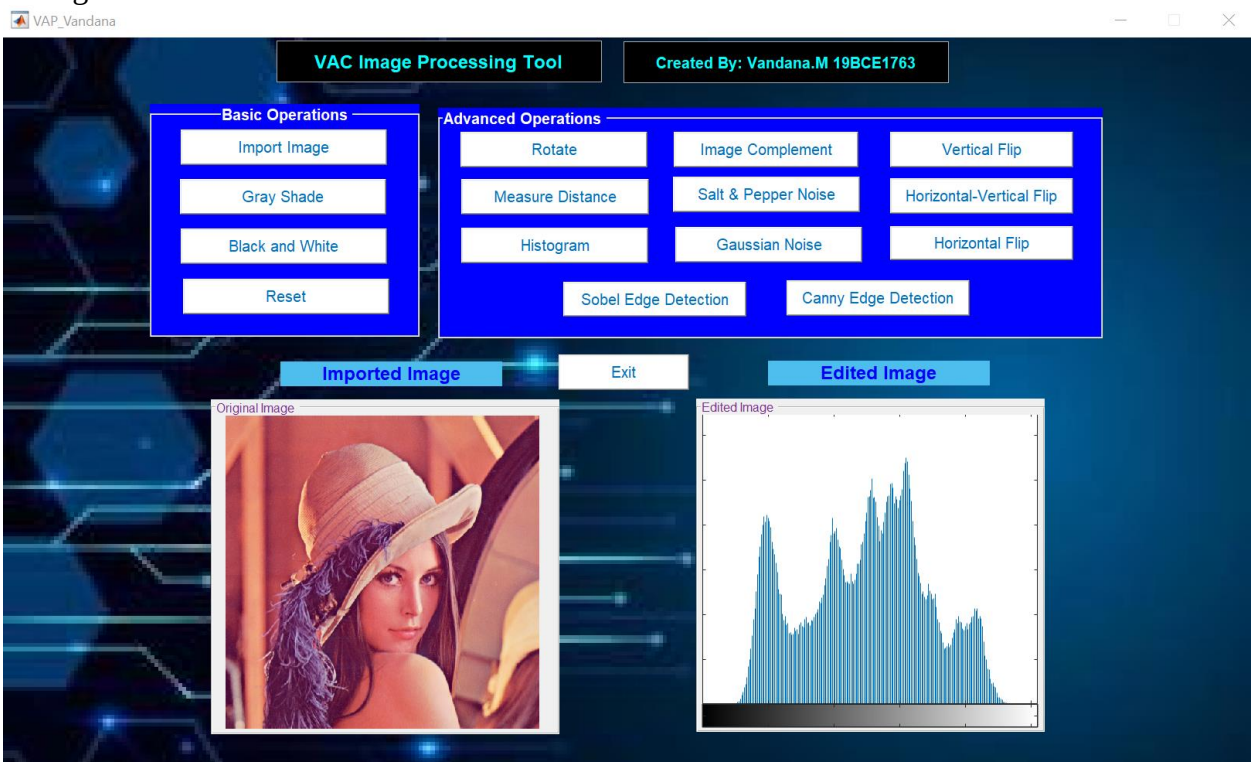


- Measure Distance

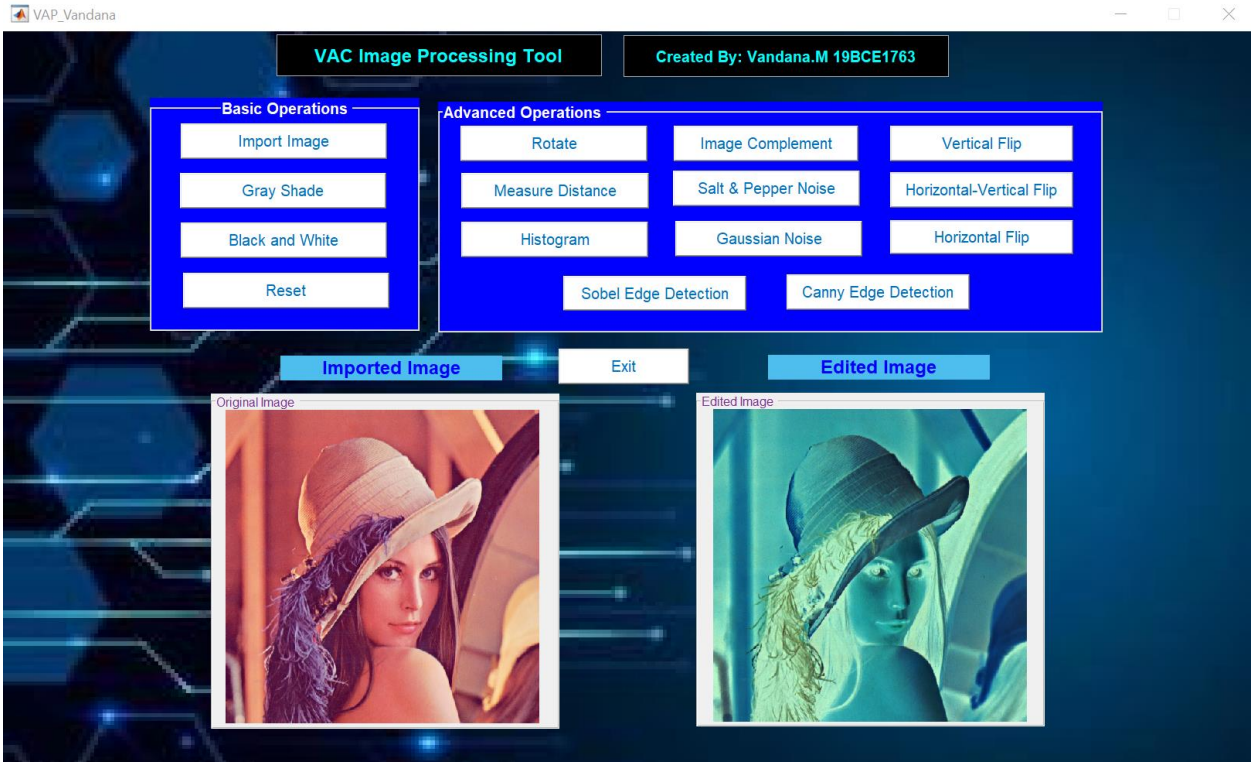




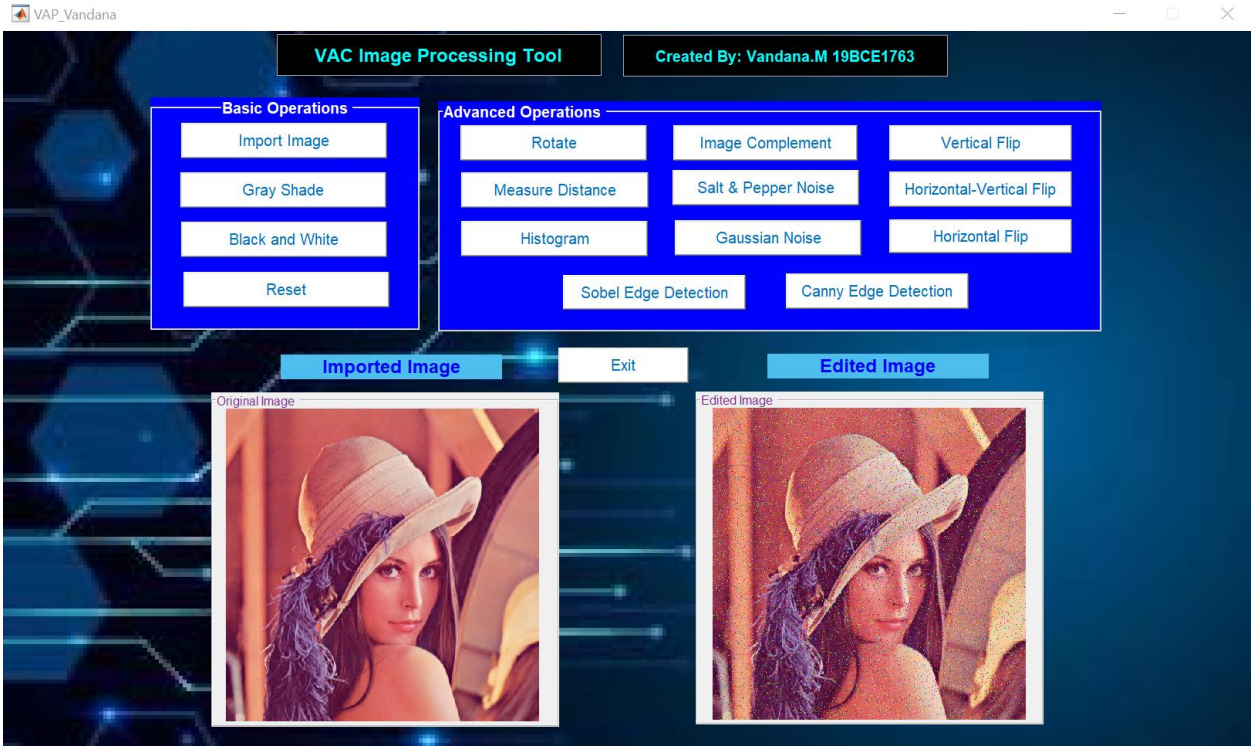
- Histogram



- Image Complement

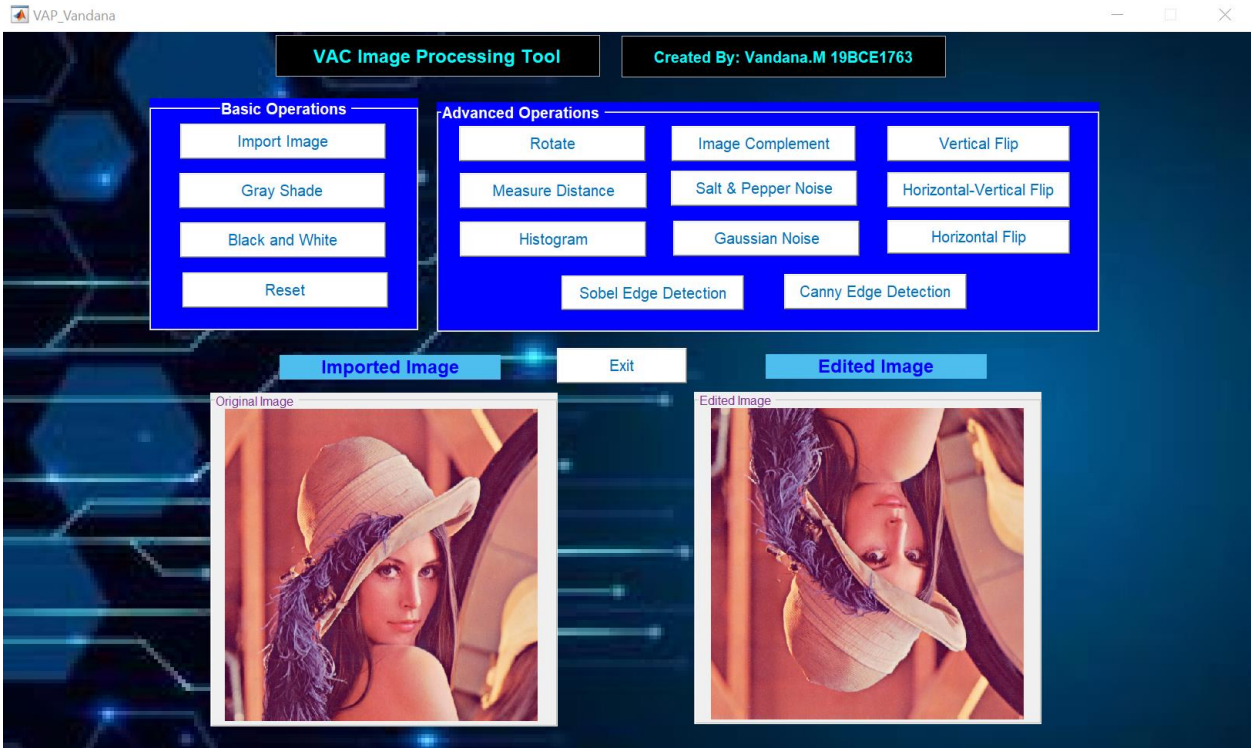


