作业 2：图像边缘检测和边缘链接

刘潇远 161220083 liuxy@smail.nju.edu.cn QQ:1920403010

(南京大学 计算机科学与技术系，南京 210093)

1 实现细节

（阐述清楚所实现的方法即可）

1.1 边缘检测

图像处理中的边缘检测就是把一幅图片的边缘给提取出来，其核心操作就是一个卷积的过程，即建立一个卷积核，与原图的每个像素点及其邻域进行卷积操作，且此邻域尺寸与小矩阵尺寸大小相同；同时这个卷积核还得具有与在图片中图像轮廓边缘上的像素点及其邻域作卷积运算时，卷积运算为一个较大的值，并把该值作为一个新图像中相同位置的像素点的值；而对于图像中的平滑区域，卷积核与这其中的像素点邻域乘积应为零或为较小值；这样，输出的图像便是一幅原图像边缘部分被像素值较大，其他区域像素值较小或为零的灰度图像。

由于一幅待处理图像实际上可看作一个常量，所以边缘检测结果好坏的关键在于与之进行运算的卷积核的构造，这样的卷积核也被称为算子、滤波器、模板。对于一幅图像来说，边缘区域肯定是几个像素点间的像素值相差较大的区域。若一幅图像可由二元离散函数 f（x，y）表示，对 f（x，y）求导，由于 f（x，y）是离散的，对其求导就是对其进行差分，得到的值越大，说明在原图像中该处的边缘越明显，反之若求导值为 0 或很小，说明此处为平滑规整图像区域。对于二元离散函数 f（x，y）求导，用差分来代替求导，可表示为下列等式：

$$df(x,y)=\frac{\partial f(x)}{\partial x}+\frac{\partial f(y)}{\partial y}=f(x+1,y)-f(x,y)+f(x,y+1)-f(x,y)$$

在此基础上，衍生下述多种算法，具体每个算法的实现在后面进行阐述，现阐述实现过程，不同算法的实现上的区别，大致就在于卷积核的选择不同。算法伪代码如下

（1）确定 x 方向和 y 方向上的卷积核

（2）对图像中每个像素及其邻域（超出部分补 0）

（2.1）进行卷积操作

（2.2）根据阈值判断该点是否为边缘

（2.2.1）如果大于阈值，该点是边缘，将其写入输出图像

（3）输出图像

对于 Laplacian 和 Canny 算子，后面会详述与该伪代码的区别

1.1.1 Roberts 算子

Roberts 算子是边缘检测算子中最古老的一种，也是最简单的一种，在部分功能上的限制. Roberts 算子是非对称的，且不能检测 45°倍数的边缘所以这种检测算子的使用明显少于其他几种算子，其形式如下

x 方向卷积核：

0	0	0
0	1	0

0	0	-1

y 方向卷积核：

0	0	0
0	0	1
0	-1	0

1.1.2 Prewitt 算子

Prewitt 算子粗暴地改进了 Roberts 算子的缺点，将邻域内所有点都纳入计算，尽管依旧 naïve，但是总算比 Roberts 算子取得了一点点更好的效果

x 方向卷积核

-1	-1	-1
0	0	0
1	1	1

y 方向卷积核

-1	0	1
-1	0	1
-1	0	1

1.1.3 Sobel 算子

Sobel 只是 Prewitt 的一个扩展，采用了加权的思想，考虑到离中心像素点的距离对于中心像素点梯度贡献值不同，越靠近中心点，贡献越大，所以 Sobel 算子将曼哈顿距离为 1 的点其权重设为 2，距离为 2 的点权重为 1。改进后的 Sobel 算子如下

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

-1	0	1
-2	0	2

-1	0	1

1.1.4 Laplacian 算子

Lapacian 算子是对前面三种算子的全面改进，Laplacian 算子基于以下两条原理：

（1）灰度变化与图像尺寸无关，因此他们的检测要求使用不同尺寸的算子；

（2）灰度的突然变化会在一阶导数中引起波峰或波谷，或在二阶导数中等效地引起零交叉。

根据以上两条原理，好用的边缘检测算子应当具有以下两个特点：

（1）能计算图像中每一点处的一阶导数或二阶导数的数字近似的微分算子；

（2）能被“调整”以便在任何期望的尺寸上起作用。

而满足以上两个特点的滤波器高斯拉普拉斯：

$$\nabla^2 G$$

$$\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$$

这是因为：

（1）算子的高斯部分会模糊图像，从而在尺寸上将结构的灰度（包括噪声）降低到远小于的程度。和均值滤波器平滑相比，高斯函数在空间和频率两个域平滑图像，因而在原图像中引入不存在的人为干扰）的可能性小。

（2）一阶导数用于检测灰度突变，但它们是有方向的算子，而二阶的拉普拉斯算子各项同性，因此对任何模板方向的灰度变化都有相等的响应，从而避免了使用多 个模板去计算图像中任何点处的最强响应。

