A logo for a university

AI-generated content may be incorrect.

**Lab Report 5 & 6**

**Digital Image Processing**

**CSE438**

**Section:** 03

**Semester:** Spring-2025

**Submitted To:**

**Md. Asif Khan Rifat**

**Lecturer**

Department of Computer Science

and Engineering

**Submitted By:**

                                                                         Suddip Paul Arnab

**2022-1-60-356**

**Date of submission:** 7 May 2025

1. **Apply** Fourier transform to transform any image (above) from the spatial domain to the frequency domain. Apply inverse Fourier transform to transform the image from the frequency domain to the spatial domain.

original\_img = imread('Picture1.jpg');

gray\_img = rgb2gray(original\_img);

gray\_img = im2double(gray\_img);

% Fourier Transform

F = fft2(gray\_img); % Apply 2D Fourier Transform

F\_shifted = fftshift(F); % Shift zero-frequency to the center

% Magnitude Spectrum for visualization

magnitude\_spectrum = log(1 + abs(F\_shifted));

% Inverse Fourier Transform

F\_ishifted = ifftshift(F\_shifted); % Inverse shift

reconstructed\_img = ifft2(F\_ishifted);% Apply Inverse Fourier Transform

reconstructed\_img = abs(reconstructed\_img); % Take magnitude (real part)

% Display Results

figure('Name','Fourier Transform and Inverse', 'NumberTitle','off');

subplot(1,3,1), imshow(gray\_img, []), title('Original Image');

subplot(1,3,2), imshow(magnitude\_spectrum, []), title('Magnitude Spectrum');

subplot(1,3,3), imshow(reconstructed\_img, []), title('Reconstructed Image');

**A close-up of a person's face

AI-generated content may be incorrect.**

1. **Apply** three types of high pass filtering in the frequency domain in **Figure 1** andfind out which one is better to produce the enhanced image (sharpen) for the given image (output must show all steps as shown in **Figure 2**).

i. Ideal high pass filter (IHPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

n = 2;

IHPF = double(D > D0);

F\_filtered = F\_shifted .\* IHPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

figure("Name","Ideal high pass filter","NumberTitle","on");

sgtitle('Ideal high pass filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(IHPF, []), xlabel('(c) Ideal high pass filter', 'FontSize', 12);

subplot(2,3,4), mesh(IHPF), xlabel('(d) Ideal high pass filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

A collage of images of a light source

AI-generated content may be incorrect.

ii. Butterworth high pass filter (BHPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

n = 2;

BHPF = 1 ./ (1 + (D0 ./ D).^(2\*n));

F\_filtered = F\_shifted .\* BHPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

figure("Name","Butterworth high pass filter","NumberTitle","on");

sgtitle('Butterworth high pass filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(BHPF, []), xlabel('(c) Butterworth high pass filter', 'FontSize', 12);

subplot(2,3,4), mesh(BHPF), xlabel('(d) Butterworth high pass filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

A collage of images of a brain

AI-generated content may be incorrect.

iii. Gaussian high pass filter (GHPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

n = 2;

GHPF = 1 - exp(-(D.^2) / (2\*(D0^2)));

F\_filtered = F\_shifted .\* GHPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

figure("Name","Gaussian high pass filter","NumberTitle","on");

sgtitle('Gaussian high pass filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(GHPF, []), xlabel('(c) Gaussian high pass filter', 'FontSize', 12);

subplot(2,3,4), mesh(GHPF), xlabel('(d) Gaussian high pass filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

A collage of images of a human head

AI-generated content may be incorrect.

Among the three high pass filters applied—Ideal, Butterworth, and Gaussian—the Gaussian High Pass Filter (GHPF) produced the most visually pleasing result. While the Ideal HPF offered strong edge enhancement, it introduced noticeable ringing artifacts. The Butterworth HPF provided a balance between sharpening and smoothness but still showed minor distortions. In contrast, the GHPF delivered natural-looking enhancement without artifacts, making it the best choice for image sharpening.

