数字图像处理 实验二

一、实验完成情况

- 分别实现了 Roberts 算子，Prewitt 算子，Sobel 算子和 Canny 算子，用以进行边缘检测，并对不同的方法进行了测试和比较。
- 在各种边缘检测方法中，实现了根据输入图像自动选择阈值的功能。
- 实现了边缘链接的方法 my_edgelinking.m

代码运行方法

- 程序的入口为 edge_test.m
- 在此处选择要进行边缘检测的图片：

```
% Load the test image
imgTest = im2double(imread('rubberband_cap.png'));
% imgTest = im2double(imread('bird.png'));
% imgTest = im2double(imread('giraffe.png'));
% imgTest = im2double(imread('lenna.tiff'));
% imgTest = im2double(imread('dva.jpg'));
% imgTest = im2double(imread('moon.jpg'));
% imgTest = im2double(imread('logo.jpg'));
% imgTest = im2double(imread('noise.png'));
% imgTest = im2double(imread('noise2.png'));
imgTestGray = rgb2gray(imgTest);
figure; clf;
imshow(imgTestGray);
```

- 点击“运行”即可，程序将依次输出原图像和使用 Roberts 算子、Prewitt 算子、Sobel 算子及 Canny 算子获得的边缘图像。
- 对于 rubberband_cap.png 图像，下面这段代码将产生边缘链接后的结果。

```
[rows, ~] = size(edgePoints);
for i  = 1 : rows
    Bxpc = my_edgelinking(imgEdge4, edgePoints(i, 1), edgePoints(i, 2));
    hold on
    plot(Bxpc(:,1), Bxpc(:,2), 'w', 'LineWidth', 1);
end
```

二、边缘检测：使用 Roberts 算子，Prewitt 算子与 Sobel 算子

从代码实现的角度来看，这三种方法是很类似的，其实质都是空间域滤波的模板计算，只是各自所选用的模板不同而已。同时，MATLAB 为我们提供了图像的空间域滤波函数 imhist，得益于此，这三种方法的实现就很容易了，如下所示：

```
switch method
    case 'Roberts'
        wx = [0, 0, 0; 0, 1, 0; 0, 0, -1];
        wy = [0, 0, 0; 0, 0, 1; 0, -1, 0];
        scale = 6;
    case 'Prewitt'
        wx = [-1, 0, 1; -1, 0, 1; -1, 0, 1];
        wy = [-1, -1, -1; 0, 0, 0; 1, 1, 1];
        scale = 3;
    case 'Sobel'
        wx = [-1, 0, 1; -1, 0, 1; -1, 0, 1];
        wy = [-1, -1, -1; 0, 0, 0; 1, 1, 1];
        scale = 2;
end
gx = abs(imfilter(input, wx, 'replicate'));
gy = abs(imfilter(input, wy, 'replicate'));
g = gx.^2+ gy.^2;
% Select the threshold automatically
cutoff = scale * sum(g(:), 'double') / numel(g);
output = g > cutoff;
output = bwmorph(output, 'thin');
```

这三种方法都需要指定一个阈值，从而根据各点梯度值和阈值的大小关系确定可能的边缘。然而，对于不同的图像选用同一个阈值显然是不合适的，这可能导致某些图像的边缘检测效果非常差（具体地说，如果选取的阈值过大，则会导致检测出的边缘很不完整；而如果选取的阈值过小，则又会产生大量的“假边缘”）。同时，对于每张图片的每种方法手动指定不同的阈值也是不现实的。这无疑会为函数的调用者增添额外的负担，也不利于后续比较各个方法的效果。

为了解决这一问题，我研读了 MATLAB 内置 edge 函数的源代码。我们知道，edge 函数是不强制要求用户预先输入阈值的，而是可以根据输入的图像自动选择一个阈值，同时还能保证有较好的效果。对于 Roberts 算子，edge 函数中是求取各点的梯度值的平均值，再进行一定的缩放，来作为阈值的。我在 my_edge 函数的实现中沿用了这一做法，具体可见上面的代码。而对于 Prewitt 和 Sobel 算子，edge 函数则是调用某个 mex 库函数来计算阈值，无法看到具体的实现。但是，我沿用了与 Roberts 算子类似的求阈值的方法，经过测试，也能得到不错的效果。

三、边缘检测：使用 Canny 算子

Canny 算子是目前最优的阶梯型边缘检测算法。算法主要包括四个步骤：用高斯滤波器平滑图像，计算各点梯度的幅值和方向，对梯度幅值进行非极大值抑制，用双阈值算法检测和连接边缘。

