Exercise 2, HT2020
Local Operators and Fourier Analysis

Assignment in partial fulfilment of the requirements for the course

Computer-Assisted Image Analysis I

UPPSALA
UNIVERSITET

Master's Programme in Computer Science

Department of Information Technology

Authors:
Abraham Mathewos Meja
Marcello Pietro Vendruscolo
Shubhomoy Biswas

November 25, 2020

Contents

List of Figures

1 Question 1

Examine at least 3 different filter kernels, among which there should be at least one sharpening (edge enhancing) and one smoothing filter and apply them in different sizes to the image cameraman.png, e.g. sizes 3×3***,*** 7×7 ***and*** 31×31***. Note that some filters are only available in one size when using fspecial. Include at least three figures in your report. One showing the original image, one figure showing the image after sharpening, and one figure showing the image after smoothing. For each filter, explain what the filter does to the image, and explain the effect of the different filter sizes***

The filters are applied to the original cameraman.png image to obtain the smoothed and sharpened images, as shown in Figure 1, with Average and Log filters, respectively. The **averaging filter**, in this particular case of size 7×7, smoothes areas of sharp intensity transitions and reduces insignificant (w.r.t the size of the kernel) noise while not modifying large (w.r.t the size of the kernel) uniform regions with slowly varying intensities. It averages out the pixel intensities which prevents the high frequency variations of the image. This process results in an overall blurred image. Increasing the size of the filter increases the smoothing effect.

The **log filter**, which corresponds to the Laplacian of Gaussian filter, in this particular case of size 7×7 and $\sigma = 0.5$, enhances sharp variations in intensity and other discontinuities while setting areas of constant intensity to 0. Therefore, it does not provide directional information but only produces magnitudes. When adding the resultant image with the original image, the "crisp" image is obtained, as also shown in the Figure 1.

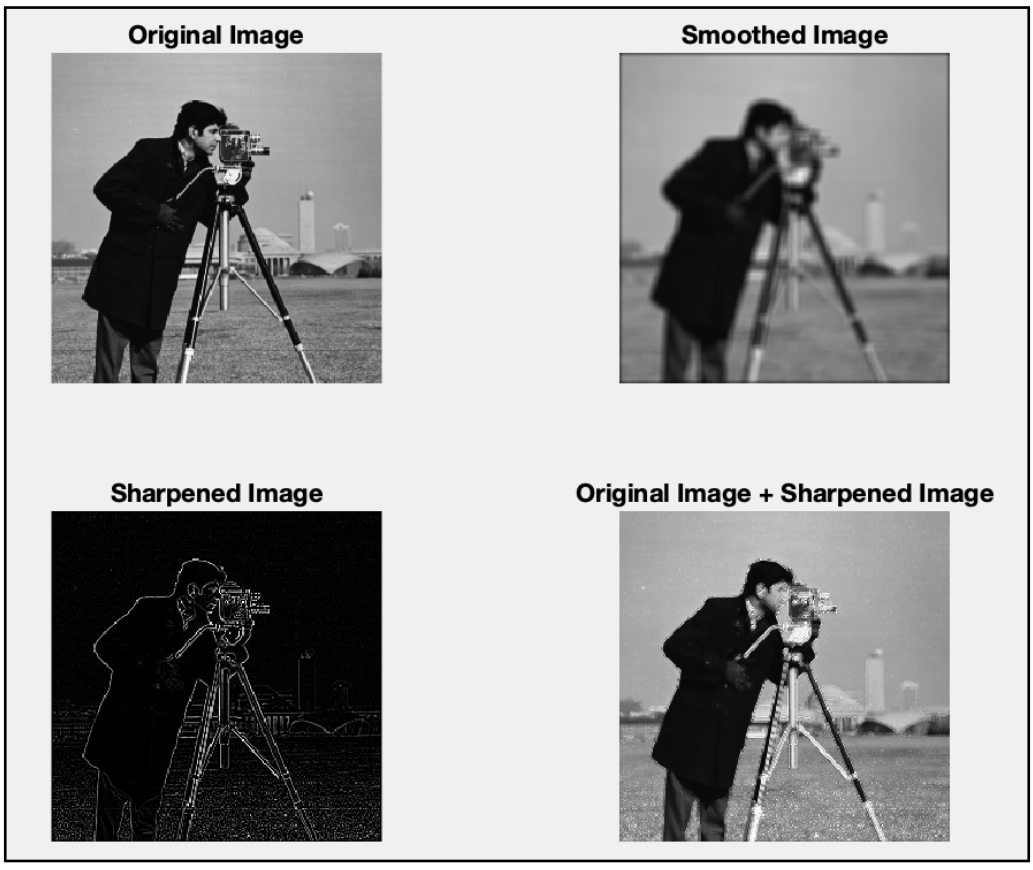

Figure 1: This figure shows the original cameraman.png image as well as the result of applying the Average and Log filters of size 7×7 to the original image, respectively.

Although the type of filter applied to an image defines the effect observed in the outcome, as noticed in Figure 1, the size of the chosen filter plays an important role in the visibility of such effect. In Figure 2, it is possible to observe the blurring effect differences caused by convolving the Average filter of size 3×3, 7×7, and 31×31 to the *cameraman.png* image, respectively. The conclusion is that, for smoothing filters, the size of the filter has a direct relationship with the blurring effect observed. Moreover, in Figure 3, it is possible to observe the difference of sharpening effect caused by convolving the Log filter of size 3×3, 7×7, and 31×31 to the *cameraman.png* image, respectively. The conclusion is that, for the Log filter, the size of the filter has a direct relationship with the edge enhancing effect observed.

Figure 2: This figure shows the outcome of convolving the Average filter of size 3×3, 7×7, and 31×31 to the cameraman.png, respectively.

Figure 3: This figure shows the outcome of convolving the Log filter of size 3×3, 7×7, and 31×31 to the cameraman.png, respectively.

2 Question 2

Are the filters with filter kernels 'average', 'disk' and 'gaussian' examples of low-pass, band-pass or high-pass filters?

The term 'low-pass filters' in spatial domain refers to filters in the frequency domain that allow frequencies below a threshold (i.e., low-frequencies) to pass while stopping frequencies above such threshold (i.e., high-frequencies), as illustrated in Figure 4. As explained in *Question 1*, sharp transitions in intensity (i.e., edges and discontinuities) can be understood as high-frequencies and by stopping them from passing, such low-pass filters smooth out edge intensity transitions and *possibly even remove noises* - depending on the ratio between the size of the noise and the size of the filter. Therefore, as the 'average', 'disk' and 'gaussian' filters are all examples of smoothing filters, as shown in Figure 5, they can be considered low-pass filters.

