西安交通大学

数字图像处理作业报告

作业 6：图像噪声和恢复

摘要：本次报告主要围绕图像的噪声污染与恢复、维纳滤波器的推导与应用两方面展开，详细推导了维纳滤波器的算法，介绍了通过 MATLAB 实现以下操作的具体过程：（1）图像不同方式的加噪及去噪；（2）图像的模糊处理与恢复。去噪与模糊恢复均采用多种算法实现，通过对比得出各算法的优缺点。

关键词：高斯噪声，椒盐噪声，反谐波均值滤波，维纳滤波，约束最小二乘方滤波，MATLAB

姓　名：胡　欣　盈

班　级：自　动　化　94

学　号：2194323176

提交日期：2022 年 3 月 31 日

目　录

一、基本概念及原理

（1）高斯噪声

高斯噪声是指它的概率密度函数服从高斯分布的一类噪声。如果一个噪声，它的幅度分布服从高斯分布，而它的功率谱密度又是均匀分布的，则称它为高斯白噪声。高斯白噪声的二阶矩不相关，一阶矩为常数，是指先后信号在时间上的相关性。其产生原因如下：

1、图像传感器在拍摄时市场不够明亮、亮度不够均匀；

2、电路各元器件自身噪声和相互影响；

3、图像传感器长期工作，温度过高。

其概率密度函数为：

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

（2）椒盐噪声

椒盐噪声又称脉冲噪声，它随机改变一些像素值，是由图像传感器，传输信道，解码处理等产生的黑白相间的亮暗点噪声，往往由图像切割引起。

椒盐噪声具体可以分为白盐噪声和胡椒噪声。白盐噪声又称白噪声，是在图像中添加一些随机的白色像素点(255)；胡椒噪声是在图像中添加一些随机的黑色像素点(0)；而通常说的椒盐噪声则是在图像中既有白色像素点，又有黑色像素点。

其概率密度函数为：

$$P(z) = \begin{cases} P_a & z = a \\ P_b & z = b \\ 1 - P_a - P_b & else \end{cases}$$

（3）反谐波均值滤波器

反谐波均值滤波器表达式如下所示：

$$\hat{f}(x,y)=\frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$$

其中 Q 称为滤波器的阶数。逆谐波均值滤波器能够减少或消除椒盐噪声对图像产生的影响。当 Q 大于 0 时，该滤波器消除胡椒噪声；当 Q 小于 0 时，该滤波器消除盐粒噪声。因此，该滤波器不能够同时消除胡椒噪声和盐粒噪声。特殊的，当 Q=0 时，逆谐波均值滤波器简化为算数均值滤波器，当 Q=-1 时，则简化为谐波均值滤波器。具体原理如下：

$$\hat{f}(x,y)=\frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}=\sum_{(s,t)\in S_{xy}}\frac{g(s,t)^{Q}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}g(s,t)$$

上式可以看作求(x, y)邻域内所有(s, t)点的加权平均值，权重的分母$\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}$是一个常数，因此，我们只需考虑分子$g(s,t)^{Q}$的大小。

当 Q>0 时，$g(s,t)^{Q}$对 $g(s,t)$有增强作用，由于“胡椒”噪声值为 0，对加权平均结果影响较小，所以滤波后噪声点处(x, y)取值和周围其他值更接近，有利于消除“胡椒”噪声。

当 Q<0 时，$g(s,t)^{Q}$对 $g(s,t)$有削弱作用，由于“白盐”噪声值为 255，取倒数后较小，对加权平均结果影响较小，所以滤波后噪声点处(x, y)取值和周围其他值更接近，有利于消除“白盐”噪声。

（4）维纳滤波器

维纳滤波也称为最小均方误差滤波，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，具体推导过程如下所示：

图像的退化模型为

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

其中 f(x,y)为原始图像，h(x,y)为退化函数，η(x, y)为噪声函数，g(x,y)为退化的图像。假设 f 与η不相关，η为 0 均值的平稳随机过程。根据已退化图像 g(x,y)利用线性估计器估计原始图像

$$\hat{f}(x,y)=w(x,y)*g(x,y)$$

其中$\hat{f}(x,y)$为恢复的图像，$w(x,y)$为恢复滤波器。

最小化均方误差为

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

最小化均方误差可以由正交原则求解，即：最优求解的误差与观测的信号值不相关

$$\mathrm{E[e(n_1,n_2)g^*(m_1,m_2)]=0}$$

进一步分解得

$$\begin{aligned}\mathrm{E[e(n_1,n_2)g^*(m_1,m_2)]} &= \mathrm{E[\hat{f}(x,y)g^*(m_1,m_2)]} \\ &= \mathrm{E[w(n_1,n_2)*g(n_1,n_2)g^*(m_1,m_2)]}\end{aligned}$$

