**Digital Image Processing**

**Homework #2**

Mustafa Al-Barahmeh……0175630

Osama Salman……………0171927

Zeyad Al-Najjar ………….0175642

**Problem 1**

To solve this problem, we used predefined functions from the OpenCV library. We used *cv2.erode* to get the minimum filter mask, cv2.dilate for the maximum filter mask and cv2.medianBlur for the median filter mask. So, our code looks like this:



First, let us look at the noisy image we are working with, and the original image we are trying to obtain:

|  |  |
| --- | --- |
| Noisy image | Original image |

The functions used gave the following result:

|  |  |  |
| --- | --- | --- |
| Minimum-filtered image | Maximum-filtered image | Median-filtered image |

The minimum filter function produced an even noisier image because it amplified the noisy pixels due to their low values. The PSNR for this image when comparing it with the original clear image was (8.943), the visual corruption this filter did explains the very low value of the PSNR.

The maximum filter function produced a fairly acceptable image as it succeeded in eliminating the low-valued noisy pixels. Although, it pushed the image to become brighter than it is. The PSNR for this image was (22.621). This is an acceptable value with a good resultant image.

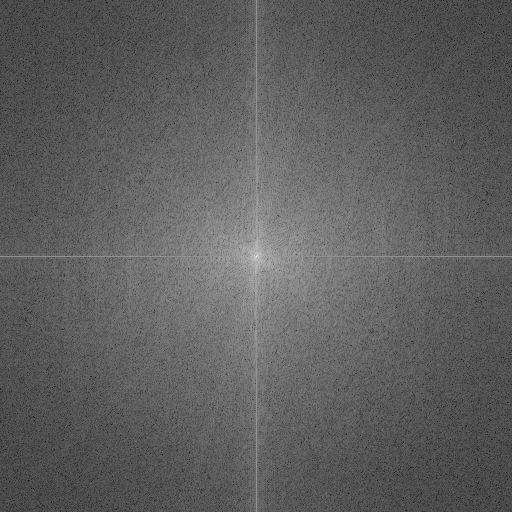
The median filter function produced the best result yet. It eliminated the noisy pixels and managed to maintain the image’s brightness. The PSNR for this image was (34.272) which is considered a high value and it suggests a high similarity between the median-blurred image and the original clear image which is visually apparent.

**Problem 2**

We read the *f16.gif* image and computed its *FFT* using the built-in functions in the numpy library. Then, we extracted the magnitudes and the phases in sperate objects to apply the necessary operations. This is how we did it:



This is how the shifted, log-scaled magnitude looks like:



Log-scaled magnitude spectrum

To calculate the total power in the image, we need the sum of the squared magnitudes of the image FFT. The DC component power can be calculated by squaring the magnitude of *F(0, 0)*. This is how we did it:



The total power was calculated to be 2346758362628097 and the DC component power 2206878835450041. The ratio of the DC component power to the total power was 94.04% which confirms that the DC component is the largest component in the spectrum.

We transformed the image back to the spatial domain excluding the magnitude property (setting it to 1) and then we scaled it linearly using this code:



This was the resultant image:

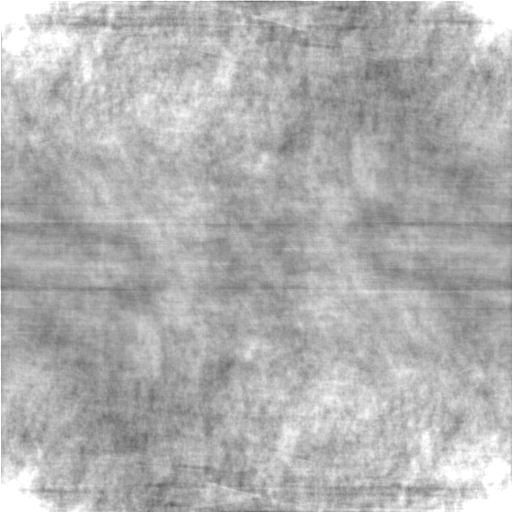


Linearly-scaled, phase-reconstructed image

We multiplied the phase component by 0.1 and reconstructed the image in the spatial domain. This is how we did it:



This is what the operation yielded:



The resultant image was viciously distorted because the phase component carries the image details. So, multiplying it by 0.1 causes a huge loss of details.

**Problem 3**

For this problem, we figured out that the notch filter can be used to produce all the other filters. So, we implemented it and the code looked like this:



This function takes multiple notches (centers, radii) as a parameter, applies each notch to the created mask, and finds and applies the corresponding opposite notch about the center.

For the ideal low-pass filter, we tested out a couple of possible radii that reduce the noise and settled on a radius of 228 pixels as it had the highest PSNR when comparing it to the original, clear image. To apply a low-pass filter of this radius, we pass this value to the notch-reject filter function along with the center set as the image’s center. The notch-reject filter returns a mask that rejects the area specified by the center and radius, so, to obtain a low-pass filter, we simply invert the mask. This is the code to this operation:



These are the filter mask obtained, the filtered spectrum, and the filtered image in the frequency domain:

|  |  |  |
| --- | --- | --- |
| Low-pass filter mask | Filtered spectrum | Filtered image |

The PSNR for the filtered image when comparing it with the original, noise-free image turned out to be (28.081) which is not that high. The filter reduced the noise in the image, but in order to do that it had to wipe out a small portion of the original image details. The ringing effect produced by this filter is quite visible.

For the ideal notch-reject filter