

Lab Report

Digital Image Processing

ECL-415

**Koushik Valluri (BT17ECE001)**

**UNDER THE GUIDANCE OF**

**DR. Tapan Kumar Jain**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION**

**ENGINEERING IIIT, NAGPUR**

**2020-2021**

Index

|  |  |
| --- | --- |
| S. No. | Name of Experiment |
|  |  |
| 1. | Basic Image operations |
|  |  |
| 2. | Histogram Equalization |
|  |  |
| 3. | Applying different kinds of Filters. |
|  |  |
| 4. | Histogram Specification |
|  |  |
| 5. | Edge Detection using DWT2 |
|  |  |
| 6. | Edge Detection using Sobel |
|  |  |
| 7. | Histogram Stretching |
|  |  |
| 8. | Near Neighbourhood Algorithm |
|  |  |
| 9. | Separating RGB Components |
|  |  |
| 10. | Watermarking |
|  |  |
| 11. | Zig\_Zag\_DCT |
|  |  |
| 12. | DCT based Compression |
|  |  |
| 13. | Run Length Encoding |
|  |  |
| 14. | Toboggan Contrast Enhancement |
|  |  |
| 15. | Delta Modulation |
|  |  |

**Git Hub Link:**

# **https://github.com/KoushikValluri/DIP**

**LAB -1**

**Aim :-** Basic Image operations.

**Code :-**

**# BT17ECE001**

**clc;**

**clear;**

**close all;**

**i = imread ("buddy.jpg"); #reading image**

**[x1,y1,z1] = size(i); #size of image**

**[i,cmap] = imread("buddy.jpg");**

**gray\_map = rgb2gray(i);#RGB to GraySCALE Image.**

**imshow(gray\_map); #showing image.**

**j = [0:10]; #creating a matrix of 1\*11 with values 0 to 10**

**R = ones(1,11);**

**im = j'\*R; #multipling transpose of j with row matrix**

**#having 11 times 1.**

**imshow(im,[]); #intensity.**

**I1 = gray\_map'; #rotating image.**

**figure;**

**imshow(I1); #rotated image.**

**imf=imfinfo('buddy.jpg'); #To determine the details of image.**

**pixelValue = i(3,3); #getting pixel value.**

**sprintf('the value of the pixel is %d',pixelValue);**

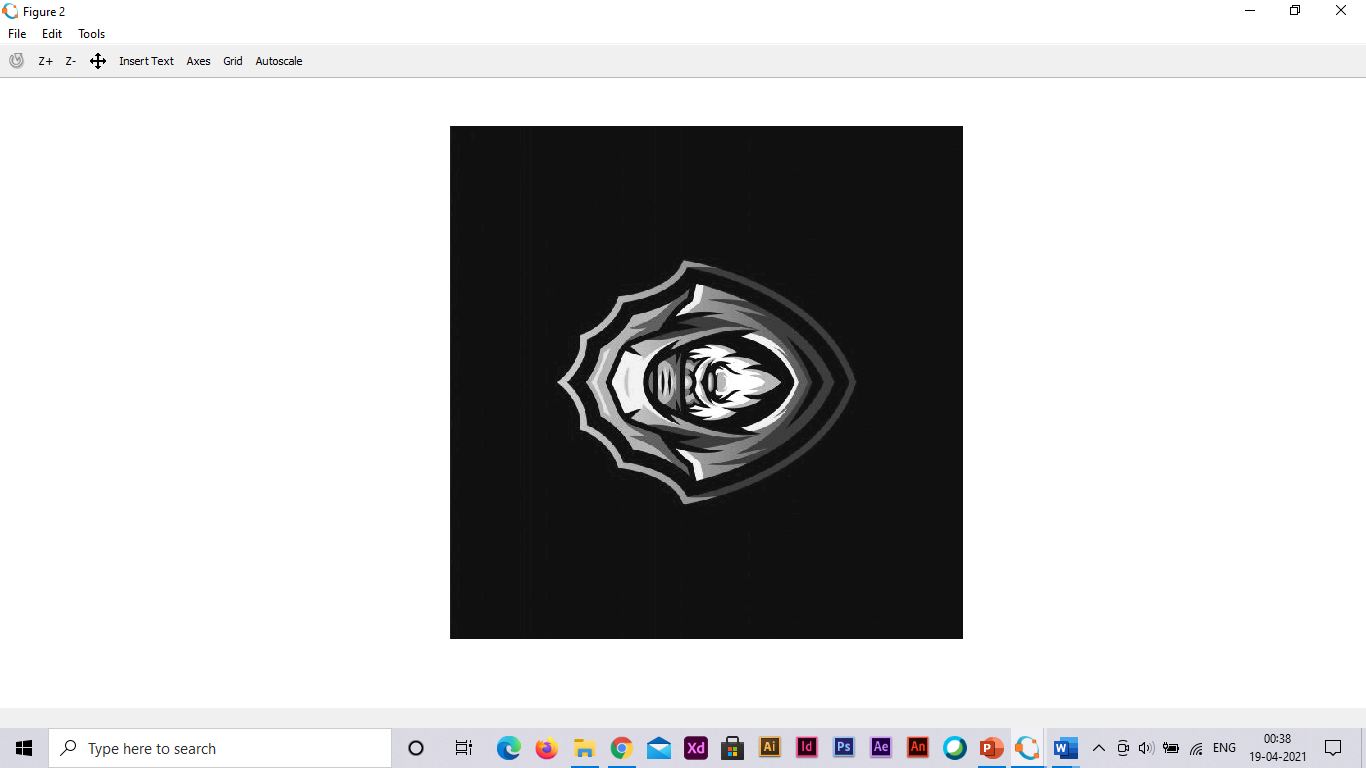
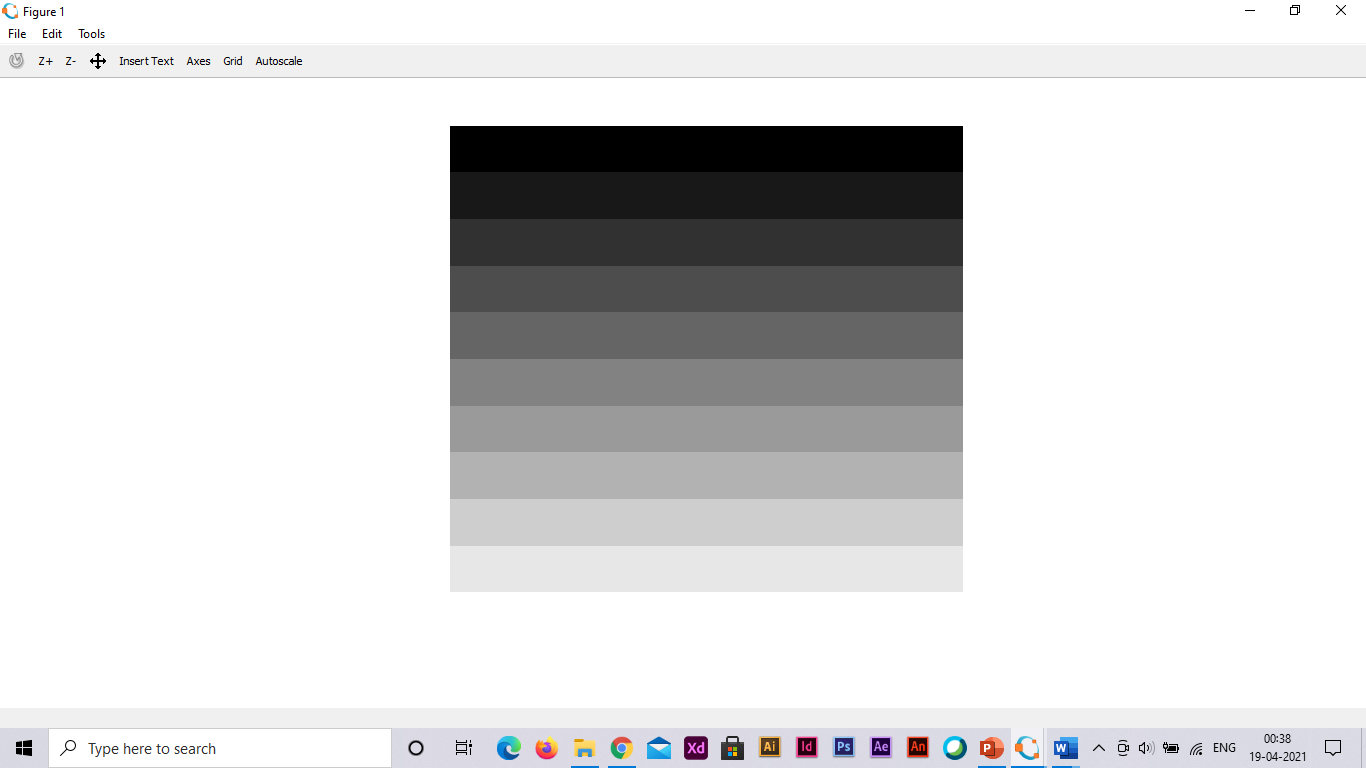
**rgb = i(4,18,:); #this commond will return rgb color value of the pixel 4,16**

