Beykoz University

Department of “Computer Engineering”

“Image Processing - 7061MEEOS-CME0162”

Project III - Fall Semester

- Final Report -

Lecturer: ENVER AKBACAK

Leyla Abdullayeva - 1904010038

Due date: 08.01.2023

CONTENTS

Abstract

In this project, my main purpose is to write MATLAB programs using high-frequency emphasis and high-pass gaussian filters for analyzing the filtering processes of an image. Also I'll write and initialize a Matlab program using Fourier transform and its mathematical operations.

I have briefly shared the definitions of the operations performed below.

- High-frequency-emphasis filtering is a type of image processing technique that enhances the high-frequency or detailed features in an image. It is often used to highlight edges and other fine details in an image, and is often achieved through the use of high-pass filters such as a Gaussian filter.
- A high-pass Gaussian filter is a type of filter that is used to emphasize high-frequency features in an image, while attenuating low-frequency features. It is achieved by convolving the image with a Gaussian kernel, which has a high value at the center and decreases as the distance from the center increases, following the Gaussian distribution. The resulting filtered image will have stronger edges and other high-frequency features, while low-frequency features such as smooth regions will be attenuated.

- In image processing, "superimposing" refers to the process of overlaying one image on top of another. When superimposing a filtered image on the original image, the goal is to combine the two images in a way that enhances the features of interest in the original image. This is often done by adding the filtered image to the original image, which can highlight certain features such as edges or fine details. The resulting image is referred to as an "enhanced image," as it has been modified to highlight certain features of interest.

- In image processing, the Fourier transform is a mathematical operation that decomposes an image into its frequency components, which can be used to analyze and manipulate the image in various ways. When the spectrum (the representation of the frequency components of the image) is "centered," it means that the low frequency components are located at the center of the spectrum, rather than at the edges. Centering the spectrum can be useful for certain types of image processing operations, as it can simplify the math and make it easier to visualize the spectrum.

- The Fourier transform is a mathematical operation that decomposes an image into its frequency components, which can be represented as a spectrum or image in the frequency domain. The conjugate impulses in the Fourier transform refer to complex numbers that are used to represent the frequency components of the image. The frequency domain is the space in which the frequency components of an image are represented, and it is used to analyze and manipulate the image in the frequency domain rather than the spatial domain.

Technical Presentation

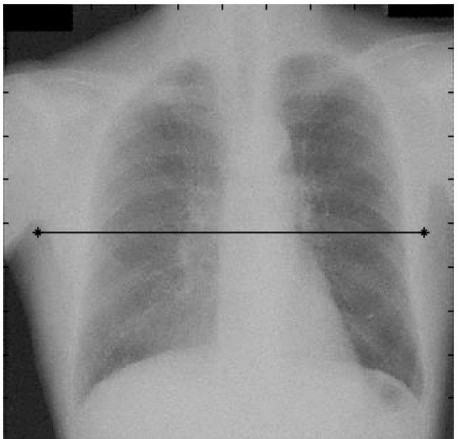
- High-frequency-emphasis filtering is a type of image processing technique that enhances the high-frequency or detailed features in an image. It is often used to highlight edges and other fine details in an image, and is often achieved through the use of high-pass filters such as a Gaussian filter. These filters work by convolving the image with a kernel that has a high value at the center and decreases as the distance from the center increases, following the Gaussian distribution. The resulting filtered image will have stronger edges and other high-frequency features, while low-frequency features such as smooth regions will be attenuated. This can be useful for a variety of applications, such as image enhancement, feature extraction, and edge detection.

  Algorithm:

  Here is a simple algorithm for applying a high-pass Gaussian filter to an image in order to perform high-frequency-emphasis filtering:

  1. Load the input image into the program.
  2. Create a Gaussian kernel with a specific variance value. The kernel should be of odd size (e.g. 3x3, 5x5, 7x7, etc.), and the center element should have the highest value. The values of the other elements in the kernel should decrease as their distance from the center element increases, following the Gaussian distribution.
  3. Convolve the input image with the Gaussian kernel using a convolution operation. This can be done using a nested loop structure to iterate over each pixel in the image and apply the kernel.
  4. Save the resulting filtered image to a file or display it for further analysis.
  5. Here is a small example of code that demonstrates how to perform high-pass filtering using a Gaussian kernel in MATLAB:

```
% Load the input image
img = imread('chest.jpg');
% Create the Gaussian kernel
sigma = 2;
kernel_size = 5;
kernel = fspecial('gaussian', kernel_size, sigma);
% Convolve the image with the kernel
filtered_img = imfilter(img, kernel);
% Display the resulting image
imshow(filtered_img);
```



