Indian Institute of Information Technology Bhagalpur

Computer Vision and Image Processing lab Report

(EC316)

Submitted by:

SNEH RANJAN	**2001047**	**ECE**
SAGAR SONI	**2001092**	**ECE**
PARV CHOUDHARY	**2001089**	**ECE**
SANI KUMAR	**2001125**	**ECE**

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Bhagalpur-83210, Bihar, INDIA

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that ***Mr. SNEH RANJAN(2001047), Mr. SAGAR SONI(2001092), Mr. PARV CHOUDHARY(2001089), Mr. SANI KUMAR(2001125)*** have satisfactorily completed the course in **Computer Vision and Image Processing *(EC316)*** during the academic year2022-2023.

Date: 20/04/2023

Place: IIIT Bhagalpur

Dr. Sandeep raj
and
Mr. Rajan Kumar

EXPERIMENT NO :- 1

AIM :- WRITE A OCTAVE PROGRAM TO DO THE FOLLOWING CHANGE IN AN IMAGE: COMPLEMENT OF IMAGE,FLIPPING UPSIDE DOWN AND LEFT-RIGHT,ROTATION BY 45 DEGREES.

APPARATUS REQUIRED :-

1. OCTAVE SOFTWARE

THEORY :-

IMREAD FUNCTION :-

A = imread(filename) reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.

IMCOMPLEMENT FUNCTION :-

compute image complement or negative. intuitively this corresponds to the intensity of bright and dark regions being reversed. the exact operation performed is dependent on the class of the image.inverts the values within the range of the data type. this is equivalent to **`bitcmp (a).`**

flipping an image upside down or left to right involves reversing the order of its rows or columns, respectively. for example, to flip an image upside down, the first row of pixels is swapped with the last row, the second row is swapped with the second-to-last row, and so on. to flip an image left to right, the first column of pixels is swapped with the last column, the second column is swapped with the second-to-last column, and so on. rotating an image by 45 degrees involves changing the position of its pixels according to a rotation matrix. the rotation matrix for a counterclockwise rotation by an angle θ

CODE :-

```
pkg load image
img=imread('sample.jpg');%image read
subplot(331);%assign position
imshow(img);%output image
title("inserted image / original image ");%assign name
c=bitcmp(img);%perform complement operation
subplot(332);%assign position
imshow(c);%output image
```

```
title("complemented image");%assign name
% Flip the image upside down
img_ud = flipud(img);
% Flip the image left to right
img_lr = fliplr(img);
% Rotate the image by 45 degrees
img_rot = imrotate(img, 45);
subplot(333);
imshow(img_rot);
title('Rotated Image');
subplot(334);
imshow(img_ud);
title('Upside Down Image');
subplot(335);
imshow(img_lr);
title('Left-Right Image');
```

OUTPUT:-

RESULT :-

up on performing the experiment in octave i have observed that the imread command is pre installed in the software itself and was performing the read operation given the path of the directory. and the imcomplement was able to invert the values of the image according using the bits of the image. Able to make the image mirror by vertical and horizontal and obtained the rotated image by 45 degree angle anticlockwise or counter clock wise direction.

PRECAUTIONS :-

1. clear all the previously runed codes using clc command.
2. use the commands accordingly to the given syntax.

CONCLUSION :-

by performing the experiment i was able to learn how to take an image as input and display it as output using the octave commands **imread(a)** and have performed the image complement using the command **bitcmp(a)** and learnt how it works. And have performed image mirror by vertical and horizontal and obtained the rotated image by 45 degree angle anticlockwise or counter clock wise direction.

EXPERIMENT NO :- 2

AIM :-

Write a Octave program to perform following actions on Image:

1. RGB to HSV image HSV to Gray image
2. RGB to Gray image
3. RGB to Binary image
4. Change Contrast of original image (Very high & Very Low).

APPARATUS REQUIRED :-

1. OCTAVE SOFTWARE.

THEORY :-

RGB_MAP=RGB2HSV(RGB_MAP):-

octave supports **conversion from the rgb color system to the hsv color system and vice versa**. it is also possible to convert from a color rgb image to a grayscale image. hsv_map = rgb2hsv (rgb_map) hsv_img = rgb2hsv (rgb_img).

***GRAY_MAP* = RGB2GRAY *(RGB_MAP)* :-**

transform an image or colormap from red-green-blue (rgb) color space to a grayscale intensity image.

the input may be of class uint8, int8, uint16, int16, single, or double. the output is of the same class as the input.

implementation note: the grayscale intensity is calculated as

$$i = 0.298936*r + 0.587043*g + 0.114021*b$$

RGB TO BINARY :- (IM2BW);

convert image to binary, black and white, by threshold. the input image *img* can either be a grayscale or rgb image. in the later case, *img* is first converted to grayscale with rgb2gray. input can also be an indexed image *x* in which case the colormap *cmap* needs to be specified the value of *threshold* should be in the range [0,1] independently of the class of *img*. values from other classes can be converted to the correct value with im2double: bw = im2bw (img_of_class_uint8, im2double (thresh_of_uint8_class)); for an automatic threshold value, consider using graythresh. the argument *method* is a string that specifies a valid algorithm available in graythresh. the following are equivalent: bw = im2bw (img, "moments"); bw = im2bw (img, graythresh (img, "moments"));

Contrast :-

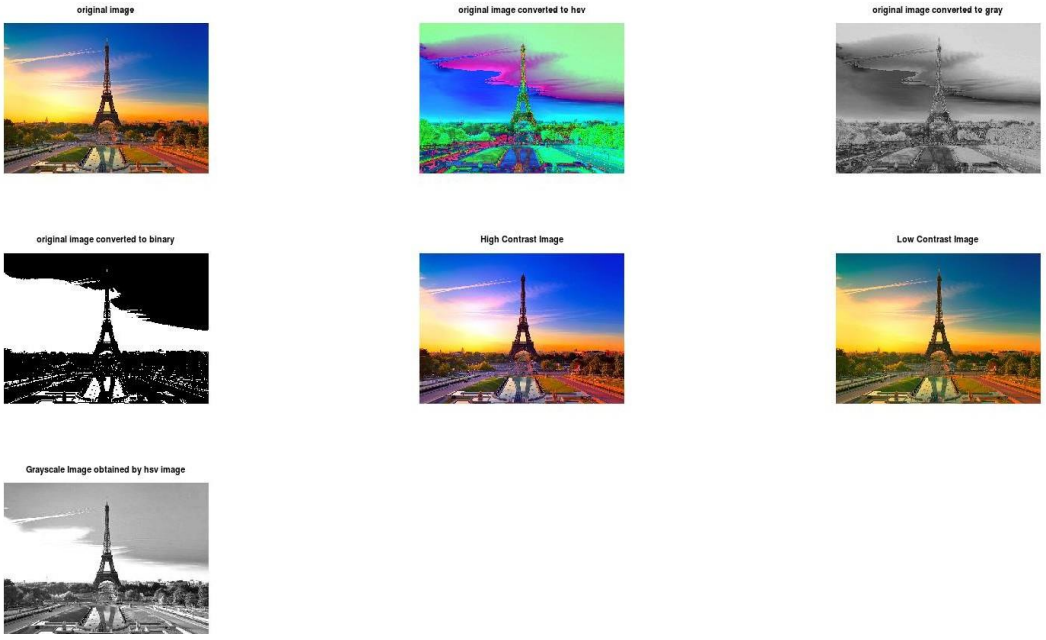
It is the difference in luminance or color that makes an object distinguishable. In visual perception of the real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view . Changing the contrast of an image involves changing the values of its pixels to make the dark areas darker and the light areas lighter. This can be done by using a mathematical formula to adjust the pixel values or by using a graphical tool to interactively adjust the contrast