**R = rgb(:,:,1); #to see the value of red color.**

**G = rgb(:,:,2); #to see the value of green color.**

**B = rgb(:,:,3); #to see the value of blue color.**

**RESULTS:**

****

**LAB 2**

**Aim :-** Histogram Equalization.

**Code :-**

imshow(uint8(Mat)); # BT17ECE001

# Histogram Equalization.

clc;

clear;

close all;

# Representing Image with pixel values.

Mat = [52 55 61 59 79 61 76 61;

62 59 55 104 94 85 59 71;

63 65 66 113 144 104 63 72;

64 70 70 126 154 109 71 69;

67 73 68 106 122 88 68 68;

68 79 60 70 77 66 58 75;

69 85 64 58 55 61 65 83;

70 87 69 68 65 73 78 90];

Mat = double(Mat);

s = size(Mat);

# For Getting the Histogram of the original image

histogram = zeros(1,256);

for i = 1:s(1)

for j = 1:s(2)

for k = 0:255

if Mat(i,j) == k

(k+1) = (k+1)+1;

end

end

end

end

# Generating frequency out of histogram by dividing by total no. of pixels

freq = (1/(s(1)\*s(2)))\*histogram;

# Generating CDF out of freq.

cdf = zeros(1,256);

cdf(1) = freq(1);#First value of cdf is same as pdf.

for i = 2:256

cdf(i) = cdf(i-1)+(i);

end

hist\_eq = zeros(s);

for i = 1:s(1)#for tracing the rows of our image

for j = 1:s(2)#for tracing the columns of our image

a = (Mat(i,j)+1);#pixel values in our image

hist\_eq(i,j) = cdf(a);#Making the output image using cdf as the transformation function

end

end

# For Getting the Histogram of the image

histogram = zeros(1,256);

for i = 1:s(1)

for j = 1:s(2)

for k = 0:255

if hist\_eq(i,j) == k

histogram(k+1) = histogram(k+1)+1;

end

end

end

end

# Plotting Figures.

subplot(2,2,1);

stem();

subplot(2,2,3);

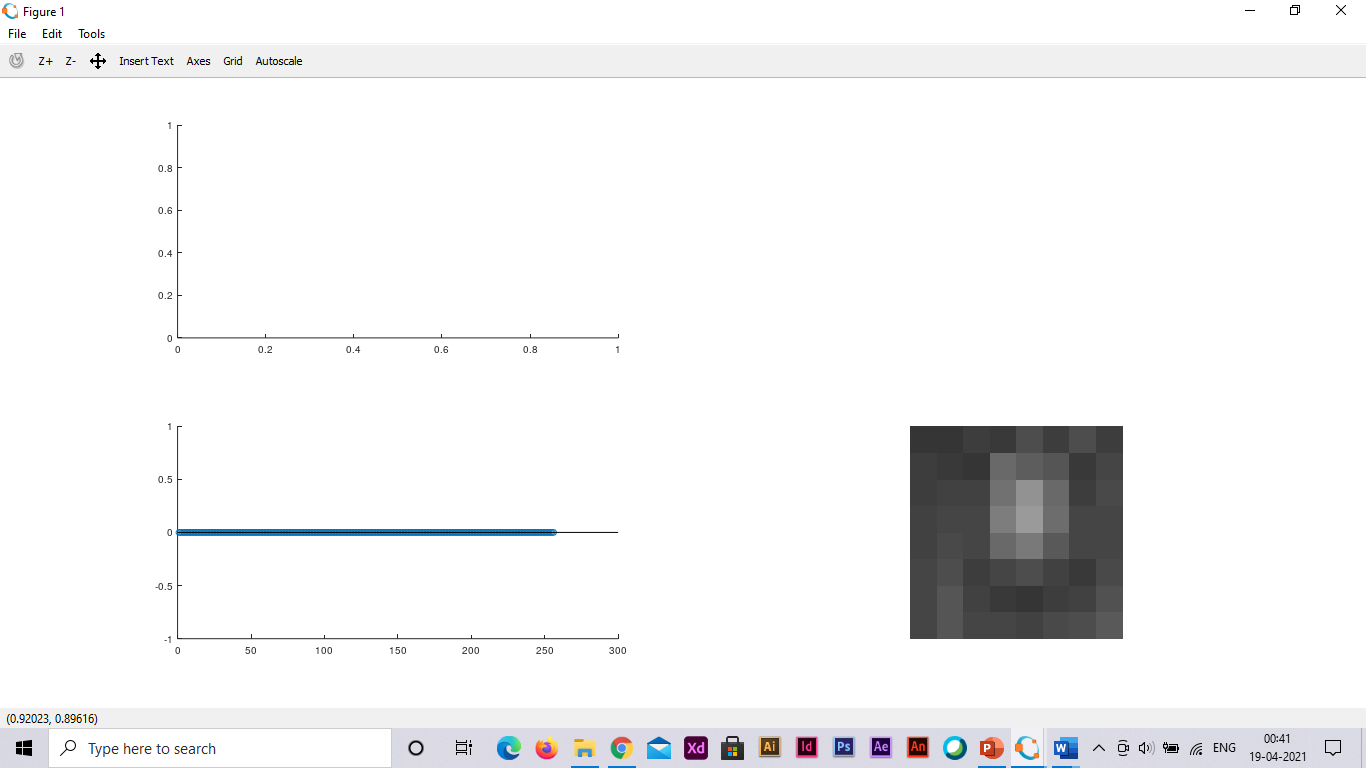
stem(histogram);

subplot(2,2,2);

imshow(uint8(hist\_eq));

subplot(2,2,4);

RESULTS:



**LAB 3**

**Aim :-** Applying different kinds of Filters.

**Code :-**

# BT17ECE001

# Applying different kinds of Filters.

clc; clear all; close all;

I = imread('buddy.jpg'); #Importing the Image

I = imresize(I, 0.5); #Resizing Image

I = rgb2gray(I); #Converting coloured image to gray

figure(1);

imshow(I); #showing image

# Filter Designing

HPF = [-1,-1,-1;-1,8,-1;-1,-1,-1];

#This is a simple Laplacian Mask

LPF = [1,1,1;1,1,1;1,1,1] .\* (1/9);

#This is a averaging filter mask

# Applying the Filters

I\_HPF = conv2(I,HPF);

I\_LPF = conv2(I,LPF);

figure(2);

