课程实践 4：图像的特征提取和增强

实验目的：

了解直方图均衡化原理。

了解边缘检测原理，比较各算子的表现。

了解 hough 变换原理以及如何应用来检测圆。

实验要求：

1、编程实现将右图直方图均衡化。

2、试用边缘检测算子(sobel, prewitt, roberts, LOG, Canny 等) 检测右图 plate.jpg 的边缘，然后运用 hough 变换，提取图中的圆。

实验环境： MATLAB R2018a

原理和方法：

①直方图均衡化算法：

1.列出原始图像灰度级 f_j，$j=0,1,\ldots,k,\ldots,L-1$；

2.统计各灰度级的象素数目 n_j，$j=0,1,\ldots,k,\ldots,L-1$；

3.计算原始图像直方图 $P_f(f_j)=n_j/n$，n 为原始图像总的象素数目；

4.计算累积分布函数 $c(f)=\sum_{j=0}^{m}P_g(g_i),\ i=0,1,\ldots,m,\ldots,L-1$；

5.应用转移函数，计算映射后的灰度级 $g_i=INT[(g_{\max}-g_{\min})c(f)+g_{\min}+0.5]$；

6.统计映射后各灰度级的象素数目 n_i, i=0,1,…,p-1；

7.计算输出图像直方图 $P_g(g_i)=\ n_i/n,\ i=0,1,\ldots,p-1$；

8.用 f_j 和 g_i 的映射关系，修改原始图像灰度级，获得直方图近似均匀分布的输出图像。

②Sobel 算子

主要用作边缘检测，在技术上，它是一离散性差分算子，用来运算图像亮度函数的灰度之近似值。在图像的任何一点使用此算子，将会产生对应的灰度矢量或是其法矢量。

Sobel 卷积因子为：

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

该算子包含两组 3x3 的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以 A 代表原始图像，Gx 及 Gy 分别代表经横向及纵向边缘检测的图像灰度值，其公式如下：

$$G_x=\begin{bmatrix}-1 & 0 & +1\\ -2 & 0 & +2\\ -1 & 0 & +1\end{bmatrix}\times A,\quad G_y=\begin{bmatrix}+1 & +2 & +1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}\times A$$

图像的每一个像素的横向及纵向灰度值通过以下公式结合，来计算该点灰度的大小：

$$G=\sqrt{G_x^2+G_y^2}$$

通常，为了提高效率 使用不开平方的近似值：

$$|G|=|G_x|+|G_y|$$

如果梯度 G 大于某一阀值 则认为该点(x,y)为边缘点。

Sobel 算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息，边缘定位精度不够高。当对精度要求不是很高时，是一种较为常用的边缘检测方法。

③Prewitt 算子

Prewitt 边缘算子是一种边缘样板算子，利用像素点上下，左右邻点灰度差，在边缘处达到极值检测边缘，对噪声具有平滑作用。

卷积因子如下：

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

Prewitt 算子不仅能检测边缘点而且还能抑制噪声的影响，因此对灰度和噪声较多的图像处理比较好。

④Roberts 算子

卷积因子如下：

+1	0
0	-1

Gx

0	+1
-1	0

Gy

Roberts 算子采用对角线方向相邻两像素之差近似梯度幅值检测边缘。检测水平和垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感。

⑤LOG 算子

LOG 算子的基本思想是首先将图像与高斯滤波器进行卷积，这一步即平滑了图像又降低了噪声，孤立的噪声点和较小的结构组织将被滤除。然后利用拉普拉斯算子找出图像中的陡

峭边缘，并且考虑那些具有局部梯度最大值的点。图像的平滑过程减少了噪声的影响，并且抵消了由拉普拉斯算子的二阶导数所引入的噪声影响。

⑥Canny 算子

Canny 边缘检测算法可以分为以下 5 个步骤：

- 使用高斯滤波器，以平滑图像，滤除噪声。

在得到高斯模板宽度的情况下，利用高斯函数生成一维高斯滤波模板。得到一维高斯滤波模板后，利用该一维模板先对原图像进行一次行（x 方向）滤波，然后再进行一次列（y 方向)滤波得到平滑后的图像。