CODE :-

```
pkg load image%load image
img=imread('sample.jpg');%read image
subplot(331);%assign position
imshow(img);%output image
title('original image');%assign name
b=rgb2hsv(img);%perform rgb to hsv operation
```

```
subplot(332);%assign position
imshow(b);%output image
title('original image converted to hsv');%assign name
c=rgb2gray(b);%perform rgb to Gray operation
subplot(333);%assign p[osition
imshow(c);%output image
title('original image converted to gray');%assign name
e=im2bw(img);%perform image to binary operation
subplot(334);%assign position
imshow(e);%output image
title('original image converted to binary');%assign name
% Change the contrast of the image to very high
img_high = imadjust(img, stretchlim(img), [0 1]);
% Change the contrast of the image to very low
img_low = imadjust(img, [0 1], stretchlim(img));
% Display the original and contrast-adjusted images
subplot(335);
imshow(img_high);
title('High Contrast Image');
subplot(336);
imshow(img_low);
title('Low Contrast Image');
% Convert the V channel of the HSV image to grayscale
img_gray = b(:,:,3);
subplot(337);
imshow(img_gray);
title('Grayscale Image obtained by hsv image');
```

OUTPUT:-



RESULT :-

upon performing the code i was able to understand that there are many inbuilt functions which are used to change the image from one type to another type such as **rgb2hsv ,rgb2gray,im2bw,hsvtogray,highcontast,lowcontrast of an iamge** etc.

PRECAUTIONS :-

1. make sure to load the image first and then we are able to use the inbuilt functions which are required.

CONCLUSION :-

by performing this experiment i was able to learn how the inbuilt functions work like :- **rgb2hsv,rgb2gray,im2bw,rgb2hsv,img(:,:,3) imadjust()→ for low and high contrast etc,** and have observed the output images accordingly.

EXPERIMENT NO :- 3

AIM :-

TO PERFORM SAMPLING UPON AN IMAGE IN OCTAVE .

APPARATUS REQUIRED :-

1. OCTAVE.

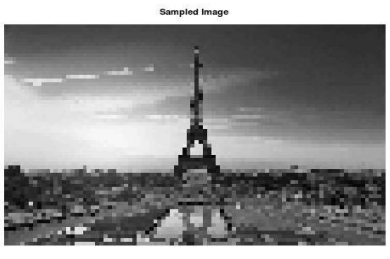
THOERY :-

the **octave**-forge **image** package provides functions for processing **images**. the package also provides functions for feature extraction, **image** statistics, spatial and geometric transformations, morphological operations, linear filtering, and much more. select category: analysis and statistics corr2 compute correlation coefficients of **images**. edge

CODE:-

```
clc;%clear command
clear all;%clear command
close all;%clear command
pkg load image;%load iamge
n=8;%assign value for a variable
i=imread('sample.jpg');%input image
subplot(2,2,1);%assign position
imshow(i);%output image
title('origianl iamge');%assign name
img = rgb2gray(imread('sample.jpg'));%perform rgb to gray operation
a=size(img);%get image size
w=a(2);%assign width of image
h=a(1);%assign height of an image
im=zeros(100);%assign extra pixesl im to the image
for i=1:n:h%run for loop
 for j=1:n:w%run for loop
     for k=0:n-1
         for l=0:n-1
             im(i+k,j+l)=img(i,j);

 end
 end
 end
end
subplot(2,2,2);%assign position
imshow(uint8(img));title('Original Image converted to gray');%outptut image
after opertaion
subplot(2,2,3);%assign position
imshow(uint8(im));title('Sampled Image');%output image and assig name to it
```

OUTPUT:-



RESULT :-

upon performing the experiment i was able to know that in order to make samples of an image we need to resize the image and then apply the necessary things like converting the original rgb image to Gray and then make samples .

PRECAUTIONS :-

1. make sure there is no error in the code
2. make sure to load the image using the pkg load image command

CONCLUSION :-

by performing this experiment i was able to know that how sampling happens in an image and have observed that the sampled image is some how blur when compared to the original image.

EXPERIMENT NO :- 4

AIM :-

WRITE A PROGRAM IN OCTAVE TO IMPLEMENT THE FOLLOWING MODIFICATION OF AN IMAGE:

1. SMOOTHING OR AVERAGE FILTER IN SPATIAL DOMAIN.
2. TO FILL THE REGION OF INTEREST FOR THE IMAGE.
3. TO DRAWING A LINE AND A RECTANGLE ON IMAGE.

APPARATUS REQUIRED :-

OCTAVE SOFTWARE.

THEORY :-

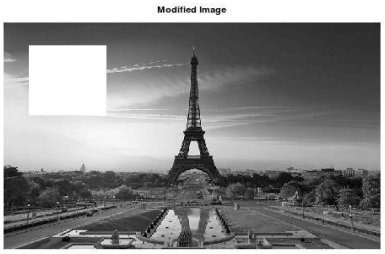
Smoothing or average filters are used for blurring and noise reduction in an image. Blurring is a pre-processing step for the removal of small details and noise reduction is accomplished by blurring 1. A smoothing filter works by replacing the value of each pixel in an image with the average of the pixel values in its neighbourhood. This can be done using a linear filter, such as a mean filter, or a non-linear filter, such as a median filter.defines a region of interest as a rectangle with its top-left corner at (50, 50) and its width and height equal to 150 pixels, and fills the region of interest with white color by setting the pixel values in the region to 255. creates a figure and displays the image using the imshow function, draws a line on the image using the line function with the specified x and y coordinates and sets its color to red and its width to 2 pixels, and draws a rectangle on the image using the rectangle function with the specified position, edge color, and line width. The hold on and hold off functions are used to allow multiple graphics objects to be drawn on the same image.

