Data Visualization HW2

(Cheng Zhong 16307110259)

1 Smoothing & Sharpening

Whether smoothing or sharpening operation, it is just using different kernels for convolution.
As for smoothing, I use the smoothing spatial filters for smoothing and laplacian operator for sharping.

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$ — Smoothing spatial filters $$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$ — Laplacian operator

And here is the code:

```
def convolution(region, filter):
    '''
    region -- the origin picture
    filter -- the convolution filter
    '''
    padding = np.zeros((np.shape(region)[0] + 1, np.shape(region)[1] + 1))
    # padding with 0
    width, height = np.shape(region)
    for i in range(1, width+1):
        for j in range(1, height+1):
            padding[i,j] = region[i-1, j-1]

    new = np.zeros((width, height))
    convwidth, convheight = np.shape(filter)
    # convolutional operation
    for i in range(1, width+1):
        for j in range(1, height+1):
            convsum = 0 # the value of new pixel
            for s in range(convwidth):
                for t in range(convheight):
                    convsum += filter[s,t] * padding[i-s+int(s/2), j-t+int(t/2)]
            new[i-1, j-1] = int(convsum)
    # form a new picture
    '''
    # When sharpening
    new = region - new
    '''
    im = Image.fromarray(new)
    im = im.convert('L')
    im.show()
    im.save('Conv.png')
```

Here is the image before and after convolution. We can find that the smoothing image is relatively blurry but the borders are also smooth. Sharpening makes the edges of the image clearer.

(a) Origin Picture

(b) Smoothing

(c) Filtering

(d) Sharpening

Figure 1: The image before and after Convolution

2 Convolution theorem of discrete Fourier transform of two-dimensional variables

First, we prove the Time shift of two-dimensional Fourier transform:

$$F(f(t-t_0,z-z_0)) = \sum_{t=0}^{M-1}\sum_{z=0}^{N-1} f(t-t_0,z-z_0)e^{-j2\pi(\frac{t\mu}{M}+\frac{zv}{N})}$$

Denote $x = t - t_0, y = z - z_0$, so

$$\begin{aligned} F(f(x,y)) &= \sum_{x=-t_0}^{M-1-t_0}\sum_{y=-z_0}^{N-1-z_0} f(x,y)e^{-j2\pi(\frac{(x+t_0)\mu}{M}+\frac{(y+z_0)v}{N})} \\ &= F(f(t,z))e^{-j2\pi(\frac{t_0\mu}{M}+\frac{z_0v}{N})} \end{aligned}$$

Since

$$
\begin{aligned}
F(\mu,v) &= \sum_{t=0}^{M-1}\sum_{z=0}^{N-1} f(t,z)e^{-j2\pi(\frac{\mu t}{M}+\frac{vz}{N})} \\
f(t,z) &= \sum_{\mu=0}^{M-1}\sum_{v=0}^{N-1} F(\mu,v)e^{-j2\pi(\frac{\mu t}{M}+\frac{vz}{N})} \\
g(x,y) &= f(x,y) * h(x,y) = \sum_{\tau_u=0}^{M-1}\sum_{\tau_v=0}^{N-1} f(\tau_u,\tau_v)h(x-\tau_u,y-\tau_v) \\
F(g(x,y)) &= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(\sum_{\tau_u=0}^{M-1}\sum_{\tau_v=0}^{N-1} f(\tau_u,\tau_v)h(x-\tau_u,y-\tau_v))e^{-j2\pi(\frac{\mu x}{M}+\frac{vy}{N})} \\
&= \sum_{\tau_u=0}^{M-1}\sum_{\tau_v=0}^{N-1} f(\tau_u,\tau_v)(\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x-\tau_u,y-\tau_v)e^{-j2\pi(\frac{\mu x}{M}+\frac{vy}{N})}) \\
&= H(x,y)\sum_{\tau_u=0}^{M-1}\sum_{\tau_v=0}^{N-1} f(\tau_u,\tau_v)e^{-2\pi(\frac{\mu x}{M}+\frac{vy}{N})} \\
&= H(\mu,v)F(\mu,v)
\end{aligned}
$$

Parallelly,

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

So,

$$
\begin{aligned}
F^{-1}(\frac{1}{MN}F(u,v) * H(u,v)) &= \frac{1}{M^2N^2}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(m,n)H(u-m,v-n)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \\
&= \frac{1}{M^2N^2}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} H(u-m,v-n)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \\
&= h(x,y)(\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(m,n)e^{j2\pi(\frac{mx}{M}+\frac{ny}{N})}) \\
&= h(x,y)f(x,y)
\end{aligned}
$$

Q.E.D

3 Filters in frequency domain

As for frequency domain filtering, we need to transform the picture into frequency domain and then filtering. After that, we can invert the filtered image to form a new image. Here is my code for filtering in frequency domain, we only need to fill it with appropriate filter.

```
def fft(img):
    # for fast fourier transformation
    f = np.fft.fft2(img)
```

```
    fshift = np.fft.fftshift(f)
    show_fshift = np.log(np.abs(fshift))
    # show the Spectrogram
    plt.imshow(show_fshift, 'gray'), plt.title('Fourier Image')
    plt.show()
    # Different filters in 3.1 and 3.2
    '''
    GLPF_filter = GLPF(img, 40)
    img_GLPF = GLPF_filter * fshift
    show_img_GLPF = np.log(np.abs(img_GLPF))
    plt.imshow(show_img_GLPF, 'gray'), plt.title('Filtered Image')
    '''
    '''
    B_filter = Bandreject_filters(show_fshift)
    img_B = B_filter * fshift
    show_img_B = np.log(np.abs(img_B))
    plt.imshow(show_img_B, 'gray'), plt.title('Filtered Image')
    '''
    plt.show()
    # for IDFT and form a new picture
    img_ifftshifted = np.fft.ifftshift(img_B)
    img_ifft = np.fft.ifft2(img_ifftshifted)
    img_real = np.abs(np.real(img_ifft))
    img_new = np.clip(img_real,0,255)
    plt.imshow(img_new, 'gray'), plt.title('New Image')
    plt.show()
```