- 计算图像中每个像素点的梯度强度和方向。

得到平滑图像后，利用高斯函数的一阶差分模板分别计算 x 方向和 y 方向的偏导数，然后计算出每个像素点梯度的幅值大小，并对其归一化。

- 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。

通俗意义上是指寻找像素点局部最大值，将非极大值点所对应的灰度值置为 0，这样可以剔除掉一大部分非边缘的点。要进行非极大值抑制，就首先要确定像素点 C 的灰度值在其 8 值邻域内是否为最大。要把当前位置的梯度值与梯度方向上两侧的梯度值进行比较，但实际上，我们只能得到 C 点邻域的 8 个点的值，要得到这两个梯度值就需要对该两个点两端的已知灰度进行线性插值。完成非极大值抑制后，会得到一个二值图像，非边缘的点灰度值均为 0，可能为边缘的局部灰度极大值点可设置其灰度为 128。

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

选择两个阈值（关于阈值的选取方法在扩展中进行讨论），根据高阈值得到一个边缘图像，这样一个图像含有很少的假边缘，但是由于阈值较高，产生的图像边缘可能不闭合，未解决这样一个问题采用了另外一个低阈值。

- 连接边缘。

在高阈值图像中把边缘链接成轮廓，当到达轮廓的端点时，该算法会在断点的 8 邻域点中寻找满足低阈值的点，再根据此点收集新的边缘，直到整个图像边缘闭合。

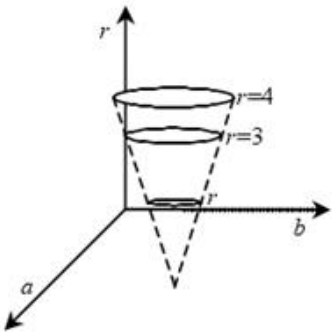
⑦Hough 变换

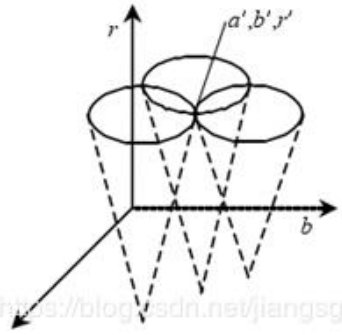
对于一个半径为 r，圆心为（a,b）的圆，可以将其表示为：$(x-a)^2+(y-b)^2=r^2$

某个圆过点（x,y），则有：$(x-a)^2+(y-b)^2=r^2$ 。

那么过点（x,y）的所有圆可以表示为（a(i),b(i),r(i)），其中 r1∈（0，无穷），每一个 i 值都对应一个不同的圆，（a(i),b(i),r(i)）表示了无穷多个过点（x,y）的圆。

图像平面上的每一点就对应于参数空间中每个半径下的一个圆，这实际上是一个圆锥。即（x,y）→（a,b,r），如下图所示。





过点（x1,y1）的所有圆可以表示为（a1(i),b1(i),r1(i)），过点（x2,y2）的所有圆可以表示为（a2(i),b2(i),r2(i)），过点（x3,y3）的所有圆可以表示为（a3(i),b3(i),r3(i)），如果这三个点在同一个圆上，那么存在一个值（a',b',r'），使得 a' = a1(k)=a2(k)=a3(k) 且 b' = b1(k)=b2(k)=b3(k) 且 r' = r1(k)=r2(k)=r3(k)，即这三个点同时在圆（a',b',r'）上。