记 $R_{fg}(\mathrm{n_1,n_2})=\mathrm{E}[f(\mathrm{k_1,k_2})\mathrm{g}^*(k_1-n_1,k_2-n_2)]$ (1)

$$R_g(n_1,n_2)=E[\mathrm{g}(k_1,k_2)g^*(k_1-n_1,k_2-n_2)] \quad (2)$$

$$R_{fg}(\mathrm{n_1-m_1,n_2-m_2})=\sum_{\mathrm{k_1}}\sum_{\mathrm{k_2}}\mathrm{w(k_1,k_2)\,R_g(n_1-k_1-m_1,n_2-k_2-m_2)}$$

$$=\mathrm{w(n_1-m_1,n_2-m_2)*R_g(n_1-m_1,n_2-m_2)}\qquad(3)$$

换元得

$$R_{fg}(n_1,n_2)=\mathrm{w(n_1,n_2)*R_g(n_1,n_2)}$$

等式两端同时取傅里叶变换得

$$\mathrm{W(\omega_1,\omega_2)}=\frac{\mathrm{R_{fg}(\omega_1,\omega_2)}}{\mathrm{R_g(\omega_1,\omega_2)}}\qquad(4)$$

为非因果维纳滤波器，其中

$$\mathrm{P_{fg}(\omega_1,\omega_2)}=\sum_{\mathrm{n_1}}\sum_{\mathrm{n_2}}\mathrm{R_{fg}(n_1,n_2)e^{-j(n_1\omega_1+n_2\omega_2)}}$$

是两个平稳随机过程的协功率谱，首先对(1)式进行化简得

$$R_{fg}(n_1,n_2)=E[f(k_1+\mathrm{n_1},k_2+\mathrm{n_2})g^*(k_1,k_2)]$$

$$=\mathrm{E[f(k_1+n_1,k_2+n_2)(h^*(k_1,k_2x,y)*f^*(k_1,k_2)+\eta^*(k_1,k_2))]}$$

$$=\mathrm{E}\left[\mathrm{f(k_1+n_1,k_2+n_2)}\left(\sum_{\mathrm{l_1}}\sum_{\mathrm{l_2}}\mathrm{h^*(l_1,l_2)f^*(k_1-l_1,k_2-l_2)+\eta^*(k_1,k_2)}\right)\right]$$

$$=\sum_{\mathrm{l_1}}\sum_{\mathrm{l_2}}\mathrm{h^*(l_1,l_2)\,E[f(k_1+n_1,k_2+n_2)(f^*(k_1-l_1,k_2-l_2)+\eta^*(k_1,k_2))]}$$

$$=\mathrm{h^*(-n_1,-n_2)}*R_f(n_1,n_2)$$

两端同时取傅里叶变换得

$$\mathrm{P_{fg}(\omega_1,\omega_2)=H^*(\omega_1,\omega_2)P_f(\omega_1,\omega_2)}\qquad(5)$$

对(2)式进行化简得

$$\mathrm{R_g(n_1,n_2)=E[g(k_1+n_1,k_2+n_2)g^*(k_1,k_2)]}$$

$$= \sum_{m_1}\sum_{m_2}\sum_{l_1}\sum_{l_2} h(m_1,m_2)h^*(l_1,l_2)f(k_1+n_1-m_1,k_2+n_2-m_2)f^*(k_1 - l_1,k_2-l_2)$$

$$+ \sum_{m_1}\sum_{m_2} h(m_1,m_2)f(k_1+n_1-m_1,k_2+n_2-m_2)\eta^*(k_1,k_2)$$

$$+ \sum_{l_1}\sum_{l_2} h^*(l_1,l_2)f^*(k_1-l_1,k_2-l_2)\eta(k_1+n_1,k_2+n_2) + R_\eta(n_1,n_2)$$

$$= \sum_{m_1}\sum_{m_2}\sum_{l_1}\sum_{l_2} h(m_1,m_2)h^*(l_1,l_2)R_f(l_1+n_1-m_1,l_2+n_2-m_2)$$

$$+ R_\eta(n_1,n_2)$$

$$= \sum_{l_1}\sum_{l_2} h^*(l_1,l_2)h(l_1+n_1,l_2+n_2) * R_f(l_1+n_1,l_2+n_2) + R_\eta(n_1,n_2)$$

$$= h^*(-n_1,-n_2) * h(n_1,n_2) * R_f(n_1,n_2) + R_\eta(n_1,n_2)$$

两端同时取傅里叶变换

$$P_g(\omega_1,\omega_2) = |H(\omega_1,\omega_2)|^2 P_f(\omega_1,\omega_2) + P_\eta(\omega_1,\omega_2) \qquad (6)$$

