Laboratory 5 Frequency Domain Filtering

11810506 JIA Jiyuan

2021/03/31

Objective:

1. Compare the results of frequency domain and spatial domain filtering.
2. Implement the Gaussian low and high pass filtering.
3. Implement the Butterworth notch filter to the input images.
4. Understand and master the basic principle of image Fourier transform, pay attention to the mathematical basis of Fourier transform.
5. Master the basic steps of image frequency-domain filtering, review each step of frequency-domain filtering, as well as the reasons for the steps and the problems to be solved, and support the test in principle.

Keywords: frequency domain, spatial domain matching, Gaussian low and high pass filter, Butterworth notch filter

Introduction:

Frequency Domain Filters are used for smoothing and sharpening of image by removal of high or low frequency components. Sometimes it is possible of removal of extremely high and extremely low frequency. Frequency domain filters are different from spatial domain filters as it basically focuses on the frequency of the images. It is basically done for two basic operation i.e., Smoothing and Sharpening.

There are three types of filtering, low pass filters, high pass filters and band pass filters. Low pass filter removes the high frequency components that means it keeps low frequency components. It is used for smoothing the image. It is used to smoothen the image by attenuating high frequency components and preserving low frequency components.

High pass filter removes the low frequency components that means it keeps high frequency components. It is used for sharpening the image. It is used to sharpen the image by attenuating low frequency components and preserving high frequency components.

Band pass filter removes the lowest frequency and highest frequency components that means it keeps the moderate range band of frequencies. Band pass filtering is used to enhance edges while reducing the noise at the same time.

Fourier transform for 1D continuous situation:

$$F(u) = F(f(t)) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ut}dt$$

$$f(t) = \int_{-\infty}^{\infty} F(u)e^{j2\pi ut}du$$

Fourier transform for 1D discrete situation:

$$F(u) = \sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}, u = 0,1,2,\ldots,M-1$$

$$f(x) = \frac{1}{M}\sum_{u=0}^{M-1} F(u)e^{j2\pi ux/M}, x = 0,1,2,\ldots,M-1$$

Fourier transform for 2D continuous situation:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t,z)e^{-j2\pi(ut+vz)}dtdz$$

$$\boldsymbol{f(t,z)} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \boldsymbol{F}(u,v)\boldsymbol{e}^{\boldsymbol{j2\pi}(\boldsymbol{ut+vz})}\boldsymbol{d}udz$$

Fourier transform for 2D discrete situation:

$$\boldsymbol{F}(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} \boldsymbol{f(x,y)e}^{-\boldsymbol{j2\pi}(ux/M+vy/N)}$$

$$\boldsymbol{f(x,y)} = \frac{\boldsymbol{1}}{\boldsymbol{MN}}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} \boldsymbol{F}(u,v)\boldsymbol{e}^{\boldsymbol{j2\pi}(ux/M+vy/N)}$$

Periodicity of the DFT:

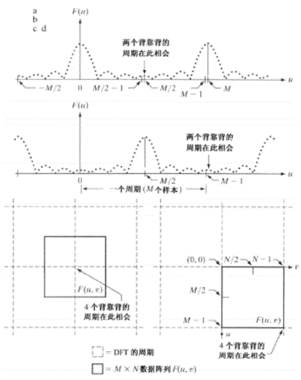
Remember from time to time that for the DFT, its space and frequency domains are always extending indefinitely along the X and Y directions. If we take just one of these periods, then we get an uncentered spectrum.

As $f(x)e^{j2\pi(u_0x/M)} \Leftrightarrow F(u-u_0)$, if we let $u_0 = M/2$, so $(-1)^x$ will times f(x) to move the $F(0)$ to the center of [0,M-1].

$$f(x,y)(-1)^{x+y} \Leftrightarrow F(u - M/2, v - N/2)$$

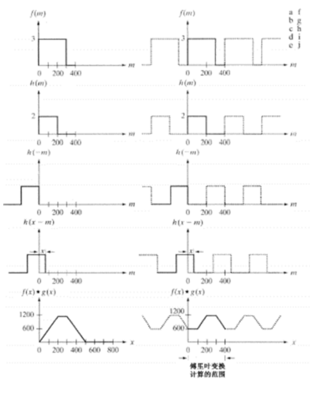
Well, mathematically, you multiply the exponent times the original function before you transforms, and because $e^{j\pi}$ is equal to 1, often when you write a program you multiply the original matrix times $(-1)^{x+y}$ to center the spectrum.