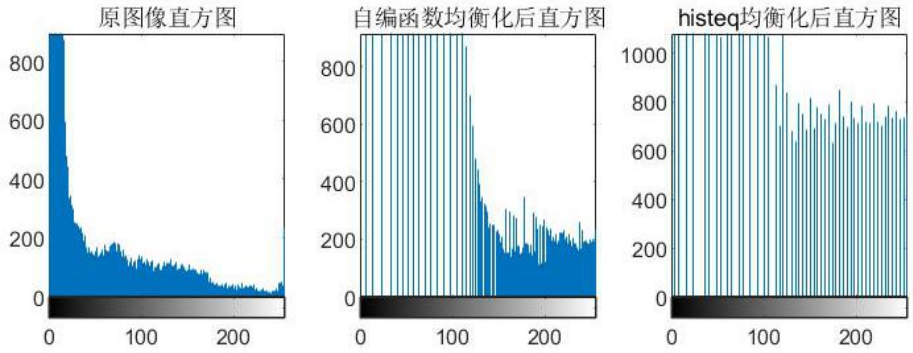
从上图可以形象的看出，三个圆锥面的交点既是同时过这三个点的圆。

上述方法是经典的 Hough 圆检测方法的原理，它具有精度高，抗干扰能力强等优点，但由于该方法的参数空间为三维，要在三维空间上进行证据累计的话，需要的时间和空间都是庞大的。

实验代码和结果：

直方图均衡化

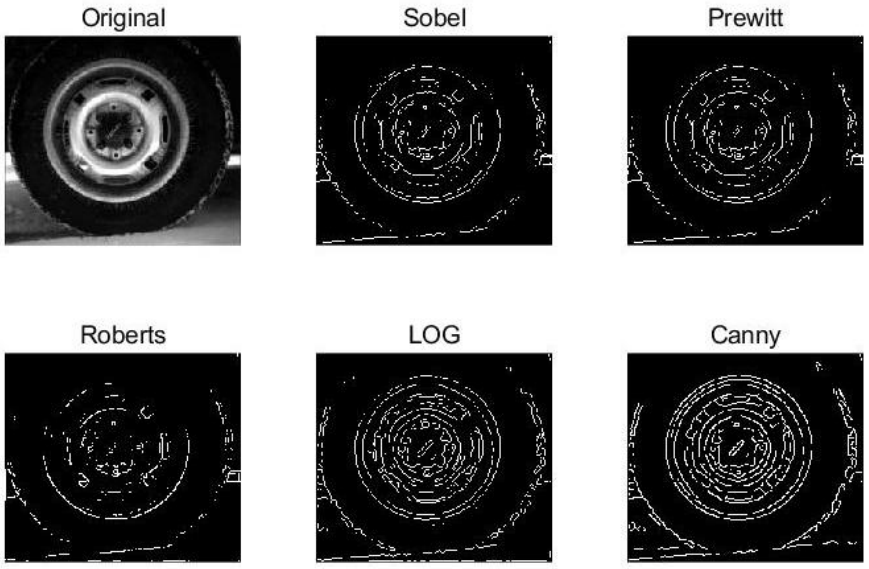
```
clear all
A = imread('plate.jpg');
B = histeq(A);
% [count,~] = imhist(A);
% l = find(count~=0);
% l = min(count(l));
H = zeros(numel(A),1);
hp = 0;
for i = 0:255
    index = find(A == i);
    hp = hp + length(index);
    H(index) = hp/numel(A);
end
Range = max(max(A))-min(min(A));
H = uint8(round(double(Range) * H+double(min(min(A)))+0.5));
H = reshape(H,size(A));

figure(1)
subplot(2,3,1),imshow(A)
title('原图像')
subplot(2,3,2),imshow(H)
title('自编函数均衡化后图')
subplot(2,3,3),imshow(B)
title('histeq均衡化后图')
subplot(2,3,4),imhist(A)
title('原图像直方图')
subplot(2,3,5),imhist(H)
title('自编函数均衡化后直方图')
subplot(2,3,6),imhist(B)
title('histeq均衡化后直方图')
```