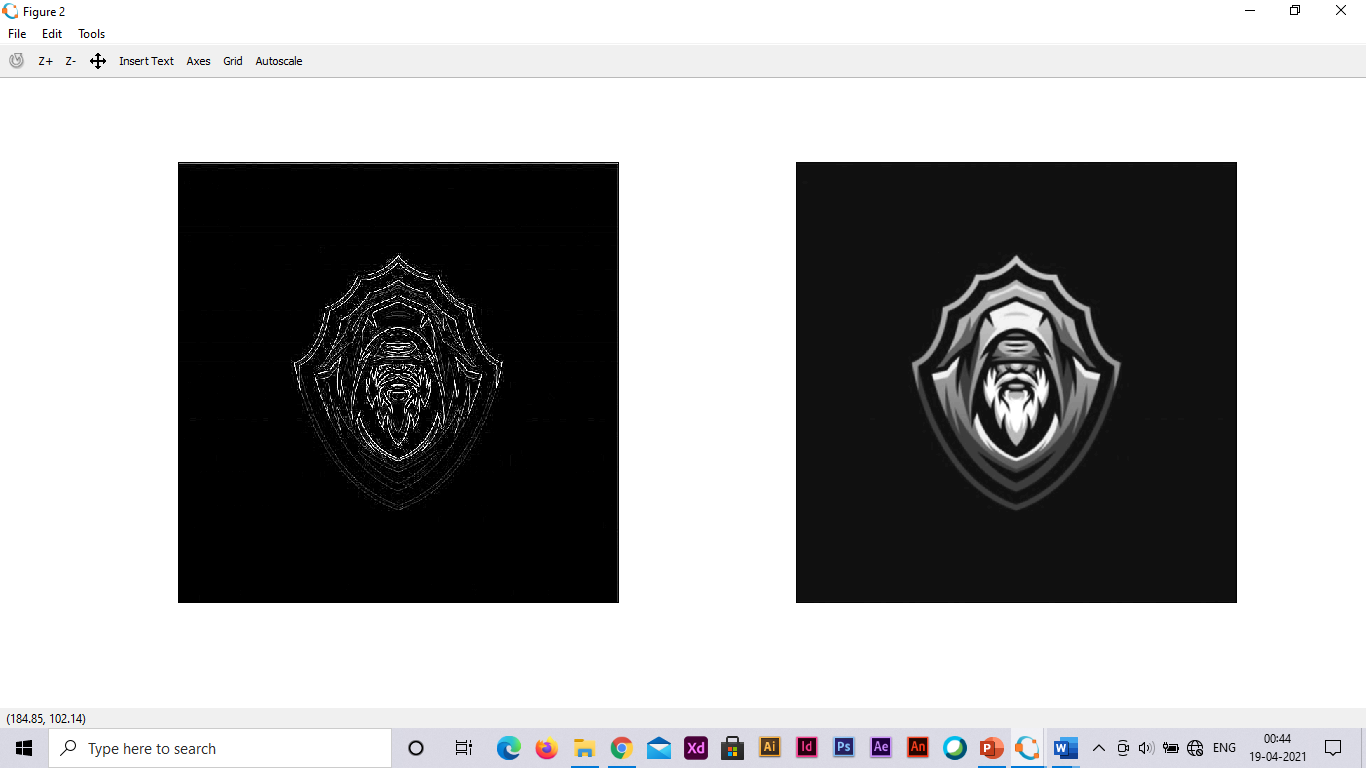
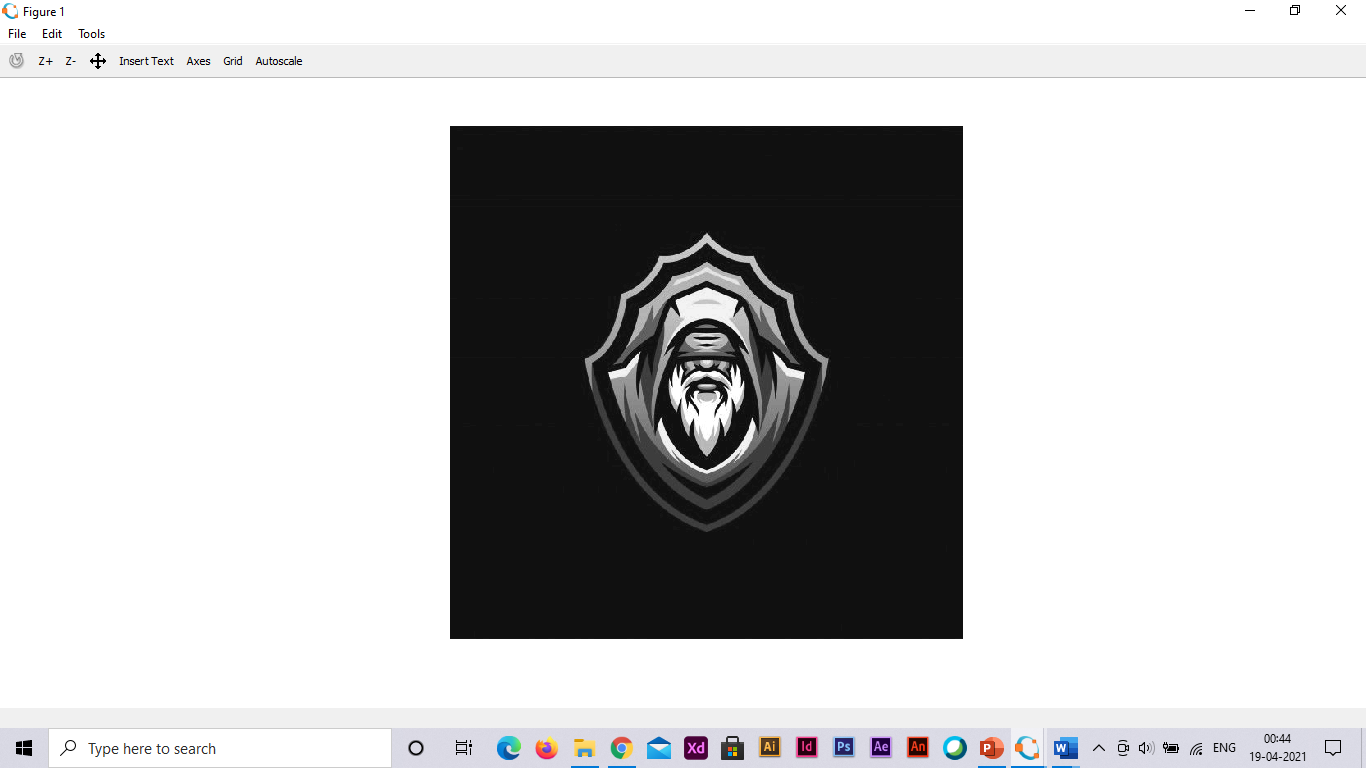
subplot(121);

imshow(uint8(I\_HPF)); #plotting after applying Laplacian Mask

subplot(122);

imshow(uint8(I\_LPF)); #plotting after applying Average Filtering Mask

RESULTS:



**LAB 4**

**Aim :-** Histogram Specification

**Code :-**

# BT17ECE001

# Histogram Specification

clc; clear all; close all;

image = [1 3 5; 4 4 3; 5 2 2]; #defining image pixels

level = [0 1 2 3 4 5 6 7]; #defining levels

pixels = zeros(1,9);

for i = 1:7

for j = 1:9

if image(j) == level(i)

pixels(i) = pixels(i) + 1; #calculating pixels

end

end

end

pixels = pixels(1:8);

cdf = zeros(1,8);

cdf(1) = pixels(1);

for i = 2:8

cdf(i) = cdf(i-1) + pixels(i); #cumulative distribution function

end

input\_equ = round(cdf\*7./9); #rounding off the values

target = [0 0 0 0 2 2 4 1]; #defining target

cdf\_t = zeros(1,8);

cdf\_t(1) = target(1);

for i = 2:8

cdf\_t(i) = cdf\_t(i-1) + target(i);

end

target\_equ = round(cdf\_t\*7./9); #rounding off the values

map = zeros(1,8);

j = 1;

for i = 1:8

for j = 1:8

if input\_equ(i) <= target\_equ(j)

map(i) = level(j); #mapping with targets

break;

end

end

end

**LAB 5**

**Aim :-** Edge Detection using DWT2

**Code :-**

# BT17ECE001

# Edge Detection using DWT2

clc; clear all; close all;