CODE :-

```
pkg load image
img=imread('sample.jpg');
a=img;
% Convert the image to grayscale
img = rgb2gray(img);
% Create a mean filter
h = fspecial('average', [5 5]);
% Apply the mean filter to the image
img_smooth = imfilter(img, h);
% Display the original and smoothed images
subplot(221);
imshow(a);
title('Original Image');
subplot(222);
imshow(img_smooth);
title('Smoothed Image');
% Define the region of interest
roi = [50 50 150 150];
% Fill the region of interest with a color
img(roi(2):roi(2)+roi(4), roi(1):roi(1)+roi(3), :) = 255;
% Display the modified images
subplot(223);
imshow(img);
title('Modified Image');
% Create a figure and display the image
subplot(224);
imshow(a);
title('line and rectangle image');
hold on;
```

```
% Draw a line on the image
x = [50 150];
y = [50 150];
line(x, y, 'Color', 'r', 'LineWidth', 2);
% Draw a rectangle on the image
x = 100;
y = 100;
w = 50;
h = 50;
rectangle('Position', [x y w h], 'EdgeColor', 'g', 'LineWidth', 2);
% Release the hold on the image
hold off;
```

OUTPUT:-



RESULT :-

A smoothing filter works by replacing the value of each pixel in an image with the average of the pixel values in its neighborhood. This can be done using a linear filter, such as a mean filter, or a non-linear filter, such as a median filter. a region of interest as a rectangle with its top-left corner at (50, 50) and its width and height equal to 150 pixels, and fills the region of interest with white color by setting the pixel values in the region to 255. Drawn an line at the top left corner and an rectangle as well .

PRECAUTIONS :-

1. make sure to use the command pkg load image

2. make sure that there are no erros in the code.

CONCLUSION :-

upon performing this experiment i was able to know how to perform the smoothing of an image and define an area and fill it with the something and was able to draw a line and an rectangle on an image by using the octave inbuilt commands.

EXPERIMENT NO :- 5

AIM :-

TO PERFORM SOME MORPHOLOGICAL OPERATIONS ON AN IMAGE SUCH AS :-

1. OPENING.
2. CLOSING.
3. EROSION.
4. DILATION.

APPARATUS REQUIRED :-

OCTAVE SOFTWARE.

THEORY :-

basically upon performing the opening and closing the images in octave we basically are choosing the strel function and assigning the shape to it and no of pixels to it to make the perfect shape.

EROSION :-

it is the morphological operation used to diminish the size of the foreground object. it is just like soil erosion and erodes away the boundary of the foreground object.
in this operation, the kernel is slid through the image and considers a pixel value 1 only when all the pixels in the structuring element have value 1. otherwise, it is eroded. in this way, the pixels near the boundary will be discarded, and a shrunk foreground object inside the image is obtained.

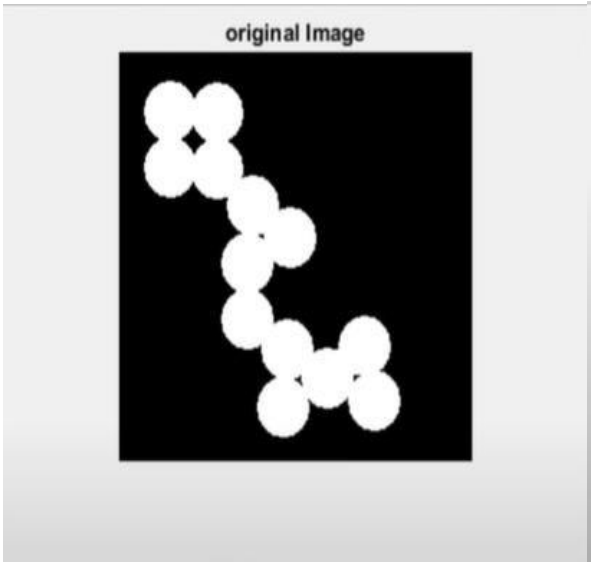
DILATION :-

it is the opposite of erosion, instead of shrinking it expands the foreground object. in this operation structuring element(kernel) is slid through the image. but, here a pixel value 1 if at least one pixel has value 1. hence, the object expands around the boundary and results in an expanded image.

CODE TO PERFORM MORPHOLOGICAL OPERATIONS IN OCTAVE :-

```
pkg load image%load image
i=imread('ssss.png');%input image
de=imread('s1.png');%imput iamge
b=im2bw(i);%convert image to binary
se1=strel('square',5);%perforig sterl operation
io=imopen(b,se1);%opening image
se2=strel('square',10);%performing sterl operation
ic=imclose(b,se2);%closing image
se3=strel('disk' , 11);%perforimg sterl opertaion
e=imerode(b,se3);%erode image
se4=strel('ball',5,5);%perforimg sterl opertaion
d=imdilate(de,se4);%dilate iamge
subplot(231);%assign position
imshow(i);%output image
title('original image');%assign name
subplot(232);%assign position
imshow(b);%output image
title('binary image of orignal image');%assign name
subplot(233);%assign position
imshow(io);%output image
title('iamge after opening operation');%assign name
subplot(234);%assign position
imshow(ic);%output image
title('image after closing operation');%assign name
subplot(235);%assign position
imshow(e);%output image
title('image after erosion');%assign name
subplot(236);%assign position
imshow(d);%output image
title('image after dilate operation ');%assign name
```

