Dip Lab #15

SUBMITED TO:

Ma'am Nazia

SUBMITTED BY:

M.JUNAID (21-SE-100)

DEPARTMENT:

SOFTWARE ENGINEERING

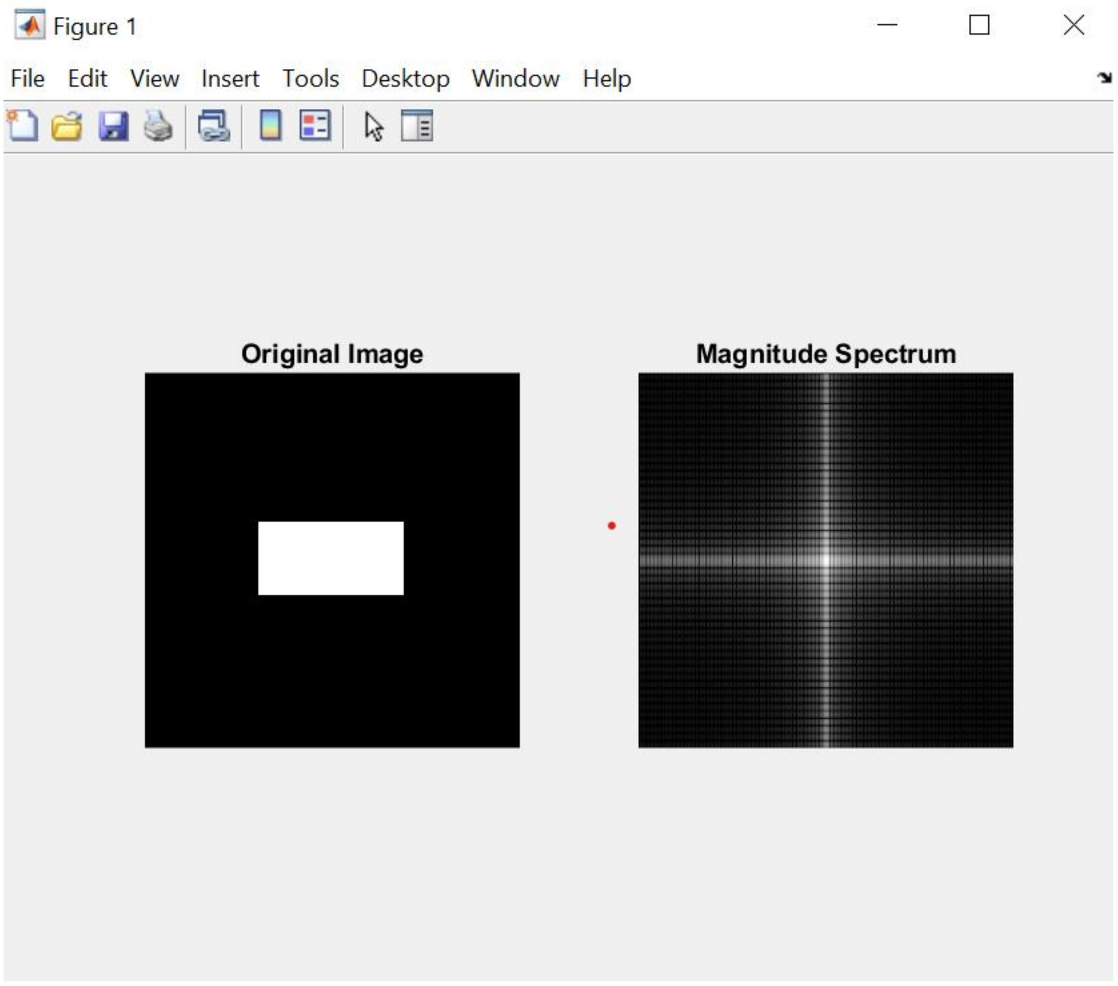
UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