可以看出，按照公式编写的出来的直方图与 MATLAB 自带函数的效果不一致，虽然图像上看结果比较相似，应该是 MATLAB 在算法上还有自适应操作。

使用Matlab内部函数edge实现边缘检测算法

```
ED1 = edge(A,'sobel');  % 用Sobel算子进行边缘检测
ED2 = edge(A,'prewitt');  % 用Prewitt算子进行边缘检测
ED3 = edge(A,'roberts');  % 用Roberts算子进行边缘检测
ED4 = edge(A,'log');  % 用LOG算子进行边缘检测
ED5 = edge(A,'canny');  % 用Canny算子进行边缘检测
figure(2)
subplot(2,3,1),imshow(A)
title('Original')
subplot(2,3,2),imshow(ED1)
title('Sobel')
subplot(2,3,3),imshow(ED2)
title('Prewitt')
subplot(2,3,4),imshow(ED3)
title('Roberts')
subplot(2,3,5),imshow(ED4)
title('LOG')
subplot(2,3,6),imshow(ED5)
title('Canny')
```

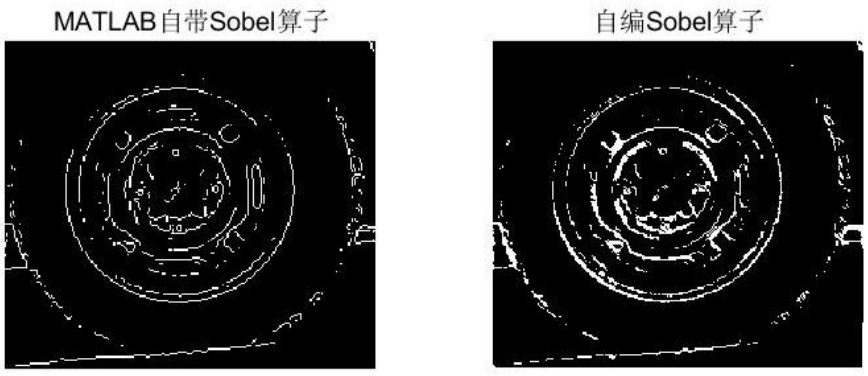


可以看出，Canny 算子的效果最好，边缘提取最为清晰连续，而 Roberts 算子的效果最差，因此在下一步的检测圆操作中，考虑使用 Canny 算子处理过的图像作为输入。

现在根据各算子原理自变函数实现边缘检测并与 MATLAB 自带函数的效果进行比较。

Sobel

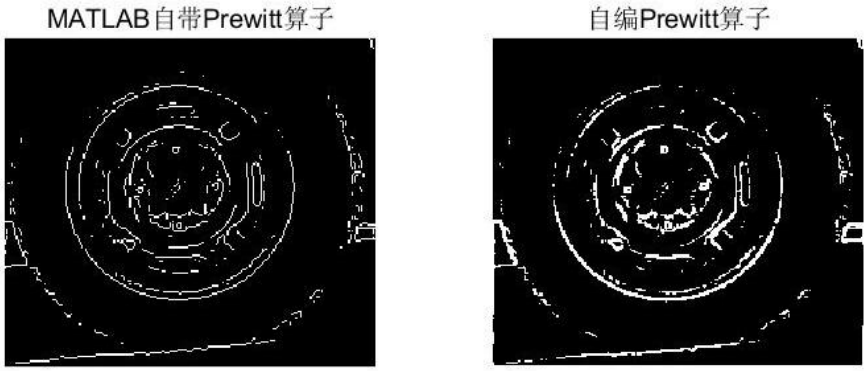
```
s_h = [-1 -2 -1;0 0 0;1 2 1];% Sobel梯度模板
s_gx = conv2(A,s_h,'same'); % 计算图像的Sobel垂直梯度
s_gy = conv2(A,s_h','same'); % 计算图像的sobel水平梯度
s_g = abs(s_gx) + abs(s_gy);  %得到图像的sobel梯度
sobel = A;
sobel(find(s_g > 250)) = 255; % 图像二值化
sobel(find(s_g <= 250)) = 0;
s_se = strel([0,0,0;0,1,0;0,0,1]);
sobel = imerode(sobel,s_se); % 腐蚀
figure()
subplot(1,2,1),imshow(ED1,[])
title('MATLAB自带Sobel算子')
subplot(1,2,2),imshow(sobel,[])
title('自编Sobel算子')
```