算法的原理在课堂上已经有了详细的介绍，要做的就是按照步骤逐一进行实现。

高斯滤波和梯度幅值计算比较简单，直接调用 MATLAB 的相关库函数即可，具体请见 my_edge.m 文件，这里不再赘述。

根据各像素点沿 x 方向和 y 方向的梯度值，可以求出梯度的方向。按照梯度的方向将各个像素点划分到四个扇区中，接着进行非极大值抑制，如下所示：

```
    sector = zeros(m, n);
    for r = 1 : m
        for c = 1 : n
            theta = atand(dy(r, c) / dx(r, c));
            if theta >= -22.5 && theta < 22.5
                sector(r, c) = 0;
            elseif theta >= -67.5 && theta < -22.5
                sector(r, c) = 1;
            elseif theta >= 22.5 && theta < 67.5
                sector(r, c) = 3;
            else
                sector(r, c) = 2;
            end
        end
    end
imgTemp = zeros(m, n);
for r = 2 : m - 1
    for c = 2 : n - 1
        switch sector(r, c)
            case 0
                n1 = magGrad(r, c + 1);
                n2 = magGrad(r, c - 1);
            case 1
                n1 = magGrad(r - 1, c + 1);
                n2 = magGrad(r + 1, c - 1);
            case 2
                n1 = magGrad(r - 1, c);
                n2 = magGrad(r + 1, c);
            case 3
                n1 = magGrad(r + 1, c + 1);
                n2 = magGrad(r - 1, c - 1);
        end
        if magGrad(r, c) < n1 || magGrad(r, c) < n2
            imgTemp(r, c) = 0;
        else
            if magGrad(r, c) >= highThresh
                output(r, c) = 1;
                imgTemp(r, c) = highThresh;
            elseif magGrad(r, c) >= lowThresh
                imgTemp(r, c) = lowThresh;
            else
                imgTemp(r, c) = 0;
            end
        end
    end
end
```

非极大值抑制的过程，就是在每一点上，将邻域的中心像素 M 与沿着梯度方向的两个像素相比，如果 M 的梯度值不比两个相邻像素的梯度值大，则令$M = 0$。同时，这里还为双阈值算法做了准备，即：将梯度值小于低阈值的像素点赋值为 0，将梯度值介于低阈值和高阈值之间的像素点赋值为低阈值，对于梯度值大于等于高阈值的像素点，则直接判定为边缘上的点，同时将其梯度值赋值为高阈值。

最后，完成双阈值算法，以连接边缘。做法就是，对于每一个梯度值为低阈值的中心像素，搜索其 8 邻域的像素中是否有梯度值为高阈值的。若有，则将该中心店也判定为边缘上的点。代码如下所示，最终输出的是 logical 类型的二值图像，即边缘检测结果：

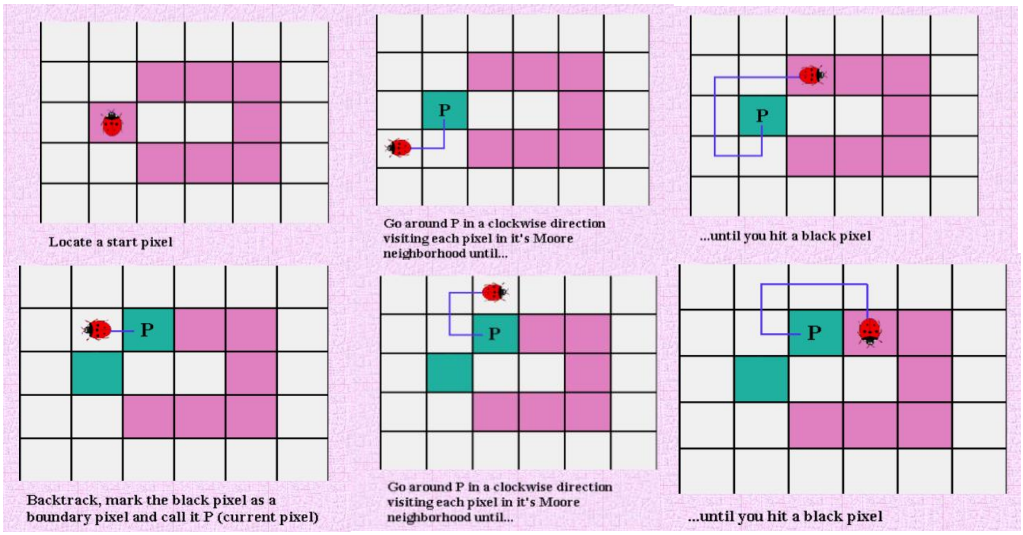
```
for r = 2 : m - 1
    for c = 2 : n -1
        if imgTemp(r, c) == lowThresh
            cmpns = [imgTemp(r - 1, c - 1), imgTemp(r - 1, c), imgTemp(r - 1, c + 1), ...
                imgTemp(r, c - 1), imgTemp(r, c + 1), ...
                imgTemp(r + 1, c - 1), imgTemp(r + 1, c), imgTemp(r + 1, c + 1)];
            if ~isempty(find(cmpns) == highThresh)
                output(r, c) = 1;
            end
        end
    end
end
output = logical(output);
```

