2020

数字图像处理第四次报告

学生姓名：任泽华

班级：自动化 71

学号：2171411498

提交日期：2020-3-23

摘要 ：

本报告主要工作：自己编程实现了低通滤波，包括 gauss 滤波和中值滤波，并分析了二者的优缺点；自己编程实现了 unsharp masking、sobel、Laplace、和 canny 三种方法的高通滤波；在 canny 算法中比较了自己编程实现的内容与系统函数的差别，实现了二级像素判断的深度优先搜索算法；比较了几种不同的方法在边界检查和图像增强方面的优劣。

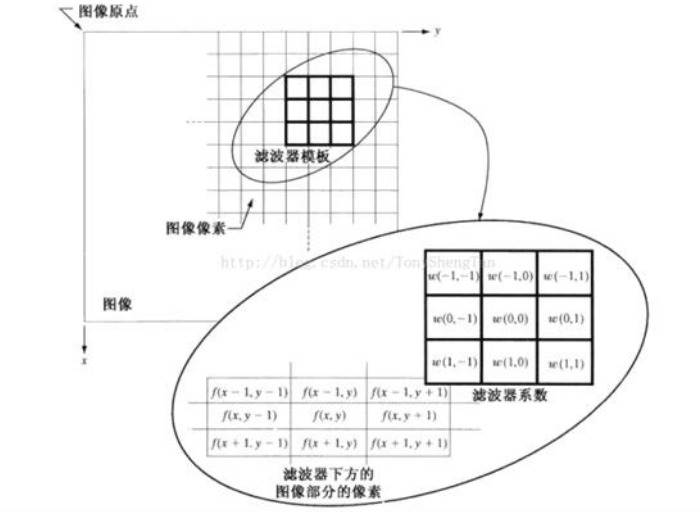
本报告软件运行环境为 MATLAB R2018b，所有代码均为自己编写，在编写过程中主要参考了 CSDN 相关帖子、两篇外文文献与博客园相关博客。（参考文献）

一、 空域低通滤波器：分别用高斯滤波器和中值滤波器去平滑测试图像 test1 和 2，模板大小分别是 3x3 ， 5x5 ，7x7； 分析各自优缺点；

——利用固定方差 sigma=1.5 产生高斯滤波器. 附件有产生高斯滤波器的方法； 分析各自优缺点；

1. 滤波原理

空域滤波说白了就是一个邻域运算，通过规定每个像素点的邻域范围，对邻域像素进行一定的计算得到该点变换后的值，这就是空域滤波的原理。通常的邻域包括上下左右，广义的邻域还包括四个角；再扩展开来说，邻域可以是 3x3 ， 5x5 ，7x7 等等更大范围的邻域。如果输出是这些点的线性函数，那么这种滤波称为线性滤波（均值滤波、高斯滤波），否则都叫做非线性滤波（中值滤波、边缘保持）。



使用大小为 $m \times n$ 的滤波器对大小为 $M \times N$ 的图像进行线性空间滤波，线性计算如下：

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

说白了就是利用模板的值与对于空域区域点乘求和。

（1）高斯滤波

对于高斯滤波，我们在空域主要采用高斯核来实现，首先是课本上提到的二维高斯分布函数：

$$p(x1, x2) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x_1^2 + x_2^2}{2\sigma^2}\right)$$

而高斯核（二位空域高斯滤波的模板）就是基于这个公式实现的：
比如说一个 3*3 模板大小的高斯滤波器

$$\frac{1}{2\pi\sigma^2} \begin{bmatrix} \exp\left(-\frac{2}{2\sigma^2}\right) & \exp\left(-\frac{1}{2\sigma^2}\right) & \exp\left(-\frac{2}{2\sigma^2}\right) \\ \exp\left(-\frac{1}{2\sigma^2}\right) & 1 & \exp\left(-\frac{1}{2\sigma^2}\right) \\ \exp\left(-\frac{2}{2\sigma^2}\right) & \exp\left(-\frac{1}{2\sigma^2}\right) & \exp\left(-\frac{2}{2\sigma^2}\right) \end{bmatrix}$$

但是这样的滤波器有一个问题，就是计算结果与原始像素数量级有差别，原因是系数没有归一化，所以在考虑计算结果和原图像数量级相同的前提下要对这个滤波器进行归一化处理。这样就可以把原来的计算进一步变成以高斯模板为系数的加权平均操作。通过改变 σ 的值可以改变滤波器的性质。

（2）中值滤波

中值滤波相比高斯滤波就显得比较简单了，其核心原理就是求出这个模板范围内的中位数，用它来代替中点的像素值，这样也可以除去许多变化大的点，起到整体的平滑效果。
公式为：

