实验四 图像分割

实验目的：

1. 掌握 Prewitt、sobel、Roberts、LoG、Canny 边缘检测算子原理
2. 学会使用 edge 函数进行边缘检测
3. 使用霍夫变换进行线检测；
4. 掌握几种阈值处理法进行图像分割
5. 了解区域生长法、分裂合并法进行图像分割的原理

实验内容：

1. 在数字图像中，检测亮度不连续的三种基本类型：点、线和边缘。

(1)进行点检测的方法比较简单，我们可以使用模板[-1 -1 -1;-1 8 -1;-1 -1 -1]来进行检测，如果在模板的中心位置|R|>=T,那么孤立的点就检测出来了。其中 T 是阈值，可以设置成滤波结果中的最大值。测试图像为‘dian.tif’，流程如下：

1）先用模板对图像进行滤波，对应语句如下：

```
w=[-1 -1 -1;-1 8 -1;-1 -1 -1];
g=abs(imfilter(im2double(f),w));
```

拉普拉斯算子

2)设置 g 中的最大值为阈值：

```
T=max(g(:));
```

3）在g中寻找所有g>=T的点，就是检测出来的最大响应点。

```
g=g>=T;
```

最后请大家把结果和原图显示出来。

(2)线的检测：测试图像为‘wirebond_mask.png’，假设要找到 45° 的线，可以使用模板[2 -1 -1;-1 2 -1;-1 -1 2]来检测；用 matlab 程序完成对下面图像的线的检测。具体步骤如下：