I = imread('buddy.jpg'); #Importing the Image

I = imresize(I, 0.5); #Resizing Image

I = rgb2gray(I); #Converting coloured Image to Gray

figure(1);

imshow(I); #showing image

# Edges using DWT2

[eA, eH, eV, eD] = dwt2(I,'haar');

figure(2);

subplot(221);

imshow(uint8(eA)); #showing image

subplot(222);

imshow(uint8(eH)); #showing image

subplot(223);

imshow(uint8(eV)); #showing image

subplot(224);

imshow(uint8(eD)); #showing image

**LAB 6**

**Aim :-** Edge Detection using Sobel

**Code :-**

# BT17ECE001

# Edge Detection using Sobel

clc; clear all; close all;

I = imread('buddy.jpg'); #Importing the Image

I = imresize(I, 0.5); #Resizing Image

I = rgb2gray(I); #Converting coloured Image to Gray

figure(1);

imshow(I); #showing image

# Edges using Sobel

EdgeDetection\_Sobel = edge(I,'sobel');

figure(4);

imshow(EdgeDetection\_Sobel); #Showing Image

title("Edge Detection using Sobel");

**LAB 7**

**Aim :-** Histogram Stretching.

**Code :-**

# BT17ECE001

# Histogram Stretching

clc;clear;close all;

SMin = 10;

SMax = 255;

I = imread("buddy.jpg");

I\_gray = rgb2gray(I); #Converting coloured Image to Gray

[r c] = size(I\_gray); #Size of above gray image

len = r \* c; #Total pixels

I\_Vector = I\_gray(:); #Convert the matrix into vector

I\_sort = sort(I\_Vector); #Sort in ascending order

I\_uniq = unique(I\_sort); #Finding unique values

IMin = min(I\_uniq);

IMax = max(I\_uniq);

# Finding Slope

Slope = (SMax-SMin)./(IMax-IMin);

S\_New = Slope.\*(I\_uniq-IMin)+SMin;

# Stretching

I\_stretch = zeros(r,c);

for jj = 1:length(I\_uniq)

I\_stretch(I\_gray == I\_uniq(jj)) = S\_New(jj);

end

I\_stretch = uint8(I\_stretch); #Stretched Image

figure;

subplot(2,2,1);

imshow(I\_gray); #showing original image

title("Original Image");

subplot(2,2,2);

imshow(I\_stretch); #showing stretched image

title("Stretched Image");

subplot(2,2,3);

histogram(I\_gray); #showing Histogram of Original Image

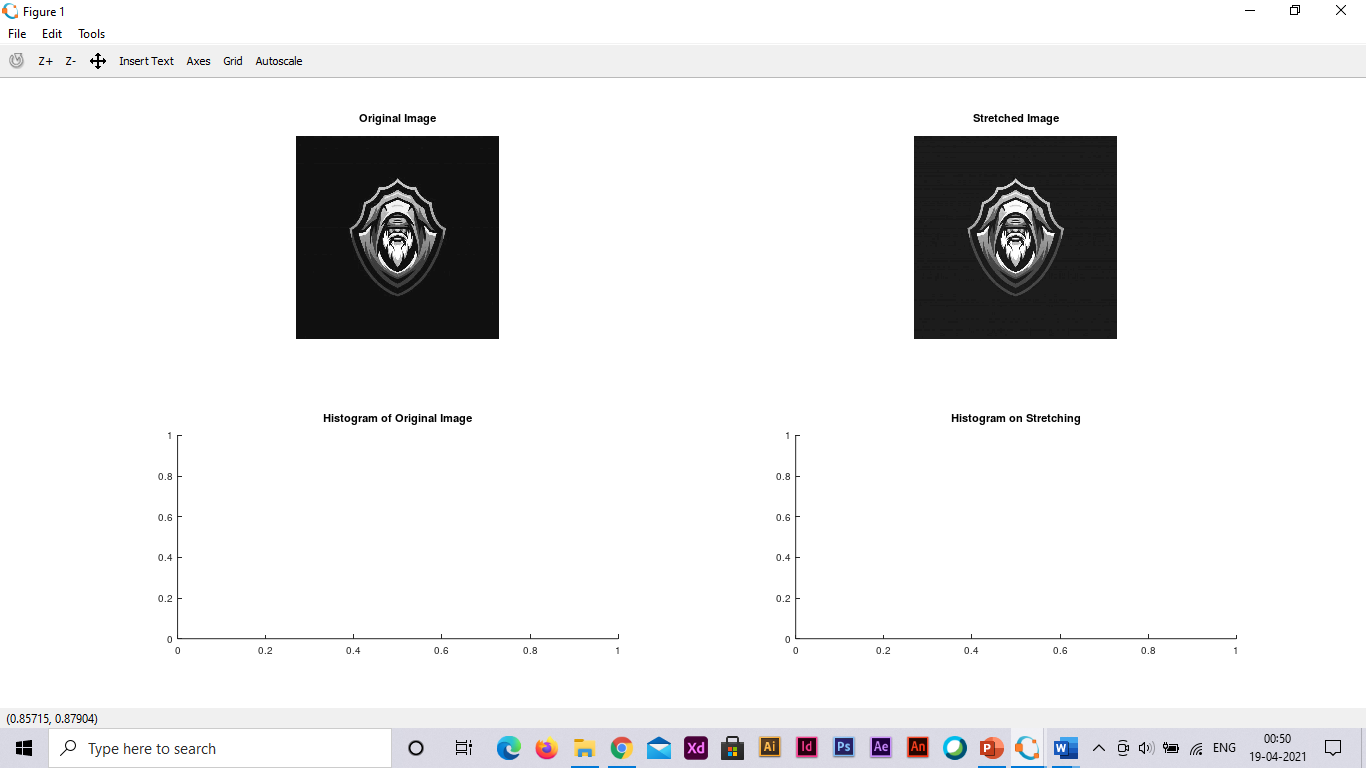
title("Histogram of Original Image");

subplot(2,2,4);

histogram(I\_stretch); #showing Histogram after Stretching

title("Histogram on Stretching");

RESULTS:



**LAB 8**

**Aim :-** Near Neighbourhood Algorithm

**Code :-**

# BT17ECE001

# Near Neighbourhood Algorithm

clc;clear all;close all;

Image = imread("buddy.jpg"); #Importing the Image

Image = rgb2gray(Image); #Converting coloured Image to Gray

imshow(Image); #Displaying Original Image

title("Original\_Image");

# Padding zeros

Hori = size(Image,1);

Wide = size(Image,2);

padded\_i = [zeros(Hori,1),Image,zeros(Hori,1)];

padded\_i = [zeros(1,Wide+2);padded\_i;

zeros(1,Wide+2)];

# Near Neighbourhood Algorithm

Image\_New = zeros(Hori,Wide);

for r = 1:Hori

for c = 1:Wide

r\_init = r+1;

c\_init = c+1;

cur\_pixel = padded\_i(r\_init,c\_init);

N\_Pixel = 1\*(cur\_pixel<padded\_i(r\_init,c\_init+1))+...

2\*(cur\_pixel<padded\_i(r\_init-1,c\_init+1))+...

4\*(cur\_pixel<padded\_i(r\_init-1,c\_init))+...

8\*(cur\_pixel<padded\_i(r\_init-1,c\_init-1))+...