$$g(x,y) = median\{g(s,t)\}，(s,t) \in S_{xy}$$

2. 高斯滤波

（1）实现算法

使用 matlab 自己编写了 gauss 函数，输入为对应的计算区域进而 σ 值，输出为该点滤波后的值。保存为 gauss.m 函数。

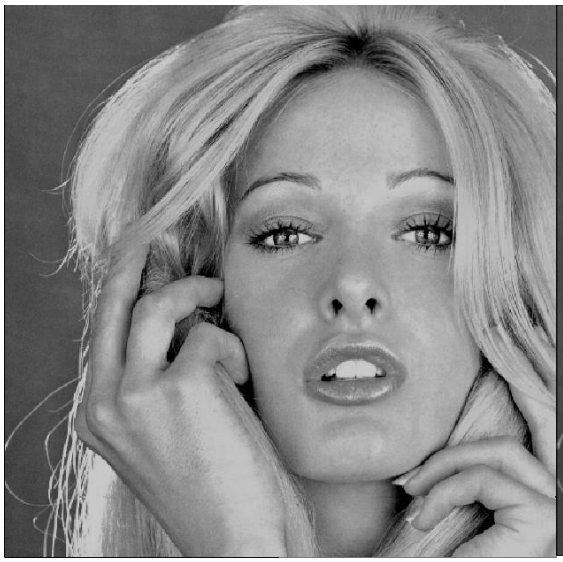
（2）效果图（σ =1.5）

原图

3×3

5×5

7×7



原图

3×3

5×5

7×7

3. 中值滤波

（1）实现算法

主函数采用和 gauss 相同的内容，仅吧子函数改为 media，即输入一个空域范围，返回这个范围内的中位数作为中点的像素值。

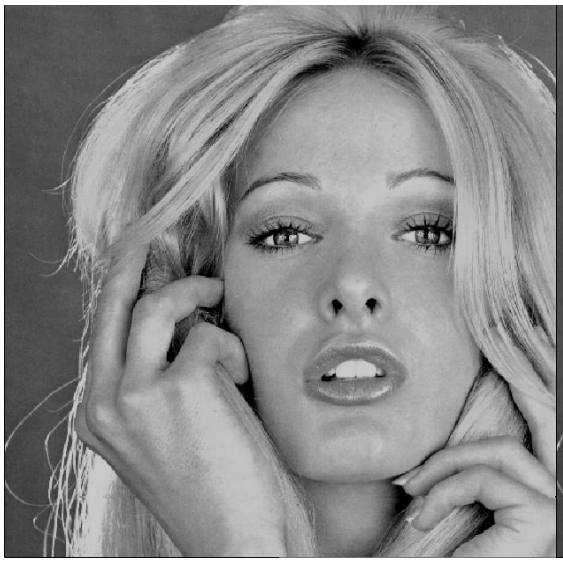
（2）效果图

原图

3×3

5×5

7×7



原图

3×3

5×5

7×7

4. 分析优缺点

（1）高斯与中值

高斯滤波器是线性平滑滤波器，适用于消除高斯噪声，所以使用高斯滤波器后，图像整体颜色对比度等等不会发生改变；而中值滤波器是非线性滤波器，主要功能是使拥有不同灰度的点更接近于它的相邻点，反而会使得图片对比度发生变化。变得更加失真。

下图分别是 7×7 模板处理 test1 的结果，可以很明显地看出区别：


高斯滤波


中值滤波

（2）不同的模板大小

从 2.3.问的分析中可以看出，模板越大，经过处理的数据越多，中点像素也就越接近与它周围的像素。所以说模板越大，图像的模糊效果越好。

（3）不同 σ 大小

比较 7×7 模板下不同 σ 大小对图片的影响。可以看出，随着 σ 的增大，图片变得清晰，原因是 σ 增大时，高频信息可以更多地通过，所以图片保留了更多的细节信息，显得更加清晰。

原图　　σ =0.5

σ =1.5　　σ =2.5

二、　利用高通滤波器滤波测试图像

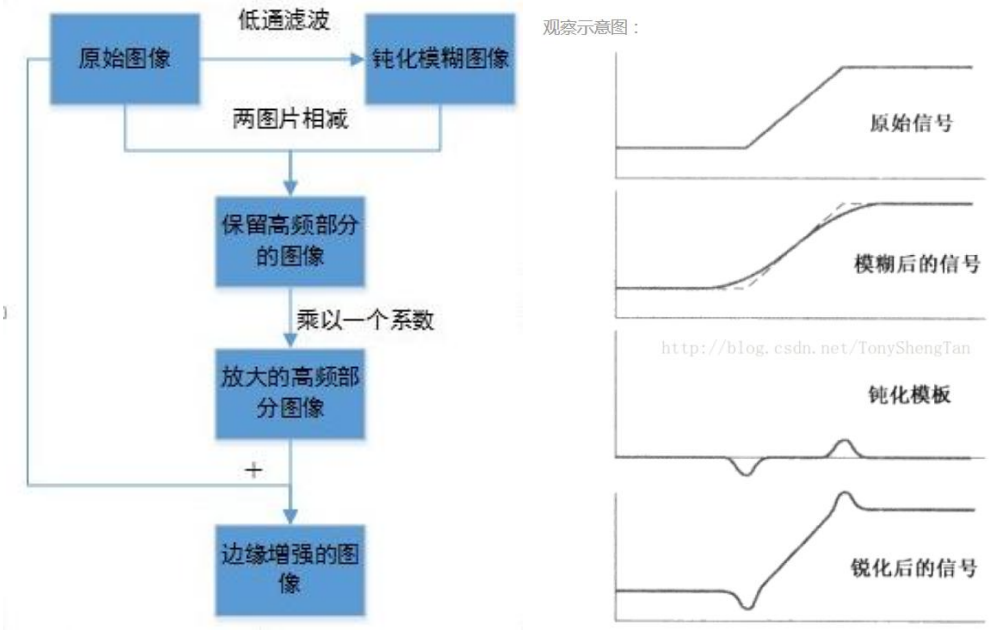
test3,4：

——包括 unsharp masking, Sobel edge detector, and Laplace edge detection；Canny algorithm.分析各自优缺点；

1. 滤波原理

（1）unsharp masking

线性反锐化掩模（UnSharp Masking，UM）算法。首先将原图像低通滤波后产生一个钝化模糊图像，将原图像与这模糊图像相减得到保留高频成份的图像，再将高频图像用一个参数放大后与原图像叠加，这就产生一个增强了边缘的图像。



$\bar{f}(x,y)$）表示模糊图像：$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y)$

加上一个权重部分：$g(x,y) = f(x,y) - k * g_{mask}(x,y)$

（2）Sobel edge detector

Sobel 是一阶导数的边缘检测算子，在算法实现过程中，通过 3×3 模板作为核与图像中的每个像素点做卷积和运算，然后选取合适的阈值以提取边缘。

其中 x 方向和 y 方向分别计算，完成后再将其总体计算。

$$S_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\quad S_y=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$$

∇f *的幅值表示为：* $M(x,y)=mag(\nabla f)=\sqrt{G_x+G_x}$

∇f *的方向表示为：* $\theta=\arctan(G_y/G_x)$

可以近似表示为：$M(x,y)\approx$

$$|(z_7+2z_8+z_9)-(z_1+2z_2+z_3)|+|(z_3+2z_6+z_9)-(z_1+2z_4+z_7)|$$

（3）Laplace edge detection

拉普拉斯算子是二阶微分的边缘检测算子，Laplace 算子定义为：

$$\nabla^2 f=\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}$$

X 方向为：$\frac{\partial^2 f}{\partial x^2}=f(x+1,y)+f(x-1,y)-2f(x,y)$

Y 方向为：$\frac{\partial^2 f}{\partial y^2}=f(x,y+1)+f(x,y-1)-2f(x,y)$

故离散二位拉普拉斯算子为：

$$\nabla^2 f(x,y)=f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)-4f(x,y)$$

对应的模板为：

0	1	0
1	-4	1
0	1	0

也可以使用扩展的 Laplace 模板：

1	1	1
1	-8	1
1	1	1

通常我们使用拉普拉斯算子进行增强时要用如下公式：

$g(x,y) = f(x,y) + c[\nabla^2 f(x,y)]$　其中 c=-1（对应上面的模板）

（4）Canny algorithm

Canny 边缘检测于 1986 年由 JOHN CANNY 首次在论文《A Computational Approach to Edge Detection》中提出的。Canny 发现，在不同视觉系统上对边缘检测的要求较为类似，因此，可以实现一种具有广泛应用意义的边缘检测技术。

