作业三 实验报告

姓名：踪浩然　学号：181860158

院系：计算机科学与技术系　邮箱：181860158@smail.nju.edu.cn

实验内容

- 使用不同方法实现边缘检测
- 实现对边缘检测得到的二值图像的边缘连接

实现细节

基本边缘检测方法

首先介绍三种基本边缘检测方法，三种方法虽然使用不同的梯度算子，但基本方法是一致的。

- 使用梯度算子对图像进行卷积，得到图像的梯度
- 对梯度阈值化，将梯度值大于阈值的位置识别为边缘点

1.Robert方法

如下是Robert算子

−1	0	0	−1
0	1	1	0

Roberts

Robert算子相对简单，但对噪声比较敏感，实现代码如下

```
%Robert.m
function output = Robert(input,scale,~)

[m,n]=size(input);
gx=zeros(m-1,n-1);
gy=zeros(m-1,n-1);
for i=1:m-1
    for j=1:n-1
        gx(i,j)=input(i+1,j+1)-input(i,j);
        gy(i,j)=input(i+1,j)-input(i,j+1);
    end
end
g=gx.^2+gy.^2;
thresh=scale*mean(g(:),'double');
output=g>thresh;
end
```

其中阈值的确定是根据图像梯度的平均值的倍数，也就是将梯度大于平均值数倍的点识别为边缘点

2.Prewitt方法

如下是Prewitt算子

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

Prewitt算子在近似图像梯度上要优于Robert算子，代码如下

```
function output = Prewitt(input,scale,~)

[m,n]=size(input);
gx=zeros(m-2,n-2);
gy=zeros(m-2,n-2);
for i=1:m-2
    for j=1:n-2
        gx(i,j)=input(i+2,j)+input(i+2,j+1)+input(i+2,j+2)-input(i,j)-
input(i,j+1)-input(i,j+1);
        gy(i,j)=input(i,j+2)+input(i+1,j+2)+input(i+2,j+2)-input(i,j)-
input(i+1,j)-input(i+2,j);
    end
end
g=gx.^2+gy.^2;
thresh=scale*mean(g(:),'double');
output=g>thresh;
end
```

3.Sobel算子

Sobel算子如下

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel

Sobel算子相较于Prewitt算子有更好的平滑噪声的能力，代码如下

```
function output = Sobel(input,scale,~)
[m,n]=size(input);
gx=zeros(m-2,n-2);
gy=zeros(m-2,n-2);
for i=1:m-2
    for j=1:n-2
        gx(i,j)=input(i+2,j)+2*input(i+2,j+1)+input(i+2,j+2)-
input(i,j)-2*input(i,j+1)-input(i,j+1);
        gy(i,j)=input(i,j+2)+2*input(i+1,j+2)+input(i+2,j+2)-
input(i,j)-2*input(i+1,j)-input(i+2,j);
    end
```

```
end
g=gx.^2+gy.^2;
thresh=scale*mean(g(:),'double');
output=g>thresh;
end
```

高级边缘检测方法

1.Marr-Hildreth边缘检测

该方法的大致步骤如下

- 生成LoG离散滤波器，其中滤波器大小为大于6σ的最小奇数
- 使用滤波器对图像进行卷积
- 寻找零交叉点作为边缘点

代码如下

```
function output = Marr(input,scale,sigma)
[m,n] = size(input);
output = false(m,n);
%生成LoG滤波器
LoG = fspecial('log',ceil(sigma*3) * 2 + 1,sigma);
LoG = LoG - mean(LoG(:));
%卷积
g = imfilter(input,LoG,'replicate');
%设置阈值
thresh = scale * mean(abs(g(:)));
%寻找零交叉点
for i=2:m-1
    for j=2:n-1
        if(g(i,j)*g(i+1,j+1)<0 && abs(g(i,j)-g(i+1,j+1))>thresh)
            output(i,j)=1;
        end
        if(g(i,j)*g(i+1,j)<0 && abs(g(i,j)-g(i+1,j))>thresh)
            output(i,j)=1;
        end
        if(g(i,j)*g(i,j+1)<0 && abs(g(i,j)-g(i,j+1))>thresh)
            output(i,j)=1;
        end
    end
end
end
```

根据课件，零交叉点应当是该点存在两个相对的邻域点满足符号相反且差大于阈值，并且应该能够得到单像素边缘，但我尝试实现之后发现没有得到单像素边缘，因此我的实现里改为该点与某个邻域点满足符号相反且差大于阈值。这样实现后得到了单像素边缘。