Task#01

```
% Define image dimensions
M = 256; % Height
N = 256; % Width

% Create black background image
image = zeros(M, N);
```

```
% Define rectangle parameters
rect_width = 100;
rect_height = 50;
rect_x = round((N - rect_width) / 2); % Center the
rectangle horizontally
rect_y = round((M - rect_height) / 2); % Center the
rectangle vertically

% Add white rectangle to the image
image(rect_y:rect_y+rect_height-1,
rect_x:rect_x+rect_width-1) = 1;

% Calculate the DFT with zero-padding
image_fft = fftshift(fft2(image, 2*M, 2*N));

% Calculate magnitude spectrum
magnitude_spectrum = abs(image_fft);

% Brighten the display
brightened_spectrum = log(1 + magnitude_spectrum);

% Display the original image and its magnitude
spectrum
figure;
subplot(1, 2, 1);
imshow(image);
title('Original Image');

subplot(1, 2, 2);
imshow(brightened_spectrum, []);
title('Magnitude Spectrum');
```

Output:

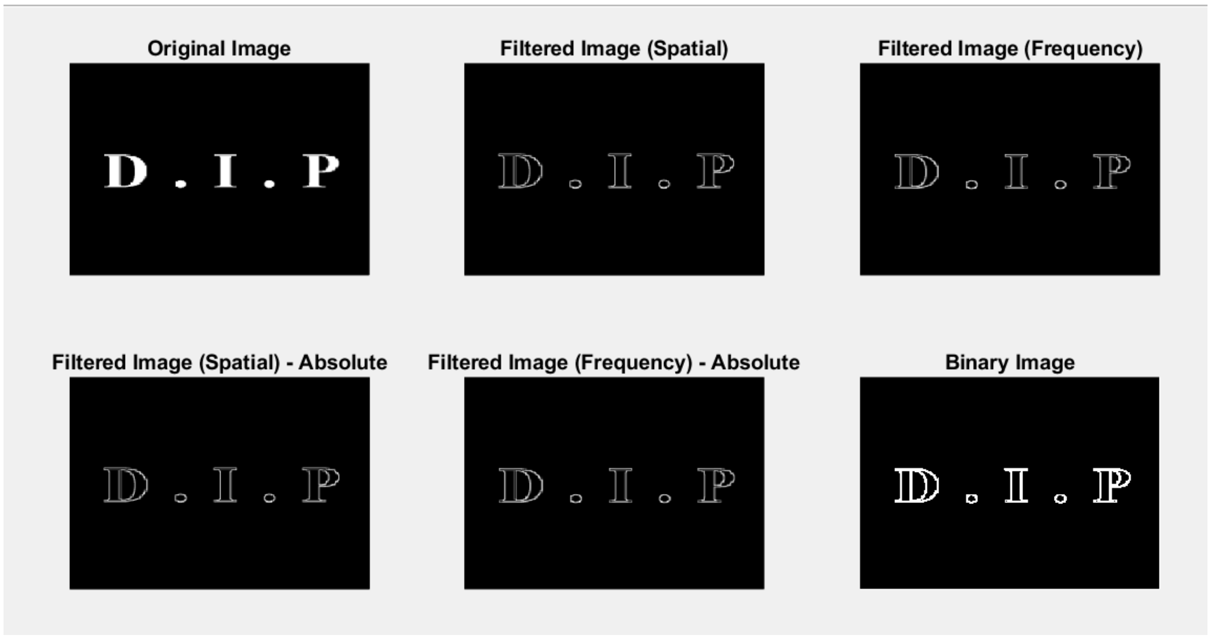


Task#02

```
% Read the image
originalImage = imread('E:\dip_img.png');

% Convert the image to grayscale
grayImage = rgb2gray(originalImage);

% Create Sobel filter kernels
sobelHorizontal = [-1 0 1; -2 0 2; -1 0 1];
sobelVertical = [-1 -2 -1; 0 0 0; 1 2 1];

% Apply Sobel filter in spatial domain
filteredSpatial = abs(conv2(double(grayImage),
sobelHorizontal, 'same')) + abs(conv2(double(grayImage),