OUTPUT:-


original Image


original Image

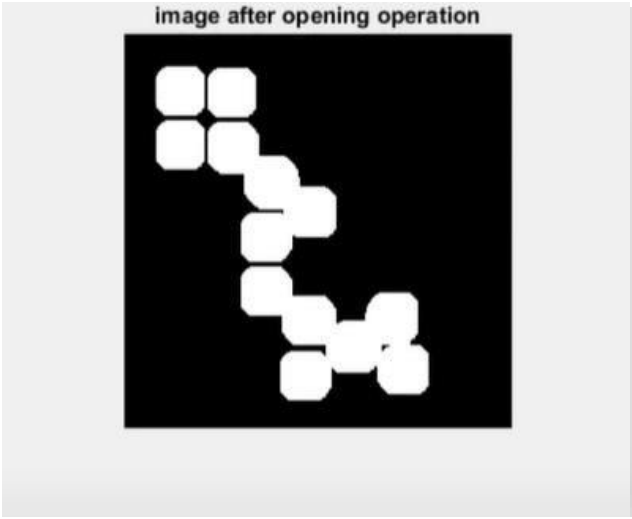


image after opening operation

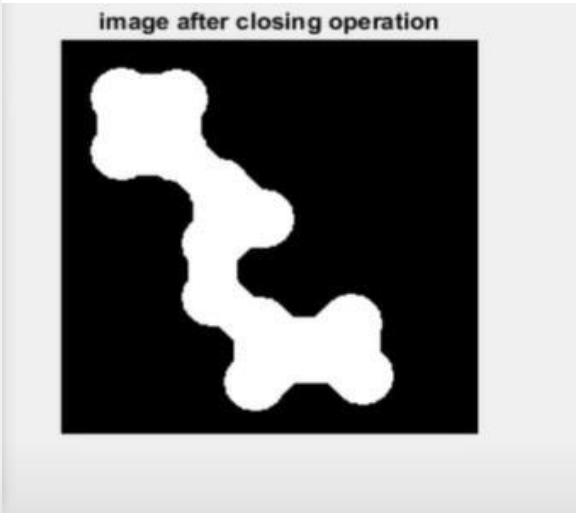


image after closing operation

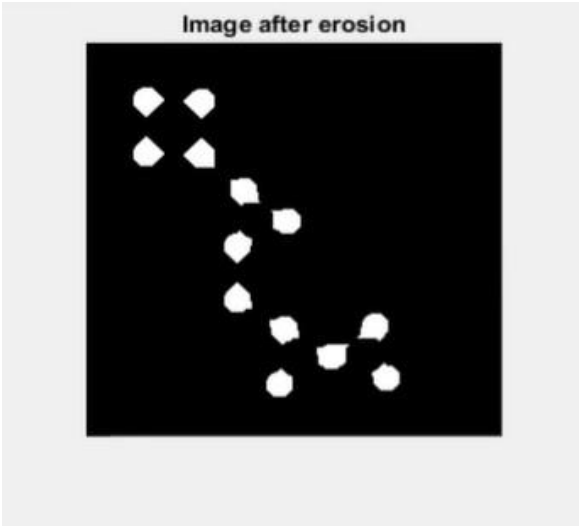


Image after erosion

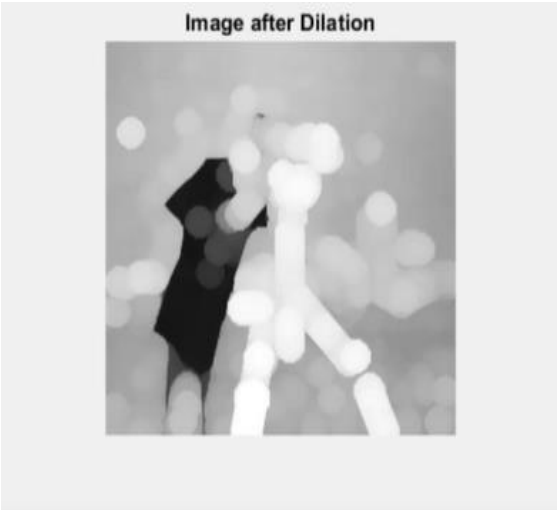


Image after Dilation

RESULT :-

up on performing this experiment i was able to understand that basically the morphological operations are those operations which are used to change the shape of the image using the help of sterl function which us present in the octave as inbult. dilation makes the image blur and erosion operation make the image object sizes changes.

PRECAUTIONS :-

3. make sure to use the command pkg load image
4. make sure that there are no erros in the code.

CONCLUSION :-

upon performing this experiment i was able to know how morphological operations are usefull to change the image into our desired images and are used in daily lifes in photo editing softwares.

EXPERIMENT NO:- 6

AIM:-

TO OBATIN HISTOGRAM EQUAIZATION OF THE BOTH ORIGINAL AND NEGATIVE IMAGE.

APPARATUS REQUIRED:-

OCTAVE SOFTWARE

THEORY:-

the negative of an image is achieved by replacing the intensity 'i' in the original image by 'i-1', i.e. the darkest pixels will become the brightest and the brightest pixels will become the darkest. image negative is produced by subtracting each pixel from the maximum intensity value1. the transformation function used in image negative is : s = t ® = (l – 1) – r where l - 1 is the max intensity value, s is the output pixel value and r is the input pixel value1

histogram equalization is a method in image processing that adjusts the contrast of an image by modifying the image's histogram. it is an intensity transformation process that uniformly distributes the image histogram over the entire intensity axis by choosing a proper intensity transformation function.

this method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values2. it can enhance the image's contrast by spreading out the most frequent pixel intensity values or stretching out the intensity range of the image.

OCTAVE CODE for NEGATIVE OF AN IMAGE:-

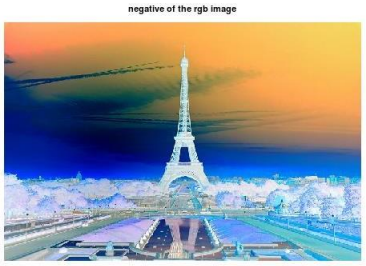
```
pkg load image
i=imread('sample.jpg'); %insert the rgb image
b=rgb2gray(i); %converting the rgb image to gray
l=256; %declearing the variable for use
r=(l-1)-i; %converting the rgb image to negative
r2=(l-1)-b;%converting the gray image to negative
subplot(221);%assigning the position
imshow(i);%output the original image
title('original rgb image'); %title
subplot(222);%assigning the position
imshow(r);%output the negative of the original image
title('negative of the rgb image'); %title
subplot(223);%assigning the position
imshow(b);%output of the gray original image
title('original gray image');%title
subplot(224);%assigning the position
imshow(r2);%output the negative of the gray original image
title('negative of gray image');%title
```

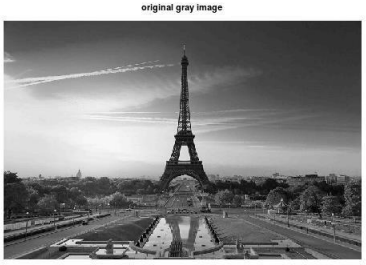
OCTAVE CODE TO PERFORM HISTOGRAM:-

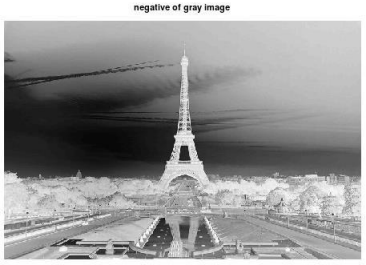
```
% Load the image
pkg load image
img = imread('sample.jpg');
% Convert the image to grayscale
img = rgb2gray(img);
%Perform histogram equalization on the original image
equ = histeq(img);
% Create the negative image
negative_img = 255 - img;
% Perform histogram equalization on the negative image
negative_equ = histeq(negative_img);
% Display the results
subplot(2, 2, 1), imshow(img), title('Original Image');
```

```
subplot(2, 2, 2), imshow(equ), title('Equalized Image');
subplot(2, 2, 3), imshow(negative_img), title('Negative Image');
subplot(2, 2, 4), imshow(negative_equ), title('Equalized Negative Image');
```

