

Indian Institute of Information Technology, Nagpur

Digital Image Processing Lab Report

Submitted To : Dr. Tapan Kumar Jain HOD OF ECE

Electronics and Communication Engineering Dept.

Submitted By : Sasidhar pechetti(BT18ECE008)

Semester VI

Electronics and Communication Engineering Dept.

|  |  |
| --- | --- |
| 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 |
|  |  |

**GITHUB LINK:-** [**https://github.com/sasi06/DIP-codes**](https://github.com/sasi06/DIP-codes)

**LAB - 1**

**AIM:-** To create a Ramp Image and do operations on it.

**SOFTWARE:-** Octave

# CODE:-

## ##Creating a Ramp image

##sasidhar

##BT18ECE008

pkg load image m=[0:9];

##Creating a matrix 0 to 9 Z=ones(1,10);

in=m'\*Z;

##mutiply transpose of m with row matrix having

10 times intensity figure(1) imshow(in,[])

##we first read the image whose operation is to

performed z=imread('wp2465273.JPG'); figure(2)

image(z);

title('The Orignal Image');

#conversion of image to rgb to gray J =rgb2gray(z);

figure(3)

imshow(J);

title('RGB TO Grey Image');

##convert image to its matrix form and rotate image

J1=J';

figure(4) imshow(J1);

##getting the pixel value pixel\_value=z(2,2);

sprintf('The value of the pixel is

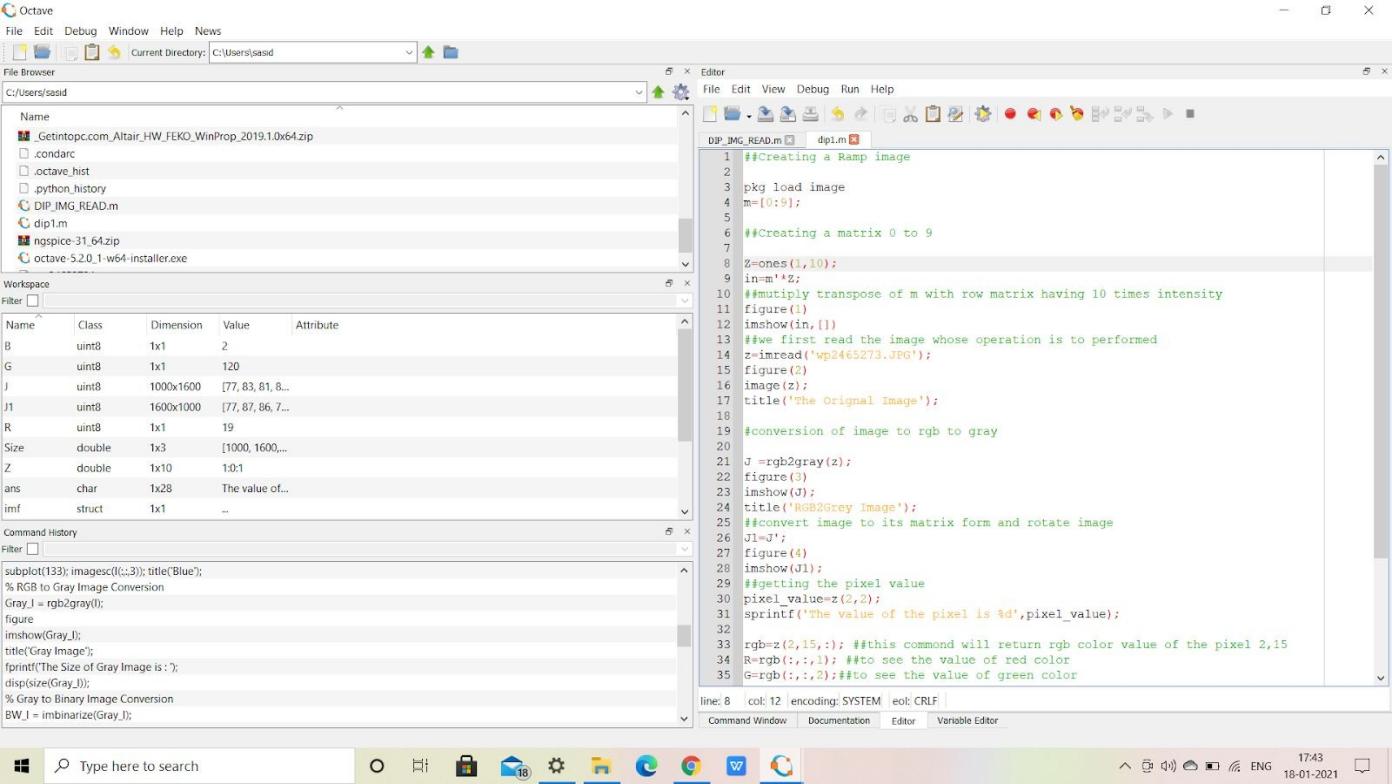
%d',pixel\_value);