由于拉普拉斯算子计算过程实质为对 f（x，y）求二次导，这对于强对比边缘来说具有更好的检测作用，但是无疑也对噪声更加敏感，因此在进行拉普拉斯运算之前应当先对图像进行高斯滤波以平滑图像。显然，拉普拉斯适用于检测较强对比度的图像边缘，而对于灰度变化相对缓慢的过渡区域边缘，其检测结果可能为一条较粗的边缘线或者一个封闭狭长的区域，所以对于正常未被高对比度处理的图像来说，基于拉普拉斯边缘检测算子处理之后的边缘都存在过零点现象，我们便可通过检测过零点来检测图像的边缘。

由此，拉普拉斯算子边缘检测伪代码如下：

（1）对输入图像进行高斯滤波

（2）采用如下拉普拉斯卷积核

0	1	0
1	-4	1
0	1	0

或

1	1	1
1	-8	1
1	1	1

（3）对图像中每个像素及其邻域（超出部分补 0）

（3.1）进行卷积操作

（3.2）根据阈值判断该点是否为边缘

（3.2.1）如果大于阈值，该点是边缘，将其写入输出图像

（4）输出图像

1.1.5 Canny 算子

Canny 算子是目前最有效的边缘检测算子。

首先对输入的灰度图像进行高斯平滑滤波。高斯滤波实现方法有两种：离散化窗口滑动卷积、傅里叶变换。前者易实现，所以我用的就是滑动窗口卷积。离散化窗口滑动卷积主要利用高斯核实现，即一个奇数大小的高斯模板，在这里我选择的是 5*5 的窗口大小，这个选择是根据尝试得出的，5*5 的窗口大小能使 Canny 算子最终成像更加稳健。

第二步是对图像进行卷积，标准的 Canny 算子卷积使用的是下述模板

x 方向：

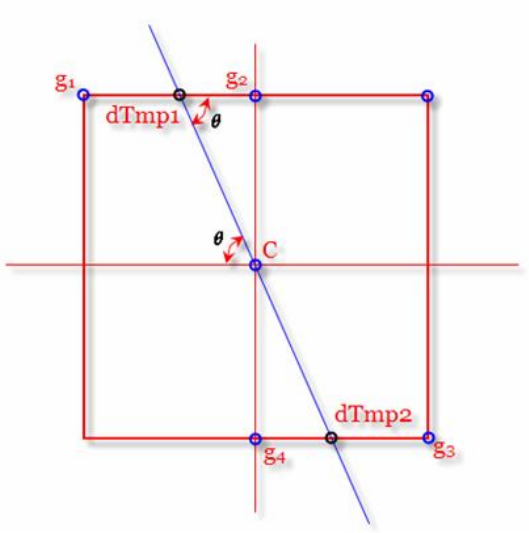
-1	1
-1	1

y 方向：

1	1
-1	-1

但是我考虑到最开始的三种算法，Sobel 是效果最好的，因此在 Canny 算子中采用 Sobel 算子进行卷积，应当能够进一步提高 Canny 算子的效果，因此这一步我采用的 Sobel 算子进行卷积运算。

第三步是非极大值抑制。在上面求出的梯度幅值矩阵中，值越大的元素代表其梯度越大，但它不一定是边缘像素，因此需要进行非极大值抑制，也就是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。如下图所示（图源网络）



若要判定 C 点是否为 8 邻域内最大梯度值点，只需要判断 C 是否比 C 的梯度方向上 dTmp1 和 dTmp2 点

大，是则保留，不是则将 C 点灰度值置 0。而 dTmp1 和 dTmp2 的梯度值可以通过插值得到。

最后一步是双阈值检测。上面得到的边缘有些为假边缘，且有边缘断裂的问题，如果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，因此还要采用一个低阈值，当到达轮廓的端点时，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。因此程序需要两个参数，一个是高阈值，一个是低阈值，对于梯度高于高阈值的像素点，其为真边缘，将其选为边缘，对于梯度介于高阈值和低阈值的像素点，其为若边缘，需要对其继续进行判断，以消除噪音等造成的孤立若边缘，判断方法是检测其 8 邻域内是否有点为高阈值，若有，则将该点也判为边缘，而对于梯度低于低阈值的点，判为非边缘。伪代码如下：

（1）对输入图像进行高斯滤波

（2）对图像进行 Sobel 卷积

（3）对图像的卷积结果，比较每个像素点其与梯度方向上两个相邻像素点的卷积值，如果该像素点卷积值是最大的，则保留该像素点的卷积值，否则将其进行抑制，即将其卷积值置 0。