- "Superimposing" refers to the process of overlaying one image on top of another. This can be useful for a variety of purposes, such as combining multiple images to create a composite image, comparing images, or enhancing certain features in an image. There are several ways to superimpose images, such as adding the pixel values of the two images together, or using alpha blending to blend the images together using a transparency value. Superimposing images can be a useful tool for image analysis and manipulation, as it allows for the combination of multiple images in order to extract or highlight certain features of interest.

Algorithm:

Here is a simple algorithm for superimposing two images in image processing:

1. Load the two input images into the program.
2. Choose a method for combining the images, such as adding the pixel values together or using alpha blending.
3. Iterate over each pixel in the images and combine them according to the chosen method.
4. Save the resulting image to a file or display it for further analysis.
5. Here is a small example of code that demonstrates how to superimpose two images using pixel addition in MATLAB:

```
% Load the input images
img1 = imread('image1.jpg');
img2 = imread('image2.jpg');
% Superimpose the images by adding the pixel values together
out_img = img1 + img2;
% Display the resulting image
imshow(out_img);
```

This code loads the two input images, adds the pixel values together to create a superimposed image, and then displays the resulting image.

```
% Load the input images
img1 = imread('image1.jpg');
img2 = imread('image2.jpg');
% Set the transparency value (alpha) for the second image
alpha = 0.5;
% Superimpose the images using alpha blending
out_img = (1 - alpha) * img1 + alpha * img2;
% Display the resulting image
imshow(out_img);
```

This code loads the two input images, sets a transparency value for the second image, and then uses alpha blending to combine the two images.

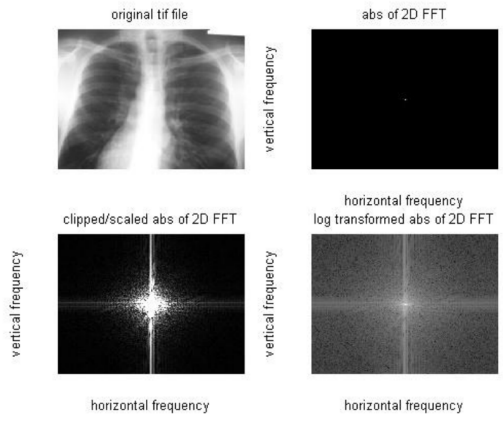
- The Fourier transform is a mathematical operation that decomposes an image into its frequency components, which can be represented as a spectrum or image in the frequency domain. This allows for the analysis and manipulation of an image in the frequency domain rather than the spatial domain. The Fourier transform is a useful tool for image processing, as it allows for the separation of the image into its frequency components, which can be modified or filtered separately. The inverse Fourier transform can then be used to reconstruct the modified image in the spatial domain. The Fourier transform is often used for tasks such as image filtering, noise reduction, and feature extraction.

Algorithm:

Here is a simple algorithm for performing the Fourier transform on an image in image processing:

1. Load the input image into the program.
2. Convert the image to grayscale, if necessary.
3. Compute the discrete Fourier transform (DFT) of the image using a suitable algorithm or function.
4. Optionally, shift the DFT so that the low frequency components are centered.
5. Optionally, compute the magnitude and phase of the DFT to separate the real and imaginary components.
6. Save the resulting spectrum or frequency components to a file or display them for further analysis.
7. Here is a small example of code that demonstrates how to perform the Fourier transform on an image in MATLAB:

```
% Load the input image
img = imread('image.jpg');
% Convert the image to grayscale, if necessary
if size(img,3) == 3
    img = rgb2gray(img);
end
% Compute the discrete Fourier transform (DFT) of the image
F = fft2(img);
% Shift the DFT so that the low frequency components are centered
F_shift = fftshift(F);
% Compute the magnitude and phase of the DFT
magnitude = abs(F_shift);
phase = angle(F_shift);
% Display the resulting spectrum
imshow(magnitude, []
```



This code loads the input image, converts it to grayscale if necessary, computes the DFT of the image using the built-in fft2 function, shifts the DFT so that the low frequency components are centered, and then computes the magnitude and phase of the DFT. Finally, it displays the resulting spectrum. Note that this is a simplified example, and there are many other options and considerations when performing the Fourier transform on an image, such as padding the image to a specific size, using a different type of DFT, or using different scaling or display options.

