ECC 603: Image Processing and Machine Vision
(IPMV)

Submitted in partial fulfilment of the requirements. for the degree of

Bachelor of Engineering

by

Name: Dinesh Kakade

Roll No: 22104B2010

Supervisor: Prof. Amit Maurya

Department of Electronics and Telecommunication Engineering

VIDYALANKAR INSTITUTE OF TECHNOLOGY, MUMBAI

(Ac. Yr. 2024-25)

INDEX

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

Dept of Electronics & Telecommunication.
Lab Manual of VLSI SEM-5

IPMV EXPERIMENT 1

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC
Date of Performance	11/01/2024
Date of Submission	18/01/2024

Title : Fundamental of Image Processing: Generation, Colour Manipulation, and Grayscale Conversion

Aim : is to generate, read, separate colour channels, and convert RGB images to grayscale.

Algorithm:

1. Create a horizontal line (`hl`), a vertical line (`vl`), a plus sign (`plus), and a cross sign (`cross`) using matrix operations.

2. Utilize subplots and the `imshow` function to display the horizontal line, vertical line, plus sign, and cross sign.

3. Read the image using the `imread` function and display it in a subplot.

4. Extract the red (`r`), green (`g`), and blue (`b`) colour channels from the original image and display them in separate subplots.

5. Generate a grayscale image (`new`) by averaging the values of the red, green, and blue channels. Display the grayscale image in a subplot.

Takeaway:

1] This MATLAB code provides a practical illustration of fundamental image processing concepts, including the generation of basic shapes, reading and displaying images, colour channel separation, and grayscale conversion. It serves as a hands-on introduction to common operations used in image processing tasks, aiding in understanding and implementing basic algorithms for image manipulation.

2] We came across several image formats like JPEG, PNG, GIF, TIFF, BMP, RAW, SVG, WebP, HEIF, and ICO etc.

3] Out of the above format Bitmap is best of all but it is not commonly used. Pros and Cons Of bitmap are:

Advantages of BMP:

1. Lossless Quality: BMP is an uncompressed image format, preserving the original image quality without any loss.

2. Simplicity: BMP files are relatively simple and easy to understand, making them straightforward for basic image storage.

Limitations of BMP:

1. File Size: Being uncompressed, BMP files are large, which can be a significant drawback when considering storage and transmission over the internet.

2. Lack of Compression: The absence of compression means that BMP files do not benefit from the space-saving advantages offered by compressed formats, making them less practical for certain use cases.

3. Limited Features: BMP does not support features like transparency or multiple layers, which are crucial for various applications.

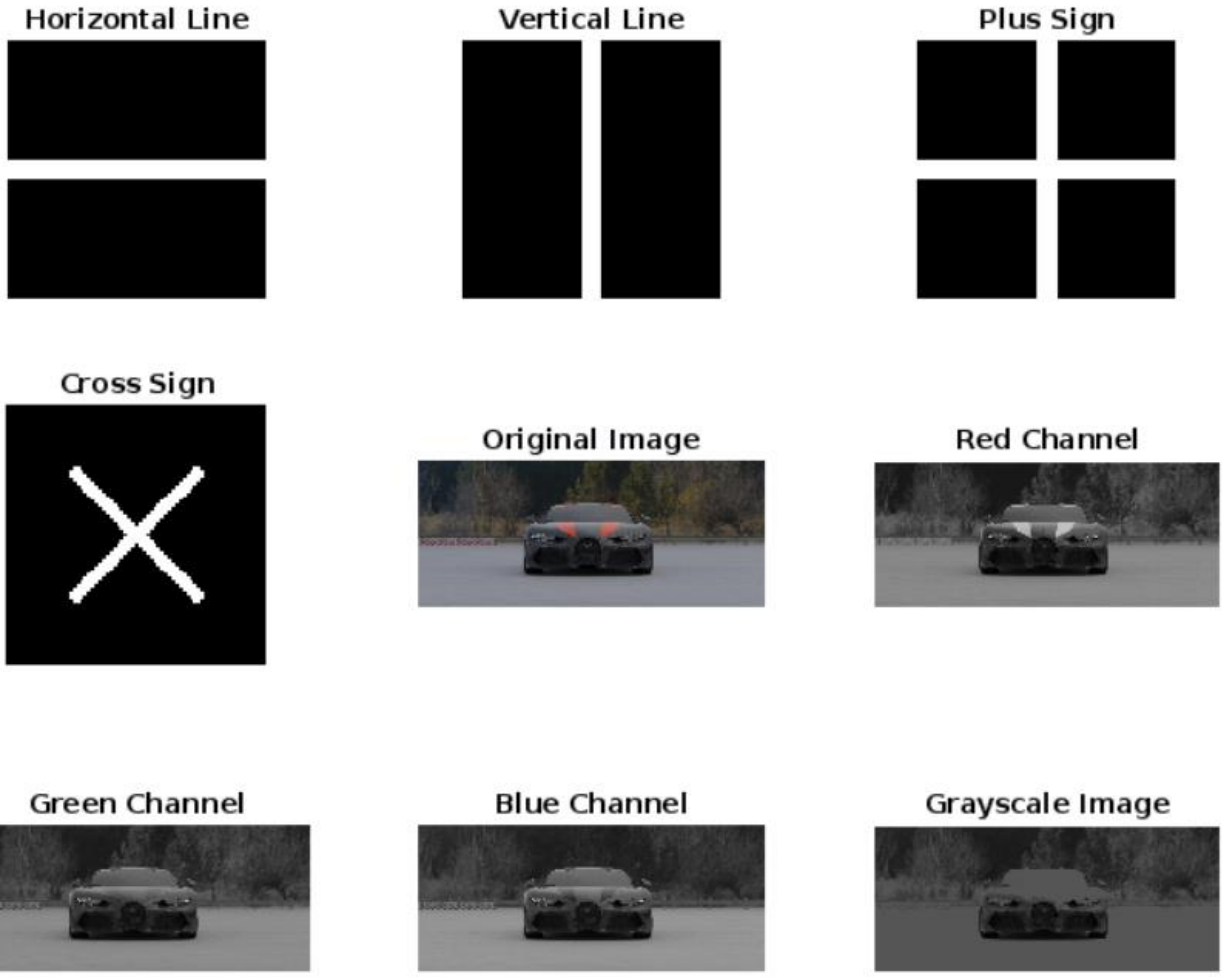
4] We also learned that if we rotate an image and it goes out of the canvas then the MATLAB automatically will extend the canvas.

Program / Output

```
clear; close all; clc;
% Creating cross, plus and displaying lines
hl = [zeros(23, 50); ones(4, 50); zeros(23, 50)];
vl = hl';
plus = hl + vl;
cross = imrotate(plus, 45);
subplot(3, 3, 1);
imshow(hl);
title('Horizontal Line');
subplot(3, 3, 2);
imshow(vl);
title('Vertical Line');
subplot(3, 3, 3);
imshow(plus);
title('Plus Sign');
subplot(3, 3, 4);
imshow(cross);
title('Cross Sign');
% Reading Images
photo = imread("0080.png");
subplot(3, 3, 5);
imshow(photo);
title('Original Image');
```

```
% Spliting Colour Channel
r = photo(:,:,1);
g = photo(:,:,2);
b = photo(:,:,3);
subplot(3, 3, 6);
imshow(r);
title('Red Channel');
subplot(3, 3, 7);
imshow(g);
title('Green Channel');
subplot(3, 3, 8);
imshow(b);
title('Blue Channel');
% Generating Grey Scale Image
new = (r + g + b) / 3;
subplot(3, 3, 9);
imshow(new);
title('Grayscale Image');
```

Output :



```
photo = imread('0241.png');
grayPhoto = rgb2gray(photo);
imshow(grayPhoto);
subplot(1,2,1);
imshow(photo);
subplot(1,2,2);
imshow(grayPhoto);
```

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

Department of Electronics and Telecommunication Engineering

Semester	T.E. Semester VI – EXTC Engineering
Subject	Image Processing and Machine Vision
Laboratory Teacher	Prof. Amit Maurya
Laboratory	M518

Name: Dinesh Dilip Kakade

Roll No. 22104B2010

Batch: 4

TE EXTC B

Date of Performance: 18/01/2024

Date of Submission: 25/01/2024

Title: Image Processing Techniques

Aim: Demonstrate various image processing techniques

Software Requirement: MATLAB Online

Algorithm:

1. Read a grayscale image file into a variable (`img`).

2. Use the `imbinarize` function to convert the grayscale image to a binary image.

3. Use the `imadd` function to add a constant value to each pixel of the original image (`img`) to create brighter versions.

4. Use the `imsubtract` function to subtract a constant value from each pixel of the original image (`img`) to create darker versions.

5. Set pixel values above a specified threshold (`thresh`) in the original image (`img`) to the maximum intensity (255).
6. Create a negative image by subtracting each pixel value from the maximum intensity (255).

7. Convert the pixel values of the original image (`img`) to a double between 0 and Apply a logarithmic transformation to enhance lowintensity details.

8. Convert the pixel values of the original image (`img`) to a double between 0 and Apply a powerlaw transformation to adjust image intensity using a power function.

9. Use the `imcomplement` function to create the negative of the original image (`img`).

10. Create a 3x3 subplot. Display the original image and the processed images in separate subplots with appropriate titles.

Takeaway:

- In image processing ,brightness manipulation can be achieved through pixel addition or subtraction .
- Adding pixel values increses image brightness, while substracting values decreases it.
- Darkness the original image using pixel wise subtraction.
- The resulting threshing preserves original details in regions with intensity value below threshold.
- However ,pixels with values equal to or above threshold are set to the maximum intensity, leading to background distruction.

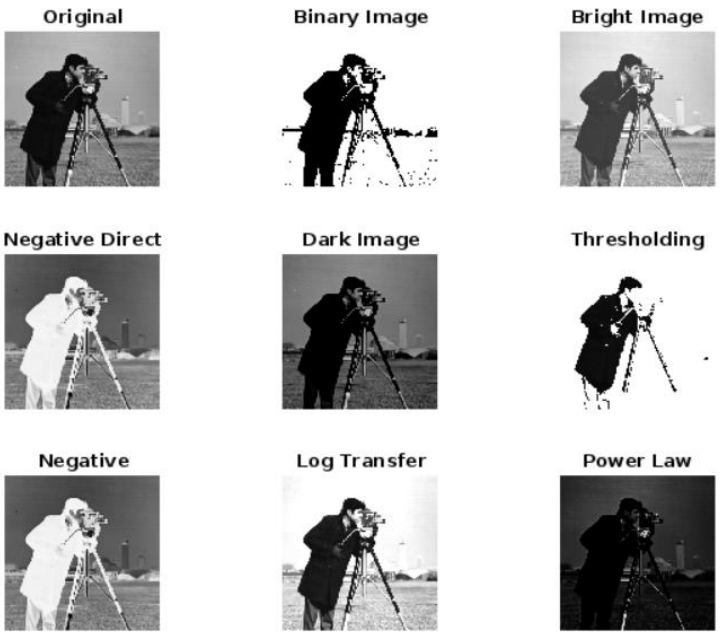
Title: Image processing enhancement Examples

Aim: Demonstrate various image processing enhancement techniques

```
% Clear the workspace, close all figures, and clear the command window
clear; close all; clc;
% Load the grayscale image (Cameraman)
img = imread("cameraman.tif");
% Binarize the image using the imbinarize function
bwimg1 = imbinarize(img);
% Brighten the image using pixel-wise addition
brightimg1 = imadd(img, 50);
brightimg2 = imadd(img, 100);
% Darken the image using pixel-wise subtraction
darkimg1 = imsubtract(img, 50);
% Get the size (dimensions) of the image
[m, n] = size(img);
% Thresholding using a specified threshold
thresh = 20;
threshimg1 = img;

for i = 1:m
      for j = 1:n
      % Check if the pixel value is greater than or equal to the threshold
         if (img(i, j) >= thresh)
            % Set the pixel value to the maximum intensity (255)
            threshimg1(i, j) = 255;
         else
         % Keep the original pixel value
         threshimg1(i, j) = img(i, j);
         end
         % Calculate the negative of each pixel value
         negimg1(i, j) = 255 - img(i, j);
      end
end

% Perform log transformation
doubleimg1 = double(img) / 255;
logimg1 = 2 * log(doubleimg1 + 1);
% Perform power-law transformation
powerlawimg1 = doubleimg1.^4;
% Create negative image using imcomplement
negimg = imcomplement(img);
% Display the processed images in a 3x3 subplot
1
subplot(3, 3, 1);
imshow(img);
title('Original')
subplot(3, 3, 2);
imshow(bwimg1);
title('Binary Image')
subplot(3, 3, 3);
imshow(brightimg1);
title('Bright Image +50')
subplot(3, 3, 4);
```

```
imshow(negimg);
title('Negative Direct')
subplot(3, 3, 5);
imshow(darkimg1);
title('Dark Image')
subplot(3, 3, 6);
imshow(uint8(threshimg1));
title('Thresholding')
subplot(3, 3, 7);
imshow(negimg1);
title('Negative')
subplot(3, 3, 8);
imshow(logimg1);
title('Log Transfer')
subplot(3, 3, 9);
imshow(powerlawimg1);
title('Power Law')
```