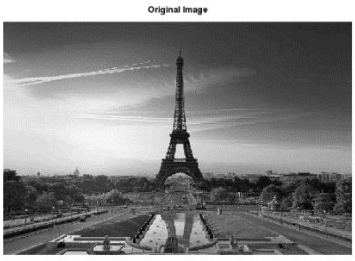
OUTPUT OF THE NEGATIVE IMAGE CODE:-

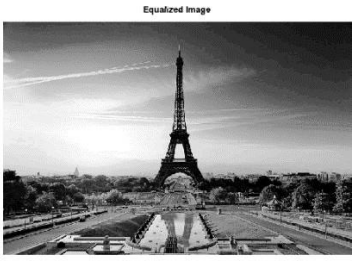


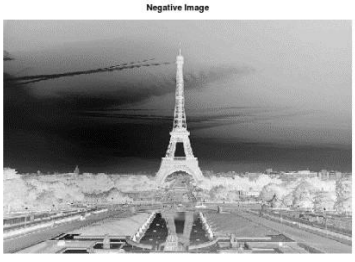


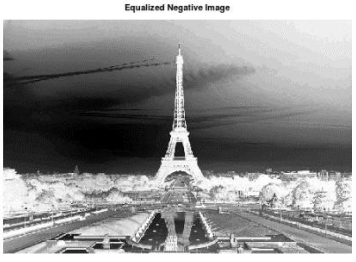


OUTPUT OF THE HISTOGRAM CODE:-









RESULT: -

by observing the output images i was clear that the negative of an image is basically the changing the intensity of the original image form **i** to **i-1** .and have observed that the darkest pixels has become brightest and vise versa.

upon observing the output images usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values2. it can enhance the image's contrast by spreading out the most frequent pixel intensity values or stretching out the intensity range of the image.

PRECAUTIONS: -

1. make sure to use the command pkg load image
2. make sure that there are no erros in the code.

CONCLUSION: -

upon performing this experiment i was able to obtain the negative of an image from both the Gray and rgb image as well and both of them look some thing strange when compared to the original images as their intensities have been reduced by -1 for each pixel.upon performing the experiment i was able to gain knowledge about what is histogram and how is it implemented to images and how it works.

EXPERIMENT NO :- 7

AIM:-

TO PERFORM VARIOUS EDGE DETECTION METHODS FOR GIVEIMAGE(PREWITT,SOBEL,ROBERT).

APPARATUS REQUIRED:-

OCTAVE SOFTWARE.

THEORY:-

edge detection is a fundamental tool in image processing, machine vision and computer vision. it includes a variety of mathematical methods that aim at identifying edges or curves in a digital image where the image brightness changes sharply or has discontinuities1.

there are several edge detection operators such as sobel, robert's cross, laplacian of gaussian, and prewitt operator2. the purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world.

it can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to discontinuities in depth, discontinuities in surface orientation, changes in material properties and variations in scene illumination.

roberts, sobel, and prewitt are all gradient-based edge detection methods used to find edge pixels in an image. these methods work by approximating the gradient magnitude of the image and identifying areas where the gradient magnitude is high, indicating a sharp change in brightness.

THE ROBERTS :-

operator computes the sum of squares of the differences between diagonally adjacent pixels in an image through discrete differentiation. then the gradient approximation is made1.

THE SOBEL:

operator is a discrete differentiation operator that computes the gradient approximation of image intensity function for image edge detection. at the pixels of an image, the sobel operator produces either the normal to a vector or the corresponding gradient vector. it uses two 3 x 3 kernels or masks which are convolved with the input image to calculate the vertical and horizontal derivative approximations respectively1.

THE PREWITT :-

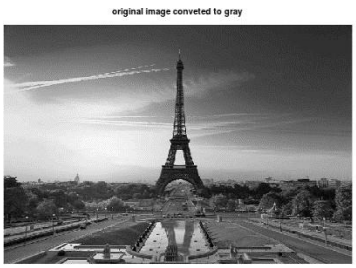
operator is similar to the sobel operator and also detects vertical and horizontal edges of an image. it is one of the best ways to detect the orientation and magnitude of an image. it uses two 3 x 3 kernels or masks

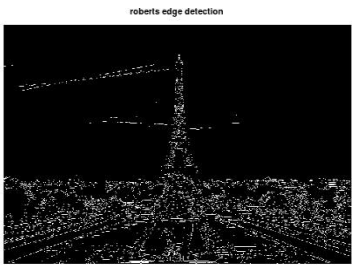
which are convolved with the input image to calculate the vertical and horizontal derivative approximations respectively

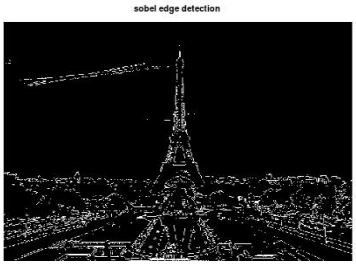
CODE :-

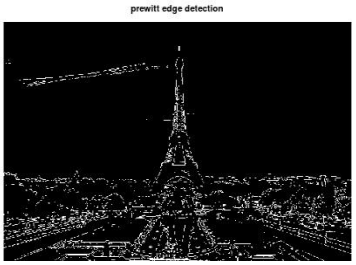
```
% Load image package
pkg load image;%image load
% Read image
im= imread('sample.jpg');%read image
% Convert image to grayscale
im = rgb2gray(im);%convert the image into rgb to gray
% Perform edge detection using different methods
bw_roberts = edge(im, 'Roberts');%perform roberts edge
bw_sobel = edge(im, 'Sobel');%perform sobel edge
bw_prewitt = edge(im, 'Prewitt');%perform prewitt edgr
% Display results
subplot(221);%assign position
imshow(im);%outptu image
title('original image conveted to gray');%assign name
subplot(222);%assign position
imshow(bw_roberts);%outptu image
title('roberts edge detection');%assign name
subplot(223);%assign position
imshow(bw_sobel);%outptu image
title('sobel edge detection');%assign name
subplot(224);%assign position
imshow(bw_prewitt);%outptu image
title('prewitt edge detection');%assign name.
```

OUTPUT OF THE ABOVE CODE :-









RESULT :-

the result of applying edge detection methods such as roberts, sobel, and prewitt to an image is a binary image where the edges in the original image are highlighted. the edges are represented by white pixels while the non-edges are represented by black pixels. the resulting binary image can be used for further analysis or processing.

each method has its own strengths and weaknesses and may produce slightly different results when applied to the same image. for example, some methods may be more sensitive to noise or may be better at detecting edges in certain directions. it is important to choose the right method for your specific application and experiment with different methods to see which one produces the best results for your needs.

PRECAUTIONS :-

1. make sure to use the command pkg load image
2. make sure that there are no erros in the code.

CONCLUSION :-

edge detection is a fundamental tool in image processing that can be used to identify edges or curves in a digital image where the image brightness changes sharply or has discontinuities.there are several edge detection methods available, including roberts, sobel, and prewitt, which use different mathematical approaches to identify edges in an image. each method has its own strengths and weaknesses and may produce slightly different results when applied to the same image. it is important to choose the right method for your specific application.

EXPERIMENT NO :- 8

AIM :-

USE LOGICAL(OR/AND/NOT)AND ARITHEMATIC(+,-) OPERATIONS WITH DIFFERENT IN AGES.

