Examples of Filtering on the Frequency Domain

Wang Zhuoyang[1]

[1] *12112907, Department of Electrical and Electronic Engineering, SUSTech. Email: glverfer@outlook.com*

Abstract

Too lazy to write an abstract.

Keywords: Frequency Domain Processing; Filtering; Digital Image Processing

Contents

1 Introduction

Digital Image Processing Laboratory Reports (2023)

Corresponding author
Wang Zhuoyang

This experiment is primarily focused on processing images in the frequency domain.

The first task involves applying the Sobel operator to the image in both the spatial and frequency domains and studying the effects of certain details in the process. The second and third tasks

require the use of frequency domain filtering to apply ideal low-pass and Gaussian filters to the image, respectively. The fourth task involves using a Notch Filter to eliminate specific peaks in the frequency domain, in order to remove features such as stripes in the original image.

This report will only analyze and report on the first and fourth tasks. The code for the second and third tasks is included in the attachment, but will not be analyzed or presented, with the exception of answering some of the questions in the experimental requirements.

2 Sobel Filtering on Spatial and Frequency Domain

2.1 Algorithm Principle and Pseudo-code

Principle

The method used for processing in the spatial domain is to apply convolution. The conventional approach and vectorization implementation of convolution have been elaborated in detail in the previous report and will not be reiterated here.

Our focus this time is on the implementation in the frequency domain. The basic idea of the above operation is to transform the original image $f(x, y)$ and the convolution kernel $h(x, y)$ into $F(\mu, \nu)$ and $H(\mu, \nu)$ in the frequency domain through fast Fourier transform. Since the product in the frequency domain is related to the convolution in the spatial domain, multiplying them to obtain $G(\mu, \nu)$ in the frequency domain, and then using fast Fourier inverse transform, we can obtain $g(x, y)$ in the spatial domain.

Add some detail processing to the algorithm, the process is as follows:

1. Perform zero-padding on the original image $f(x, y)$ and the convolution kernel h represented in the spatial domain.
2. Multiply the original image $f(x, y)$ by the phase factor, then perform Fourier transform to obtain $F(\mu, \nu)$.
3. Multiply the convolution kernel $h(x, y)$ by the phase factor, then perform Fourier transform to obtain $H(\mu, \nu)$, and multiply by the phase factor again.
4. Multiply the two to obtain $G(\mu, \nu)$.
5. Perform inverse Fourier transform, multiply by the phase factor to restore and obtain $g(x, y)$.

Where:

1. The edge padding is because the frequency domain multiplication actually corresponds to the periodic convolution in the spatial domain, which is different from the convolution performed in the spatial domain. Therefore, it is necessary to pad the two images to the same size, and the size should not be less than the sum of their side lengths minus one.
2. The phase factor is $(-1)^{x+y}$, which aims to move the frequency spectrum center to the image center for easy observation and arithmetic operations on the frequency spectrum.
3. The last step of obtaining $H(\mu, \nu)$ also requires multiplying by the phase factor to align the center of the convolution kernel, as explained in the last section of Questions and Answers, the first item.

Pseudo-code

The pseudo-code for the frequency domain method (Algorithm 1) is provided below, and the specific implementation can be referred to in the source code.

Algorithm 1: Convolution by frequency domain processing.

Input: f, h
Output: g

$target\_dim \leftarrow (f.height + h.height - 1, f.width + h.width - 1)$
$f\_p \leftarrow$ Fill f with zeros to the right and bottom directions to $target\_dim$
$h\_p \leftarrow$ Fill f with zeros to $target\_dim$ with h staying at center
$shifter \leftarrow (-1)^{x+y}$
$F \leftarrow$ FFT2($f\_p \times shifter$)
$H \leftarrow i \times$ imag(FFT2($h\_p \times shifter$)) $\times shifter$
$G \leftarrow F \times H$
$g \leftarrow$ real(IFFT2(G)) $\times shifter$

2.2 Python Implementation and Complexity Analysis

Implementation

The implementation of the main program can be found in Code ???1. The 'regulator' is a pre-written class that processes the results, and the role of 'GrayScalingRegulator' is to scale and round the result to within the uint8 range.

```
kernel_sobel = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
# Spatial domain
img_1_spatial = linear.convolution(img_1, kernel_sobel, regulator=regulator.
    GrayScalingRegulator)
# Frequency domain
img_1_freq = frequency.convolution_freq(img_1, kernel_sobel, regulator=
    regulator.GrayScalingRegulator)
```

Listing 1. Sobel filtering on spatial and frequency domain.