2.Canny边缘检测

该方法的步骤大致如下

- 对图像进行高斯模糊（高斯滤波器大小依然选择大于6σ的最小奇数）
- 求图像的梯度大小与方向（用Sobel卷积近似）

- 非最大抑制（将像素与其梯度方向上的两个邻域点比较，如果本像素灰度值最大则保留，否则抑制）
- 滞后阈值（使用两个阈值，从图像中提取强弱边缘点，阈值比值1:2.5左右）
- 连通性分析（强边缘点均保留，弱边缘点只保留那些与强边缘点相邻的）

```
function [output] = Canny(input,low,sigma)
%对图像进行高斯模糊
N = ceil(sigma*3) * 2 + 1;
gauss=Gauss(sigma,N,N);
img= imfilter(input, gauss, 'replicate');
%使用Sobel算子卷积，近似求图像梯度
[m,n]=size(img);
gx=zeros(m-2,n-2);
gy=zeros(m-2,n-2);
for i=1:m-2
    for j=1:n-2
        gx(i,j)=img(i+2,j)+2*img(i+2,j+1)+img(i+2,j+2)-img(i,j)-2*img(i,j+1)-
img(i,j+1);
        gy(i,j)=img(i,j+2)+2*img(i+1,j+2)+img(i+2,j+2)-img(i,j)-2*img(i+1,j)-
img(i+2,j);
    end
end
%梯度大小与方向
M = sqrt(gx.*gx+gy.*gy);
a = atan2d(gy,gx);
dir=zeros(m-2,n-2);
dir((-22.5<=a&a<22.5)|(abs(a)>=157.5))=1;%水平边缘，垂直梯度
dir((-157.5<a&a<=-112.5)|(22.5<=a&a<67.5))=2;%-45度边缘
dir((-112.5<a&a<=-67.5)|(67.5<=a&a<112.5))=3;%垂直边缘，水平梯度
dir((-67.5<a&a<-22.5)|(112.5<=a&a<157.5))=4;%+45度边缘

[m,n] = size(M);
%非最大抑制
gn = zeros(m,n);
for i=2:1:m-1
    for j=2:1:n-1
        switch dir(i,j)
            case 1
                m1=M(i-1,j);
                m2=M(i+1,j);
            case 2
                m1=M(i-1,j-1);
                m2=M(i+1,j+1);
            case 3
                m1=M(i,j-1);
                m2=M(i,j+1);
            case 4
                m1=M(i-1,j+1);
                m2=M(i+1,j-1);
        end
        if(m1<M(i,j)&&m2<M(i,j))
            gn(i,j)=M(i,j);
        end
    end
end
%滞后阈值
low = low*mean(gn(:),'double');
```

```
%阈值比例固定为1:2.5
high = low * 2.5;
gnh=(gn>=high);
gnl=(gn>=low);
gnl = gnl-gnh;

output = gnh;
%连通性分析
for i=2:1:m-1
    for j=2:1:n-1
        if gnl(i,j)>0
            if(gnh(i-1,j-1)||gnh(i-1,j)||gnh(i-1,j+1)||gnh(i,j-
1)||gnh(i,j+1)||gnh(i+1,j-1)||gnh(i+1,j)||gnh(i+1,j+1))
                output(i,j)=1;
            end
        end
    end
end

end
```

边缘连接方法

与库函数所用的方法一致，这里使用的是教材十一章提到的Moore边缘连接算法，其基本思路如下

- 首先从一个边缘像素x出发，顺时针遍历其八邻域，找到其邻域内的第一个边缘点y以及上一个被遍历的不是边缘点的邻域点z，即y，z均为x邻域点，且z不是边缘点而y是边缘点
- 将y加入边缘集合，显然z在y的八邻域内，则从z开始，继续顺时针遍历y的八邻域，找到下一个边缘点
- 以此类推，直到回到最初的边缘像素x

对于简单的单环闭合边缘，这一算法是可行的，但对于非闭合边缘，这里的终止条件存在问题

比如对于一条简单的线段边缘，如果从线段中间某个点（非端点）出发，该算法将向线段一端追踪，到达端点后返回，当回到出发点时结束，显然此时线段另一方向上的边缘点没有被追踪到

因此修改终止条件为：当第二次从同一方向回到出发点时，算法结束

实现代码如下

```
function output = my_edgelinking(input, row, col)
pos = [row,col];
%边缘每个点至多经过两次
img = 2 * input;
%对邻域的标号，顺时针遍历
% [ 2 ][ 3 ][ 4 ]
% [ 1 ][ X ][ 5 ]
% [ 8 ][ 7 ][ 6 ]
neighborhood = [ 0 -1; -1 -1; -1 0; -1 1; 0 1; 1 1; 1 0; 1 -1 ];
exitdir = [ 7 7 1 1 3 3 5 5 ];
%找到出发点，设置初次进入方向
for i=1:8
    tmp = pos + neighborhood(i,:);
    if img(tmp(1),tmp(2))
        initpos = tmp;
        initdir = exitdir(i);
```

```
        output = initpos;
        break;
    end
end
tmpdir = initdir;
%开始遍历追踪边缘
%第二次同方向回到出发点 或 邻域内无可经过边缘点 时终止
flag = true;
while flag
    flag = false;
    for i=1:8
        dir = mod(tmpdir+i-1,8);
        if dir == 0
            dir = 8;
        end
        %当前遍历到的点
        pos = tmp + neighborhood(dir);
        if img(pos(1),pos(2))>0
            %找到边缘点,可经过次数减一
            img(pos(1),pos(2))=img(pos(1),pos(2))-1;
            tmp = pos;
            %确定进入方向
            tmpdir = exitdir(dir);
            output = [output;tmp];
            flag = true;
            if all(tmp==initpos)&&(tmpdir==initdir)
                %同方向进入出发点，结束
                flag = false;
            end
            break;
        end
    end
end

end
```

