CS 484, Spring 2019
Homework Assignment 1:
Binary Image Analysis & Linear Filtering

Full name: Fuad Aghazada
ID: 21503691
Used Programming language: Python3

Question 1

In order to implement *dilation* and *erosion* morphological operations, the first notation has been used. For dilation, for every pixel in the source image, the program checks whether the given structuring element has at least one intersected pixel: in this case, the value of the pixel at the center of the structuring element is put into the current location in the output image. For erosion, this condition is replaced by the having full intersection between the structuring element and current location in the source image.

Question 2

Using the morphological operations that were written for the question above (Q1), from the given input image the following result is obtained:

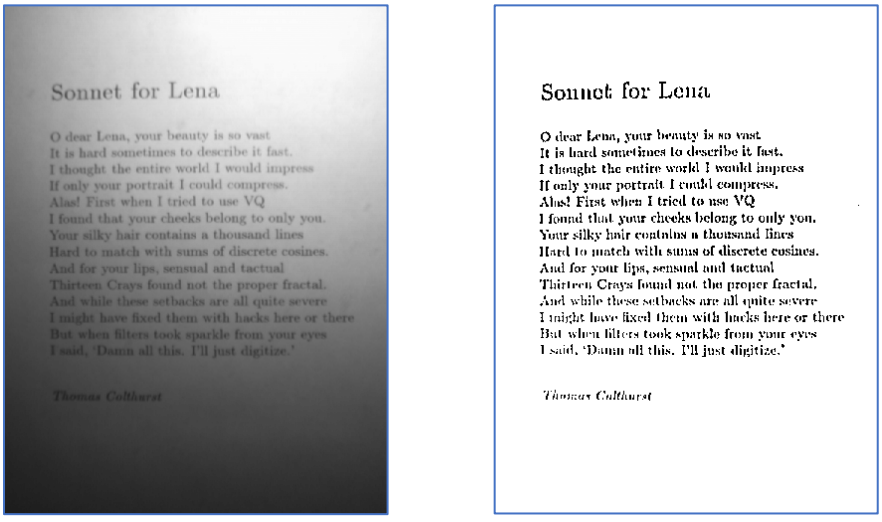

Figure 1. Left: input image, Right: output

The following operations have been done:

- Global thresholding was not useful, because of the different light effects on different areas of the image, so adaptive thresholding has been used.
- Adaptive thresholding (implemented by me) with the parameters of the locality dimension of 5 and constant (C = 5) has been applied in order to remove light/dark area. Having higher parameters, the desired output image is immediately obtained, so using these parameters created a place for applying morphological operations.
- After having the binary image, erosion with 1x3 (white) structuring element has been applied to connect broken white spaces among the text.
- After the erosion, since the space between the letters and lines got small, dilation with a 1x1 (white) structuring element was necessary to be applied.

Source code:

```
src_img = load_img_2D('sonnet.png')

thr_img = adaptive_threshold(src_img, (5, 5), 5, 0)

strc_el = np.ones((1, 3)).astype(np.uint8) * 255
out_img = erosion(thr_img, strc_el)

strc_el = np.ones((1, 1)).astype(np.uint8) * 255
out_img = dilation(thr_img, strc_el)

save_img_from_array(out_img, out_name = 'sonnet_out.png')
```

Question 3

In order to implement convolution on spatial domain the following function prototype has been used:

```
convole(source_image, filter, mode = 0)
```

Here *source_image* is the input image, *filter* is the given filter and an optional parameter *mode* is for applying the average-based (mean) or order-based (median) computation.

Iterating through the input image, for each “available” (ignoring border parts as black) pixel, the convolved pixel has been calculated either as a mean of the pixels in the neighbor of the filter dimensions, or as a median of them after getting multiplied with the filter values.