方法一:特殊的线使用特殊的模板检测

方法二:使用霍夫变换检测,点线对偶性

1）读图像：

```
f=im2double(imread('wirebond_mask.png'));
```

2）生成模板：

```
w=[2 -1 -1;-1 2 -1;-1 -1 2];
```

3）进行滤波：

```
g=imfilter(f,w);
```

4）显示结果：

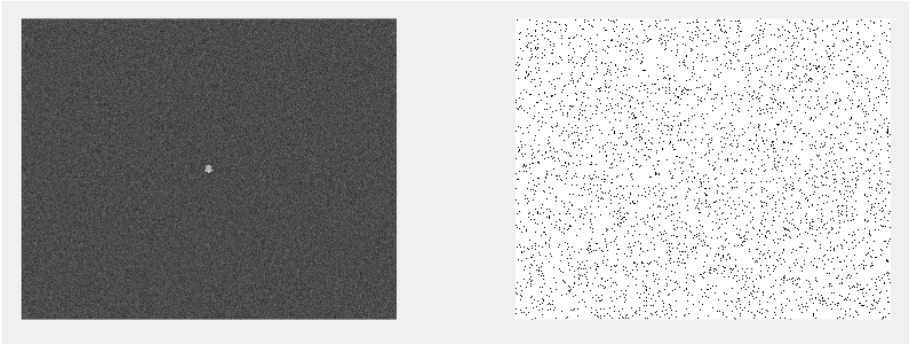
```
imshow(g,[])
```

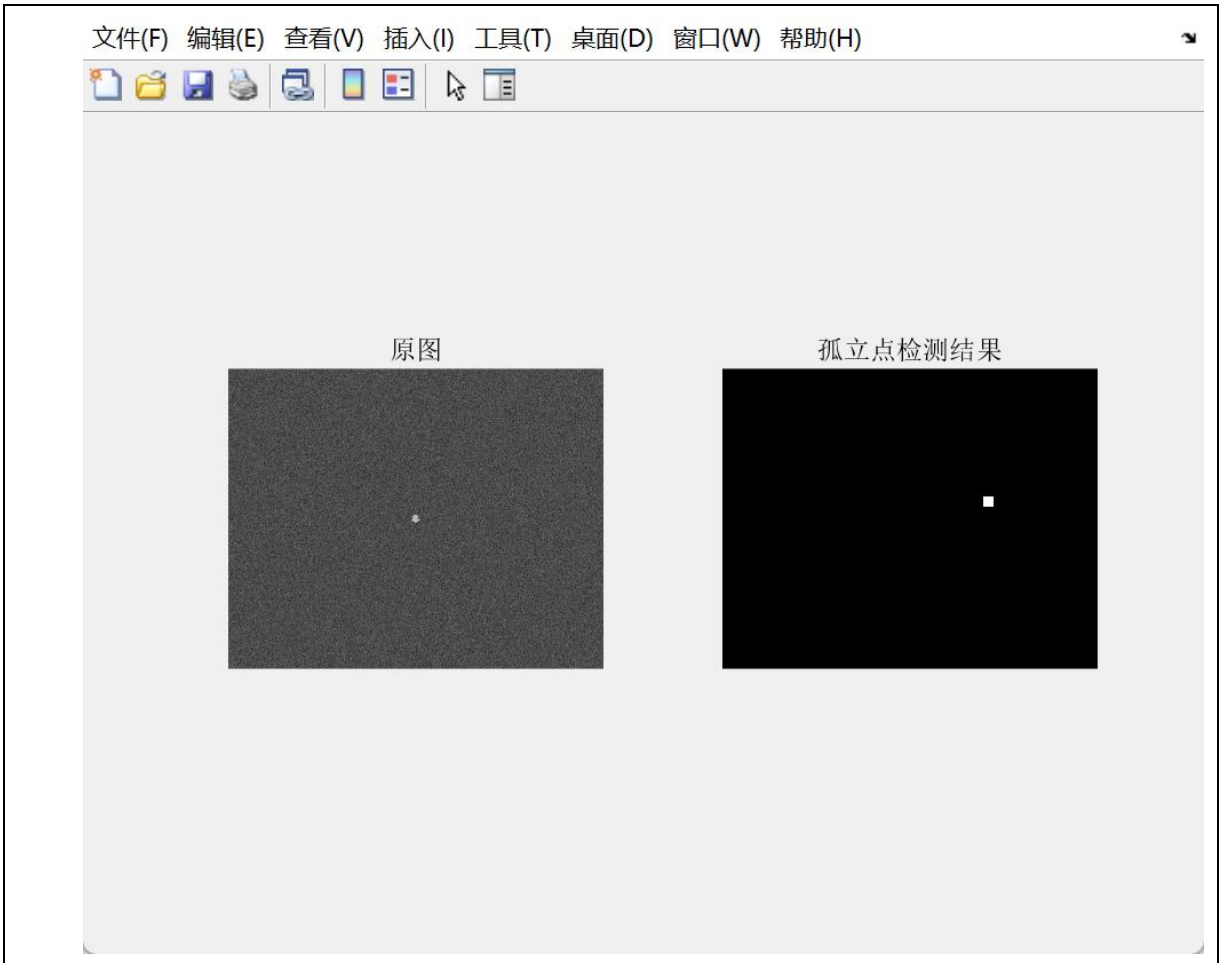
请将实验代码截图贴在此处：

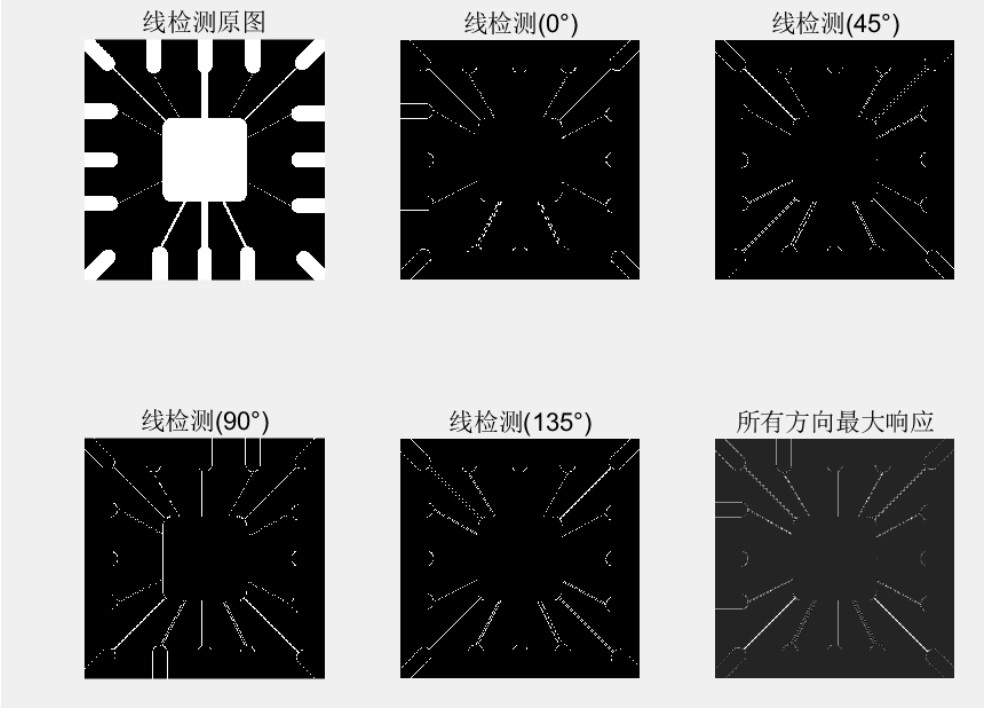
```
%% 图像分割实验
% 本实验包含点检测和线检测两部分

%% 点检测
% 读取图像
f_point = imread('dian.tif');
% 拉普拉斯算子（点检测模板）
w_point = [-1 -1 -1; -1 8 -1; -1 -1 -1];
% 应用滤波器并取绝对值
g_point = abs(imfilter(im2double(f_point), w_point));
% 设置阈值（使用最大值作为阈值）
T = max(g_point(:));
% 阈值处理
g_point_binary = g_point >= T;

%% 线检测
% 读取图像
f_line = im2double(imread('wirebond_mask.png'));
% 生成 45°线检测模板
w_line_45 = [2 -1 -1; -1 2 -1; -1 -1 2];
% 应用滤波器到正确的图像
g_line_45 = imfilter(f_line, w_line_45);

% 添加其他方向的线检测模板
w_line_0 = [-1 -1 -1; 2 2 2; -1 -1 -1];    % 水平线检测
w_line_90 = [-1 2 -1; -1 2 -1; -1 2 -1];   % 垂直线检测
w_line_135 = [-1 -1 2; -1 2 -1; 2 -1 -1];  % 135°线检测

% 应用其他方向的滤波器
g_line_0 = imfilter(f_line, w_line_0);
g_line_90 = imfilter(f_line, w_line_90);
g_line_135 = imfilter(f_line, w_line_135);

% 计算所有方向的最大响应
g_line_max = max(cat(3, g_line_0, g_line_45, g_line_90, g_line_135), [], 3);

%% 显示结果
% 点检测结果显示
figure('Name', '点检测结果');
subplot(1, 2, 1);
imshow(f_point);
title('点检测原图');
subplot(1, 2, 2);
imshow(g_point_binary);
```

```
title('点检测结果');

% 线检测结果显示 - 所有结果放在一个图中
figure('Name', '线检测结果');
subplot(2, 3, 1);
imshow(f_line);
title('线检测原图');
subplot(2, 3, 2);
imshow(g_line_0, []);
title('线检测(0°)');
subplot(2, 3, 3);
imshow(g_line_45, []);
title('线检测(45°)');
subplot(2, 3, 4);
imshow(g_line_90, []);
title('线检测(90°)');
subplot(2, 3, 5);
imshow(g_line_135, []);
title('线检测(135°)');
subplot(2, 3, 6);
imshow(g_line_max, []);
title('所有方向最大响应');
```

请将运行结果贴在此处：







2. 使用霍夫变换进行线检测。霍夫变换利用参数空间和图像空间的点线对偶性来进行线的检测。请大家学会使用图像处理工具箱中提供的霍夫变换函数 hough()、houghpeaks()、houghlines()来完成线的检测。测试图像为’bld.png’，步骤如下：

图像空间一个点对应参数中的一条线

参数中的一条线对应图像空间的一条线

（1）先来生成测试图像看看霍夫变换到底是什么意思

```
f=zeros(101,101);
f(1,1)=1;f(101,1)=1;f(1,101)=1;
f(101,101)=1;f(51,51)=1;
```

（2）调用霍夫变换的函数，看看变换后的结果是什么

```
H=hough(f);
imshow(H,[])
```

（3）通过前面的例子发现，霍夫变换是通过点线对偶性来检测图像中的边缘。了解了霍夫变换的原理后，下面用‘bld.png’这副图像来测试一下.

```
I = imread('bld.png');
f = im2double(rgb2gray(I));
[BW2,tc] = edge(f,'canny',[0.04 0.10],1.5);
% 计算和显示霍夫变换
[H,theta,rho]=hough(BW2,'ThetaResolution',0.2);
imshow(H,[],'XData',theta,'YData',rho,'InitialMagnification','fit');
xlabel('\theta'),ylabel('\rho')
axis on,axis normal

% 寻找 5 个有意义的霍夫变换的峰值
peaks=houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x=theta(peaks(:,2));
y=rho(peaks(:,1));
hold on
plot(x,y,'s','color','red');

%寻找线段
lines=houghlines(BW2,theta,rho,peaks);
figure,imshow(BW2),hold on
for k=1:length(lines)
    xy=[lines(k).point1;lines(k).point2];
    plot(xy(:,1),xy(:,2),'lineWidth',2,'Color',[.8 .8 .8]);
    %绘制线段端点
    plot(xy(1,1),xy(1,2),'x','linewidth',2,'color','yellow');
    plot(xy(2,1),xy(2,2),'x','linewidth',2,'color','red');
end
```

过线最多的 5 个点,对应图像中的五条线

请将实验代码截图贴在此处：
% 先来生成测试图像看看霍夫变换到底是什么意思

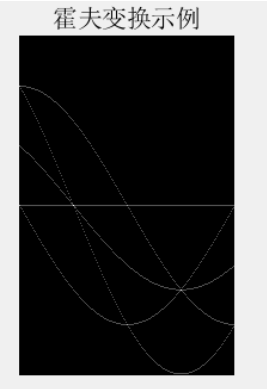
```
f = zeros(101, 101);
f(1, 1) = 1; f(101, 1) = 1; f(1, 101) = 1;
f(101, 101) = 1; f(51, 51) = 1;

% 调用霍夫变换的函数，看看变换后的结果是什么
H = hough(f);
figure, imshow(H, [])
title('霍夫变换示例')

% 通过前面的例子发现，霍夫变换是通过点线对偶性来检测图像中的边缘
% 了解了霍夫变换的原理后，下面用'bld.png'这副图像来测试一下
I = imread('bld.png');
f = im2double(rgb2gray(I));

% 使用不同的算子进行边缘检测
[BW1, tp] = edge(f, 'prewitt');
[BW2, tc] = edge(f, 'canny', [0.04 0.10], 1.5);
[BW3, ts] = edge(f, 'sobel');
[BW4, tr] = edge(f, 'roberts');
[BW5, tl] = edge(f, 'log');

% 可视化对比不同边缘检测算子的结果
figure('Name', '边缘检测算子对比');
subplot(2, 3, 1);
imshow(f);
title('原图');
subplot(2, 3, 2);
imshow(BW1);
title('Prewitt 算子');
subplot(2, 3, 3);
imshow(BW2);
title('Canny 算子');
subplot(2, 3, 4);
imshow(BW3);
title('Sobel 算子');
subplot(2, 3, 5);
imshow(BW4);
title('Roberts 算子');
subplot(2, 3, 6);
imshow(BW5);
title('LoG 算子');

% 对每种边缘检测结果进行霍夫变换和线段检测
% Canny 算子
[H_canny, theta_canny, rho_canny] = hough(BW2, 'ThetaResolution', 0.2);
```