这里的方向是指上一个被遍历的邻域点在本像素的相对位置（也即是上面提到的z点相对于y点的位置）

比如当前x点顺时针遍历发现4号点为边缘点y，上一个被遍历的非边缘点为3号点z，z在y的1号位上，因此y的进入方向为1，所以`exit_direction(4)=1`，该数组的其他位置以此类推

这里的实现不同于前面提到的算法，对边缘点的经过次数进行了限制，每个边缘点至多经过两次

这是因为，可以看出Moore方法事实上得到的是区域外边缘，但前面进行的边缘检测由于没有进行边缘细化，得到的边缘并不是完全的单像素边缘，因此如果出发点选择了边缘环靠内侧的点，可能会导致一直在外边缘转圈，无法回到出发点。为了尽量避免这一状况，实现时限制每个边缘点至多经过两次，从而防止算法无法终止。

结果

实验设置

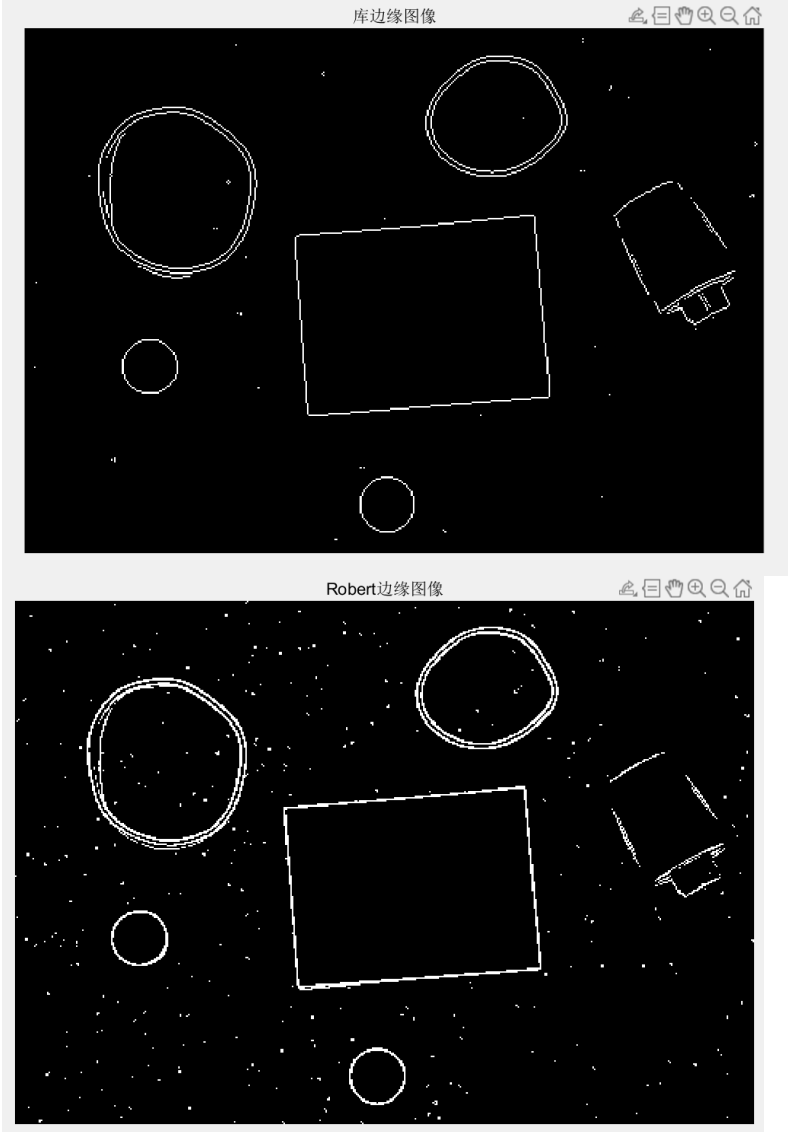
使用软件为MATLAB R2020a，测试脚本为`edge_test.m`，`my_edge.m`为边缘检测函数，根据参数调用不同方法，`my_edgelinking.m`为边缘检测函数