- Salt & Pepper Noise

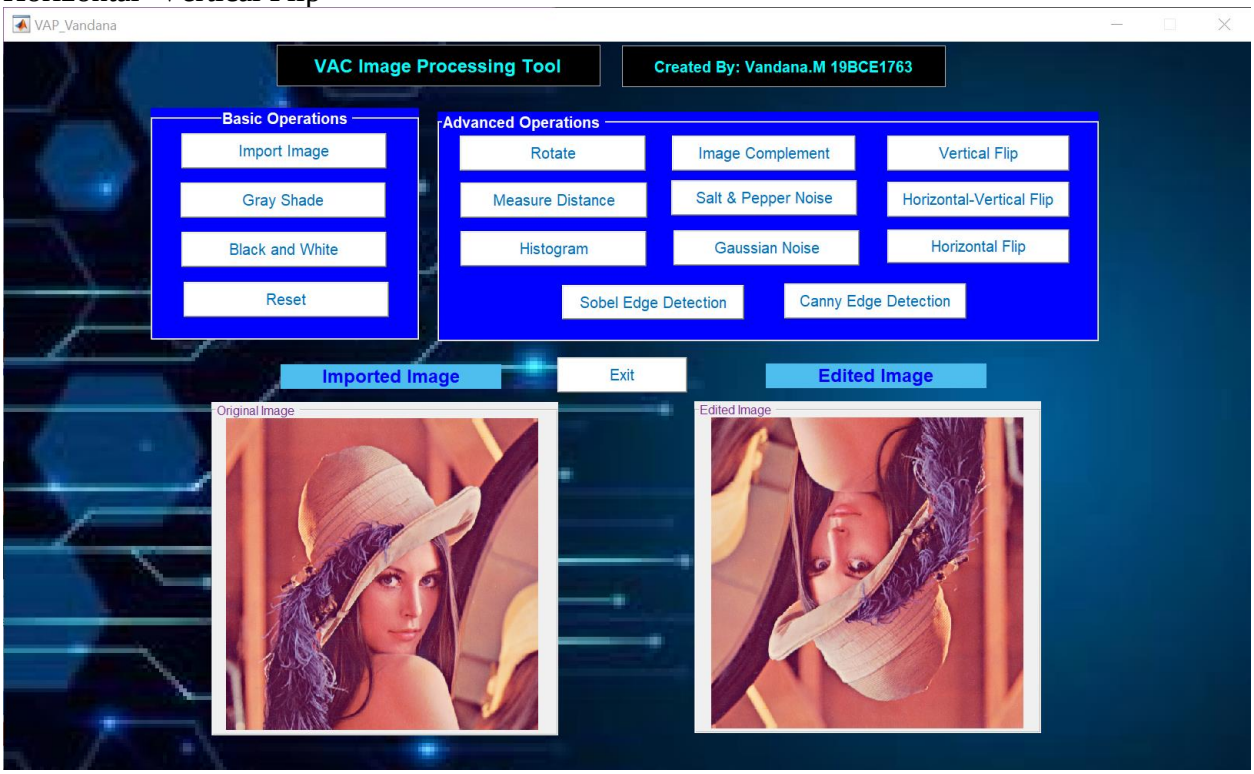


- Gaussian Noise

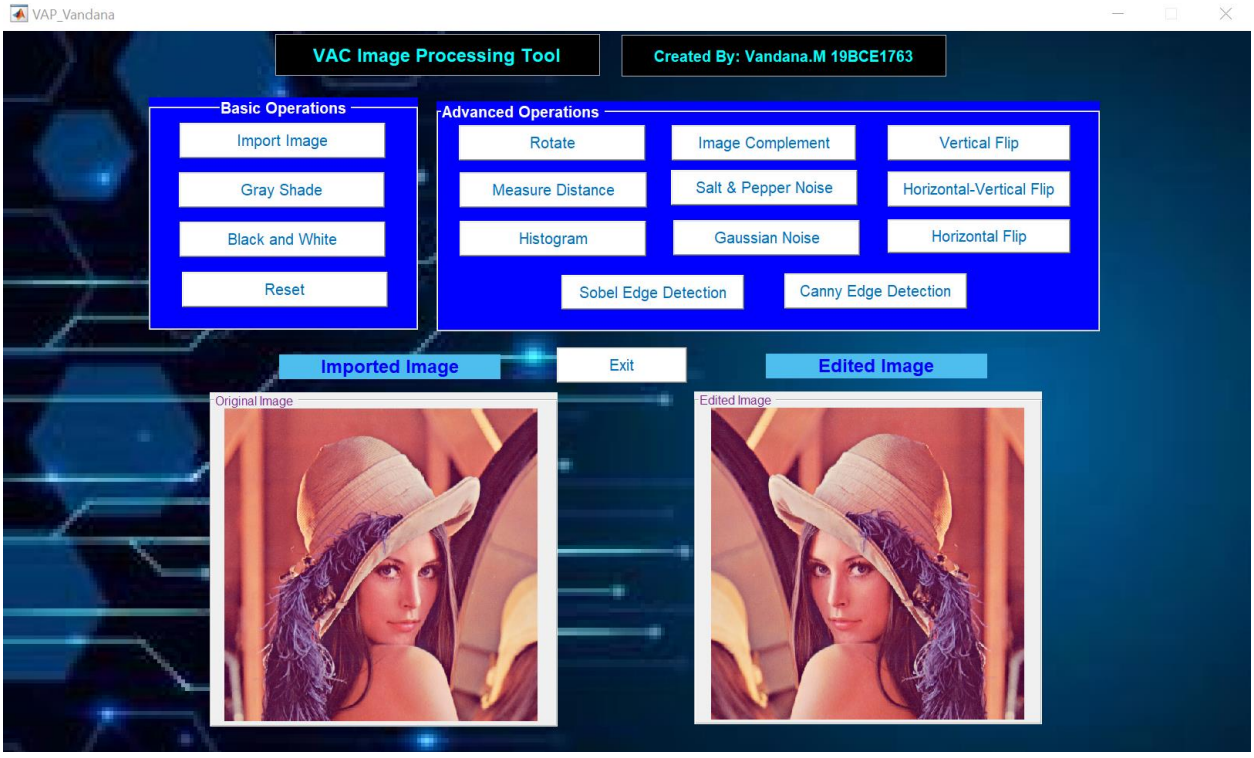
- Vertical Flip



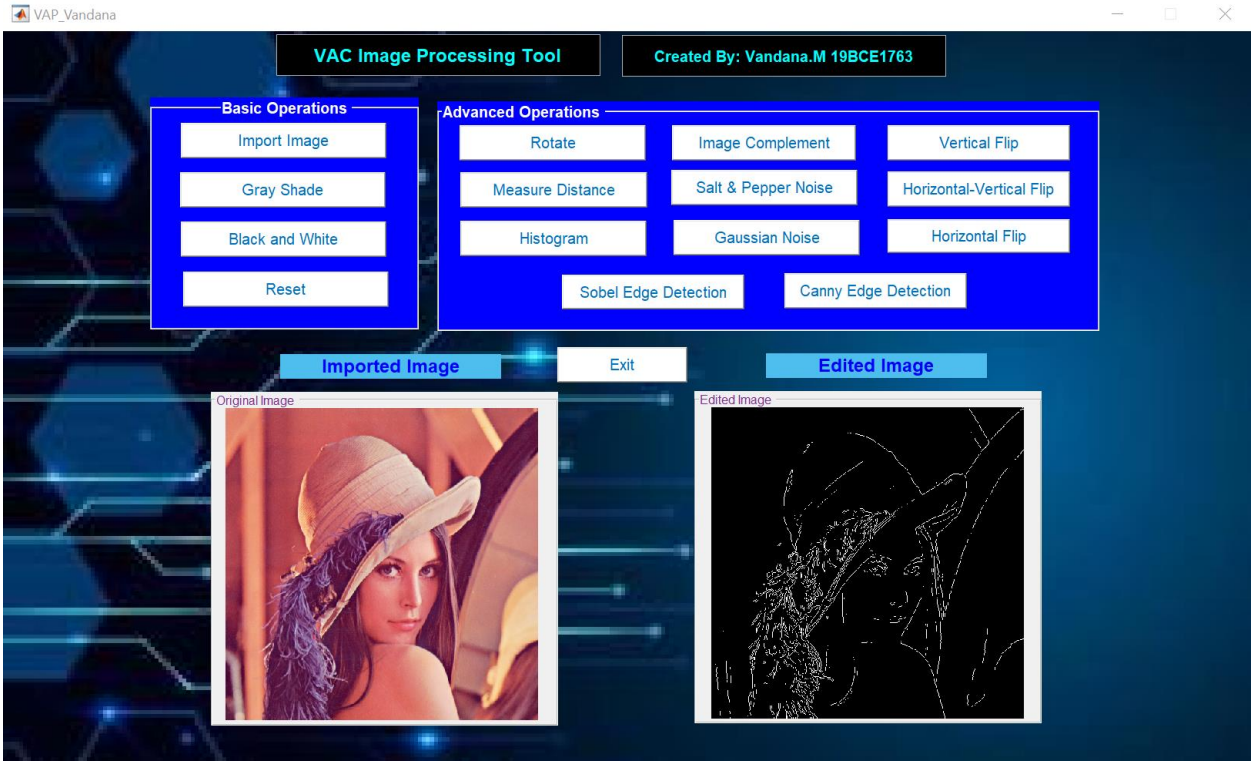
- Horizontal- Vertical Flip