sobelVertical, 'same'));

% Apply Fourier Transform to the grayscale image
fourierImage = fft2(double(grayImage));
```

```
% Create Sobel filter kernels in frequency domain
sobelHorizontalFreq = fft2(sobelHorizontal,
size(grayImage, 1), size(grayImage, 2));
sobelVerticalFreq = fft2(sobelVertical, size(grayImage,
1), size(grayImage, 2));

% Apply Sobel filter in frequency domain
filteredFreq = abs(ifft2(fourierImage .*
sobelHorizontalFreq)) + abs(ifft2(fourierImage .*
sobelVerticalFreq));

% Threshold the filtered image
thresholdValue = 100;
binaryImage = filteredFreq > thresholdValue;

% Display the images
figure;
subplot(2, 3, 1), imshow(originalImage), title('Original
Image');
subplot(2, 3, 2), imshow(filteredSpatial, []),
title('Filtered Image (Spatial)');
subplot(2, 3, 3), imshow(filteredFreq, []),
title('Filtered Image (Frequency)');
subplot(2, 3, 4), imshow(abs(filteredSpatial), []),
title('Filtered Image (Spatial) - Absolute');
subplot(2, 3, 5), imshow(abs(filteredFreq), []),
title('Filtered Image (Frequency) - Absolute');
subplot(2, 3, 6), imshow(binaryImage), title('Binary
Image');
```

Output:

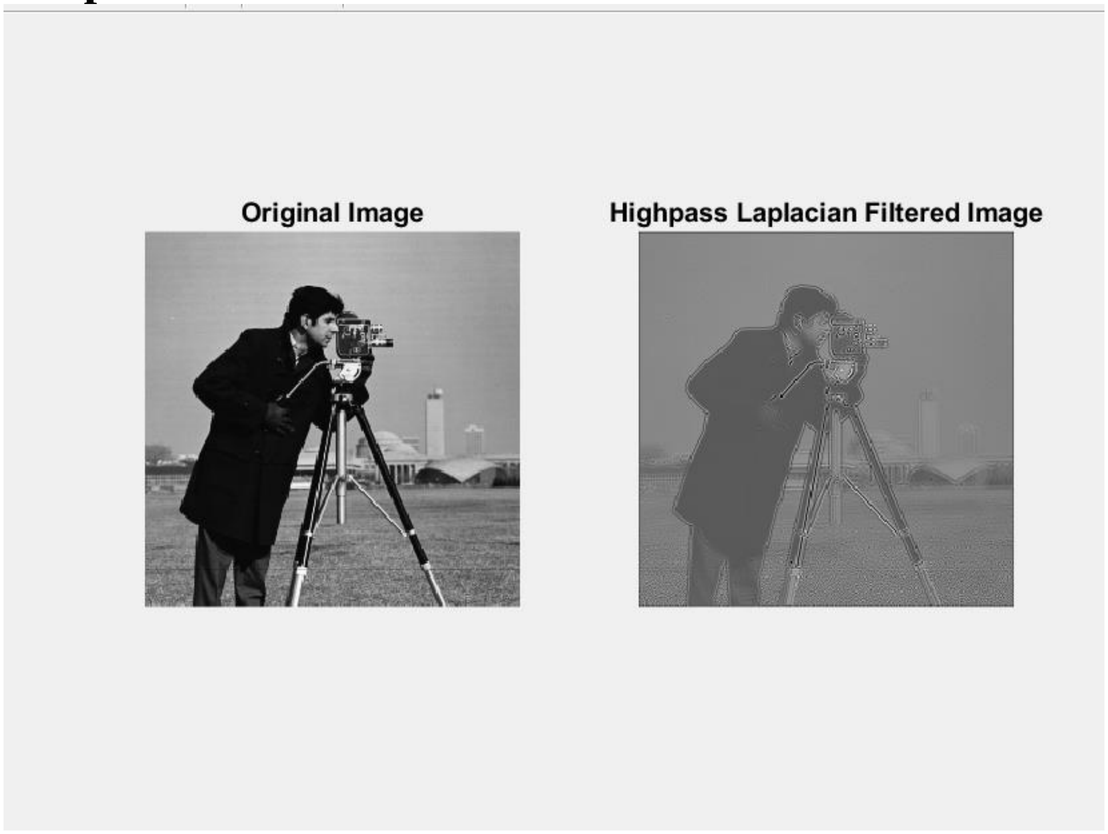

Original Image
D . I . P
Filtered Image (Spatial)
Filtered Image (Frequency)
Filtered Image (Spatial) - Absolute
Filtered Image (Frequency) - Absolute
Binary Image