实验结果

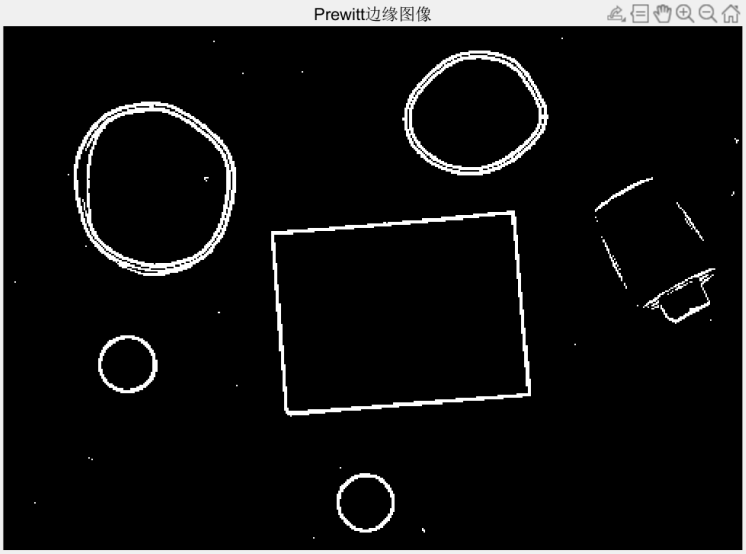
边缘检测

- rubberband_cap.png

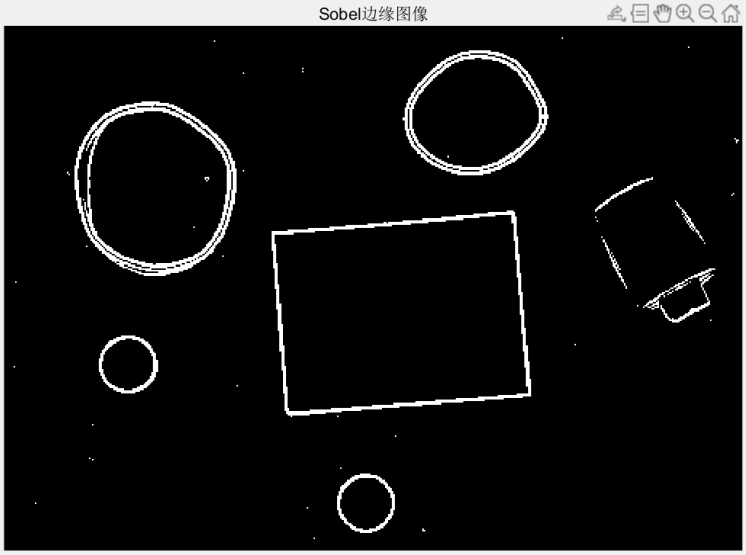
其中基本边缘检测scale=5，Marr方法sigma = 4;scale = 0.5，Canny方法sigma = 2; scale = 4;