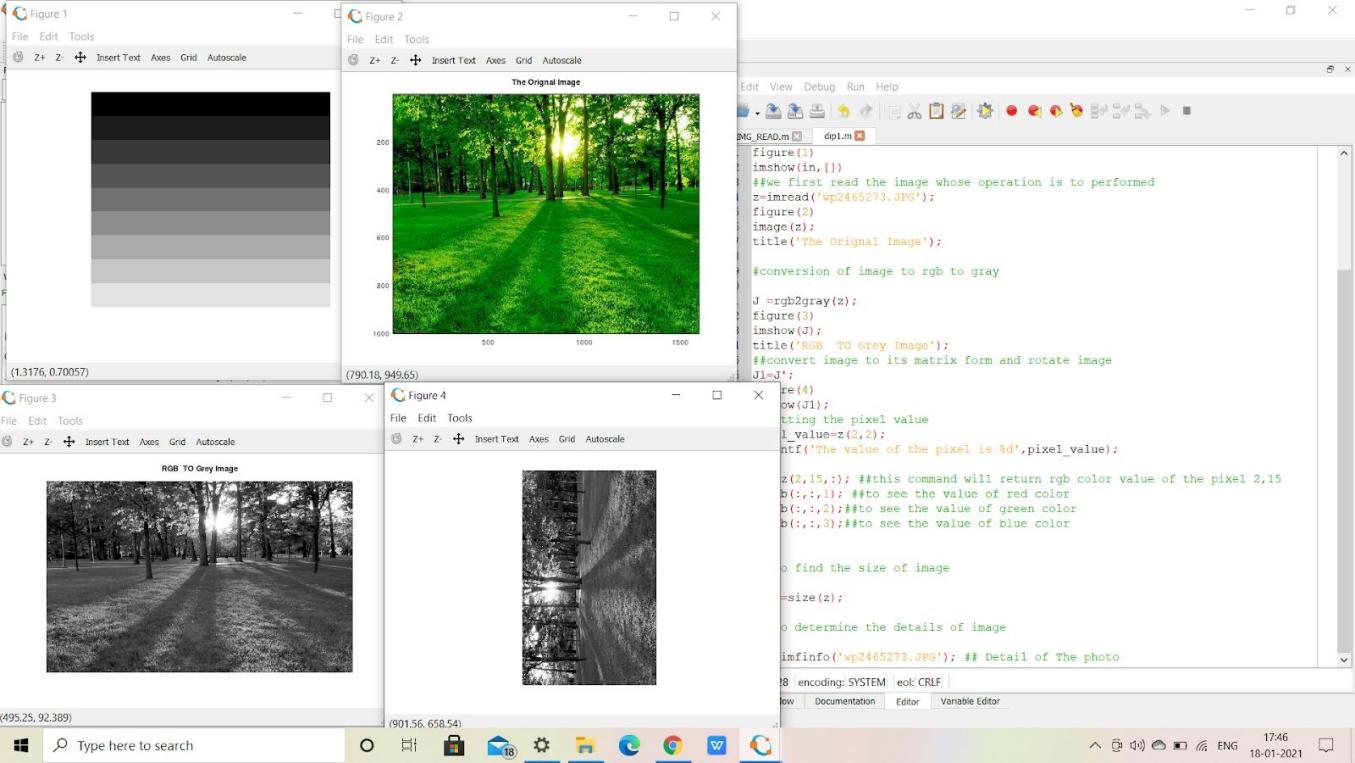
rgb=z(2,15,:); ##this command will return rgb color value of the pixel 2,15

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

## To find the size of image Size=size(z);

## To determine the details of image imf=imfinfo('wp2465273.JPG'); ## Detail of The photo



output:-

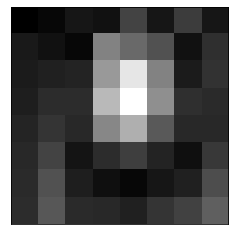
### LAB 2 HISTOGRAM EQUALIZATION

**SOFTWARE:- PYTHON(JYPITER,COLAB) CODE:-**

|  |  |
| --- | --- |
| ● | import numpy as np |
|  | h=np.array([[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],  ])h |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **O/P-** | **array([[** | **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]])** |

#### from matplotlib import pyplot as plt plt.imshow(h, cmap = 'gray') plt.xticks([]), plt.yticks([]) plt.show()

**O/P- **

* **unique\_elements, counts\_elements = np.unique(h, return\_counts=True)**
* **for val,count in zip(unique\_elements, counts\_elements):**
* **print(val,count) O/P-52 1**

|  |  |  |
| --- | --- | --- |
| **55** | **3** |  |
| **58** | **2** |
| **59** | **3** |
| **60** | **1** |
| **61** | **4** |
| **62** | **1** |
| **63** | **2** |
| **64** | **2** |
| **65** | **3** |
| **66** | **2** |
| **67** | **1** |
| **68** | **5** |
| **69** | **3** |
| **70** | **4** |
| **71** | **2** |
| **72** | **1** |
| **73** | **2** |
| **75** | **1** |
| **76** | **1** |
| **77** | **1** |
| **78** | **1** |
| **79** | **2** |
| **83** | **1** |
| **85** | **2** |
| **87** | **1** |
| **88** | **1** |
| **90** | **1** |
| **94** | **1** |
| **104** |  | **2** |
| **106** |  | **1** |
| **109** |  | **1** |
| **113** |  | **1** |
| **122** |  | **1** |
| **126** |  | **1** |
| **144** |  | **1** |
| **154** |  | **1** |