Shifting:



According to the figure it is obvious that the signal is symmetrical to the x axis, if we done the fft to the original figure, we could see that the fft figure is centered at original point, so we should move the fft figure to right bottom to get the full fft figure in one fft period. So we should done $f(x,y)(-1)^{x+y}$ in time domain to obtain a shift in frequency domain.

Zero padding:



When we do Fourier transform, we will use DFT to save time. DFT will assume the signal is periodic, when we hold the signal A's one period, flip the signal B's one period, and slip the flipped B on A will cause the calculation of B and neighbor period of A. So we should padding the zero to avoid this error. Let $P \geq A + B - 1$

Symmetric of the DFT:

Because any function can be represented as the sum of an odd function and an even function. And it is easy to prove by plugging this into the formula for the Fourier transform, that the odd function FT is odd, and the even function FT is even. If we introduce the concepts of virtual and real functions, it is also easy to prove that the FT of a real even function is a real even function, and the FT of a real odd function is an imaginary odd function. The FT of a virtual even function is a virtual even function, and the FT of a virtual odd function is a real odd function.

Real and Symmetric of $\boldsymbol{H(u,v)}$:

Besides the content mentioned above, it is obvious that filter can only change the magnitude of image's fft, no phase change can he caused, so we must let the $H(u,v)$ be real and even.

Sobel filter:

Principle:

Given a space (time) filter, the digital signal can be filtered directly in the space (time) domain or in the frequency domain. The mathematical operation of spatial filtering is correlation, and the corresponding frequency domain is dot multiplication of the conjugate of frequency domain data and the filter.

Question formulation:

Spatial filtering:

The Sobel operator is a discrete differentiation operator. It combines Gaussian smoothing and differential derivation to calculate the approximate gradient of image gray function. What is the edge - the place where the pixel value jumps, one of the salient features of the image.

Sobel operator differentiates in the x and y directions:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

$$G = |G_x| + |G_y|$$

Frequency filtering:

Since time-domain filtering belongs to the linear convolution of finite sequence, the frequency-domain filtering is the process of finding time-domain linear convolution by using the discrete Fourier transform (DFT), and the DFT is essentially the circular convolution for periodic sequence in time-domain filtering. To satisfy the consistency of the calculation results of circular convolution and linear convolution, the time domain signal x(m point) and the filter h(n point) must be of the same length. This can be done by using the method of zero completion, so that the length of x and h is $L \geq m + n - 1$.The purpose of this is also to avoid the so-called frequency-entanglement error (aliasing) in the convolution of periodic functions caused by period-proximity.

The general practice is to extend the data in the space (time) domain to 2 times the data length by the way of back-end complements zero.

Since the filter and the extended signal should keep the same length, 0 is also needed to be extended.

Unilateral continuation, filter H is left.

Procedure:

Frequency domain filtering is divided into the following seven steps:

Given an input image with the size of $M * N$, set the filling parameters $f(x, y)$; let $P = 2M, Q = 2N$.

Add the necessary number of zeros to $f(x, y)$ to form a filled image $f_p(x, y)$ with the size of $P * Q$.

Multiply the $(-1)^{x+y}$ by moving it to the center of its transformation.

Calculate the DFT of the image obtained in the previous step and get $F(u, v)$.

A real symmetric filtering function $H(u, v)$ is generated, its size is $P * Q$, and its center is at $P/2, Q/2$. The array is multiplied to form a product $G(u, v) = H(u. v)F(u, v)$.

The processed image is obtained as follows: $g_p(x, y) = \{real[\zeta^{-1}[G(u, v)]]\}(-1)^{x+y}$, where the parasitic complex component caused by inaccurate calculation is ignored and the real part is selected, and the subscript p indicates that the processed array is filled.