Task#03

```
% Read the image
img = imread('cameraman.tif');

% Convert the image to grayscale if it's not already
if size(img, 3) == 3
    img = rgb2gray(img);
end

% Convert the image to double precision for processing
img = double(img);

% Define Laplacian filter
laplacian_filter = [0 -1 0; -1 4 -1; 0 -1 0];

% Apply Laplacian filter
filtered_img = conv2(img, laplacian_filter, 'same');

% Enhance edges using highpass filter
enhanced_img = img - filtered_img;

% Normalize the image
enhanced_img = uint8(255 * mat2gray(enhanced_img));

% Display original and filtered images
subplot(1,2,1), imshow(uint8(img)), title('Original
Image');
```

```
subplot(1,2,2), imshow(enhanced_img), title('Highpass
Laplacian Filtered Image');

% % Save the filtered image
% imwrite(enhanced_img,
'highpass_laplacian_filtered_image.jpg');
```

Output:

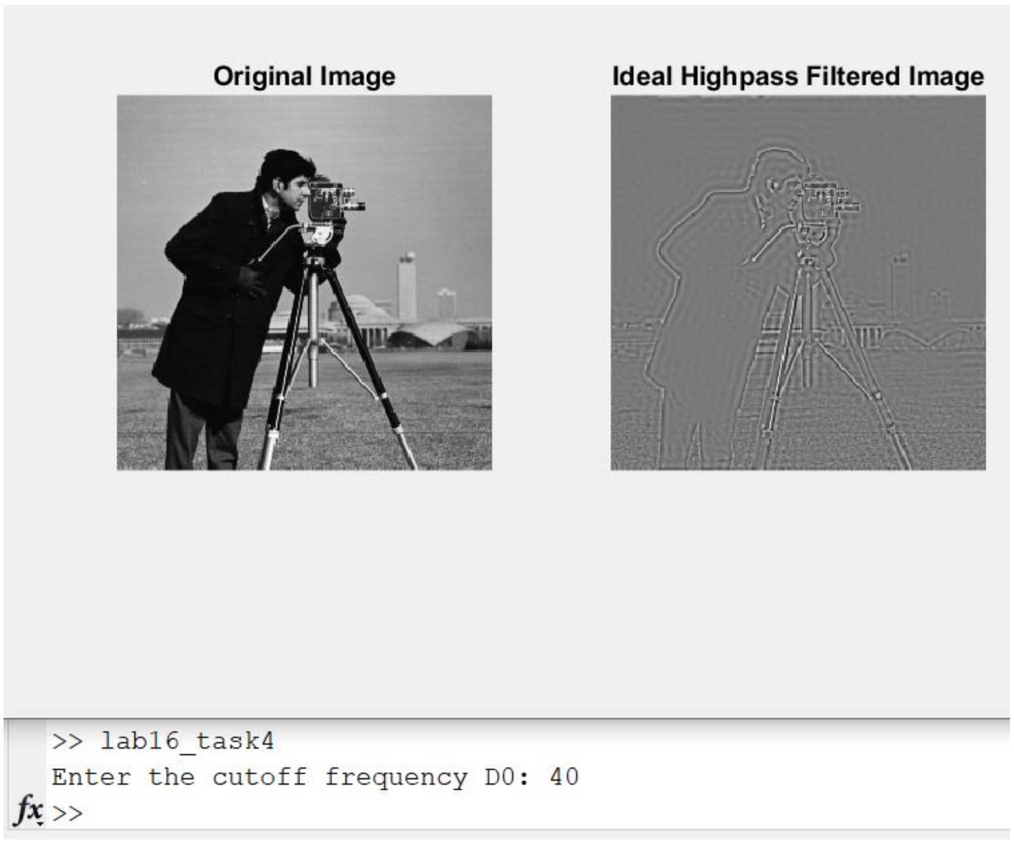


Task#04

```
% Read the image
img = imread('cameraman.tif');

% Convert the image to grayscale if it's not
already
if size(img, 3) == 3
    img = rgb2gray(img);
end
```

```
% Convert the image to double precision for
processing
img = double(img);

% Get the size of the image
[M, N] = size(img);

% Center of the image
cx = round(M / 2);
cy = round(N / 2);

% Define cutoff frequency
D0 = input('Enter the cutoff frequency D0: ');

% Create a meshgrid for frequency domain
[u, v] = meshgrid(1:N, 1:M);

% Compute the distance from the center of the
frequency domain
D = sqrt((u - cx).^2 + (v - cy).^2);

% Define the ideal highpass filter
H = double(D > D0);

% Apply the filter in frequency domain
filtered_img = ifftshift(fft2(img)) .* H;

% Inverse Fourier transform to get back to spatial
domain
filtered_img =
real(ifft2(ifftshift(filtered_img)));

% Normalize the image
filtered_img = uint8(255 *
mat2gray(filtered_img));

% Display original and filtered images