（4）对非极大值抑制后的卷积结果进行双边缘检测

（5）输出图像

1.1.6 高斯降噪实现

高斯降噪采用 matlab 自带的卷积核生成函数 fspecial，生成高斯卷积核，再使用 mablab 自带的二维卷积函数 filter2 进行卷积

1.2 边缘链接

边缘链接采用的简单的启发式算法，对于输入的二值图像，首先寻找边缘像素点，之后从该像素点开始，采用深度优先搜索找出其所在连通分支（8-连通）的全部边缘点后输出（注意，我实现的边缘检测函数的末尾，采用 matlab 自带的 logical 函数，对算法输出进行了二值化，可以直接拿来进行边缘链接，但是为了排除我的边缘检测函数与库函数不一致可能带来的问题，我最终采用了库函数的边缘识别图像，将这个图像分别使用库函数 bwtraceboundary 和我的函数 my_edgelinking 进行边缘链接）。

2 结果

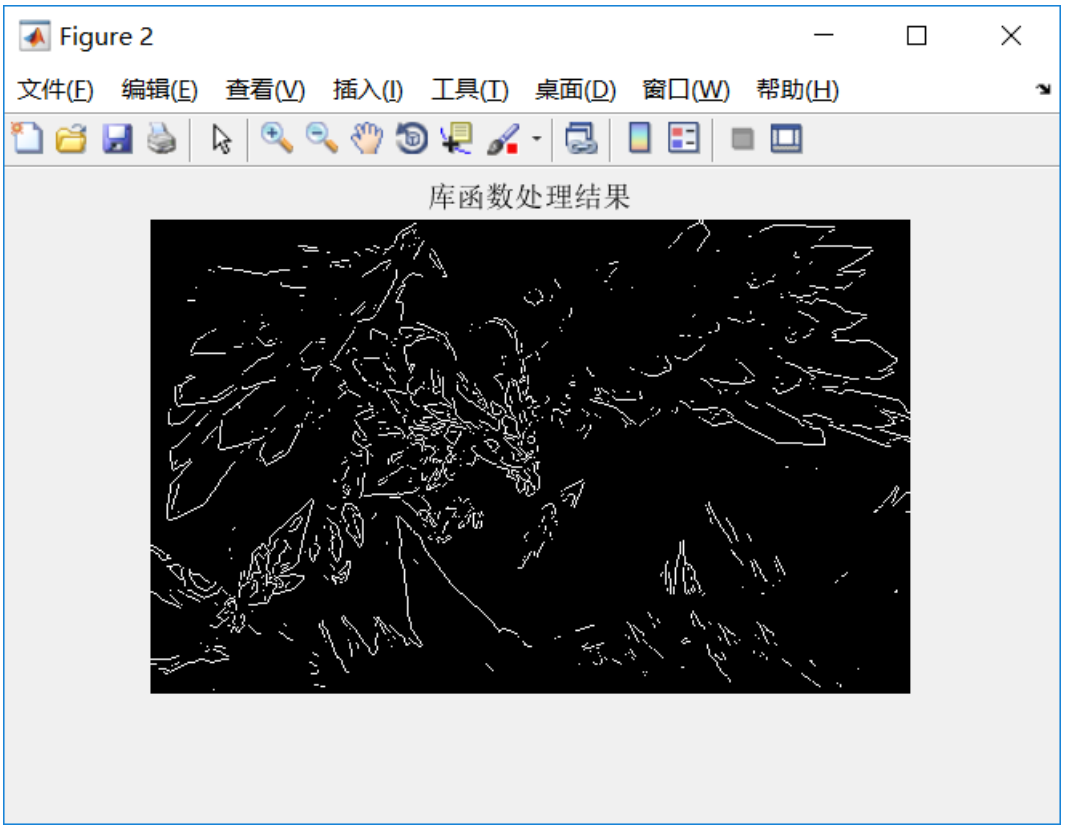
2.1 实验设置

我对五种算子进行了调参，Roberts 算子阈值设置为 0.12，Prewitt 算子的阈值设置为 0.3，Sobel 算子的阈值设置为 0.4，laplacian 算子的阈值设置为 0.1，Canny 算子的阈值设置为 0.1。

实验中一共有 9 张测试图片：bird.png、giraffe.png、noise.png、noise2.png、rubberband_cap.png、1.bmp、2.bmp 和 3.bmp，事实上我还用了另外三张测试图片，但是在边缘链接时，总是因为图片过大，bwtraceboudary 库函数中途中止，因此我最终删去了图片

