中山大学计算机学院

信号与系统

综合实验报告

(2022学年春季学期)

小组成员	分工
李昊伟	实验设计与实现、报告撰写
李君豪	文献调研
唐梓杰	文献调研
谭夷翔	文献调研

1. 实验题目

实验三：滤波器设计在图像处理中的应用

2. 实验的相关知识准备

1. 二维傅里叶变换

通俗来讲，一维傅里叶变换是将一个一维的信号分解成若干个三角波。这个信号说白了就是一个时域上的函数，不管是离散的还是连续的。

在上学期的《数学分析III》中我们已经学过，函数可以推广到多元。那么，一个具有二元变量的离散信号，比如图像，也会有它对应的二维傅里叶变换。

参照《数字图像处理》第四章的内容，我们可以推广得到二维图像的傅里叶变换对以及它的性质。由于篇幅原因，这里只贴出二维图像傅里叶变换对的公式。

DFT(傅里叶变换):

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

IDFT(傅里叶反变换):

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

> 在二维离散情况下，我们用`(x,y)`表示空间变量，`(u,v)`表示时间变量。

对于一个M * N的图像，可以看作M行N列的二维数组，先对每行做一维FFT，将结果作为一个新的二维数组。再对新的二维数组每列做一维FFT。而在处理二维IFFT的时候，跟二维FFT差不多，只要在公共方法中控制正负值，来区分是FFT还是IFFT。

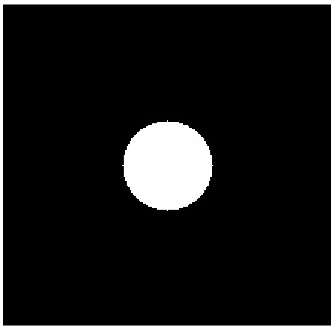
在opencv中和numpy中，都实现了对图像的二维傅里叶变换和反变换，在本实验中将直接使用。