In order to test the function, Sobel operator (horizontal) has been used as a filter. The following figure shows the result:

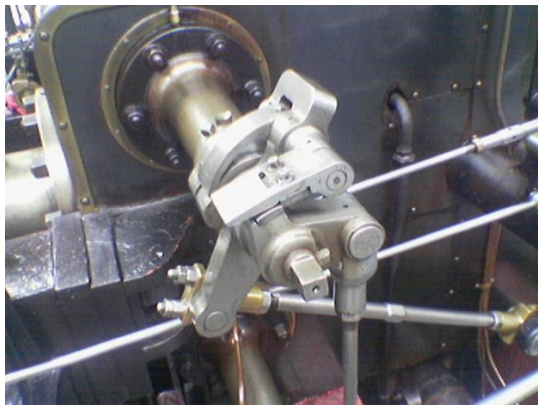


Figure 2. Left: input image, Right: output

Question 4

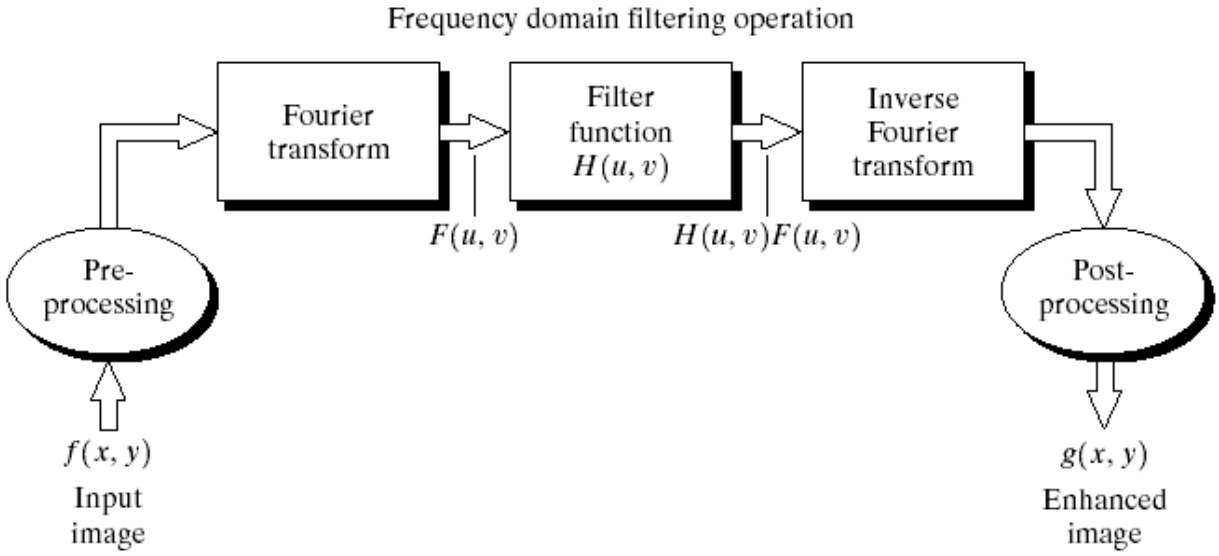

Figure 3. Applied steps for gaussian filtering in frequency domain

Step1:

Given the source image (Figure 4a), Gaussian kernel (Figure 4b) with the image size has been calculated for every pixel (sigma = 9) and at the end, to normalize the function, it has been divided by its sum.

Step2:

Fourier transform of source image and gaussian kernel has been calculated (Figure 4c,d).

Step3:

The calculated Fourier transforms have been multiplied element-wise (Figure 4e).

Step4:

For the multiplied result, the inverse Fourier transform has been calculated as an output image (Figure 4f).