将（5）（6）代入（4）得

$$W(\omega_1,\omega_2) = \frac{H^*(\omega_1,\omega_2)P_f(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2 P_f(\omega_1,\omega_2) + P_\eta(\omega_1,\omega_2)}$$

$$W(\omega_1,\omega_2) = \frac{H^*(\omega_1,\omega_2)|F(\omega_1,\omega_2)|^2}{|H(\omega_1,\omega_2)|^2|F(\omega_1,\omega_2)|^2 + |N(\omega_1,\omega_2)|^2}$$

故表达式由下式给出

$$\hat{F}(u,v) = W(u,v)G(u,v)$$

$$= \frac{H^*(u,v)S_f(u,v)}{|H(u,v)|^2 S_f(u,v) + S_\eta(u,v)} G(u,v)$$

$$= \frac{1}{H(u,v)} \frac{S_h(u,v)}{S_h(u,v) + \frac{S_\eta(u,v)}{S_f(u,v)}} G(u,v)$$

（5）约束最小二乘方滤波

对于约束最小二乘方滤波，期望是找一个最小准则函数 C，定义如下：

$$C=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\nabla^2 f(x,y)\right]^2$$

$$\left\|g-H\hat{f}\right\|^2=\|\eta\|^2$$

$$\|\omega\|^2=\omega^T\omega$$

约束最小二乘方滤波的计算公式如下

$$\hat{F}(u,v)=\frac{H^*(u,v)}{|H(u,v)|^2+\frac{1}{\lambda}|C|^2}G(u,v)$$
$$=\frac{H^*(u,v)}{|H(u,v)|^2+\frac{1}{\gamma}|P(u,v)|^2}G(u,v)$$

其中，γ是一个参数，必须对它进行调整以满足条件，$P(u,v)$是函数

$$P(x,y)=\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix}$$

二、高斯噪声的产生与恢复

题目要求在测试图像上产生高斯噪声（需能指定噪声的均值和方差），用多种滤波器恢复图像并分析其优缺点。

在 MATLAB 中，首先用 imread()将图像读入，根据前述原理构建函数 GaussianNoise(Img,av,std)以产生高斯噪声，其中 Img 为输入图像，av 为产生噪声的均值，std 为产生噪声的方差，再构建高斯滤波器 GaussianFilter(Img,masksize,sigma) 与中值滤波器

MedianFilter(Img,masksize)，这两种滤波器在作业 4 中已有详细说明，故而在此不再赘述。

添加高斯噪声后的图像如图 2-1 所示，通过观察可以发现，高斯噪声的效果与均值、方差两个参数密切相关，方差越大时越模糊，颗粒感更大、更重，均值越大时，图像越亮、越偏白。