Result :



Published with MATLAB® R2023b

IPMV EXPERIMENT 3

Name	Dinesh Kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC
Date of Performance	25/01/2024
Date of Submission	1/02/2024

Title : Fundamental of Different Image Processing techniques

Aim : Generation of different image processing techniques 1. Gray level slicing 2. Contrast stretching 3. What is Histogram? 4.Logical operations on binary images(OR, AND ,NOT, EXOR) hint any 5. binary images as input 2.bit plane slicing

Algorithm:

1. Load two images, "pout.tif" and "trees.tif".
2. Set lower and upper thresholds for binary image generation (lt and ut). For each pixel in "moon.tif". If the pixel intensity is between lt and ut, set corresponding pixels in "gswb" and "gsb" to 255 (white). Otherwise, set corresponding pixels in "gswb" to 0 and "gsb" to the original intensity.
3. Set parameters for contrast stretching (r1, s1, r2, s2). Calculate slope values for contrast stretching (l, m, n). For each pixel in "pout.tif". If the pixel intensity is less than r1, apply the linear transformation using slope l. If the pixel intensity is between r1 and r2, apply the linear transformation using slope m. If the pixel intensity is greater than or equal to r2, apply the linear transformation using slope n.
4. Display the original and processed "moon" images along with their histograms.Display the original and contrast-stretched "pout" images along with their histograms.
5. Load the "cameraman.tif" image.Convert it to binary image "bin1". Load the "peppers.png" image, convert it to grayscale, and then to binary image "bin2". Perform logical operations. Compute bitwise NOT ("notimg"). Compute bitwise AND ("and_result"). Compute bitwise NAND ("nand_result"). Compute bitwise OR ("or_result").Compute bitwise NOR ("nor_result").
6. Display the original binary images, bitwise NOT, AND, NAND, OR, and NOR results.These six steps outline the main operations performed in the given MATLAB code.

Takaway:

1. The code demonstrates thresholding on the "pout.tif" image, creating binary images based on specified lower and upper intensity thresholds.
2. Contrast stretching is applied to the "trees.tif" image to enhance its visual contrast, using specified parameters for stretching.
3. The code displays original and processed images along with their histograms, providing a visual representation of pixel intensity distributions.
4. Binary operations like NOT, AND, NAND, OR, and NOR are performed on binary images created from the "cameraman.tif" and "peppers.png" images.
5. The code showcases a structured approach to image processing, involving thresholding, contrast stretching, logical operations, and visualization, providing insights into fundamental image processing techniques

Program / Output

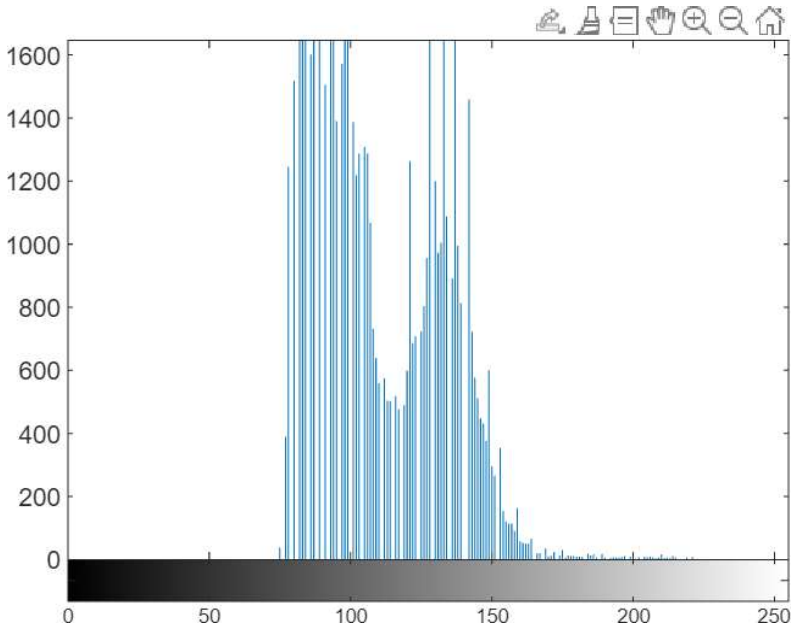
```
%% Title:
% Aim:

clear;
close all;
clc;
img= imread("pout.tif");
%img= imread("trees.tif");

lt =input("enter lower value:");    %enter lower value
ut =input("enter upper value:");     %enter upper value
%lt= 10;                      % set up the fixed lower threshhold
%ut= 30;                       %set up the fixed upper threshold

r1 = 80;
s1 =20;           %contrast stretching values
r2 = 180;
s2 = 255;

l= (s1-0)/(r1-0);
m= (s2-s1)/(r2-r1);  %slopes for contrast stretching in different regions
n= (255-s2)/(255-r2);

[row,col] =size(img);

for i=1:1:row
    for j=1:1:col
    if(img(i,j)>=lt && img(i,j)<=ut)
       gswb(i,j)=255;          %grayscale slicing without background
        gsb(i,j)=255;          %grayscale slicing  with background
    else
       gswb(i,j)=0;       %grayscale slicing without background
         gsb(i,j)=img(i,j); %grayscale slicing with background
    end

    if (img(i,j)<r1)
        contimg(i,j)=l*img(i,j);    % image contrst stretching
    elseif (img(i,j)>=r1 && img(i,j)<=r2)
        contimg(i,j)=m*(img(i,j)-r1)+s1;

    else
        contimg(i,j)=n*(img(i,j)-r2)+s2;
```

```
    end
end
end
%subplot(1,2,1); imshow(img);
%subplot(1,2,2); imshow(contimg);

subplot(2,2,1); imshow(img);
title("original img");
subplot(2,2,2);imshow(gswb);
title(" withoutBG ");
subplot(2,2,3);imshow(gsb);
title("withBG");
subplot(2,2,4); imshow(contimg);
title("contrst img");

figure; imhist(img);
```

output:

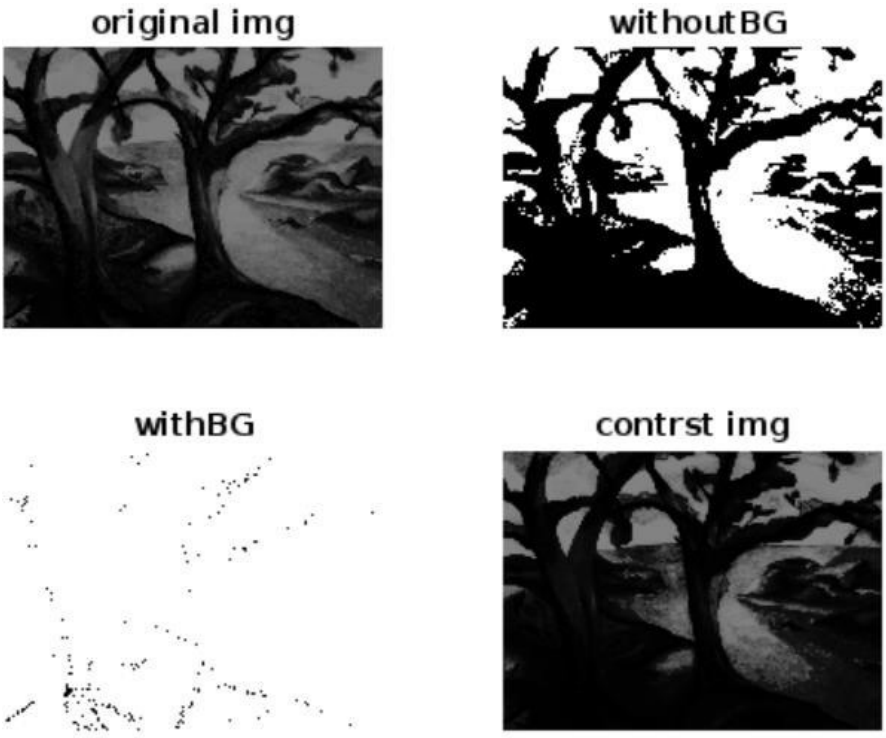
Lower threshold: 40

Upper threshold:80



Lower threshold: 50

Upper threshold:150

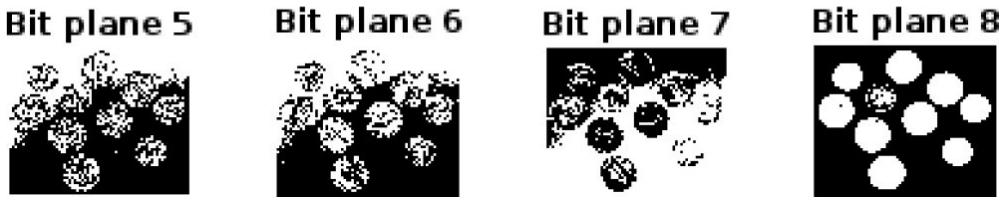


Homework:

Program:

```
clc; clear all; close all;
% Read the image 'coins.png' and convert it to double
A = imread('coins.png');
A = double(A);
% Display bit planes from 1 to 8
for plane = 1:8
% Extract the specified bit plane using bitget
B = bitget(A, plane);
% Create subplots to display each bit plane
subplot(2, 4, plane);
% Display the bit plane image
imshow(B);
% Set the title for each subplot indicating the bit plane number
title(['Bit plane ' num2str(plane)]);
end
```

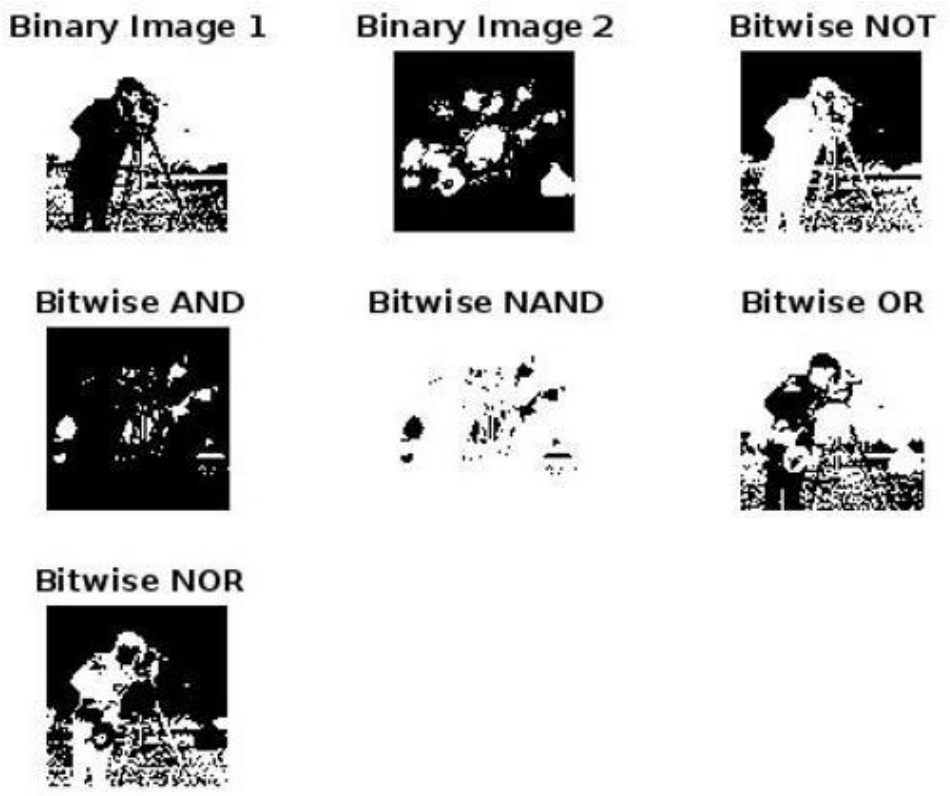
output:



Program:

```
% Binary image operations and logical operations
camera = imread("cameraman.tif");
bin1 = im2bw(camera);     %IMAAGE TO BINARY
rgb = imread("peppers.png");
pepper = rgb2gray(rgb);
bw = im2bw(pepper);
bin2 = imresize(bw, [256, 256]);
notimg = ~bin1;   %notimage
and_result = bin1 & bin2;      %AND IMAGE
nand_result = ~(bin1 & bin2); %NAND   IMAGE

or_result = bin1 | bin2;        %OR IMAGE
nor_result = ~(bin1 | bin2);          %NOR IMAGE
% Display binary image operations
figure;
subplot(3, 3, 1);
imshow(bin1);
title('Binary Image 1');
subplot(3, 3, 2);
imshow(bin2);
title(Binary Image 2);
subplot(3, 3, 3);
imshow(notimg);
title('Bitwise NOT');
subplot(3, 3, 4);
imshow(and_result);
```

```
title('Bitwise AND');
subplot(3, 3, 5);
imshow(nand_result);
title('Bitwise NAND');
subplot(3, 3, 6);
imshow(or_result);
title('Bitwise OR');
subplot(3, 3, 7);
imshow('Bitwise NOR');
title('Bitwise NOR')
```



IPMV EXPERIMENT 4

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	01 – 02 - 2024
Date of Submission	08 – 02 - 2024

Title : Add noise to a 8 bit white image and gray scale image.