APPARATUS REQUIRED :-

OCTAVE INSTALLED

THEORY:-

arithmetic operations such as addition, subtraction, multiplication, and division can be used to perform a variety of tasks in image processing. for example, image subtraction can be used to detect differences between two or more images of the same scene or object. image addition can be used to combine multiple images into a single image, while image multiplication and division can be used to adjust the brightness and contrast of an image.

logical operations such as and, or, and not are often used to combine two (mostly binary) images. in the case of integer images, the logical operator is normally applied in a bitwise way. for example, the logical and operation can be used to mask out certain parts of an image by combining it

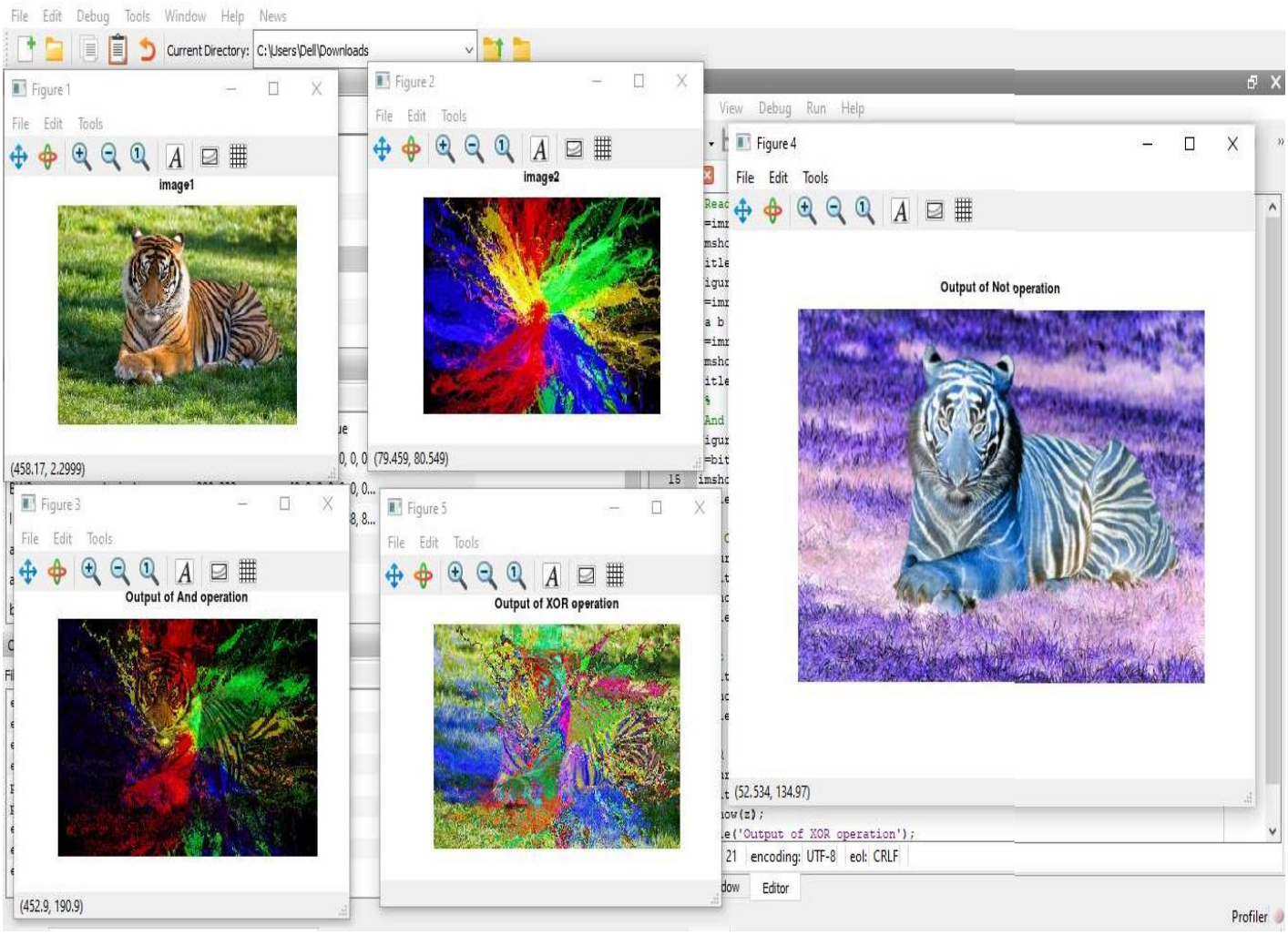
with a binary mask image. the logical or operation can be used to combine two binary images into a single image that contains all the white pixels from both input images.

both arithmetic and logical operations are applied in a pixel-by-pixel way, which means that the value of a pixel in the output image depends only on the values of the corresponding pixels in the input images. hence, the images must be of the same size.

CODE for logical operation:-

```
%%
%Reading Required Images & required Preprocessing
x=imread('tiger1.jpeg');
imshow(x);
title('image1');
figure;
y=imread('color.jpeg');
[a b c]=size(x);
y=imresize(y,[a,b]);
imshow(y);
title('image2');
%%
%And Operation
figure;
z=bitand(x,y);
imshow(z);
title('Output of And operation');
%%
%Or Operation
figure;
z=bitor(x,y);
imshow(z);
title('Output of Or operation');
%%
%Not Operation
z=bitcmp(x);
imshow(z);
title('Output of Not operation');
%%
%XOR Operation
figure;
z=bitxor(x,y);
imshow(z);
title('Output of XOR operation');
```

