Image & Video Processing: Assignment 3

Kyla Kaplan
April 2024

Contents

1 Frequency Domain Processing

1 Gaussian Filtering

For the first exercise we are asked to implement a Gaussian filter in both the spatial and frequency domains. We are asked to demonstrate that convolving an image with a Gaussian filter with standard deviation σ_s in the spatial domain is equivalent to point-wise multiplication in the frequency domain with Gaussian filter with standard deviation $\sigma_f = \frac{1}{2\sigma_s \pi}$. Firstly we are asked to recreate the image below, being similar to the one given in the assignment, with size 1024×1024. We are also asked to assume padding of zero values.

Firstly, in order to recreate the given image, of a black square with a gray-ish border, I proceeded with the following code:

```
% given image size
image_size = 1024;

% intensity 0.8 (for the gray ones)
img = 0.8 * ones(image_size, image_size);

% black square
center_start = image_size / 4;
center_end = 3 * image_size / 4;
img(center_start:center_end, center_start:center_end) = 0;
```

The full code for this exercise can be found in `ex1.m`, `spatial_gaussian.m`, `frequency_gaussian.m` and `bonus_ex1.m`, respectively.

1.1 Spacial Domain

Similarly to the previous assignment, I set the kernel size of the spatial filter to be:

```
kernel_s = 4*sigma_s+1; %sigma_s = 5
```

in order to ensure the size of the kernel fully encapsulates most of the Gaussian distribution. I set $\sigma_s = 5$ for the following examples. The Gaussian kernel in this case is small, as it is a 21×21 matrix

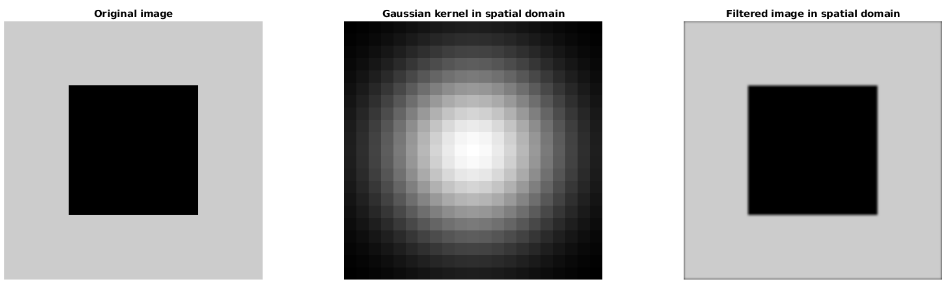

Figure 1: Results of Spacial Filtering: Original (left) vs. Result (right)

This part of the code is quite similar to a previous assignment we had completed.

```
function [img_spacial, kernel_s, gf_spatial] = spatial_gaussian(img, sigma_s)
    kernel_s = 4*sigma_s+1;
    gf_spatial = fspecial('gaussian', kernel_s, sigma_s);
    img_spacial = conv2(img,gf_spatial,'same');
end
```

1.2 Frequency Domain

For the transformation in the frequency domain , σ_f is initialized to be:

```
    sigma_f = (1/(2*pi*sigma_s));
```

While retaining the same sigma value, and running the following code (can be found in the corresponding file named `frequency_gaussian.m`:

```
function [img2_filtered,img_2dft,gf_freq,img2_filtered_fft]=frequency_gaussian(img,sigma_s)
    ...
    % Convert to frequency domain (2D Fourier Transformation)
    img_2dft = fft2(img);
    % Shift lower frequencies to the center of fourier spectrum (shift the DC component)
    img_2dft = fftshift(img_2dft);

    % Computing gaussian filter
    [X,Y]= meshgrid(-size(img,1)/2:size(img,1)/2-1,-size(img,2)/2:size(img,2)/2-1);
    D = sqrt(X.^2 + Y.^2);
    D = D/max(D(:));
    D = exp(-D.^2/2/sigma_f^2);
    gf_freq = D/max(D(:)); %normalize D again jic

    % Perform gaussian filtering in frequency domain (point-wise operation)
    img2_filtered_fft = img_2dft .* gf_freq;

    % inverse fourier transformation (and shift back DC component)
    % Bring image back to spatial domain
    img2_filtered = ifft2(ifftshift(img2_filtered_fft));
end
```