Aim : Add Gaussian, Salt & pepper and sinusoidal noise to 8 bit white and gray scale image.

Software required :- MATLAB online

Algorithm:-

1. Clear workspace, close figures, and clear the command window.

2. Initialize a white image of size 100x100 pixels with intensity 255 (`uint8` data type). Display the original image in the first subplot.

3. Set the parameters `l` and `t` for Gaussian noise.Generate Gaussian noise using `imnoise` and add it to the original image. Display the image with Gaussian noise in the second subplot.

4. Generate salt and pepper noise using `imnoise` with a density of 0.4. Display the image with salt and pepper noise in the third subplot.

5. Convert the original image to double for sinusoidal noise generation. Define parameters for sinusoidal noise such as frequency and amplitude. Create a meshgrid and generate sinusoidal noise. Add the sinusoidal noise to the original image and display it in the fourth subplot.

6. Load the grayscale image "cameraman.tif". Display the original grayscale image in the first subplot.

7. Add Gaussian noise, salt and pepper noise, and sinusoidal noise to the grayscale image, similar to the steps for the 8-bit white image. Display the corresponding noisy images in the second, third, and fourth subplots.

Takeaway:

1. The code creates an 8-bit white image and a grayscale image ("cameraman.tif") for experimentation with various types of noise.

2. Three types of noise (Gaussian, salt and pepper, and sinusoidal) are applied to both the 8-bit white image and the grayscale image to simulate real-world image distortions.

3. The code utilizes MATLAB's subplot functionality to visually compare the original images with their corresponding noisy versions, making it easy to observe the effects of different types of noise.

4. Parameters such as noise intensity, density, frequency, and amplitude are adjustable, allowing for experimentation and customization of the generated noisy images.

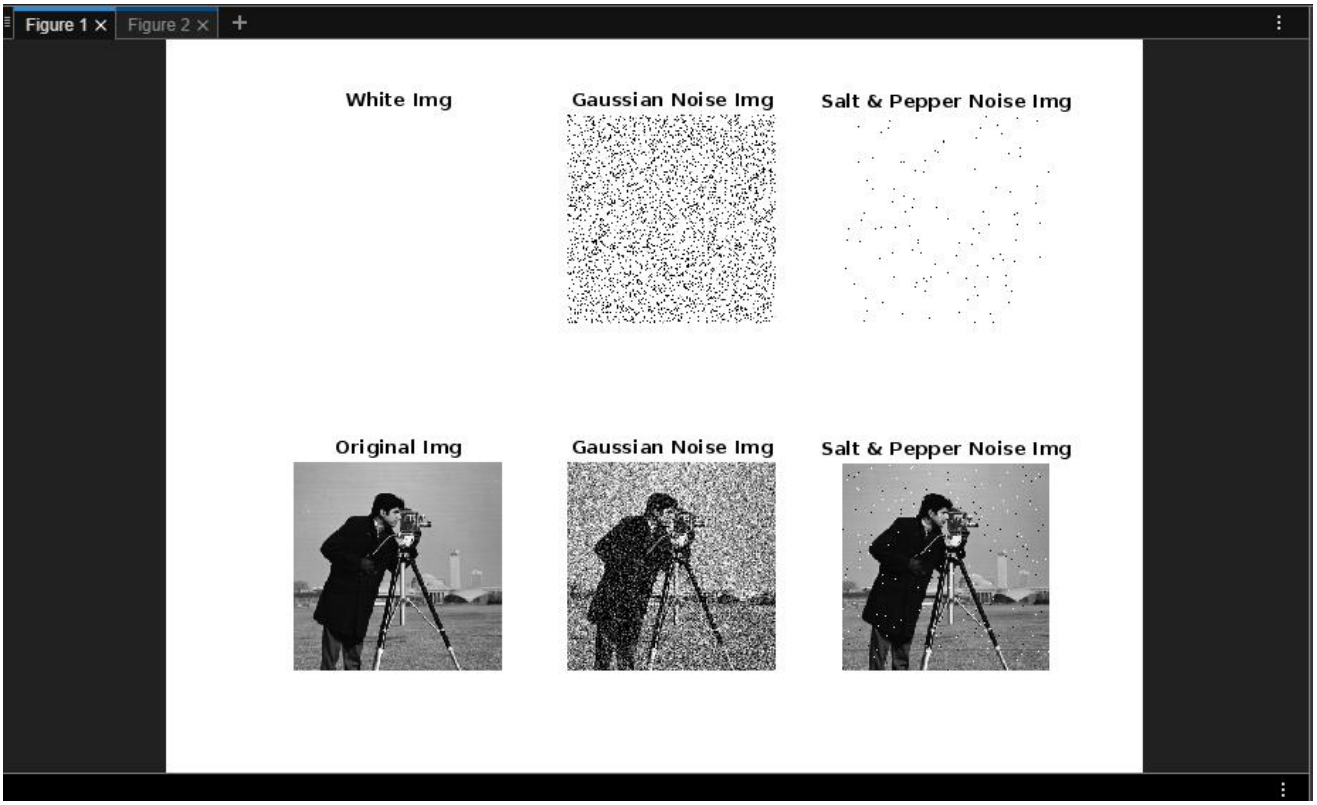
5. The code efficiently handles data types, converting images to `uint8` for the 8-bit white image and `double` for the grayscale image, ensuring compatibility during noise generation and addition.

Program :-

```
% Clearing variables, closing figures, and clearing command window
clear; close all; clc;
% Creating image of white color
white_img = uint8(ones(256,256) * 255);
% Parameters for gaussian noise
m = 5; % mean
v = 25; % variance
% Adding Gaussian noise to the white image
gauss_noise = imnoise(white_img, 'gaussian', m, v);
% Adding salt and pepper noise to the white image
sp_noise = imnoise(white_img, 'salt & pepper', 0.01);
% Creating sinusoidal noise
[rows, cols] = size(white_img);
frequency = 0.3;
amplitude = 0.1;
[x, y] = meshgrid(1:cols, 1:rows);
noise = amplitude * sin(2 * pi * frequency * x);
% Adding sinusoidal noise to the white image
noisy_image = im2uint8(im2double(white_img) + noise);
% Loading another image
img = imread('cameraman.tif');
% Parameters for gaussian noise
m = 0; % mean
v = 0.05; % variance
% Adding Gaussian noise to the image
gauss_img = imnoise(img, 'gaussian', m, v);
% Adding salt and pepper noise to the image
sp_img = imnoise(img, 'salt & pepper', 0.01);
% Creating sinusoidal noise
[rows, cols] = size(img);
frequency = 0.3;
amplitude = 0.1;
[x, y] = meshgrid(1:cols, 1:rows);
noise = amplitude * sin(2 * pi * frequency * x);
% Adding sinusoidal noise to the image
noisy_img = im2uint8(im2double(img) + noise);
% Displaying all the images in one figure
figure;
subplot(2,3,1);
imshow(white_img);
title("White Img");
subplot(2,3,2);
imshow(gauss_noise);
title("Gaussian Noise Img");
subplot(2,3,3);
imshow(sp_noise);
title("Salt & Pepper Noise Img");
```

```
subplot(2,3,4);
imshow(img);
title("Original Img");
subplot(2,3,5);
imshow(gauss_img);
title("Gaussian Noise Img");
subplot(2,3,6);
imshow(sp_img);
title("Salt & Pepper Noise Img");
% Adding the sinusoidal noise image
figure;
subplot(2,3,1);
imshow(white_img);
title("White Img");
subplot(2,3,2);
imshow(noisy_image);
title("Sinusoidal Noise Img");
subplot(2,3,3);
imshow(noisy_img);
title("Sinusoidal Noise Img");
```

Output :



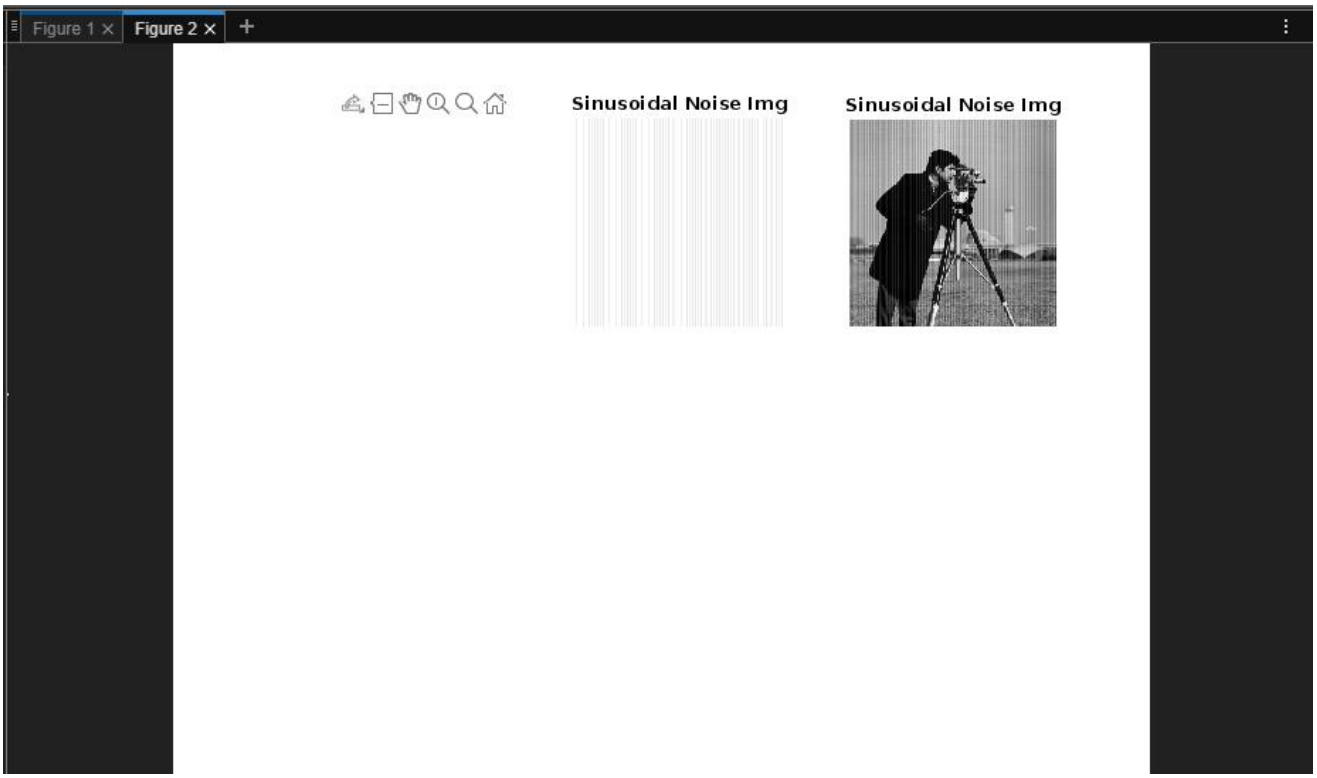


Figure 1
Figure 2
Sinusoidal Noise Img
Sinusoidal Noise Img

IPMV
EXPERIMENT 5

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	0– 02 - 2024
Date of Submission	22 – 02 - 2024

Title : mask of image

Aim : Add Gaussian, Salt & pepper and sinusoidal noise to image and filtered this image and show mask ‘

Software required :- MATLAB online

Algorithm:-

- Read three images
- Define parameters
- mask_size: Size of the mask for filters (set to 5).
- sigma: Sigma value for the Gaussian filter (set to 1.5).
- Create a meshgrid for x and y coordinates.
- Define a Gaussian mask using the meshgrid and sigma value.
- Normalize the Gaussian mask.
- Convolve the Gaussian mask with the g_img using conv2 function, maintaining the same size as the original image.
- Display Gaussian filter mask values.
- Define a sinusoidal mask (a simple notch filter) with all values initialized to 1.
- Set the central value of the mask to 0 to suppress sinusoidal noise.
- Convolve the sinusoidal mask with the sinosoidal_img using conv2 function, maintaining the same size as the original image.