2.2 实验结果

2.2.1 bird.png



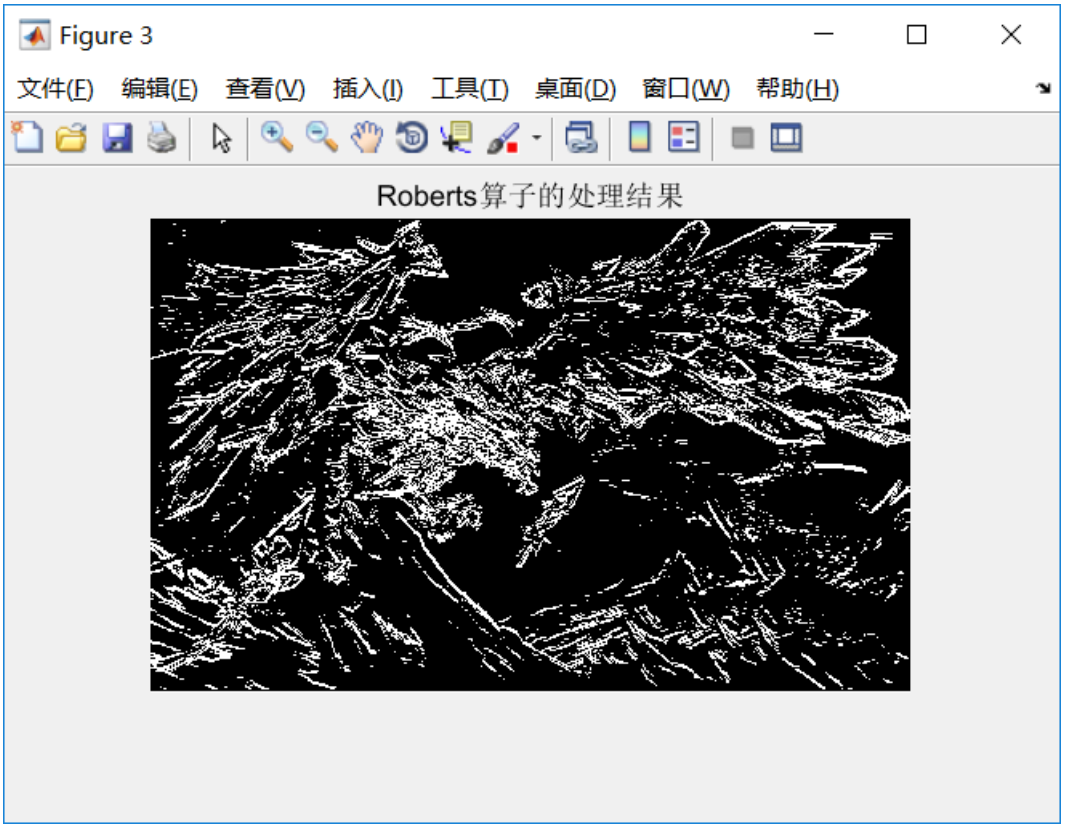


Figure 3
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

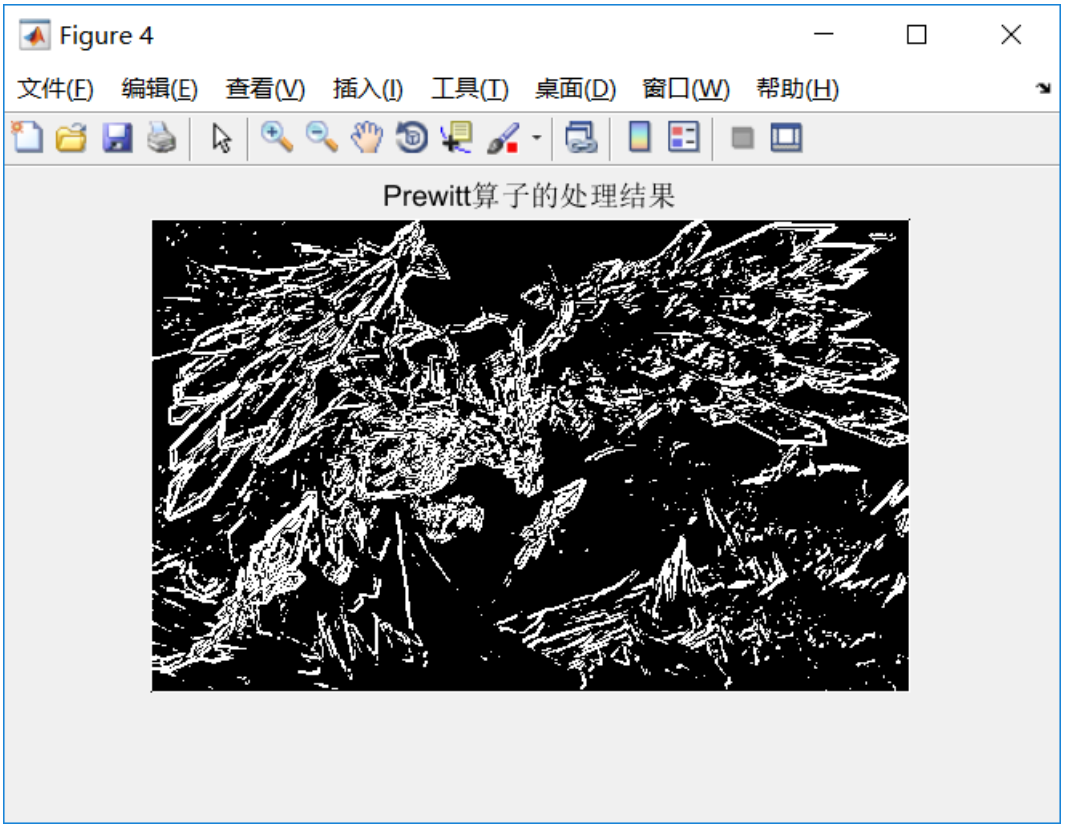