The function `fft2` (as shown in class) computes the Fourier transform of the input image (`img` in this case), and `fftshift` shifts the DC (a.k.a. zero-frequency) component to the ”center” of the Fourier spectrum (which I save in `img_2dft`). Going forward, we compute the Gaussian Filter kernel by creating a meshgrid with the appropriate coordinates (which match the dimensions of `img`). I use the variable `D` to store the computed radial distances from the center of the grid, and normalizing it afterwards. Then we compute the actual Gaussian Filter `gf_freq` based on the squared radial distance and std, and then we normalize it again to ensure we preserve the original image intensity. We apply the filter to the image in the frequency domain in a point-wise manner, and then we use the Inverse Fourier Transformation functions `ifftshift` and `ifft2` to shift the frequency spectrum back to the original arrangement, and to compute the inverse 2D transformation of the image to bring it back to the spatial domain.

When visualizing the results we utilize the `log` and `abs` functions to discern the intensity levels of the high and low components, as well as using absolute values for all points that could be complex numbers as well. In `ex1.m`, with results before:

```
    imshow(log(abs(img_2dft)));
    title('FFT of padded and shifted image');
    imshow(gf_freq);
    title('Gaussian kernel for frequency domain');
    imshow(log(abs(img2_filtered_fft)));
    title('FFT of Filtered image in frequency domain');
    imshow(img2_filtered);
    title('Filtered image in frequency domain');
```

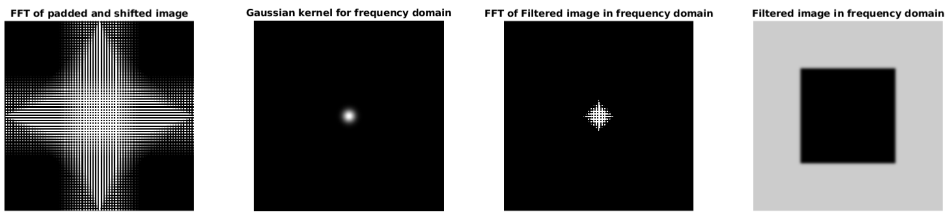
FFT of padded and shifted image
Gaussian kernel for frequency domain
FFT of Filtered image in frequency domain
Filtered image in frequency domain

Figure 2: Interim processes of applying the Gaussian filter in the frequency domain

Seeing both of the transformations below we can compare the results visually:

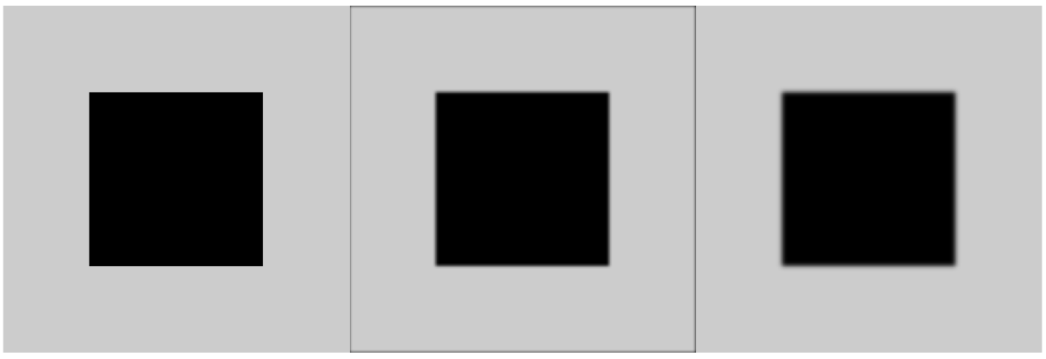

Figure 3: Original(left), Spatial transformation (middle), Frequency domain transformation (right)

As is visible above, both approaches result in undeniably similar results.