subplot(1,2,1), imshow(uint8(img)),
title('Original Image');
subplot(1,2,2), imshow(filtered_img), title('Ideal
Highpass Filtered Image');

% % Save the filtered image
% imwrite(filtered_img,
'ideal_highpass_filtered_image.jpg');
```

Output:



```
>> lab16_task4
Enter the cutoff frequency D0: 40
fx >>
```

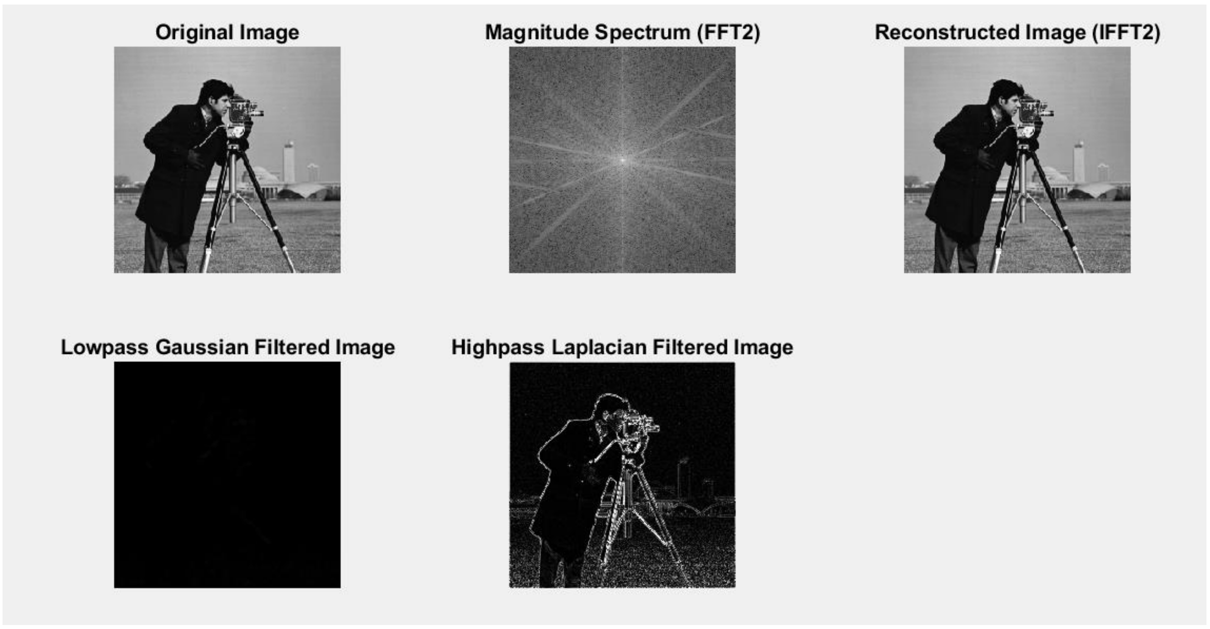
Task#05

```
% Read the image
img = imread('cameraman.tif');

% Convert the image to double precision for
processing
img = double(img);

% Display the original image
figure;
subplot(2, 3, 1);
imshow(uint8(img));
title('Original Image');
```

```
% Apply FFT2
fft_img = fft2(img);

% Display the magnitude spectrum of FFT2
subplot(2, 3, 2);
imshow(log(1 + abs(fftshift(fft_img))), []);
title('Magnitude Spectrum (FFT2)');

% Apply IFFT2
ifft_img = ifft2(fft_img);

% Display the reconstructed image using IFFT2
subplot(2, 3, 3);
imshow(uint8(ifft_img));
title('Reconstructed Image (IFFT2)');

% Define lowpass Gaussian filter
[M, N] = size(img);
sigma = 20;
[X, Y] = meshgrid(1:N, 1:M);
centerX = ceil(N/2);
centerY = ceil(M/2);
gaussian_filter = exp(-((X - centerX).^2 + (Y -
centerY).^2) / (2*sigma^2));

% Apply the lowpass Gaussian filter in frequency
domain
filtered_fft_img = fft_img .* gaussian_filter;
% Reconstruct the image using IFFT2 after applying
the filter
filtered_img = ifft2(filtered_fft_img);

% Display the filtered image
subplot(2, 3, 4);
imshow(uint8(abs(filtered_img)));
title('Lowpass Gaussian Filtered Image');
% Define highpass Laplacian filter
laplacian_filter = fspecial('laplacian', 0);

% Apply the highpass Laplacian filter
highpass_filtered_img = imfilter(img,
laplacian_filter);
```

```
% Display the highpass filtered image
subplot(2, 3, 5);
imshow(uint8(abs(highpass_filtered_img)));
title('Highpass Laplacian Filtered Image');
```