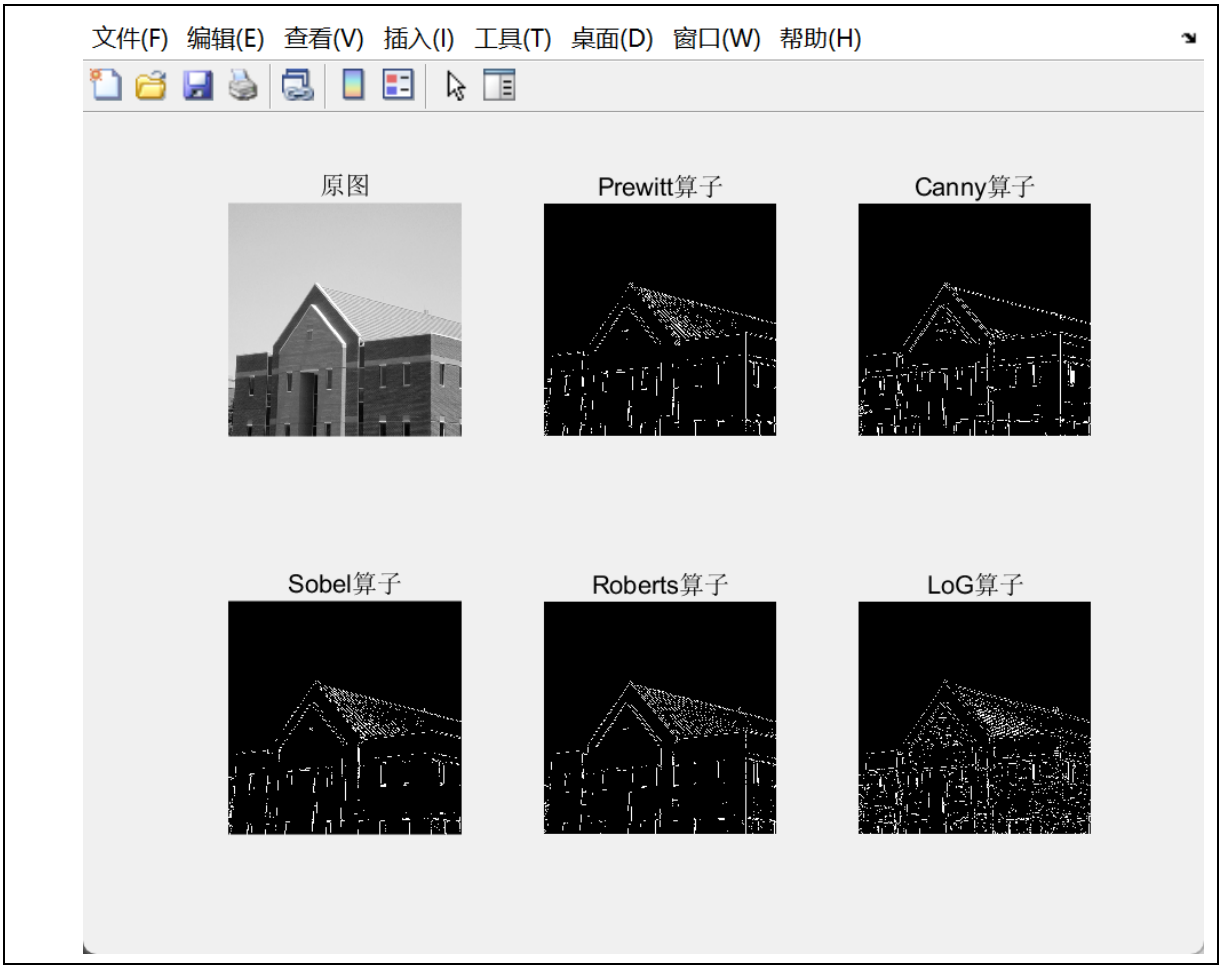
```
figure, imshow(H_canny, [], 'XData', theta_canny, 'YData', rho_canny,
'InitialMagnification', 'fit');
xlabel('\theta'), ylabel('\rho')
title('Canny 算子的霍夫变换')
axis on, axis normal

% 寻找 5 个有意义的霍夫变换的峰值
peaks_canny = houghpeaks(H_canny, 5, 'threshold', ceil(0.3*max(H_canny(:))));
x = theta_canny(peaks_canny(:, 2));
y = rho_canny(peaks_canny(:, 1));
hold on
plot(x, y, 's', 'color', 'red');

% 寻找线段
lines_canny = houghlines(BW2, theta_canny, rho_canny, peaks_canny);
figure, imshow(BW2), hold on
title('Canny 算子的线段检测')
for k = 1:length(lines_canny)
    xy = [lines_canny(k).point1; lines_canny(k).point2];
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', [.8 .8 .8]);
    % 绘制线段端点
    plot(xy(1, 1), xy(1, 2), 'x', 'linewidth', 2, 'color', 'yellow');
    plot(xy(2, 1), xy(2, 2), 'x', 'linewidth', 2, 'color', 'red');
end

% Prewitt 算子
[H_prewitt, theta_prewitt, rho_prewitt] = hough(BW1, 'ThetaResolution', 0.2);
figure, imshow(H_prewitt, [], 'XData', theta_prewitt, 'YData', rho_prewitt,
'InitialMagnification', 'fit');
xlabel('\theta'), ylabel('\rho')
title('Prewitt 算子的霍夫变换')
axis on, axis normal

peaks_prewitt = houghpeaks(H_prewitt, 5, 'threshold',
ceil(0.3*max(H_prewitt(:))));
x = theta_prewitt(peaks_prewitt(:, 2));
y = rho_prewitt(peaks_prewitt(:, 1));
hold on
plot(x, y, 's', 'color', 'red');

lines_prewitt = houghlines(BW1, theta_prewitt, rho_prewitt, peaks_prewitt);
figure, imshow(BW1), hold on
title('Prewitt 算子的线段检测')
for k = 1:length(lines_prewitt)
    xy = [lines_prewitt(k).point1; lines_prewitt(k).point2];
```