Canny 边缘检测的一般标准包括：

(1)以低的错误率检测边缘，也即意味着需要尽可能准确的捕获图像中尽可能多的边缘。

(2)检测到的边缘应精确定位在真实边缘的中心。

(3)图像中给定的边缘应只被标记一次，并且在可能的情况下，图像的噪声不应产生假的边缘

Canny 边缘检测算法可以分为以下 5 个步骤：

(1)使用高斯滤波器，以平滑图像，滤除噪声。

(2)计算图像中每个像素点的梯度强度和方向。

(3)应用非极大值（Non-Maximum Suppression）抑制，以消除边缘

检测带来的杂散响应。

(4)应用双阈值（Double-Threshold）检测来确定真实的和潜在的边缘。

(5)通过抑制孤立的弱边缘最终完成边缘检测。

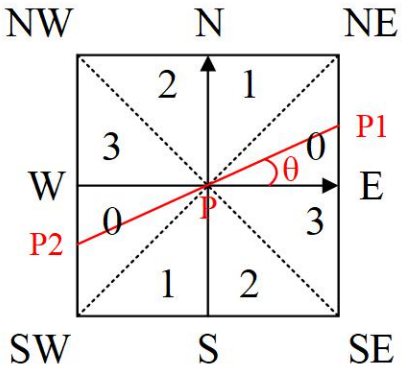
Canny 算法前两步在前文都有体现，梯度用 sobel 算子计算即可，只不过要精确计算梯度的计算大小与方向。在下面我们主要解释后三步：

1) 非极大值抑制

(1)将当前像素的梯度强度与沿正负梯度方向上的两个像素进行比较。

(2)如果当前像素的梯度强度与另外两个像素相比最大，则该像素点保留为边缘点，否则该像素点将被抑制。

如图所示：



将梯度分为 8 个方向，分别为 E、NE、N、NW、W、SW、S、SE，其中 0 代表 0~45°,1 代表 45~90°，2 代表-90~-45°，3 代表-45~0°。像素点 P 的梯度方向为 theta，则像素点 P1 和 P2 的梯度线性插值为：

$$\tan(\theta) = G_y / G_x$$

$$G_{p1} = (1 - \tan(\theta)) \times E + \tan(\theta) \times NE$$

$$G_{p2} = (1 - \tan(\theta)) \times W + \tan(\theta) \times SW$$

2) 双阈值检测

如果边缘像素的梯度值高于高阈值，则将其标记为强边缘像素；如果边缘像素的梯度值小于高阈值并且大于低阈值，则将其标记为弱边缘像素；如果边缘像素的梯度值小于低阈值，则会被抑制。阈值的选择取决于用户自己定义的参数。

3) 抑制孤立的弱边缘

对于弱边缘像素有一些争论，因为这些像素可以从真实边缘提取也可以是因噪声或颜色变化引起的。为了获得准确的结果，应该抑制由后者引起的弱边缘。通常，由真实边缘引起的弱边缘像素将连接到强边缘像素，而噪声响应未连接。为了跟踪边缘连接，通过查看弱边缘像素及其 8 个邻域像素，只要其中一个为强边缘像素，则该弱边缘点就可以保留为真实的边缘。当然，**只要一个像素与强边缘接触，那么这条弱边缘都被保留**。

这个思想可以使用搜索算法实现，在本题中我们采用深度优先搜索算法实现：

(1)准备一个栈 s，一个队列 q，设联通指示变量 connected 为假。从图像的第一个点开始，进入 2。

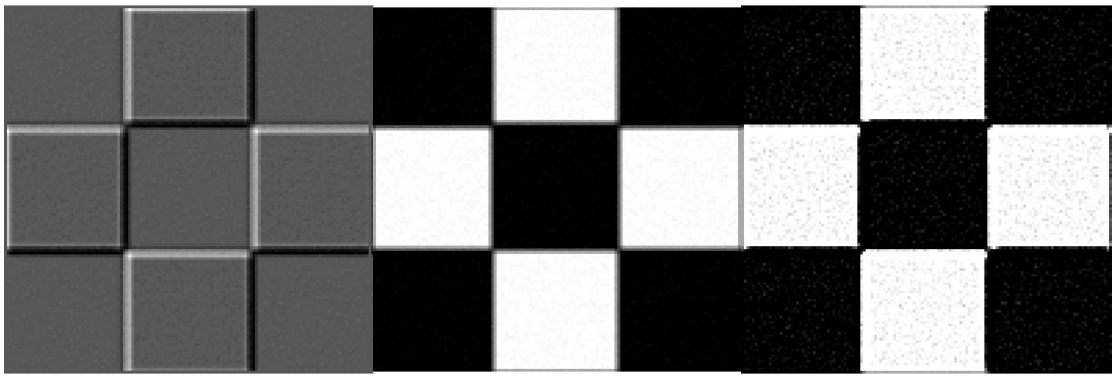
(2)如果这个点是弱边界点并且没有被标记，把它标记，并把它作为第一个元素放入栈 s 中，同时把它放入记录连通曲线的队列 q，进入 3。如果这个点不是弱边界或者已经被标记过，到图像的下一个点，重复 2。

(3)从栈 s 中取出一个元素，查找它的 8 像素领域。如果一个领域像素是弱边界并且没有被标记过，把这个领域像素标记，并加入栈 s 中，同时加入队列 q。同时查找领域对应的强边界图，如果有一个像素是强边界，表示这条弱边界曲线和强边界联通，设置 connected 为真。重复 3 直到栈中没有元素了。如果 connected

为假，则依次从队列 q 中取出每个元素，清空标记。如果 connected 为真，保留标记。
(4)清空队列 q，设置 connected 为假，移动到图像的下一个点，到 2。