Takeaway:

It illustrates how to design filter masks for different types of noise. For example, Gaussian filter mask is designed based on sigma value to blur the image, a simple notch filter is used to suppress sinusoidal noise, and a median filter is employed to remove salt and pepper noise.

1. The code utilizes MATLAB's subplot functionality to visually compare the original images with their corresponding noisy versions, making it easy to observe the effects of different types of noise.

2. Parameters such as noise intensity, density, frequency, and amplitude are adjustable, allowing for experimentation and customization of the generated noisy images.

3. The code efficiently handles data types, converting images to `uint8` for the 8-bit white image and `double` for the grayscale image, ensuring compatibility during noise generation and addition.

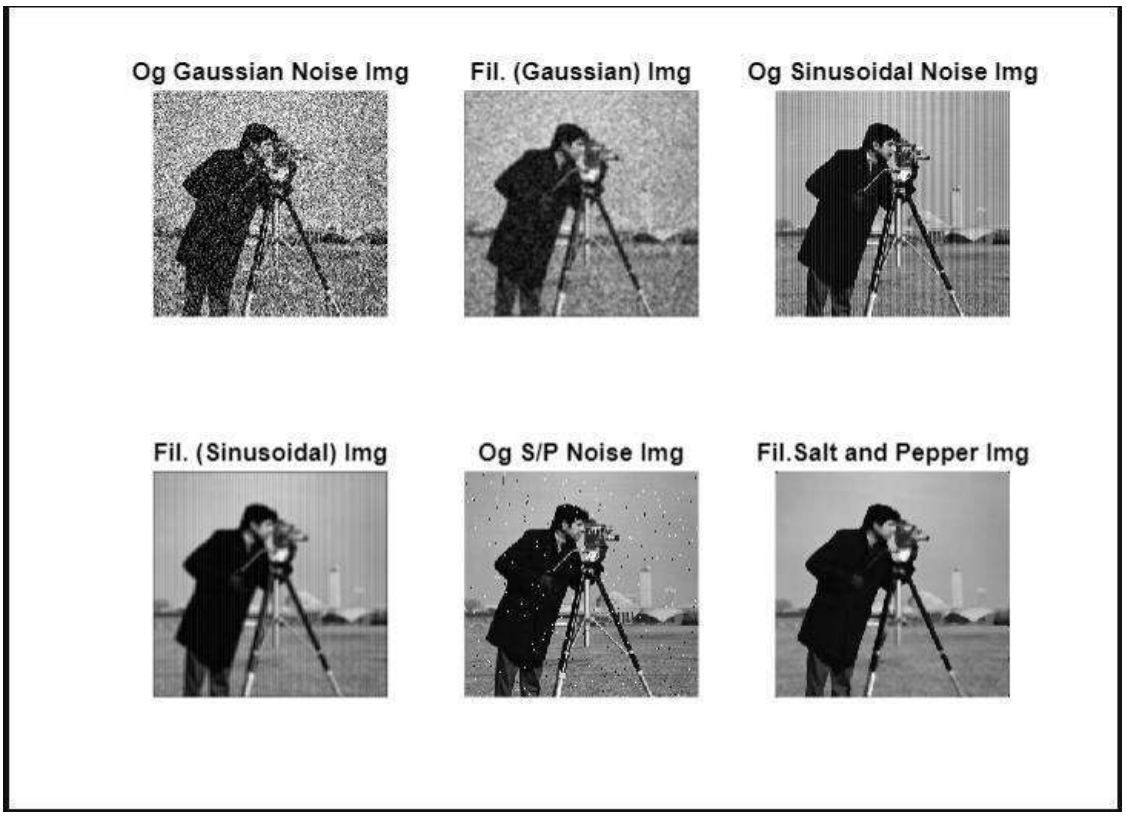
Program :-

```
% Read images
gauss_img = imread('gauss_img.tif');
noisy_img = imread('noisy_img.tif');
sp_img = imread('sp_img.tif');
% Define parameters
mask_size = 5; % Mask size for filters
sigma = 1.5; % Sigma for Gaussian filter
% Gaussian Filter
[x, y] = meshgrid(-floor(mask_size/2):floor(mask_size/2),  -floor(mask_size/2):floor(mask_size/2));
gaussian_mask = exp(-(x.^2 + y.^2) / (2 * sigma^2));
gaussian_mask = gaussian_mask / sum(gaussian_mask(:));
filtered_gaussian_image = conv2(gauss_img, gaussian_mask, 'same');
% Display Gaussian filter mask values
disp('Gaussian Filter Mask Values:');
disp(gaussian_mask);
% Sinusoidal Filter
% You need to define your sinusoidal filter. This could be a band-stop or
%notch filter
% I'll provide a simple example of a notch filter to suppress sinusoidal noise
sinusoidal_mask = ones(mask_size); % Simple notch filter
sinusoidal_mask(floor(mask_size/2), floor(mask_size/2)) = 0; % Suppress sinusoidal noise
filtered_sinusoidal_image = conv2(noisy_img, sinusoidal_mask, 'same');
% Display Sinusoidal filter mask values
disp('Sinusoidal Filter Mask Values:');
disp(sinusoidal_mask);
% Salt and Pepper Filter
% For salt and pepper noise, you can use a median filter
filtered_sp_image = medfilt2(sp_img, [mask_size mask_size]);
% Display Salt and Pepper filter mask values (not applicable)
% Display results
figure;
subplot(2, 3, 1);
imshow(gauss_img, []);
title('Og Gaussian Noise Img');
subplot(2, 3, 2);
imshow(filtered_gaussian_image, []);
title('Fil. (Gaussian) Img');
subplot(2, 3, 3);
imshow(noisy_img, []);
title('Og Sinusoidal Noise Img');
subplot(2, 3, 4);
imshow(filtered_sinusoidal_image, []);
title('Fil. (Sinusoidal) Img');
```

```
subplot(2, 3, 5);
imshow(sp_img, []);
title('Og S/P Noise Img');
subplot(2, 3, 6);
imshow(filtered_sp_image, []);
title('Fil.Salt and Pepper Img');
```

Output :

```
Gaussian Filter Mask Values:

    0.0144    0.0281    0.0351    0.0281    0.0144
    0.0281    0.0547    0.0683    0.0547    0.0281
    0.0351    0.0683    0.0853    0.0683    0.0351
    0.0281    0.0547    0.0683    0.0547    0.0281
    0.0144    0.0281    0.0351    0.0281    0.0144

Sinusoidal Filter Mask Values:

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



IPMV
EXPERIMENT 6

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	22 – 02 - 2024
Date of Submission	29 – 02 - 2024

Title : Apply the FT and Spatial domain filtering on image

Aim : Apply 2D FT and applying Ideal LPF and HPF on image in spatial domain filtering.

Software required :- MATLAB online

Algorithm:-

1. Clear workspace, close figures, and clear the command window.
2. Define simple binary image and compute the 2D fourier yransform if image.
3. Shift the zero-frequency component to center and display original image(im 3D also) and also display magnitude of fourier and shifted fourier transform.

For filtering in freqency domain:

4. Read an image and compute 2D fourier transform.
5. shift zero frequency component and set size of image
6. Initialize the LPF and HPF and define cut off frequncy and then Iterate over each pixel in the frequncy domain .
7. calculate the distance from the center of frequency domain and apply ideal LPF.
8. Apply LPF and HPF to shifted FT and also inverse FT on filter image.
9. Display images

Takeaway:

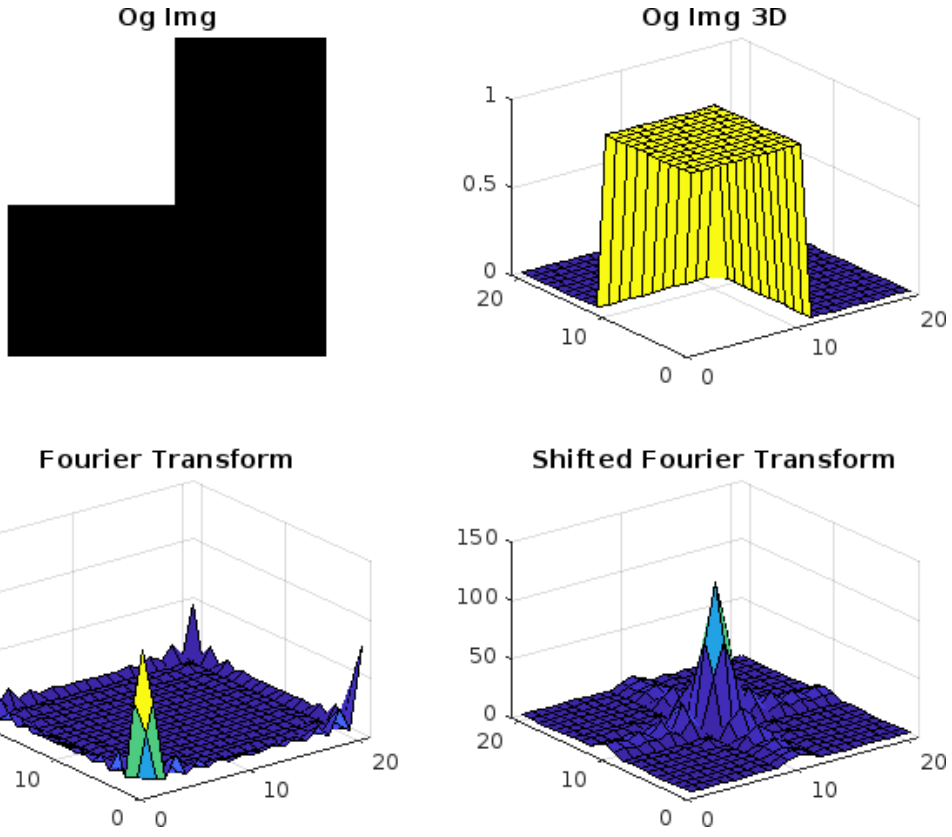
- In this experiment we do the process of applying ideal low-pass and high-pass filters to an image in the frequency domain using Fourier transforms, and then visualizing the results in both frequency and spatial domains.
- This technique is often used in image processing for tasks such as image enhancement, noise reduction, and feature extraction.
- **Ideal** filter have a sharp transition between bands
- Ideal filter are not directly implementable in real world application.

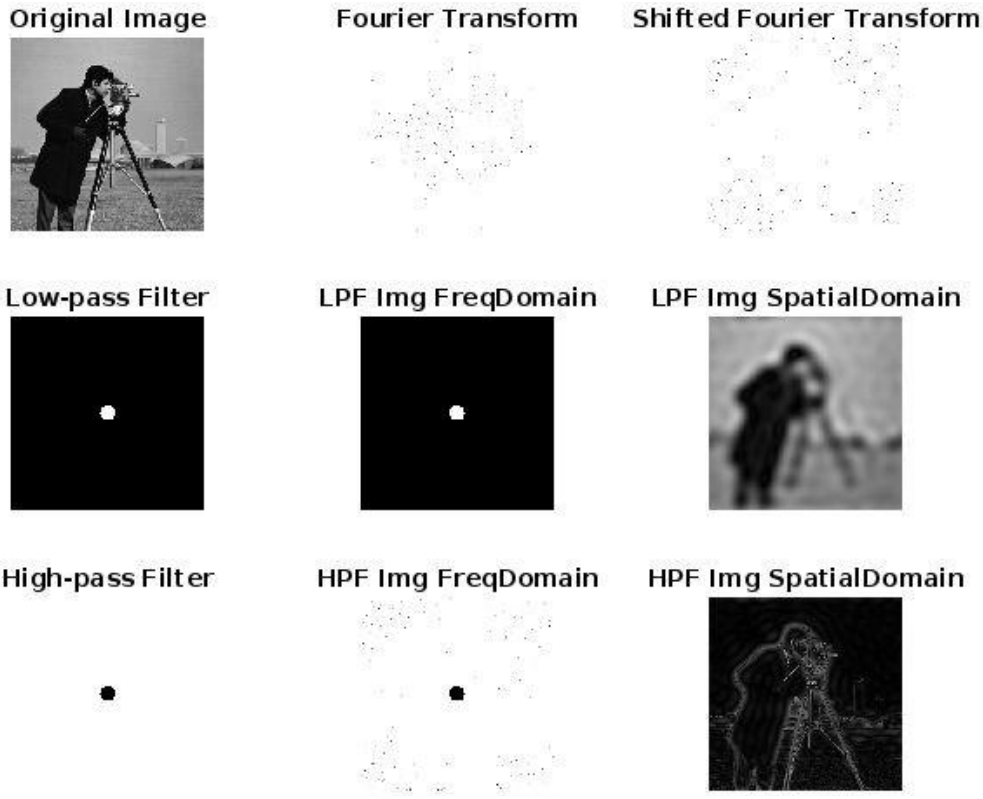
Code :