* **hist=np.asarray((unique\_elements, counts\_elements)) #converting element,count into array**

**print(hist) print(hist[0][:])**

**print(hist[1][:])**

**o/p:-[[ 52 55 58 59 60 61 62 63 64 65 66 67 68 69 70 71**

**72 73 75 76 77 78 79 83 85 87 88 90 94 104 106 109 113 122**

**126 144 154][ 1 3 2 3 1 4 1 2 2 3 2 1 5 3**

**4 2 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1**

**1 1 1 1]] [ 52 55 58 59 60 61 62 63 64 65 66 67 68 69**

**70 71 72 73 75 76 77 78 79 83 85 87 88 90 94 104 106 109**

**113 122 126 144 154]**

**[1 3 2 3 1 4 1 2 2 3 2 1 5 3 4 2 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1 1 1**

**1 1]**

* **cdf=hist[1][:].cumsum() print(cdf) print(cdf.min(),len(cdf))**

**o/p-**

**[ 1 4 6 9 10 14 15 17 19 22 24 25 30 33 37 39 40 42 43 44 45 46 48**

**49 51 52 53 54 55 57 58 59 60 61 62 63 64]1 37**

* **out=[]**

**for i in range(len(cdf)): eq=round(((cdf[i]-1)/63)\*255) out.append(eq) h\_eq=np.uint8(np.asarray(out)) h\_eq**

**o/p-**

**array([ 0, 12, 20, 32, 36, 53, 57, 65, 73, 85, 93, 97, 117,**

**130, 146, 154, 158, 166, 170, 174, 178, 182, 190, 194, 202, 206,**

**210, 215, 219, 227, 231, 235, 239, 243, 247, 251, 255],**

**dtype=uint8)**

**hist[0][:] o/p-**

**array([ 52, 55, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,**

**69, 70, 71, 72, 73, 75, 76, 77, 78, 79, 83, 85, 87,**

**88, 90, 94, 104, 106, 109, 113, 122, 126, 144, 154])**

* **kv = {hist[0][i]: h\_eq[i] for i in range(len(h\_eq))} print(kv)**

**print() print(kv.get(55),type(kv.get(55))) print()**

**print(kv[55])**

**o/p-{52: 0, 55: 12, 58: 20, 59: 32, 60: 36, 61: 53, 62: 57, 63: 65, 64:**

**73, 65: 85, 66: 93, 67: 97, 68: 117, 69: 130, 70: 146, 71: 154, 72:**

**158, 73: 166, 75: 170, 76: 174, 77: 178, 78: 182, 79: 190, 83: 194, 85:**

**202, 87: 206, 88: 210, 90: 215, 94: 219, 104: 227, 106: 231, 109: 235,**

**113: 239, 122: 243, 126: 247, 144: 251, 154: 255}**

**12 <class 'numpy.uint8'> 12**

* **g=h print(g[1][1]) print(g)**

**o/p-**

**59**

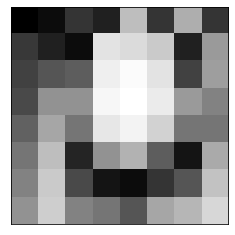
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **[[** | **52** | **55** | **61** | **59** | **79** | **61** | **76** | **61]** |
| **[** | **62** | **59** | **55** | **104** | **94** | **85** | **59** | **71]** |
| **[** | **63** | **65** | **66** | **113** | **144** | **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]]** |

#### for i in range(8): for j in range(8): b=kv.get(g[i][j]) g[i][j]=b #b=kv.get(g[i,j]) #g[i,j]=b

**print(g)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **o/p-[[** | **0** | **12** | **53** | **32** | **190** | **53** | **174** | **53]** |
| **[ 57** | **32** | **12** | **227** | **219** | **202** | **32** | **154]** |  |
| **[ 65** | **85** | **93** | **239** | **251** | **227** | **65** | **158]** |  |
| **[ 73** | **146** | **146** | **247** | **255** | **235** | **154** | **130]** |  |
| **[ 97** | **166** | **117** | **231** | **243** | **210** | **117** | **117]** |  |
| **[117** | **190** | **36** | **146** | **178** | **93** | **20** | **170]** |  |
| **[130** | **202** | **73** | **20** | **12** | **53** | **85** | **194]** |  |
| **[146** | **206** | **130** | **117** | **85** | **166** | **182** | **215]]** |  |