The final processing result $g(x, y)$ is obtained by extracting the $M * N$ region from the upper left quadrant of $g(x, y)$.

Experiment:

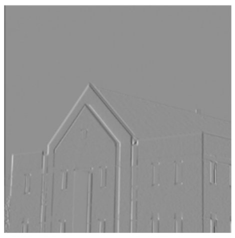
```python
import matplotlib.pyplot as plt
import numpy as np
import cv2


def sobel_filter_spatial(input_image):
    img = cv2.imread(input_image, 0)
    h = img.shape[0]
    w = img.shape[1]
    img_cp = np.pad(img, ((1, 1), (1, 1)), mode='constant', 
constant_values=0)
    image_new = np.zeros(img.shape)
    mask = [[1, 0, -1], [2, 0, -2], [1, 0, -1]]
    for i in range(1, h):
        for j in range(1, w):
            image_new[i, j] = np.sum(mask * img_cp[i - 1:i + 2, j - 1:j + 2])

    a = np.min(image_new)
    b = 255 / (np.max(image_new) - a)
    for i in range(0, h):
        for j in range(0, w):
            image_new[i, j] = int((image_new[i, j] - a) * b)
    plt.imshow(image_new, cmap='gray')
    plt.axis('off')
    plt.savefig(input_image.strip(".tif") + "
sobel_filter_spatial_11810506.tif", bbox_inches="tight",
                pad_inches=0.0)
    return 0
```

```python


def sobel_filter_frequency(input_image):
    img = cv2.imread(input_image, 0)
    h = img.shape[0]
    w = img.shape[1]
    image_new = np.zeros([2 * h, 2 * w])
    image_new[0:h, 0:w] = img

    I, J = np.ogrid[0:2 * h, 0:2 * w]
    mask = np.full((2 * h, 2 * w), -1)
    mask[(I + J) % 2 == 0] = 1
    image_new = mask * image_new

    image_new_fft = np.fft.fft2(image_new)

    kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    DFT_kernel = np.pad(kernel, (
        ((image_new.shape[0] - kernel.shape[0] + 1) // 2, (image_new.shape[0] - kernel.shape[0]) // 2),
        ((image_new.shape[1] - kernel.shape[1] + 1) // 2, (image_new.shape[1] - kernel.shape[1]) // 2)), mode='constant', constant_values=0)

    row, col = DFT_kernel.shape
    I, J = np.ogrid[0:row, 0:col]
    mask = np.full((row, col), -1)
    mask[(I + J) % 2 == 0] = 1
    DFT_kernel = mask * DFT_kernel

    DFT_kernel_fft = np.fft.fft2(DFT_kernel)

    output_frequency = np.real(np.fft.ifft2(image_new_fft * DFT_kernel_fft))

    row, col = output_frequency.shape
    I, J = np.ogrid[:row, :col]
    mask = np.full((row, col), -1)
    mask[(I + J) % 2 == 0] = 1
    output_frequency = mask * output_frequency

    # calculation error!!!!! -1 is important!!!!!!!!
    output_frequency = output_frequency[int(row / 2)-1:row-1, int(col / 2)-1:col-1]

    a = np.min(output_frequency)
    b = 255 / (np.max(output_frequency)-a)
    for i in range(0, h):
        for j in range(0, w):
            output_frequency[i, j] = int((output_frequency[i, j] - a)*b)
    plt.imshow(output_frequency, cmap='gray')
    plt.axis('off')
    plt.savefig(input_image.strip(".tif") + "sobel_filter_frequency_11810506.tif", bbox_inches="tight",
                pad_inches=0.0)
    return 0


sobel_filter_spatial("Q5_1.tif")
sobel_filter_frequency("Q5_1.tif")
```

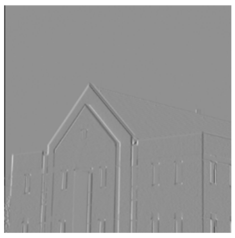
Results and analysis:

Results:

Spatial domain



Frequency domain



Analysis:

Both of filtering in frequency in spatial domain and frequency domain will lead to same results.