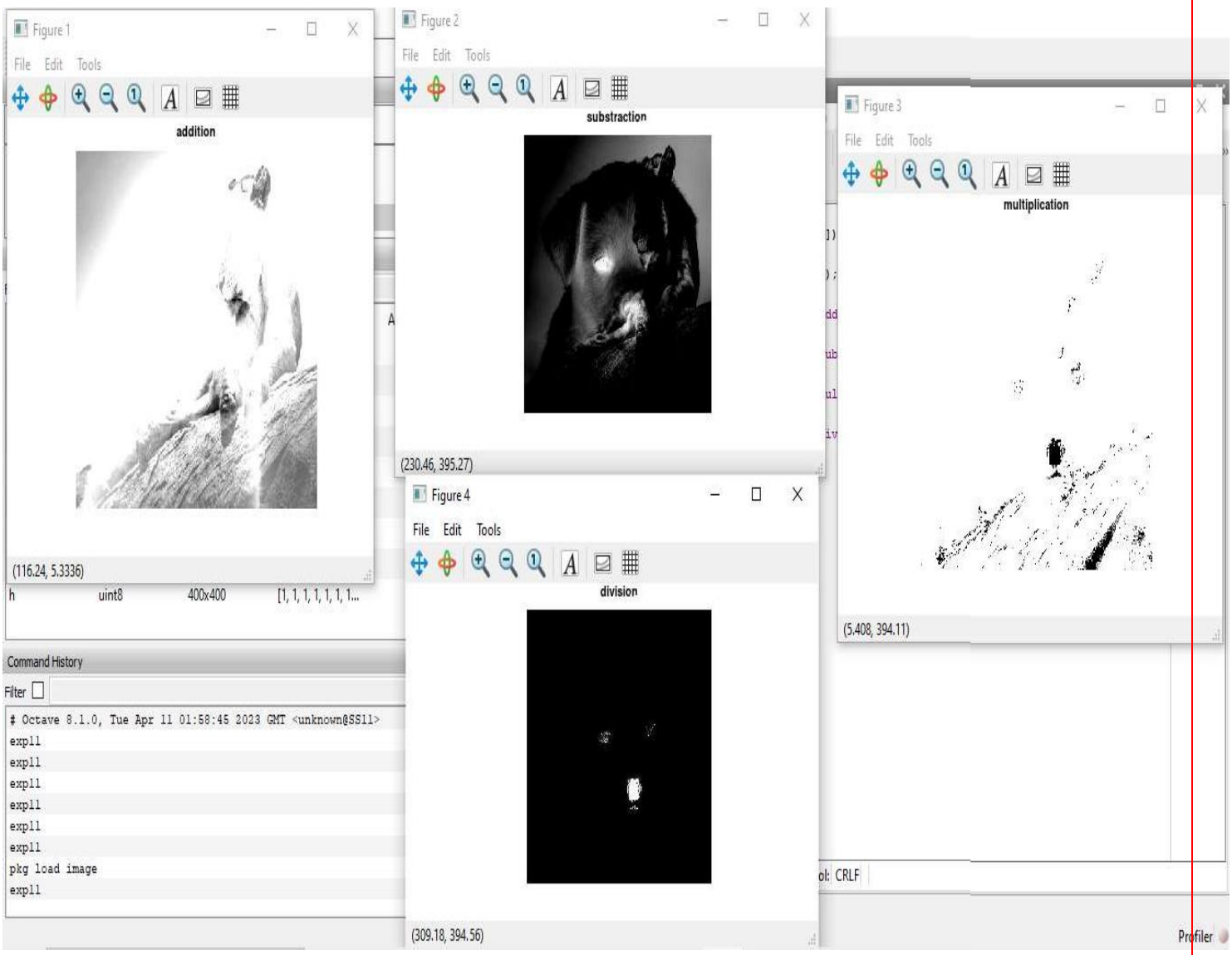
OUTPUT OF THE ABOVE CODE :-



CODE for Arithmetic operation:-

```
a=imread ('lion.jpeg');
b = imresize (a, [400,400]);
c=imread ('doggy.jpg');
d= imresize (c, [400,400]);
e=imadd(b,d);
figure,imshow(e),title('addition');
f=imsubtract(b,d);
figure,imshow(f),title('substraction');
g=immultiply(b,d);
figure,imshow(g),title('multiplication');
h=imdivide(b,d);
figure,imshow(h),title('division')
figure,imshow(g),title('multiplication');
figure,imshow(h),title('division');
```

OUTPUT OF THE ABOVE CODE :-



RESULT :-

all the operations are seems to be working fine but the **and** and **or** operations seems to be specific that we are not getting the images of these after performing the operations the reason behind this is

if the minimum pixel value for the image after the logical and operation is 0 and the maximum pixel value is 1, it means that the image contains both black and white pixels. similarly, if the minimum and maximum pixel values for the image after the logical or operation are both 1, it means that all pixels in the image are white.

it could be helpful to check if the input images are binary images (i.e., images that contain only black and white pixels) before performing the logical operations. logical operations such as and and or are typically applied to binary images. if the input images are not binary, you could try converting them to binary images using a thresholding operation before performing the logical operations.

PRECAUTIONS :-

1. make sure to use the command pkg load image
2. make sure that there are no erros in the code.

CONCLUSION:-

up on performing this experiment i was able to understand how to perform arithmetic and logical operations on an image and as well as upon two different images with same pixels sizes. it was a great experience.

EXPERIMENT NO :- 9

AIM :-

PERFORM THE FOLLOWING :-

1. REMOVE ALTERNATIVE ROWS AND COLOUMNS OF AN IMAGE.
2. SHOW EVERY COLOUR COMPONENT OF AN IMAGE.

APPARATUS USED :-

OCTAVE INSTALLED

THEROY :-

REMOVING ALTERNATE ROWS AND COLUMNS AND DISPLAYING COLOR COMPONENTS IN AN IMAGE:

removing alternate rows and columns from an image can be thought of as a simple way to down sample or reduce the size of the image. this can be useful for reducing the amount of data that needs to be processed or transmitted, or for displaying the image on a lower-resolution display. however, this method can result in a loss of detail and sharpness in the image. there are other methods for down sampling an image that can preserve more detail, such as using interpolation or more advanced techniques like wavelet transforms.

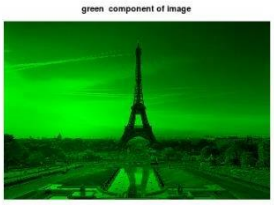
as for displaying colour components of an image, it can be useful for understanding the composition of an image and how different colours combine to create the final image. each pixel in a colour image is typically represented by three colour components: red, green, and blue (rgb). these components represent the intensity of each primary colour at that pixel. by separating these components and displaying them individually, you can see how each component contributes to the overall appearance of the image. this can be useful for tasks such as colour correction or image analysis.

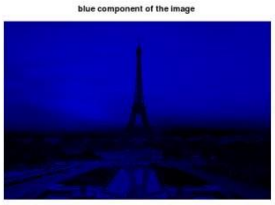
OCTAVE CODE :-

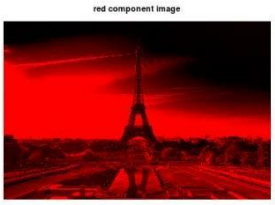
```
pkg load image
img = imread('sample.jpg'); % Read the image
a=img;
i= img(1:5:end,1:5:end, :); % Remove alternate rows and columns
figure; % Create a new figure
title('output image after removing alternate rowsa nad coloumns ');
subplot(231);
 imshow(img);
 title('Original Image'); % Display the original image
subplot(233);
imshow(i);%displaying the output image
title('image after removvveing  alternative  rows and coloumns');
% obtain red componentt of the image
img(:,:,2)=0;
img(:,:,3)=0;
subplot(236);
imshow(img);
```

```
title('red component image');
%obtain green component of the image
img=a;
img(:,:,1)=0;
img(:,:,3)=0;
subplot(234);
imshow(img);
title('green  component of image');
% obtain blue component of the image
img=a;
img(:,:,1)=0;
img(:,:,2)=0;
subplot(235);
imshow(img);
title('blue component of the image ');
```

OUTPUT OF THE ABOVE IMAGE :-







RESULT :-

Removing alternate rows and columns from an image results in an image that is half the size in both dimensions. This means that the

resulting image will have one-quarter of the total number of pixels as the original image. This can be useful for reducing the amount of data that needs to be processed or transmitted, or for displaying the image on a lower-resolution display however, removing alternate rows and columns can also result in a loss of detail and sharpness in the image. this is because the process discards information from the original image. there are other methods for down sampling an image that can preserve more detail, such as using interpolation or more advanced techniques like wavelet transforms.