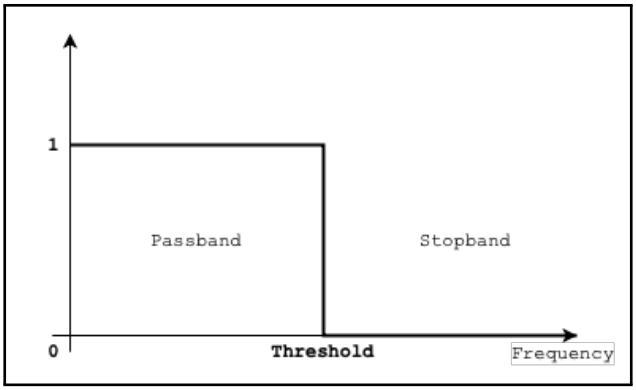

Figure 4: This figure illustrates the behaviour of a low-pass filter in the frequency domain.

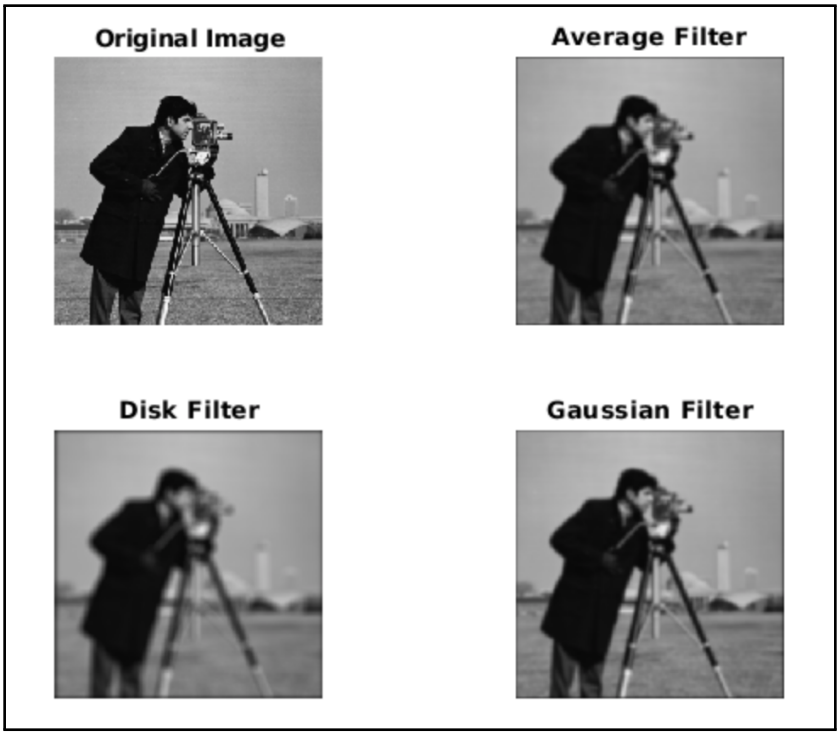

Figure 5: This figure shows the smoothing effect obtained when convolving the Average, Disk and Gaussian filters, respectively.

3 Question 3

Demonstrate how you can synthesize low-pass, band-pass and high-pass filtered images using simple arithmetics and filter kernels mentioned in Question 2.

imageAnalysis_lab2.m

```
%% Question 3

%Original image.
cameraMan_original = imread('images/cameraman.png');

%Using the Average filter as the foundation to synthesize low-pass,
%band-pass and high-pass filtered images.
smoothing_filter_average = fspecial('average',5);
smoothing_filter_average2 = fspecial('average',50);

%Low-pass filtered image can be synthesized by filtering the image directly
%with the low-pass Average filters.
cameraMan_lowpass = imfilter(cameraMan_original,smoothing_filter_average);
cameraMan_lowpass2 = imfilter(cameraMan_original,smoothing_filter_average2);

%High-pass filtered image can be synthesized by subtracting the low-frequencies
%from the original image. This can be achieved by subtracting the
%cameraMan_lowpass image from the original image, remaining with high
%frequencies.
cameraMan_highpass = cameraMan_original - cameraMan_lowpass;

%Band-pass filtered image can be synthesized by subtracting the low and
%high frequencies from the original image. Note that two different sizes of
%smoothing filtering were utilised to get a difference
cameraMan_bandpass = (cameraMan_original- cameraMan_lowpass2 - cameraMan_highpass);

figure
subplot(1,3,1), imshow(cameraMan_lowpass); title('Low-pass Image');
subplot(1,3,2), imshow(cameraMan_bandpass); title('Band-pass Image');
subplot(1,3,3), imshow(cameraMan_highpass); title('High-pass Image');

```

Figure 6: Matlab code for synthesizing low-pass, band-pass and high-pass filtered images using simple arithmetics and filter kernels.

Figure 7: Resultant images obtained by applying the above filters - low-pass, band-pass and high-pass - on cameraman.png image.

4 Question 4

The Matlab function $fspecial$ ***can produce filter kernels for Sobel filters. Use this functionality to demonstrate Sobel filtering on cameraman.png and wagon.png.***

The **Sobel filter** generated by the $fspecial$ function in Matlab, named Sobel Horizontal-Top in this exercise, identifies sharp intensity transitions (i.e., edges, discontinuities and noise) in the vertical descent direction when moving from light to dark regions. However, the rotation of this Sobel Horizontal-Top filter by $180°$ creates another version of the Sobel filter, named Sobel Horizontal-Bottom in this exercise, which identifies sharp intensity transitions in the vertical descent direction when moving from dark to light regions. The transpose of the Sobel Horizontal-Top filter creates a third version of the Sobel filter, named Sobel Vertical-Left in this exercise, which identifies sharp intensity transitions in the horizontal direction (left to right) when moving from light to dark regions. However, the rotation of this Sobel Vertical-Left filter by $180°$ creates the forth version of the Sobel filter, named Sobel Vertical-Right in this exercise, which identifies sharp intensity transitions in the horizontal direction (left to right) when moving from dark to light regions. Figure 8 and Figure 9 shows the outcome images after applying each of the mentioned Sobel filters to the *cameraman.png* and *wagon.png* images, respectively. It is relevant to mention that each of the obtained responses provide directional information (i.e., edge direction).