In the spatial domain, convolution is used for implementation, and the specific implementation of 'linear.convolution' is shown in Code 2.

```
def convolution(img, kernel, pad_mode='constant', regulator=NonRegulator):
    size = kernel.shape[0]
    rad = size // 2
    img = np.pad(img.astype(np.int32), ((rad, rad), (rad, rad)), mode=
    pad_mode)
    kernel_spanned = np.expand_dims(np.reshape(kernel, -1), axis=(1, 2))
    pts = [(idx // size - rad, idx % size - rad) for idx in range(size *
    size)]
    moved = np.array([np.roll(np.roll(img, x, axis=0), y, axis=1) for (x, y)
     in pts])
    res = np.sum(moved * kernel_spanned, axis=0)
    return regulator(res[rad:-rad, rad:-rad])
```

Listing 2. The implementation of convolution.

In the frequency domain, the steps mentioned earlier are used for implementation, and the specific implementation is shown in Code 3.

```
def shifting_mat(dim):
    return (-1) ** sum(np.meshgrid(range(dim[1]), range(dim[0])))

def pad_fh_in_pair(f, h, mode='min'):
    # mode='constant', constant_values=0.
    f_r, f_c = f.shape
    h_r, h_c = h.shape
    f_p, h_p = f, h
```

```
    if mode == 'min':
        fac_r, fac_c = f_r % 2, f_c % 2
        f_p = np.pad(f_p, ((0, h_r - 1 + fac_r), (0, h_c - 1 + fac_c)))
        h_p = np.pad(h_p, ((f_r // 2 + fac_r, f_r // 2 - 1 + fac_r), (f_c //
 2 + fac_c, f_c // 2 - 1 + fac_c)))
    elif mode == 'double':
        f_p = np.pad(f_p, ((0, f_r), (0, f_c)))
        h_p = np.pad(h_p, ((f_r - h_r // 2, f_r - 1 - h_r // 2), (f_c - h_c
// 2, f_c - 1 - h_c // 2)))
    return (f_p, h_p)

def compute_spectrum(f): # Shifted
    shifter = shifting_mat(f.shape)
    return np.fft.fft2(f * shifter)

def compute_FH_in_pair(f, h, pad_mode='min'): # Shifted
    f_p, h_p = pad_fh_in_pair(f, h, pad_mode)
    shifter = shifting_mat(f_p.shape)
    return (compute_spectrum(f_p), 1j * np.imag(compute_spectrum(h_p)) *
shifter)

def recover_spatial(G, deshift=True, regulator=NonRegulator):
    shifter = shifting_mat(G.shape)
    return regulator(np.real(np.fft.ifft2(G)) * shifter)

def convolution_freq(f, h, regulator=NonRegulator):
    F, H = compute_FH_in_pair(f, h)
    g_p = recover_spatial(F * H)
    return regulator(g_p[0 : f.shape[0], 0 : f.shape[1]])
```

Listing 3. The implementation of convolution in frequency domain.

The time complexity of convolution algorithm in naive situation is $O(MNK^2)$, where M and N are the height and width of the image respectively, and K is the size of the kernel. The space complexity is the sum of the space occupied by the image and the kernel.

If vectorization approach (as shown above) is used, the theoretical time complexity remains the same, but the constant is greatly reduced. The space complexity is the product of the number of elements in the kernel and the size of the image.

If a frequency-domain algorithm is used, the time complexity is $O(MNlog_2(MN))$ (which is the time complexity of two-dimensional fast Fourier transform), and the space complexity is the sum of the space occupied by the image and the kernel (including after converting to the frequency domain).