Displaying the colour components of an image separately can provide insight into the composition of the image and how different colours combine to create the final image. Each pixel in a colour image is typically represented by three colour components: red, green, and blue (RGB). These components represent the intensity of each primary colour at that pixel. By separating these components and displaying them individually, you can see how each component contributes to the overall appearance of the image. For example, you might see that an image has a strong red component in certain areas, indicating that those areas are predominantly red in colour. This can be useful for tasks such as colour correction or image analysis

PRECAUTIONS: -

1. make sure to use the command pkg load image
2. make sure that there are no erros in the code.

CONCLUSION :-

upon performing the experiment i was able to understand by removing the alternative rows and columns in an image results in decreasing the no of pixels in an image which results in loss of quality of an image and have observed that how to generate the red ,green, blue etc.

EXPERIMENT NO :- 10

AIM :-

TO PERFORM THE FOLLOWING :-

1. USE GAMMA CORRECTION ON A RGB IMAGE.
2. USE BIT PLANE SLICING FOR AN IMAGE.
3. TO PERFORM FFT AND ITS SHIFT AND INVERSE OF AN IMAGE IN OCTAVE.

APPRATUS REQUIRED :-

OCTAVE INSTALLED

THEORY :-

GAMMA CORRECTION :-

It is used to optimize the storage and transmission of image data by taking advantage of the non-linear manner in which humans perceive light and colour. Gamma correction redistributes tonal levels closer to how our eyes perceive them, allowing for more efficient use of bits when encoding an image.

FAST FOURIER TRANSFORM :-

The Fast Fourier Transform (FFT) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, or its inverse (IDFT) 1. The DFT decomposes a sequence into its sine and cosine components, representing it in the frequency domain .When applied to an image, the 2D FFT computes the 2D DFT of the image, representing it in the frequency domain. Each point in the resulting Fourier image represents a particular frequency contained in the spatial domain image.

The fftshift function is used to shift the zero-frequency component of the Fourier image to the centre of the spectrum. This makes it easier to visualize and analyse the frequency content of the image. The inverse FFT (IFFT) computes the inverse 2D DFT of the Fourier image, reconstructing the original image in the spatial domain. The resulting image should be identical or nearly identical to the original input image.

BIT PLANE SLICING :-

bit plane slicing is a technique used in image processing to represent an image with one or more bits of the byte used for each pixel 1. it involves converting an image into multilevel binary images by dividing it into bit planes. for an 8-bit image, each pixel value is represented by 8 bits. the leftmost bit is known as the most significant bit (msb) and the rightmost bit is known as the least significant bit (lsb) . in bit plane slicing, the

image is divided into 8 bit planes, one for each bit of the pixel values. each bit plane represents a binary image where the pixel values are determined by the corresponding bit of the original pixel values.

bit plane slicing can be used for several purposes, including image compression, converting a grayscale image to a binary image, and analysing the relative importance of each bit in the image 1. in general, higher-level bit planes are useful for edge detection while lower-level bit planes add subtle features to an image .

OCATVE CODE TO PERFORM ALL THE THREE AT ONCE :-

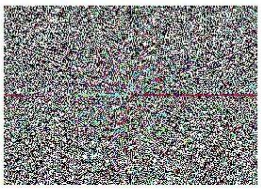
```
pkg load image
img=imread('sample.jpg');%load image
img=im2uint8(img);%convert the bits of an image
a=rgb2gray(img);%convert the image to gray
r = double(img)/255;%assign the value for a varaible
c = 1;%assign the value for a varaible
gamma = 0.6;%assign the value for a varaible
s = c * (r).^gamma;%assign the value for a varaible
F = fft2(img); % Compute the 2D FFT of the image
Fsh = fftshift(F); % Shift the zero-frequency component to the center of
the spectrum
imgInv = ifft2(Fsh); % Compute the inverse 2D FFT of the shifted spectrum
subplot(321);%assign position
imshow(img);%output image
title('original image');%assign name
subplot(322);%assign position
imshow(s);%output image
title('after performing gaama correction image looks like');%assign name
subplot(323);%assign position
img_1 = real(Fsh);%load image to formats
imshow(img_1, []);%output image
title('fft shift of the original image');%assign name
subplot(324);%assign position
imgReal = real(imgInv);
imshow(imgReal, []);%output image
title('ifft of the original image');%assign name
img = im2uint8(a); % Convert the image to 8-bit unsigned integer
bit_planes = zeros(size(a, 1), size(a, 2), 8); % Initialize an array to
store the bit planes

for i = 1:8
    bit_planes(:, :, i) = bitget(a, i); % Extract the i-th bit plane
end
figure;
for i = 1:8
    subplot(2, 4, i);%assign position
    imshow(bit_planes(:, :, i));%output image
    title(['Bit plane ', num2str(i)]);%assign name
end
```

OUTPUT:-

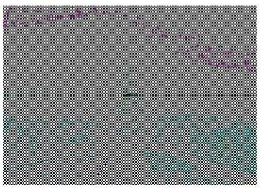
original image

after performing gaama correction image looke like

fft shift of the original image

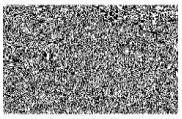


ifft of the original image



BIT PLANES :-

Bit plane 1



Bit plane 2

Bit plane 3

Bit plane 4

Bit plane 5

Bit plane 6

Bit plane 7

Bit plane 8

RESULT :-

Upon observing the output images and when compared to the original images while performing the gamma correction I was able to observe the change in colour ,contrast, brightness of the output image when compared to the input image .

While observing the FFT shifted ,inverse images of the original image I was able to under stand that we are getting only the real parts of the complex matrix rest is not able to get and the start pixel is shifted to the centre of the image and it has performed as per the given code.

While performing the bit plane slicing I was able to observe that the bit planes only comes when the taken image is as per the range of the given bit and it happens only in Gray image format we are unable to get the colour planes.

PRECAUTIONS :-

1. make sure to use the command pkg load image
2. make sure that there are no errors in the code.

CONCLUSION :-

Upon performing this experiment I was able to understand how to perform gamma correction , FFT transform of an image , and bit plane slicing etc and I have observed that in gamma correction there is a change in the input image and output image and after performing FFT I was able to observe that only real part of the image is going to shown and in bit plane slicing the image bit type should be in the range of given bits and must be of Gray image of the original image.