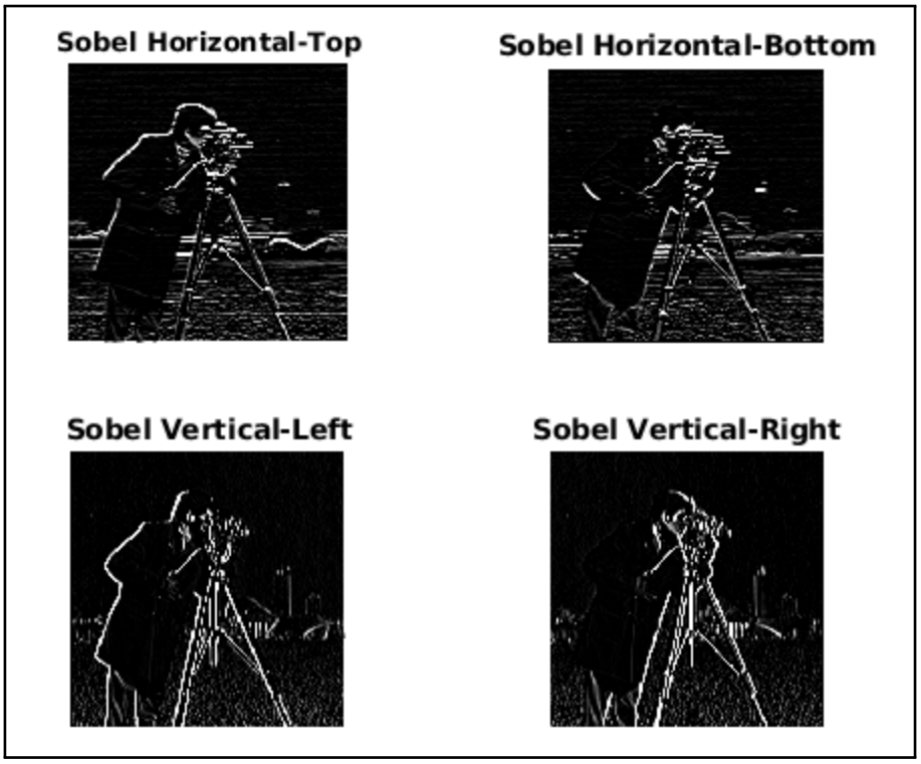

Figure 8: This figure shows the outcome images after filtering the cameraman.png image with the four possible directions of the Sobel filter.

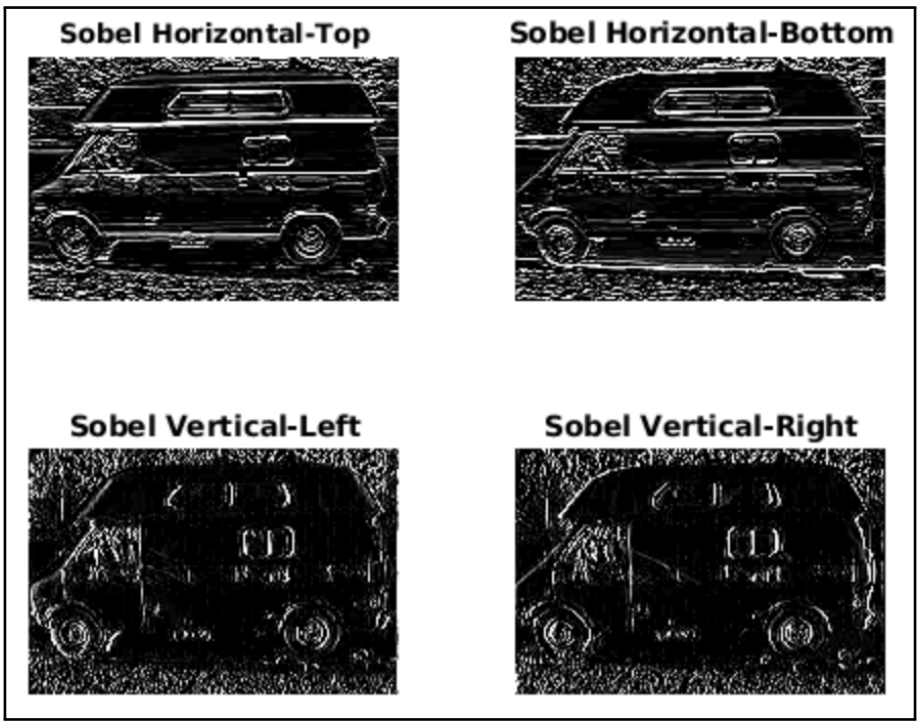

Figure 9: This figure shows the outcome images after filtering the wagon.png image with the four possible directions of the Sobel filter.

However, the individual responses obtained after filtering an original image with the four Sobel filters are usually consolidated to providing more outstanding edges at the cost of losing the directional information. The combination of the of all four possible Sobel images for the *cameraman.png* and *wagon.png* images are shown in Figure 10 and Figure 11, respectively.

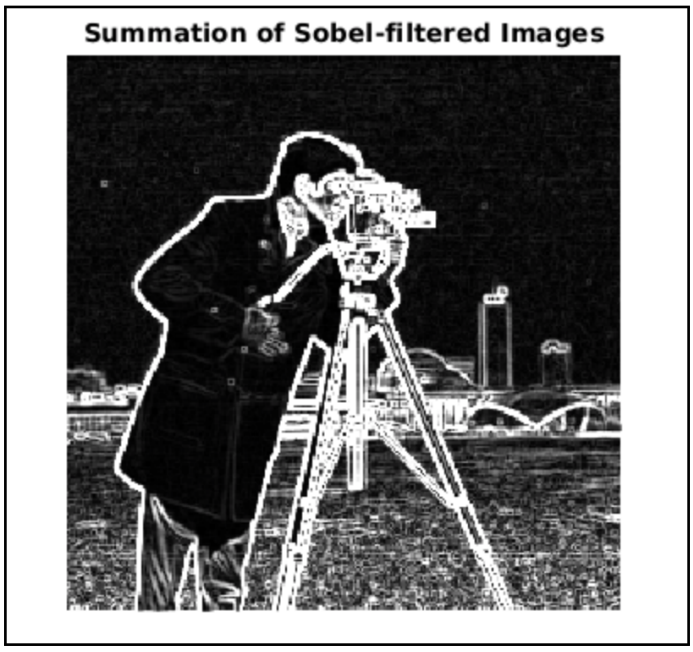

Figure 10: This figure shows the combination of all four possible Sobel images for the cameraman.png image.