一开始单纯做卷积的话，图像的毛刺很多，且边缘不够明晰，因此我进行了腐蚀操作，细化骨架，效果得到了一定的改善。

Prewitt

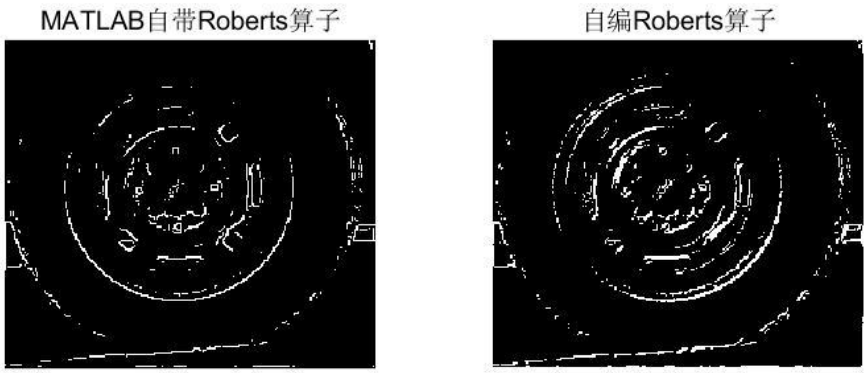
```
p_h = [-1 -1 -1;0 0 0;1 1 1];% Prewitt梯度模板
p_gx = conv2(A,p_h,'same'); % 计算图像的Prewitt垂直梯度
p_gy = conv2(A,p_h','same'); % 计算图像的Prewitt水平梯度
p_g = abs(p_gx) + abs(p_gy);  %得到图像的Prewitt梯度
prewitt = A;
prewitt(find(p_g > 250)) = 255; % 图像二值化
prewitt(find(p_g <= 250)) = 0;
figure()
subplot(1,2,1),imshow(ED2,[])
title('MATLAB自带Prewitt算子')
subplot(1,2,2),imshow(prewitt,[])
title('自编Prewitt算子')
```



除卷积模板外与 Sobel 原理相同，但仍有边缘粗细不一致的现象出现，做腐蚀的效果不好，故不采取。

Roberts

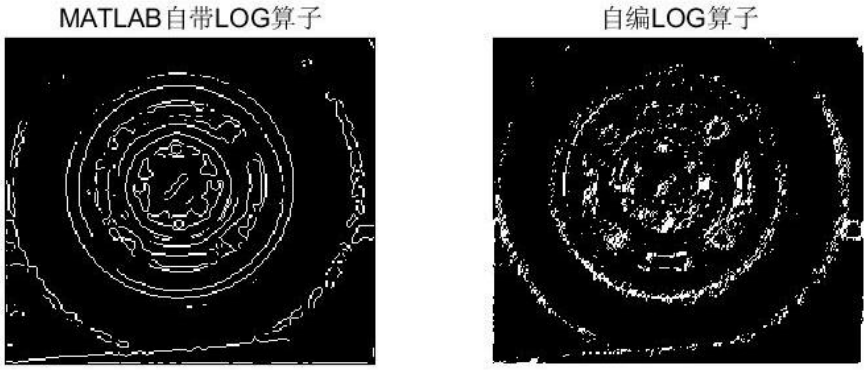
```
r_h = [-1 0;0 1];% Roberts梯度模板
r_gx = conv2(A,r_h,'same'); % 计算图像的Roberts垂直梯度
r_gy = conv2(A,r_h','same'); % 计算图像的Roberts水平梯度
r_g = abs(r_gx) + abs(r_gy);  %得到图像的Roberts梯度
roberts = A;
roberts(find(r_g > 90)) = 255; % 图像二值化
roberts(find(r_g <= 90)) = 0;
figure()
subplot(1,2,1),imshow(ED3,[])
title('MATLAB自带Roberts算子')
subplot(1,2,2),imshow(roberts,[])
title('自编Roberts算子')
```