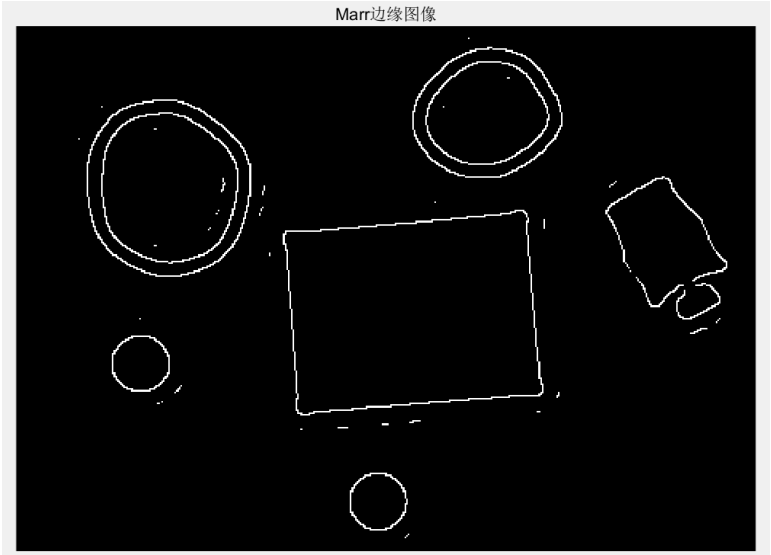


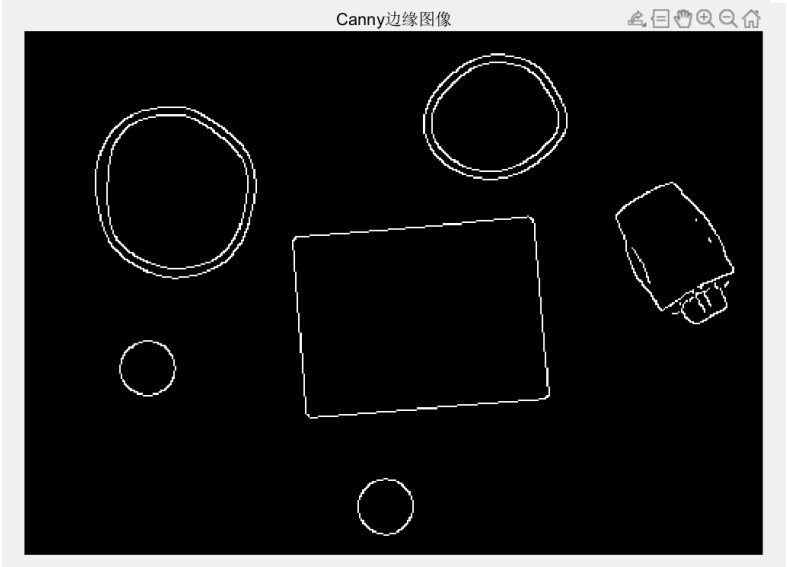
Prewitt边缘图像



Sobel边缘图像







- leaf.jpg

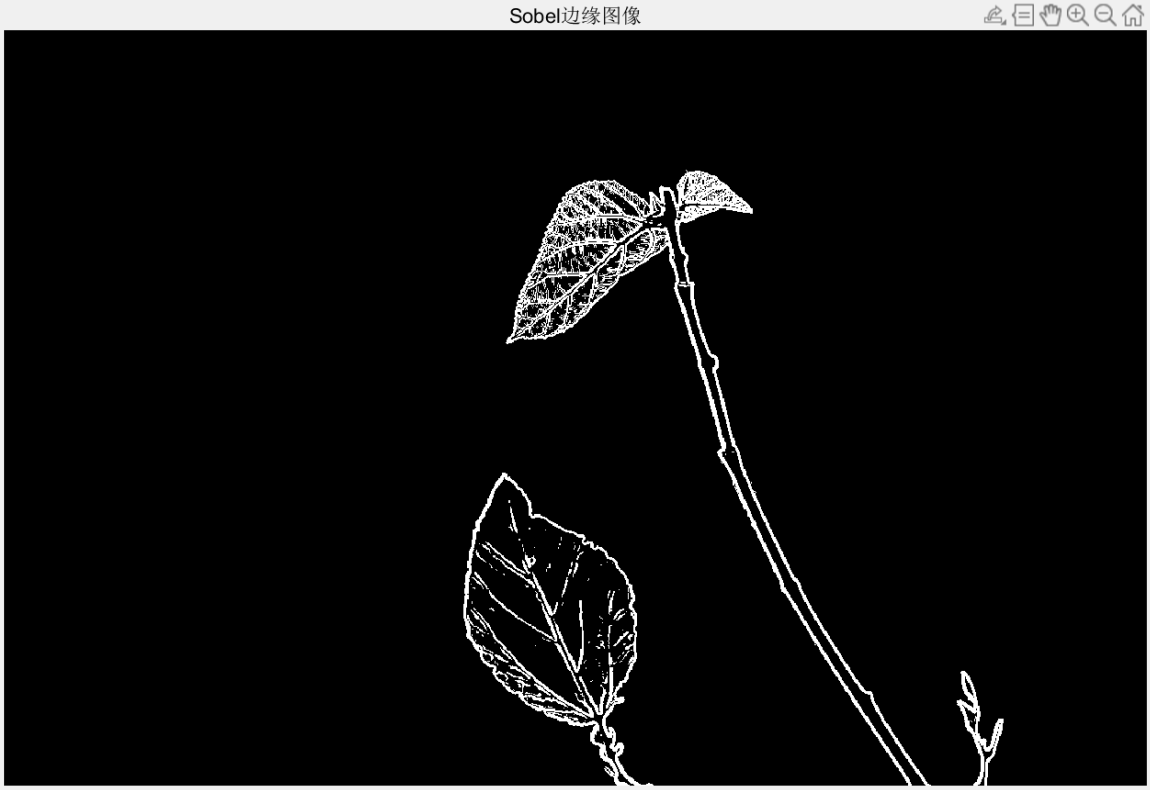
其中基本边缘检测scale=5，Marr方法sigma = 2;scale = 1，Canny方法sigma = 1; scale = 3.5;