```
% Clear workspace, close all figures, and clear command window
clear; close all; clc;

% Define a simple binary image
i = [ones(11,11), zeros(11,10); zeros(10,21)];

% Compute the 2D Fourier Transform of the image
ft = fft2(i);

% Shift the zero-frequency component to the center
sft = fftshift(ft);

% Display the original image
subplot(2,2,1); imshow(i); title('Og Img');

% Display the original image in 3D
subplot(2,2,2); surf(i); title('Og Img 3D');

% Display the magnitude of the Fourier Transform
subplot(2,2,3); surf(abs(ft)); title('Fourier Transform');

% Display the magnitude of the shifted Fourier Transform
subplot(2,2,4); surf(abs(sft)); title('Shifted Fourier Transform');

% Task 1 complete

% Open a new figure
figure;

% Read an image (e.g., cameraman.tif)
img = imread("cameraman.tif");

% Compute the 2D Fourier Transform of the image
ftimg = fft2(img);

% Shift the zero-frequency component to the center
sftimg = fftshift(ftimg);

% Get the size of the image
[row, col] = size(img);

% Initialize low-pass and high-pass filters
lpm = zeros(row, col);
hpm = zeros(row, col);

% Define the cutoff frequency
d0 = 10;
```

```
% Iterate over each pixel in the frequency domain
for i = 1:1:row
    for j = 1:1:col
        % Calculate distance from the center of the frequency domain
        d = sqrt((i - row/2)^2 + (j - col/2)^2);
        % Apply ideal low-pass filter
        if d <= d0
            lpm(i,j) = 1;
            hpm(i,j) = 0;
        else
            lmp(i,j) = 0;
            hpm(i,j) = 1;
        end
    end
end

% Apply the low-pass filter to the shifted Fourier Transform
lfof = lpm .* sftimg;

% Apply the high-pass filter to the shifted Fourier Transform
hfof = hpm .* sftimg;

% Compute the inverse Fourier Transform of the filtered images
lfos = ifft2(lfof);
hfos = ifft2(hfof);

% Display the original image
subplot(3,3,1); imshow(img); title('Original Image');

% Display the magnitude of the Fourier Transform of the original image
subplot(3,3,2); imshow(uint8(abs(ftimg))); title('Fourier Transform');

% Display the magnitude of the shifted Fourier Transform of the original image
subplot(3,3,3); imshow(uint8(abs(sftimg))); title('Shifted Fourier
Transform');

% Display the low-pass filter
subplot(3,3,4); imshow(lpm); title('Low-pass Filter');

% Display the magnitude of the low-pass filtered image
subplot(3,3,5); imshow(uint8(abs(lfof))); title('LPF Img FreqDomain');

% Display the low-pass filtered image
subplot(3,3,6); imshow(uint8(abs(lfos))); title('LPF Img SpatialDomain');

% Display the high-pass filter
subplot(3,3,7); imshow(hpm); title('High-pass Filter');

% Display the magnitude of the high-pass filtered image
subplot(3,3,8); imshow(uint8(abs(hfof))); title('HPF Img FreqDomain');

% Display the high-pass filtered image
subplot(3,3,9); imshow(uint8(abs(hfos))); title('HPF Img SpatialDomain');
```

output :


Og Img
Og Img 3D
1
0.5
0
20
10
0
0
10
20
Fourier Transform
Shifted Fourier Transform
150
100
50
0
20
10
0
0
10
20



Published with MATLAB® R2023b

IPMV
EXPERIMENT 7

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	22 – 02 - 2024
Date of Submission	29 – 02 - 2024

Title : Apply the FT and Spatial domain filtering on image

Aim : Applying Butterworth ,Guassian LPF and HPF on image in spatial domain filtering.

Software required :- MATLAB online

Algorithm:-

1. Clear workspace, close figures, and clear the command window.
2. Define simple binary image and compute the 2D fourier yransform if image.
3. Shift the zero-frequency component to center and display original image(im 3D also) and also display magnitude of fourier and shifted fourier transform.

For filtering in freqency domain:

4. Read an image and compute 2D fourier transform.
5. shift zero frequency component and set size of image
6. Initialize the LPF and HPF and define cut off frequncy and then Iterate over each pixel in the frequncy domain .
7. calculate the distance from the center of frequency domain and apply ideal LPF.
8. Apply LPF and HPF to shifted FT and also inverse FT on filter image.
9. Display images

Takeaway:

- In this experiment we do the process of applying Butterworth ,guassian low-pass and high-pass filters to an image in the frequency domain using Fourier transforms, and then visualizing the results in both frequency and spatial domains.
- This technique is often used in image processing for tasks such as image enhancement, noise reduction, and feature extraction.
- Butterworth filter are commonly used in application where a smooth frequency response is desired .
- Gaussian filter are widly used for noise reduction ,image blurring and prossesing task.
- Gaussian filter have a finite support in spatial domain,which means they computationally efficient.

Code :

```
% Clear workspace, close all figures, and clear command window
clear; close all; clc;

figure;

% Read an image (e.g., cameraman.tif)
img = imread("cameraman.tif");

% Compute the 2D Fourier Transform of the image
ftimg = fft2(img);

% Shift the zero-frequency component to the center
sftimg = fftshift(ftimg);

% Get the size of the image
[row, col] = size(img);

% Initialize low-pass and high-pass filters
lpm = zeros(row, col);
hpm = zeros(row, col);

% Define the cutoff frequency and filter order
d0 = 10;
n = 5; % Filter order

% Iterate over each pixel in the frequency domain
for i = 1:1:row
    for j = 1:1:col
        % Calculate distance from the center of the frequency domain
        d = sqrt((i - row/2)^2 + (j - col/2)^2);
        % Apply Butterworth low-pass filter
        lpm(i,j) = 1 / (1 + (d/d0)^(2*n));
        % Apply Butterworth high-pass filter (complementary to low-pass)
        hpm(i,j) = 1 - lpm(i,j);
    end
end

% Apply the low-pass filter to the shifted Fourier Transform
lfof = lpm .* sftimg;

% Apply the high-pass filter to the shifted Fourier Transform
hfof = hpm .* sftimg;

% Compute the inverse Fourier Transform of the filtered images
lfos = ifft2(lfof);
hfos = ifft2(hfof);
```

```
% Display the original image
subplot(3,3,1); imshow(img); title('Og Image');

% Display the magnitude of the Fourier Transform of the original image
subplot(3,3,2); imshow(uint8(abs(ftimg))); title('Fourier Transform');

% Display the magnitude of the shifted Fourier Transform of the original image
subplot(3,3,3); imshow(uint8(abs(sftimg))); title('Shifted Fourier
Transform');

% Display the low-pass filter
subplot(3,3,4); imshow(lpm); title('Butterworth LPF');

% Display the magnitude of the low-pass filtered image
subplot(3,3,5); imshow(uint8(abs(lfof))); title('LPF Img FreqDomain');

% Display the low-pass filtered image
subplot(3,3,6); imshow(uint8(abs(lfos))); title('LPF Img SpatialDomain');

% Display the high-pass filter
subplot(3,3,7); imshow(hpm); title('Butterworth HPF');

% Display the magnitude of the high-pass filtered image
subplot(3,3,8); imshow(uint8(abs(hfof))); title('HPF Img FreqDomain');

% Display the high-pass filtered image
subplot(3,3,9); imshow(uint8(abs(hfos))); title('HPF Img SpatialDomain');

figure;

% Compute the 2D Fourier Transform of the image
ftimg1 = fft2(img);

% Shift the zero-frequency component to the center
sftimg1 = fftshift(ftimg1);

% Get the size of the image
[row, col] = size(img);

% Initialize low-pass and high-pass filters
lpm1 = zeros(row, col);
hpm1 = zeros(row, col);

% Define the cutoff frequency and filter order
d01 = 10;
n1 = 5; % Filter order

% Iterate over each pixel in the frequency domain
for i1 = 1:1:row
    for j1 = 1:1:col
        % Calculate distance from the center of the frequency domain
        d1 = sqrt((i1 - row/2)^2 + (j1 - col/2)^2);
        % Apply Butterworth low-pass filter
        lpm1(i1,j1) = exp(-d1^(2)/2d01^(2));
```

```
        % Apply Butterworth high-pass filter (complementary to low-pass)
        hpm1(i1,j1) = 1 - lpm(i1,j1);
    end
end

% Apply the low-pass filter to the shifted Fourier Transform
lfof1 = lpm1 .* sftimg1;

% Apply the high-pass filter to the shifted Fourier Transform
hfof1 = hpm1 .* sftimg1;

% Compute the inverse Fourier Transform of the filtered images
lfos1 = ifft2(lfof1);
hfos1 = ifft2(hfof1);

% Display the original image
subplot(3,3,1); imshow(img); title('Og Image');

% Display the magnitude of the Fourier Transform of the original image
subplot(3,3,2); imshow(uint8(abs(ftimg1))); title('Fourier Transform');

% Display the magnitude of the shifted Fourier Transform of the original image
subplot(3,3,3); imshow(uint8(abs(sftimg1))); title('Shifted Fourier
Transform');

% Display the low-pass filter
subplot(3,3,4); imshow(lpm1); title('Guassian LPF');

% Display the magnitude of the low-pass filtered image
subplot(3,3,5); imshow(uint8(abs(lfof1))); title('LPF Img FreqDomain');

% Display the low-pass filtered image
subplot(3,3,6); imshow(uint8(abs(lfos1))); title('LPF Img SpatialDomain');

% Display the high-pass filter
subplot(3,3,7); imshow(hpm1); title('Guassian HPF');

% Display the magnitude of the high-pass filtered image
subplot(3,3,8); imshow(uint8(abs(hfof1))); title('HPF Img FreqDomain');