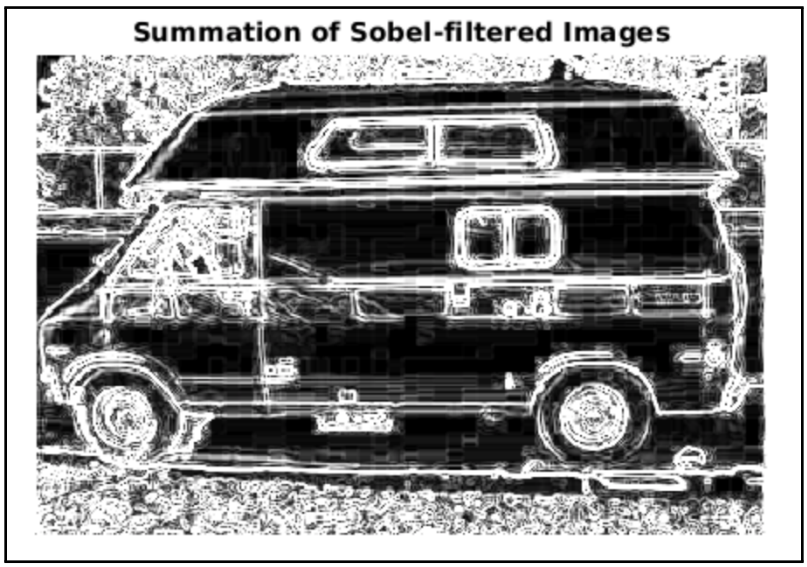

Figure 11: This figure shows the combination of all four possible Sobel images for the wagon.png image.

5 Question 5

Open the image wagon_shot_noise.png. Perform median filtering on the image using different sizes of the filter masks.

Figure 12 shows the original *wagon_shot_noise.png* image, which is corrupted with the well-known salt-and-pepper noise, and the outcome images after performing the median filtering using kernels of size 3×3, 7×7 and 11×11, respectively. It is relevant to mention that using the 3×3 median filter was enough to remove the noise while the 7×7 and 11×11 median filters were large enough to start removing image features.

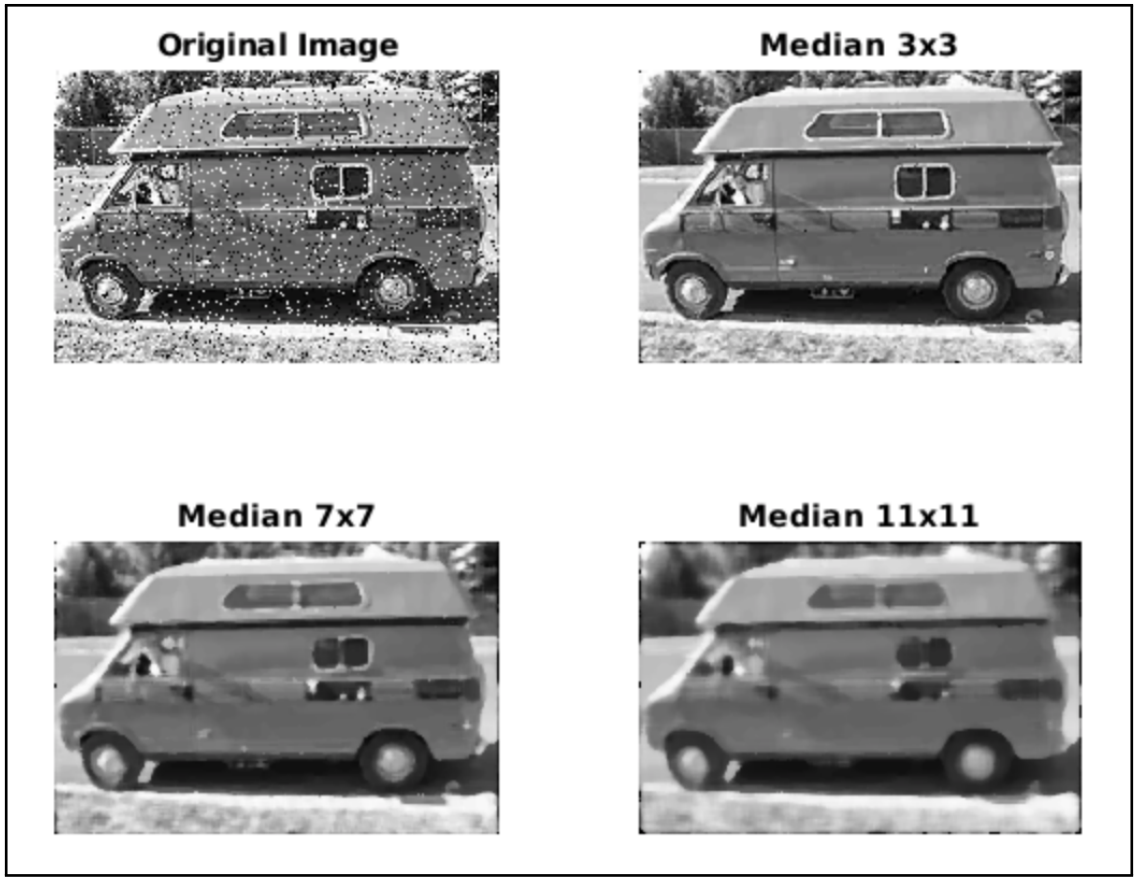

Figure 12: This figure shows the original wagon_shot_noise.png image as well as the outcome images after performing the median filtering with kernels of different sizes.

6 Question 6

Compare visually the effect of median filtering to the effect of mean and Gauss filtering. Explain the differences on the image wagon_shot_noise.png. How does median filtering work compared to mean and Gauss filtering?

While the mean and the Gaussian lowpass filters reduced the noise to some extent, but it also considerably blurred the original image. The median filter showed a more high-grade performance regarding noise reduction without blurring the original image as much as the former filters. The superiority of the median filter over the mean and gaussian low-pass filters, in this particular case with the appearance of impulse noise, is made evident in Figure 13, which shows the original image as well as the outcome of each filtering operation.

