SUSTech_EE326_lab5

Topic: Image Filtering in Frequency Domain

Author: 11911521钟新宇

Project: lab5 report for Digital Image Processing

1. Introduction

In the theory class, we learned about filtering images in the frequency domain. In this lab, we will use solel operator, ideal low-pass filter, Gaussian filter, and Butterworth notch filter to process images and summarize the laws of frequency domain filtering.

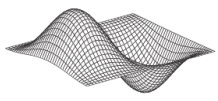
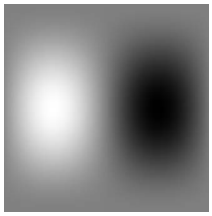
2. Sobel filter

In the previous experiment, we understand that the sobel operator is a type of gradient operator in the spatial domain, similar to the robert operator. sobel operator weights the difference between the intensities of three pixel points in the x and y directions, which means that the sobel operator can smooth the intensities in the same direction and make the rate of change in the gradient direction more significant. The sobel operator sharpens the image and makes the edges of the image sharper.

In this example, we start with a spatial mask and show how to generate the corresponding filter in the frequency domain. Then, we compare the filtering results obtained using frequency domain and spatial techniques.

2.1 Solution

The following figure shows the spatial mask of a sobel operator and its 3D image in the frequency domain.

	Image		Image		Image
The spatial mask		Perspective view in frequency domain		Filter shown as an image	

We use python to program, and only the important parts of the code are shown in the report.

The spatial domain filtering of the sobel operator has been demonstrated in the previous lab, and the procedure is as follows.

1. Firstly we zero-fill the edges of the original image.
2. Then we multiply each neighborhood with the sobel mask and sum up.
3. Finally we update the intensity of the image center point with the summation result.

```
for i in range(row):
    for j in range(col):
        temp = np.sum(img_pad[i:i + 3, j:j + 3] * kernel)
        img_out[i, j] = temp
```

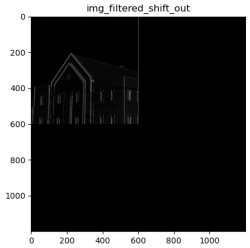
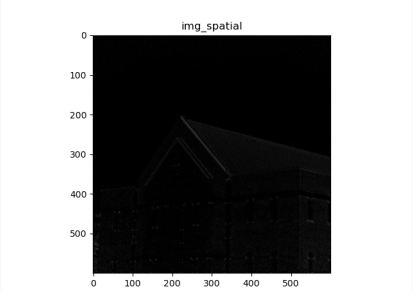
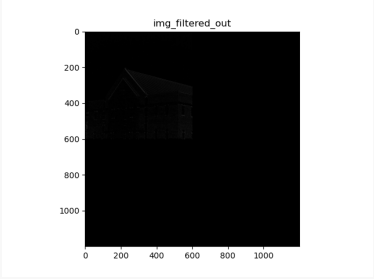
The process of frequency domain filtering is similar to spatial domain filtering, the difference is that we first transform the image and the mask into the frequency domain and multiply them together, and then use the inverse transform to obtain the filtered image.

I use the `fft2`, `ifft2` and `fftshift` functions in the `numpy` package for the 2D fast Fourier transform.

```
kernel_fft = np.fft.fft2(kernel_pad)
img_fft = np.fft.fft2(img_pad)
img_filtered = np.multiply(img_fft, kernel_fft)
img_filtered = np.fft.fftshift(img_filtered)
img_filtered = np.fft.ifft2(img_filtered)
```

2.2 Result & Analysis

The results are as follows.

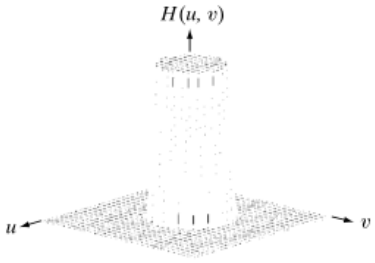
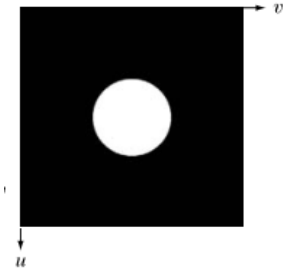
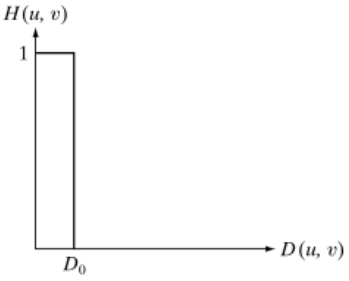
	Image		Image
The original image		Filtered with shift	
Filtered in the spatial domain		Filtered without shift	

We can observe that the spatial domain filtering results are approximately the same as the frequency domain filtering results, but there are some differences.