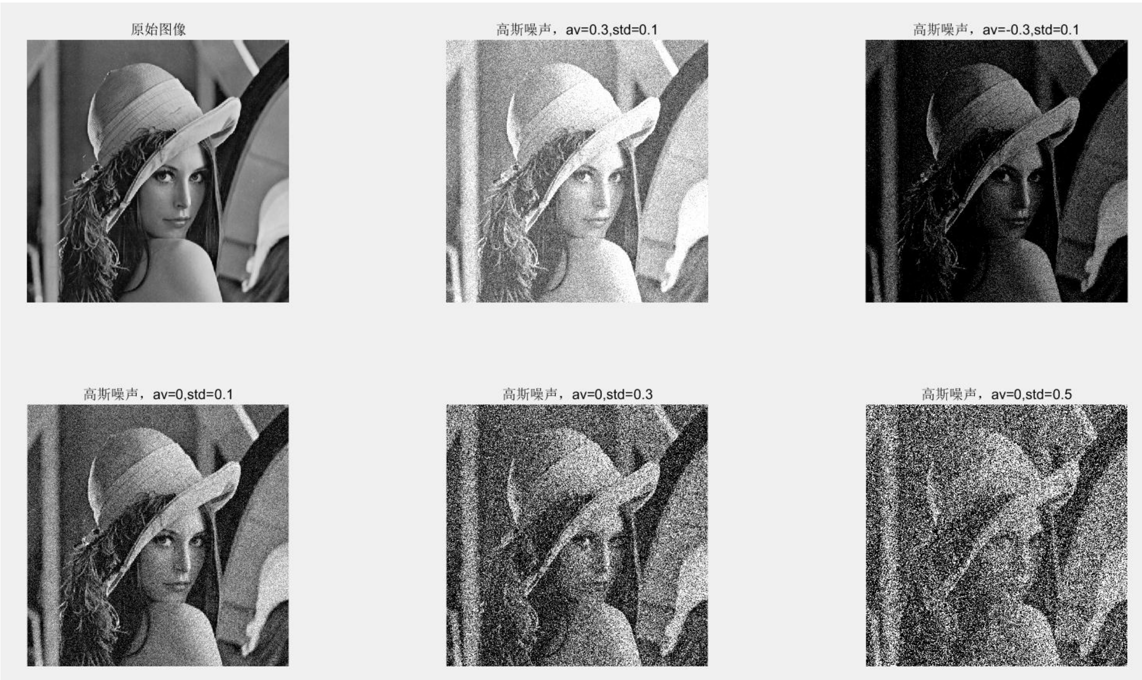

图 2-1 经过不同均值、方差高斯噪声处理的图像

经过模板大小为 5x5 的高斯滤波（sigma=1.5）与中值滤波处理后的图片如图 2-2 所示，可以看出，两种滤波方式的恢复效果基本相当，高斯滤波略优一些，得到的图像稍微光滑一些。另外，中值滤波得到的图像亮度要稍高于高斯滤波得到的图像。

图 2-2 滤波处理前后的图像

三、椒盐噪声的产生与恢复

本题要求在测试图像中加入椒盐噪声(椒和盐噪声密度均是 0.1)；并用学过的滤波器恢复图像，再使用反谐波分析 Q 大于 0 和小于 0 的作用。

在 MATLAB 中，根据前述原理构建函数 SaltPepperNoise(Img,a,b)以产生椒盐噪声，其中 Img 为输入图像，a、b 分别对应“胡椒噪声”与“白盐噪声”；再构建反谐波均值滤波函数 Contraharmonic(Img,Q)。

添加椒盐噪声后的图像如图 3-1 所示。经过模板大小为 5x5 的高斯滤波（sigma=1.5）与中值滤波处理后的图片如图 3-2 所示，结论与处理高斯噪声的结论基本一致。

图 3-1 椒盐噪声处理后的图像

图 3-2 滤波处理前后的图像

使用反谐波函数 Contraharmonic(Img,Q)处理受到椒盐噪声污染的图像，处理结果如图 3-3 所示。可以看出，Q 为正数时，对胡椒噪声有较好的恢复效果，而对白盐噪声则恢复效果很差，几乎将整幅图

像处理成白色。当Q为负数时，对白盐噪声有很好的恢复效果，而对胡椒噪声的恢复效果很差，几乎将整幅图像处理成黑色。这与前述原理也是相符合的。

图 3-3 反谐波均值滤波前后的图像

四、模糊处理与恢复

本题要求推导维纳滤波器并构建模糊滤波器模糊 lena 图像（45 度方向，T=1）；再在模糊处理后的 lena 图像中增加高斯噪声（均值=0，方差=10 pixels）以产生模糊图像；最后分别利用维纳滤波与约束最小二乘方滤波，恢复图像，并分析算法的优缺点。

维纳滤波器的具体推导详见第一部分的原理。

根据教材中给出的计算运动模糊时所需的退化函数 $H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)}$ 可构建函数 Motion(image,a,b,T) 实现模糊处理，再根据前述原理可构建函数 Weina(image, K)、Zuixiao(image,gamma,noise_var) 以实现图像恢复。

处理结果如图 4-1 所示，维纳滤波器中参数 K 取值为 0.03，约束最小二乘方滤波中参数 gamma 值为 0.001。由图可知，约束最小二乘方滤波得到的图像较维纳滤波器得到的图像更为清晰。而维纳滤波器得到的图像中并不能消除残留的拖影，相比之下约束最小二乘方滤波的得到的图像对拖影的去除较好。