1.3 Bonus

For the bonus we are invited to measure and compare both of the filtering techniques based on their complexity in terms of time. The relevant code can be found in `bonus_ex1.m`.

```
for i = benchmark_sigmas % benchmark_sigmas = 1:100;
    tic;
    spatial_gaussian(img, i);
    time1 = toc;
    tic;
    frequency_gaussian(img, i);
    time2 = toc;
    results(i, :) = [i, time1, time2];
end
```

As we can see from Figure 4 below, the frequency domain filtering process is faster than doing the transformation in the spatial domain for σ values over 10. This is in line with the expectation that the spatial domain depends on the size of the images kernel, and for the frequency domain the computational complexity depends on the Fourier spectrum of the image. Thus, we can consider the frequency domain filtering to be an especially beneficial technique for cases when multiple filters need to be applied on the same image, as the complexity would stay linear throughout.

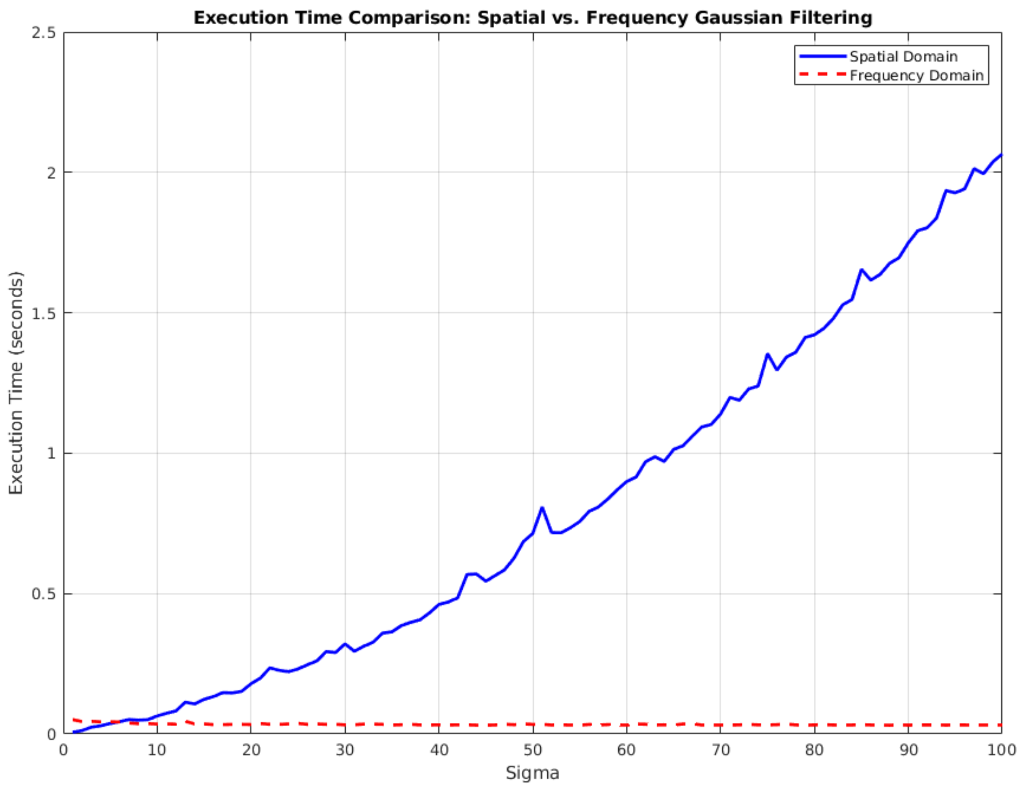

Figure 4: Benchmarking results

2 Image Restoration

For this part of the assignment we are asked to remove a ”checkerboard-looking” pattern from the given image `san_domenico.png` shown in Figure 5. By looking closer we can see from the image that the checkerboard pattern seems to be a slightly rotated periodic signal, which in turn implies that we could map that repetitive nature to certain points in the frequency domain. As we can see below, the points are indeed visible, and as a way to remove them, we can create some circular overlayed points/masks at the same positions in order to ”cover” them (almost close/similar to a notch filter due to the implemented condition below). The full code for the assignment can be found in the respective file `ex2.m`.