1. The frequency domain filtered image seems to be larger than the original image. This is because we zero-fit the image, which is to ensure that no aliasing occurs during the sampling process.
2. The outline of the image after fftshift is clearer. This is because the 2D Fourier transform will put the DC component, which has the highest energy of the image, at the origin, but the origin of the image is the upper left corner, so the DC component will be scattered to the four corners of the image, so it looks unintuitive. When the image is fftshifted, the origin of the image is shifted to the center of the image, and the DC component in the frequency domain is also in the center, so the outline of the image in the spatial domain will be clearer.

3. Ideal filter

In the frequency domain, the frequency response of a one-dimensional ideal low-pass filter is a rectangular window. Components below the cutoff frequency can be passed, and components above the cutoff frequency will be filtered. In the two-dimensional image, the three-dimensional perspective view of the ideal low-pass filter is a cylinder. This is shown in the figure below.

Perspective plot	Filter displayed as an image	Filter radial cross section
		

3.1 Solution

The frequency response of a two-dimensional ideal low-pass filter can be expressed in the following form.

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \leq D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases} \tag{1}$$

where D_0 is a positive constant and $D(u,v)$ is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle; that is,

$$D(u,v) = \left[(u - P/2)^2 + (v - Q/2)^2\right]^{1/2} \tag{2}$$

That is, we can distinguish high-frequency components from low-frequency components based on the distance of each frequency component from the origin in the frequency domain.

Therefore we can get the code for ideal low-pass filter in the frequency domain.

```python
def ideal_mask(a, b, d0):
    x = np.array(np.linspace(0, a - 1, a) - a / 2)
    y = np.array(np.linspace(0, b - 1, b) - b / 2)
    mask = np.zeros((a, b))
    for i in range(a):
        for j in range(b):
            d = np.sqrt(x[i] ** 2 + y[j] ** 2)
            if d <= d0:
                mask[i, j] = 1
    return mask
```

When processing the images, we follow the following steps.

1. Calculate the frequency domain expression of the image
2. Multiply the result with the ideal low-pass filter
3. Calculate the inverse Fourier transform of the multiplier
4. Adjust the intensity of the output image so that it is within 0 to 255

That is,

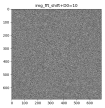
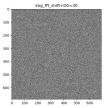
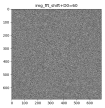
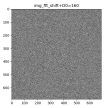
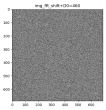
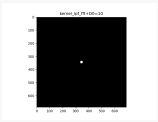
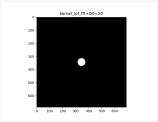
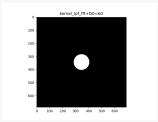
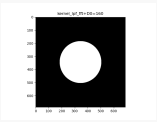
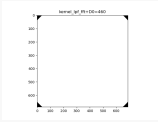
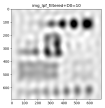
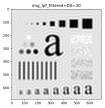
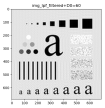
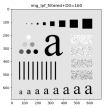
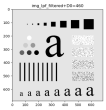
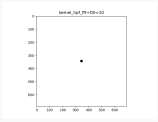
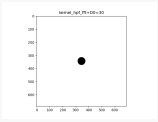
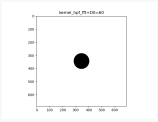
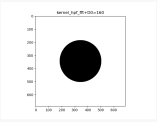
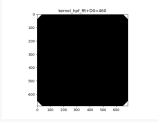
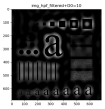
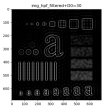
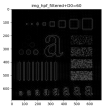
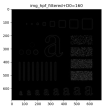
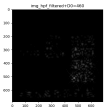
```python
img_fft = np.fft.fft2(input_image)
img_fft_shift = np.fft.fftshift(img_fft)

kernel_lpf_fft = ideal_mask(row, col, d0)

img_lpf_filtered = np.multiply(img_fft_shift, kernel_lpf_fft)
img_lpf_filtered = np.fft.fftshift(img_lpf_filtered)
img_lpf_filtered = np.fft.ifft2(img_lpf_filtered)
img_lpf_filtered = np.real(img_lpf_filtered)
img_lpf_filtered = range_normalize(img_lpf_filtered)
```

Once we have obtained the low-pass filter, the process of obtaining the high-pass filter becomes very simple. Simply create a matrix with all ones and let it subtract the mask of the low-pass filter, the result is the frequency domain mask of the high-pass filter.

```python
kernel_hpf_fft = np.ones((row, col)) - kernel_lpf_fft
```

The process of high-pass filtering is similar to low-pass filtering.

