Image and Video Processing Assignment 2

Arseni Loika

May 2024

1 Gaussian Filtering

To verify the similarity between the application of Gaussian filter in spatial and frequency domain, first a simple padded image was created according to specifications. For spatial domain, the in-built MATLAB function `imgaussfilt()` was used, with the parameter $\sigma_s = 10$. For frequency domain, the Gaussian filter was written from scratch using Professor Didyk's lab code as a base. The value of parameter was $\frac{1}{2\sigma_s\pi} = \frac{1}{2\cdot 10\pi} \approx 0.016$. To handle Fourier and inverse Fourier transformations, in-built functions such as `fft2()`, `fftshift()` and `ifft2()` were used. The remaining implementation details can be seen in the code below:

```matlab
close all;
% Make image
dim = 1024;
img = ones(dim, dim);
img = img .*0.9;
center = dim/2;
R = 250;
img(center, center) = 0;
img(center:center+R, center:center+R) = 0;
img(center-R:center, center-R:center) = 0;
img(center-R:center, center:center+R) = 0;
img(center:center+R, center-R:center) = 0;

% Spatial domain
img = padarray(img, [10, 10], 0, 'both');
sigma = 10;
I_blur = imgaussfilt(img, sigma);

% Frequency domain
% compute FT
I_ft = fft2(img);

% compute gaussian (low-pass) filter
[X,Y]= meshgrid(-522:521,-522:521);
D = sqrt(X.^2 + Y.^2);
D = D/max(D(:));
sigma_freq = (1 / (2*pi*sigma));
D = exp(-D.^2/2/sigma_freq^2);
D = D/max(D(:));

I_ft_filtered= I_ft .* fftshift(D);
I_out = ifft2(I_ft_filtered);

% visualize
% ... plotting code omitted
```

Which yielded the following results:

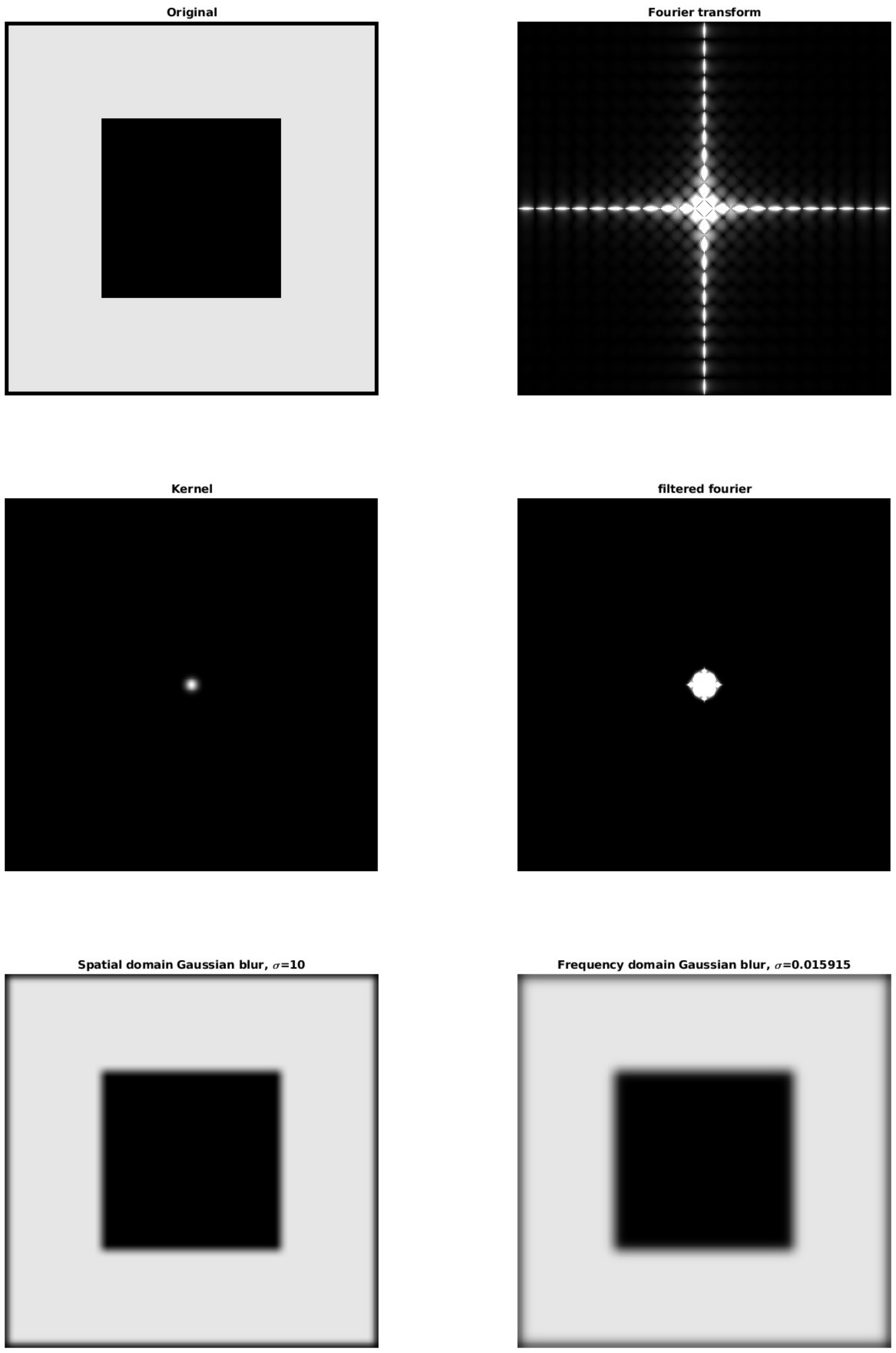
Original
Fourier transform
Kernel
filtered fourier
Spatial domain Gaussian blur, σ=10
Frequency domain Gaussian blur, σ=0.015915

Figure 1: Gaussian Filter Application and Result Stages

It can be concluded from the images that both methods produce a highly similar result, however the frequency domain method yields a noticeably stronger filter with the same σ_s.