Alternative Method

Multiplying by the shift factor in the above process is to move the center point of the function in the dual space. This is mainly done for two engineering purposes:

1. Convenient for observing.
2. Facilitating zero-padding of convolution kernels (otherwise, zeros would need to be filled in the center of the image).

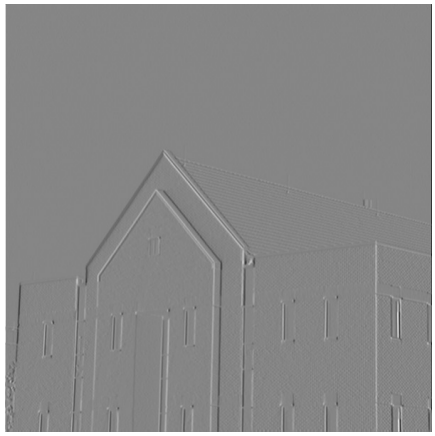
However, in reality, we can avoid multiplying by the shift factor and solve the zero-padding problem using the roll and pad functions. The benefit of doing so is that it eliminates the need to consider the shift factor and the center point of the spectrum, reduces the number of computations, and improves efficiency.

Specific code is not provided as there is not much improvement in efficiency.

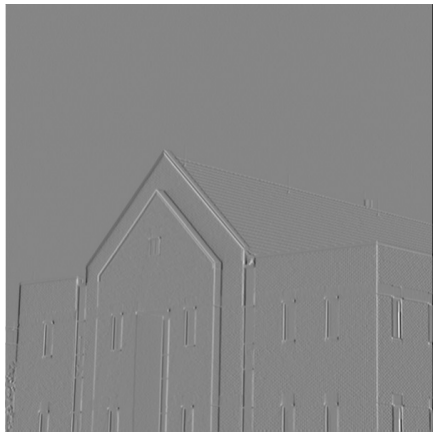
As for complexity: the time complexity is the same as before but with a smaller constant; the space complexity is similar, saving storage space for the shift factor.

2.3 Results

Here are the results of the two methods (Figure 1), after being processed by gray scaling. It can be observed that the results of the two methods are exactly the same.



(a) Spatial method.



(b) Frequency method.

Figure 1. The results for the Task 1.

Remarks

That is the correct answer.

One thing to note during the processing is the difference between convolution and correlation operations. The two operations are equivalent to applying the kernel with a 180-degree rotation. If the Sobel operator is applied to the image using correlation operation in the spatial domain, the result would be as shown in Figure 2.

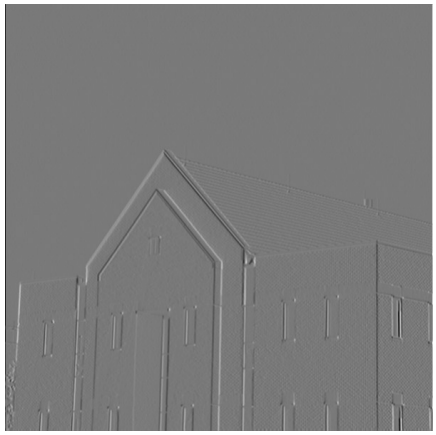


Figure 2. Using the correlation.

Another thing to note is the dimension of the pictures and the spectrum. If the edges are not processed properly and there is a slight difference in black border or white border, it will lead to visual differences after grayscale scaling (even if there is no essential difference).

3 Applying Notch Filtering to Eliminate Stripes

3.1 Algorithm Principle and Pseudo-code

Principle

Notch filters are generated to filter the frequency spectrum selectively. In this task we choose Butterworth Notch Filter to eliminate the stripes in the raw image. The frequency spectrum of the original image of this problem is shown in Figure 3.

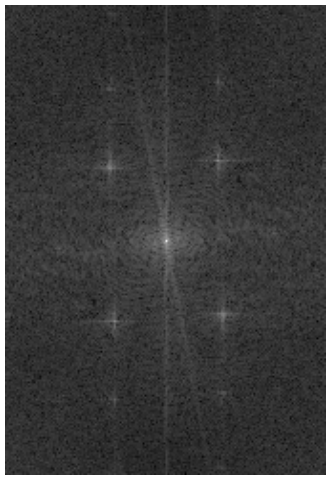


Figure 3. The spectrum of the raw image.