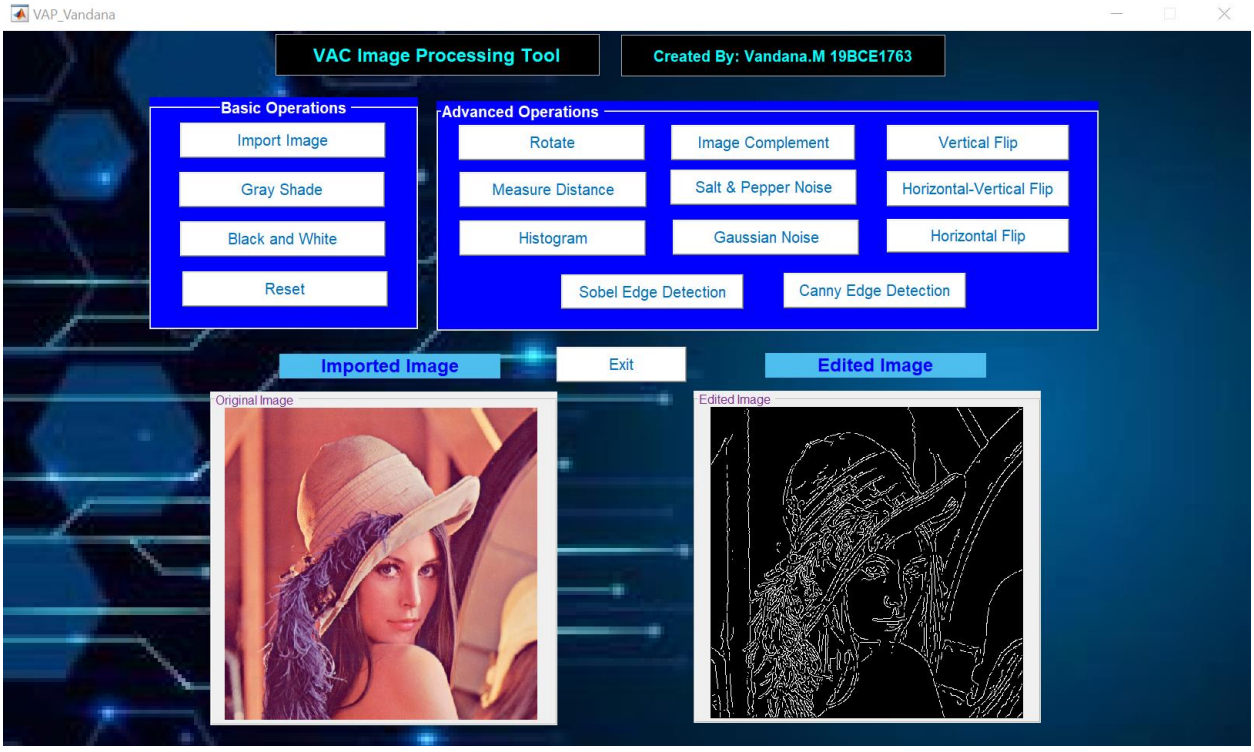
- Horizontal Flip



- Sobel Edge Detection



- Canny Edge Detection



CONCLUSION

In this course, I explored various forms of noises, filters, image manipulation such as rotation, translation, addition, subtraction, scalar and vector multiplication and division, steganography, edge detection and so on. All of these were explored on MATLAB software and have been helpful in understanding and learning about the basics of image processing. We had different lab experiments which are depicted above and a final project that I used GUIDE framework to develop an application. In conclusion, I have been able to acquire the basic understanding and this course served as a crucial stepping stone for more exploration in image processing.

REFERENCES

[1] "MATLAB Documentation - MathWorks India." Matlab, in.mathworks.com/help/matlab. Accessed 20 Apr. 2022.