以下是Lena图在空间域与时间域上的形式：

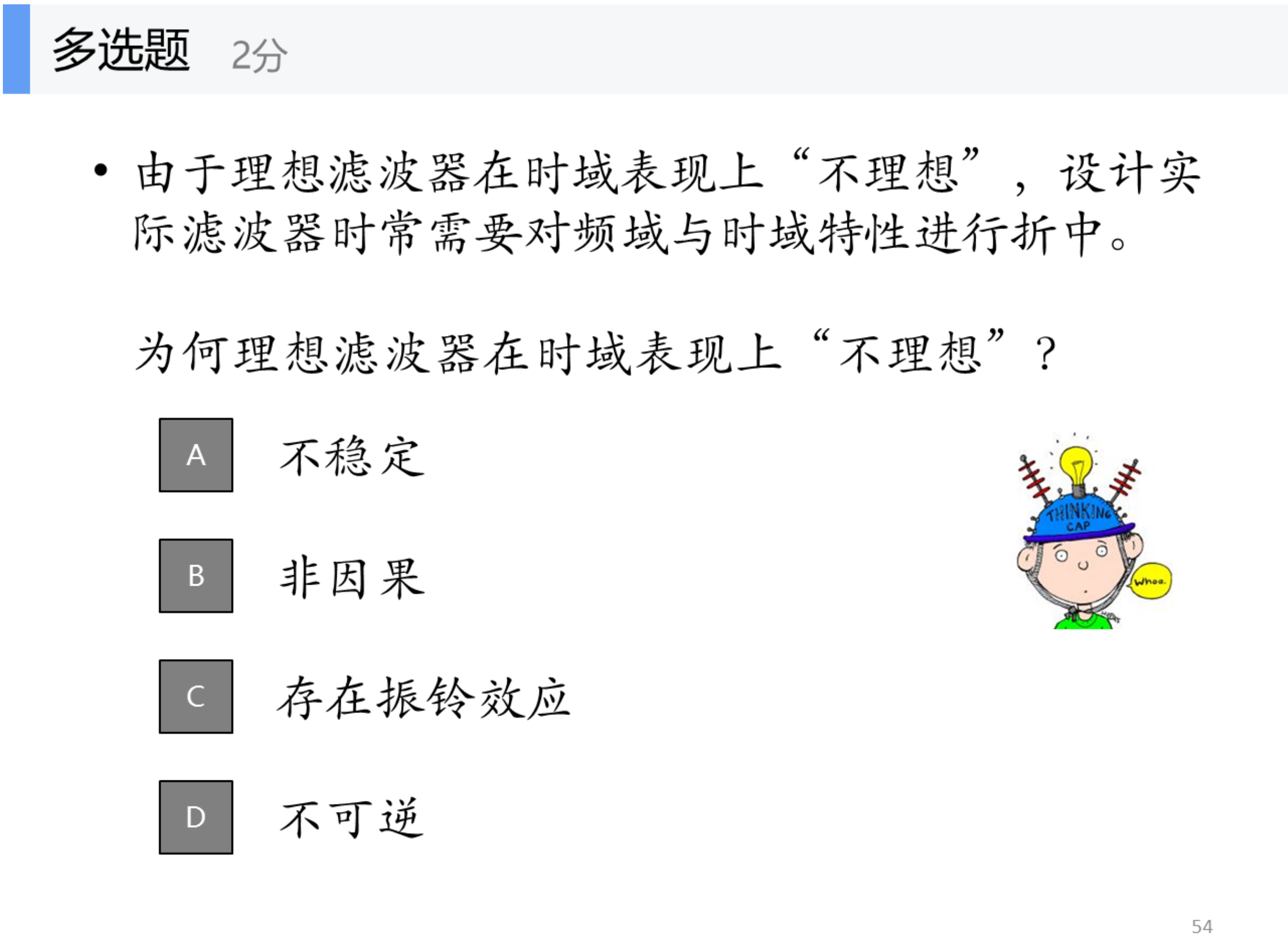
2. 巴特沃斯滤波器

巴特沃斯（Butterworth）滤波器是一种具有最大平坦幅度响应的低通滤波器。

说起低通滤波器，我们第一个想到的是理想低通滤波器。也就是把限定低频范围内所有信号成分保留，然后把不在限定范围内的所有信号成分滤去。



这样的滤波器，如课堂习题

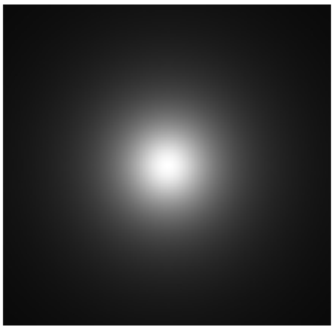


会引起振铃效应。在实验当中我们将分析振铃效应的成因，并发现如果使用巴特沃斯滤波器会有效地消除振铃效应。

截止频率位于距离原点D__0处的n阶巴特沃斯低通滤波器的传递函数定义为:

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

直观上来讲，该滤波器会使非截止频率范围内的信号成分通过。如图：



在实验中，我将分析振铃现象的成因以及实现没有振铃现象的巴特沃斯滤波器。

3.退化函数

对于一个图像的运动模糊过程，在空间域上似乎是比较复杂的。我们很想用频域上的一些加加减减乘乘除除来表示它，这是可以做到的吗？首先，我们要知道一个图像的退化过程是怎样用一个函数来表示的，这就是退化函数。

我们首先了解一些概念。

如果我们把f(x , y)视为空间域上的图像信号的话，那么我们可以把退化过程描述为：

$$g(x,y)=H[f(x,y)]+\eta(x,y) \qquad (1)$$

其中H表示作用在图像上的退化,η表示加性噪声，就是一个与位置无关，直接加在空间域图像上的噪声项。

- 线性：

  $$H[af_1(x,y)+bf_2(x,y)]=aH[f_1(x,y)]+bH[f_2(x,y)] \qquad (2)$$

- 位置不变性(类比时不变性)：

  $$H[f(x-\alpha,y-\beta)]=g(x-\alpha,y-\beta) \qquad (3)$$

  也就是说图像中任意一点的响应只取决于其输入，而与位置本身无关。

- 点扩散函数(PSF)

  我们推广一维的取样函数到二维，得到：

  $$f(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\alpha,\beta)\times\delta(x-\alpha,y-\beta)d\alpha d\beta$$

  代入(1), 假设H是线性系统， 得到：

  $$g(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\alpha,\beta)\times h(\delta(x-\alpha,y-\beta))d\alpha d\beta$$

  已经很明了了，这里的h(x , α , β , y) = h(δ (x - α , y - β))对应着一维里面的单位冲激响应，图像里的一个冲击为一个光电，响应会是一个模糊化了的光点，所以将冲激响应称为点扩散函数。