四、边缘链接

关于边缘链接（追踪）的内容课堂上没有直接提及，通过查阅网络上的资料，我找到了摩尔邻域(Moore Neighborhood)算法[4]，并进行了实现。

算法的基本思想是：对于输入的二值图像，从选定的黑色像素（假设为 S）出发，沿顺时针方向搜索 S 的摩尔邻域（即 8 邻域）中的像素。当遇到黑色像素时，向进入该像素的方向回退一步，并从此位置开始接着沿顺时针方向进行搜索。上述过程循环进行，直至回到像素点 S。全过程中遇到的所有黑色像素点，即为该二值图像的（其中一条）边界上的点。

下图是摩尔邻域算法的一个形象的演示（来自[5]）：



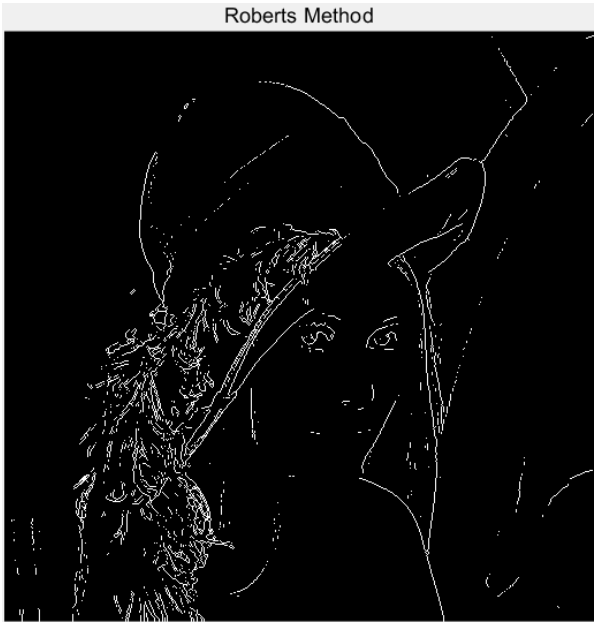
具体实现请见 my_edgelinking.m（实现过程参考了[6]中的代码）。