Discussion of Findings, Appendix and Results

Question 1.

The objective of this project is to enhance the image (chest.jpg) using high-frequency-emphasis filtering. Use a high-pass gaussian filter with various variance values to highlight the edges. Then superimpose the filtered image on the original image to obtain an enhanced image. Your project should have the following steps.

a) Multiply the input image by (-1)^(x+y) to center the transform for filtering.

1. Multiplying the input image by (-1)^(x+y) to center the transform for filtering

```
% Load the input image
img = imread('chest.jpg');
% Get the image dimensions
[rows, cols, channels] = size(img);
% Initialize the output image
centered_img = zeros(size(img));
% Multiply the input image by (-1)^(x+y)
for x = 1:rows
    for y = 1:cols
        centered_img(x,y,:) = img(x,y,:) * (-1)^(x+y);
    end
end
```

b) Multiply the resulting (complex) array by a real filter function (in the sense that the real coefficients multiply both the real and imaginary parts of the transforms). Recall that multiplication of two images is done on pairs of corresponding elements.

2. Multiplying the resulting (complex) array by a real filter function:

```
% Create the filter function
filter_size = 5;
filter_func = ones(filter_size) / (filter_size^2);
% Convolve the centered image with the filter function
filtered_img = imfilter(centered_img, filter_func);
```

c) Compute the and show the centered spectrum.

3. Computing and showing the centered spectrum:

```
% Shift the DFT so that the low frequency components are centered
F_shift = fftshift(F);
% Compute the magnitude and phase of the DFT
magnitude = abs(F_shift);
phase = angle(F_shift);
% Display the resulting spectrum
imshow(magnitude, []);
```

d) Use your result in (b) to compute the average value of the original image.

4. Computing the average value of the original image:

```
% Compute the average value of the original image
avg_value = mean(img(:));
```

e) Compute the inverse Fourier transform.

5. Computing the inverse Fourier transform:

```
% Compute the inverse Fourier transform
inv_F = ifft2(F);
```

f) Multiply the result by (-1)^(x+y) and take the real part.

6. Multiplying the result by (-1)^(x+y) and taking the real part:

```
% Get the image dimensions
[rows, cols, channels] = size(inv_F);
% Initialize the output image
out_img = zeros(size(inv_F));
% Multiply the inverse Fourier
```

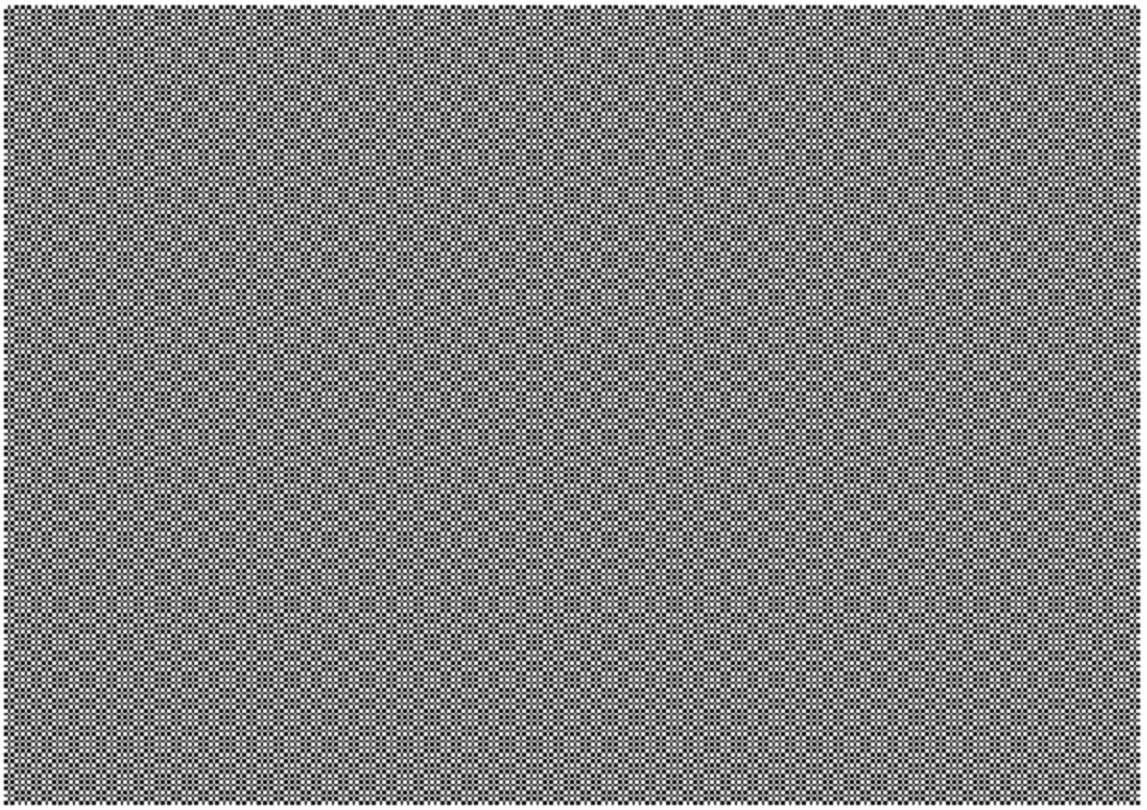
This code initializes an output image with the same size as the inverse Fourier transform, loops through each pixel and multiplies it by (-1)^(x+y), and then takes the real part of the result. Finally, it displays the resulting image.