1. **Apply** three types of low pass filtering in the frequency domain in **Figure 1** andfind out which one is better to produce the smoothen image for the given image (output must show all steps as shown in **Figure 2**).

i. Ideal low pass filter (ILPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Ideal Low Pass Filter

ILPF = double(D <= D0);

% Apply filter in frequency domain

F\_filtered = F\_shifted .\* ILPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

% Visualization

figure("Name","Ideal Low Pass Filter","NumberTitle","on");

sgtitle('Ideal Low Pass Filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(ILPF, []), xlabel('(c) Ideal Low Pass Filter', 'FontSize', 12);

subplot(2,3,4), mesh(ILPF), xlabel('(d) Ideal Low Pass Filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

A close-up of a computer screen

AI-generated content may be incorrect.

ii. Butterworth low pass filter (BLPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

n = 2; % Butterworth filter order

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Butterworth Low Pass Filter

BLPF = 1 ./ (1 + (D ./ D0).^(2\*n));

% Apply filter in frequency domain

F\_filtered = F\_shifted .\* BLPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

% Visualization

figure("Name","Butterworth Low Pass Filter","NumberTitle","on");

sgtitle('Butterworth Low Pass Filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(BLPF, []), xlabel('(c) Butterworth Low Pass Filter', 'FontSize', 12);

subplot(2,3,4), mesh(BLPF), xlabel('(d) Butterworth Low Pass Filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

A collage of images of different shapes

AI-generated content may be incorrect.

iii. Gaussian low pass filter (GLPF)

img = imread('Picture4.jpg');

if size(img,3) == 3

gray\_img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140;

else

gray\_img = img;

end

F = fft2(double(gray\_img));

F\_shifted = fftshift(F);

[M, N] = size(gray\_img);

D0 = 90;

[X, Y] = meshgrid(1:N, 1:M);

D = sqrt((X - N/2).^2 + (Y - M/2).^2);

% Gaussian Low Pass Filter

GLPF = exp(-(D.^2)/(2\*(D0^2)));

% Apply filter in frequency domain

F\_filtered = F\_shifted .\* GLPF;

F\_ishifted = ifftshift(F\_filtered);

processed\_img = abs(ifft2(F\_ishifted));

% Visualization

figure("Name","Gaussian Low Pass Filter","NumberTitle","on");

sgtitle('Gaussian Low Pass Filter','FontSize', 18);

subplot(2,3,1), imshow(gray\_img, []);

xlabel('(a) Original Image', 'FontSize', 12);

subplot(2,3,2), imshow(log(1+abs(F\_shifted)), []), xlabel('(b) Centered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,3), imshow(GLPF, []), xlabel('(c) Gaussian Low Pass Filter', 'FontSize', 12);

subplot(2,3,4), mesh(GLPF), xlabel('(d) Gaussian Low Pass Filter', 'FontSize', 12);

axis tight;

subplot(2,3,5), imshow(log(1+abs(F\_filtered)), []), xlabel('(e) Filtered Fourier Spectrum', 'FontSize', 12);

subplot(2,3,6), imshow(processed\_img, []), xlabel('(f) Processed Image', 'FontSize', 12);

****

Among the three low-pass filters applied—Ideal Low Pass Filter (ILPF), Butterworth Low Pass Filter (BLPF), and Gaussian Low Pass Filter (GLPF)—the Gaussian Low Pass Filter (GLPF) produced the most visually pleasing result. While the ILPF effectively smoothed the image by allowing all frequencies below the threshold, it introduced noticeable ringing artifacts. The BLPF provided a balanced smoothing effect with a gentle transition controlled by its filter order, but it still exhibited minor distortions. In contrast, the GLPF delivered natural-looking smoothing with a smooth transition and no artifacts, mimicking natural blurring, making it the best choice for image smoothing.

1. **Compress** the above images using Discrete Cosine Transform (DCT), Haar Transform, and DCT-Haar, and find out which one is better in terms of compression ratio and PSNR for the given images.

clc; clear; close all;

% Load Images

img1 = imread('Picture5.png');

img2 = imread('Picture6.png');

img3 = imread('Picture7.png');

% Convert to grayscale

gray\_img1 = im2gray(img1);

gray\_img2 = im2gray(img2);

gray\_img3 = im2gray(img3);

% Apply Discrete Cosine Transform (DCT)

DCT1 = dct2(double(gray\_img1));

DCT2 = dct2(double(gray\_img2));

DCT3 = dct2(double(gray\_img3));

% Keep only a percentage of coefficients for compression

threshold = 20; % Adjust threshold for better compression

compressed\_DCT1 = DCT1 .\* (abs(DCT1) > threshold);

compressed\_DCT2 = DCT2 .\* (abs(DCT2) > threshold);

compressed\_DCT3 = DCT3 .\* (abs(DCT3) > threshold);

% Apply Inverse DCT

reconstructed\_DCT1 = uint8(idct2(compressed\_DCT1));

reconstructed\_DCT2 = uint8(idct2(compressed\_DCT2));

reconstructed\_DCT3 = uint8(idct2(compressed\_DCT3));