% Display the high-pass filtered image
subplot(3,3,9); imshow(uint8(abs(hfos1))); title('HPF Img SpatialDomain');
```

output :

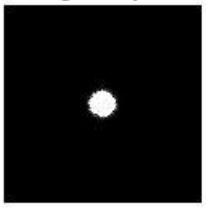
Og Image

Fourier Transform

Shifted Fourier Transform

Butterworth LPF

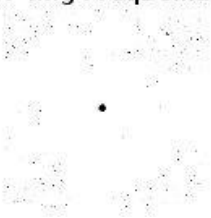
LPF Img FreqDomain



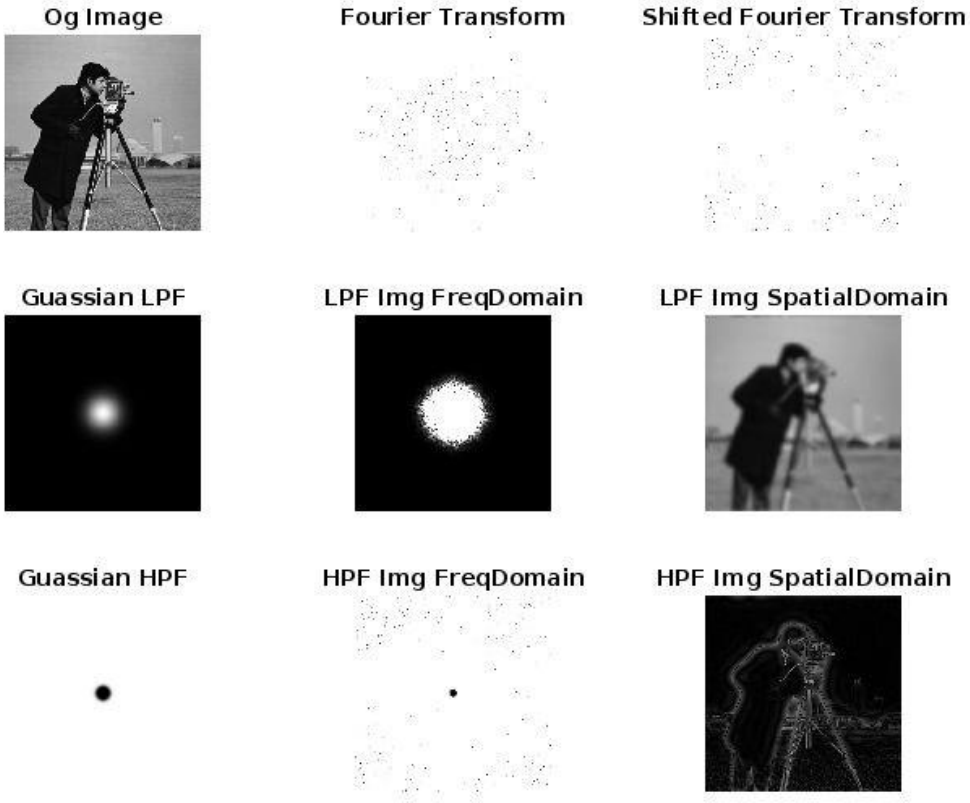
LPF Img SpatialDomain

Butterworth HPF

HPF Img FreqDomain



HPF Img SpatialDomain



Published with MATLAB® R2023b

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

Department of Electronics and Telecommunication Engineering

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE-EXTC B
Date of performance	07/03/2024
Date of Submission	14/03/2024
LaboratoryTeacher	Prof.Amit Maurya

Title: Morphological Image Processing: Erosion, Dilation, Opening, Closing, and Boundary Extraction

Aim:

The aim of this experiment is to explore and understand fundamental morphological image processing operations, including erosion, dilation, opening, and closing. Additionally, the experiment aims to demonstrate the extraction of different types of boundaries from binary images.

Software Requirement: MATLAB Online

Algorithm:

- Initialization:
- Read and preprocess the input image.
- Erosion and Dilation:
- Apply erosion and dilation operations using different structuring elements.
- Display the results.
- Opening and Closing:
- Perform opening and closing operations using erosion and dilation.
- Display the results.
- Boundary Extraction:
- Extract inner, outer, and overall boundaries.

- Display the boundary images.
- Thinning and Thickening:
- Apply thinning and thickening operations to the binary image.
- Display the results.
- Hit and Miss Transformation (Optional):

Takeaway:

1. Understanding basic morphological operations like erosion, dilation, opening, and closing.
2. Extracting boundaries from binary images for analysis.
3. Exploring advanced operations such as thinning and thickening.
4. Optionally, applying Hit-and-Miss Transformation for pattern detection.
5. Gaining practical experience in MATLAB for image processing tasks

```
clear; close all; clc;

I1=imread("dineshk.png");
I2=rgb2gray(I1);
img=imbinarize(I2);

s1=[1,1,1,1,1,1,1,1,1,1,1];%structure element
s2=[0,1,0;1,1,1;0,1,0];
s3= strel('disk',10);


e1=imerode(img,s1); % % Inbuilt command for erosion
e2=imerode(img,s2);
e3=imerode(img,s3);


subplot(2,3,1); imshow(img); title('Original')
subplot(2,3,2); imshow(e1);  title('Erode Line Element')
subplot(2,3,3); imshow(e2); title('Erode Plus Element')
subplot(2,3,4); imshow(e3); title('Erode Circle Element')


figure;
d1=imdilate(img,s1); % Inbuilt command for dilation
d2=imdilate(img,s2);
d3=imdilate(img,s3);

subplot(2,3,1); imshow(img); title('Original')
subplot(2,3,2); imshow(d1); title('Dilate Line Element')
subplot(2,3,3); imshow(d2); title('Dilate Plus Element')
subplot(2,3,4); imshow(d3); title('Dilate Circle Element')

figure;
%Opening
o1=imdilate(e1,s1); % Opening using dilation
o2=imdilate(e2,s2);
o3=imdilate(e3,s3);
% Opening using inbuilt function
oi1=imopen(img,s1);
oi2=imopen(img,s2);
oi3=imopen(img,s3);
subplot(2,4,1); imshow(img); title('Original')
subplot(2,4,2); imshow(o1);  title('Open1')
subplot(2,4,3); imshow(o2);  title('Open2')
subplot(2,4,4); imshow(o3);  title('Open3')
subplot(2,4,5); imshow(oi1);   title('Open Inbuilt 1')
subplot(2,4,6); imshow(oi2);   title('Open Inbuilt 2')
subplot(2,4,7); imshow(oi3);   title('Open Inbuilt 3')

figure;
% Closing
c1 = imerode(d1,s1); % Closing using erosion
```

```
c2 = imerode(d2,s2);
c3 = imerode(d3,s3);

ci1=imclose(img,s1);% Closing using inbuilt function
ci2=imclose(img,s2);
ci3=imclose(img,s3);

subplot(2,4,1); imshow(img); title('Original')
subplot(2,4,2); imshow(c1); title('Close1')
subplot(2,4,3); imshow(c2); title('Close2')
subplot(2,4,4); imshow(c3); title('Close3')
subplot(2,4,5); imshow(ci1);   title('Close Inbuilt 1')
subplot(2,4,6); imshow(ci2);   title('Close Inbuilt 2')
subplot(2,4,7); imshow(ci3);   title('Close Inbuilt 3')

%%boundary extraction
figure;
ob =d2-img;
ib =img-e2;
ovb =d2-e2;

subplot(2,3,1); imshow(ob); title("outer boundary");
subplot(2,3,2); imshow(ib); title("inner boundary");
subplot(2,3,3); imshow(ovb); title("overall boundary");

%Thinning of Image
ThinImag = bwmorph(img, 'thin');
%Thickening of Image
ThickImag = bwmorph(img, 'thicken');

figure;
subplot(2,3,1); imshow(img); title('Original')
subplot(2,3,2); imshow(ThinImag); title("Thinned Image")
subplot(2,3,3); imshow(ThickImag); title("Thickened Image")
%Hit and Miss Transformation
% Define the Hit-and-Miss structuring element
hit_miss_element = [0, 1, 0; 1, 1, 1; 0, 1, 0];
% Perform the Hit-and-Miss Transformation
hit_miss_result = bwhitmiss(img, hit_miss_element);

figure;
subplot(2,3,1); imshow(img); title('Original')
subplot(2,3,2);
imshow(hit_miss_result);
title('Hit and Miss Transformation');
% Display the structuring element
subplot(2,3,3);
imshow(hit_miss_element, 'InitialMagnification', 'fit');
title('Structuring Element');
```

Original

Erode Line Element

Erode Plus Element

Erode Circle Element

Original

Dilate Line Element

Dilate Plus Element

Dilate Circle Element

Original

Open1

Open2

Open3

Open Inbuilt 1

Open Inbuilt 2

Open Inbuilt 3

Original

Close1

Close2

Close3

Close Inbuilt 1

Close Inbuilt 2

Close Inbuilt 3

outer boundary

inner boundary

overall boundary

Original

Thinned Image

Thickened Image

Published with MATLAB® R2023b

IPMV EXPERIMENT 9

Name	Dinesh Dilip Kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	14-03-2024
Date of Submission	21-03-2024

Title :- Image segmentation

Aim : Perform segmentation on MRI image, extract the image in different segments and display extracted part.

Software required :- MATLAB online

Algorithm:-

- Read the input MRI image in grayscale.
- Convert the grayscale image to a binary image.
- Define a disk-shaped structuring element for morphological operations.
- Perform morphological opening on the binary image to remove noise and smooth the edges.
- Label connected components in the binary image to identify distinct regions.
- Keep only the region labeled as ‘number’ and set all other regions to zero.
- Define another disk-shaped structuring element for morphological closing to further refine the selected region.
- Perform morphological closing on the selected region to fill any small holes and gaps.
- Multiply the binary segmented region with the original grayscale image to extract the desired part.
- Display the original image and the extracted part side by side for visualization.

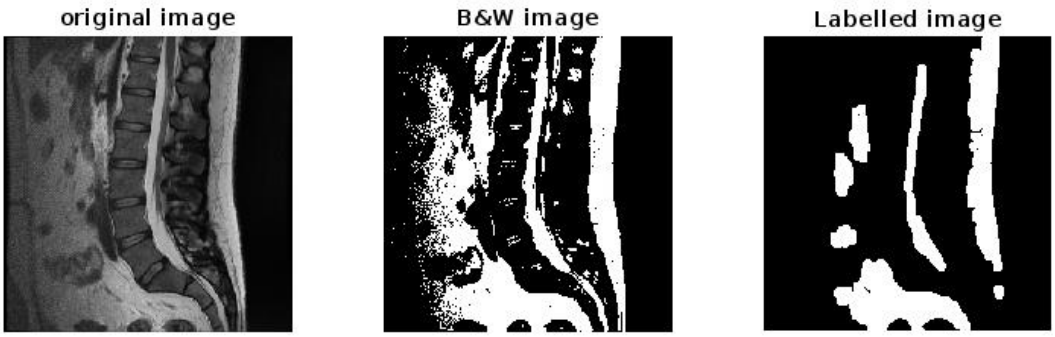
Takeaway :-

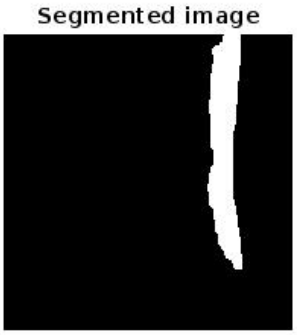
- Morphological operations like opening and closing are applied to the binary image using structuring elements. These operations help in smoothing, refining, and segmenting regions of interest based on their shape and connectivity.
- Labeling connected components facilitates precise region identification and extraction, enabling focused analysis.
- By isolating and extracting specific regions of interest, targeted analysis or feature extraction becomes feasible, improving diagnostic or recognition accuracy.
- Fine-tuning parameters like structuring element size and labeling thresholds significantly influences segmentation accuracy, requiring careful adjustment.
- Utilizing visualization techniques enhances understanding and validation of image processing outcomes, enabling qualitative assessment.

Program :-

```
%image segmentation
clear; close all; clc;

img = imread("Spine MRI.jpeg"); %Gray img of MRI

% img1= im2gray(img);

img2= im2bw(img, graythresh(img)); %Binary image

se= strel('disk',10); %structure element

k2=imopen(img2,se); %morphological opening

b=bwlabel(k2); %label

b(b~=6)=0; %keep only the region labeled as 6 and set other zero

se2=strel('disk',18); %structure element

k3=imclose(b,se2); %morphological opening

%multiply the binary segmented region with the original grayscale image
k4=k3.*double(img);