2. unsharp masking

mask 大小：5×5； σ：1.5；k：5



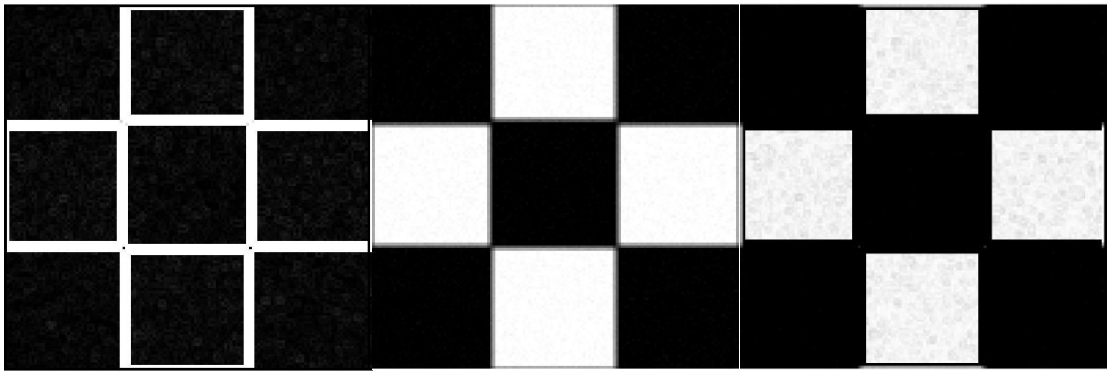
高通　　原图　　增强

高通　　原图　　增强

注：高通结果按照最大和最小值区间等分灰度显示

3. Sobel edge detector

mask 大小：3×3；叠加倍数：k=-1



高通　　原图　　增强

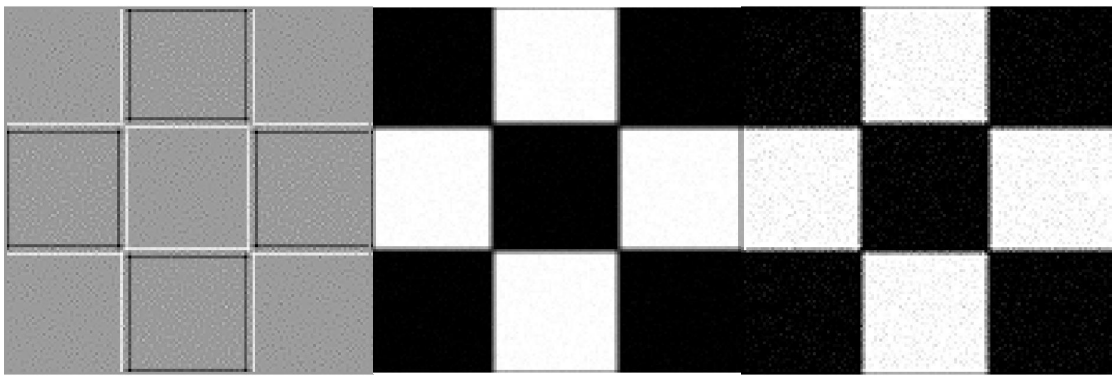
高通　　原图　　增强

注：高通结果按照[0,255]区间等分灰度显示

4. Laplace edge detection

mask 大小：3×3；叠加倍数：k=-1

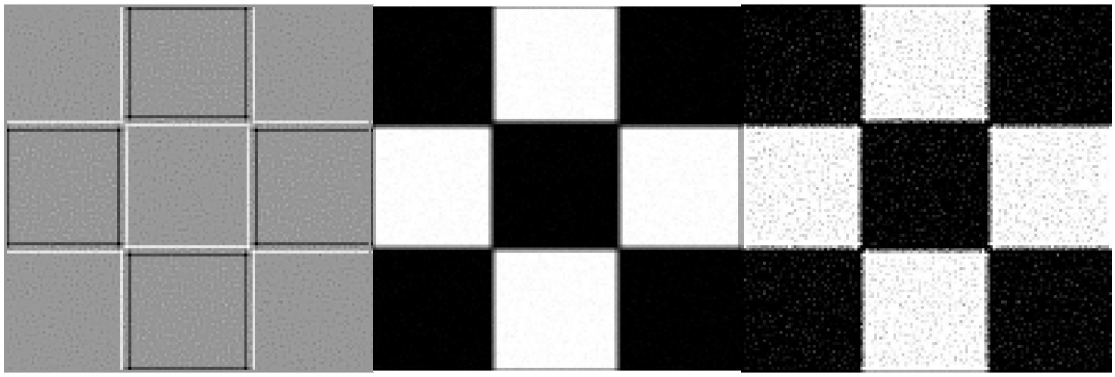
原始算子：



高通　　原图　　增强

高通　　原图　　增强

改进算子：



高通　　原图　　增强

高通　　原图　　增强

注：高通结果按照最大和最小值区间等分灰度显示

5. Canny algorithm

mask 大小：5×5； σ：1.5；

高阈值在全体像素排名中的位置：0.9

——这个参数是把第三步中得到的边界图片画出一个直方图，在高阈值位置左边所有像素占总像素的百分比，即：不是边界像素的百分比，当然，也包括第五步检测出的和一级像素接触的二级像素。