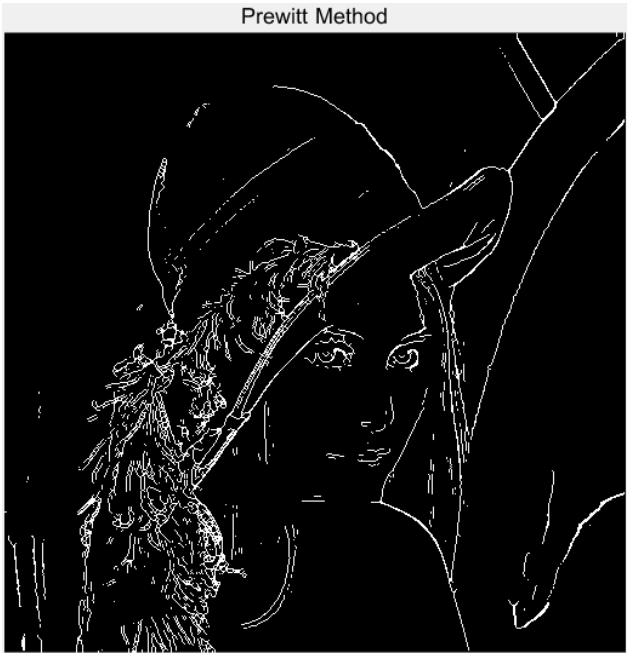
五、实验结果展示

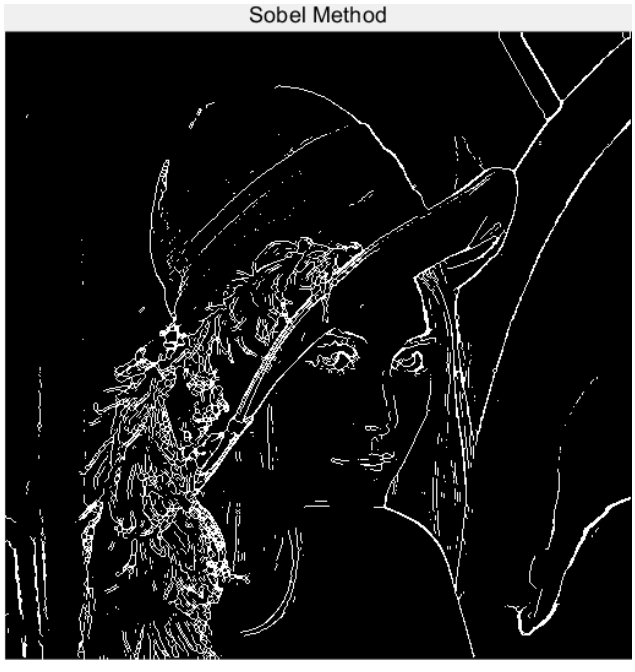
（1）边缘检测

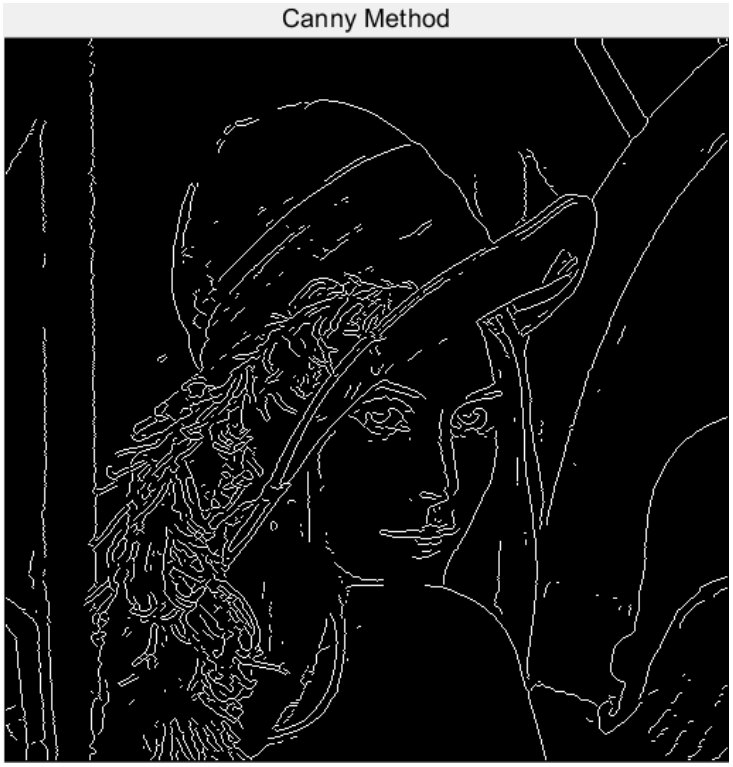
为了验证各种方法的结果，我选取了多个图像进行验证。下面将部分结果展示如下（顺序依次是原图像、Roberts 算子的结果、Prewitt 算子的结果、Sobel 算子的结果和 Canny 算子的结果）：