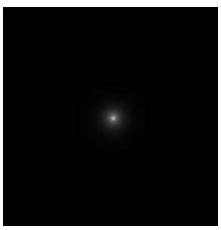
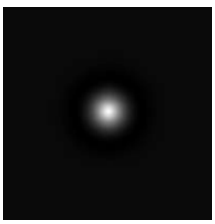
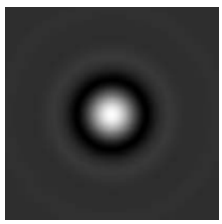
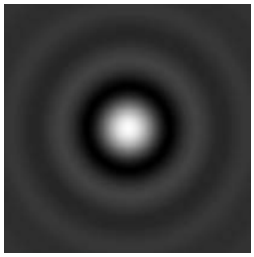
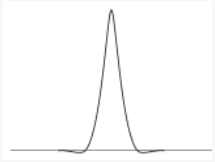
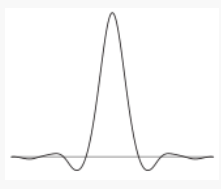
3.2 Result and Analysis

	D0=10	D0=30	D0=60	D0=160	D0=460
FFT of image					
Low-pass filter					
Low-pass filtered image					
High-pass filter					
High-pass filtered image					

We came to two conclusions.

1. When the cutoff frequency is small, the low-pass filter preserves the low-frequency components, so it will blur the image, while the high-pass filter will preserve the contours of the image and will sharpen the image.
2. As the cutoff frequency increases, the low-pass filter result will be drawn clearer and clearer, and the high-pass filter result will become more and more blurred and eventually disappear. This is because as the cutoff frequency increases, the information of the image is more likely to pass through the low-pass filter.

In addition, we also learned in the theory class that the spatial domain image of the ideal low-pass filter in the frequency domain will show a ringing effect, as shown in the figure below.

2D				
1D				

This is because the frequency response of a one-dimensional ideal low-pass filter is a rectangular window, which is sinc function in the time domain, and the sinc function will constantly cross zero. The frequency response of the two-dimensional ideal low-pass filter is a cylinder, which is also a sinc function in each direction in the spatial domain, and then the combination is one concentric circle after another. Since the grayscale range of the image is 0 to 255, the negative region is considered as 0, which is expressed as one black concentric circle in the image.

4. Gaussian filter

4.1 Solution

According to the frequency response of Gaussian filter, we can implement Gaussian filter in python.

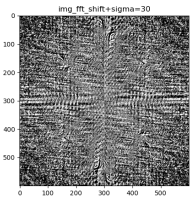
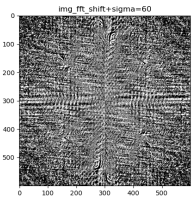
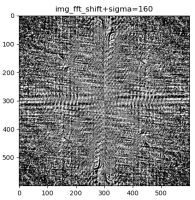
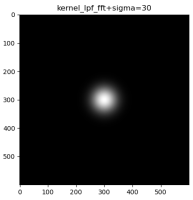
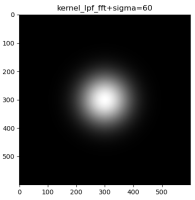
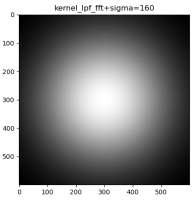
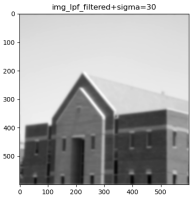
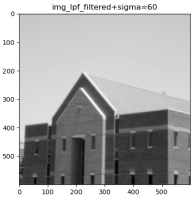
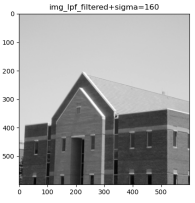
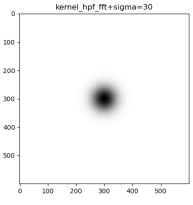
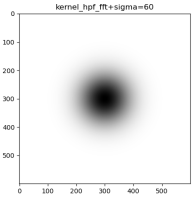
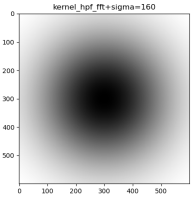
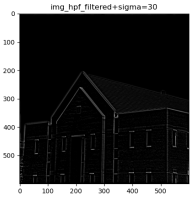
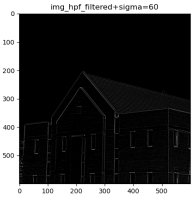
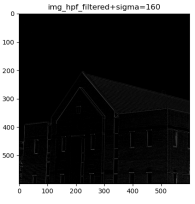
$$H(u,v) = e^{-D^2(u,v)/2\sigma^2} \tag{3}$$

```
def gussian_mask(a, b, sigma):
    x, y = np.meshgrid(np.linspace(0, a - 1, a), np.linspace(0, b - 1, b))
    x = x - a / 2
    y = y - b / 2
    mask = np.exp(-((x ** 2 + y ** 2) / (2.0 * sigma ** 2)))
    return mask
```