Figure 4
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

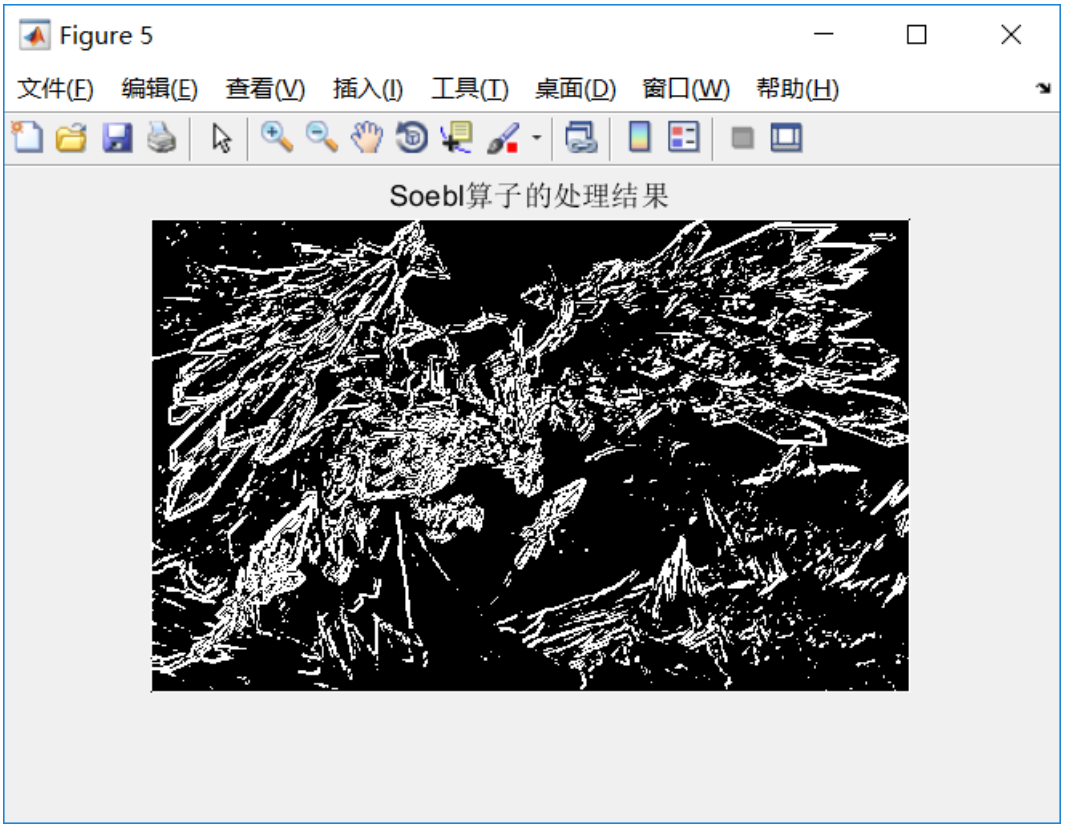


Figure 5
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