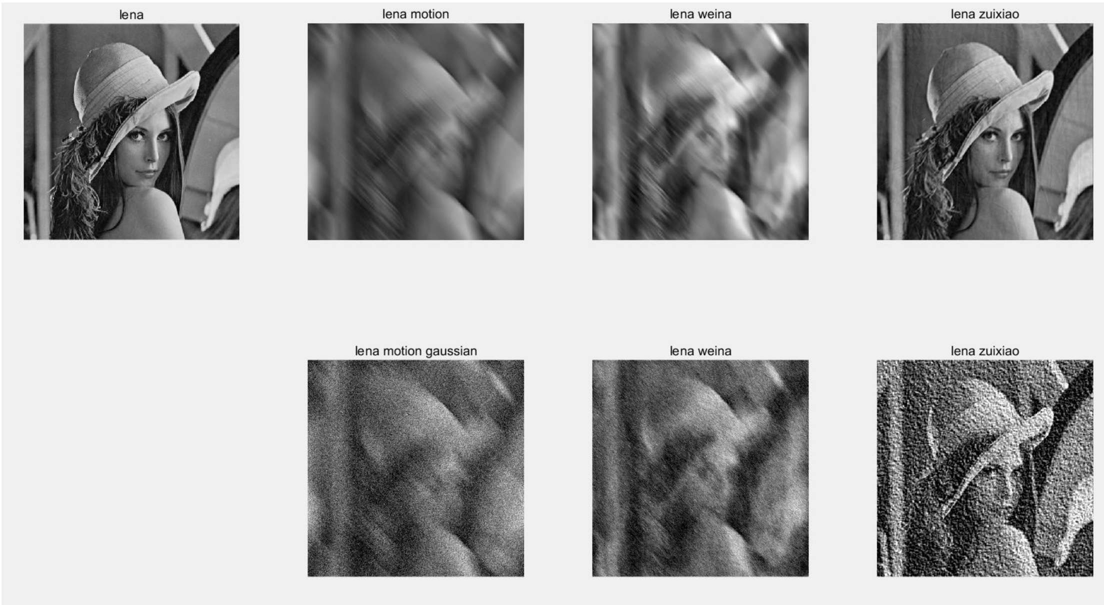

图 4-1 模糊滤波与恢复处理前后的图像

附录

附录 1：参考文献

[1]阮秋琦,阮宇智等.数字图像处理(第三版)(美)冈萨雷斯[M]，2003,电子工业出版社

[2]门敬文•数字图像处理 MATLAB 版[M].2007.2,国防工业出版社.

[3]数字图像处理——虚宇宸轩-CSDN 博客

附录 2：源代码

（1）噪声添加与恢复

```
clc
clear

lena=imread('C:\Users\LENOVO\Desktop\ImageProcessHomework\Homework6\lena.bmp');

%task1
lena_g1=GaussianNoise(lena,0,0.1)
lena_g2=GaussianNoise(lena,0,0.3)
lena_g3=GaussianNoise(lena,0,0.5)
lena_g4=GaussianNoise(lena,0.3,0.1)
lena_g5=GaussianNoise(lena,-0.3,0.1)

figure
subplot(2,3,1);imshow(lena);title('原始图像');
subplot(2,3,4);imshow(lena_g1);title('高斯噪声，av=0,std=0.1');
subplot(2,3,5);imshow(lena_g2);title('高斯噪声，av=0,std=0.3');
subplot(2,3,6);imshow(lena_g3);title('高斯噪声，av=0,std=0.5');
subplot(2,3,2);imshow(lena_g4);title('高斯噪声，av=0.3,std=0.1');
subplot(2,3,3);imshow(lena_g5);title('高斯噪声，av=-0.3,std=0.1');

lena_gg=GaussianFilter(lena_g1,5,1.5)
lena_gm=MedianFilter(lena_g1,5)

figure
subplot(2,2,1);imshow(lena);title('原始图像');
subplot(2,2,2);imshow(lena_g1);title('高斯噪声，av=0,std=0.1');
subplot(2,2,3);imshow(lena_gg);title('高斯滤波，5*5，sigma=1.5');
subplot(2,2,4);imshow(lena_gm);title('中值滤波，5*5');

%task2
lena_s1=SaltPepperNoise(lena,0,0.1)
lena_s2=SaltPepperNoise(lena,0.1,0)
lena_s3=SaltPepperNoise(lena,0.1,0.1)

figure
subplot(2,2,1);imshow(lena);title('原始图像');
subplot(2,2,2);imshow(lena_s1);title('椒盐噪声，a=0,b=0.1');
subplot(2,2,3);imshow(lena_s2);title('椒盐噪声，a=0.1,b=0');
subplot(2,2,4);imshow(lena_s3);title('椒盐噪声，a=0.1,b=0.1');