Output:

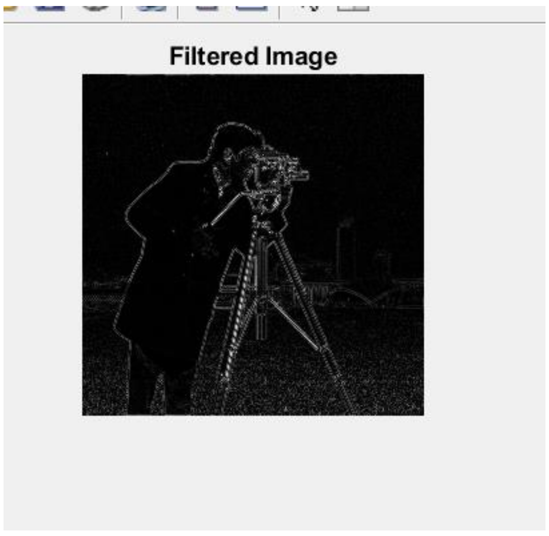

Original Image
Magnitude Spectrum (FFT2)
Reconstructed Image (IFFT2)
Lowpass Gaussian Filtered Image
Highpass Laplacian Filtered Image

Task#06

```
% Read the image
image = imread('cameraman.tif'); % Replace
'your_image_path.jpg' with the actual path to your
image file

% Convert the image to double precision
image = im2double(image);

% Define the size of the filter (e.g., same as the
image size)
M = size(image, 1);
N = size(image, 2);

% Create a meshgrid for frequency domain
```

```
coordinates
[u, v] = meshgrid(1:N, 1:M);

% Compute the distance from the origin
D = sqrt((u - M/2).^2 + (v - N/2).^2);

% Set the value of D0 (given as 120)
D0 = 120;

% Compute the filter response
H = 1 - exp(-(D.^2) / (2 * D0^2));

% Apply the filter to the image in the frequency
domain
filtered_image = fftshift(fft2(image)) .* H;

% Inverse Fourier transform to get the filtered
image in spatial domain
filtered_image = ifft2(ifftshift(filtered_image));

% Display the filtered image
imshow(abs(filtered_image), []);
title('Filtered Image');
```

Output:


Filtered Image