* **plt.imshow(g, cmap = 'gray') plt.xticks([]), plt.yticks([]) plt.show()**

**o/p-**

**Bit Plane Slicing**

**LAB -3**

* #Import required library import cv2

import numpy as np

from matplotlib import pyplot as plt import urllib

import cv2

img = cv2.imread('lena.jpg')

im = cv2.imread('lena.jpg',cv2.IMREAD\_GRAYSCALE) #cv2.imshow('image',im)

plt.imshow(img, cmap = 'gray')

plt.show() img

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) print(gray)

print(gray.shape) print(gray[1][1]) img

def convertToBinary(my\_dec): l=[]

while my\_dec>0: #print(my\_dec) rem=my\_dec%2 my\_dec=my\_dec//2 #print(my\_dec) l.append(rem) #print(l)

l.reverse() new=np.zeros(8 -len(l)) ow=np.append(new,l) return ow

k=convertToBinary(2) print(k) k=convertToBinary(200) print(k,type(k)) out=[]

for i in range(gray.shape[0]):

for j in range(gray.shape[1]): #print(img[i][j]) outp=convertToBinary(gray[i][j]) out.append(outp)

print(outp,outp[1]) #out jv=np.asarray(out) print(jv,jv.shape)

print(jv[0][1],jv[1][1])

from IPython.display import Image Image(url='https://i2.wp.com/theailearner.com/wp-content/uploads/2019/0 1/bit-plane-slicing.png?w=662&ssl=1')

one = np.array([int(i[0]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1]) two= np.array([int(i[1]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1]) three = np.array([int(i[2]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1]) four = np.array([int(i[3]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1]) five = np.array([int(i[4]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1])

six = np.array([int(i[5]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1]) seven= np.array([int(i[6]) for i in jv],dtype = np.uint8).reshape(img.shape[0],img.shape[1])

eight = np.array([int(i[7]) for i in jv],dtype = np.uint8)

.reshape(img.shape[0],img.shape[1]) print(one,two,three) plt.imshow(one, cmap = 'gray') plt.show()

plt.imshow(one, cmap = 'gray') plt.title("1st plane") plt.show()

plt.imshow(two, cmap = 'gray') plt.title("2nd plane") plt.show()

plt.imshow(three, cmap = 'gray') plt.title("3rd plane") plt.show()

plt.imshow(four, cmap = 'gray') plt.title("4th plane") plt.show()

plt.imshow(five, cmap = 'gray') plt.title("5th plane") plt.show()

plt.imshow(six, cmap = 'gray') plt.title("6th plane") plt.show()

plt.imshow(seven, cmap = 'gray') plt.title("7th plane") plt.show()

plt.imshow(eight, cmap = 'gray') plt.title("8th plane") plt.show()

**OUTPUT AND SOURCES PHOTOS:-**



|  |  |  |
| --- | --- | --- |
| **array([[[118,** | **134,** | **230],** |
| **[118,** | **134,** | **230],** |
| **[117,**  **...,** | **133,** | **229],** |
| **[122,** | **128,** | **223],** |
| **[141,** | **149,** | **242],** |
| **[111,** | **118,** | **215]],** |
| **[[117,** | **133,** | **229],** |
| **[117,** | **133,** | **229],** |
| **[117,**  **...,** | **133,** | **229],** |
| **[138,** | **140,** | **234],** |
| **[154,** | **157,** | **248],** |
| **[120,** | **120,** | **214]],** |
| **[[117,** | **133,** | **229],** |
| **[117,** | **133,** | **229],** |
| **[115,** | **133,** | **228],** |
| **...,**  **[114,** | **109,** | **200],** |
| **[113,** | **104,** | **194],** |
| **[ 79,** | **63,** | **151]],** |
| **...,** |  |  |
| **[[ 68,** | **31,** | **87],** |
| **[ 63,** | **27,** | **87],** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **[ 60,**  **...,** | | **24,** | **90],** | |
| **[ 74,** | | **51,** | **126],** | |
| **[ 77,** | | **58,** | **139],** | |
| **[ 78,** | | **61,** | **152]],** | |
| **[[ 66,** | | **31,** | **87],** | |
| **[ 62,** | | **26,** | **86],** | |
| **[ 59,** | | **23,** | **89],** | |
| **...,** | |  |  | |
| **[** | **78,** | **57,** | **136],** | |
| **[** | **85,** | **66,** | **151],** | |
| **[** | **77,** | **66,** | **159]],** | |
| **[[** | **52,** | **20,** | **77],** | |
| **[** | **67,** | **35,** | **94],** | |
| **[** | **61,** | **25,** | **91],** | |
| **...,** | | | | |
| **[** | **87,** | **68,** | **155],** |  |
| **[** | **89,** | **73,** | **167],** |  |
| **[** | **87,** | **73,** | **178]]],** | **dtype=uint8)** |