%display result
subplot(2,3,1); imshow(img); title("original image");
subplot(2,3,2); imshow(img2); title("B&W image");
subplot(2,3,3); imshow(k2); title("Labelled image");
subplot(2,3,4); imshow(k3); title("Segmented image");

figure;
subplot(2,2,1); imshow(img); title("original image");
subplot(2,2,2); imshow(k4,[]); title("Extracted image");
```

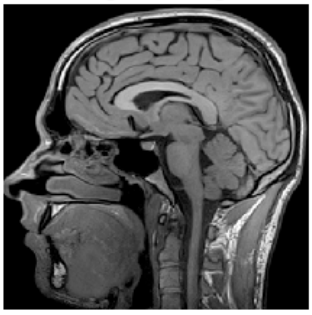
Output :


original image
B&W image
Labelled image


Segmented image


original image
Extracted image

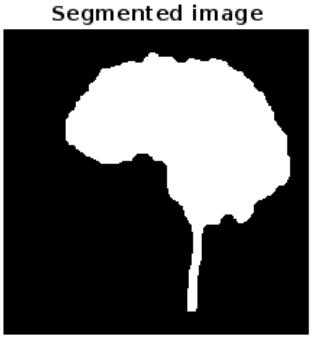
original image



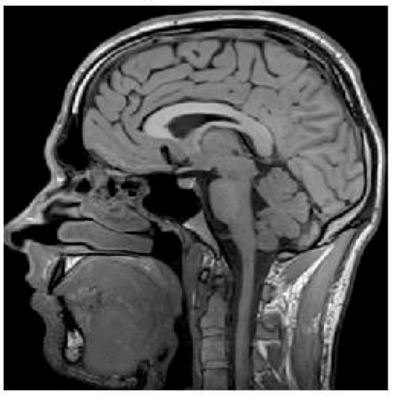
B&W image

Labelled image

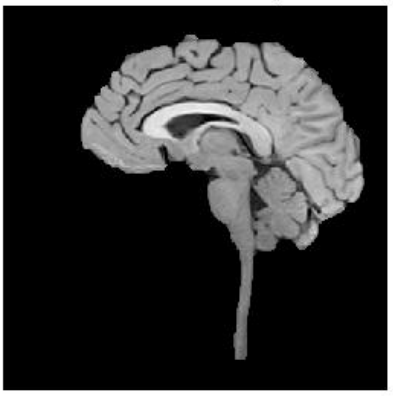
Segmented image



original image



Extracted image



IPMV
EXPERIMENT 10

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	21 – 03 - 2024
Date of Submission	28 – 03 - 2024
Subject	Image Processing Lab
Prof. In-charge	Prof. Amit Maurya

Title : Analysis of Edge Detection Operators.

Aim : To evaluate and compare the effectiveness of various edge detection operators.

Software required :- MATLAB online

Algorithm:-

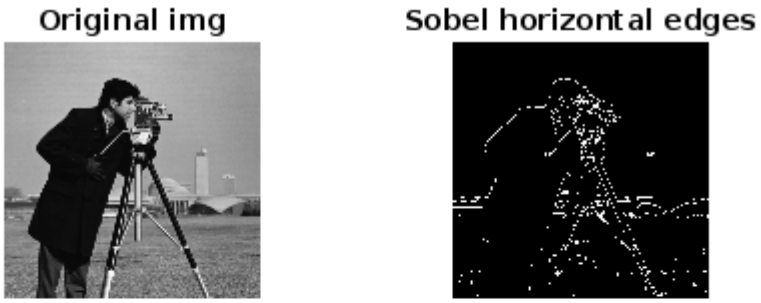
1. Read the original image
2. Apply Sobel operator to detect horizontal edges and display the result.
3. Apply Sobel operator to detect vertical edges and display the result.
4. Apply Sobel operator to detect both horizontal and vertical edges and display the result.
5. Apply Roberts operator to detect both horizontal and vertical edges and display the result.
6. Apply Prewitt operator to detect both horizontal and vertical edges and display the result.
7. Apply Laplacian filter to the image and display the result.
8. Apply Laplacian of Gaussian (LOG) edge detection with a sigma value of 3 to the image and display the result.
9. Apply Canny edge detection to the image and display the result.

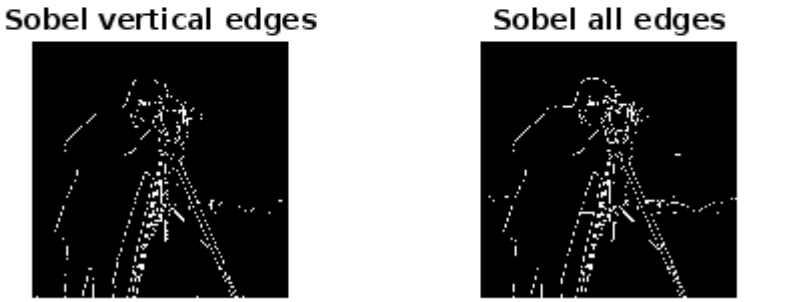
Takeaway:

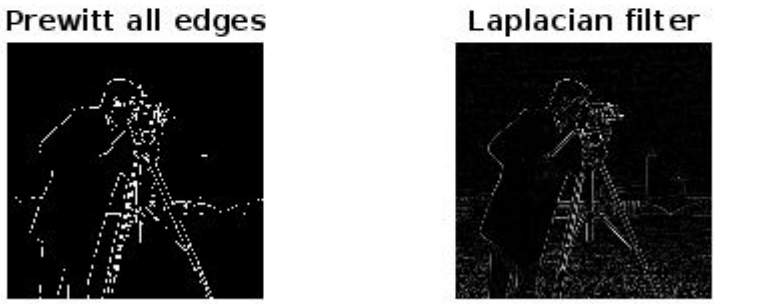
- Explored various edge detection techniques including Sobel, Roberts, Prewitt, Laplacian filter, LOG, and Canny.
- Visualized edge detection results for horizontal, vertical, and combined edges.
- Recognized the impact of parameter choices, like sigma value in LOG, on edge detection quality.
- Considered factors such as noise level and computational complexity for selecting suitable techniques.
- Evaluated trade-offs between sensitivity, specificity, and computational efficiency of each technique.

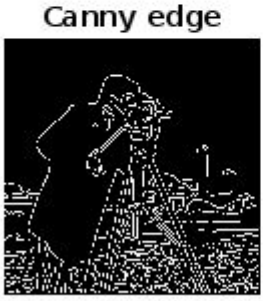
```
clear; close all; clc;

% Read the colour image and convert it to a grey scale image
% Display the original image
% rgb_img = imread("flower.jpg");
% img= rgb2gray (rgb_img);

img=imread("cameraman.tif");

% Apply Sobel operator
% Display only the horizontal edges
sobelhz = edge (img, 'sobel', 'horizontal');

% Apply Sobel operator
% Display only the vertical edges
sobelvrt = edge (img, 'sobel', 'vertical');

% Apply Sobel operator
% Display both horizontal and vertical edges
sobelvrthz = edge (img, 'sobel', 'both');

% Apply Roberts operator
% Display both horizontal and vertical edges
robertsedg= edge(img, 'roberts');

% Apply Prewitt operator
% Display both horizontal and vertical edges
robertsedg = edge (img, 'prewitt');

% Apply Laplacian filter
f = fspecial('laplacian');
lapedg = imfilter (img, f, 'symmetric');

% Apply LOG edge detection
% The sigma used is 3
f = fspecial('log', [15, 15], 3.0);
logedg1 = edge(img, 'zerocross', [], f);

% Apply Canny edge detection
cannyedg = edge(img, 'canny');

%Displaying Resultant Images
subplot(2, 2, 1); imshow(img); title('Original img');
subplot(2, 2, 2); imshow(sobelhz, []); title('Sobel horizontal edges');
subplot(2, 2, 3); imshow(sobelvrt, []); title('Sobel vertical edges');
subplot(2, 2, 4); imshow(sobelvrthz, []); title('Sobel all edges');

figure;
subplot(2, 2, 1); imshow(img); title('Original img');
subplot(2, 2, 2); imshow(robertsedg, []); title('Roberts all edges');
subplot(2, 2, 3); imshow(robertsedg, []); title('Prewitt all edges');
subplot(2, 2, 4); imshow(lapedg, []); title('Laplacian filter');
```