  同样类比一维，由于复指数信号是线性位置不变系统的特征函数，单位冲激响应的傅里叶变换是特征值，不用推导也能想象到以下结论：

  $$g(x,y)=h(x,y)*f(x,y)+\eta(x,y)$$

  $$G(u,v)=H(u,v)\times F(u,v)+N(u,v)$$

事实上，任何一个线性空间不变退化系统，如果它具有加性噪声，那它退化后的频域图G(u,v)可以被表示为：

$$G(u,v)=H(u,v)\times F(u,v)+N(u,v)$$

4. 逆滤波和维纳滤波

已知有

$$G(u,v)=H(u,v)\times F(u,v)+N(u,v)$$

可以推出F(u , v)的估计:

$$\hat{F}(u,v)=F(u,v)+\frac{N(u,v)}{H(u,v)}$$

这就是逆滤波，直接忽略噪声，把 H 除掉就好。可是如果H(u , v)为0或者很小的值，$\frac{N(u,v)}{H(u,v)}$会很容易支配估计值，使得恢复的图像成为一团混乱。

于是引入维纳滤波。

维纳滤波也称最小均方误差滤波，它能处理被退化函数退化和噪声污染的图像。该滤波方法建立在图像和噪声都是随机变量的基础之上，目标是找到未污染图像f(x,y)的一个估计，使它们之间的均方误差最小。

这种误差度量用以下式子表示:

$$e^2=E[(f-\hat{f})^2]$$

其中，E是参数的期望值。其数学推导比较复杂，篇幅原因这里不作展开，直接给出估计值的表达式:

$$\hat{F}(u,v)=[\frac{1}{H(u,v)}\times\frac{|H(u,v)|^2}{|H(u,v)|^2+K}]\times G(u,v)$$

3. 实验内容

1. 实验任务1：

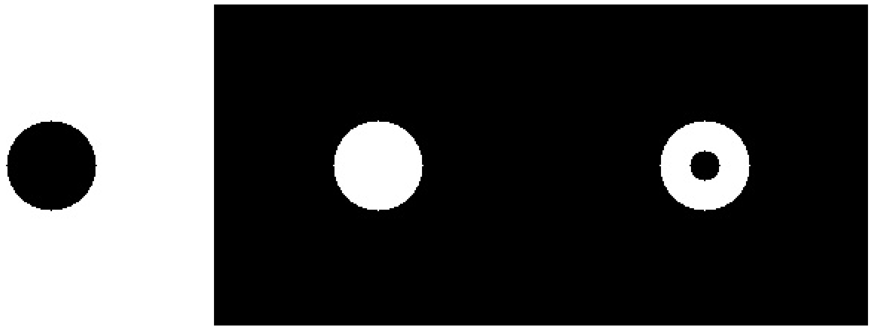
对数字图像进行不同参数下的理想滤波（低通、高通、带通），并分析得到的结果

- **解决问题的方法分析**
  1. 将图像经过二维傅里叶变换，变换到频域上去，并且通过移动，使得频率图显现中间低频，外围高频的格局。
  2. 设计相应的滤波器掩膜。
  3. 将相应的掩膜与频域图像相乘，再将图像变换回去。

- **实验过程与关键代码:**
  1. 将图像经过二维傅里叶变换，变换到频域上去，并且通过移动，使得频率图显现中间低频，外围高频的格局。

```
def dft(img):
    img_float32 = np.float32(img)
    dft = cv2.dft(img_float32,flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    return dft , dft_shift
```

  2. 设计相应的滤波器掩膜。



```
def high_pass_filter(shape, radius):
    h , w = shape
    mask = np.ones((shape[0] , shape[1] , 2), dtype=np.uint8)
```