同上，除卷积模板外原理相同，边缘粗细不一致的现象仍然存在，做腐蚀的效果也不好，故不采取。

LOG

```
l_h = [0 0 1 0 0;0 1 2 1 0;1 2 -16 2 1;0 1 2 1 0;0 0 1 0 0];% Roberts梯度模板
l_gx = conv2(A,l_h,'same'); % 计算图像的Roberts垂直梯度
l_gy = conv2(A,l_h','same'); % 计算图像的Roberts水平梯度
l_g = abs(l_gx) + abs(l_gy);  %得到图像的Roberts梯度
LOG = A;
LOG(find(l_g > 250)) = 255; % 图像二值化
LOG(find(l_g <= 250)) = 0;
l_se = strel([0,0,1;0,1,0;0,1,0]);
LOG = imerode(LOG,l_se); % 腐蚀
figure()
subplot(1,2,1),imshow(ED4,[])
title('MATLAB自带LOG算子')
subplot(1,2,2),imshow(LOG,[])
title('自编LOG算子')
```



进行了一定的腐蚀操作后，毛刺现象依然很严重。

Canny

```
[m n] = size(A);
A = double(A);
% 高斯滤波
c_h = fspecial('gaussian',[5 5]);
canny = imfilter(A,c_h,'replicate');
% sobel边缘检测
c_h = fspecial('sobel');
c_gx = imfilter(canny,c_h,'replicate');    % 求横边缘
c_gy = imfilter(canny,c_h','replicate');     % 求竖边缘
canny = sqrt(c_gx.^2+c_gy.^2);
c_gmax = max(canny(:));
canny = canny/c_gmax; % 归一化
temp = canny;
counts = imhist(canny,64);
high_threshold = find(cumsum(counts) > 0.7*m*n,1,'first') / 64; % 设定非边缘点比例确定高门限
low_threshold = 0.4 * high_threshold;  % 设定低门限为高门限乘以比例因子
% 非极大抑制 & 双阈值
for q = 2 : m-1
    for p = 2 : n-1
        if((c_gx(q,p) <= 0 && c_gx(q,p) > -c_gy(q,p)) ...
                || (c_gy(q,p) >= 0 && c_gx(q,p) < -c_gy(q,p)))  % 0-45度
            d = abs(c_gy(q,p) / c_gx(q,p));
            gradmag = canny(q,p);
            gradmag1 = canny(q,p+1)*(1-d) + canny(q-1,p+1)*d;  % 插值
            gradmag2 = canny(q,p-1)*(1-d) + canny(q+1,p-1)*d;
```

```
        elseif((c_gx(q,p) > 0 && -c_gy(q,p) >= c_gx(q,p)) ...
                || (c_gx(q,p) < 0 && -c_gy(q,p) <= c_gx(q,p)))  % 45-90度
            d = abs(c_gx(q,p)/c_gy(q,p));
            gradmag = canny(q,p);
            gradmag1 = canny(q-1,p)*(1-d) + canny(q-1,p+1)*d;
            gradmag2 = canny(q+1,p)*(1-d) + canny(q+1,p-1)*d;
        elseif((c_gx(q,p) <= 0 && c_gx(q,p) > c_gy(q,p)) ...
                || (c_gx(q,p) >= 0 && c_gx(q,p) < c_gy(q,p)))   % 90-135度
            d = abs(c_gx(q,p)/c_gy(q,p));
            gradmag = canny(q,p);
            gradmag1 = canny(q-1,p)*(1-d) + canny(q-1,p-1)*d;
            gradmag2 = canny(q+1,p)*(1-d) + canny(q+1,p+1)*d;
        elseif((c_gy(q,p) < 0 && c_gx(q,p) <= c_gy(q,p)) ...
                || (c_gy(q,p) > 0 && c_gx(q,p) >= c_gy(q,p)))   % 135-180度
            d = abs(c_gy(q,p)/c_gx(q,p));
            gradmag = canny(q,p);
            gradmag1 = canny(q,p-1)*(1-d) + canny(q-1,p-1)*d;
            gradmag2 = canny(q,p+1)*(1-d) + canny(q+1,p+1)*d;
        end
        if(gradmag >= gradmag1 && gradmag >= gradmag2) % 局部极大值点，保留
            if(gradmag >= high_threshold)    % 大于阈值上界的保留，得到强边界
                temp(q,p) = 255;
            elseif(gradmag >= low_threshold) % 大于阈值下界的保留，得到弱边界
                temp(q,p) = 125;
            else
                temp(q,p) = 0;  % 其余丢弃
            end
        else
            temp(q,p) = 0;  % 非局部极大值点，抑制
        end
    end
end
```

```
num = 0;       % 当前堆栈个数
flag = zeros(80000,2);        % 堆栈
temp_flag = zeros(80000,2);  % 临时堆栈
for q = 2 : m-1      % 高阈值判别，检查高阈值邻域8个方向范围内是否存在低阈值