低阈值与高阈值之比：0.4

——用于划分肯定不是边界的像素，使用这个阈值在第四步中除去这些没有争议的像素

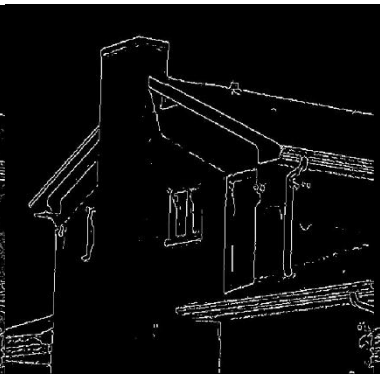
实现过程：

原图

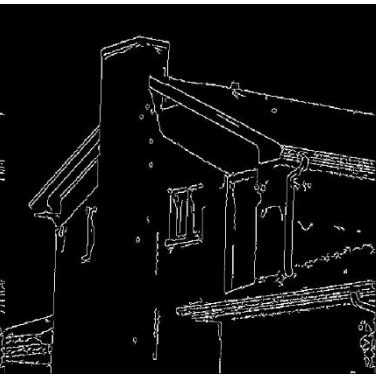
高斯模糊

梯度

全部边界



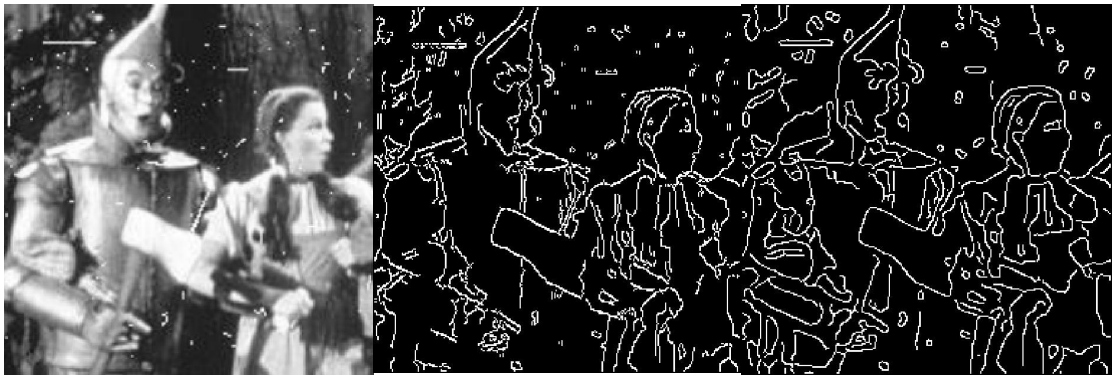
强边界



最终效果

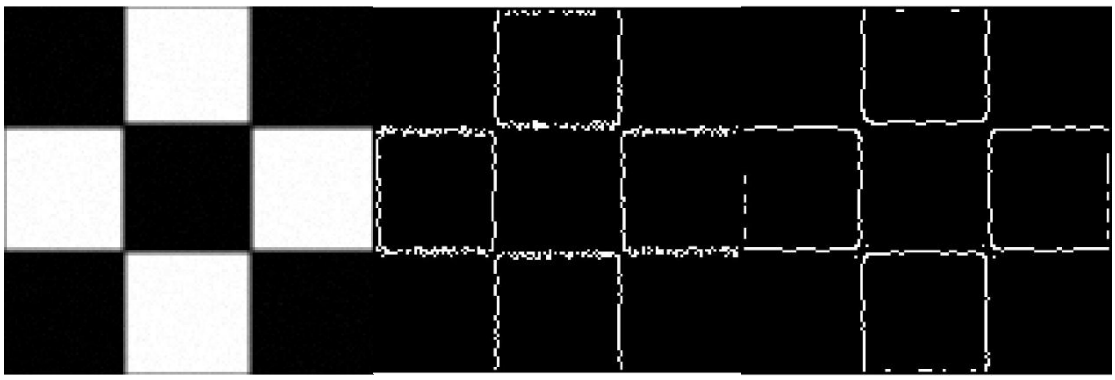
注：全部边界中白色的为强边界，灰色的为弱边界

自己的函数与系统函数对比：

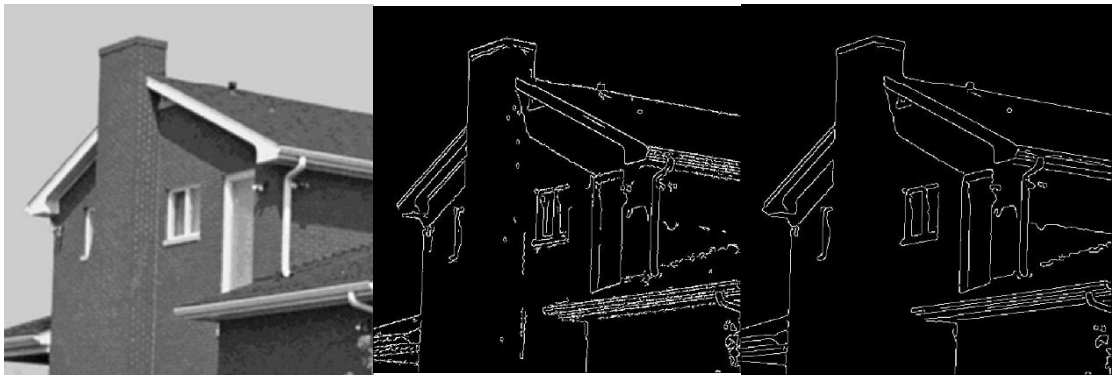


原图　　自己函数　　系统函数

原图　　自己函数　　系统函数



原图　　自己函数　　系统函数



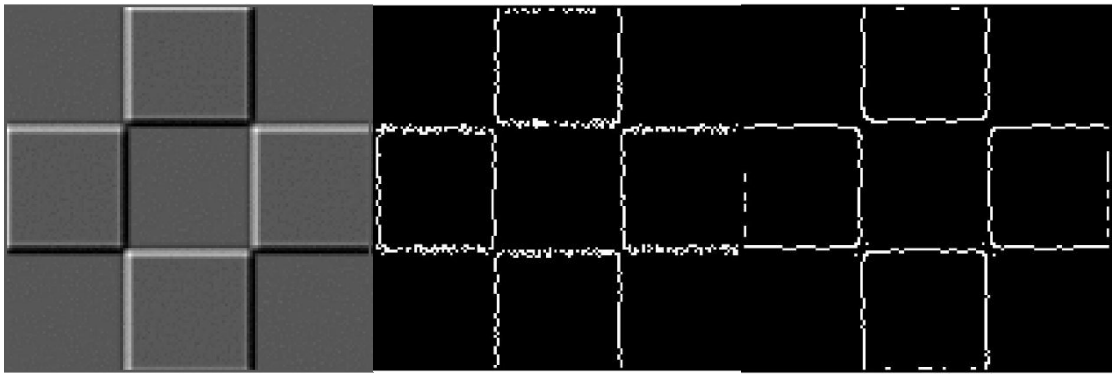
原图　　自己函数　　系统函数

可以看出，自己的函数与系统函数比较接近但仍有差距，主要表现在噪点的处理和边界平滑性等方面

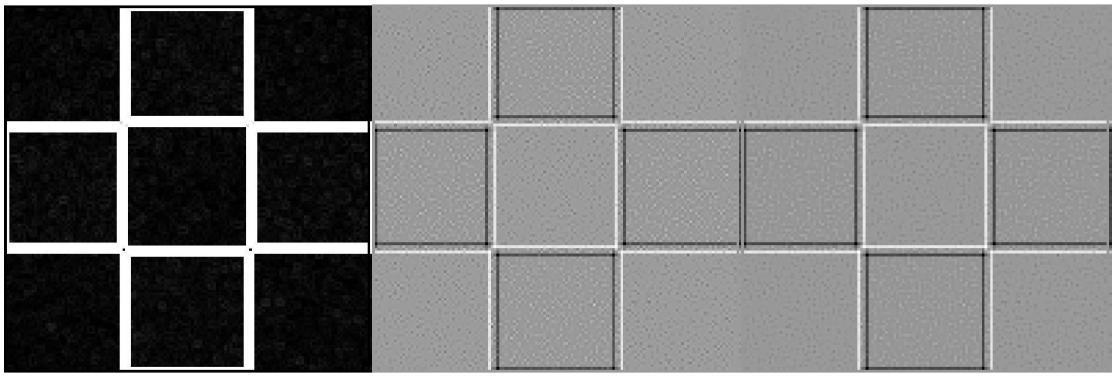
6. 简要分析

（1）边界检测

在边界检测方面，我们以图 3 为例：



Unsharp　　canny 自编　　canny 系统



Sobel　　Laplace1　　Laplace2

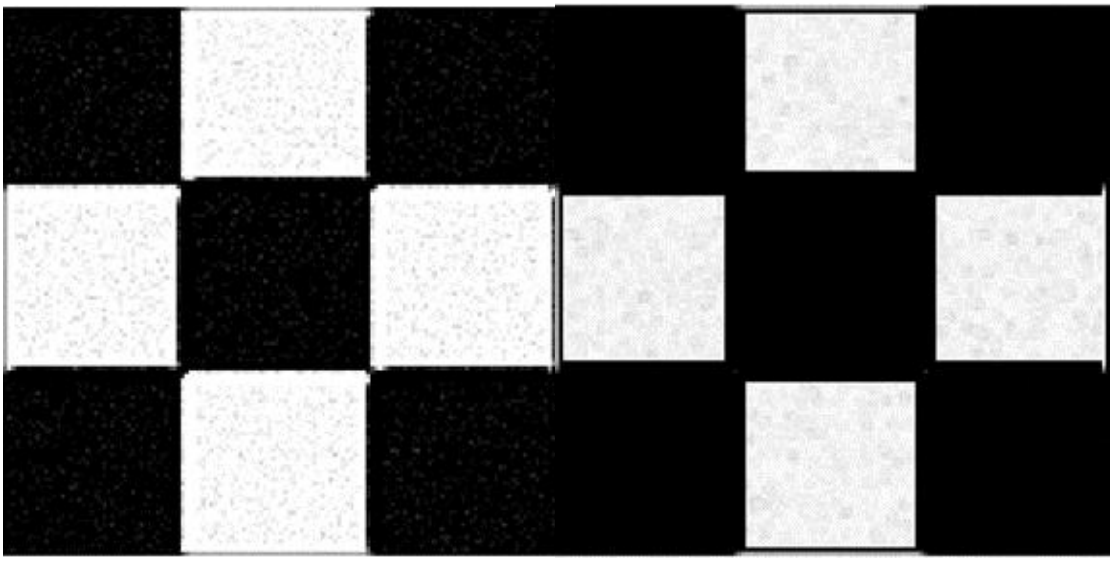
从上图可以明显看出，unsharp 和 Laplace 最不适合用于边界检测，因为它们自身产生的矩阵包括负数，这就使得显示成为困难，按区间显示时大部分 0 值成为灰色，会使得显示变得不明显。而且 Laplace 有一个很明显的特点就是双边缘效应，这就进一步为边界检测带来了困难。而 sobel 算子虽然在边界检测方面还可以，但是无奈当边界过

度太过缓慢时，sobel 算子检测出的边缘就会变得特别粗。相反 canny 算法本身就有防止边界过粗的设计，因此效果最好。

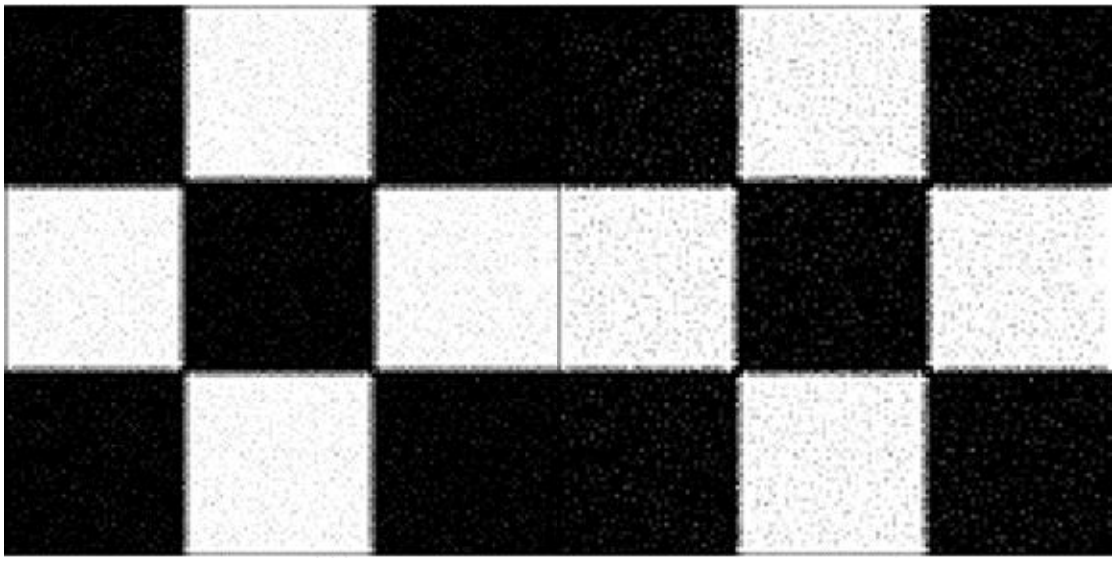
（2）图像增强

在图像增强方面，我们图三、四分别做对比，

由于 canny 算法不包括图像增强的功能，我们不做讨论。



Unsharp　　sobel



Laplace1　　Laplace2

从图三我们可以看出，sobel 算子对于变化很慢的边缘会极大地放大，

所以它增强后的图片边缘是最粗的。但是它对于噪声又显得不太敏感。与之相比的是 Laplace 算子，由于其为图像的二阶导数信息，所以对于细小的变化特别敏感，对于噪声也会起到放大作用。可以看出，随着 Laplace 算子越精确，对噪声的放大越强。而 unsharp masking 方法比较好的保留了图片的原始信息，而且不会放大噪声。