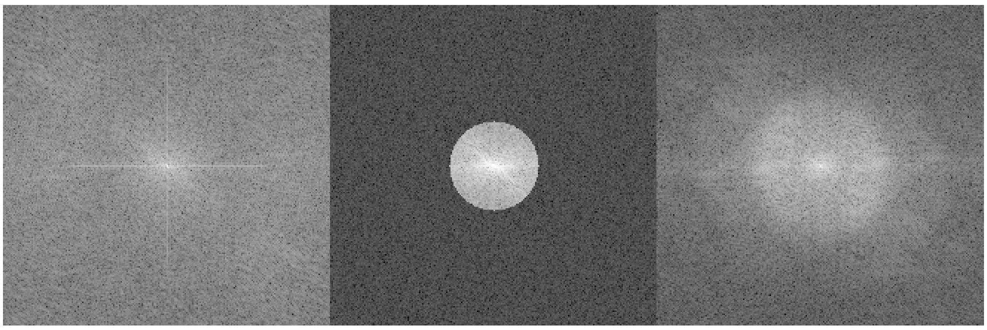
```
    cv2.circle(mask, (int(w / 2), int(h / 2)), radius, (0, 0, 0), -1)
    filter_for_show = np.full(shape[:2], 255 , dtype=np.uint8)
    cv2.circle(filter_for_show, (int(w / 2), int(h / 2)), radius, (0, 0, 0), -1)
    return mask , filter_for_show

def high_pass_filtering(filter_range , shape , spectrum):
    # 理想高通滤波器: 将中心半径30内的谐波全部滤掉(置为0)
    mask , high_pass_filter_for_show = high_pass_filter(shape , filter_range)
    filtered_spectrum = mask * spectrum
    # 将频谱从中心低频的状态移动回原来的状态
    shifted_back = np.fft.ifftshift(filtered_spectrum)
    # 傅里叶反变换
    img_back = cv2.idft(shifted_back)
    # 计算幅度值
    img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
    # 由于快速傅里叶变换的性质，这里除上采样个数
    img_back = np.uint8(img_back / (shape[0] * shape[1]))

    show_high_pass = get_spectrum_as_picture(filtered_spectrum)
    cv2.imwrite("./result/high_pass/high_pass_filtered_spectrum1.jpg" , show_high_pass)
    cv2.imwrite("./result/high_pass/high_pass_filter.jpg" , high_pass_filter_for_show)
    return img_back , mask
### 低通，带通同理，见源码
```

3. 将相应的掩膜与频域图像相乘，再将图像变换回去

- **结果分析:**

1. 输出的图片(分别是高通、低通、带通):

2. 输出的图片的频谱(分别是高通、低通、带通):



3. 振铃现象:

可以看到理想低通滤波器产生的图片表面很不光滑，有很多如同振铃使得空气振动产生的气泡状。究其原因，是由于像素值快速变化造成的。

由于理想低通滤波器在频率域中、把其中一维视为定值，将二维函数看成是个一维函数。发现它是一个方波信号。那么，在空间域，其信号具有sinc函数形状。将图像的每一个像素视为一个离散冲激，它的强度和灰度成正比。我们熟悉，频率域相乘，空间域卷积。那么就相当于在冲击的地方复制这个sinc函数。事实上，二维理想低通滤波器的mask在频域上类似一个小山包，相当于四处搬运这个sinc小山包。sinc的中心波瓣是引起模糊的主因，而外侧较小的波瓣是造成振铃的主要原因。

实验任务2.