```

```
lena_sg=GaussianFilter(lena_s3,5,1.5)
lena_sm=MedianFilter(lena_s3,5)

figure
subplot(2,2,1);imshow(lena);title('原始图像');
subplot(2,2,2);imshow(lena_s3);title('椒盐噪声，a=0.1，b=0.1');
subplot(2,2,3);imshow(lena_sg);title('高斯滤波，5*5，sigma=1.5');
subplot(2,2,4);imshow(lena_sm);title('中值滤波，5*5');

lena_wc1=Contraharmonic(lena_s1,1)
lena_wc2=Contraharmonic(lena_s1,-1)
lena_bc1=Contraharmonic(lena_s2,1)
lena_bc2=Contraharmonic(lena_s2,-1)
lena_sc1=Contraharmonic(lena_s3,1)
lena_sc2=Contraharmonic(lena_s3,-1)
figure
subplot(3,3,1);imshow(lena_s1);title('白盐噪声');
subplot(3,3,2);imshow(lena_wc1);title('白盐噪声，Q=1');
subplot(3,3,3);imshow(lena_wc2);title('白盐噪声，Q=-1');
subplot(3,3,4);imshow(lena_s2);title('胡椒噪声');
subplot(3,3,5);imshow(lena_bc1);title('胡椒噪声，Q=1');
subplot(3,3,6);imshow(lena_bc2);title('胡椒噪声，Q=-1');
subplot(3,3,7);imshow(lena_s3);title('椒盐噪声');
subplot(3,3,8);imshow(lena_sc1);title('椒盐噪声，Q=1');
subplot(3,3,9);imshow(lena_sc2);title('椒盐噪声，Q=-1');

%加高斯噪声：
function Img_out=GaussianNoise(Img,av,std)
[M,N]=size(Img);
u1=rand(M,N);   u2=rand(M,N);
x=std*sqrt(-2*log(u1)).*cos(2*pi*u2)+av;
Img_out=uint8(255*(double(Img)/255+x));
end

%加椒盐噪声：
function Img_out=SaltPepperNoise(Img,a,b)
[M,N]=size(Img);
x=rand(M,N);
Img_out=Img;
Img_out(find(x<=a))=0;
Img_out(find(x>a&x<(a+b)))=255;
end

% 高斯滤波：
```

```
function Img_out=GaussianFilter(Img,masksize,sigma)
for i=1:masksize
    for j=1:masksize
        x=i-ceil(masksize/2);
        y=j-ceil(masksize/2);
        h(i,j)=exp(-(x^2+y^2)/(2*sigma^2))/(2*pi*sigma^2);
    end
end
Img_out=uint8(conv2(Img,h,'same'));
end

% 中值滤波：
function Img_out=MedianFilter(Img,masksize)
exsize=floor(masksize/2);   %各方向扩展大小
Imgex=padarray(Img,[exsize,exsize],'replicate','both'); %扩展图片
[m,n]=size(Img);
Img_out=Img;    %将 Img_out 准备为和 Img 相同的 size
for i=1:m
    for j=1:n
        neighbor=Imgex(i:i+masksize-1,j:j+masksize-1);  %截取邻域
        Img_out(i,j)=median(neighbor(:));   %中值滤波
    end
end
end

%反谐波均值滤波：
function Img_out=Contraharmonic(Img,Q)
[M,N]=size(Img);
ImgSize=3;   ImgSize=(ImgSize-1)/2;
Img_out=Img;
for x=1+ImgSize:1:M-ImgSize
    for y=1+ImgSize:1:M-ImgSize
        is=Img(x-ImgSize:1:x+ImgSize,y-ImgSize:1:y+ImgSize);
        Img_out(x,y)=sum(double(is(:)).^(Q+1))/sum(double(is(:)).^(Q));
    end
end
end
```