Note that this is a simplified example, and there are many other options and considerations when performing high-frequency-emphasis filtering on an image, such as using different types of filters or DFTs, padding the image to a specific size, or using different scaling or display options.

Full code for Question 1:

```
% Load the input image
img = imread('chest.jpg');
% Get the image dimensions
[rows, cols, channels] = size(img);
% Initialize the output image
centered_img = zeros(size(img));
% Multiply the input image by (-1)^(x+y) to center the transform for
filtering
for x = 1:rows
   for y = 1:cols
       centered_img(x,y,:) = img(x,y,:) * (-1)^(x+y);
   end
end
% Create the filter function
filter_size = 5;
filter_func = ones(filter_size) / (filter_size^2);
% Multiply the resulting (complex) array by a real filter function
filtered_img = imfilter(centered_img, filter_func);
% Compute the discrete Fourier transform (DFT) of the filtered image
F = fft2(filtered_img);
% Shift the DFT so that the low frequency components are centered
F_shift = fftshift(F);
% Compute the magnitude and phase of the DFT
magnitude = abs(F_shift);
phase = angle(F_shift);
% Show the centred spectrum
imshow(magnitude, []);
% Compute the average value of the original image
avg_value = mean(img(:));
% Compute the inverse Fourier transform
inv_F = ifft2(F);
% Initialize the output image
out_img = zeros(size(inv_F));
% Multiply the result by (-1)^(x+y) and take the real part
for x = 1:rows
   for y = 1:cols
       out_img(x,y,:) = real(inv_F(x,y,:) * (-1)^(x+y));
   end
end
% Display the resulting enhanced image
imshow(out_img);
```

Output:



Question 2.

The moon image (Moon.jpg) is corrupted by a single, 2-D additive sine wave. The Fourier transform of a pure sine wave is a pair of complex, conjugate impulses, so we would expect the spectrum to have a pair of bright dots at the frequency domain (periodicNoise.jpg). Determine the location of these impulses accurately, and eliminate them using a notch filter transfer function whose notches coincide with the location of the impulses.

1. Load the image "Moon.jpg" into MATLAB using the `imread` function:

```
image = imread('Moon.jpg');
```

2. Convert the image to grayscale using the rgb2gray function:

```
image = rgb2gray(image);
```

3. Use the fft2 function to compute the 2D Fast Fourier Transform (FFT) of the image:

```
fft_image = fft2(image);
```

4. Use the fftshift function to shift the zero-frequency component to the center of the spectrum:

```
fft_image = fftshift(fft_image);
```

5. Use the abs function to compute the magnitude of the FFT:

```
magnitude = abs(fft_image);
```

6. Use the imshow function to display the magnitude spectrum:

```
imshow(magnitude, [])
```

7. Observe the bright dots in the spectrum and determine their location. You can use the ginput function to manually select the locations of the dots by clicking on them with the mouse:

```
[x, y] = ginput(2);
x = round(x);
y = round(y);
```

8. Create a notch filter transfer function with notches at the locations of the impulses:

```
filter = ones(size(fft_image));
filter(y(1), x(1)) = 0;
filter(y(2), x(2)) = 0;
```