Unsharp

sobel

Laplace1

Laplace2

对图四的比较更能说明问题：对于 sobel 算子，仍然会进行一个加粗边缘的增强。比较优化 Laplace 算子与 unsharp masking 方法，可以

发现二者的增强效果差不多，可以看出图四比之图三，噪声信息少了很多，所以 Laplace 算子的优势体现了出来，说明它确实可以提取更多的细节信息。

附录

1. 参考文献

[1] Rafael C. Gonzalez（拉斐尔 C. 冈萨雷斯）,Richard E. Woods （理查德 E. 伍兹）. 数字图像处理(第三版)（英文版).北京：电子工业出版社. 2017 年.
[2] Hairong Qi ,"Lecture Notes on the Gaussian Distribution"
[3]反锐化掩模(unsharp masking)
http://blog.sina.com.cn/s/blog_5a69abfb0101fsgh.html
[4]边缘检测之 Canny https://www.cnblogs.com/techyan1990/p/7291771.html
[5]Canny 边缘检测算法的实现
https://www.cnblogs.com/mightycode/p/6394810.html
[6] JOHN CANNY, A Computational Approach to Edge Detection, MEMBER, IEEE

2. 源代码

（1）第一问

1) Gauss 函数

%pic 为一个图像片段，sigma 为方差，num 为计算结果

```
function num = gauss(pic,sigma)
[r,c]=size(pic);

pad=floor(r/2);%偏移量

para=zeros(r,c);
for i=1:r
```

```
    for j=1:c
        para(i,j)=exp(-((i-pad)^2+(j-pad)^2)/2/sigma^2);
    end
end
para=para/2/pi/sigma^2;
tol=sum(sum(para));
para=para/tol;
num=0;
for i=1:r
    for j=1:c
        num=num+para(i,j)*pic(i,j);
    end
end
end
```

2) Midia 函数

```
%pic 为一个图像片段，num 为计算结果;返回中位数

function num = media(pic)
[r,~]=size(pic);
pic=sort(pic);
center=floor(r/2)+1;
num=pic(center,center);
end
```