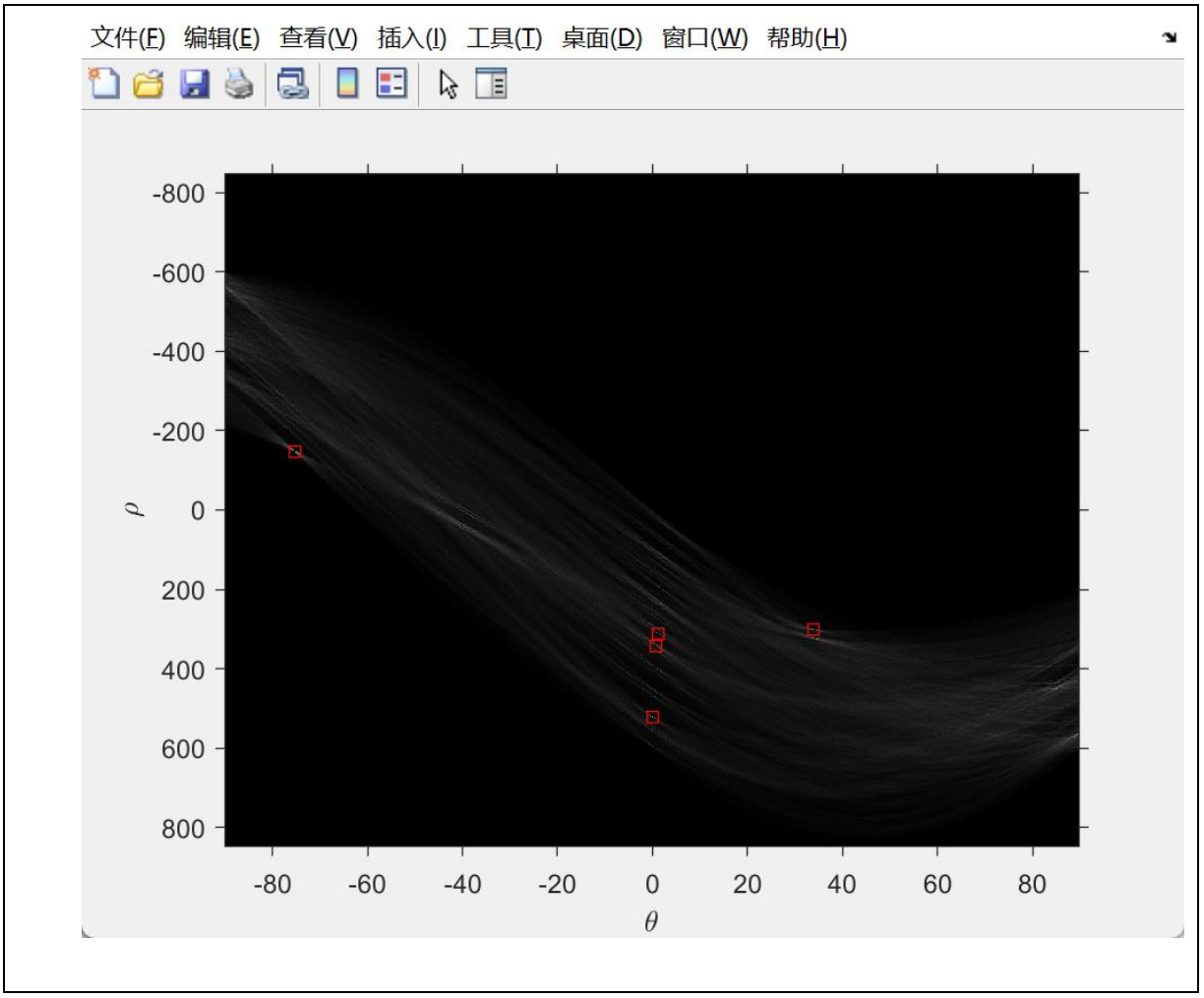
```
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', [.8 .8 .8]);
    plot(xy(1, 1), xy(1, 2), 'x', 'linewidth', 2, 'color', 'yellow');
    plot(xy(2, 1), xy(2, 2), 'x', 'linewidth', 2, 'color', 'red');
end

% Sobel 算子
[H_sobel, theta_sobel, rho_sobel] = hough(BW3, 'ThetaResolution', 0.2);
figure, imshow(H_sobel, [], 'XData', theta_sobel, 'YData', rho_sobel,
'InitialMagnification', 'fit');
xlabel('\theta'), ylabel('\rho')
title('Sobel 算子的霍夫变换')
axis on, axis normal

peaks_sobel = houghpeaks(H_sobel, 5, 'threshold', ceil(0.3*max(H_sobel(:))));
x = theta_sobel(peaks_sobel(:, 2));
y = rho_sobel(peaks_sobel(:, 1));
hold on
plot(x, y, 's', 'color', 'red');

lines_sobel = houghlines(BW3, theta_sobel, rho_sobel, peaks_sobel);
figure, imshow(BW3), hold on
title('Sobel 算子的线段检测')
for k = 1:length(lines_sobel)
    xy = [lines_sobel(k).point1; lines_sobel(k).point2];
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', [.8 .8 .8]);
    plot(xy(1, 1), xy(1, 2), 'x', 'linewidth', 2, 'color', 'yellow');
    plot(xy(2, 1), xy(2, 2), 'x', 'linewidth', 2, 'color', 'red');
end

% 对比可视化不同算子的线段检测结果
figure('Name', '不同算子的线段检测对比');
subplot(2, 2, 1);
imshow(f); title('原图');

subplot(2, 2, 2);
imshow(BW2); hold on;
for k = 1:length(lines_canny)
    xy = [lines_canny(k).point1; lines_canny(k).point2];
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', 'red');
end
title('Canny 算子线段检测');

subplot(2, 2, 3);
imshow(BW1); hold on;
for k = 1:length(lines_prewitt)
```

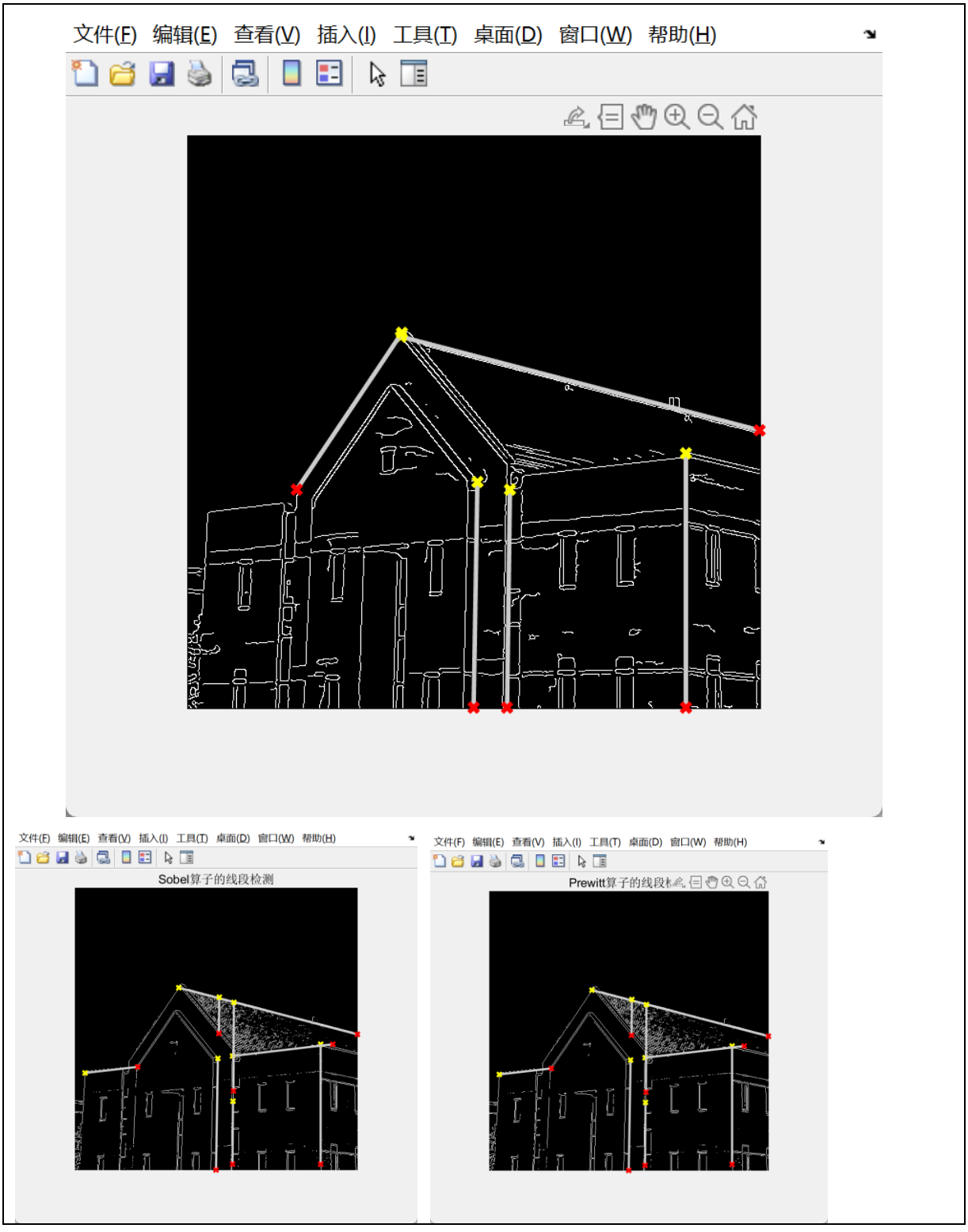
```
    xy = [lines_prewitt(k).point1; lines_prewitt(k).point2];
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', 'green');
end
title('Prewitt 算子线段检测');

subplot(2, 2, 4);
imshow(BW3); hold on;
for k = 1:length(lines_sobel)
    xy = [lines_sobel(k).point1; lines_sobel(k).point2];
    plot(xy(:, 1), xy(:, 2), 'lineWidth', 2, 'Color', 'blue');
end
title('Sobel 算子线段检测');
```