    for p = 2 : n-1
        if(temp(q,p) == 255)
            canny(q,p) = 255;
            if(temp(q-1,p-1) == 125)
                temp(q-1,p-1) = 255;
                canny(q-1,p-1) = 255;
                if((q-1 > 1) && (p-1 > 1))
                    num = num + 1;
                    flag(num,1) = q-1;
                    flag(num,2) = p-1;
                end
            end
            if(temp(q-1,p) == 125)
                temp(q-1,p) = 255;
                canny(q-1,p) = 255;
                if(q-1 > 1)
                    num = num + 1;
                    flag(num,1) = q-1;
                    flag(num,2) = p;
                end
            end
            if(temp(q-1,p+1) == 125)
                temp(q-1,p+1) = 255;
                canny(q-1,p+1) = 255;
                if((q-1 > 1) && (p+1 < n))
                    num = num + 1;
                    flag(num,1) = q-1;
                    flag(num,2) = p+1;
                end
            end
```

```
            if(temp(q,p-1) == 125)
                temp(q,p-1) = 255;
                canny(q,p-1) = 255;
                if(p-1 > 1)
                    num = num + 1;
                    flag(num,1) = q;
                    flag(num,2) = p-1;
                end
            end
            if(temp(q,p+1) == 125)
                temp(q,p+1) = 255;
                canny(q,p+1) = 255;
                if(p+1 < n)
                    num = num + 1;
                    flag(num,1) = q;
                    flag(num,2) = p+1;
                end
            end
            if(temp(q+1,p-1) == 125)
                temp(q+1,p-1) = 255;
                canny(q+1,p-1) = 255;
                if((q+1 < m) && (p-1 > 1))
                    num = num + 1;
                    flag(num,1) = q+1;
                    flag(num,2) = p-1;
                end
            end
            if(temp(q+1,p) == 125)
                temp(q+1,p) = 255;
                canny(q+1,p) = 255;
                if(q+1 < m)
                    num = num + 1;
                    flag(num,1) = q+1;
                    flag(num,2) = p;
                end
            end
            if(temp(q+1,p+1) == 125)
                temp(q+1,p+1) = 255;
                canny(q+1,p+1) = 255;
                if((q+1 < m) && (p+1 < n))
                    num = num + 1;
                    flag(num,1) = q+1;
                    flag(num,2) = p+1;
                end
            end
        end
    end
end
done = num;  % 完成标志,等于0表示当前连线已完成
while done ~= 0
    num = 0;
    for temp_num = 1:done
        q = flag(temp_num,1);
        p = flag(temp_num,2);
        if(temp(q-1,p-1) == 125)
            temp(q-1,p-1) = 255;
            canny(q-1,p-1) = 255;
            if((q-1 > 1) && (p-1 > 1))
                num = num + 1;
                temp_flag(num,1) = q-1;
                temp_flag(num,2) = p-1;
            end
        end
        if(temp(q-1,p) == 125)
            temp(q-1,p) = 255;
            canny(q-1,p) = 255;
            if(q-1 > 1)
                num = num + 1;
```