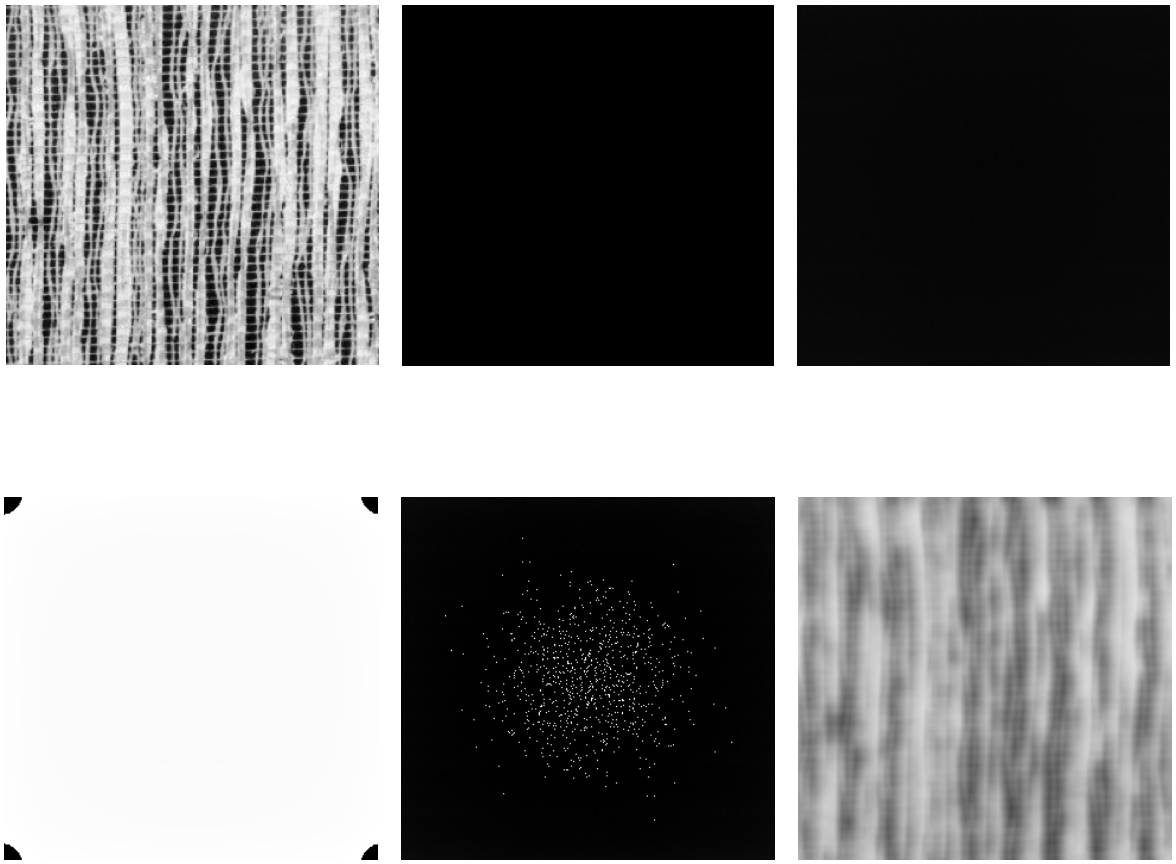

Figure 4. a, b, c
d, e, f

Source code:

```
def gaussian_filter(source_image, sigma = 0.2):

	(M, N) = source_image.shape

	# Calculating gaussian kernel
	g_func = np.ones(source_image.shape)
	for i in range(0, M):
		for j in range(0, N):
			g_func[i, j] = math.exp(-(i * i + j * j) / (2 * sigma * sigma)) / (sigma
			* sigma * math.pi * 2.0)
```

```
        # Normalizing
        g_func /= np.sum(g_func)

        # Fourier transform of source_image & filter
        fft_src_img = np.fft.fft2(source_image)
        fft_filter_func = np.fft.fft2(g_func)


        fft_out = np.multiply(fft_src_img, fft_filter_func)
        out = np.fft.ifft2(fft_out)
        out = out.real

        return np.log(fft_src_img).real, g_func, np.log(fft_filter_func).real, out,
np.log(fft_out).real
```