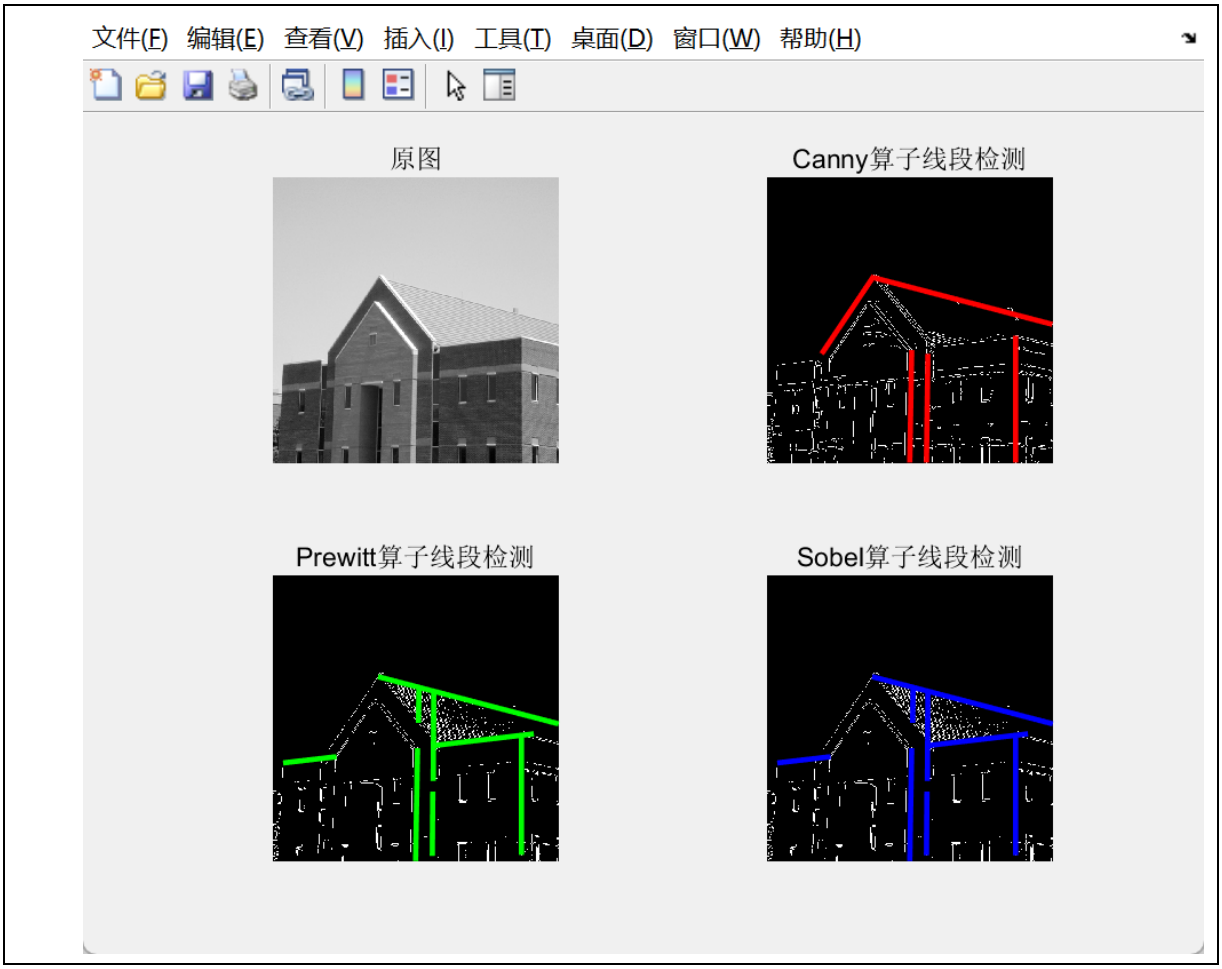
请将运行结果贴在此处：




文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
原图
Prewitt算子
Canny算子
Sobel算子
Roberts算子
LoG算子


文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
-800
-600
-400
-200
0
200
400
600
800
ρ
-80
-60
-40
-20
0
20
40
60
80
θ


文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Sobel算子的线段检测
Prewitt算子的线段检



3. 边缘检测是图像分割中经常讨论的。目前已经有非常多的边缘检测算子，例如：sobel、prewitt、roberts、LoG、canny。接下来请大家用 edge 函数进行边缘检测，分别使用不同的边缘检测算子，分析不同算子的检测性能。具体步骤如下：

1）读图像，并进行数据类型的转换：

```
I = imread('bld.png');
I = im2double(rgb2gray(I));
```

2）用不同的算子进行边缘检测：

```
[BW1,tp] = edge(I,'prewitt');
[BW2,tc] = edge(I,'canny');
[BW3,ts] = edge(I,'sobel');
[BW4,tr] = edge(I,'roberts');
[BW5,tl] = edge(I,'log');
```

3）显示结果：

```
figure, imshow(BW1)
figure, imshow(BW2)
figure, imshow(BW3)
figure, imshow(BW4)
figure, imshow(BW5)
```

4）请大家分析这几种算子的性能，把回答写在下面：

答：

答：

1. Prewitt 算子：一阶微分算子，对噪声敏感度中等，边缘定位准确性中等，检测出较粗的边缘。

2. Canny 算子：综合性能最佳，使用高斯滤波减少噪声影响，通过非极大值抑制和双阈值检测得到细而连续的边缘。

3. Sobel 算子：一阶微分算子，对噪声有一定抑制能力，边缘定位准确性中等，与 Prewitt 相似但更强调中心像素。

4. Roberts 算子：最简单的一阶微分算子，计算量小但对噪声敏感，边缘定位准确但容易产生不连续边缘。

5. LoG 算子：二阶微分算子，对噪声敏感度高，但边缘定位准确，能检测出边缘的闭合轮廓。

从实验结果可以看出：

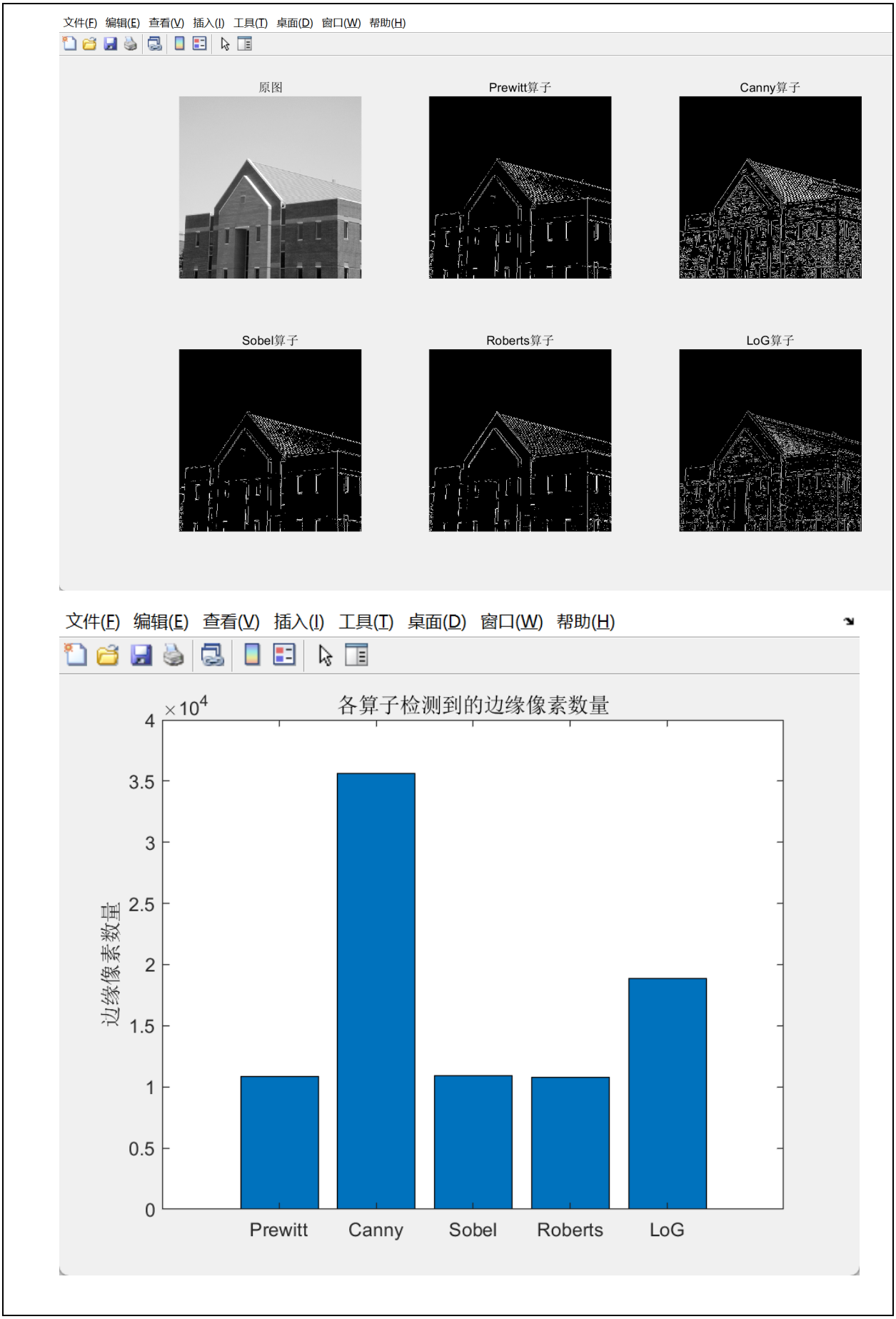
- Canny 算子检测效果最好，边缘连续且细腻，抗噪声能力强
- Roberts 算子边缘最不连续，但计算简单
- LoG 算子检测出的边缘较粗，但闭合性好
- Prewitt 和 Sobel 算子性能相近，是较为平衡的选择