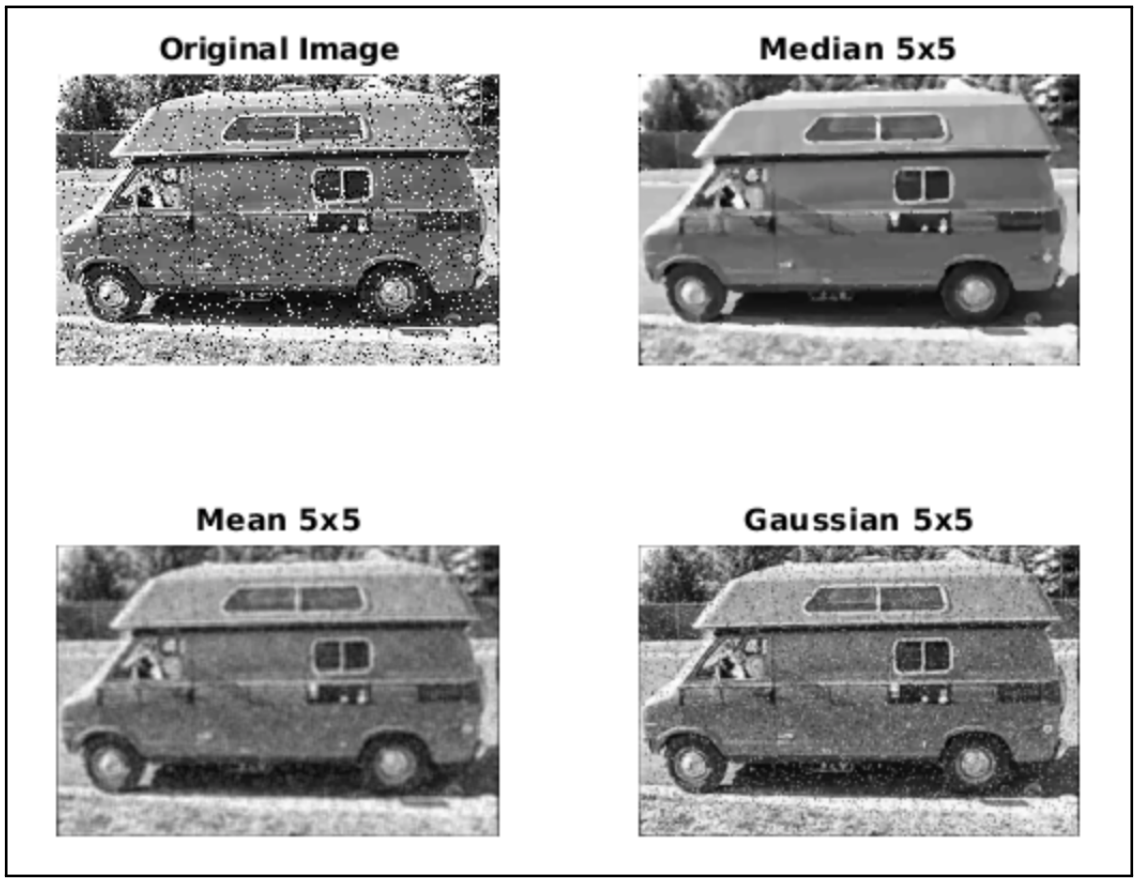

Figure 13: This figure shows the original image as well as the output images of the filtering with the Median, Mean and Gaussian filters, respectively. The size of the kernels was 5×5 in all cases.

The observed behaviour is not surprising though. The resulting intensity value of a given pixel when using the mean or the Gaussian lowpass filters is computed by performing a weighted sum of the original intensity value of the pixel with the intensity value of the surrounding pixels (i.e., neighbourhood). The weight assigned to each pixel depends on the chosen filter and this distribution of weights across the kernel is what produces different behaviours. On the other hand, the response (i.e., the resulting intensity value of a given pixel) when using the median filter, which can be categorised as an order-statistic filter, depends on the ranking of the pixels comprised in the area encompassed by

the kernel. The median, also known as the 50th percentile, represents the pixel intensity value located in the middle of the array of intensity values after sorting is performed. In this manner, the median filter completely excludes the intensity value of outlier pixels that end up being located at the extremities of the sorted array.

This is the theory detail supporting the aforementioned better noise reduction performance when comparing the types of filter. Moreover, for relatively small size median filters, the blurring effect is removed as the filter does not compute the weighted sum of intensity values in the neighbourhood of a given pixel.

7 Question 7

In general the median filter is more time consuming, why?

The median filter is a nonlinear filter based on the ranking of the pixels comprised in the area encompassed by the kernel. Depending on the algorithm design, though naive, convolving the median filter can be more time consuming than other filters as it requires sorting every pixel in the window to find the median. Experimentally shown in Figure 14, for small kernel sizes such as 3×3 and 5×5, the time difference taken to convolve each of the two different kernels was not significant. However, as the size of the kernels grew, this difference became more significant and that is the reason why the size of the kernels in Figure 14 was set to 9×9.

Nevertheless, depending on the implementation and the chosen sorting algorithm, the median filter can be more or less time-consuming. It is relevant mentioning that the median filter is a non-separable filter which, in comparison to separable filters that are convolved in their simpler form, requires a longer time to computationally process the convolving operation.

imageAnalysis_lab2.m

```
87      %% Question 07:
88 -    wagon_shot_noise_original = imread('images/wagon_shot_noise.png');
89
90 -    smoothing_filter_average = fspecial('average',9);
91
92 -    start_time_average = tic;
93 -    a = imfilter(wagon_shot_noise_original,smoothing_filter_average);
94 -    elapsed_time_average = toc(start_time_average);
95
96 -    start_time_median = tic;
97 -    b = medfilt2(wagon_shot_noise_original,[9 9]);
98 -    elapsed_time_median = toc(start_time_median);
99
100 -   str1 = sprintf('The time taken for convolving the Average filter was %0.5e',elapsed_time_average);
101 -   str2 = sprintf('The time taken for convolving the Median filter was %0.5e\n',elapsed_time_median);
102
103 -   disp(str1)
104 -   disp(str2)
105
```

COMMAND WINDOW

New to MATLAB? See resources for Getting Started.

```
The time taken for convolving the Average filter was 8.88000e-04
The time taken for convolving the Median filter was 1.82100e-03

The time taken for convolving the Average filter was 8.87000e-04
The time taken for convolving the Median filter was 1.69000e-03

The time taken for convolving the Average filter was 9.00000e-04
The time taken for convolving the Median filter was 1.65800e-03

>>
```

Figure 14: This figure shows the difference in time taken to convolve the Average filter and the Median filter across the same image, both with kernel size of 9×9.

8 Question 8

Implement your own code for 3x3 median filtering. You may use the Matlab function median that computes the median element of a vector. Use for instance two nested for-loops to iterate your filter for every neighborhood in the image. The exact behavior on the borders is not so important for this exercise and you may cut some corners here if it helps you.

```
I = imread('images/wagon_shot_noise.png');
pad = padarray(I, [1, 1], 'both');

new_img = [];
for r = 1:size(I, 1)
    for c = 1:size(I, 2)
        r_bottom = r + 2;
        c_right = c + 2;

        m = median(median(pad(r:r_bottom, c:c_right)));
        new_img(r,c) = m;
    end
end

subplot(1,2,1); imshow(I); title('Original');
subplot(1,2,2); imshow(uint8(new_img)); title('Median Filter');
```

Figure 15: Code snippet for applying 3x3 Median filter.