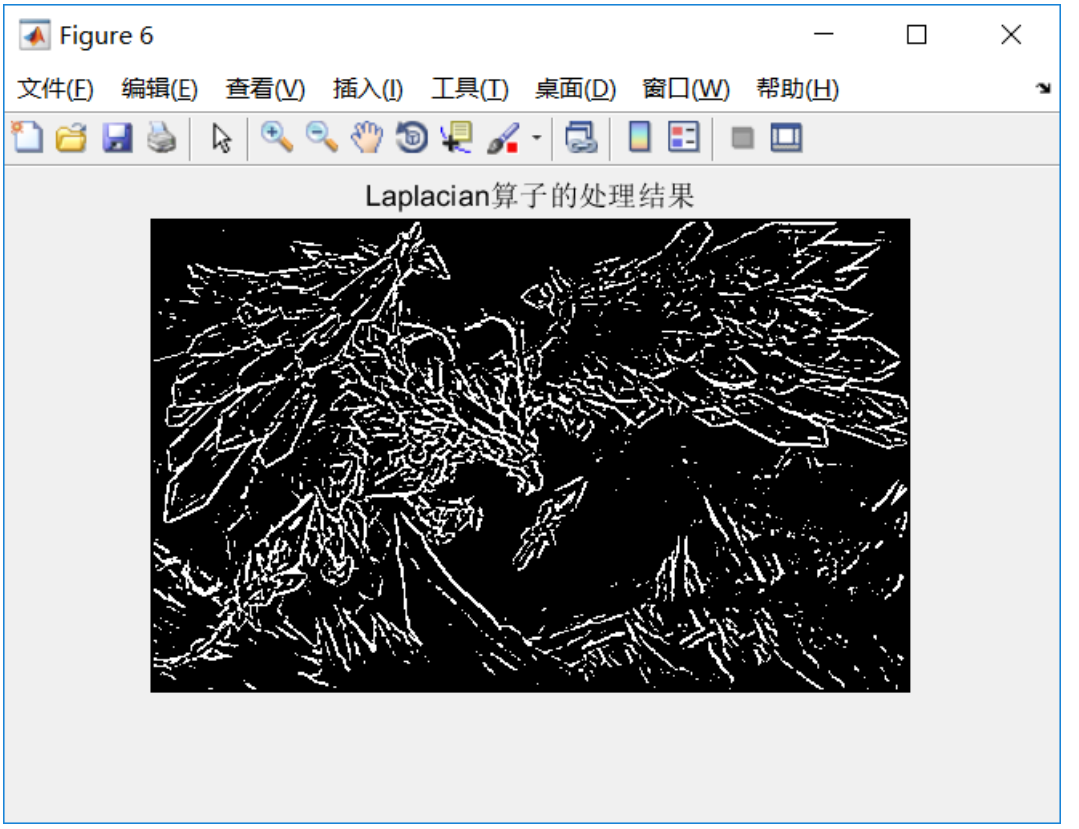


Figure 6
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

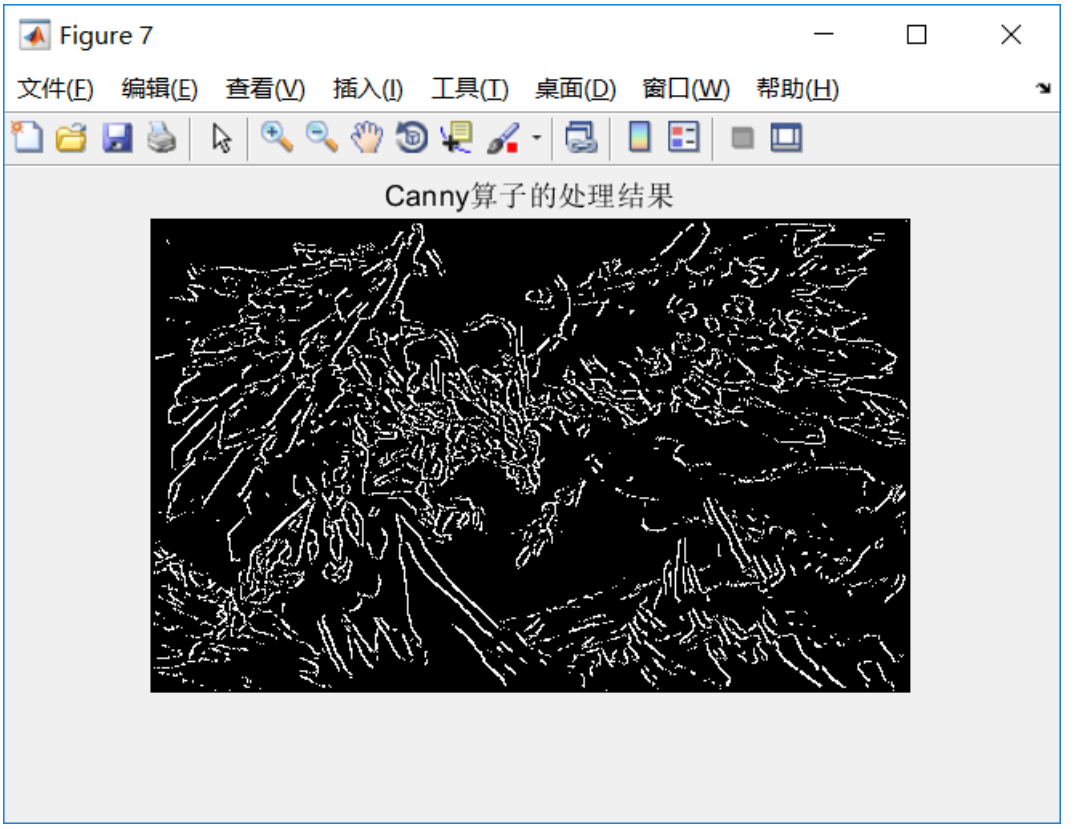