The correspondence between frequency components and image appearance can be used. Some enhancement tasks that are difficult to express in the spatial domain become quite common in the frequency domain.

Filtering is more intuitive in the frequency domain, which can explain some properties of spatial filtering.

You can specify the filter in the frequency domain, do the inverse transformation, and then use the result filter in the spatial domain as the guidance of the spatial domain filter

Generally, filtering in frequency will save times for DFT's help.

Error tips:

```
#calculation error!!!!! -1 is important!!!!!!!!
output_frequency = output_frequency[int(row / 2)-1:row-1, int(col / 2)
1:col-1]
```

Gaussian low pass and high pass:

Principle:

Gaussian filter is a linear filter, which can effectively suppress noise and smooth the image. Its acting principle is like the mean filter, which takes the mean value of the pixels in the filter window as the output. The coefficient of the window template is different from that of the average filter. The coefficient of the template of the average filter is the same, which is 1.However, the template coefficient of Gaussian filter decreases with the increase of distance from the template center. Therefore, the Gaussian filter is less fuzzy than the mean filter.

Most of the image noise belongs to Gaussian noise, so Gaussian filter is widely used. Gaussian filter is a linear smoothing filter, suitable for eliminating Gaussian noise, widely used in image denoising.

It can be simply understood as that Gaussian filtering denoising is the weighted average of the pixel value of the whole image. The value of each pixel is obtained after the weighted average of its own value and other pixel values in the neighborhood.

The specific operation of Gaussian filtering is as follows: a user-specified template (or convolution, mask) is used to scan each pixel in the image, and the weighted average gray value of the pixel in the neighborhood determined by the template is used to replace the value of the pixel in the center of the template.

In image processing, Gaussian filtering is generally realized in two ways, one is by discretization window sliding window convolution, the other is by Fourier transform. The most common is the first sliding-window implementation. Only when the discretized window is large and the calculation amount with the sliding-window is very large (i.e., the implementation using a separable filter), the implementation method based on Fourier variation may be considered.

Question formulation:

$$h(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

Where (x,y) is the point coordinate, which can be considered as an integer in image processing; Sigma is standard deviation. To get the template of a Gaussian filter, the Gaussian function can be discretized, and the value of the Gaussian function can be used as the coefficient of the template. For example, to generate a 3×3 Gaussian filter template, sample with the center of the template as the coordinate origin. The coordinates of the template in each position are shown below (X axis horizontal to the right, Y axis vertical down).

We can also filter the image in frequency domain, $H(u,v) = e^{-D^2(u,v)/2D_0^2}$ $D(u,v)$ is the distance between center and specific frequency in figure of frequency distribution.

Procedure:

1. Convert the image into array.
2. Shift the center by multiplicate $(-1)^{x+y}$
3. fft2d will finish the Fourier transformation.

4. Create the gaussian function as a mask for specific width and height of the image.
5. Times the shifted fft of image with mask to keep the wanted partition.