16\*(cur\_pixel<padded\_i(r\_init,c\_init-1))+...

32\*(cur\_pixel<padded\_i(r\_init+1,c\_init-1))+...

64\*(cur\_pixel<padded\_i(r\_init+1,c\_init))+...

128\*(cur\_pixel<padded\_i(r\_init+1,c\_init+1));

Image\_New(r,c) = N\_Pixel; #New Image

end

end

Image\_New = uint8(Image\_New); #Converting into uint

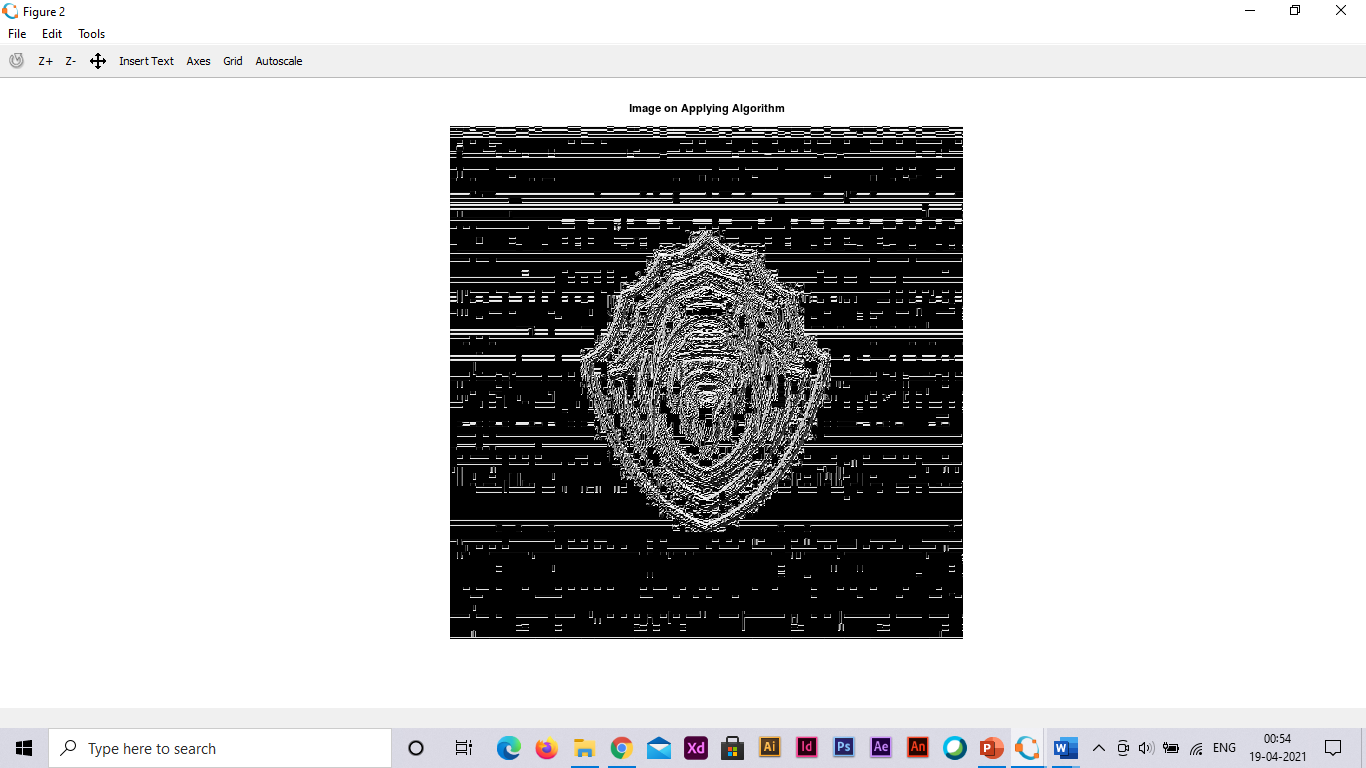
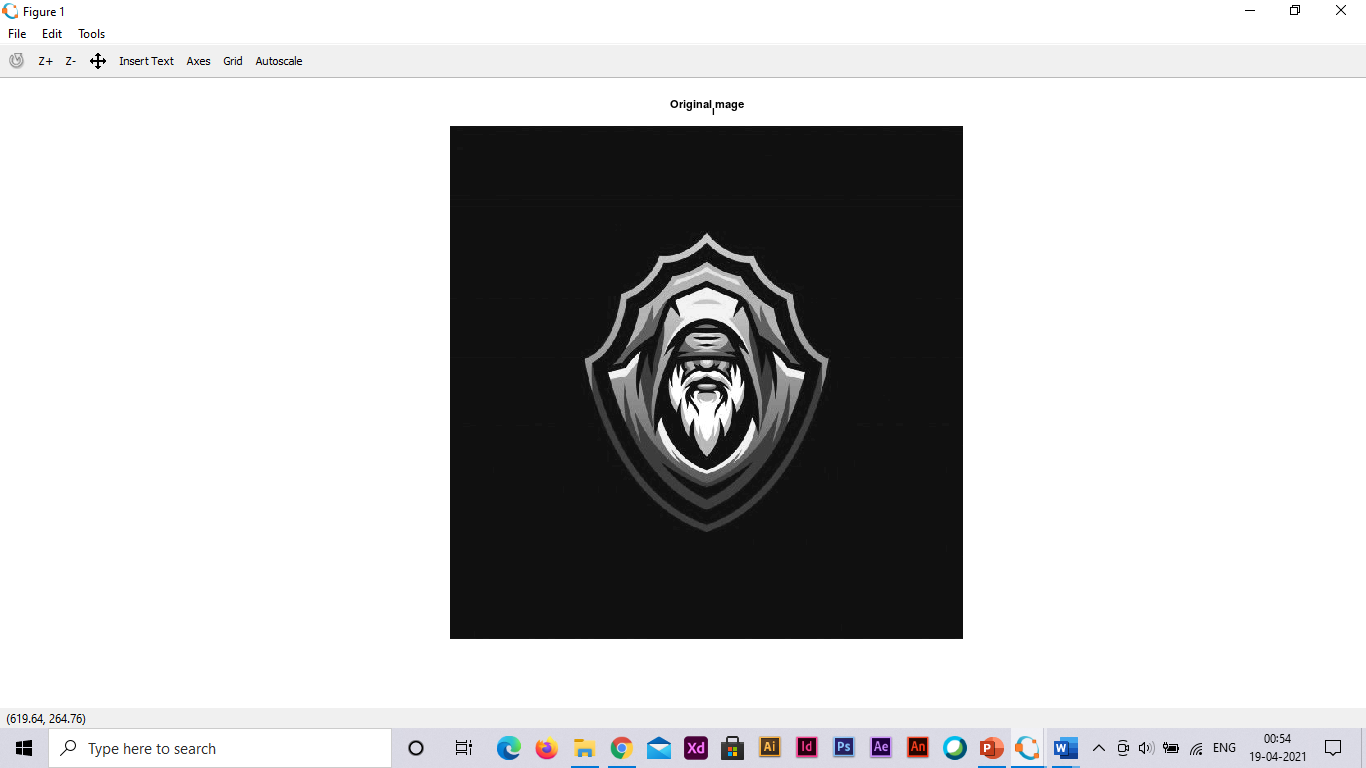
# Plotting Image

figure(2);

imshow(Image\_New); #showing image after Applying Algorithm

title("Image on Applying Algorithm");

RESULTS:



**LAB 9**

**Aim :-** Separating RGB Components

**Code :-**

# BT17ECE001

# Separating RGB Components

clc;clear all;close all;