From the spectrum of the original image, we can see that there are four pairs (eight in total) of isolated frequency peaks, which result in diagonal stripes in the image. In order to eliminate these stripes, we can design notch filters to selectively remove these frequency peaks.

The above process is specific to the input image and requires manual calibration of the frequency peak positions. The frequency peaks in this image are clearly artificially set. They are visually divided into three equal parts horizontally and six equal parts vertically, with positions (relative to the center of the image) at $(-28, -82), (-28, -41), (-28, 41), and (-28, 82)$.

We can generate a Butterworth Notch Filter for each coordinate by setting a notch at the coordinate and its symmetrical point about the center. After obtaining four filters, we multiply them together and then multiply the result with the frequency spectrum of the original image to obtain the filtered image.

Pseudo-code

The pseudo-code for the frequency domain method (Algorithm 2) is provided below, and the specific implementation can be referred to in the source code.

Algorithm 2: Frequency filtering using a specific H.

Input: f, H
Output: g

$target\_dim \leftarrow (f.height * 2, f.width * 2)$
$f\_p \leftarrow$ Fill f with zeros to the right and bottom directions to $target\_dim$
$shifter \leftarrow (-1)^{x+y}$
$F \leftarrow$ FFT2($f\_p \times shifter$)
$G \leftarrow F \times H$
$g \leftarrow$ real(IFFT2(G)) $\times shifter$

As we have learned above, it is required to generate a spectrum with four pairs of notch filters that have stopbands. The formula for one pair is provided in Equation 1, and pseudocode will not be presented here.

$$H_{NR}(\mu,\nu)=\prod_{k=1}^{3}\left[\frac{1}{1+\left[\frac{D_{0k}}{D_k(\mu,\nu)}\right]^{2n}}\right]\left[\frac{1}{1+\left[\frac{D_{0k}}{D_{-k}(\mu,\nu)}\right]^{2n}}\right] \tag{1}$$

Where,

$$\begin{cases} D_k(\mu,\nu)=(\mu-\frac{M}{2}-\mu_k)^2+(\nu-\frac{N}{2}-\nu_k)^2 \\ D_{-k}(\mu,\nu)=(\mu-\frac{M}{2}+\mu_k)^2+(\nu-\frac{N}{2}+\nu_k)^2 \end{cases} \tag{2}$$

In which, M and N is the height and the width of the spectrum, (μ_k,ν_k) are the stop bands (points) of the notch filter.

3.2 Python Implementation and Complexity Analysis

Implementation

The implementation of the main program can be found in Code 4.

```
img_3_F_disp = frequency.magnitude_spectrum_disp(frequency.compute_spectrum(
    img_3))
order = 4
D0 = 6
x0 = -28
H_p = np.ones((img_3.shape[0] * 2, img_3.shape[1] * 2))
for y0 in [-82, -41, 41, 82]:
    H_p_0 = frequency.H_butterworth_NRF(img_3.shape, (x0, y0), order, D0,
    double_pad=True)
    H_p = H_p * H_p_0
img_3_nrf = frequency.filter_freq_pad(img_3, H_p, is_H_padded=True,
    regulator=regulator.GrayCuttingRegulator)
img_3_nrf_F_disp = frequency.magnitude_spectrum_disp(frequency.
    compute_spectrum(img_3_nrf))
```