请将实验代码截图贴在此处：

```
% 边缘检测算子性能对比实验

% 1）读图像，并进行数据类型的转换
I = imread('bld.png');
I = im2double(rgb2gray(I));

% 2）用不同的算子进行边缘检测
[BW1, tp] = edge(I, 'prewitt');
[BW2, tc] = edge(I, 'canny');
[BW3, ts] = edge(I, 'sobel');
[BW4, tr] = edge(I, 'roberts');
[BW5, tl] = edge(I, 'log');

% 3）显示结果 - 使用子图进行更好的可视化对比
figure('Name', '边缘检测算子对比', 'Position', [100, 100, 1000, 600]);

subplot(2, 3, 1);
imshow(I);
title('原图');

subplot(2, 3, 2);
imshow(BW1);
title('Prewitt 算子');

subplot(2, 3, 3);
imshow(BW2);
```

```
title('Canny 算子');

subplot(2, 3, 4);
imshow(BW3);
title('Sobel 算子');

subplot(2, 3, 5);
imshow(BW4);
title('Roberts 算子');

subplot(2, 3, 6);
imshow(BW5);
title('LoG 算子');

% 4）算子性能分析
% 计算每种算子检测到的边缘像素数量
prewitt_pixels = sum(BW1(:));
canny_pixels = sum(BW2(:));
sobel_pixels = sum(BW3(:));
roberts_pixels = sum(BW4(:));
log_pixels = sum(BW5(:));

% 显示边缘像素数量对比
figure('Name', '边缘检测算子性能对比');
bar([prewitt_pixels, canny_pixels, sobel_pixels, roberts_pixels, log_pixels]);
set(gca, 'XTickLabel', {'Prewitt', 'Canny', 'Sobel', 'Roberts', 'LoG'});
title('各算子检测到的边缘像素数量');
ylabel('边缘像素数量');
```

请将运行结果贴在此处：


文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
原图
Prewitt算子
Canny算子
Sobel算子
Roberts算子
LoG算子
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
各算子检测到的边缘像素数量
×10^4
4
3.5
3
2.5
2
1.5
1
0.5
0
边缘像素数量
Prewitt
Canny
Sobel
Roberts
LoG

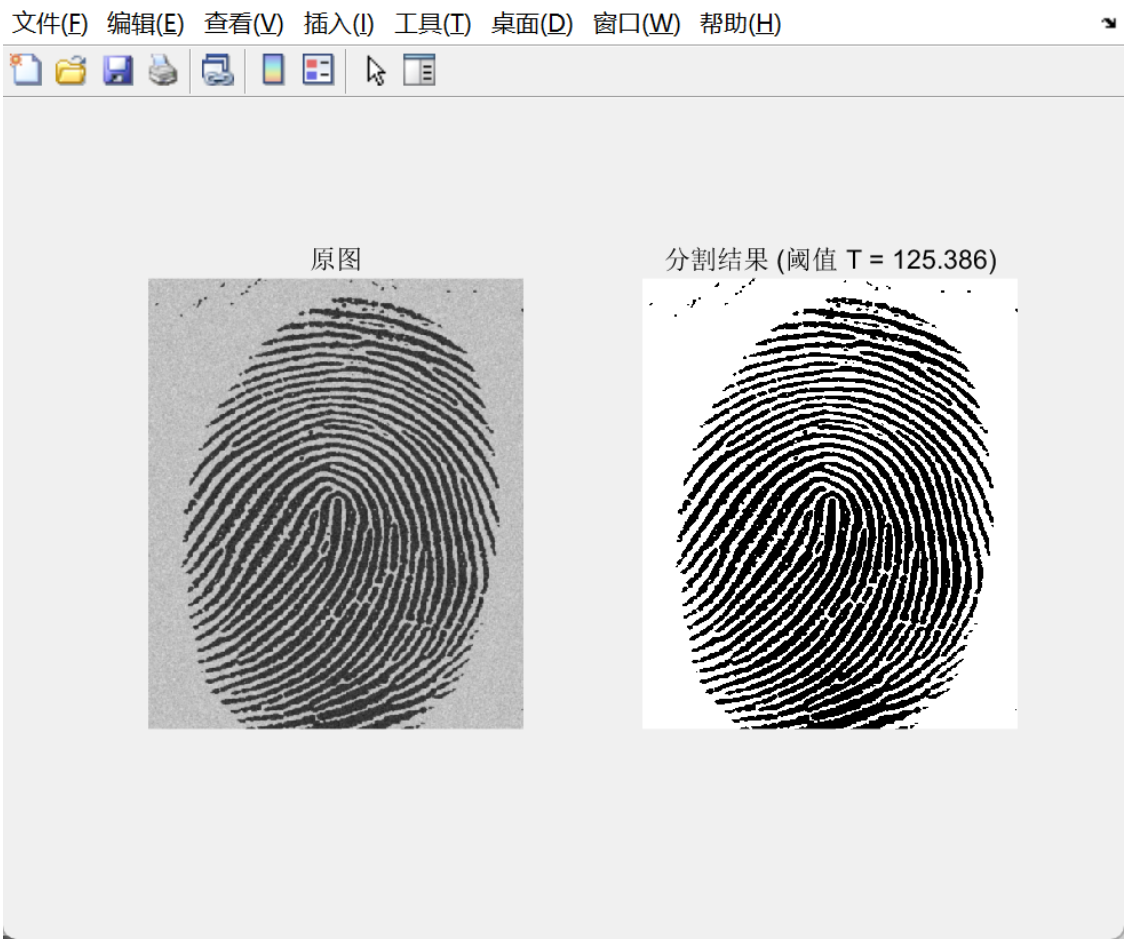
4. 基于阈值的图像分割

（1）基本全局阈值处理。其原理是基于图像的直方图来选择阈值进行分割。参考程序如下，请对测试图像‘fingerprint.tif’进行处理，把结果显示出来。

```
clear all,close all, clc
f=imread(' fingerprint.tif ');
%用全局阈值法分割
count=0;
T=mean2(f);
done=false;
while ~done
      count =count + 1;
      g=f>T;
      Tnext=0.5*(mean(f(g))+mean(f(~g)));
      done=abs(T-Tnext)<0.5;
      T=Tnext;
end
g1=im2bw(f,T/255);
```

```
clear all,close all, clc
f=imread('fingerprint.tif');
%用全局阈值法分割
count=0;
T=mean2(f);
done=false;
while ~done
    count =count + 1;
    g=f>T;
    Tnext=0.5*(mean(f(g))+mean(f(~g)));
    done=abs(T-Tnext)<0.5;
    T=Tnext;
end
g1=im2bw(f,T/255);

% 显示原图和分割结果
figure('Name', '基本全局阈值分割');
subplot(1, 2, 1);
imshow(f);
title('原图');
subplot(1, 2, 2);
imshow(g1);
title(['分割结果 (阈值 T = ' num2str(T) ')']);

% 显示迭代次数
disp(['迭代次数: ' num2str(count)]);
disp(['最终阈值: ' num2str(T)]);
```