- **lenna.tiff**

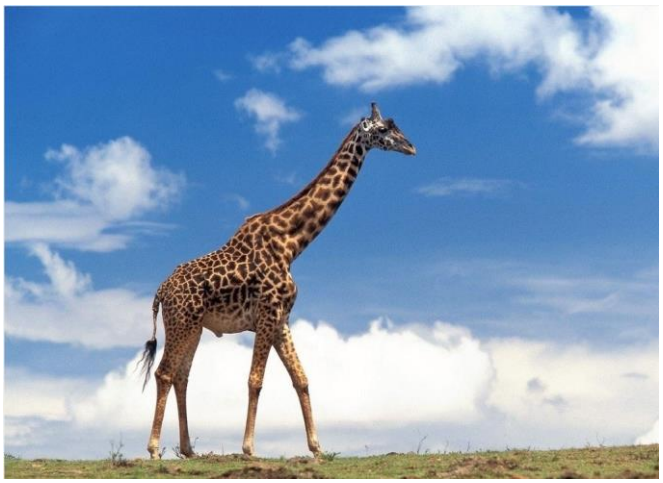


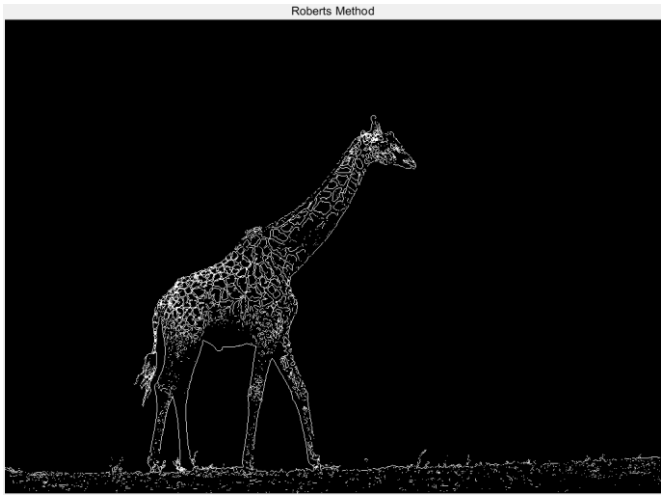


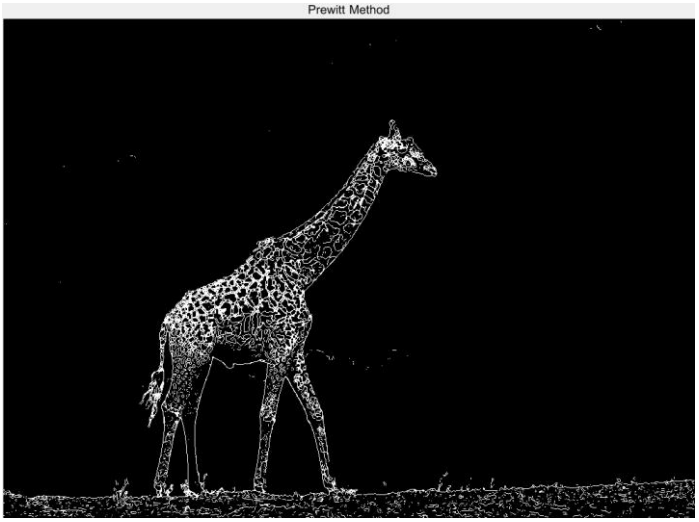


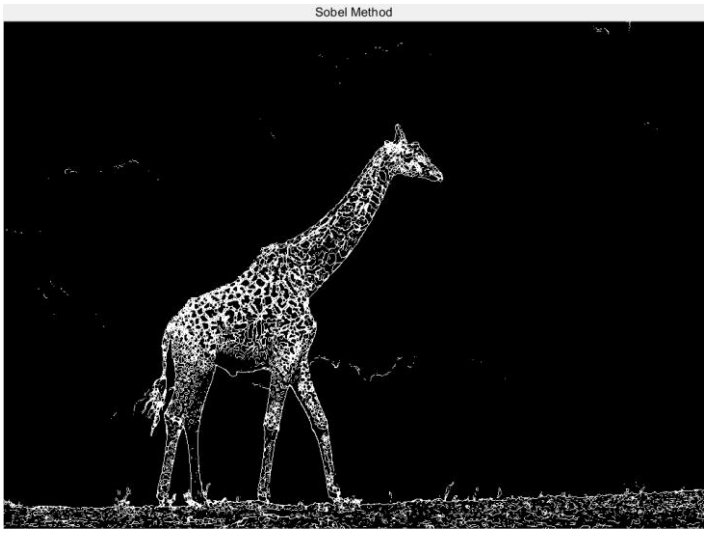


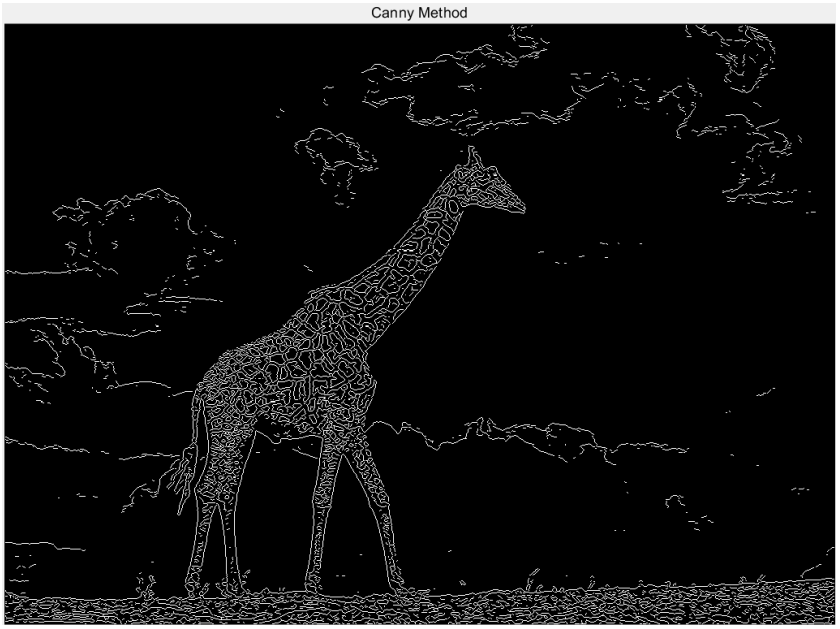
- **giraffe.png**



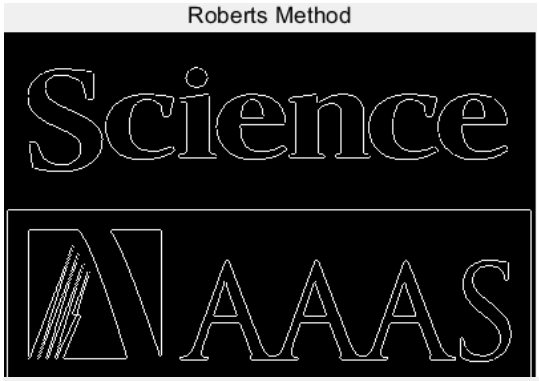


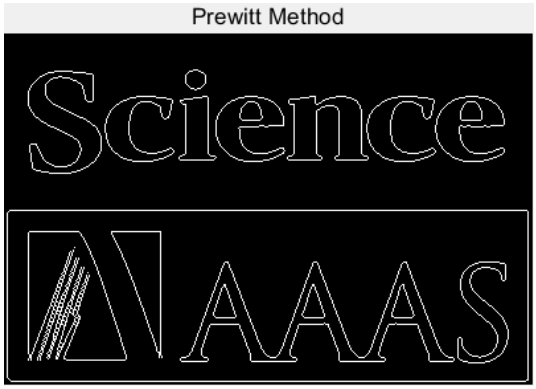

Prewitt Method

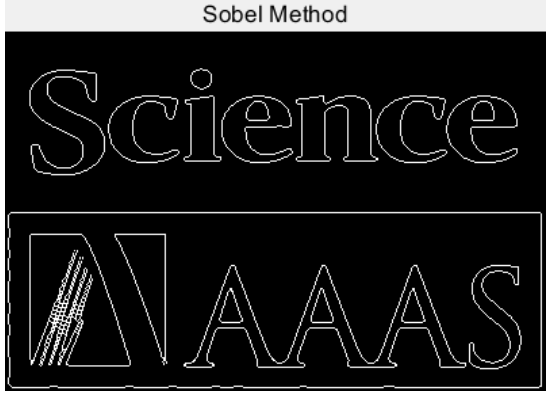

Sobel Method

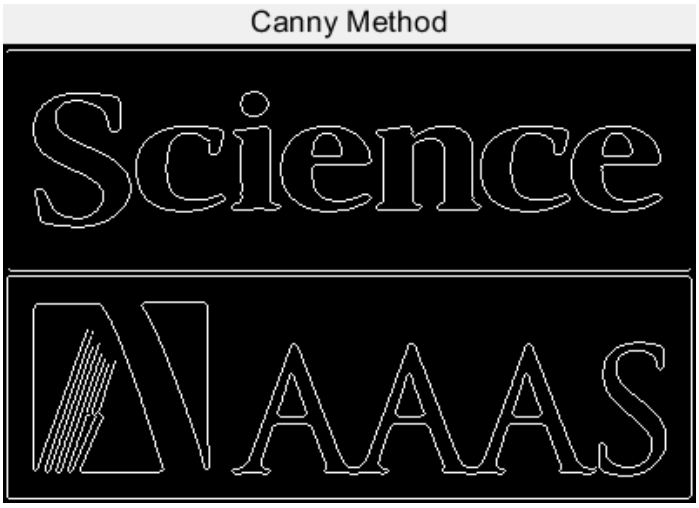

Canny Method