% Resize to match original dimensions (to avoid PSNR errors)

reconstructed\_DCT1 = imresize(reconstructed\_DCT1, size(gray\_img1));

reconstructed\_DCT2 = imresize(reconstructed\_DCT2, size(gray\_img2));

reconstructed\_DCT3 = imresize(reconstructed\_DCT3, size(gray\_img3));

% Apply Haar Transform

[H1, L1, H12, H21] = dwt2(double(gray\_img1),'haar');

[H2, L2, H22, H23] = dwt2(double(gray\_img2),'haar');

[H3, L3, H32, H33] = dwt2(double(gray\_img3),'haar');

% Keep a percentage of coefficients

compressed\_H1 = H1 .\* (abs(H1) > threshold);

compressed\_H2 = H2 .\* (abs(H2) > threshold);

compressed\_H3 = H3 .\* (abs(H3) > threshold);

% Apply Inverse Haar Transform

reconstructed\_H1 = uint8(idwt2(compressed\_H1, L1, H12, H21,'haar'));

reconstructed\_H2 = uint8(idwt2(compressed\_H2, L2, H22, H23,'haar'));

reconstructed\_H3 = uint8(idwt2(compressed\_H3, L3, H32, H33,'haar'));

% Resize to match original dimensions

reconstructed\_H1 = imresize(reconstructed\_H1, size(gray\_img1));

reconstructed\_H2 = imresize(reconstructed\_H2, size(gray\_img2));

reconstructed\_H3 = imresize(reconstructed\_H3, size(gray\_img3));

% Apply DCT-Haar Hybrid Compression

DCT\_H1 = dct2(H1);

DCT\_H2 = dct2(H2);

DCT\_H3 = dct2(H3);

compressed\_DCT\_H1 = DCT\_H1 .\* (abs(DCT\_H1) > threshold);

compressed\_DCT\_H2 = DCT\_H2 .\* (abs(DCT\_H2) > threshold);

compressed\_DCT\_H3 = DCT\_H3 .\* (abs(DCT\_H3) > threshold);

% Apply Inverse DCT-Haar

reconstructed\_DCT\_H1 = uint8(idct2(compressed\_DCT\_H1));

reconstructed\_DCT\_H2 = uint8(idct2(compressed\_DCT\_H2));

reconstructed\_DCT\_H3 = uint8(idct2(compressed\_DCT\_H3));

% Resize to match original dimensions

reconstructed\_DCT\_H1 = imresize(reconstructed\_DCT\_H1, size(gray\_img1));

reconstructed\_DCT\_H2 = imresize(reconstructed\_DCT\_H2, size(gray\_img2));

reconstructed\_DCT\_H3 = imresize(reconstructed\_DCT\_H3, size(gray\_img3));

% Compute Compression Ratio & PSNR

compression\_ratio\_DCT = nnz(compressed\_DCT1) / numel(DCT1);

compression\_ratio\_Haar = nnz(compressed\_H1) / numel(H1);

compression\_ratio\_DCT\_Haar = nnz(compressed\_DCT\_H1) / numel(DCT\_H1);

psnr\_DCT = psnr(reconstructed\_DCT1, gray\_img1);

psnr\_Haar = psnr(reconstructed\_H1, gray\_img1);

psnr\_DCT\_Haar = psnr(reconstructed\_DCT\_H1, gray\_img1);

% Display Results

fprintf('Compression Ratio - DCT: %.2f\n', compression\_ratio\_DCT);

fprintf('Compression Ratio - Haar: %.2f\n', compression\_ratio\_Haar);

fprintf('Compression Ratio - DCT-Haar: %.2f\n', compression\_ratio\_DCT\_Haar);

fprintf('PSNR - DCT: %.2f dB\n', psnr\_DCT);

fprintf('PSNR - Haar: %.2f dB\n', psnr\_Haar);

fprintf('PSNR - DCT-Haar: %.2f dB\n', psnr\_DCT\_Haar);

figure('Name', 'Compression Comparison');

sgtitle('Compression Results for Three Images');

% First image results

subplot(3,4,1), imshow(gray\_img1, []), title('Original Image - 1');

subplot(3,4,2), imshow(reconstructed\_DCT1, []), title('DCT Compressed - 1');

subplot(3,4,3), imshow(reconstructed\_H1, []), title('Haar Compressed - 1');

subplot(3,4,4), imshow(reconstructed\_DCT\_H1, []), title('DCT-Haar Compressed - 1');