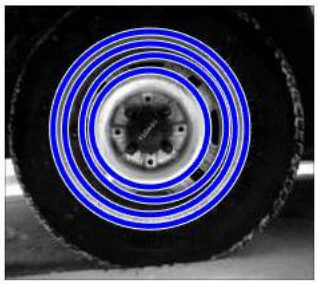
```
                temp_flag(num,1) = q-1;
                temp_flag(num,2) = p;
            end
        end
        if(temp(q-1,p+1) == 125)
            temp(q-1,p+1) = 255;
            canny(q-1,p+1) = 255;
            if((q-1 > 1) && (p+1 < n))
                num = num + 1;
                temp_flag(num,1) = q-1;
                temp_flag(num,2) = p+1;
            end
        end
        if(temp(q,p-1) == 125)
            temp(q,p-1) = 255;
            canny(q,p-1) = 255;
            if(p-1 > 1)
                num = num + 1;
                temp_flag(num,1) = q;
                temp_flag(num,2) = p-1;
            end
        end
        if(temp(q,p+1) == 125)
            temp(q,p+1) = 255;
            canny(q,p+1) = 255;
            if(p+1 < n)
                num = num + 1;
                temp_flag(num,1) = q;
                temp_flag(num,2) = p+1;
            end
        end
        if(temp(q+1,p-1) == 125)
            temp(q+1,p-1) = 255;
            canny(q+1,p-1) = 255;
```

```
            if((q+1 < m) && (p-1 > 1))
                num = num + 1;
                temp_flag(num,1) = q+1;
                temp_flag(num,2) = p-1;
            end
        end
        if(temp(q+1,p) == 125)
            temp(q+1,p) = 255;
            canny(q+1,p) = 255;
            if(q+1 < m)
                num = num + 1;
                temp_flag(num,1) = q+1;
                temp_flag(num,2) = p;
            end
        end
        if(temp(q+1,p+1) == 125)
            temp(q+1,p+1) = 255;
            canny(q+1,p+1) = 255;
            if((q+1 < m) && (p+1 < n))
                num = num + 1;
                temp_flag(num,1) = q+1;
                temp_flag(num,2) = p+1;
            end
        end
    end
    done = num;
    flag = temp_flag;
end
figure()
subplot(1,2,1),imshow(ED5,[])
title('MATLAB自带Canny算子')
subplot(1,2,2),imshow(canny,[])
title('自编Canny算子')
```

总而言之，自编函数的效果都不是非常理想，边缘非常不清晰。

hough变换提取圆

```
figure(3)
imshow(A)
c = []; r = [];
for i = 1:2:9
    [centers, radii] = imfindcircles(ED5,[10*i 10*i+79]); % 寻找指定半径范围内的圆
    c = [c;centers]; % 找出的圆的中心
    r = [r;radii]; % 找出的圆的半径
end
viscircles(c,r,'EdgeColor','b'); % 将找出的圆用蓝色线画出
```



经过一定的参数调整，效果如上，可以找到四个圆。

参考文献：

[1] Find circles using circular Hough transform - MATLAB imfindcircles - MathWorks 中国
https://ww2.mathworks.cn/help/images/ref/imfindcircles.html

[2] canny 算子分析（matlab）
https://blog.csdn.net/wzhwhust/article/details/64925264