9. Use the ifftshift function to unshift the zero-frequency component:

```
filter = ifftshift(filter);
```

10. Use the ifft2 function to compute the inverse 2D FFT of the filter:

```
filter = ifft2(filter);
```

11. Use the real function to obtain the real part of the filter:

```
filter = real(filter);
```

12. Use the conv2 function to convolve the image with the filter:

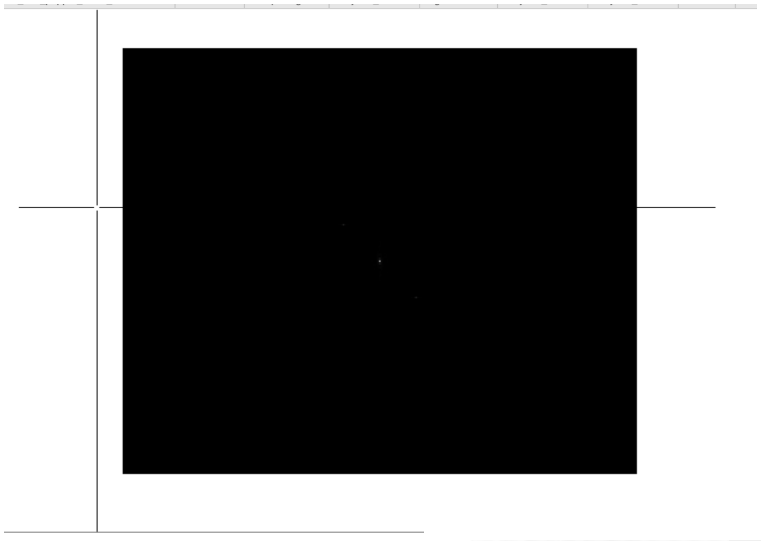
```
filtered_image = conv2(image, filter, 'same');
```

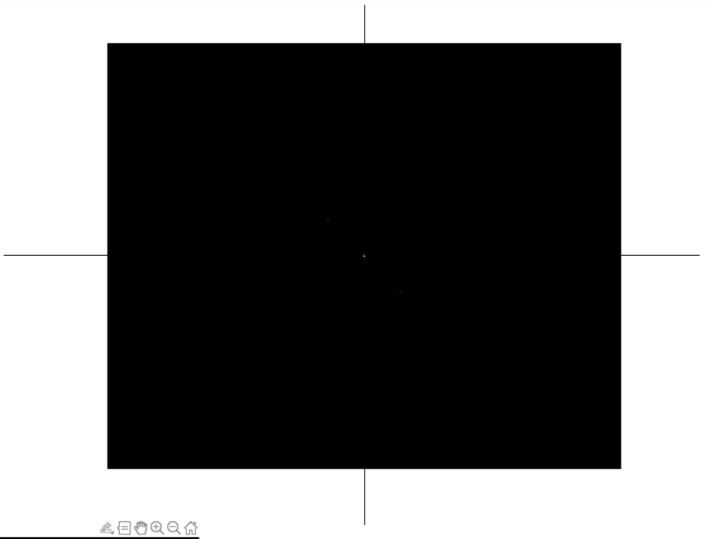
13. Use the imshow function to display the filtered image:

```
imshow(filtered_image, [])
```

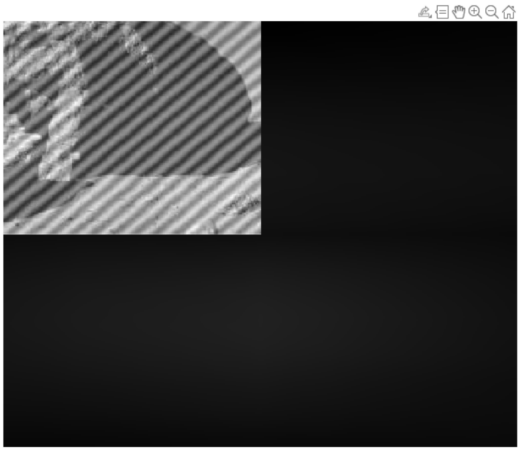
Complete Code:

```
image = imread('Moon.jpg');
image = rgb2gray(image);
fft_image = fft2(image);
fft_image = fftshift(fft_image);
magnitude = abs(fft_image);
imshow(magnitude, [])
[x, y] = ginput(2);
x = round(x);
y = round(y);
filter = ones(size(fft_image));
filter(y(1), x(1)) = 0;
filter(y(2), x(2)) = 0;
filter = ifftshift(filter);
filter = ifft2(filter);
filter = real(filter);
filtered_image = conv2(image, filter, 'same');
imshow(filtered_image, [])
```