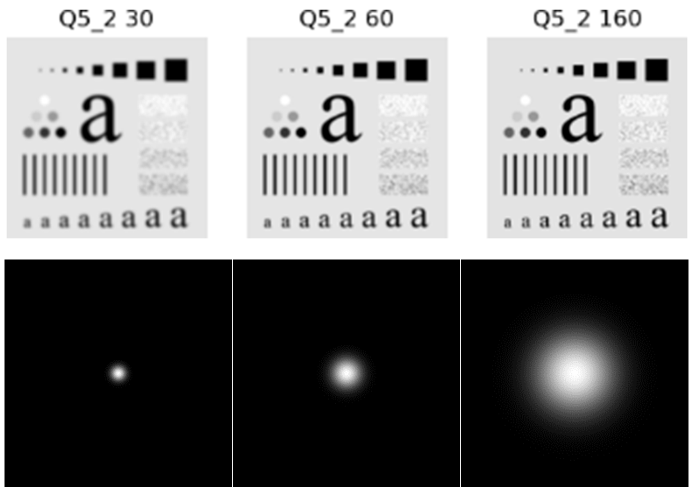
Experiment:

```python
import matplotlib.pyplot as plt
import numpy as np
import cv2


def GaussianHighFilter(image_input, d):
    img = cv2.imread(image_input, 0)
    f = np.fft.fft2(img)
    f_shift = np.fft.fftshift(f)
    s1 = np.log(np.abs(f_shift))

    def make_transform_matrix(d):
        transfor_matrix = np.zeros(img.shape)
        center_point = tuple(map(lambda x: (x - 1) / 2, s1.shape))
        for i in range(transfor_matrix.shape[0]):
            for j in range(transfor_matrix.shape[1]):
                def cal_distance(pa, pb):
                    from math import sqrt
                    dis = sqrt((pa[0] - pb[0]) ** 2 + (pa[1] - pb[1]) ** 2)
                    return dis

                dis = cal_distance(center_point, (i, j))
                transfor_matrix[i, j] = 1 - np.exp(-(dis ** 2) / (2 * (d ** 
2)))
        return transfor_matrix

    d_matrix = make_transform_matrix(d)
    new_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_shift * d_matrix)))
    return new_img


def GaussianLowFilter(image_input, d):
    img = cv2.imread(image_input, 0)
    f = np.fft.fft2(img)
    f_shift = np.fft.fftshift(f)
    s1 = np.log(np.abs(f_shift))

    def make_transform_matrix(d):
        transfor_matrix = np.zeros(img.shape)
        center_point = tuple(map(lambda x: (x - 1) / 2, s1.shape))
        for i in range(transfor_matrix.shape[0]):
            for j in range(transfor_matrix.shape[1]):
                def cal_distance(pa, pb):
                    from math import sqrt
                    dis = sqrt((pa[0] - pb[0]) ** 2 + (pa[1] - pb[1]) ** 2)
                    return dis

                dis = cal_distance(center_point, (i, j))
                transfor_matrix[i, j] = np.exp(-(dis ** 2) / (2 * (d ** 2)))
        return transfor_matrix

    d_matrix = make_transform_matrix(d)
    new_img = np.abs(np.fft.ifft2(np.fft.ifftshift(f_shift * d_matrix)))
    return new_img


plt.subplot(131)
plt.axis("off")
plt.imshow(GaussianHighFilter("Q5_2.tif", 30), cmap="gray")
plt.title('Q5_2 30')
plt.subplot(132)
plt.axis("off")
plt.title('Q5_2 60')
plt.imshow(GaussianHighFilter("Q5_2.tif", 60), cmap="gray")
plt.subplot(133)
plt.axis("off")
plt.title("Q5_2 160")
plt.imshow(GaussianHighFilter("Q5_2.tif", 160), cmap="gray")
plt.axis('off')
plt.savefig("GaussianHighFilter_11810506.tif", bbox_inches="tight", pad_inches=0.0)
plt.show()
plt.close()

plt.subplot(131)
plt.axis("off")
plt.imshow(GaussianLowFilter("Q5_2.tif", 30), cmap="gray")
plt.title('Q5_2 30')
plt.subplot(132)
plt.axis("off")
plt.title('Q5_2 60')
plt.imshow(GaussianLowFilter("Q5_2.tif", 60), cmap="gray")
plt.subplot(133)
plt.axis("off")
plt.title("Q5_2 160")
plt.imshow(GaussianLowFilter("Q5_2.tif", 160), cmap="gray")
plt.axis('off')
plt.savefig("GaussianLowFilter_11810506.tif", bbox_inches="tight",
    pad_inches=0.0)
plt.show()
plt.close()
```