```
img_fft = fft2(img);
[height, width, ~] = size(img); % get the height and width
i_select = abs(fftshift(img_fft)/(height/2)); % for better visualization
[x, y, ~] = impixel(i_select); % get the pixels
mask = ones(height, width); % create the mask
radius = 10; % set a radius (can be changed and fine-tuned)
[h, w] = meshgrid(1:height); % create a grid of the same dimensions
for i = 1:size(x)
    mask((h-x(i)).^2+(w-y(i)).^2<radius^2) = 0; %ensure circles
end
res = img_fft .* fftshift(mask); % point-wise mask application
result = abs(ifft2(res));
```

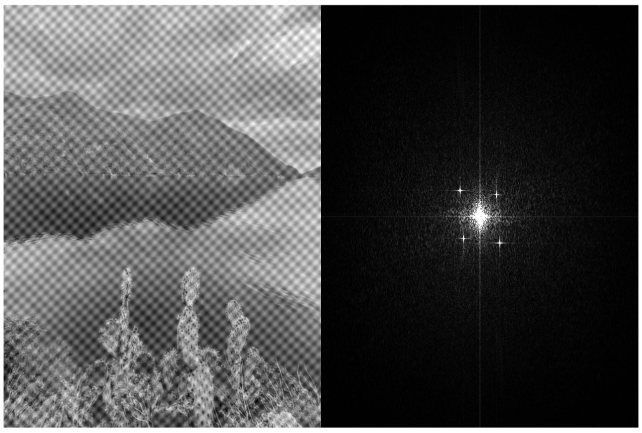

Figure 5: `san_domenico.png` (left) vs. its FFT (right)

In this implementation I utilized a similar function we had used in the first assignment, `impixel()` to be able to click on the "points" in the FFT image, and according to the pre-set radius — and mask it accordingly. The mask itself (below in Figure 6), is just manufactured circles surrounding the selected points, and directly covering them in the frequency domain.

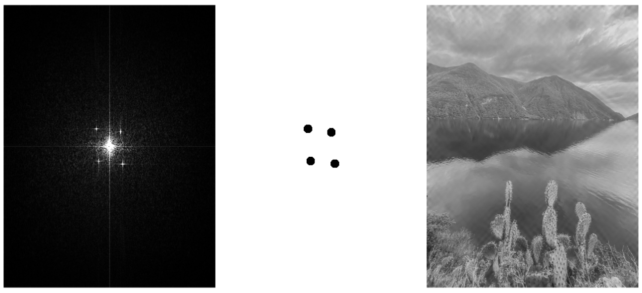

Figure 6: FFT of image (left), Manually applied mask (middle), resulting image (right)

The application in the freq. domain can be found in the submission file titled `ex2_fft_interim.png`.

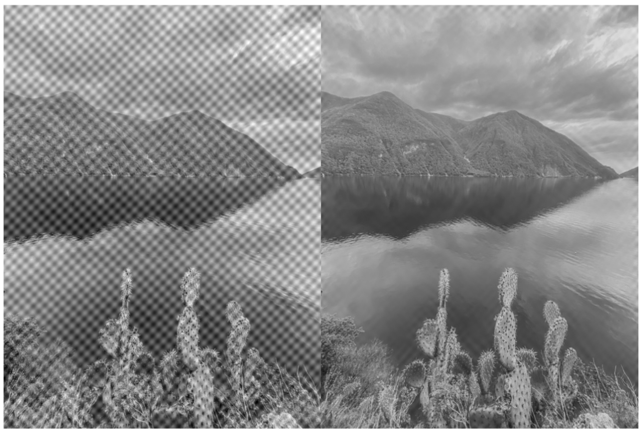

Figure 7: `san_domenico.png` (left) vs. the result (right)

The result seems to contain some slight artefacts around the border of the original pattern, but the majority of the image has been properly restored and preserved.