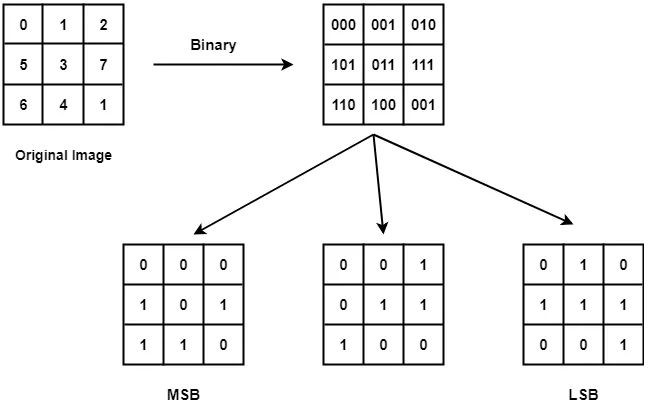
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **[[161** | **161** | **160** | **...** | **156** | **176** | **146]** |
| **[160** | **160** | **160** | **...** | **168** | **184** | **148]** |
| **[160**  **...** | **160** | **159** | **...** | **137** | **132** | **91]** |
| **[ 52** | **49** | **48** | **...** | **76** | **84** | **90]** |
| **[ 52** | **48** | **47** | **...** | **83** | **94** | **95]** |
| **[ 41** | **56** | **49** | **...** | **96** | **103** | **106]]** |
| **(225,** | **225)** |  |  |  |  |  |
| **160** |  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **[0.** | **0.** | **0.** | **0.** | **0.** | **0.** | **1.** | **0.]** |  |
| **[1.** | **1.** | **0.** | **0.** | **1.** | **0.** | **0.** | **0.]** | **<class 'numpy.ndarray'>** |
| **[0.** | **1.** | **1.** | **0.** | **1.** | **0.** | **1.** | **0.]** | **1.0** |

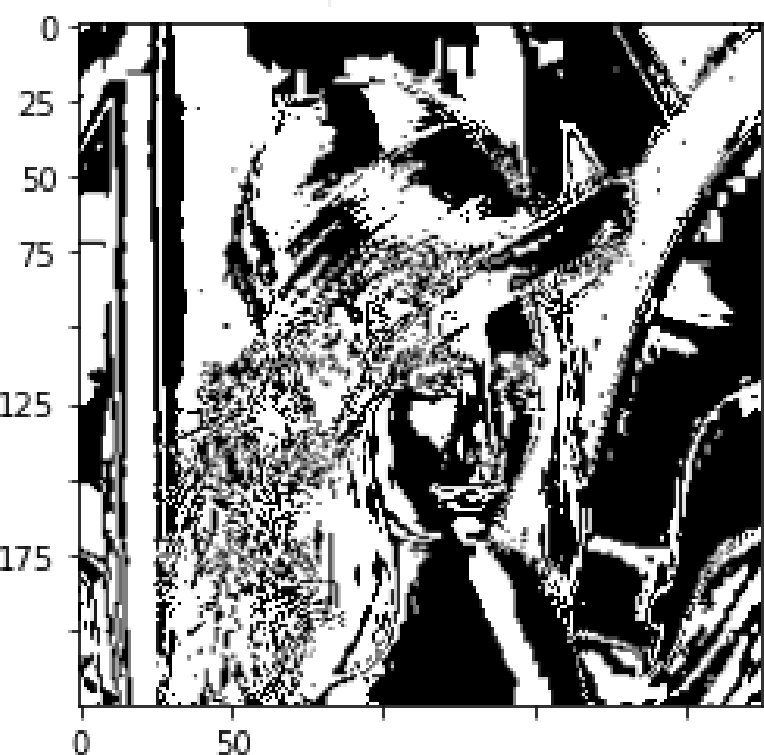
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **[[1.** | **0.** | **1.** | **...** | **0.** | **0.** | **1.]** |  |
| **[1.** | **0.** | **1.** | **...** | **0.** | **0.** | **1.]** |  |
| **[1.**  **...** | **0.** | **1.** | **...** | **0.** | **0.** | **0.]** |  |
| **[0.** | **1.** | **1.** | **...** | **0.** | **0.** | **0.]** |  |
| **[0.** | **1.** | **1.** | **...** | **1.** | **1.** | **1.]** |  |
| **[0.** | **1.** | **1.** | **...** | **0.** | **1.** | **0.]]** | **(50625, 8)** |