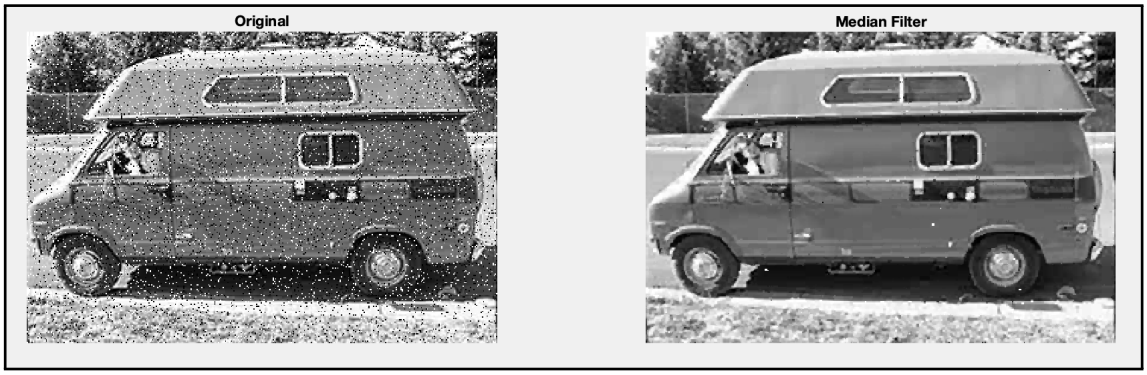

Figure 16: This figure shows the original image of wagon with noise and it's corresponding image after applying the 3x3 Median from the above code.

9 Question 9

If you implement a Gaussian filter using a large filter mask (and a large standard deviation), why do you get a black border around the image?

Figure 17 portrays the concepts enunciated in this question by exposing the consequences to the black border around the image by:

1. Keeping the kernel size fixed and changing standard deviation.
2. Keeping the standard deviation and changing the kernel size.

From the result obtained, it is possible to conclude that increasing the standard deviation as well as increasing the kernel size (within a ratio boundary) leads to a more pronounced black border around the image.

The explanation can be derived from the shape a normal distribution has and how a Gaussian filter fits in. A small standard deviation indicates that the data is closely distributed around the mean of the distribution while a large standard deviation indicates that the data is dispersed out over a wider range. Therefore, given a constant mean, *an increase in the standard deviation indicates a widening of the curve's slope* and, consequently, more weights are assigned towards the extremities (i.e., boundaries) of the Gaussian filter. Thus, when the kernel is placed near the ends of an image, the black padding pixels become more meaningful in the weighted sum, resulting in a darker pixel. In the second scenario where the standard deviation is kept constant but there is an increase in the kernel size, one can understand that, as long as the standard deviation is large enough (i.e., about 1/6 of the kernel size) to avoid assigning 0 to the coefficients of the filter, such increase results in more black padding pixels included in the average sum, which also results in a darker pixel. Therefore, by increasing the standard deviation and the kernel size, the black border effect around the image is enhanced.

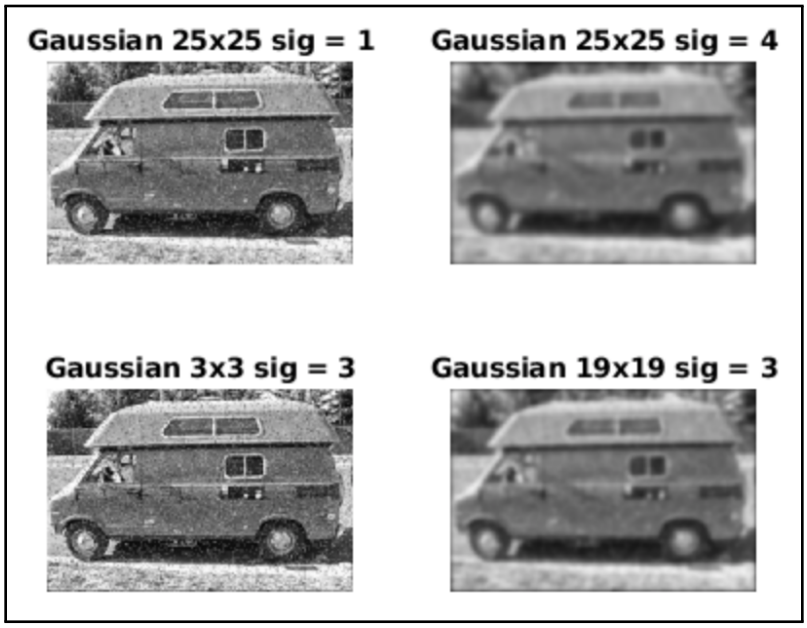

Figure 17: This figure illustrates the relationship between the standard deviation and kernel size parameters of a Gaussian filter and the resulting black border effect that can be observed when filtering an image with this type of filter.

10 Question 10

Repeat the same procedure with the image cameraman.png. Comment on the spectrum that you see and compare them to the ordinary representation of the image. You may also try other images, for example circle.png or rectangle.png

To expand the understanding of the relationship between the appearance of images in the spatial domain and the visual aspect of their corresponding shifted and enhanced Fourier transform (FT), the process of (1) computing the fast Fourier transform (FFT), (2) shifting the 0 frequency to the middle of the image, and (3) enhancing numbers close to 0 was repeated for simple images, such as rectangle.png, circle.png and lines.png, are shown in Figures 18, 19, and 20, respectively.

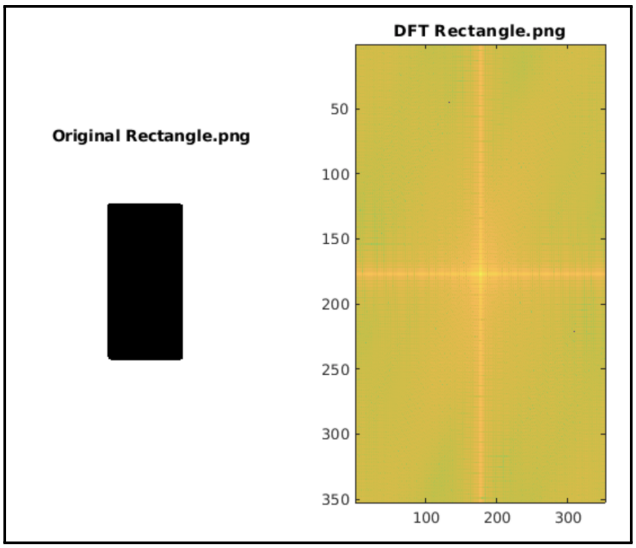

Figure 18: This figure shows the original rectangle.png image and its corresponding Fourier transform after shiftting the 0 frequency to the middle and enhancing numbers close to 0.