2 Image Restoration

A simple pipeline for image restoration was built for this task. First, the (shifted) Fourier transform of the image is calculated and is shown. The artifacts that produce the checkered pattern on the image then need to be selected manually, (implemented using the in-built impixel() function), which yields the approximate coordinates of the centers of those artifacts. Then, for every selected coordinate, a circular zero-filter is created (with a pre-determined radius) and applied to the image's frequency domain. Then, the FT of the image is bitwise-multiplied by that filter and, finally, the inverse Fourier transform is calculated from the resulting FT. The exact details of the implementation can be seen in the code below.

```
close all;
img = imread("san_domenico.png");
% img = imread("moon.png");
img = im2double(img);
img_f = fft2(img);
[height, width, ~] = size(img);

% get pixels coords - SELECT ARTIFACTS MANUALLY
[x, y, ~] = impixel(abs(fftshift(img_f)/500));

% make the filter
mask = ones(height, width);
% radius (approximated after trial and error)
R = 15;
% make circular filter
[h,w]=meshgrid(1:height);
for i = 1:size(x)
    mask((h-x(i)).^2+(w-y(i)).^2<R^2)=0;
end

% apply filter to the frequency domain
res_f = img_f .* fftshift(mask);

% apply inverse fft
out = abs(ifft2(res_f));

% visualize
% ... plotting code omitted
```

The steps and results can be seen below.

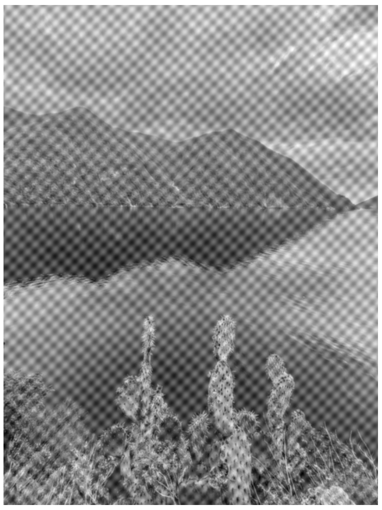

Figure 2: Original image

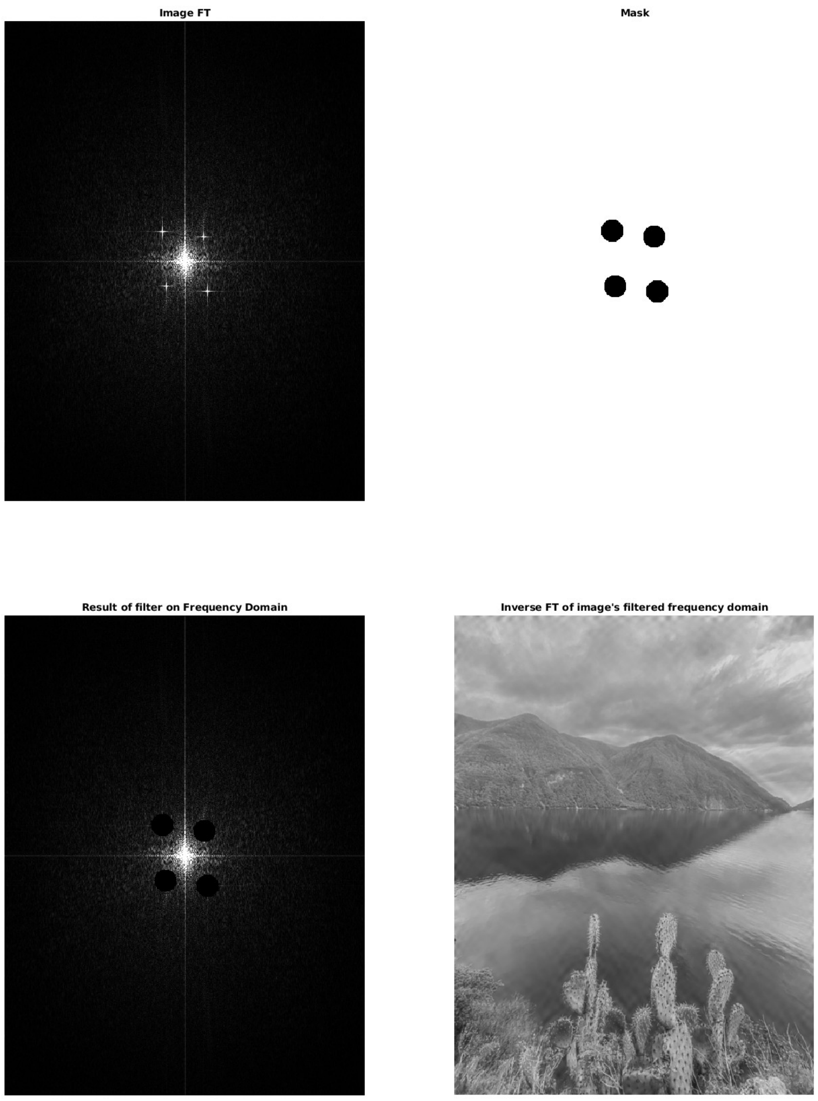
Image FT
Mask
Result of filter on Frequency Domain
Inverse FT of image's filtered frequency domain

Figure 3: Intermediate processing steps and result