3) 主函数 lowpass

```
%空域低通滤波器

masksize=5;%mask 大小

sigma=1.5;%方差大小

% g=imread('test1.pgm');
g=imread('test2.tif');
```

```
[r,c]=size(g);
padding=floor(masksize/2);
pa=zeros(r+padding*2,c+padding*2);

for i=1:r        %padding 后的矩阵

    for j=1:c
        pa(i+padding,j+padding)=g(i,j);
    end
end
g1=g;
% g1=double(g);
for i=1:r
    for j=1:c
        g1(i,j)=gauss(pa(i:i+masksize-1,j:j+masksize-1),sigma);
%           g1(i,j)=media(pa(i:i+masksize-1,j:j+masksize-1));
    end
end
% figure
% imshow(g);set(gca,'position',[0 0 1 1]);

% % title('原图');

figure
imshow(g1);set(gca,'position',[0 0 1 1]);
set(gcf,'position',[500,280,500,500])

% title('σ=1.5  高斯低通');

% title('中值滤波器')
```

（2）第二问

1) Unsharp masking

```
%空域高通滤波器 unmask
```

```
masksize=5;%mask 大小

sigma=1.5;%方差大小

k=5;%叠加倍数

% g=imread('test3_corrupt.pgm');
g=imread('test4 copy.bmp');
g=double(g);
[r,c]=size(g);
padding=floor(masksize/2);
pa=zeros(r+padding*2,c+padding*2);

for i=1:r          %padding 后的矩阵

      for j=1:c
            pa(i+padding,j+padding)=g(i,j);
      end
end
g1=g;
% g1=double(g);
for i=1:r
      for j=1:c
            g1(i,j)=gauss(pa(i:i+masksize-1,j:j+masksize-1),sigma);
%               g1(i,j)=media(pa(i:i+masksize-1,j:j+masksize-1));
      end
end

figure%原图

imshow(g,[0,255]);set(gca,'position',[0 0 1 1]);


figure%高通滤波结果

g2=g-g1;%unmask
g2(1,:)=0;g2(:,1)=0;g2(r,:)=0;g2(:,r)=0;
maxnum=max(max(g2));
minnum=min(min(g2));
```

imshow(g2,[minnum,maxnum]);set(gca,'position',[0 0 1 1]);%范围自己调整，用于清晰显示

figure%增强结果

```
gf=g+k*g2;
imshow(gf,[0,255]);set(gca,'position',[0 0 1 1]);
```

2) Sobel

%空域高通滤波器 sobel

masksize=3;%mask 大小

k=-1;%叠加倍数

```
% g=imread('test3_corrupt.pgm');
g=imread('test4 copy.bmp');
g=double(g);
[r,c]=size(g);
padding=floor(masksize/2);
pa=zeros(r+padding*2,c+padding*2);
for i=1:r        %padding 后的矩阵
    for j=1:c
        pa(i+padding,j+padding)=g(i,j);
    end
end
g1=g;
% g1=double(g);
for i=1:r
    for j=1:c
        pic=pa(i:i+masksize-1,j:j+masksize-1);

g1(i,j)=abs(pic(3,1)+2*pic(3,2)+pic(3,3)-pic(1,1)-2*pic(1,2)-pic(1,3))...
```

```
+abs(pic(1,3)+2*pic(2,3)+pic(3,3)-pic(1,1)-2*pic(2,1)-pic(3,1));
      end
end

figure%原图

imshow(g,[0,255]);set(gca,'position',[0 0 1 1]);


figure%高通滤波结果

g1(1,:)=0;g1(:,1)=0;g1(r,:)=0;g1(:,r)=0;
maxnum=max(max(g1));
minnum=min(min(g1));
imshow(g1,[minnum,maxnum]);set(gca,'position',[0 0 1 1]);


figure%增强结果

gf=g+k*g1;
imshow(gf,[0,255]);set(gca,'position',[0 0 1 1]);
```

3) Laplace

```
%空域高通滤波器 laplace

masksize=3;%mask 大小

k=-1;%叠加倍数

% g=imread('test3_corrupt.pgm');
g=imread('test4 copy.bmp');
g=double(g);
[r,c]=size(g);
padding=floor(masksize/2);
pa=zeros(r+padding*2,c+padding*2);

for i=1:r        %padding 后的矩阵
```

```
    for j=1:c
        pa(i+padding,j+padding)=g(i,j);
    end
end
g1=g;
% g1=double(g);
for i=1:r
    for j=1:c
        pic=pa(i:i+masksize-1,j:j+masksize-1);
%           g1(i,j)=pic(1,2)+pic(2,1)+pic(2,3)+pic(3,2)-4*pic(2,2); %模
板 1

        g1(i,j)=pic(1,1)+pic(1,2)+pic(1,3)+pic(2,1)+pic(2,3)...%模板 2
            +pic(3,1)+pic(3,2)+pic(3,3)-8*pic(2,2);
    end
end

figure%原图

imshow(g,[0,255]);set(gca,'position',[0 0 1 1]);


figure%高通滤波结果

g1(1,:)=0;g1(:,1)=0;g1(r,:)=0;g1(:,r)=0;
maxnum=max(max(g1));
minnum=min(min(g1));
imshow(g1,[minnum,maxnum]);set(gca,'position',[0 0 1 1]);


figure%增强结果

gf=g+k*g1;
imshow(gf,[0,255]);set(gca,'position',[0 0 1 1]);
```