The processing of the image is similar to that of an ideal low-pass filter.

```
img_fft = np.fft.fft2(input_image)
img_fft_shift = np.fft.fftshift(img_fft)

kernel_lpf_fft = gussian_mask(row, col, sigma)

img_lpf_filtered = np.multiply(img_fft_shift, kernel_lpf_fft)
img_lpf_filtered = np.fft.fftshift(img_lpf_filtered)
img_lpf_filtered = np.fft.ifft2(img_lpf_filtered)
img_lpf_filtered = np.real(img_lpf_filtered)
img_lpf_filtered = range_normalize(img_lpf_filtered)
```

4.2 Result and Analysis

	D0=30	D0=60	D0=160
FFT of image			
Low-pass filter			
Low-pass filtered image			
High-pass filter			
High-pass filtered image			

I found that the results of the Gaussian low-pass filter and the high-pass filter were similar to the results of the ideal low-pass filter. The difference is that the Gaussian filter has a smoother variation around the cutoff frequency and has a center point that is blurred in the spatial domain. Besides, the parameter σ controls the radius of the filter. Whether it is high-pass filtering or low-pass filtering, the larger the σ, the more information in the frequency domain is retained in the original image, and the clearer the image obtained.

5. Butterworth notch filters

5.1 Solution

Selective filters act on a portion of the frequency rectangle, rather than the entire frequency rectangle. Specifically, bandstop or bandpass filters deal with specific frequency bands, and trap filters deal with small regions of the frequency rectangle.

According to the frequency response of Butterworth notch filter, we can implement Butterworth notch filter in python.

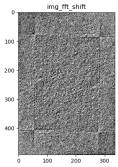
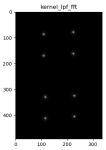
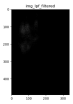
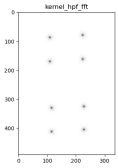
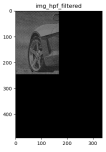
$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{4}$$

```python
def butterworth_mask(a, b, center, n, sigma):
    cx, cy = center
    x, y = np.meshgrid(np.linspace(0, a - 1, a), np.linspace(0, b - 1, b))
    x = x - cx
    y = y - cy
    d = np.sqrt(x * x + y * y)
    mask = 1 / ((1 + (d / sigma)) ** (2 * n))
    return mask
```

The processing of the image are as follows. Unlike Gaussian filters, we need to generate specific Butterworth trap filters for specific regions and then add them up.

```python
kernel_lpf_fft = np.zeros((p, q))
    for c in centers:
        kernel_lpf_fft += butterworth_mask(q, p, c, n, sigma)
```

5.2 Results

	Image		Image		Image
FFT of image		Low-pass filter		Low-pass filtered image	
High-pass filter		High-pass filtered image			

First we draw a frequency domain plot of the image, which shows that the image is affected by periodic noise that corresponds to the 8 bright spots in the frequency domain. Therefore we can use a Butterworth trap filter to eliminate these frequency components. We can see that the bright spots in the frequency domain are cancelled out and the image looks clearer.

5.3 Why Must the Filter in Frequency Domain Be Real and Symmetric

The principle of frequency domain filtering is that the frequency response of the image and the frequency response of the filter are multiplied, and then the inverse Fourier transform is performed to obtain the processed image. The frequency response of an image can be expressed as the sum of a real component and an imaginary component, so the frequency domain filtering can be expressed by the following equation.

$$g(x,y) = F^{-1}\{H(\mu,v)R(\mu,v) + jH(\mu,v)I(\mu,v)\}. \tag{5}$$

Since the phase of the image stores the image profile information, the frequency response of the filter must be purely real or purely imaginary in order not to change the phase. To simplify the calculation, we choose to use the real component of the image frequency response.

6. Summary

In this lab, I learn the principles of solel operator, ideal low-pass filter, Gaussian filter, Butterworth notch filter and some techniques of frequency domain filtering.