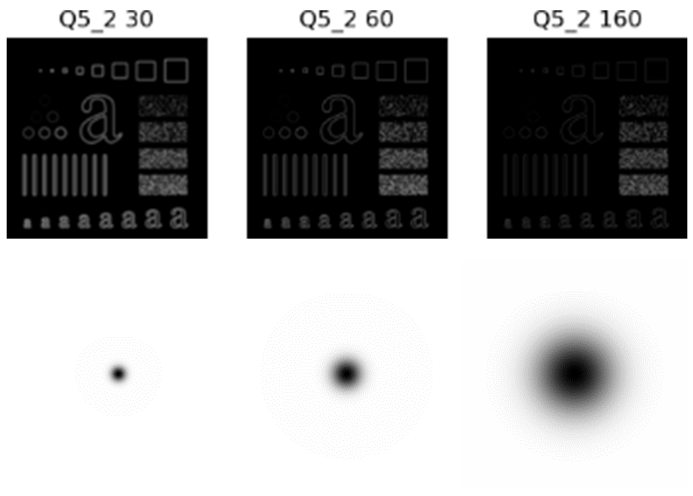
Results and analysis:

Results:

Gaussian Low Filter:



Gaussian High Filter:



Analysis:

Results: For low pass filter, as σ increased, higher partition of frequency will pass more. Then the image will become sharper, most of lower frequency reflect main information of image. For high pass filter, as σ increased, lower partition will pass less, so the image will become sharper, most of the higher frequency reflect the information of edge.

The Gaussian filter is a linear smoothing filter whose filter template is discrete to the two-dimensional Gaussian function. As the center value of the Gaussian template is the largest and the circumference decreases gradually, the filtered result is better than that of the mean filter.

The most important parameter of the Gaussian filter is the standard deviation of the Gaussian distribution σ, the standard deviation and the smoothing ability of the Gaussian filter have a great ability, the greater the σ, the wider the frequency band of the Gaussian filter, the smoothing degree of the image is better. By adjusting σ parameters, the noise suppression and image blurring can be balanced.

Butterworth notch filters:

Principle:

Butterworth filter (Butterworth filter) is a kind of electronic filter, it is also called the maximum flat filter. The Butterworth filter is characterized by a frequency response curve that is as flat as possible in the pass band, with no ripple, and gradually drops to zero in the stop band.

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

With larger n, higher order, the higher frequency component will be filtered. the larger n, the better selectivity of the filter.

Question formulation:

Procedure:

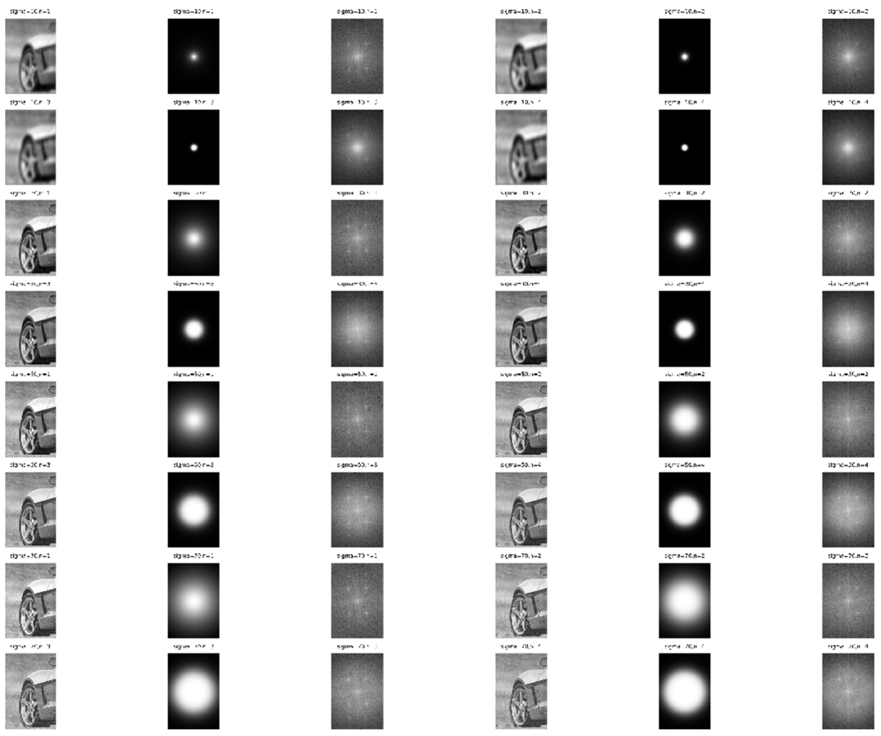
1. Load the image into array.
2. Generate the Butterworth filter with specific dimension
3. Use $(-1)^{x+y}$ to shift.
4. Times the image with mask to obtain the output.
5. Ifft2d will transform the output into output image.