Figure 7
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

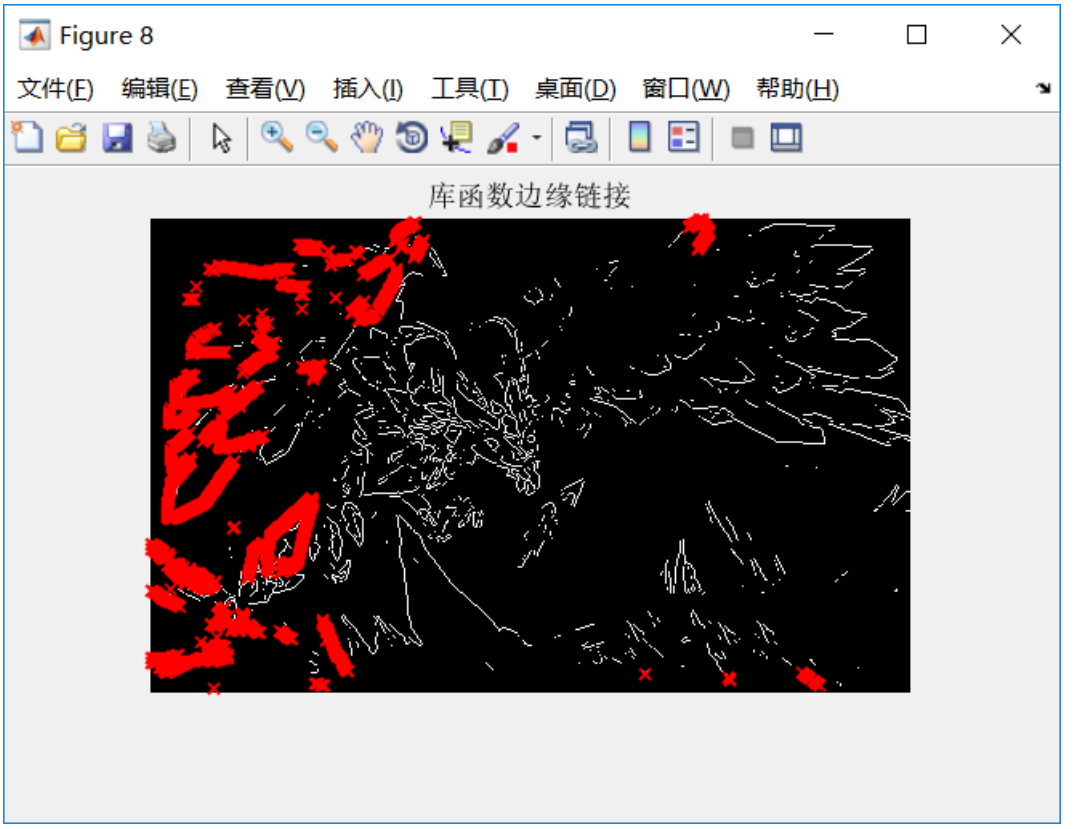


Figure 8
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