I = imread("buddy.jpg"); #Importing the Image

r = I;

r(:,:,2) = 0;

r(:,:,3) = 0; #Making Zero Blue and Green Component

g = I;

g(:,:,1) = 0;

g(:,:,3) = 0; #Making Zero Red and Blue Component

b = I;

b(:,:,1) = 0;

b(:,:,2) = 0; #Making Zero Red and Green Component

imshow(I);

figure(1);

imshow(r); #Showing Image with only Red Component

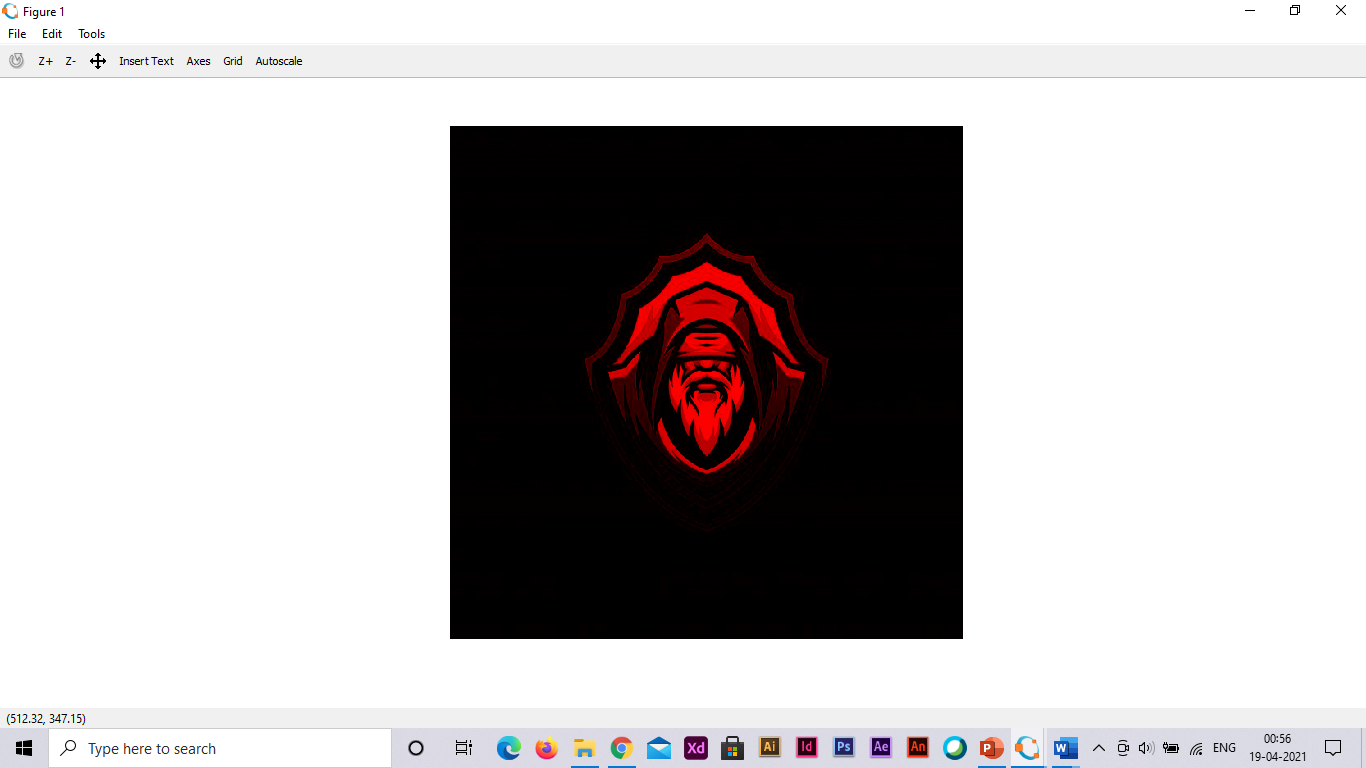
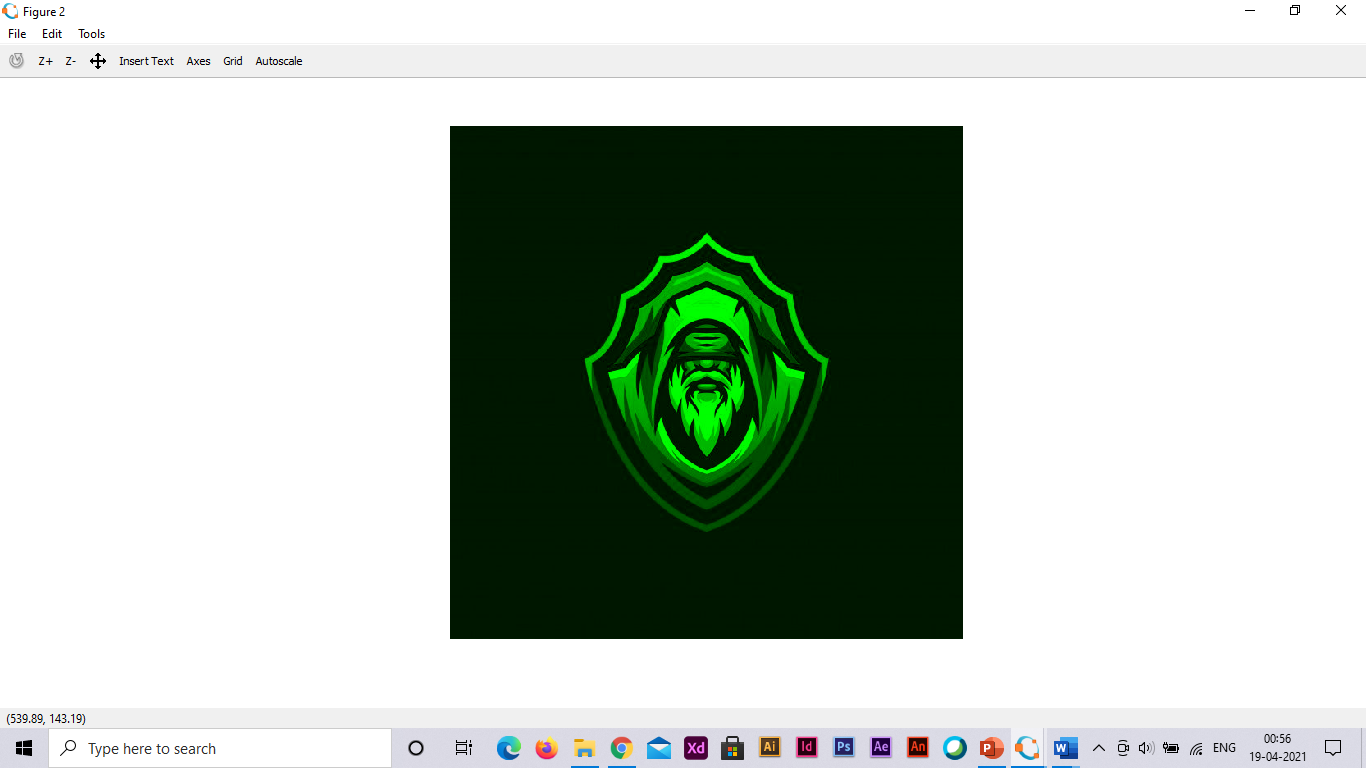
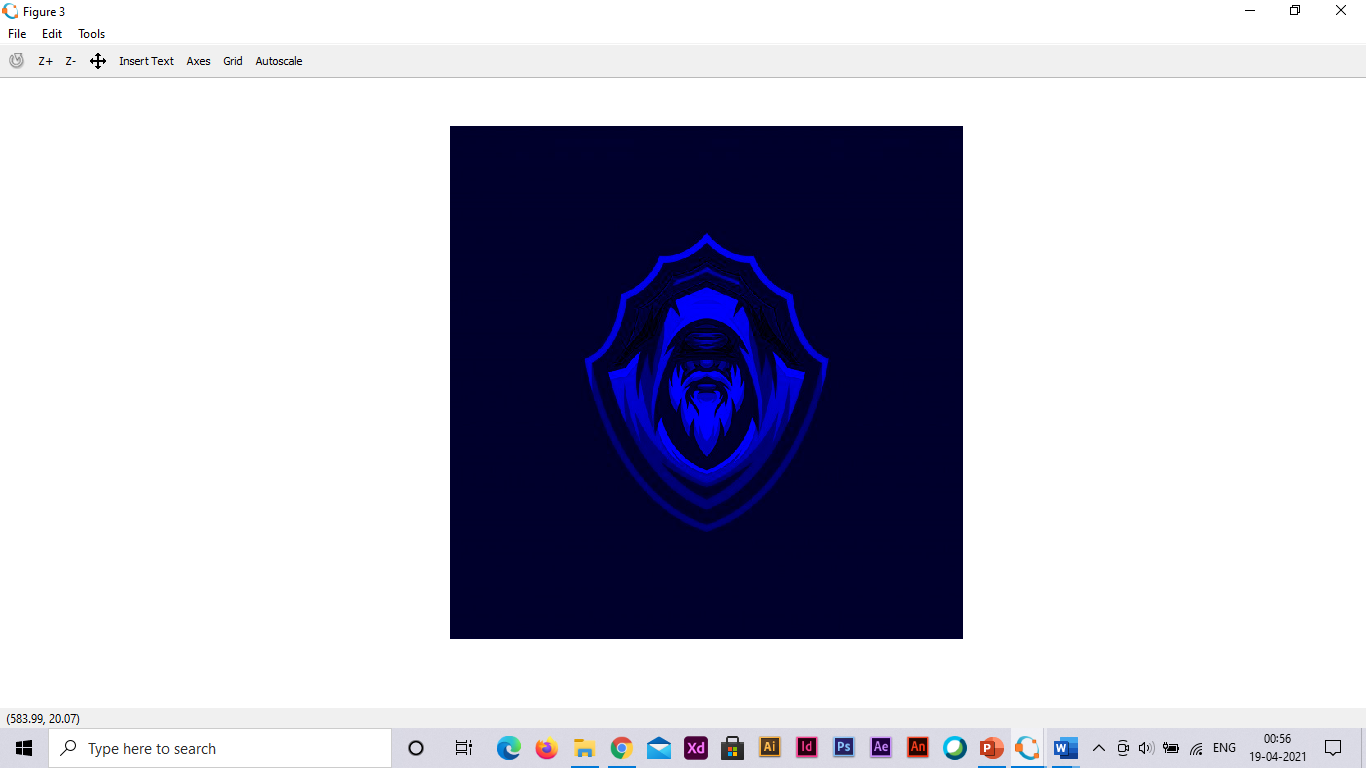
figure(2);