#### 0.0 0.0

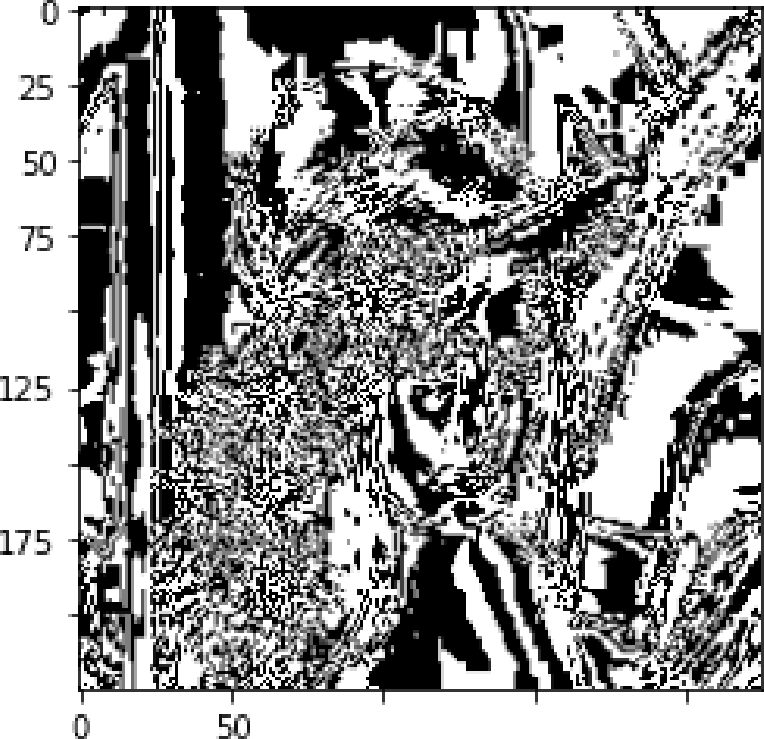


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **[[1** | **1** | **1** | **...** | **1** | **1** | **1]** |  |
| **[1** | **1** | **1** | **...** | **1** | **1** | **1]** |
| **[1**  **...** | **1** | **1** | **...** | **1** | **1** | **0]** |
| **[0** | **0** | **0** | **...** | **0** | **0** | **0]** |
| **[0** | **0** | **0** | **...** | **0** | **0** | **0]** |
| **[0** | **0** | **0** | **...** | **0** | **0** | **0]]** | **[[0 0 0 ... 0 0 0]** |
| **[0** | **0** | **0** | **...** | **0** | **0** | **0]** |  |
| **[0**  **...** | **0** | **0** | **...** | **0** | **0** | **1]** |  |
| **[0** | **0** | **0** | **...** | **1** | **1** | **1]** |  |
| **[0** | **0** | **0** | **...** | **1** | **1** | **1]** |  |
| **[0** | **0** | **0** | **...** | **1** | **1** | **1]]** | **[[1 1 1 ... 0 1 0]** |
| **[1** | **1** | **1** | **...** | **1** | **1** | **0]** |  |
| **[1**  **...** | **1** | **0** | **...** | **0** | **0** | **0]** |  |
| **[1** | **1** | **1** | **...** | **0** | **0** | **0]** |  |
| **[1** | **1** | **1** | **...** | **0** | **0** | **0]** |  |
| **[1** | **1** | **1** | **...** | **1** | **1** | **1]]** |  |