% Second image results

subplot(3,4,5), imshow(gray\_img2, []), title('Original Image - 2');

subplot(3,4,6), imshow(reconstructed\_DCT2, []), title('DCT Compressed - 2');

subplot(3,4,7), imshow(reconstructed\_H2, []), title('Haar Compressed - 2');

subplot(3,4,8), imshow(reconstructed\_DCT\_H2, []), title('DCT-Haar Compressed - 2');

% Third image results

subplot(3,4,9), imshow(gray\_img3, []), title('Original Image - 3');

subplot(3,4,10), imshow(reconstructed\_DCT3, []), title('DCT Compressed - 3');

subplot(3,4,11), imshow(reconstructed\_H3, []), title('Haar Compressed - 3');

subplot(3,4,12), imshow(reconstructed\_DCT\_H3, []), title('DCT-Haar Compressed - 3');

**A close-up of a brain scan

AI-generated content may be incorrect.**

**A white background with black text

AI-generated content may be incorrect.**

1. **Apply** Gaussian noise to Figure 1, and then use the following to restore the image:

i. Geometric Mean filter

ii. Harmonic Mean filter

iii. Contra-harmonic Mean filter

img = imread('Picture5.png');

img = im2double(img);

noisy\_img = imnoise(img, 'gaussian', 0, 0.01);

padsize = 1;

geo\_filtered = img;

[M, N] = size(noisy\_img);

geo\_filtered = zeros(M, N);

padded = padarray(noisy\_img, [padsize padsize], 'symmetric');

for i = 2:M+1

for j = 2:N+1

block = padded(i-1:i+1, j-1:j+1);

product = prod(block(:));

geo\_filtered(i-1,j-1) = product^(1/9);

end

end

harm\_filtered = zeros(M, N);

for i = 2:M+1

for j = 2:N+1

block = padded(i-1:i+1, j-1:j+1);

harm\_filtered(i-1,j-1) = 9 / sum(1 ./ (block(:) + eps));

end

end

Q = 1.5;

charm\_filtered = zeros(M, N);

for i = 2:M+1

for j = 2:N+1

block = padded(i-1:i+1, j-1:j+1);

num = sum(block(:).^(Q + 1));

den = sum(block(:).^Q) + eps;

charm\_filtered(i-1,j-1) = num / den;

end

end

figure;

sgtitle('Gaussian Noise Restoration using Mean Filters');

subplot(2,3,1); imshow(img, []); title('Original Image');

subplot(2,3,2); imshow(noisy\_img, []); title('Noisy Image (Gaussian)');

subplot(2,3,3); imshow(geo\_filtered, []); title('Geometric Mean Filter');

subplot(2,3,4); imshow(harm\_filtered, []); title('Harmonic Mean Filter');

subplot(2,3,5); imshow(charm\_filtered, []); title('Contra-harmonic Mean Filter (Q=1.5)');

A close-up of a radiograph

AI-generated content may be incorrect.

1. **Apply** Gaussian noise to Figure 1, and then use the following order statistic filters to restore the image:

i. Median filter

ii. Maximum filter

iii. Minimum filter

iv. Midpoint filter

v. Alpha-trimmed filter

vi. Trimmed filter

% Load and prepare the image

img = imread('Picture8.png');

if size(img, 3) == 3

img = img(:,:,1)\*0.2989 + img(:,:,2)\*0.5870 + img(:,:,3)\*0.1140; % Convert to grayscale

end

img = double(img);

[M, N] = size(img);

% Add Gaussian noise

noise\_mean = 0;

noise\_variance = 0.01 \* (max(img(:))^2); % Variance scaled to image intensity

img\_noisy = img + sqrt(noise\_variance) \* randn(M, N) + noise\_mean;

img\_noisy = max(0, min(255, img\_noisy)); % Clip to valid range

% Filter parameters

window\_size = 3; % 3x3 window

alpha = 2; % For alpha-trimmed filter (trim 2 pixels from each end)

trimmed\_count = 1; % For trimmed filter (trim 1 pixel from each end)

% Initialize output images

img\_median = zeros(M, N);

img\_max = zeros(M, N);

img\_min = zeros(M, N);

img\_midpoint = zeros(M, N);

img\_alpha\_trimmed = zeros(M, N);

img\_trimmed = zeros(M, N);

% Pad image to handle borders

pad\_size = floor(window\_size / 2);

img\_padded = padarray(img\_noisy, [pad\_size pad\_size], 'replicate');