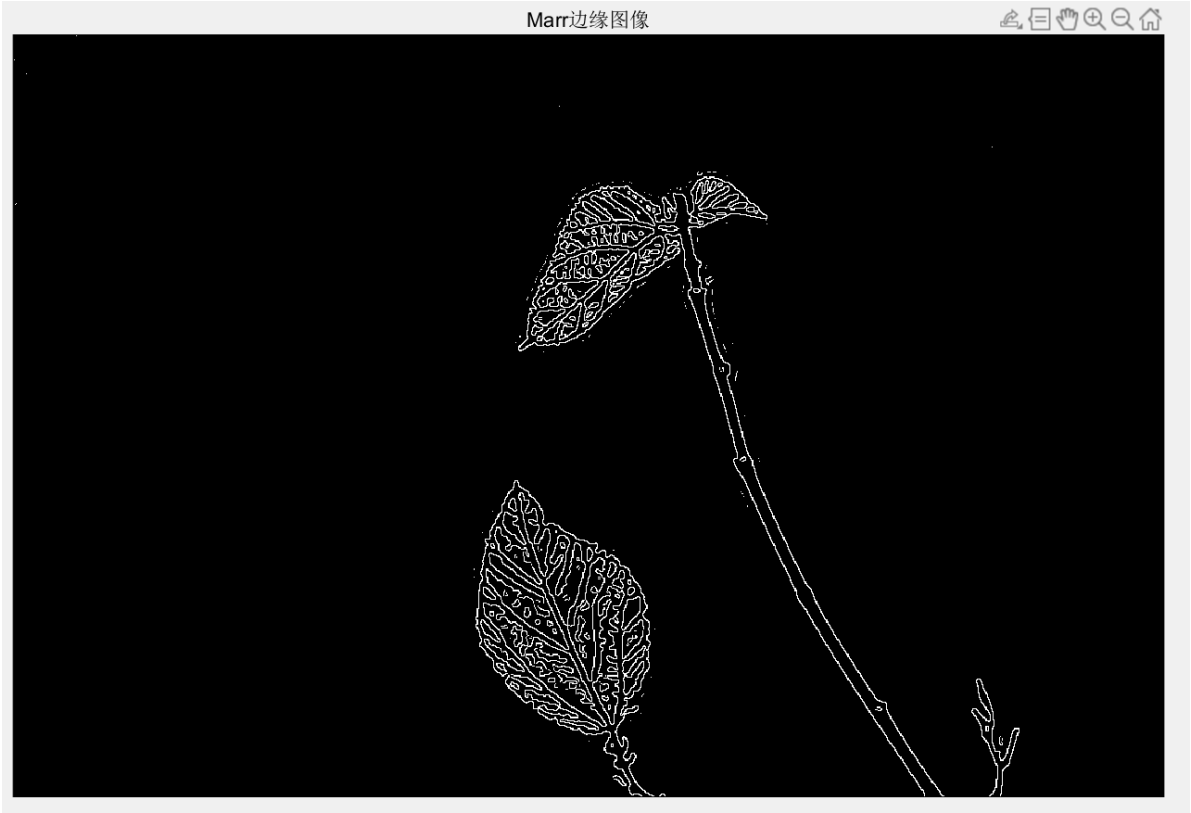
库边缘图像

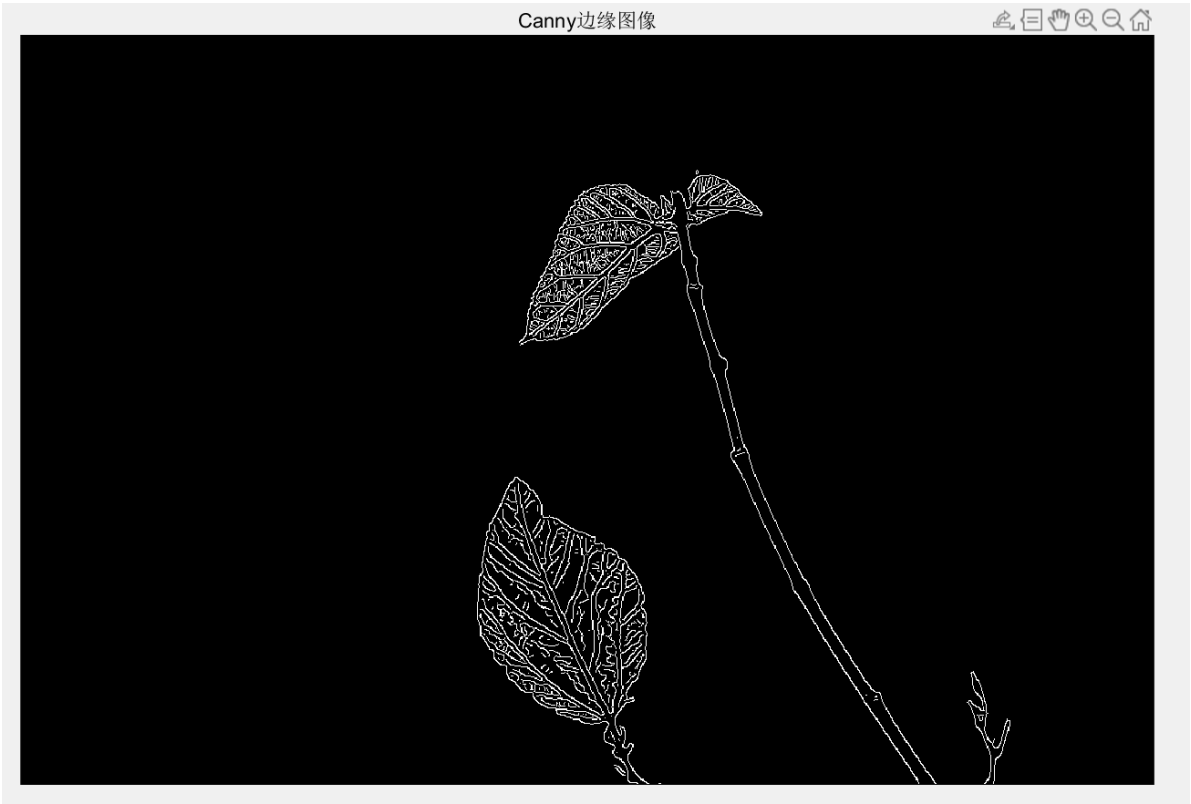
Robert边缘图像

Prewitt边缘图像

Sobel边缘图像







- noise.jpg

其中基本边缘检测scale=5，Marr方法sigma = 3;scale = 0.75，Canny方法sigma = 1; scale = 3;