3rd plane 4th plane

100



100

190

700

100

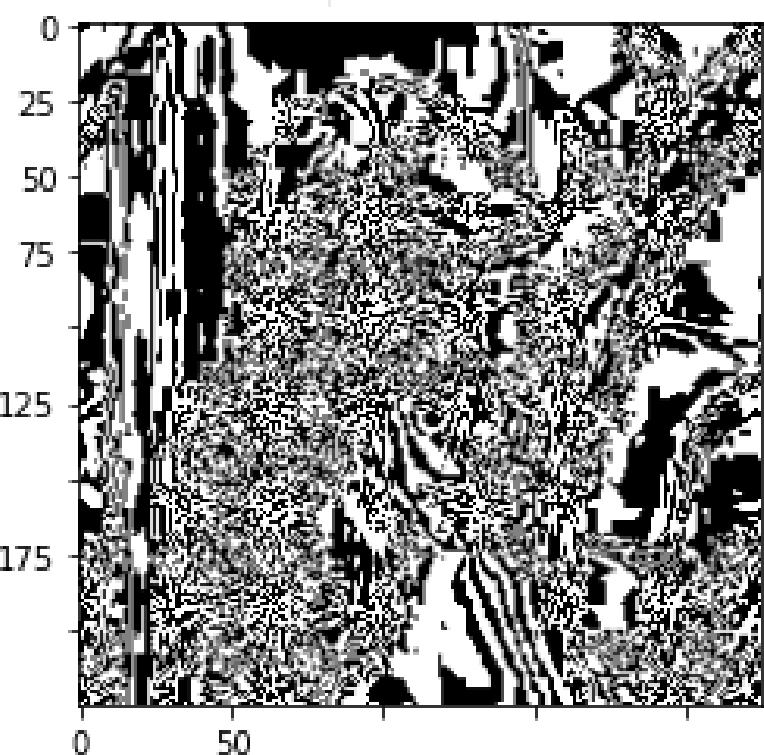
15g

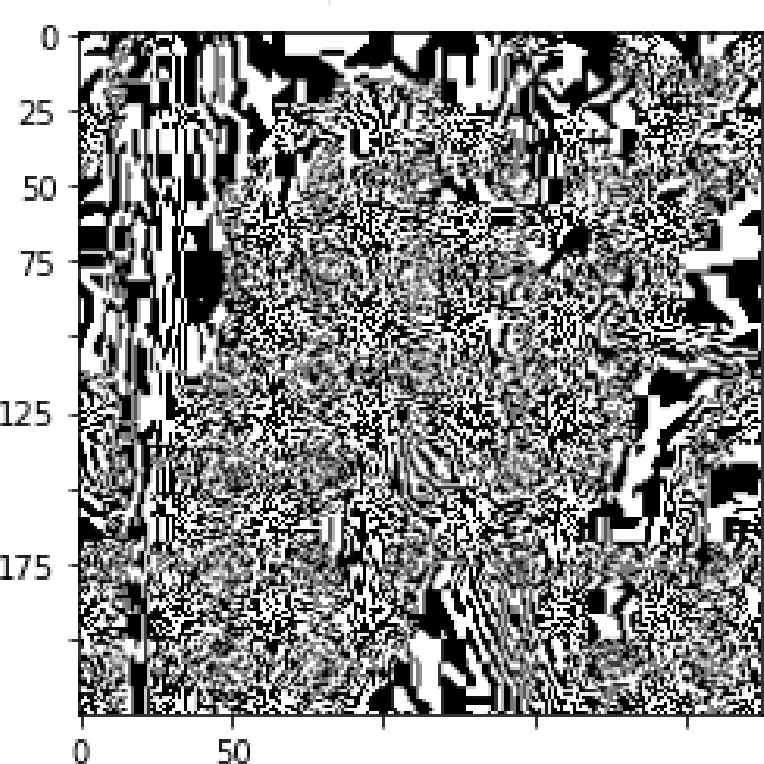
200

190

700

100 15g 200

Sth plane



6th plane

100

190

700

100

15g

200

100

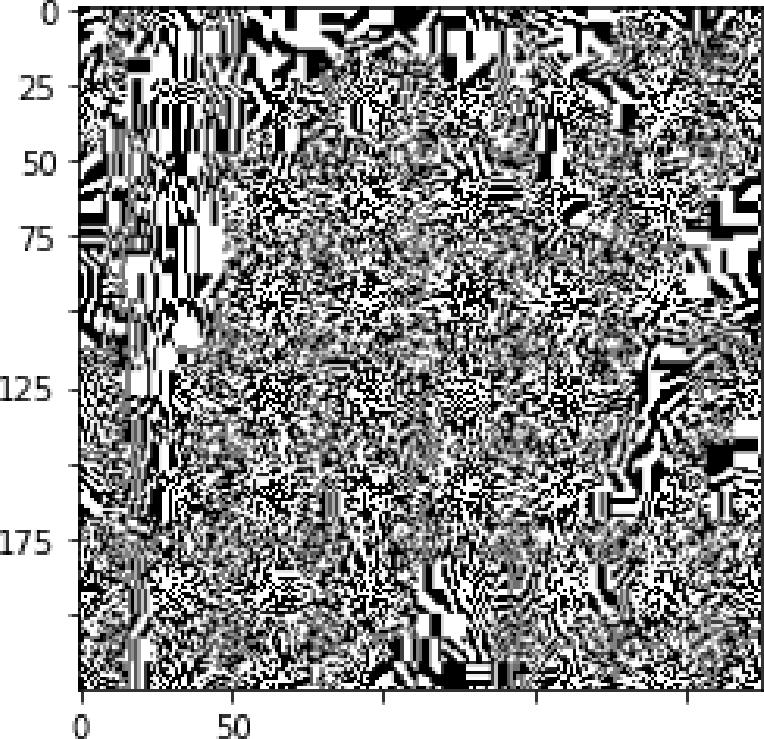
190

700

100 15g 200

7th plane

8th pla ne



100

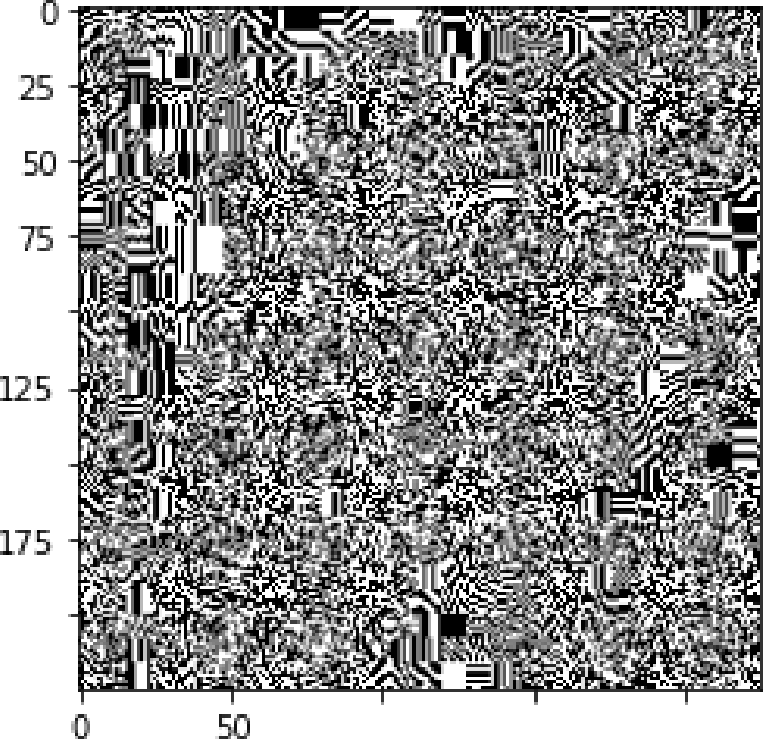
190

700

100

15g

200



100

150

700

100

150

700

**LAB 3**

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

**CODE:-**

##sasidhar

##BT18ECE008

# Applying different kinds of Filters.

clc; clear all; close all;

I = imread('bhanu.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

**LAB 4**

**Aim :-** Histogram Specification

**Code :-**

# sasidhar

# BT18ECE008

# 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 :-**

# SASIDHAR

# BT18ECE008

# Edge Detection using DWT2

clc; clear all; close all;

I = imread('bhanu.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 :-**

# SASIDHAR

# BT18ECE008

# Edge Detection using Sobel

clc; clear all; close all;

I = imread('bhanu.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 :-**

# SASIDHAR

# BT18ECE008

# Histogram Stretching

clc;clear;close all;

SMin = 10;

SMax = 255;

I = imread("lena.png");

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");

**LAB 8**

**Aim :-** Near Neighbourhood Algorithm

**Code :-**

# SASIDHAR

# BT18ECE008

# Near Neighbourhood Algorithm

clc;clear all;close all;

Image = imread("lena.tif"); #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");

**LAB 9**

**Aim :-** Separating RGB Components

**Code :-**

# SASIDHAR

# BT18ECE008

# Separating RGB Components

clc;clear all;close all;

I = imread("lena.tif"); #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

**LAB 10**

**Aim :-** Watermarking

**Code :-**

# SASIDHAR

# BT18ECE008

# Watermarking

clc; clear all; close all;

I = rgb2gray(imread("bhanu.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

**LAB 11**

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