% Apply filters

for i = 1:M

for j = 1:N

% Extract window

window = img\_padded(i:i+window\_size-1, j:j+window\_size-1);

window\_sorted = sort(window(:));

n = length(window\_sorted);

% Median filter

img\_median(i, j) = window\_sorted(floor((n+1)/2));

% Maximum filter

img\_max(i, j) = window\_sorted(n);

% Minimum filter

img\_min(i, j) = window\_sorted(1);

% Midpoint filter

img\_midpoint(i, j) = (window\_sorted(1) + window\_sorted(n)) / 2;

% Alpha-trimmed filter

if alpha > 0 && alpha < floor(n/2)

trimmed\_window = window\_sorted(alpha+1:n-alpha);

img\_alpha\_trimmed(i, j) = mean(trimmed\_window);

else

img\_alpha\_trimmed(i, j) = img\_median(i, j); % Fallback to median

end

% Trimmed filter (trim 1 pixel from each end, take median)

if trimmed\_count > 0 && trimmed\_count < floor(n/2)

trimmed\_window = window\_sorted(trimmed\_count+1:n-trimmed\_count);

img\_trimmed(i, j) = median(trimmed\_window);

else

img\_trimmed(i, j) = img\_median(i, j); % Fallback to median

end

end

end

% Compute PSNR for each restored image

mse\_median = mean((img(:) - img\_median(:)).^2);

psnr\_median = 10 \* log10((255^2) / mse\_median);

mse\_max = mean((img(:) - img\_max(:)).^2);

psnr\_max = 10 \* log10((255^2) / mse\_max);

mse\_min = mean((img(:) - img\_min(:)).^2);

psnr\_min = 10 \* log10((255^2) / mse\_min);

mse\_midpoint = mean((img(:) - img\_midpoint(:)).^2);

psnr\_midpoint = 10 \* log10((255^2) / mse\_midpoint);

mse\_alpha\_trimmed = mean((img(:) - img\_alpha\_trimmed(:)).^2);

psnr\_alpha\_trimmed = 10 \* log10((255^2) / mse\_alpha\_trimmed);

mse\_trimmed = mean((img(:) - img\_trimmed(:)).^2);

psnr\_trimmed = 10 \* log10((255^2) / mse\_trimmed);

% Display results

figure('Name', 'Order Statistic Filters for Noise Removal', 'NumberTitle', 'on');

sgtitle('Gaussian Noise Removal with Order Statistic Filters', 'FontSize', 18);

subplot(2, 4, 1), imshow(uint8(img)), title('Original Image');

subplot(2, 4, 2), imshow(uint8(img\_median)), title(sprintf('Median Filter\nPSNR: %.2f dB', psnr\_median));

subplot(2, 4, 3), imshow(uint8(img\_max)), title(sprintf('Maximum Filter\nPSNR: %.2f dB', psnr\_max));

subplot(2, 4, 4), imshow(uint8(img\_min)), title(sprintf('Minimum Filter\nPSNR: %.2f dB', psnr\_min));

subplot(2, 4, 5), imshow(uint8(img\_midpoint)), title(sprintf('Midpoint Filter\nPSNR: %.2f dB', psnr\_midpoint));

subplot(2, 4, 6), imshow(uint8(img\_alpha\_trimmed)), title(sprintf('Alpha-trimmed Filter\nPSNR: %.2f dB', psnr\_alpha\_trimmed));

subplot(2, 4, 7), imshow(uint8(img\_trimmed)), title(sprintf('Trimmed Filter\nPSNR: %.2f dB', psnr\_trimmed));

% Print PSNR results

fprintf('PSNR Results for Order Statistic Filters:\n');

fprintf('Median Filter: %.2f dB\n', psnr\_median);