4) Canny

(1)gat_canny 子函数

```
function cannyArry = get_canny(img,masksize,sigma,percentOfPixelsNotEdges,thresholdRatio)
% img=double(imread('test4 copy.bmp'));
% % img=double(rgb2gray(imread('1.png')));
% masksize=5;%mask 大小
% sigma=1.5;%方差大小
% percentOfPixelsNotEdges=0.9;%非边缘占比
% thresholdRatio=0.4;%低阈值与高阈值之比
%% padding
[r,c]=size(img);
z=zeros(r,c);%与原图等大的零矩阵（分配内存用）
padding=floor(masksize/2);
pa=zeros(r+padding*2,c+padding*2);%padding 的矩阵
for i=1:r        %求 padding 后的矩阵
    for j=1:c
        pa(i+padding,j+padding)=img(i,j);
    end
end
for i=1:padding%用原图的边缘填充 padding 矩阵的边缘
    pa(i,1+padding:c+padding)    = img(1,:);%top
%     pa(i,1:padding)=pa(i,1+padding);
%     pa(1,c+padding+1:c+2*padding)=pa(i,c+padding);
```

```
        pa(i+r+padding,1+padding:c+padding)= img(r,:);%bottom
%         pa(i+r+padding,1:padding)=pa(i+r+padding,1+padding);
%
pa(1+r+padding,c+padding+1:c+2*padding)=pa(i+r+padding,c+padding);
        pa(1+padding:r+padding,i)    = img(:,1);%left
        pa(1:padding,i)=pa(1+padding,i);
        pa(r+padding+1:r+2*padding,i)=pa(r+padding,i);
        pa(1+padding:r+padding,i+c+padding)= img(:,c);%right
        pa(1:padding,i+c+padding)=pa(1+padding,i+c+padding);

pa(r+padding+1:r+2*padding,i+c+padding)=pa(r+padding,i+c+padding);
end
%% 高斯模糊

imgGauss=img;%高斯模糊后的图片

for i=1:r
        for j=1:c

imgGauss(i,j)=gauss(pa(i:i+masksize-1,j:j+masksize-1),sigma);
        end
end
%       figure;imshow(imgGauss,[0,255]);set(gca,'position',[0       0       1
1]);set(gcf,'position',[500,280,500,500])
%% sobel 算子计算梯度

gradx=z;%x 方向梯度

grady=z;%y 方向梯度

gradm=z;%梯度大小

dir=z;%梯度方向
```

```
for i=1:r
    for j=1:c
        pic=pa(i:i+masksize-1,j:j+masksize-1);

grady(i,j)=pic(1,1)+2*pic(1,2)+pic(1,3)-pic(3,1)-2*pic(3,2)-pic(3,3);

gradx(i,j)=pic(1,3)+2*pic(2,3)+pic(3,3)-pic(1,1)-2*pic(2,1)-pic(3,1);
        gradm(i,j)=sqrt(gradx(i,j)^2+grady(i,j)^2);
        theta=atand(grady(i,j)/gradx(i,j))+90;
        if (theta >= 0 && theta < 45)
            dir(i,j) = 2;
        elseif (theta >= 45 && theta < 90)
            dir(i,j) = 3;
        elseif (theta >= 90 && theta < 135)
            dir(i,j) = 0;
        else
            dir(i,j) = 1;
        end
    end
end
maxgrad = max(gradm(:));

%归一化

gradm=gradm/maxgrad;
%       figure;imshow(gradm);set(gca,'position',[0 0 1
1]);set(gcf,'position',[500,280,500,500])

%% 非极大值抑制 & 双阈值检测

counts=imhist(gradm, 64);
highThresh=find(cumsum(counts) >
percentOfPixelsNotEdges*r*c,1,'first') / 64;
lowThresh=thresholdRatio*highThresh;

edge1=z;%初步筛选出的边界

for i=2:r-1
    for j=2:c-1
```

```
            E =    gradm(i,j+1);%各方向梯度大小

            S =    gradm(i+1,j);
            W =    gradm(i,j-1);
            N =    gradm(i-1,j);
            NE = gradm(i-1,j+1);
            NW = gradm(i-1,j-1);
            SW = gradm(i+1,j-1);
            SE = gradm(i+1,j+1);

            gradnum=gradm(i,j);%当前像素梯度大小

            tantheta=abs(grady(i,j)/gradx(i,j));
            if dir(i,j)==0
                grad1=E*(1-tantheta)+NE*tantheta;
                grad2=W*(1-tantheta)+SW*tantheta;
            elseif dir(i,j)==1
                grad1=N*(1-tantheta)+NE*tantheta;
                grad2=S*(1-tantheta)+SW*tantheta;
            elseif dir(i,j)==2
                grad1=N*(1-tantheta)+NW*tantheta;
                grad2=S*(1-tantheta)+SE*tantheta;
            else
                grad1=W*(1-tantheta)+NW*tantheta;
                grad2=E*(1-tantheta)+SE*tantheta;
            end

            if gradnum>=grad1 && gradnum>=grad2%非极大值抑制


                if gradnum>=highThresh%双阈值检测

                    edge1(i,j)=2;
                elseif gradnum>=lowThresh
                    edge1(i,j)=1;
                end
            end
        end
    end
```

```
%        figure;imshow(edge1,[0,2]);set(gca,'position',[0        0        1
1]);set(gcf,'position',[500,280,500,500])
%% 抑制孤立低阈值点
edge2=edge1;%最终边缘
edge2(edge2==1)=0;
flags=z;%该点是否被遍历的标志
for i=2:r-1
    for j=2:c-1
        if edge1(i,j)==1 && flags(i,j)~=1%该点是次级边缘且未被标
识
            edgeNowS={};%存放当前边界像素的栈
            edgeNowQ={};%存放遍历出的所有边界坐标的队列
            edgePos = [i,j];
            flags(i,j)=1;%标识
            change=0;%是否要修改二级边缘为一级（1 是 0 否）
            edgeNowS = [edgeNowS,[i,j]];
            edgeNowQ = [edgeNowQ,[i,j]];
            while ~isempty(edgeNowS)%循环指导寻找完毕
                [ii,jj,edgeNowS]=popstack(edgeNowS);
                for m=ii-1:ii+1%遍历 8 个邻居
                    for n=jj-1:jj+1
                        if edge1(m,n)==1 && flags(m,n)~=1%周
围也有二级边缘且未标识
                            edgeNowS=[edgeNowS,[m,n]];%   入
```

```
栈
                                edgeNowQ=[edgeNowQ,[m,n]];% 入
队
                                flags(m,n)=1;%标识
                            elseif edge1(m,n)==2%周围有一级边缘
                                change=1;%需要修改二级边缘为一
级
                            end
                        end
                    end
                end
                if change==1%若有一个二级边缘与一级边缘相邻，这
条边改为一级边缘
                    while ~isempty(edgeNowQ)
                        [ii,jj,edgeNowQ]=popstack(edgeNowQ);
                        edge2(ii,jj)=2;
                    end
                end
            end
        end
end
%       figure;imshow(edge2,[0,2]);set(gca,'position',[0      0      1
1]);set(gcf,'position',[500,280,500,500])
edge2(edge2==2)=1;
cannyArry=edge2;
end%函数末尾

%% 子函数

function [ii,jj,stack] = popstack(stack)%弹栈
```

```
ii=stack{1}(1);
jj=stack{1}(2);
stack=stack(2:end);
end
```

(2)比较自编函数与系统函数的主函数 test

```
% g=double(imread('test4 copy.bmp'));
g=double(imread('test3_corrupt.pgm'));
% g=imread('test2.tif');
% g=double(rgb2gray(imread('1.png')));
edge1=get_canny(g,5,1.5,0.9,0.4);
% edge2=edge(g,'canny',0.2031,1.5);
edge2=edge(g,'canny',0.0625,1.5);
figure;imshow(edge1);
set(gca,'position',[0 0 1 1]);
set(gcf,'position',[500,280,500,500])
figure;imshow(edge2);
set(gca,'position',[0 0 1 1]);
set(gcf,'position',[500,280,500,500])
```