imshow(g); #Showing Image with only Green Component

figure(3);

imshow(b); #Showing Image with only Blue Component

RESULTS:



**LAB 10**

**Aim :-** Watermarking

**Code :-**

# BT17ECE001

# Watermarking

clc; clear all; close all;

I = rgb2gray(imread("buddy.jpg")); #Importing the Image

I = imresize(I, 0.5); #Resizing Image

imshow(I); #Showing Image

WM = imread('WM.png'); #Importing WaterMark Image

WM = padarray(WM, 213, 0,'pre');

# Bit Slicing

I\_BitSliced = bitand(I, 248); # Take first 5 MSB bits

WM\_BitSliced = bitsrl(WM, 5); # Take first 3 MSB bits

Final\_I = bitor(I\_BitSliced, WM\_BitSliced);

figure(1);

imshow(Final\_I); #showing image

# Recovering the Watermark

Recovered\_WM = bitsll(Final\_I, 5); #Recovering Watermark

figure(2);

imshow(Recovered\_WM); #Showing Watermark

# DWT2

[eAI, eHI, eVI, eDI] = dwt2(I, 'haar');

[eAWM, eHWM, eVWM, eDWM] = dwt2(WM, 'haar');

eA = 0.8\*eAI + 0.2\*eAWM;

Final\_WMImage = idwt2(eA, eHI, eVI, eDI, 'haar');

figure(3);

imshow(uint8(Final\_WMImage)); #Final Watermark Image

# Recovering the watermark

[eA, eH, eV, eD] = dwt2(Final\_WMImage, 'haar');

eA = (eA - 0.8\*eAI)/0.2;

Final\_RI = idwt2(eA, eHWM, eVWM, cDWM, 'haar'); #Final Recovered Image

figure(4);

imshow(uint8(Final\_RI)); #Showing Final Recovered Image

# BT17ECE001

# Delta Modulation

function [t m]=Delta\_Modulation(w, D)

#D=amplitude of signal

#t=output binary sequence

#w=step size

#Defining the Variables

t=0:2\*pi/100:2\*pi;

z=D\*sin(t);

plot(z)

hold on

t=[0];

h=0;

for j=1:length(z)-1

if h(j)<=z(j)

d=1;

h(j+1)=h(j)+w;

else

d=0;

h(j+1)=h(j)-w;

end

t=[t d];

end

stairs(h);

hold off;

m=sum((z-h).^2)/length(z); #Delta Modulation

end

# BT17ECE001

# Delta Modulation

function [t m]=Delta\_Modulation(w, D)

#D=amplitude of signal

#t=output binary sequence

#w=step size

#Defining the Variables

t=0:2\*pi/100:2\*pi;

z=D\*sin(t);

plot(z)

hold on

t=[0];

h=0;

for j=1:length(z)-1

if h(j)<=z(j)

d=1;

h(j+1)=h(j)+w;

else

d=0;

h(j+1)=h(j)-w;

end

t=[t d];

end

stairs(h);

hold off;

m=sum((z-h).^2)/length(z); #Delta Modulation

end

**LAB 11**

**Aim :-** Zig\_Zag\_DCT.

**Code :-**

# BT17ECE001

# Zig\_Zag\_DCT

clc; clear all; close all;

# Initialise the variables

I = rgb2gray(imread('buddy.jpg'));

VMax = size(I, 1);

HMax = size(I, 2);

VMin = 1;

HMin = 1;

m = VMin;

n = HMin;

O = zeros(1, VMax\*HMax);

# Algorithm for Zig-Zag pattern

k = 1;

while ((m <= VMax) && (n <= HMax))

if (mod(n + m, 2) == 0) #going up

if (m == VMin)

O(k) = I(m, n); #if encountered the 1st line

if (n == HMax)

m = m + 1;

else

n = n + 1;

end

k = k + 1;

elseif ((n == HMax) && (m < VMax)) #if encountered the last column

O(k) = I(m, n);

m = m + 1;

k = k + 1;

elseif ((m > VMin) && (n < HMax)) #rest of the cases

O(k) = I(m, n);

m = m - 1;

n = n + 1;

k = k + 1;

end

else #going down

if ((m == VMax) && (n <= HMax)) #if encountered the last line

O(k) = I(m, n);

n = n + 1;

k = k + 1;

elseif (n == HMin) #if encountered the first column

O(k) = I(m, n);

if (m == VMax)

n = n + 1;

else

m = m + 1;

end

k = k + 1;

elseif ((m < VMax) && (n > HMin)) #rest of the cases

O(k) = I(m, n);

m = m + 1;

n = n - 1;

k = k + 1;

end

end

if ((m == VMax) && (n == HMax)) #if encountered bottom right element

O(k) = I(m, n);

break

end

end

**LAB 12**

**Aim :-** DCT based Compression

**Code :-**

# BT17ECE001

# DCT based Compression

clc; clear all; close all;

I=double((imread('buddy.jpg'))); #Importing the Image

[x1,y1,d]=size(I); #Finding the size of an Image

a=min(x1,y1);

if(d>2)

display("Image is coloured ")

else

display("Image is grayscale ");

end

# First I select the threshold value for DCT coefficients

# For coloured Image I used threshold values of 4,40 and 400

# Then For grayscale Images I used Threshold value of 20,70 and 90

p = "Give the threshold value? ";