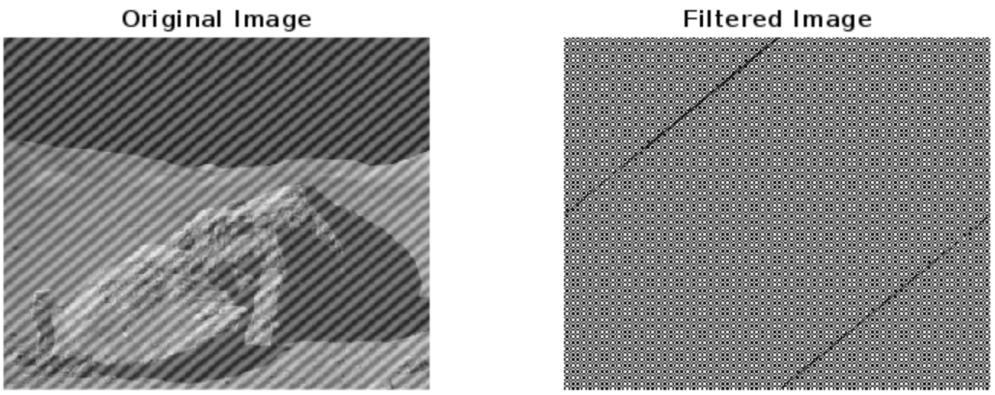


Notch Filter



```
% Load image and convert to grayscale
I = rgb2gray(imread('Moon.jpg'));
% Calculate 2D FFT of image
S = fft2(I);
% Shift spectrum so that center is at (0,0)
S = fftshift(S);
% Find location of impulses in spectrum
[max_val, max_idx] = max(S(:));
[i, j] = ind2sub(size(S), max_idx);
% Create notch filter transfer function
filter = ones(size(S));
filter(i,j) = 0;
filter(size(S,1)-i+1,size(S,2)-j+1) = 0;
% Shift spectrum back so that center is at (0,0)
filter = ifftshift(filter);
% Multiply spectrum by notch filter transfer function
S_filtered = S .* filter;
% Invert filtered spectrum
I_filtered = ifft2(S_filtered);
% Display original and filtered images
figure;
subplot(1,2,1);
imshow(I);
title('Original Image');
subplot(1,2,2);
imshow(I_filtered);
title('Filtered Image');
```

Output:

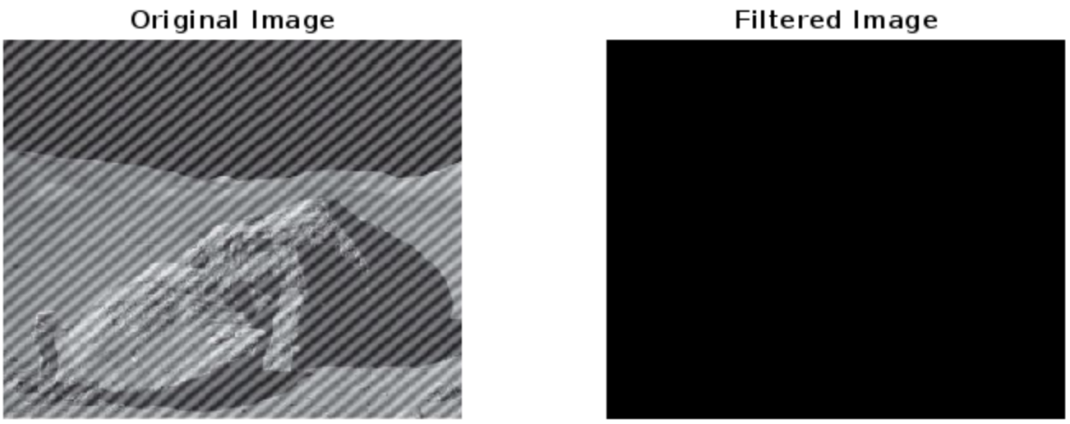


Solution steps in numeric order:

1. Load the corrupted image into MATLAB using the imread function.
2. Apply a 2D DFT to the image using the fft2 function.
3. Shift the zero-frequency component to the center of the spectrum using the fftshift function.
4. Find the location of the periodic noise in the frequency spectrum by using the find function to locate the bright dots in the spectrum.
5. Create a notch filter by setting the value of the frequency spectrum at those locations to zero.
6. Shift the zero-frequency component back to the original position using the ifftshift function.
7. Apply an inverse 2D DFT to the filtered frequency spectrum using the ifft2 function to obtain the filtered image.
8. Display the original and filtered images side by side using the subplot and imshow functions.

```
% Load the corrupted image
img = imread('Moon.jpg');
% Apply 2D DFT to the image
f = fft2(img);
% Shift the zero-frequency component to the center of the spectrum
fshift = fftshift(f);
% Find the location of the periodic noise
[y, x] = find(fshift > 200);
% Create a notch filter by setting the value of the frequency spectrum
at those locations to zero
fshift(y, x) = 0;
% Shift the zero-frequency component back to the original position
f_filtered = ifftshift(fshift);
% Apply inverse 2D DFT to obtain the filtered image
filtered_img = ifft2(f_filtered);
% Take the real part of the filtered image
filtered_img = real(filtered_img);
% Display the original and filtered images
subplot(1, 2, 1), imshow(img), title('Original Image');
subplot(1, 2, 2), imshow(filtered_img), title('Filtered Image');
```

output image:




```
% Plot the magnitude of the frequency spectrum
figure, imshow(log(abs(fshift)), []), title('Frequency
Spectrum');
```

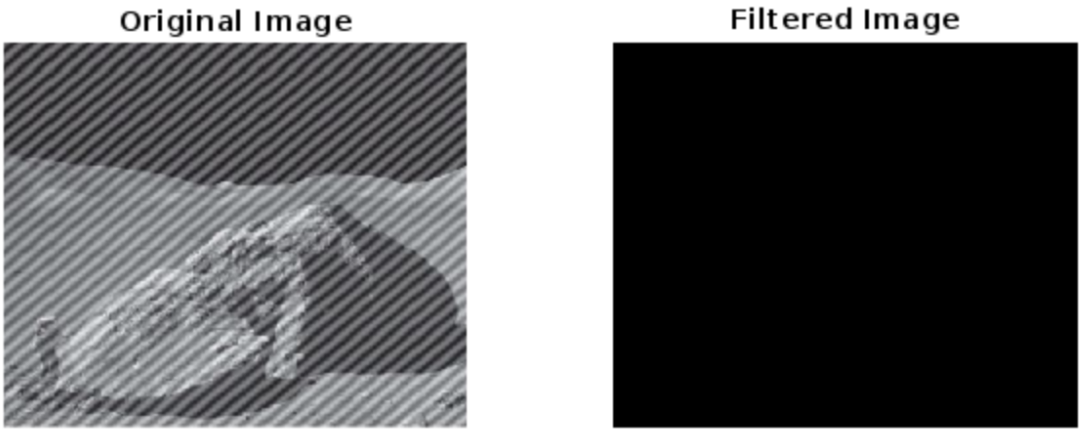
This will display the magnitude of the frequency spectrum in a separate figure. You can then use this figure to manually identify the location of the periodic noise and adjust the code accordingly.

Here is the revised code that includes this troubleshooting step:

```
% Load the corrupted image
img = imread('Moon.jpg');
% Apply 2D DFT to the image
f = fft2(img);
% Shift the zero-frequency component to the center of the
spectrum
fshift = fftshift(f);
% Plot the magnitude of the frequency spectrum
figure, imshow(log(abs(fshift)), []), title('Frequency
Spectrum');
% Find the location of the periodic noise
[y, x] = find(fshift > 200);
% Create a notch filter by setting the value of the frequency
spectrum at those locations to zero
fshift(y, x) = 0;
% Shift the zero-frequency component back to the original
position
f_filtered = ifftshift(fshift);
% Apply inverse 2D DFT to obtain the filtered image
filtered_img = ifft2(f_filtered);
% Take the real part of the filtered image
filtered_img = real(filtered_img);
% Display the original and filtered images
subplot(1, 2, 1), imshow(img), title('Original Image');
subplot(1, 2, 2), imshow(filtered_img), title('Filtered
Image');
```

This code will display the magnitude of the frequency spectrum, which you can use to identify the location of the periodic noise.

Final Output:



References

https://www.mathworks.com/

https://www.geeksforgeeks.org/noise-removal-using-median-filter-in-c/