库边缘图像

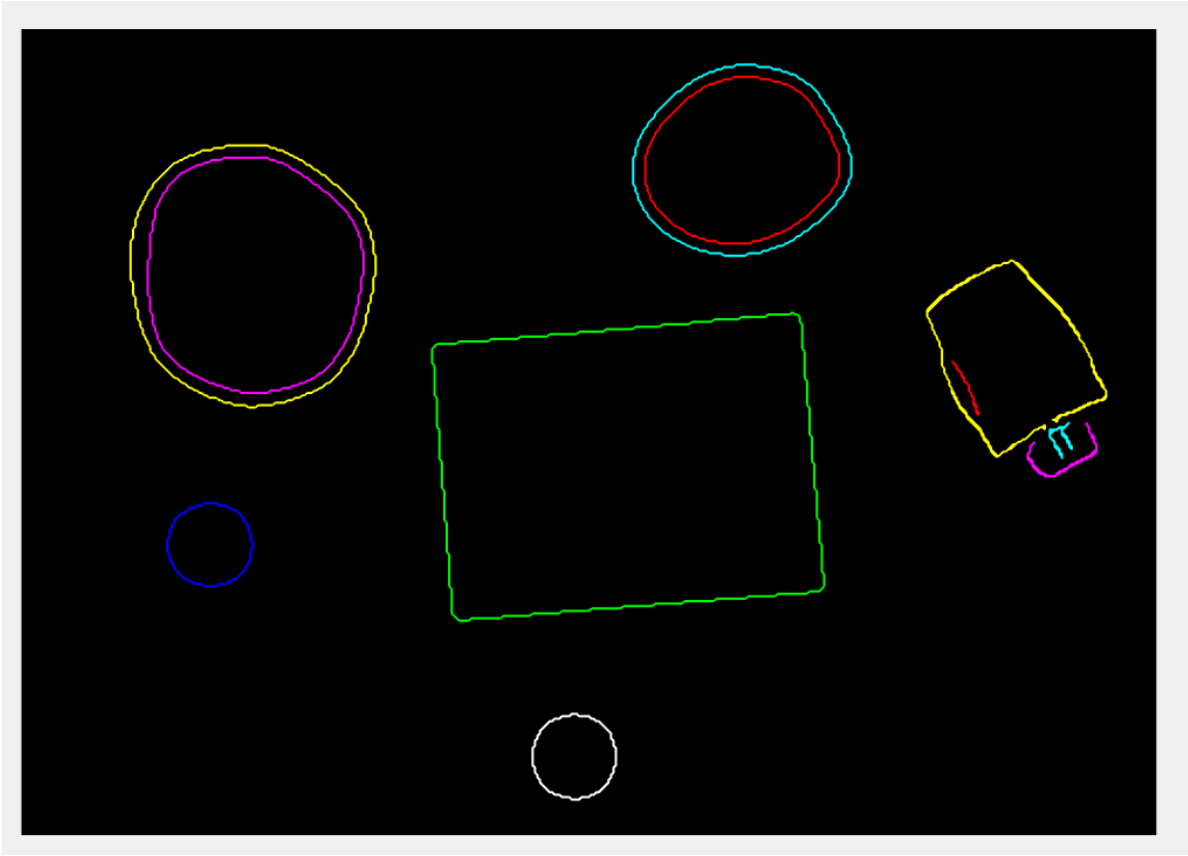
Robert边缘图像

Prewitt边缘图像

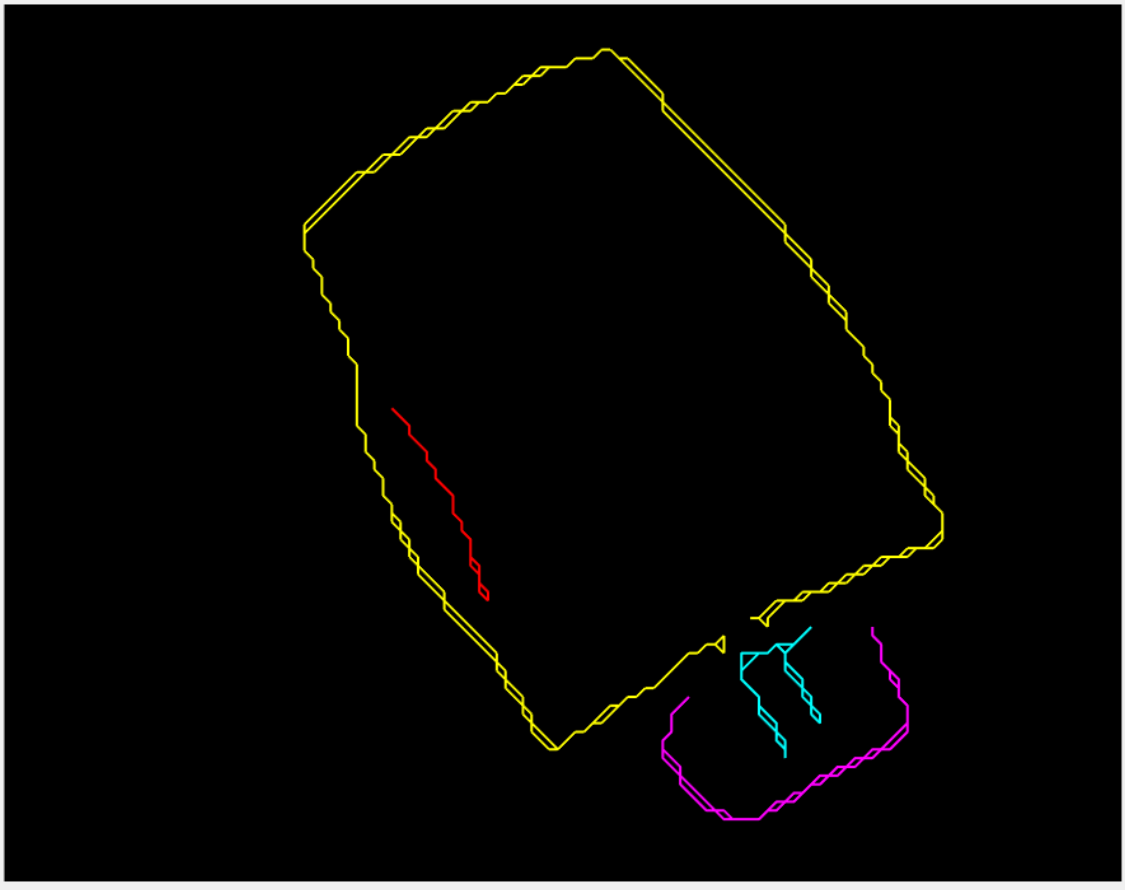
Sobel边缘图像

边缘连接

rubberband_cap.png



其中放大杯子部分的边缘图像



结果分析

边缘检测

从rubberband_cap.png 的结果可以看出，基本边缘检测方法都可以检测出比较明显的边缘，Robert方法由于缺少平滑，所以很难去除噪声，同时这三者都缺少边缘细化的步骤，所以都无法得到单像素的边缘。

库函数默认使用的是Sobel方法，但多了一步边缘细化，可以看出库函数的结果和自己实现的结果除了细化部分以外是比较相近的。

Marr实现了单像素的边缘，但也能看出“意大利通心粉”效应的存在，Canny方法明显优于其他方法，受噪声影响很小的同时保留了绝大部分边缘信息。Canny方法虽然没有实现完全的单像素边缘，但基本接近。

leaf.jpg对应的是基本没有噪声的情况，后两种方法反映了更多的边缘信息，但Marr方法的“意大利通心粉”效应依旧明显，而Canny的效果更好。

noise.jpg中存在有比较明显的噪声，无论是哪种方法都难以去除，但从文字轮廓的保留与其他边缘的体现上，也是Canny方法明显更优。

边缘连接

边缘连接使用的是Canny方法的结果，可以看出不管是简单的连续闭合边缘还是比较复杂的间断边缘，边缘连接都能正常跟踪。Moore方法跟踪的是外边缘，且终止条件是回到出发点，因此跟踪间断边缘时一定会形成一个围绕边缘的环路，从杯子部分的边缘追踪可以看出这一点。