Experiment:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
from matplotlib.ticker import MultipleLocator


def butterworthPassFilter(image_input, sigma, n):
    image_shift = np.fft.fftshift(np.fft.fft2(cv2.imread(image_input, 0)))

    center_point = [[124, 84]]

    def make_transform_matrix(center_p):
        transform_matrix = np.zeros(image_shift.shape)
        for i in range(transform_matrix.shape[0]):
            for j in range(transform_matrix.shape[1]):
                def cal_distance(pa, pb):
                    from math import sqrt
                    distance = sqrt((pa[0] - pb[0]) ** 2 + (pa[1] - pb[1]) **
2)
                    return distance

                dis = cal_distance(center_p, (i, j))
                transform_matrix[i, j] = (1 / (1 + (dis / sigma) ** (2 * n)))
        return transform_matrix

    mask = np.ones(image_shift.shape)
    for center in center_point:
        mask = mask * make_transform_matrix(center)

    output_frequency = np.multiply(image_shift, mask)

    new_img = np.abs(np.fft.ifft2(np.fft.ifftshift(output_frequency)))

    return new_img, mask * 255,np.log(np.abs(output_frequency))


img = cv2.imread("Q5_3.tif", 0)
f = np.fft.fft2(img)
f_shift = np.fft.fftshift(f)
s1 = np.log(np.abs(f_shift))
plt.imshow(s1)
x_major_locator = MultipleLocator(20)
y_major_locator = MultipleLocator(20)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
ax.yaxis.set_major_locator(y_major_locator)
plt.tick_params(labelsize=4)
plt.grid()
plt.savefig("frequency 0", bbox_inches="tight",
            pad_inches=0.0, dpi=1000)
plt.close()

i = 1
for sigma in [10, 30, 50, 70]:
    for n in [1, 2, 3, 4]:
        plt.subplot(8, 6, i)
        plt.imshow(butterworthPassFilter("Q5_3.tif", sigma, n)[0], cmap="gray")
        plt.title("sigma=" + str(sigma) + ",n=" + str(n), fontsize=2,pad=2)
        plt.axis("off")

        plt.subplot(8, 6, i + 1)
        plt.imshow(butterworthPassFilter("Q5_3.tif", sigma, n)[1], cmap="gray")
        plt.title("sigma=" + str(sigma) + ",n=" + str(n), fontsize=2,pad=2)
        plt.axis("off")

        plt.subplot(8, 6, i + 2)
        plt.imshow(butterworthPassFilter("Q5_3.tif", sigma, n)[2], cmap="gray")
        plt.title("sigma=" + str(sigma) + ",n=" + str(n), fontsize=2, pad=2)
        plt.axis("off")
        i = i + 3
plt.savefig("butterworthPassFilter_11810506.tif", bbox_inches="tight", pad_inches=0.0, dpi=5000)
plt.close()
```

Results and analysis:

Results:



Analysis:

See the results of lab, for a low order n, high σ will cause over deleting and blurred image. The high order n precise σ will delete the unneeded partition more precisely. So, if we want to make a clear image, we need to get the precise dimension of σ, and use high order n to make a precise deleting.

In my experiment, I think n = 3, σ = 40, give the best answer. In my conclusion, high n and suitable σ will give a satisfied answer.

Conclusion:

In this experiment, we implemented the filtering in frequency domain and spatial domain by Sobel calculator. We use gaussian filter to compare the function of high pass and low pass filter. Butterworth filters help us to deleting the noise frequency more precisely.

Compare with frequency and spatial filtering, we find frequency filtering will save time, as we invent the DFT, which save lots of time of calculation.

In frequency domain, we can get more frequency information to remove some small area distribution noise in frequency image.