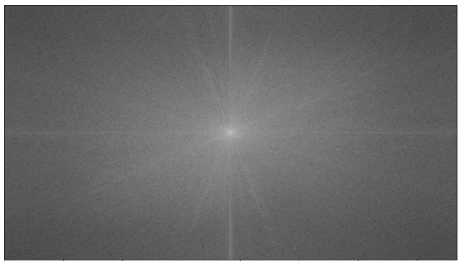
3.1 GLPF

In this part, we use the GLPF as filter.

```
def GLPF(img, D):
    # size of filter
    width, height = np.shape(img)[1], np.shape(img)[0]
    u = range(width)
    v = range(height)
    u, v = np.meshgrid(u, v)
    #  Gaussian operation
    low_pass_filter = np.sqrt( (u - width/2)**2 + (v - height/2)**2)
    G = np.exp((-1 * low_pass_filter**2) / (2 * D ** 2 ))
    return np.clip(G, 0, 1)
```

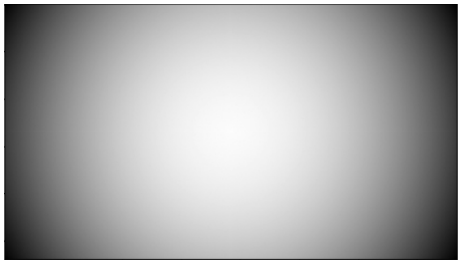
and here is the spectrogram of the original image, the spectrogram of the result in the frequency domain, and the new image after operation. We can find that after GLPF, the image becomes more blurred.

(a) Origin Image



(b) Spectrogram

(c) New Image



(d) Spectrogram after filtering

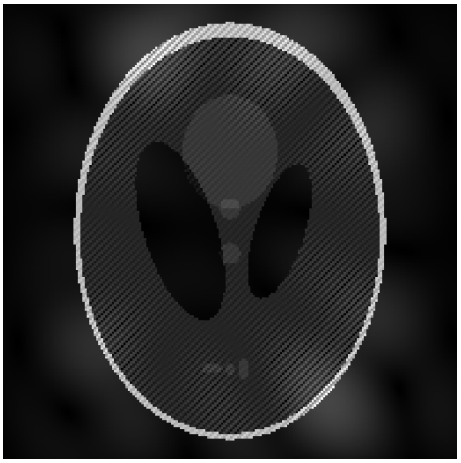
Figure 2: The image before and after filtering in frequency domain

3.2 Bandreject filter

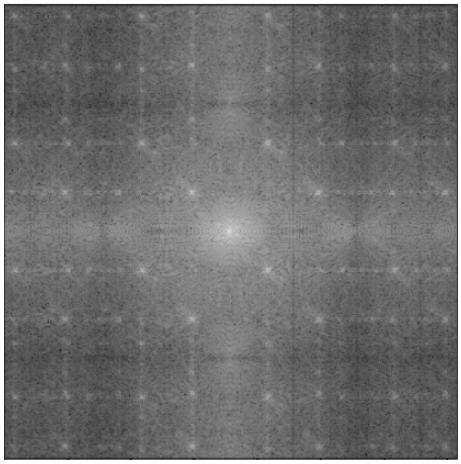
In this part, we use the band-pass filter as filter.

```
def Bandreject_filters(img):
    width, height = np.shape(img)[0], np.shape(img)[1]
    B_filters = np.zeros((width,height))
    # try different filters
    #B_filters[img < 9] = 1
    B_filters[310:400,:] = 1
    B_filters[:, 310:400] = 1
    #B_filters[:,300:400] = 1
    return B_filters
```

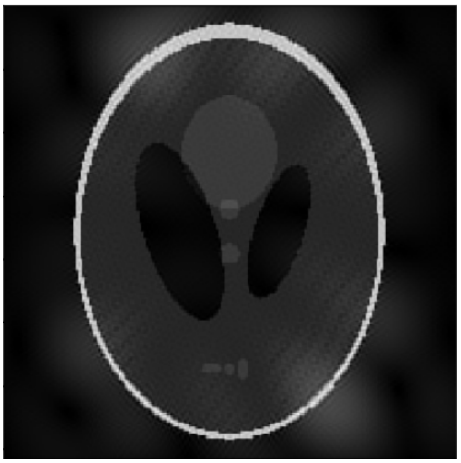
and here is the spectrogram of the original image, the spectrogram of the result in the frequency domain, and the new image after operation. From the spectrogram we can see there are many points with high absolute value, which represent the stripes in the image. We can find that after filter, we can remove these points and remove the stripes in the image.



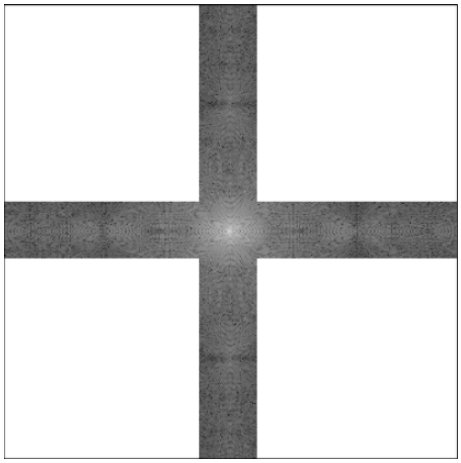
(a) Origin Image



(b) Spectrogram



(c) New Image



(d) Spectrogram after filtering