Listing 4. Butterworth notch filtering

The function ‘magnitude_spectrum_disp’ is used to take the amplitude value and gamma enhance the spectrum image for display purposes. The function ‘compute_spectrum’ is used to calculate the spectrum, and ‘filter_freq_pad’ is for frequency filtering. ‘H_butterworth_NRF’ is the function used to calculate the Butterworth notch filter. Some of the code implementation is shown below (Code 3.2).

```
def filter_freq(f, H, regulator=NonRegulator): # No padding.
    f_r, f_c = f.shape
    F = compute_spectrum(f)
    G = F * H
    g = recover_spatial(G)
    return regulator(g)

def filter_freq_pad(f, H, is_H_padded=False, regulator=NonRegulator): #
    Double padded.
    f_r, f_c = f.shape
    f_p = np.pad(f, ((0, f_r), (0, f_c)))
    H_p = H
    if not is_H_padded:
        H_p = np.pad(H_p, (((f_r + 1) // 2, f_r // 2), ((f_c + 1) // 2, f_c
    // 2)))
```

```
    return regulator(filter_freq(f_p, H_p, regulator=regulator)[:f_r, :f_c])

def H_butterworth_NRF(dim, position_wrt_center, order, d_cutoff, double_pad=
    False): # Notch reject filter.
    if double_pad:
        dim = (dim[0] * 2, dim[1] * 2)
        position_wrt_center = [position_wrt_center[0] * 2,
    position_wrt_center[1] * 2]
        d_cutoff = d_cutoff * 2 # TODO: ?
    o_x, o_y = dim[1] // 2 + 1, dim[0] // 2 + 1
    x_mesh, y_mesh = np.meshgrid(range(dim[1]), range(dim[0]))
    D2 = (x_mesh - o_x + 0.5 - position_wrt_center[0]) ** 2 + (y_mesh - o_y
    + 0.5 - position_wrt_center[1]) ** 2
    res = 1 / (1 + (d_cutoff ** 2 / D2) ** order)
    res = res * np.fliplr(np.flipud(res))
    return res
```

The time complexity of the algorithm is $O(MNK + MNlog_2(MN))$, where the former term is the time complexity of filter generation, and K is the number of point pairs. The latter term is the time complexity of FFT.

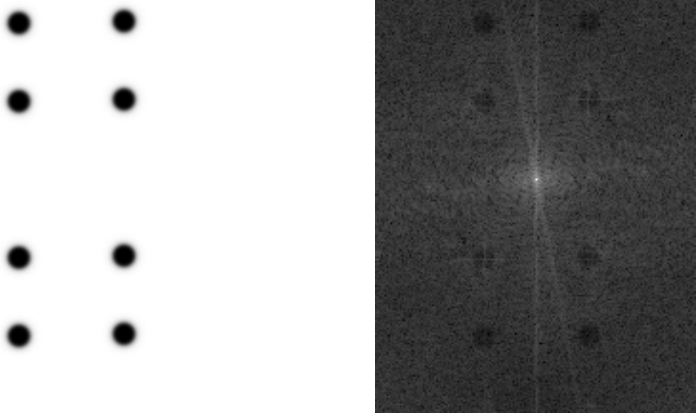
The space complexity of the algorithm is similar, occupying two times the size of the spectrum during the spectral multiplication process, plus the space of the image and its spectrum.

Alternative Method

You can improve efficiency by using the same approach as for Task 1. In addition, it is possible to calculate H without using a loop and cumulative multiplication. Although this would sacrifice generality, it can improve efficiency and reduce space waste. However, it may not make much difference in practical terms.

3.3 Results

First, the filter obtained by calculating based on four pairs of points is shown in the following Figure 4 (a). Figure 4 (b) displays the result of multiplying the original image spectrum with the filter spectrum (enhanced by gamma correction, hence brighter).



(a) Spectrum for H.

(b) Spectrum for the filtered image.

Figure 4. The spectrum for the Notch Filter and the result.

And the result in spatial domain is in Figure 5.

Figure 5. The image with the stripes eliminated.

Remarks

The result is as expected, but there are two major flaws in the method:

1. It is too dependent on specific images, and has poor universality. Moreover, in most cases, the stripes are not so regular.
2. The elimination is incomplete, and new stripes appear on the upper part of the wheel where there were no stripes originally. This is an unavoidable problem caused by the frequency domain filtering approach cutting off a certain frequency of noise too abruptly.

4 Questions and Answers

4.1 The Shifting Operator Multiplied on H

"Explain why perform a shift in Step 4 on slide 81 of Lecture 4 in the first Exercise."