请将运行结果贴在此处：



迭代次数: 2
最终阈值: 125.386

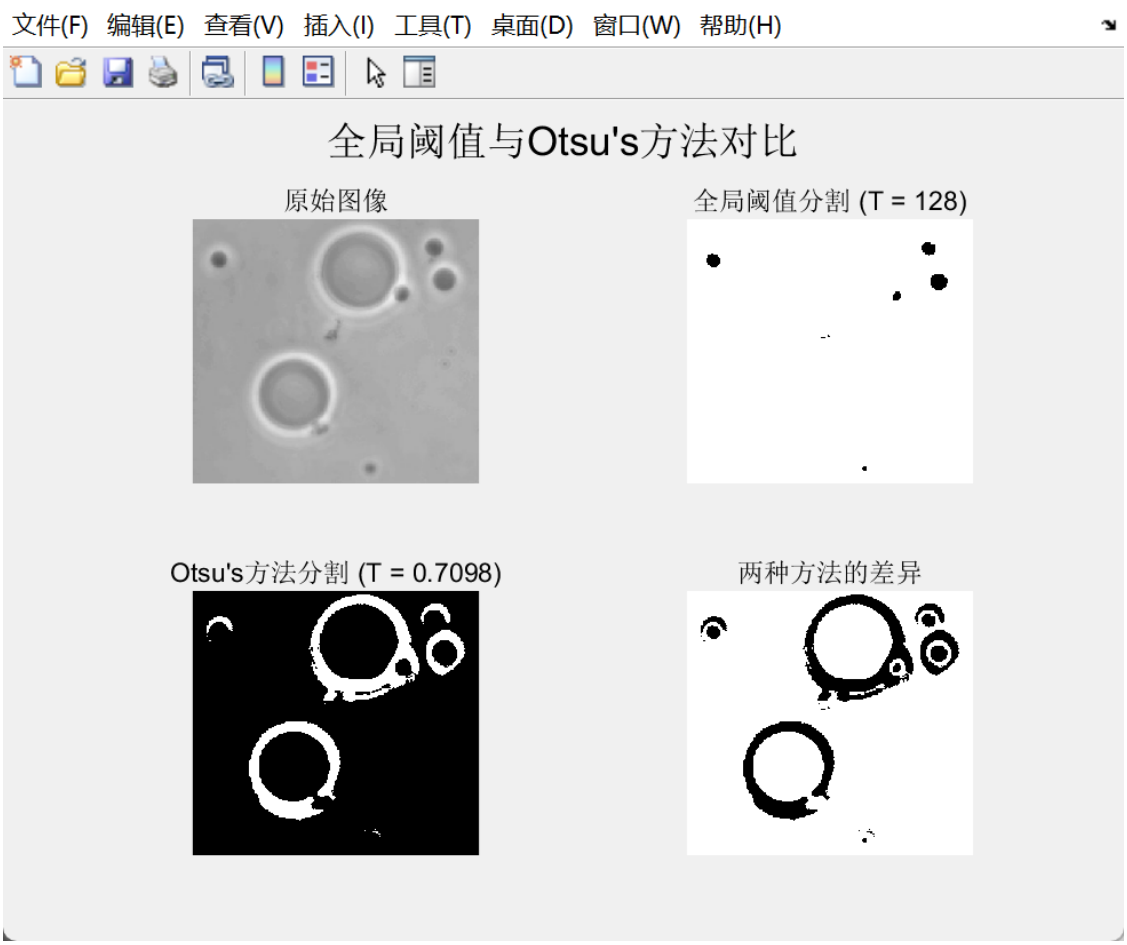
（2）先使用全局阈值的方法对‘polymersomes.tif’进行处理，观察其结果，再使用 Otsu’s 的最佳全局阈值处理对‘polymersomes.tif’进行处理，观察其结果，对比两种方法的效果。Otsu’ s 的最佳全局阈值处理步骤如下：
[T2,SM]=graythresh(f);%计算 Otsu's 阈值
g2=im2bw(f,T2);

请将实验代码截图贴在此处：

```
% 读取图像
f = imread('polymersomes.tif');

% 显示原始图像
figure;
subplot(2, 2, 1);
imshow(f);
title('原始图像');

% 方法一：使用全局阈值方法
```

```
% 选择一个固定阈值（例如，图像灰度级的中间值）
T1 = 128;
g1 = f > T1;

% 显示全局阈值分割结果
subplot(2, 2, 2);
imshow(g1);
title(['全局阈值分割 (T = ', num2str(T1), ')']);

% 方法二：使用 Otsu's 最佳全局阈值方法
[T2, SM] = graythresh(f); % 计算 Otsu's 阈值
g2 = im2bw(f, T2);

% 显示 Otsu's 方法分割结果
subplot(2, 2, 3);
imshow(g2);
title(['Otsu''s 方法分割 (T = ', num2str(T2), ')']);

% 对比两种方法
subplot(2, 2, 4);
imshow(abs(g1 - g2));
title('两种方法的差异');

% 输出阈值信息
fprintf('全局阈值: %d\n', T1);
fprintf('Otsu''s 阈值: %f (归一化值)\n', T2);
fprintf('Otsu''s 阈值对应灰度值: %d\n', round(T2*255));
fprintf('Otsu''s 方法的分离度量: %f\n', SM);

% 添加总标题
sgtitle('全局阈值与 Otsu''s 方法对比');
```

请将运行结果贴在此处：



全局阈值: 128
Otsu's 阈值: 0.709804 (归一化值)
Otsu's 阈值对应灰度值: 181
Otsu's 方法的分离度量: 0.466229

5 用区域生长法进行图像分割。

区域生长法的基本思想是根据事先定义的相似性准则，将图像中满足相似性准则的像素或者子区域聚合成更大区域的过程。其基本步骤是：

（1）先确定需要分割区域中的种子像素作为生长起点；

（2）判断周围像素是否满足事先确定的相似性准则，如果满足就合并早种子像素所在的区域；

（3）以合并区域的所有像素作为新的种子像素；

（4）重复上面的判断与合并的过程，直到再也没有满足相似性条件的像素为止。

关于区域生长法的程序我已经封装成了函数 region_growing，大家采用不同的参数调用该函数，查看程序运行的结果。具体的调用语句如下：

g=region_growing(f,110,60,30);