设计适当的非理想滤波器（如巴特沃斯滤波器）对相同的图像进行滤波，并与理想滤波结果进行对比分析；

- 解决问题的方法分析：

  前文当中已经提到过巴特沃斯滤波器的表达式，只需要根据表达式生成巴特沃斯滤波器的mask，其他的和任务一完全一样。

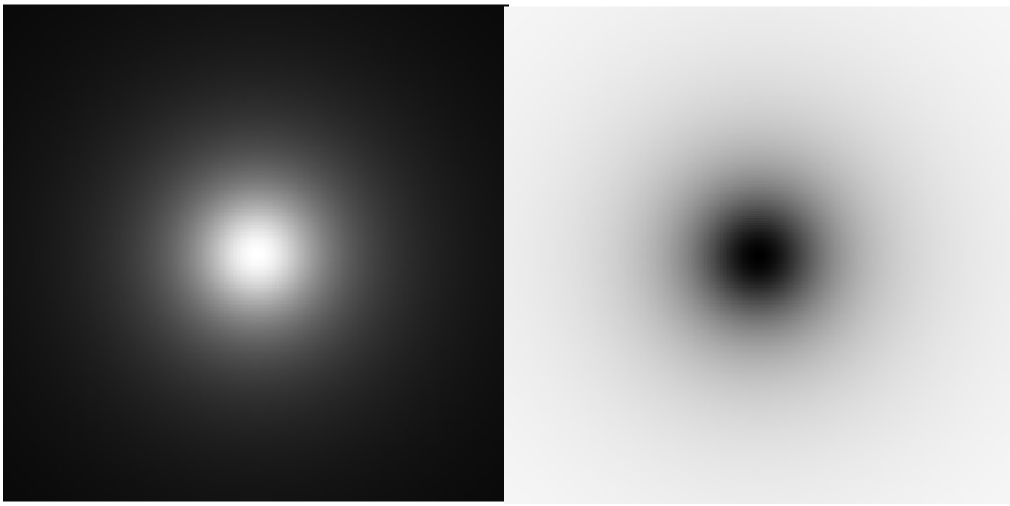
- 实验过程与关键代码：

```
def butterworth_lp_filter(shape, rank, radius):
    # 中心位置
    h, w = shape[:2]
    cx, cy = int(w / 2), int(h / 2)
    # 计算以中心为原点坐标分量
    u = np.array([[x - cx for x in range(w)] for i in range(h)], dtype=np.float32)
    v = np.array([[y - cy for y in range(h)] for i in range(w)], dtype=np.float32).T
    # 每个点到中心的距离
    dis = np.sqrt(u * u + v * v)
    mask = 1 / (1 + np.power(dis / radius, 2 * rank))
    filter_for_show = mask * np.full((shape) , 255 , dtype=np.uint8)
    mask = mask.reshape(shape[0] , shape[1] , 1).repeat(2,axis=2)
    return mask , filter_for_show

def butterworth_hp_filter(shape, rank, radius):
    # 中心位置
    h, w = shape[:2]
    cx, cy = int(w / 2), int(h / 2)
    # 计算以中心为原点坐标分量
    u = np.array([[x - cx for x in range(w)] for i in range(h)], dtype=np.float32)
    v = np.array([[y - cy for y in range(h)] for i in range(w)], dtype=np.float32).T
    # 每个点到中心的距离
    dis = np.sqrt(u * u + v * v)
    # 高通滤波
    mask = 1 - 1 / (1 + np.power(dis / radius, 2 * rank))
    filter_for_show = mask * np.full((shape) , 255 , dtype=np.uint8)
    mask = mask.reshape(shape[0] , shape[1] , 1).repeat(2,axis=2)
    return mask , filter_for_show

def butterworth_filtering(rank , filter_range , shape , spectrum , mode):
    if mode == 'low_pass':
        mask , butterworth_filter_for_show = butterworth_lp_filter(shape , rank , filter_range)
    else:
        mask , butterworth_filter_for_show = butterworth_hp_filter(shape , rank , filter_range)
    filtered_spectrum = mask * spectrum
    # 将频谱从中心低频的状态移动回原来的状态
```

```
    shifted_back = np.fft.ifftshift(filtered_spectrum)
    # 傅里叶反变换
    img_back = cv2.idft(shifted_back)
    # 计算幅度值
    img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
    # 微调,见实验报告本处注释
    img_back = np.uint8(img_back / (shape[0] * shape[1]))

    show_low_pass = get_spectrum_as_picture(filtered_spectrum)
    if mode == "low_pass":
        cv2.imwrite("./result/butterworth/butterworth_filtered_spectrum1.jpg" , show_low_pass)
        cv2.imwrite("./result/butterworth/butterworth_filter.jpg" , butterworth_filter_for_show)
    else:
        cv2.imwrite("./result/butterworth/butterworth_hp_filtered_spectrum1.jpg" , show_low_pass)
        cv2.imwrite("./result/butterworth/butterworth_hp_filter.jpg" ,
butterworth_filter_for_show)
    return img_back , mask
```

- 结果分析:

  巴特沃斯滤波器模板示意图:



巴特沃斯滤波与理想滤波对比:

巴特沃斯低通滤波器的阶数越高，其在空间域上的“震荡“就越弱，如同冈萨雷斯的《数字图像处理 第三版》图4.46所示那样。（我在网络上似乎难以找到此图），导致产生的振铃现象越弱。

实验任务3

假设图像拍摄时存在运动模糊，之后还叠加了一定程度的高斯随机噪声，试通过编程对这一图像质量退化过程进行仿真。

- 运动模糊的退化函数推导：从运动模糊的成因开始(建模估计法)

  假设图像曝光的时间为T，在这段时间里图像运动的水平、垂直距离分别是$x_0(t)$和$y_0(t)$.