**Code :-**

# SASIDHAR

# BT18ECE008

# Zig\_Zag\_DCT

clc; clear all; close all;

# Initialise the variables

I = rgb2gray(imread('bhanu.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 :-**

#SASIDHAR

# BT18ECE008

# DCT based Compression

clc; clear all; close all;

I=double((imread('bhanu.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 :-**

# SASIDHAR

# BT18ECE008

#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 :-**

# SASIDHAR

# BT18ECE008

# 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 :-**

# SASIDHAR

# BT18ECE008

# 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

**REFERENCS:-**

**1).**[**https://www.google.com/search?q=lena+internet&sa=X**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM)[**&biw=1536&bih=666&sxsrf=ALeKk03C0-UE\_7kiTRERCFeIuRdDQ**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM)[**8WQ9g:1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZ**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM)[**wpWtivzR7VM%252C6DW5TB7AAXFPyM%252C\_&vet=1&usg=AI4\_-k**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM)[**SMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM)[**7wzgGHW17DqsQ9QF6BAgREAE#imgrc=3qZwpWtivzR7VM**](https://www.google.com/search?q=lena%2Binternet&sa=X&biw=1536&bih=666&sxsrf=ALeKk03C0-UE_7kiTRERCFeIuRdDQ8WQ9g%3A1612892712414&tbm=isch&source=iu&ictx=1&fir=3qZwpWtivzR7VM%252C6DW5TB7AAXFPyM%252C_&vet=1&usg=AI4_-kSMLc5vF3-kHXVunkmI1QY0j-bccQ&ved=2ahUKEwiX-7vJrd3uAhU7wzgGHW17DqsQ9QF6BAgREAE&imgrc=3qZwpWtivzR7VM) **2).**[**https://colab.research.google.com/notebooks/intro.**](https://colab.research.google.com/notebooks/intro.ipynb)[**ipynb**](https://colab.research.google.com/notebooks/intro.ipynb)

**3).**[**https://i2.wp.com/theailearner.com/wp-content/uplo**](https://i2.wp.com/theailearner.com/wp-content/uploads/2019/01/bit-plane-slicing.png?w=662&ssl=1)[**ads/2019/01/bit-plane-slicing.png?w=662&ssl=1**](https://i2.wp.com/theailearner.com/wp-content/uploads/2019/01/bit-plane-slicing.png?w=662&ssl=1)