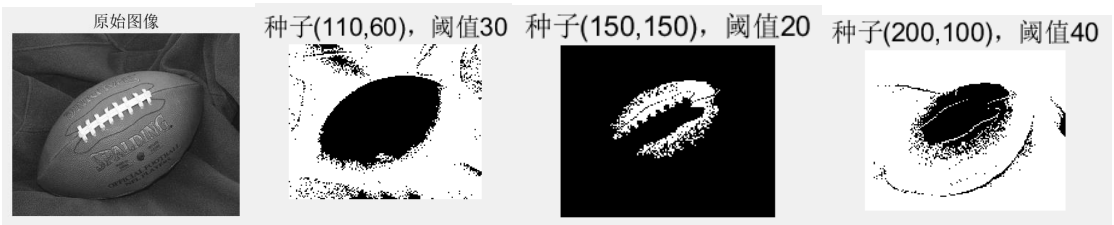
;其中 110 和 60 为生长区域的种子坐标，30 为阈值

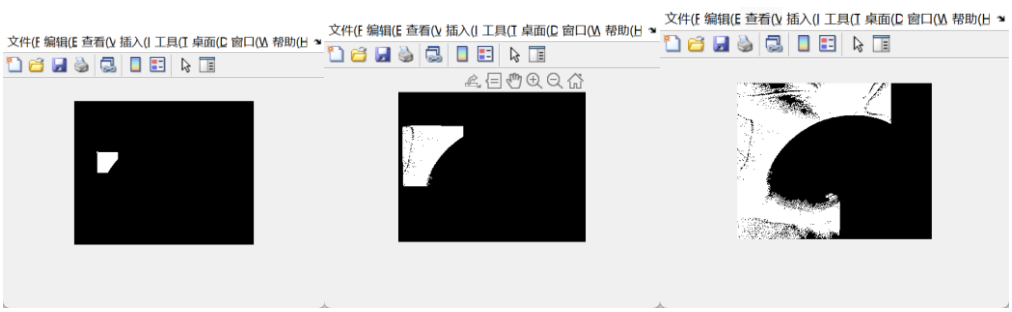
;**请大家选择不同的种子坐标和阈值，观察实验的结果**。

请将实验代码截图贴在此处：
% 读取图像

```
f = imread('football.jpg');
% 如果是彩色图像，转为灰度图
if size(f, 3) > 1
    f = rgb2gray(f);
end

% 显示原始图像
figure;
subplot(2, 2, 1);
imshow(f);
title('原始图像');

% 使用不同的种子坐标和阈值进行区域生长
% 参数 1：种子坐标(110, 60)，阈值 30
g1 = region_growing(f, 110, 60, 30);
subplot(2, 2, 2);
imshow(g1);
title('种子(110,60)，阈值 30');

% 参数 2：种子坐标(150, 150)，阈值 20
g2 = region_growing(f, 150, 150, 20);
subplot(2, 2, 3);
imshow(g2);
title('种子(150,150)，阈值 20');

% 参数 3：种子坐标(200, 100)，阈值 40
g3 = region_growing(f, 200, 100, 40);
subplot(2, 2, 4);
imshow(g3);
title('种子(200,100)，阈值 40');

% 分析结果
fprintf('不同参数的区域生长法分割结果分析：\n');
fprintf('1. 种子点位置的选择影响分割区域的起始位置\n');
fprintf('2. 阈值越大，区域生长的范围越广\n');
fprintf('3. 阈值越小，分割的区域越精细，但可能不完整\n');
```

请将运行结果贴在此处：





速度:2>1>3

6 用分裂合并法进行图像分割。

分裂合并法是根据实现确定的分裂和合并的准则，从整个图像出发，根据各个区域的不一致性，把图像或区域分裂成新的子区域；同时查找相邻区域有没有相似的特征，如果相邻子区域满足一致性准则，就合并这些相邻区域成一个比较大的区域，直到所有区域都不满足分裂合并的条件为止。

我已经将这个算法封装成了函数 split_test、predicate、splitmerge，请大家在主函数中调用 splitmerge 函数完成分割，具体调用语句如下：

g=splitmerge(f,8,@predicate);

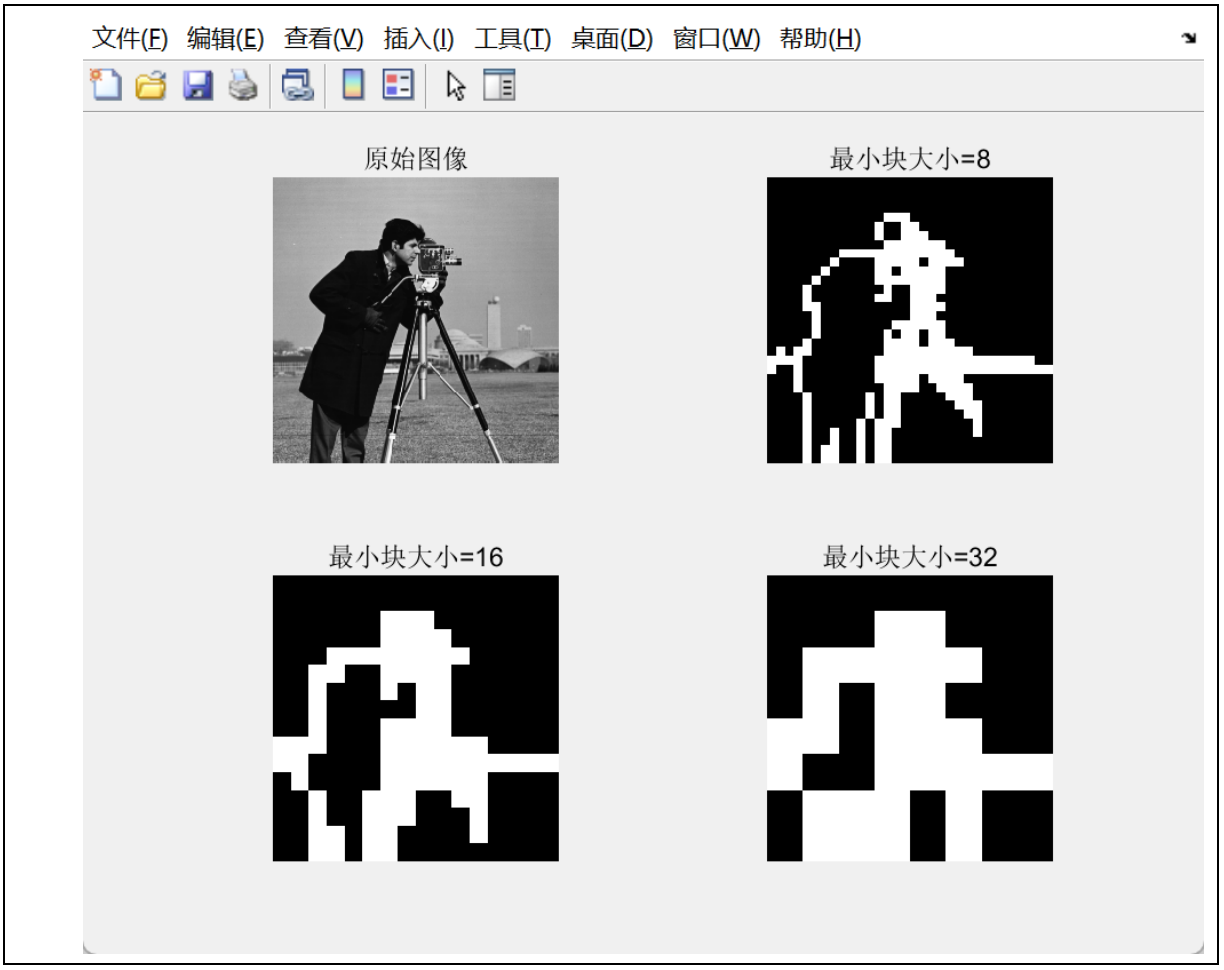
;其中第二个参数 8 是定义分解中所允许的最小的块，必须是 2 的正整数次幂

;**请大家输入不同的参数，观察运行的结果**。

请将实验代码截图贴在此处：

```
% 读取图像
f = imread('cameraman.tif');

% 使用分裂合并法进行图像分割
% 尝试不同的最小块大小参数
g1 = splitmerge(f, 8, @predicate);  % 最小块大小为 8
g2 = splitmerge(f, 16, @predicate); % 最小块大小为 16
g3 = splitmerge(f, 32, @predicate); % 最小块大小为 32

% 显示原始图像和分割结果
figure;
subplot(2,2,1), imshow(f), title('原始图像');
subplot(2,2,2), imshow(g1), title('最小块大小=8');
subplot(2,2,3), imshow(g2), title('最小块大小=16');
subplot(2,2,4), imshow(g3), title('最小块大小=32');
```

请将运行结果贴在此处：



注意：本实验报告要求直接将本 word 文档上传至课程平台