  根据照片成像原理，$g(x,y)=\int_0^T f[x-x_0(t),y-y_0(t)]dt$

  直接对其进行傅里叶变换：

  $$G(u,v)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}g(x,y)\times e^{-j2\pi(ux+vy)}dxdy=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[\int_0^T f[x-x_0(t),y-y_0(t)]dt]\times e^{-j2\pi(ux+vy)}dxdy$$

  改变积分次序：

  $$G(u,v)=\int_0^T[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f[x-x_0(t),y-y_0(t)]\times e^{-j2\pi(ux+vy)}dxdy]dt$$

  发现方括号内正好是$f(x-x_0(t),y-y_0(t))$的傅里叶变换。

  $$G(u,v)=F(u,v)\times\int_0^T e^{-j2\pi[ux_0(t)+vy_0(t)]}dt$$

  也就是我们熟悉的：

  $$G(u,v)=H(u,v)F(u,v)$$
  $$H(u,v)=\frac{T}{\pi(ua+vb)}\times sin[\pi(ua+vb)]\times e^{-j\pi(ua+vb)}$$

- 运动模糊的退化函数推导，从PSF开始

  $$h(x,y)=\frac{1}{L}\qquad 0\leqslant x\leqslant L,\ y=0\qquad(4)$$

  对式(4)中的点扩散函数做傅立叶变换：

  $$\begin{aligned}H(u,v)&=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}h(x,y)e^{-j2\pi(ux+vy)}\mathrm{d}x\mathrm{d}y\\&=\int_0^L\frac{1}{L}e^{-j2\pi ux}\mathrm{d}x\\&=\frac{\sin(\pi uL)}{\pi uL}e^{-j\pi uL}\qquad(5)\end{aligned}$$

可以发现，这里的`L = 1/a`

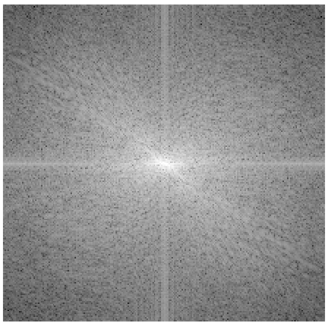
- 实验过程与关键代码
  - 生成运动模糊退化函数H

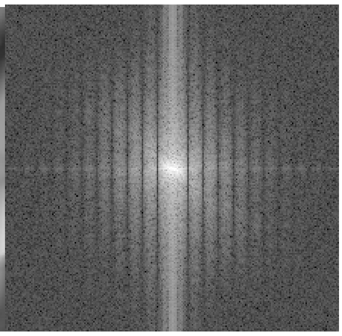
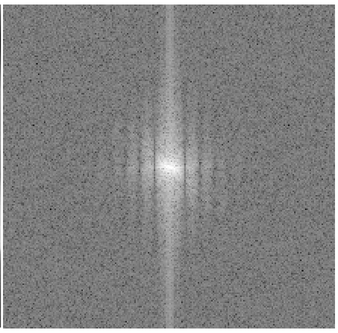
```
def degradation_function(pic , a=0, b=0 , T=1):
    [r,c] = pic.shape
    u = np.arange(r).reshape((-1,1)) - np.ceil(r/2)
    v = np.arange(c)-np.ceil(c/2)
    tmp = np.pi*(u*a + v*b)      #广播机制得到矩阵
    tmp[tmp==0]=1e-20
    H = T*np.sin(tmp)/tmp*np.exp(-1j*tmp);  # 退化函数
    return H
```

  - 进行运动模糊,添加高斯噪声

```python
def motion_blur_v2(pic , H):
    g = np.fft.ifft2(H * np.fft.fft2(pic));
    g = np.uint8(np.real(g))
    return g
```

```python
def add_gauss_noise(img, sigma):
    img = img/255
    noise = np.random.normal(0,sigma,img.shape)
    output = img + noise
    output = np.clip(output, 0, 1)
    output = np.uint8(output * 255)
    return output
```

- 实验结果分析
    - 添加运动模糊前的图像及其频谱
    -  
    - 添加运动模糊后的图像及其频谱

       
    - 添加高斯噪声后的运动模糊图像及其频谱
    -  

实验任务4

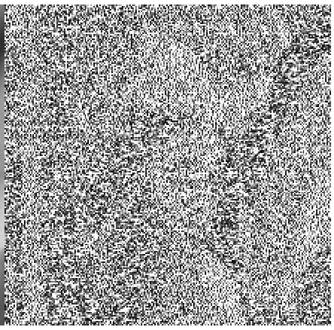
如果已知上述图像退化过程的参数，请编程实现对已退化图像的复原。

- 实验过程与关键代码
    - 尝试直接逆滤波

```python
def inverse_filtering(input, H, eps):
    input_fft = np.fft.fft2(input)
    PSF_fft = H + eps # 避免除数为0添加的一个极小量，可以防备一定的噪声
    result = np.fft.ifft2(input_fft / PSF_fft)
    result = np.abs(result)
    return result
```