APPENDIX

The code for the project is attached below.

```
function varargout = VAP_Vandana(varargin)

% Code Initialization
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VAP_Vandana_OpeningFcn, ...
                   'gui_OutputFcn',  @VAP_Vandana_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End of Code Initialization


function VAP_Vandana_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output arguments
```

```
% hObject    the handle to the object that generated the callback like buttons
% eventdata  this stores special data for some specific callbacks like key
presses or scroll actions
% handles    the current contents of the data stored with the figure. This
retrieved with guidata(hObject) at the time of the callback.
% varargin   input variable in a function definition statement that enables
the function to accept any number of input arguments.

% Choose default command line output
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

buat_axes = axes('unit', 'normalized', 'position', [0 0  1 1]);
backgroundnya = imread('bg.jpg');
imagesc(backgroundnya);
set(buat_axes, 'handlevisibility', 'off', 'visible', 'off')

% Outputs are returned to the command line.
function varargout = VAP_Vandana_OutputFcn(hObject, eventdata, handles)
% varargout  output variable in a function definition statement that enables
the function to return any number of output arguments

% Get default command line output from handles structure
varargout{1} = handles.output;

% Histogram Button
function pushbutton7_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
input=a;
input=rgb2gray(input);
axes(handles.axes2);
imhist(input);


% Measure Distance Button
function pushbutton9_Callback(hObject, eventdata, handles)
l=imdistline();
msgbox('Measured in Pixels');
dist = getDistance(l);


% Rotate Button
function pushbutton11_Callback(hObject, eventdata, handles)
a=getappdata(0, 'a');
rotate=imrotate(a,45);
axes(handles.axes2);
imshow(rotate);


% Horizontal Flip Button
function pushbutton12_Callback(hObject, eventdata, handles)
I=getappdata(0,'a');
I2=flipdim(I,2);
```

```
axes(handles.axes2);
imshow(I2);


% Image Complement Button
function pushbutton13_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
IM2=imcomplement(a);
axes(handles.axes2);
imshow(IM2);


% Salt and Pepper Noise Button
function pushbutton16_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
noise=imnoise(a, 'salt & pepper');
axes(handles.axes2);
imshow(noise);


% Gaussian Noise Button
function pushbutton17_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
noise=imnoise(a, 'gaussian');
axes(handles.axes2);
imshow(noise);


% Vertical Flip Button
function pushbutton18_Callback(hObject, eventdata, handles)
I=getappdata(0,'a');
I3=flipdim(I,1);
axes(handles.axes2);
imshow(I3);


% Horizontal Vertical Button
function pushbutton19_Callback(hObject, eventdata, handles)
I=getappdata(0,'a');
I2=flipdim(I,2);
I3=flipdim(I,1);
I4=flipdim(I3,2);
axes(handles.axes2);
imshow(I4);

% Canny Edge Detection Button
function pushbutton20_Callback(hObject, eventdata, handles)
I=getappdata(0,'a');
I=rgb2gray(I);
BW2=edge(I,'canny');
axes(handles.axes2);
imshow(BW2);


% Sobel Edge Detection Button
```

```
function pushbutton21_Callback(hObject, eventdata, handles)
I=getappdata(0,'a');
I=rgb2gray(I);
BW1=edge(I,'sobel');
axes(handles.axes2);
imshow(BW1);


% Import Image Button
function pushbutton1_Callback(hObject, eventdata, handles)
a=uigetfile();
filename=a;
setappdata(0,'filename',filename);
a=imread(a);
axes(handles.axes1);
imshow(a);
setappdata(0,'a',a);
setappdata(0,'filename',a);


% Greyshade Button
function pushbutton2_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
a_gray=rgb2gray(a);
setappdata(0,'filename', a_gray);
axes(handles.axes2);
imshow(a_gray);


% Black and White Button
function pushbutton4_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
a_bw=im2bw(a,.57);
axes(handles.axes2);
imshow(a_bw);
setappdata(0,'filename',a_bw);


% Reset Button
function pushbutton5_Callback(hObject, eventdata, handles)
a=getappdata(0,'a');
imshow(a);


% Exit Button
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
msgbox('Exiting Image Processing Tool by Vandana');
pause(1);
close();
close();
```

```
function edit1_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```