Threshold = input(p);

# For coloured images

if(d==3)

# Resizing the Image to make it square

I\_sqr=(imresize(I,[a a]));

# Calculating the size of a square Image

[m2,n2]=size(I\_sqr);

# Calculation of the DCT basis matrix

for m=1:m2

for n=1:m2

if(m==1)

a(m,n)=sqrt(1/n2)\*cos(((2\*n-1)\*(m-1)\*pi)/(2\*n2));

else

a(m,n)=sqrt(2/n2)\*cos(((2\*n-1)\*(m-1)\*pi)/(2\*n2));

end

end

end

# Calculating the DCT coefficents

DCT\_R = a\*I\_sqr(:,:,1)\*a';

DCT\_G = a\*I\_sqr(:,:,2)\*a';

DCT\_B = a\*I\_sqr(:,:,3)\*a';

# Truncating the DCT coefficients

DCT\_R(abs(DCT\_R)<Threshold)=0;

DCT\_G(abs(DCT\_G)<Threshold)=0;

DCT\_B(abs(DCT\_B)<Threshold)=0;

DCT(:,:,1)=DCT\_R;

DCT(:,:,2)=DCT\_G;

DCT(:,:,3)=DCT\_B;

# Reconstruction of the compressed Image

I\_Compress(:,:,1)=a'\*DCT\_R\*a;

I\_Compress(:,:,2)=a'\*DCT\_G\*a;

I\_Compress(:,:,3)=a'\*DCT\_B\*a;

imwrite(uint8(I\_Compress),"Compressed\_I\_Coloured.jpeg");

# Compression ratio

d\_origin=imfinfo('tulip.jpeg');

size=d\_origin.FileSize;

d\_comp=imfinfo('Compressed\_I\_Coloured.jpeg');

size1=d\_comp.FileSize;

Compression\_ratio=floor(size/size1);

# Plotting

imshow(uint8(I\_sqr)),title("Original Image"); #showing Original Image

figure;

imshow(uint8(I\_Compress)),title("Compressed Image"); #showing Compressed Image

# Putting the images

imwrite(uint8(I\_sqr),"original\_image\_colored.jpeg");

imwrite(uint8(I\_Compress),"Compressed\_I\_Coloured.jpeg");

else

# Computing the size

[x1,y1]=size(I);

a=min(x1,y1);

# Resizing the Image to make it square

I\_sqr=(imresize(I,[a a]));

# Calculating the size of a square Image

[m2,n2]=size(I\_sqr);

# Calculation of the DCT basis matrix

for m=1:m2

for n=1:n2

if(m==1)

a(m,n)=sqrt(1/n2)\*cos(((2\*n-1)\*(m-1)\*pi)/(2\*n2));

else

a(m,n)=sqrt(2/n2)\*cos(((2\*n-1)\*(m-1)\*pi)/(2\*n2));

end

end

end

# Calculate the DCT coefficents for the Image

DCT=a\*I\_sqr\*a';

# Truncating the DCT coefficients to achieve compression

DCT(abs(DCT)<Threshold)=0;

# Reconstruction of the compressed Image

Compressed\_image=a'\*DCT\*a;

# Creating a compressed Image

imwrite(uint8(Compressed\_image),'Compressed\_image\_grayscale.jpeg')

# Compression ratio

d\_origin=imfinfo('cameraman.bmp');

size=d\_origin.FileSize;

d\_comp=imfinfo('Compressed\_image\_grayscale.jpeg');

size1=d\_comp.FileSize;

Compression\_ratio=floor(size/size1);

# Plotting

colormap(gray);imagesc(I\_sqr),title('Original I');

figure;

colormap(gray);imagesc(Compressed\_image),title('Compressed I');

# Putting the images

imwrite(uint8(I\_sqr),'original\_image\_grayscale.jpeg');

imwrite(uint8(Compressed\_image),'Compressed\_image\_grayscale.jpeg');

end

**LAB 13**

**Aim :-** Run Length Encoding

**Code :-**

# BT17ECE001

#Run Length Encoding

clc;clear all;close all;

# Generation of 100 random bits

Bits = round(rand(100,1));

# Finding the count of repeated bits

# Initialisation of Variables

PreBit = Bits(1);

CurBit = PreBit;

C = 0;

Sym = []; #Notes the Symbol that is repeated

C = []; #Notes the corresponding count

for i = 1:1:numel(Bits)

CurBit = Bits(i); #Get the current Bit

if(PreBit == CurBit) #Check if it same as previous bit

C = C + 1;

else

Sym = [Sym,PreBit]; #If not matching, then save last counted bit count

C = [C,C];

C = 1; #Reinitialise the Counter

end

PreBit = CurBit;

end

Sym = [Sym,PreBit]; #Saving the details of last bit

C = [C,C];

# Finding the Run Length Code

Bi\_C = dec2bin(C); #Getting the binary equivalent

RLE\_Code = '';

for i = 1:1:numel(Sym)

RLE\_Code = strcat(RLE\_Code,num2str(Sym(i)),Bi\_C(i,:));

# RLE\_Code = Sym + C in Binary

end

# Finding if the Encoding is +ve or -ve and find Compresion Ratio

if(length(Bits) > length(RLE\_Code))

disp("Positive RLE");

CR = (length(RLE\_Code)/length(Bits)); #Compression Ratio

disp(CR);

else

disp("Negative RLE");

end

**LAB 14**

**Aim :-** Toboggan Contrast Enhancement

**Code :-**

# BT17ECE001

# Toboggan Contrast Enhancement

clc; close all; clear all;

G = [12 14 23 13; 14 11 21 18; 21 24 23 13; 12 21 20 10]; # Image input and G operator

I = [4 4 3 2; 3 7 6 1; 2 7 6 2; 0 1 0 2];

M1 = zeros(4,4);

# Algorithm of Toboggan Contrast Enhancement

for m = 1:4

for n = 1:4

min = G(m,n);

i = m;

j = n;

if(m+1 <= 4 && G(m+1,n) < min)

min = G(m+1,n);

i = m+1;

j = n;

end

if(m-1 >= 1 && G(m-1,n) < min)

min = G(m-1,n);

i = m-1;

j = n;

end

if(n-1 >= 1 && G(m,n-1) < min)

min = G(m,n-1);

j = n - 1;

i = m;

end

if(n+1 <= 4 && G(m,n+1) < min)

min = G(m,n+1);

j = n + 1;

i = m;

end

if(n+1 <= 4 && m+1 <= 4 && G(m+1,n+1) < min)

min = G(m+1,n+1);

j = n+1;

i = m+1;

end

if(n-1>= 1 && m+1 <= 4 && G(m+1,n-1) <min)

min = G(m+1,n-1);

j = n-1;

i = m+1;

end

if(n + 1 <= 4 && m - 1 >= 1 && G(m-1,n+1) < min)

min = G(m-1,n+1);

j = n + 1;

i = m - 1;

end

M1(m,n) = I(i,j);

end

end

**LAB 15**

**Aim :-** Delta Modulation

**Code :-**

# BT17ECE001

# Delta Modulation

function [t m]=Delta\_Modulation(w, D)

#D=amplitude of signal

#t=output binary sequence

#w=step size

#Defining the Variables

t=0:2\*pi/100:2\*pi;

z=D\*sin(t);

plot(z)

hold on

t=[0];

h=0;

for j=1:length(z)-1

if h(j)<=z(j)

d=1;

h(j+1)=h(j)+w;

else

d=0;

h(j+1)=h(j)-w;

end

t=[t d];

end

stairs(h);

hold off;

m=sum((z-h).^2)/length(z); #Delta Modulation

end