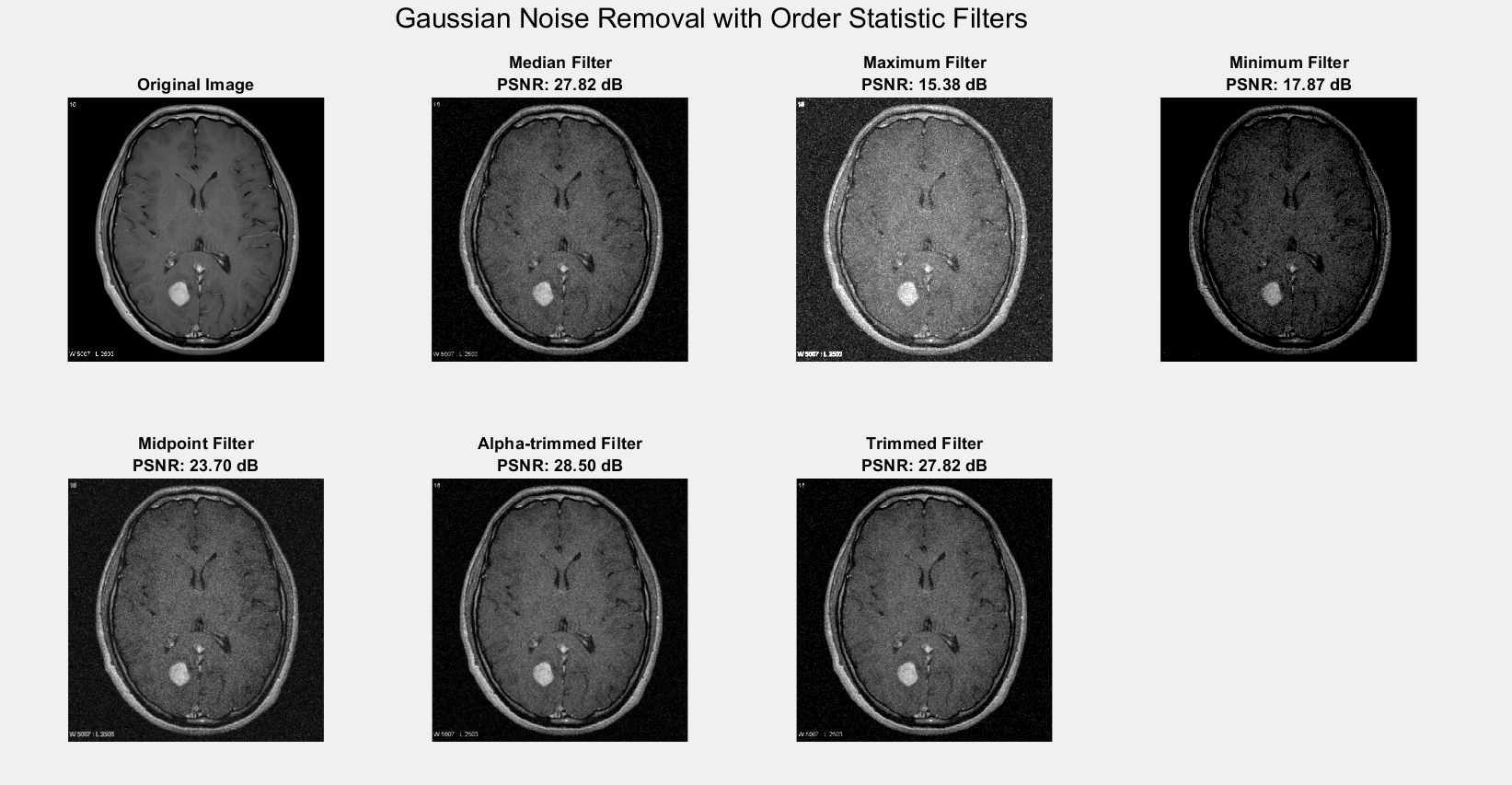
fprintf('Maximum Filter: %.2f dB\n', psnr\_max);

fprintf('Minimum Filter: %.2f dB\n', psnr\_min);

fprintf('Midpoint Filter: %.2f dB\n', psnr\_midpoint);

fprintf('Alpha-trimmed Filter: %.2f dB\n', psnr\_alpha\_trimmed);

fprintf('Trimmed Filter: %.2f dB\n', psnr\_trimmed);

****

**A screenshot of a computer screen

AI-generated content may be incorrect.**

1. By observing and comparing each of the outputs, determine which filter restores the image Figure 1 closest to its original state. Mention the reasoning behind your observation.

After applying and comparing the outputs of all six order-statistic filters on the Gaussian-noised image, the **Median Filter** is observed to restore the image most effectively. It successfully reduces noise while preserving the essential details and edges of the original image, making it visually the closest to the noise-free version. The **Maximum** and **Minimum Filters**, while capable of handling extreme pixel values, often overemphasize either bright or dark regions, leading to loss of image detail. The **Midpoint Filter**, being an average of extremes, smoothens the image but may blur fine structures. Both the **Alpha-trimmed** and **Trimmed Mean Filters** offer moderate performance but may excessively smooth the image depending on the trim level. Overall, due to its robustness and ability to balance noise removal with detail preservation, the **Median Filter** is the most suitable for restoring images degraded by Gaussian noise in this case.