- **logo.jpg**

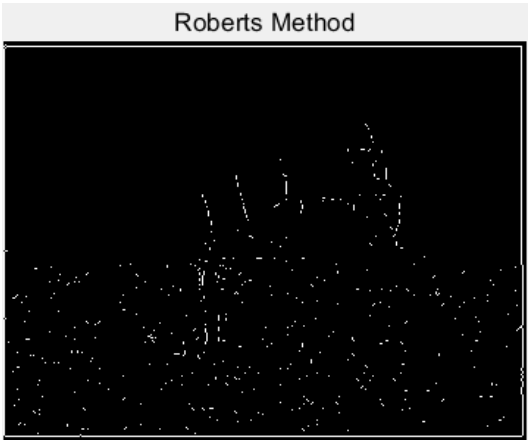


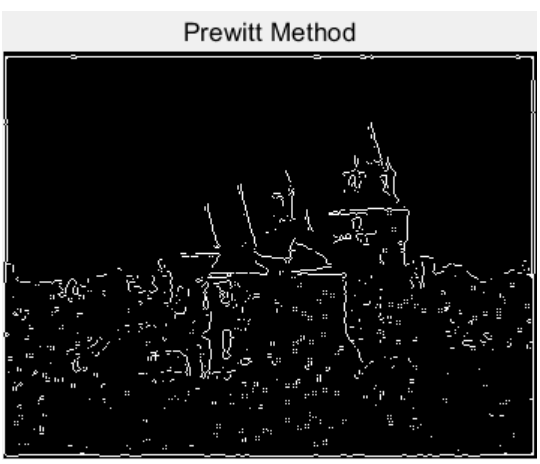


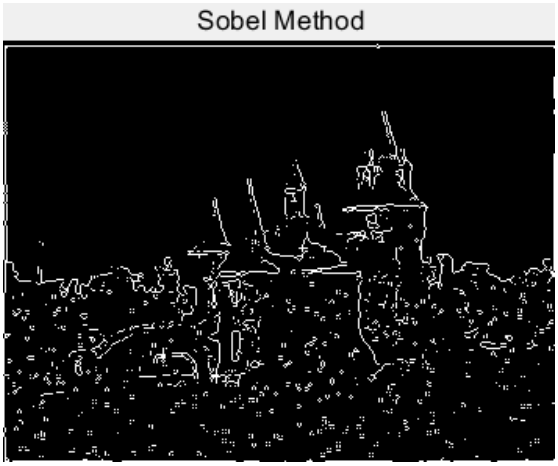


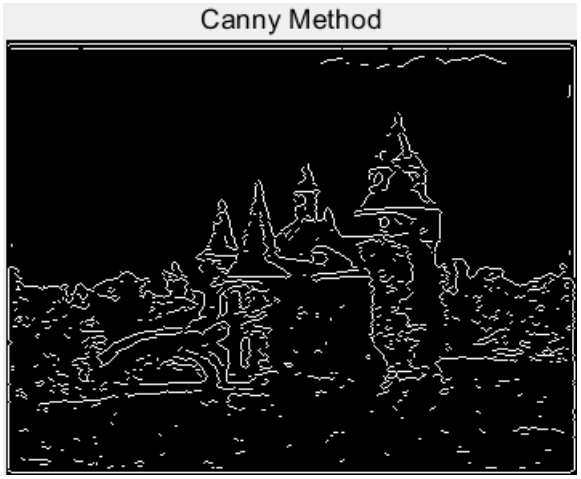


- **noise.png**

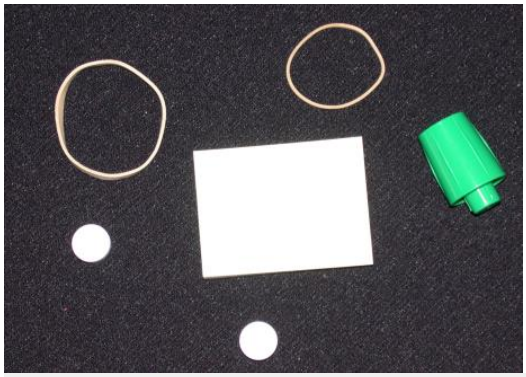


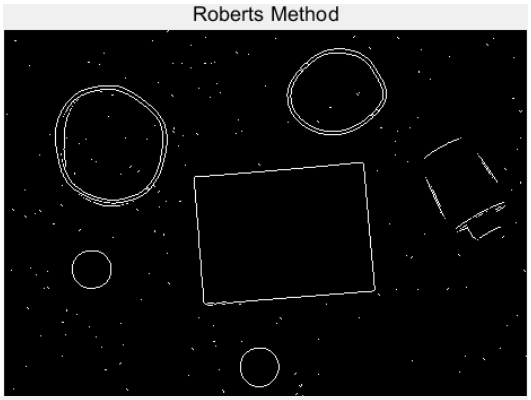


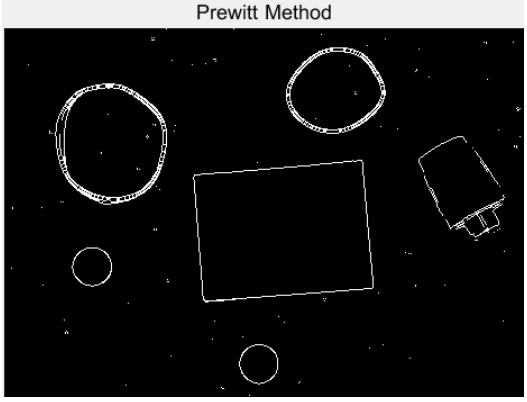


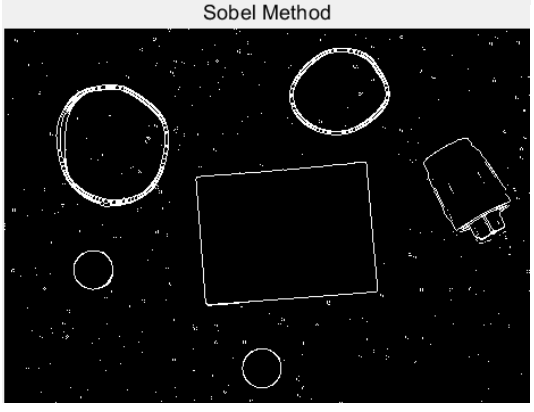


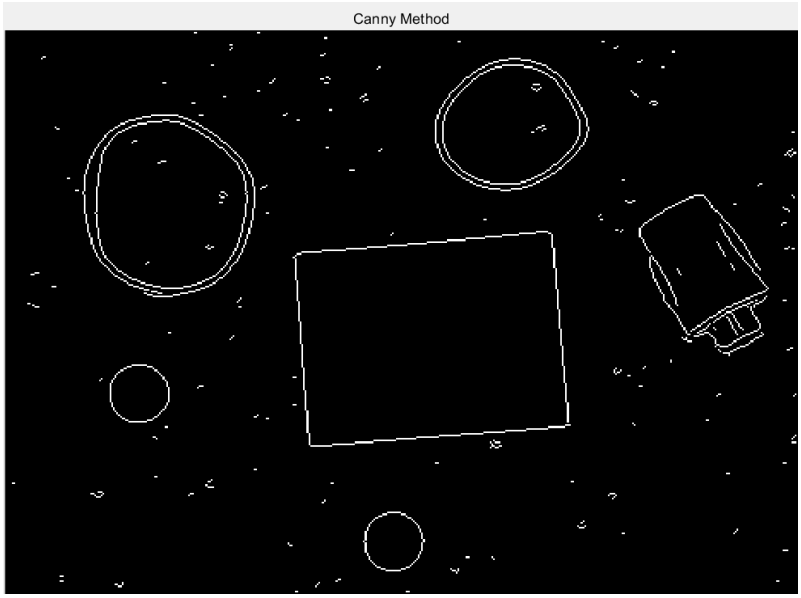
- rubberband_cap.png