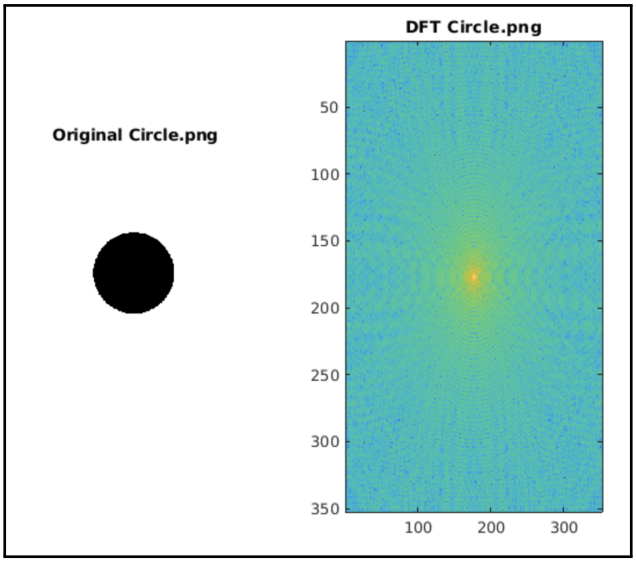

Figure 19: This figure shows the original circle.png image and its corresponding Fourier transform after shiftting the 0 frequency to the middle and enhancing numbers close to 0.

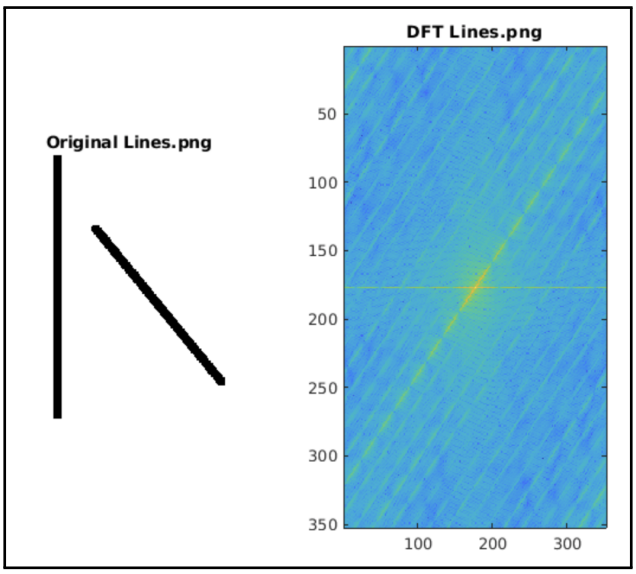

Figure 20: This figure shows the original lines.png image and its corresponding Fourier transform after shiftting the 0 frequency to the middle and enhancing numbers close to 0.

The ordinary representation of the *cameraman.png* image and its corresponding 2D (two dimensional) discrete-space Fourier transform (spectrum), shifted and enhanced, can be observed in Figure 21. In the centre of the spectrum, it is possible to notice a group of low-frequency image features. Furthermore, it is possible to observe diagonal prominent streaks that correspond to inclined sharp edges in the ordinary representation of the image, such as the front and back of the man's coat and the tripod legs edges. Moreover, the strong vertical line in the Fourier space can potentially be associated with edges in the horizontal direction in the ordinary representation of the image, such as the sharp edge transition located at the man's shoulder and other weaker horizontal edges located at the background with the buildings. These streaks in the frequency domain are associated with the frequency of the edges in the spatial domain.

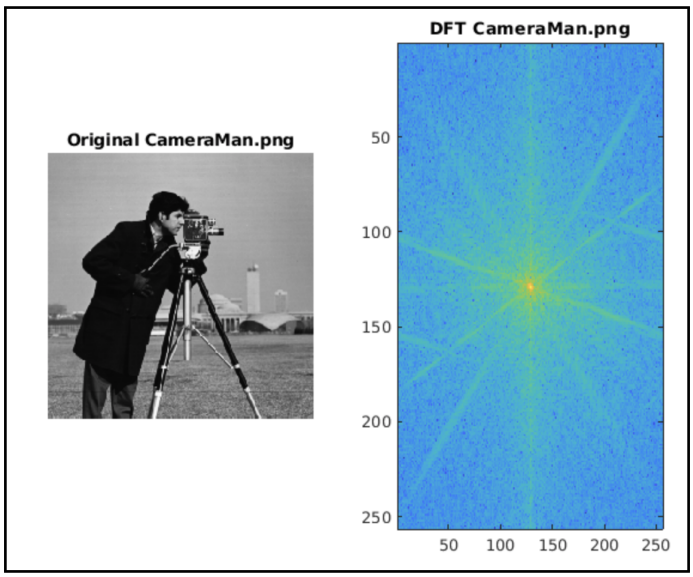

Figure 21: This figure shows the original cameraman.png image and its corresponding Fourier transform after shiftting the 0 frequency to the middle and enhancing numbers close to 0.

11 Question 11

a) Experiment with FFT of an odd and even length signal (image) of small length. For creating such a signal (image) use simple commands fftshift(fft2(rand(1,5))) and fftshift(fft2(rand(1,6))). What are the characteristic features of the result in each case?

The resulting signal vector is composed of imaginary-valued numbers. In the odd-length signal vector (size = 2n + 1), where each element on the left-half of the vector (i.e., index < n for 0-indexed array) has the corresponding conjugate in the mirror-reflected index right-half (i.e., conjugate index = size - index - 1) of the vector. Moreover, the element at the middle of the vector (index = n) has the imaginary part equals 0. The even-length signal vector presents similar behaviour, but with the consideration that the first element of the vector does not have a conjugate on the right-half of the vector, since the vector size modulus two is different from zero. The first element of the even-length vector also has the imaginary part equals 0. Figure 22 shows the characteristic features described.

```
COMMAND WINDOW
Shifted odd frequency signal of size 5 before filtering:
  -0.2053 + 0.1563i  -0.7973 + 1.0630i   2.6651 + 0.0000i  -0.7973 - 1.0630i  -0.2053 - 0.1563i

Shifted even frequency signal of size 6 before filtering:
   0.5614 + 0.0000i  -0.4430 - 0.5144i   0.0483 + 0.8334i   1.6365 + 0.0000i   0.0483 - 0.8334i  -0.4430 + 0.5144i
```

Figure 22: This figure illustrates the result of generating an odd-sized and a even-sized uniformly distributed random signals, respectively.

b) Filter the resulting, complex valued, vectors you get so that you get a real valued signal when you transform it back using ifft2. For instance, you can blank out (set to 0) certain frequencies and thereby perform filtering. You will need to take into account the characteristic features when doing this. Write down your findings.