    - 维纳滤波

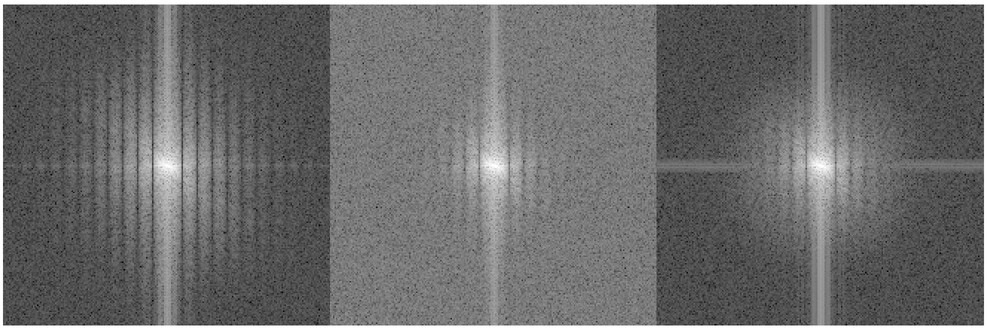
```python
def wiener_filtering(input_signal, H, K , eps):
    input_signal_cp = np.copy(input_signal) # 输入信号的副本
    input_signal_cp_fft = np.fft.fft2(input_signal_cp)  # 输入信号的傅里叶变换
    PSF_fft = H + eps
    h_abs_square = np.abs(PSF_fft)**2 # 退化函数模值的平方
    # 维纳滤波
    output_signal_fft = np.conj(PSF_fft) / (h_abs_square + K)
    output_signal = np.abs(np.fft.ifft2(output_signal_fft * input_signal_cp_fft)) # 输出信号傅里叶反变换
    return output_signal
```

- 实验结果分析
    - 直接逆滤波作用在无噪声的运动模糊图像上
    - 
    - 直接逆滤波作用在有噪声的运动模糊图像上
    - 
    - 维纳滤波作用在有噪声的运动模糊图像上