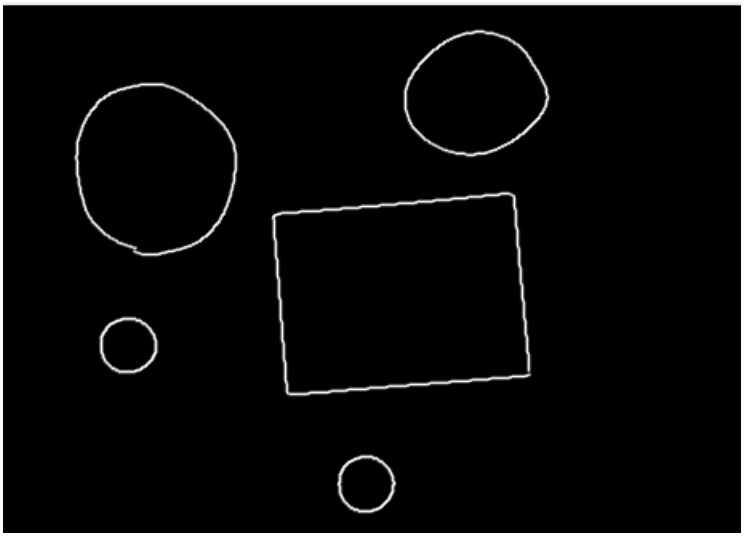




限于报告的篇幅，我这里仅选取了若干比较有代表性的图像结果进行展示。还有更多的图片存放在 asset/image 文件夹中供测试用。

（2）边缘链接

使用 MATLAB 库中的 imtool 工具，可以确定图像中各个对象的边缘点的位置，以此为参数调用 my_edgelinking 函数可以得到下面的结果：



从结果中可以发现，对于边缘间断较少的对象，边缘链接的效果比较好，而对于边缘间断很多的对象（如原图中的绿色物体），则无法得到完整的边缘（没有在图中展示出来）。

六、分析与结论

从上面的结果比较中我们可以看到：

1) 对于构图简单、边缘明显的图像（如 logo.jpg），各种方法都能获得比较好的结果。

2) 而对于构图比较复杂的图像（如 lenna.tiff，giraffe.png 等），不同的方法获得的效果就有了差别。在简单的利用模板进行空间域滤波的方法中，Sobel 算子有着最好的结果。而 Canny 算子则要更优于这些简单的方法，相应的，其实现也更为复杂。

3) 对于含有大量噪声的图像（如 noise.png），Canny 算子的效果要明显优于其他方法。

Reference

1. MATLAB 内置函数 edge.m 源码
2. Gonzalez R , Woods R , Eddins S , et al. 数字图像处理的 MATLAB 实现[M]. 清华大学出版社, 2013.
3. https://www.mathworks.com/matlabcentral/fileexchange/46859-canny-edge-detection
4. https://en.wikipedia.org/wiki/Moore_neighborhood
5. http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/ref.html
6. https://ww2.mathworks.cn/matlabcentral/fileexchange/42144-moore-neighbor-boundary-trace