Given the characteristics of the signal vectors previously discussed, the findings are (1) that filtering (i.e., setting to 0) the first and last elements of the frequency vector of odd length has produced a real-valued vector when inverse Fourier transforming the signal, and (2) that filtering the second and last elements of the frequency vector of even length has produced a real-valued vector when inverse Fourier transforming the signal. The findings are illustrated in Figure 23.

```
238      % Do filter here
239 -    frequency_odd_signal(1) = 0; frequency_odd_signal(5) = 0;
240 -    frequency_even_signal(2) = 0; frequency_even_signal(6) = 0;
241
242      % Transfer back to spatial domain from frequency domain
243 -    spatial_odd_signal = ifft2(ifftshift(frequency_odd_signal));
244 -    spatial_even_signal = ifft2(ifftshift(frequency_even_signal));
245
246 -    disp('Inverse Fourier transformed odd signal after filtering:')
247 -    disp(spatial_odd_signal)
248 -    disp('Inverse Fourier transformed even signal after filtering:')
249 -    disp(spatial_even_signal)
250
COMMAND WINDOW
Inverse Fourier transformed odd signal after filtering:
    0.2141    0.8389    1.0410    0.5411    0.0301

Inverse Fourier transformed even signal after filtering:
    0.3824    0.4278    0.5989    0.1631    0.1177   -0.0533
```

Figure 23: This figure illustrates the findings discussed.

12 Question 12

Now modify the FFT representation of cameraman.png, setting certain frequencies to 0, to create a low-pass version of the image. Use a circular symmetric filter for best result, but feel free to simplify the task a square pattern of your filter. You may blank out a part of a matrix using slices, e.g. A(20:30, 50:60) = 0. The result image should be real valued after performing ifft2. You are not allowed to use the functions real or abs or similar ways to force a real valued result. In your report, clearly explain your algorithm or include your code and comment.

```
246   %% Question 12
247
248 -  cameraman_original = imread('images/cameraman.png'); %Original image spatial domain representation
249 -  original_shifted_freq_signal = fftshift(fft2(cameraman_original)); %Fourier Transform
250 -  highpass_shifted_freq_signal = original_shifted_freq_signal; %Create a copy of Fourier Transform result
251
252 -  [nrows, ncols] = size(highpass_shifted_freq_signal); %Compute size of the Fourier Transform matrix
253
254   %The next conditional block computes the middle row and column
255 -  if mod(nrows, ncols) == 0
256 -      central_r = nrows/2 + 1;
257 -      central_c = ncols/2 + 1;
258 -  else
259 -      central_r = nrows/2;
260 -      central_c = ncols/2;
261 -  end
262
263 -  highpass_shifted_freq_signal(central_r-20:central_r+20, central_c-20:central_c+20) = 0; %Highpass filtering
264 -  lowpass_shifted_freq_signal = original_shifted_freq_signal - highpass_shifted_freq_signal; %Lowpass by simple arithmetics
265 -  output_image = ifft2(ifftshift(lowpass_shifted_freq_signal)); %Inverse Fourier Transform back to spatial domain
266
267 -  imshow(uint8(output_image)) %Show result
```

Figure 24: This figure shows the commented code which was implemented to solve this question.

Figure 25: This figure shows the output cameraman image obtained after Inverse Fourier Transforming the frequency signal filtered with the created lowpass-filter.

13 Question 13

Create a filter in the frequency-domain that suppresses the pattern in freqdist.png, but leaves the rest of the image as intact as possible. What does the filter look like? What do you see in the filtered image? Like the previous question, the result image should be real valued after performing ifft. You are not allowed to use the functions real or abs or similar ways to force a real valued result

```
I = imread('images/freqdist.png');
ft = fftshift(fft2(I));
[rows, cols] = size(ft);

F = fft2(double(I));
freq_dist = fftshift(log(1 + abs(F)));

subplot(2,2,1);imshow(I);title('Original');
subplot(2,2,2);imshow(freq_dist, []);title('Frequency Domain');

norm = fftshift(F);

norm(100:103, 91:94) = 0;
norm(100:103, 108:111) = 0;
norm(157:160, 149:151) = 0;
norm(157:160, 165:168) = 0;

op = ifft2(ifftshift(norm));

subplot(2,2,3);imshow(op, []);title('Image after ');
```

Figure 26: Matlab implementation of creating a filter in the frequency-domain that removes the pattern in freqdist.png.

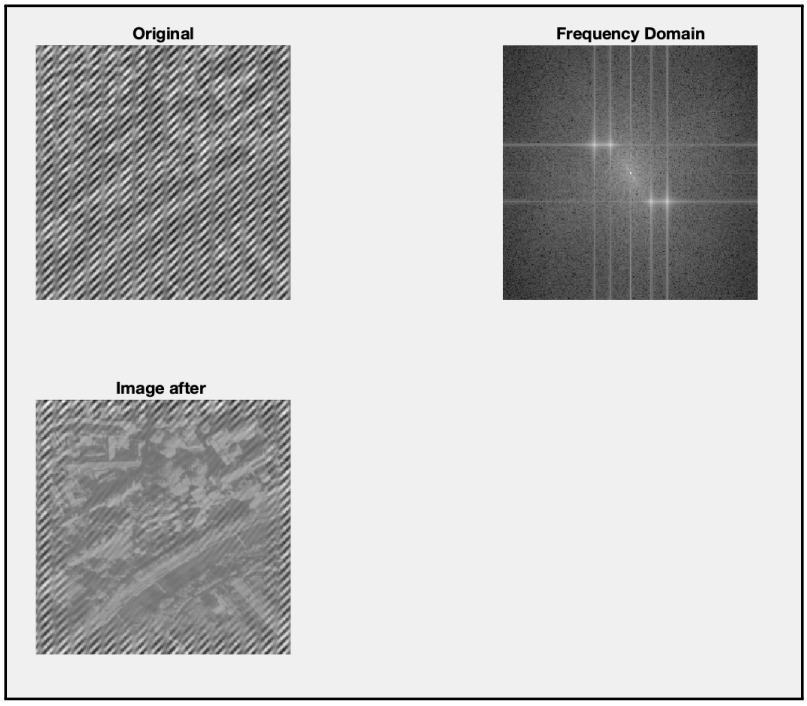

Figure 27: This figure shows original image, it's frequency domain and the resultant image after removing the high frequency spots in the frequency domain.