   - 

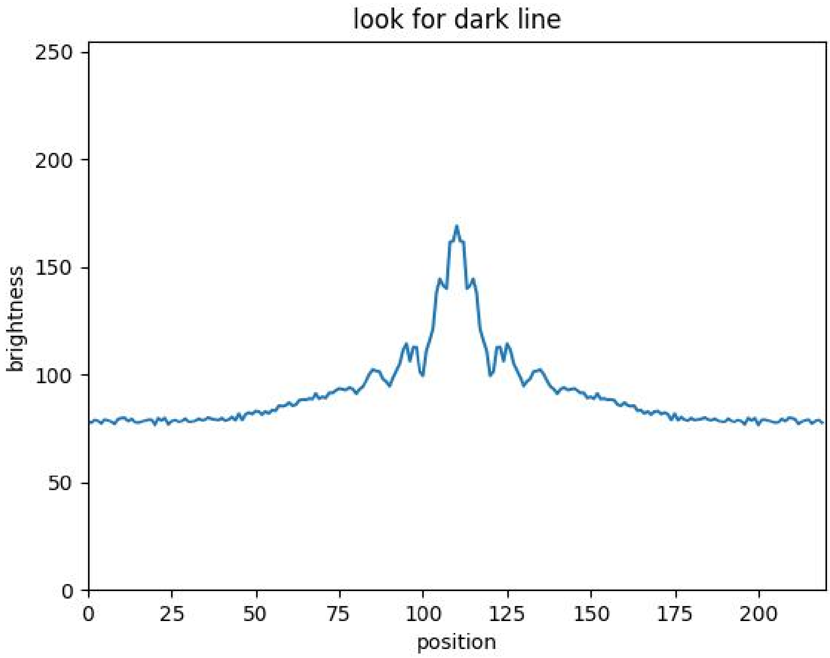
实验思考

如果退化过程的参数未知，如何进行复原？请编程实现，并与参数已知时的结果进行比较。

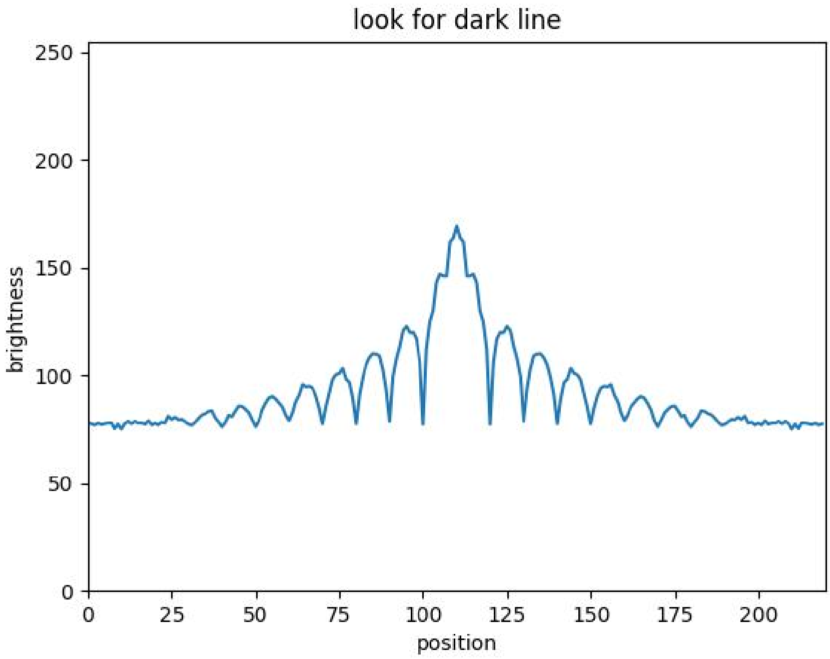
- 由于PSF是 $H(u,v) = \frac{T}{\pi(ua+vb)} \times sin[\pi(ua+vb)] \times e^{-j\pi(ua+vb)}$，是一个sinc函数，是一个显然有零点的函数，所以我们在运动模糊后的图像频谱上，看到许多暗线。在高斯噪声添加后，暗线变得不那么明显，我们可以用高斯卷积核(gaussian blur)减缓这种影响,如下图(无高斯噪声-> 有高斯噪声->高斯模糊)：
- 
- 理论推导：L与暗条纹距离之间的距离

  > 论。设图像有 N 行，对式(5)进行离散化，得到表达式：
  >
  > $$H(u) = \frac{\sin(\pi uL/N)}{\pi uL/N} \quad (6)$$
  >
  > 令 $H(u)=0$，则 $\sin(\pi uL/N)=0$，假设有2个频谱图上连续的零点 u_1，u_2，则满足 $\frac{\pi u_2 L}{N} - \frac{\pi u_1 L}{N} = \pi$，化简可得到 $u_2 - u_1 = \frac{N}{L}$，而 $(u_2 - u_1)$ 就是运动模糊图像频谱图中暗条纹之间的距离，设为 D，则得到式(7)。
  >
  > $$L = \frac{N}{D} \quad (7)$$

- 画出频谱列平均像素，观察其明暗变化，找到暗线所在的位置，根据低谷之间的距离确定D:



- 和没有添加高斯噪声的只有运动模糊的平均像素图对比，发现确立了同一个L:



- N = 216 , D = 20 , L = 10.8 ， 基本正确。
- 寻找暗线的代码

```
def show_line_in_plt(shape , spectrum , filename):
    size = spectrum.shape[0]
    sample = [ it.sum()/it.shape[0] for it in spectrum.T]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set(xlim=[0, shape[1]], ylim=[0, 255], title='look for dark line',
       ylabel='brightness', xlabel='position')
    x = np.arange(0, shape[1])
    y = sample
    ax.plot(x, y)
    plt.savefig(filename)
    plt.show()
```

实验感想

在本次实验中，遇到了不少问题，也查阅了不少资料，感觉很充实。特别关注了一些数学推导，动笔计算后感觉对一维的信号傅里叶变换对理解更透彻了。

本来就对数字图像处理感兴趣，现在确实体会到了信号与系统在图像处理方面的运用。虽然完成了实验，但是我们还会多方继续调研，了解运动模糊的相关信息，找到更好的deblur方法。我们也深深体会到，噪声就像图像的癌症，一个本来并不复杂的问题，加上噪声就会复杂起来，甚至完全无法下手。

参考资料

《数字图像处理》第三版 冈萨雷斯

《运动模糊图像PSF参数估计与图像复原研究》 廖秋香 卢在盛 彭金虎

https://blog.csdn.net/qq_42240908/article/details/112745898