（2）模糊处理与恢复

```
clc
clear

```

```
lena=imread('C:\Users\LENOVO\Desktop\ImageProcessHomework\Homework6\lena.bmp');
%task3

figure(1);
subplot(241);
imshow(uint8(lena)); title('lena');
subplot(242);
[~, ~, lena_m1] = Motion(lena, 0.1, 0.1, 1);
imshow(lena_m1); title('lena motion');
subplot(246);
lena_m2 = imnoise(lena_m1,'gaussian',0,0.01);
imshow(uint8(lena_m2)); title('lena motion gaussian');

subplot(243);
[~, ~, image_IF] = Weina(lena_m1, 0.03);
imshow(image_IF); title('lena weina');
subplot(244);
[image_IF] = Zuixiao(lena, 0.001, 0);
imshow(uint8(image_IF)); title('lena zuixiao');
subplot(247);
[~, ~, image_IF] = Weina(lena_m2, 0.03);
imshow(image_IF); title('lena weina');
subplot(248);
[image_IF] = Zuixiao(lena, 0.001, 0.001);
imshow(uint8(image_IF)); title('lena zuixiao');

%% hw6 Helper functions
%

% Motion function
function [image_F, image_G, image_IF] = Motion(image, a, b, T)
    image_F = fftshift(fft2(double(image)));
    [P, Q] = size(image_F);
    image_H = zeros(P,Q); image_G = zeros(P,Q);
    for u = 1:P
        for v = 1:Q
            x = u - P / 2;
            y = v - Q / 2;
            temp = pi * (x * a + y * b);
            if temp == 0
                temp = 1;
            end
            image_H(u,v) = (T / temp) * sin(temp) * exp(- temp * sqrt(-1));
            image_G(u,v) = image_H(u,v)*image_F(u,v);
```

```
            end
        end
        image_IF = uint8(abs(ifft2(ifftshift(image_G))));
end

% Weina function
function [image_F, image_G, image_IF] = Weina(image, K)
        image_F = fftshift(fft2(double(image)));
        [P, Q] = size(image_F);
        a = 0.1; b = 0.1; T = 1;
        image_H = zeros(P,Q); image_G = zeros(P,Q);
        for u = 1:P
            for v = 1:Q
                x = u - P / 2;
                y = v - Q / 2;
                temp = pi * (x * a + y * b);
                if temp == 0
                    temp = 1;
                end
                image_H(u,v) = (T / temp) * sin(temp) * exp(- temp * sqrt(-1));
            end
        end
        for u = 1:P
            for v = 1:Q
                image_G(u,v)=(1/image_H(u,v)*(abs(image_H(u,v)))^2/((abs(image_H(u,v)))^2+K))*image_F(u,v);
            end
        end
        image_If = ifft2(ifftshift(image_G));
        image_IF = 256.*image_If./max(max(image_If));
        image_IF = uint8(real(image_IF));
end

% Zuixiao function
function [image_IF] = Zuixiao(image, gamma, noise_var)
        [hei,wid,~] = size(image);
        % Simulate a motion blur.
        LEN = 50;
        THETA = 45;
        PSF = fspecial('motion', LEN, THETA);
        blurred = imfilter(image, PSF, 'conv', 'circular');
        % Inverse filter
        Pf = psf2otf(PSF,[hei,wid]);
        % Simulate additive noise.
```

```
    noise_mean = 0;
    blurred_noisy = imnoise(blurred, 'gaussian', ...
                        noise_mean, noise_var);
    % Try restoration using  Home Made Constrained Least Squares Filtering.
    p = [0 -1 0;-1 4 -1;0 -1 0];
    P = psf2otf(p,[hei,wid]);
    If = fft2(blurred_noisy);
    numerator = conj(Pf);
    denominator = Pf.^2 + gamma*(P.^2);
    image_IF = ifft2( numerator.*If./ denominator );

%% hw4 Helper functions
%

% Helper function
function image_C=Conv(image, maskSize, Filter)
    [image_M, image_N] = size(image);
    image_extend = wextend('2D','zpd',image,floor(maskSize/2));
    image_C = zeros([image_M, image_N]);
    for i = 1:maskSize
        for j = 1:maskSize
            image_C = image_C + double(image_extend(i:(image_M + i - 1), j:(image_N + j - 1))) * Filter(i, j);
        end
    end
end

% Median Filter by maskSize
function image_M = Median(image, maskSize)
    MFM = ones(maskSize) / maskSize^2;
    image_M = Conv(image, maskSize, MFM);
end

% Gaussian Filter by maskSize
function image_G=Gaussian(image, maskSize)
    GFM = zeros(maskSize);
    sigma = 1.5;
    for i = 1:maskSize
        for j = 1:maskSize
            x = i - floor(maskSize/2) - 1;
            y = j - floor(maskSize/2) - 1;
            GFM(i,j) = exp(-(x^2+y^2)/(2*sigma^2))/(2*pi*sigma^2);
        end
    end
```

```
    GFM = GFM / sum(sum(GFM));
    image_G = Conv(image, maskSize, GFM);
end

% Unsharp Filter by maskSize
function image_U = Unsharp(image)
    image_blur = Gaussian(image, 3);
    image_unsharp_mask = double(image) - image_blur;
    image_sharpened = double(image) + image_unsharp_mask;
    image_U = image_sharpened;
end

% Sobel Filter by maskSize
function image_S = Sobel(image)
    SFMx = [1,0,-1;2,0,-2;1,0,-1];
    SFMy = [1,2,1;0,0,0;-1,-2,-1];
    image_Sx = Conv(image, 3, SFMx);
    image_Sy = Conv(image, 3, SFMy);
    image_S = abs(image_Sx) + abs(image_Sy);
end

% Laplace Filter by maskSize
function image_L = Laplace(image)
    LFM = [1,1,1;1,-8,1;1,1,1];
    image_L = Conv(image, 3, LFM);
end

% Canny Filter by maskSize