As mentioned earlier, two shifting factors were multiplied when converting the spatial convolution kernel to a frequency domain function.

The first multiplication was performed before the FFT to move the center of the frequency spectrum to the center of the image, which facilitates subsequent algebraic computations (and controlling the symmetry of zero-padding).

The second multiplication was performed after the FFT and taking the imaginary part, with the aim of moving the spatial convolution kernel from the center of the image to the corners. Specifically, we fill the space around the spatial convolution kernel with zeros to move it to the center of the image, but convolution is performed with reference to the origin. In other words, if we directly multiply without shifting, the center of the resulting image will be the result of convolution in the upper-left corner of the image, and so on.

For example, if we use the image and 3x3 Sobel operator from Task 1, with zero-padding to P+Q-1 (i.e., minimum zero-padding) and without multiplying the unique factor, we get the result shown in Figure 7.

In particular, if we use the method of padding to twice the size of the original image, we can also obtain the correct result by taking the bottom-right corner of the resulting image (Figure 9).

4.2 The Ring Effect

"Explain what cause the ring effect in the ideal filtering. Design an experiment to verify your reasoning."

Ring effect will appear when we use ideal filters, like the situation in Figure 8.

Figure 6. The result without multiplying H with the shifting factor. (minimizing padding strategy)

Figure 7. The result without multiplying H with the shifting factor. (doubling padding strategy)

The reason for this phenomenon can be understood from the perspective of spatial domain. We can design a experiment that display the spatial image of H, in order to confirm our conclusion.

The frequency spectrum of an offline low-pass filter is a disk with an infinitely steep edge, as shown in Figure **??** (a). Figure **??** (b) shows the result of transforming it into the spatial domain (after gray scaling). It can be seen that after the transformation, the weight distribution itself has a ring effect. It is conceivable that if it is used for convolution, periodic ripples will also appear in the image at a certain distance from each edge.

Figure 8. Applying the ideal low-pass filter with a radius of 60.

(a) Spectrum for H. (padded)

(b) Spatial illustration of H.

Figure 9. The ideal low-pass filter.

4.3 The Parameters for Notch Filters

"In the above implementation 4, how the parameters in the notch filters are selected, and why."

The Butterworth Notch Filter mainly consists of three parameters: the location of the filter points, the filter order, and the cutoff frequency (radius) of the filter.

In this example, the first parameter selects four (pairs of) points, the specific reason for which is explained in the previous analysis.

The second parameter selects a 4th order, which is neither too high nor too low. A higher order can be used when it is desired to have less influence on the surrounding when not filtering, while a lower order is more natural but has a relatively weaker filtering effect.

The third parameter is selected as 6, which can be obtained by observing the frequency spectrum. It can be combined with the second parameter, for example, using a lower order in a larger range.

4.4 The Characteristic of H

"Explain why $H(\mu, \nu)$ has to be real and symmetric in the Step 5 on slide 71 of Lecture 4, which is also the case for all the filter used in this laboratory"

The explanations for the two characteristics respectively:

1. The symmetry in frequency domain ensures that the convolution kernel is a real number in the spatial domain, which guarantees the resulting image after operation is a valid digital image.

2. The symmetry in spatial domain, i.e., a symmetric convolution kernel, ensures anisotropy. Therefore, this step is not necessary.

5 Conclusion

In summary, this experiment is a simple exploration of various processing methods in the frequency domain. It includes frequency domain multiplication to achieve spatial domain convolution, as well as various filters. The experimental results all met expectations, and were evaluated and summarized in the report.

Compared to the original code, the code for this experiment was somewhat cumbersome. The main reason for this was due to issues surrounding image scale and zero-padding. Additionally, the experiment required trying out some special methods, which resulted in the program needing to consider a large number of special cases.

One of the more general takeaways from this experiment was the correspondence between some characteristics of the spatial and frequency domains, such as symmetry, rotation, and translation. This could be helpful in developing some feature extraction algorithms.

Aside from this, the experiment overlooked the influence of phase information on images, so further experiments could be conducted to explore this area. This will not be discussed further in this report.