```
figure;
subplot(2, 2, 1); imshow(img); title('Original img');
subplot(2, 2, 2); imshow(logedg1); title('Log with sigma 3');
subplot(2, 2, 3); imshow(cannyedg, []); title('Canny edge');
```









Published with MATLAB® R2024a

IPMV
EXPERIMENT 11

Name	Dinesh kakade
Roll no	22104B2010
Batch	4
Class	TE EXTC B
Date of Performance	28 – 03 - 2024
Date of Submission	04 – 03 - 2024
Subject	Image Processing Lab
Prof. In-charge	Prof. Amit Maurya

Title : Feature extraction

Aim : Perform the operation on feature extraction wavelet transform function.

Software required :- MATLAB online

Algorithm:-

1. Load the image
2. Choose a Wavelet.
3. Perform 2D Discrete Wavelet Transform (DWT).
4. Apply Feature Extraction.
5. Compute energy of coefficients at each level
6. Extract coefficients corresponding to the current level
7. Display the extracted features.
8. Combine coefficients into a single matrix
9. Display the coefficients as an image

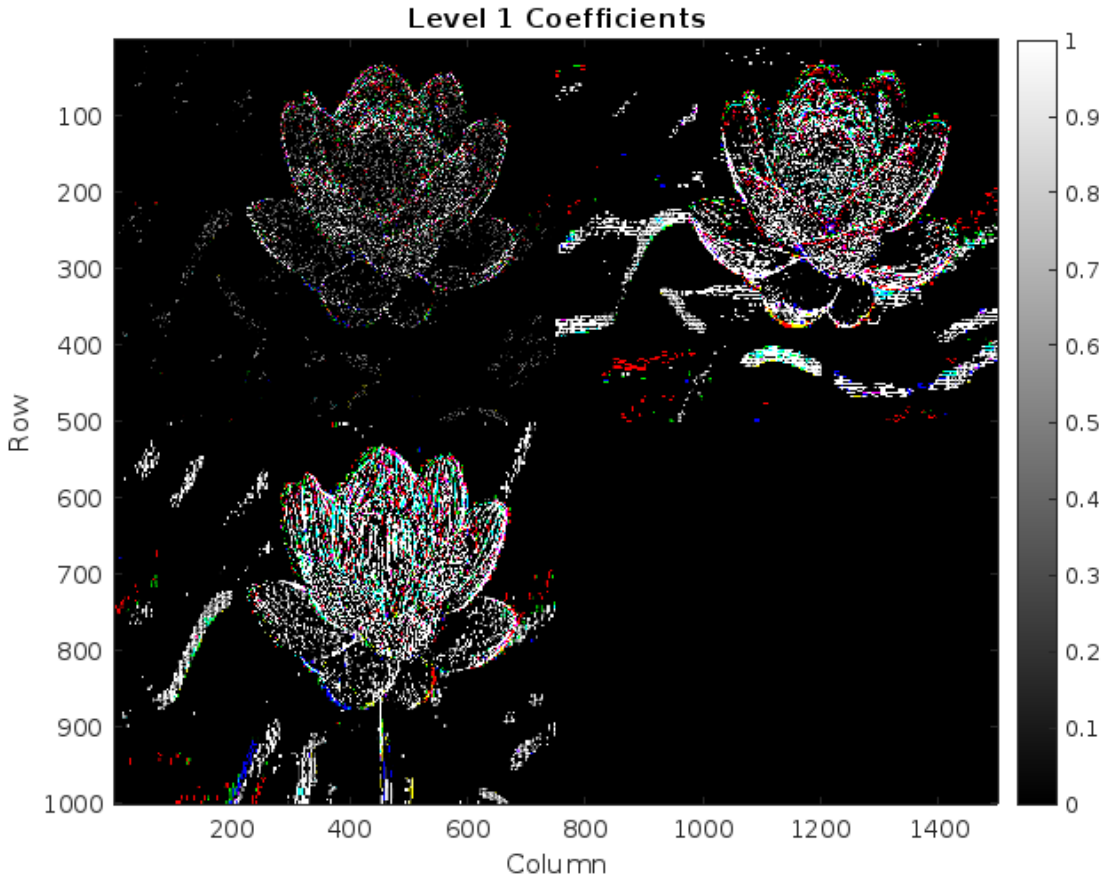
Takeaway:

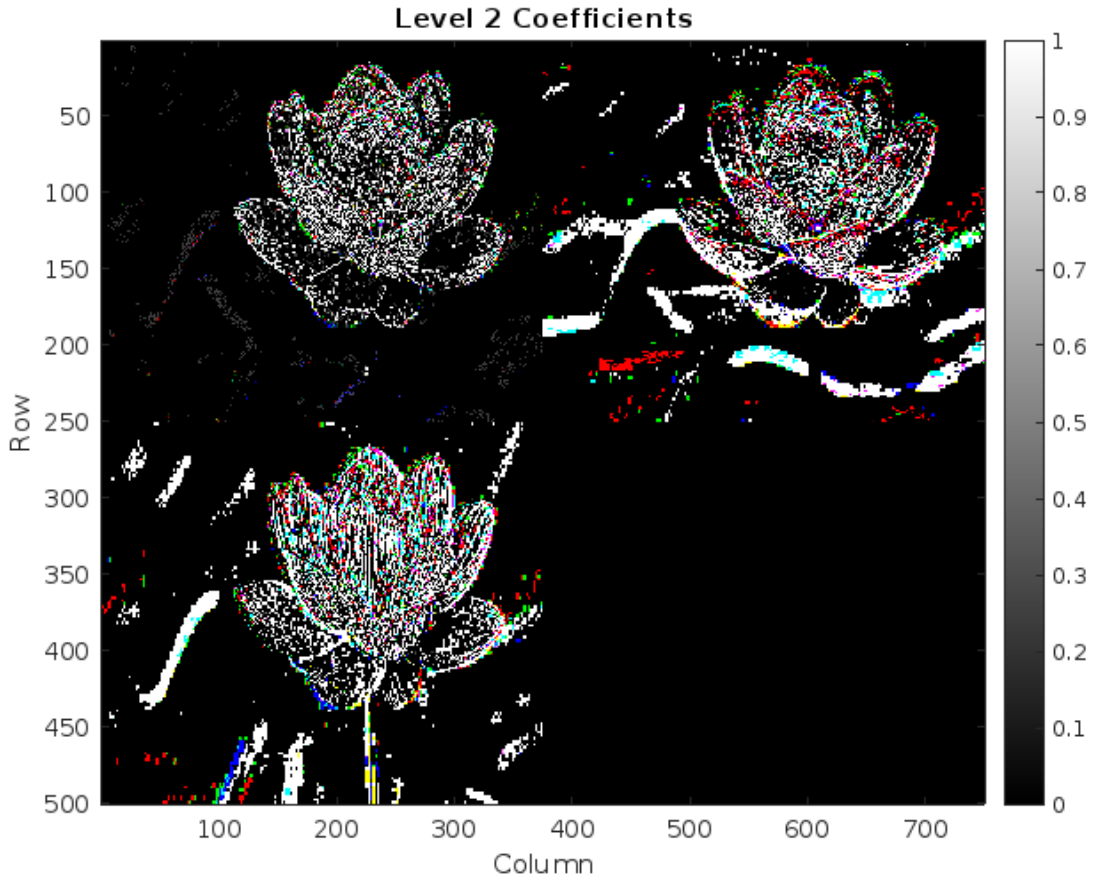
- DWT and feature extraction from wavelet coefficients find applications in various fields such as image compression, denoising, texture analysis, and image classification.
- The extracted features can be used as input to machine learning algorithms for tasks like image classification, object recognition, or anomaly detection.
- Feature extraction is a crucial step in image analysis and pattern recognition tasks as it helps in representing the image data in a more compact and informative manner.
- DWT is a powerful technique for decomposing signals or images into different frequency components, which can be useful for various image processing tasks.
- Visualizing the coefficients can aid in understanding how the image content is distributed across different frequency bands.

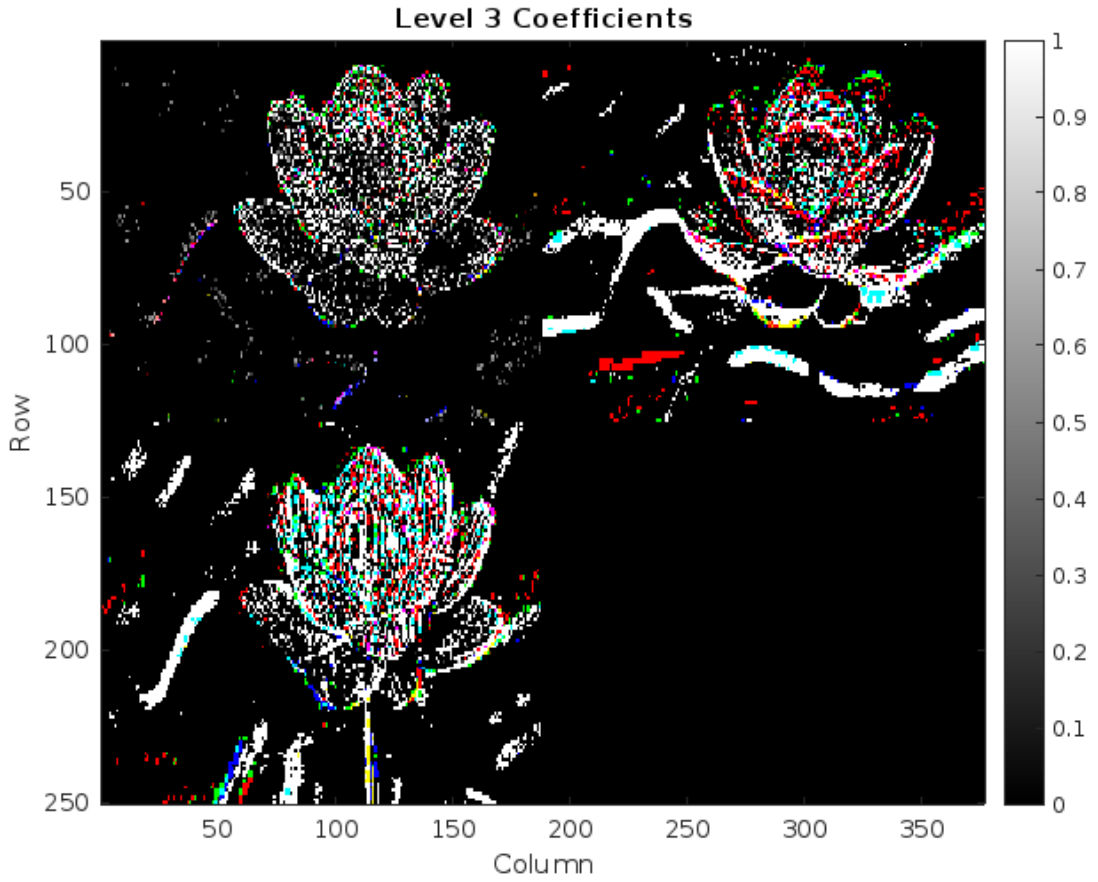
```
image = imread('flower.jpg'); % Load your image
imshow(image); title("original image");

wavelet_name = 'db1'; % Choose a wavelet, e.g., 'db4'

% Perform 2D Discrete Wavelet Transform (DWT)
level = 4; % Set the decomposition level
[C, S] = wavedec2(image, level, wavelet_name); % Perform 2D DWT

% Feature Extraction
% Example: Compute energy of coefficients at each level
features = [];
for i = 1:level
    % Extract coefficients corresponding to the current level
    [det_coeff_row, det_coeff_col, app_coeff] = detcoef2('all', C, S, i);
    % Compute energy of coefficients
    energy_row = sum(det_coeff_row(:).^2);
    energy_col = sum(det_coeff_col(:).^2);
    energy_app = sum(app_coeff(:).^2);
    features = [features, energy_row, energy_col, energy_app];
end

% Display the extracted features
disp('Extracted Features:');
disp(features);

% Visualize each level of coefficients
for i = 1:level
    % Extract coefficients corresponding to the current level
    [det_coeff_row, det_coeff_col, app_coeff] = detcoef2('all', C, S, i);

    % Combine coefficients into a single matrix
    coeff_mat = [app_coeff, det_coeff_row; det_coeff_col,
zeros(size(det_coeff_col))];

    % Display the coefficients as an image
    figure;
    imagesc(coeff_mat);
    colormap gray;
    title(['Level ' num2str(i) ' Coefficients']);
    xlabel('Column');
    ylabel('Row');
    colorbar;
end
```

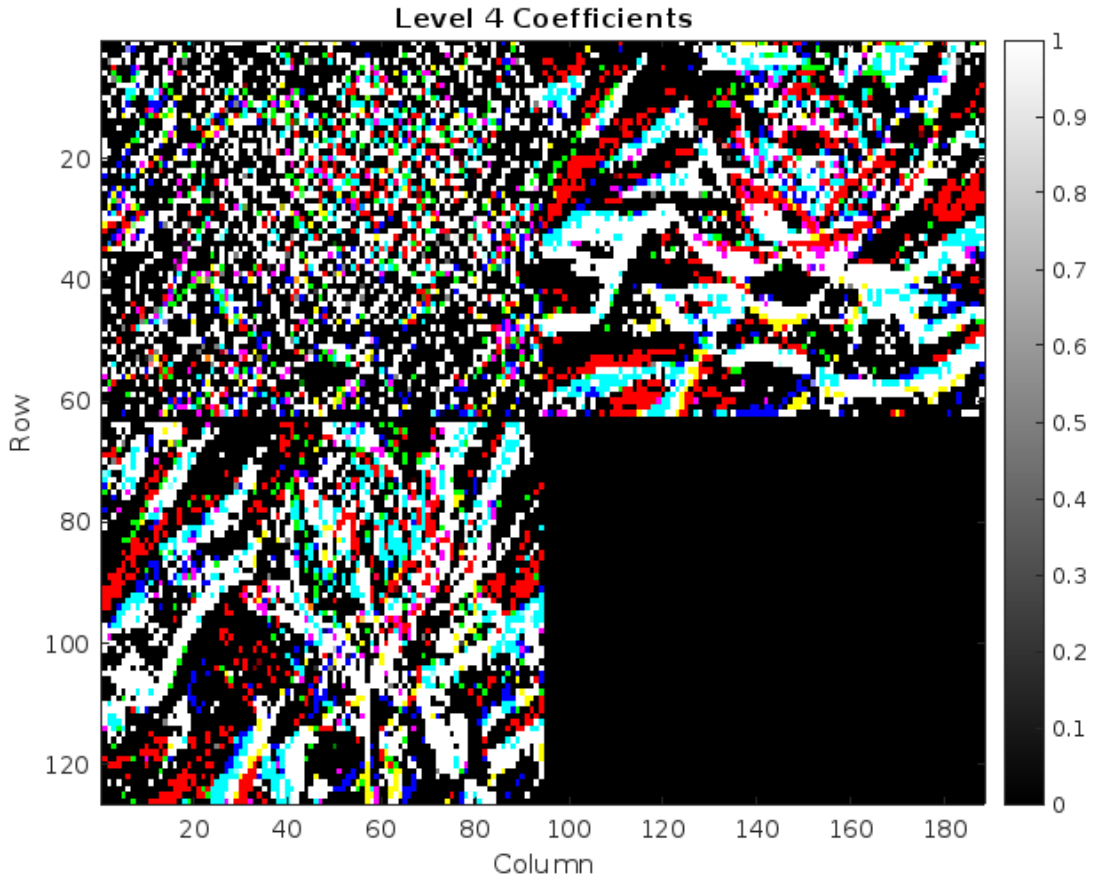
```
Extracted Features:
   1.0e+08 *

  Columns 1 through 7

    0.0765    0.0717    0.0029    0.2299    0.2230    0.0274    0.5371

  Columns 8 through 12
```

```
0.5003    0.0879    1.1493    0.9883    0.2188
```


Level 1 Coefficients
Row
Column
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
1400
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Level 2 Coefficients
Row
Column
50
100
150
200
250
300
350
400
450
500
100
200
300
400
500
600
700
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Level 3 Coefficients
Row
Column
50
100
150
200
250
300
350
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



Published with MATLAB® R2024a

MathWorks | Training Services

Course Completion Certificate

Dinesh kakade

has successfully completed **100%** of the self-paced training course

Image Processing Onramp

DIRECTOR, TRAINING SERVICES

18 January 2024

MathWorks | Training Services

Course Completion Certificate

Dinesh Kakade

has successfully completed **100%** of the self-paced training course

Image Processing with MATLAB

DIRECTOR, TRAINING SERVICES

1 February 2024