function image_C = Canny(image)
    image_C = edge(image, 'canny');
end

%% hw5 Helper functions
%

% Helper function
function [image_F, image_G, image_IF, image_r] = BLPF(image, bn, D0)
    image_F = fftshift(fft2(double(image)));
    [P, Q] = size(image_F);
    image_Ft = 0; image_Gt = 0;
    image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
    for u = 1:P
        for v = 1:Q
            image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
```

```
            image_H(u,v) = 1/(1+(image_D(u,v)/D0)^(2*bn));
            image_G(u,v) = image_H(u,v)*image_F(u,v);
            image_Fd = (abs(image_F(u,v)))^2;
            image_Ft = image_Ft+image_Fd;
            image_Gd = (abs(image_G(u,v)))^2;
            image_Gt = image_Gt+image_Gd;
        end
    end
    image_IF = uint8(real(ifft2(ifftshift(image_G))));
    image_r = image_Gt/image_Ft;
end

% GLPF function
function [image_F, image_G, image_IF, image_r] = GLPF(image, D0)
    image_F = fftshift(fft2(double(image)));
    [P, Q] = size(image_F);
    image_Ft = 0; image_Gt = 0;
    image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
    for u = 1:P
        for v = 1:Q
            image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
            image_H(u,v) = exp(-image_D(u,v)^2/(2*D0^2));
            image_G(u,v) = image_H(u,v)*image_F(u,v);
            image_Fd = (abs(image_F(u,v)))^2;
            image_Ft = image_Ft+image_Fd;
            image_Gd = (abs(image_G(u,v)))^2;
            image_Gt = image_Gt+image_Gd;
        end
    end
    image_IF = uint8(real(ifft2(ifftshift(image_G))));
    image_r = image_Gt/image_Ft;
end

% BHPF function
function [image_F, image_G, image_IF, image_r] = BHPF(image, bn, D0)
    image_F = fftshift(fft2(double(image)));
    [P, Q] = size(image_F);
    image_Ft = 0; image_Gt = 0;
    image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
    for u = 1:P
        for v = 1:Q
            image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
            image_H(u,v) = 1/(1+(D0/image_D(u,v))^(2*bn));
            image_G(u,v) = image_H(u,v)*image_F(u,v);
```

```
                image_Fd = (abs(image_F(u,v)))^2;
                image_Ft = image_Ft+image_Fd;
                image_Gd = (abs(image_G(u,v)))^2;
                image_Gt = image_Gt+image_Gd;
            end
        end
        image_IF = uint8(real(ifft2(ifftshift(image_G))));
        image_r = image_Gt/image_Ft;
end

% GHPF function
function [image_F, image_G, image_IF, image_r] = GHPF(image, D0)
        image_F = fftshift(fft2(double(image)));
        [P, Q] = size(image_F);
        image_Ft = 0; image_Gt = 0;
        image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
        for u = 1:P
            for v = 1:Q
                image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
                image_H(u,v) = 1-exp(-image_D(u,v)^2/(2*D0^2));
                image_G(u,v) = image_H(u,v)*image_F(u,v);
                image_Fd = (abs(image_F(u,v)))^2;
                image_Ft = image_Ft+image_Fd;
                image_Gd = (abs(image_G(u,v)))^2;
                image_Gt = image_Gt+image_Gd;
            end
        end
        image_IF = uint8(real(ifft2(ifftshift(image_G))));
        image_r = image_Gt/image_Ft;
end

% LHPF function
function [image_F, image_G, image_IF, image_r] = LHPF(image, c)
        image_F = fftshift(fft2(double(image)));
        [P, Q] = size(image_F);
        image_Ft = 0; image_Gt = 0;
        image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
        for u = 1:P
            for v = 1:Q
                image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
                image_H(u,v) = 1+c*4*pi^2*image_D(u,v)^2;
                image_G(u,v) = image_H(u,v)*image_F(u,v);
                image_Fd = (abs(image_F(u,v)))^2;
                image_Ft = image_Ft+image_Fd;
```

```
            image_Gd = (abs(image_G(u,v)))^2;
            image_Gt = image_Gt+image_Gd;
        end
    end
    image_IF = uint8(real(ifft2(ifftshift(image_G))));
    image_r = image_Gt/image_Ft;
end

% UHPF function
function [image_F, image_G, image_IF, image_r] = UHPF(image, k1, k2, D0)
    image_F = fftshift(fft2(double(image)));
    [P, Q] = size(image_F);
    image_Ft = 0; image_Gt = 0;
    image_D = zeros(P,Q); image_H = zeros(P,Q); image_G = zeros(P,Q);
    for u = 1:P
        for v = 1:Q
            image_D(u,v) = sqrt((u-fix(P/2))^2+(v-fix(Q/2))^2);
            image_H(u,v) = 1-exp(-image_D(u,v)^2/(2*D0^2));
            image_G(u,v) = (k1+k2*image_H(u,v))*image_F(u,v);
            image_Fd = (abs(image_F(u,v)))^2;
            image_Ft = image_Ft+image_Fd;
            image_Gd = (abs(image_G(u,v)))^2;
            image_Gt = image_Gt+image_Gd;
        end
    end
    image_IF = uint8(real(ifft2(ifftshift(image_G))));
    image_r = image_Gt/image_Ft;
end
```