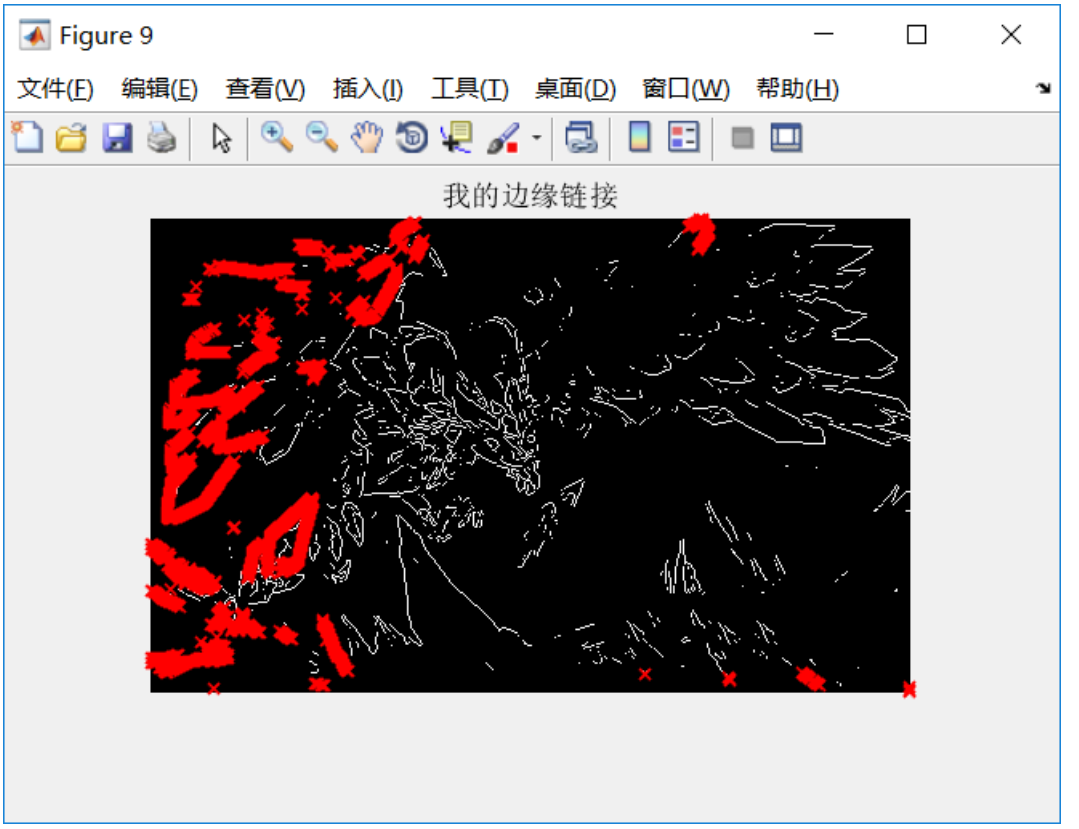
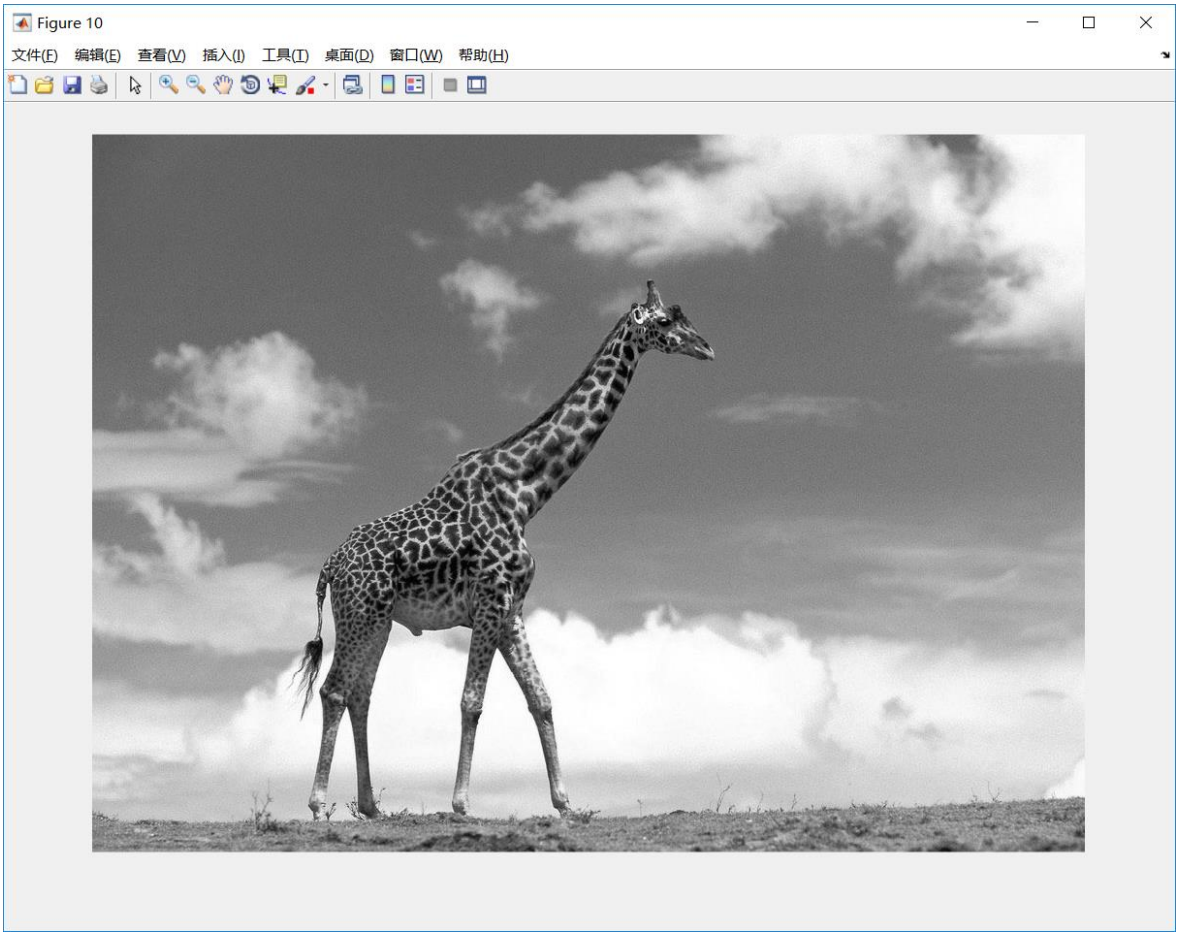


Figure 9
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.2 giraffe.png



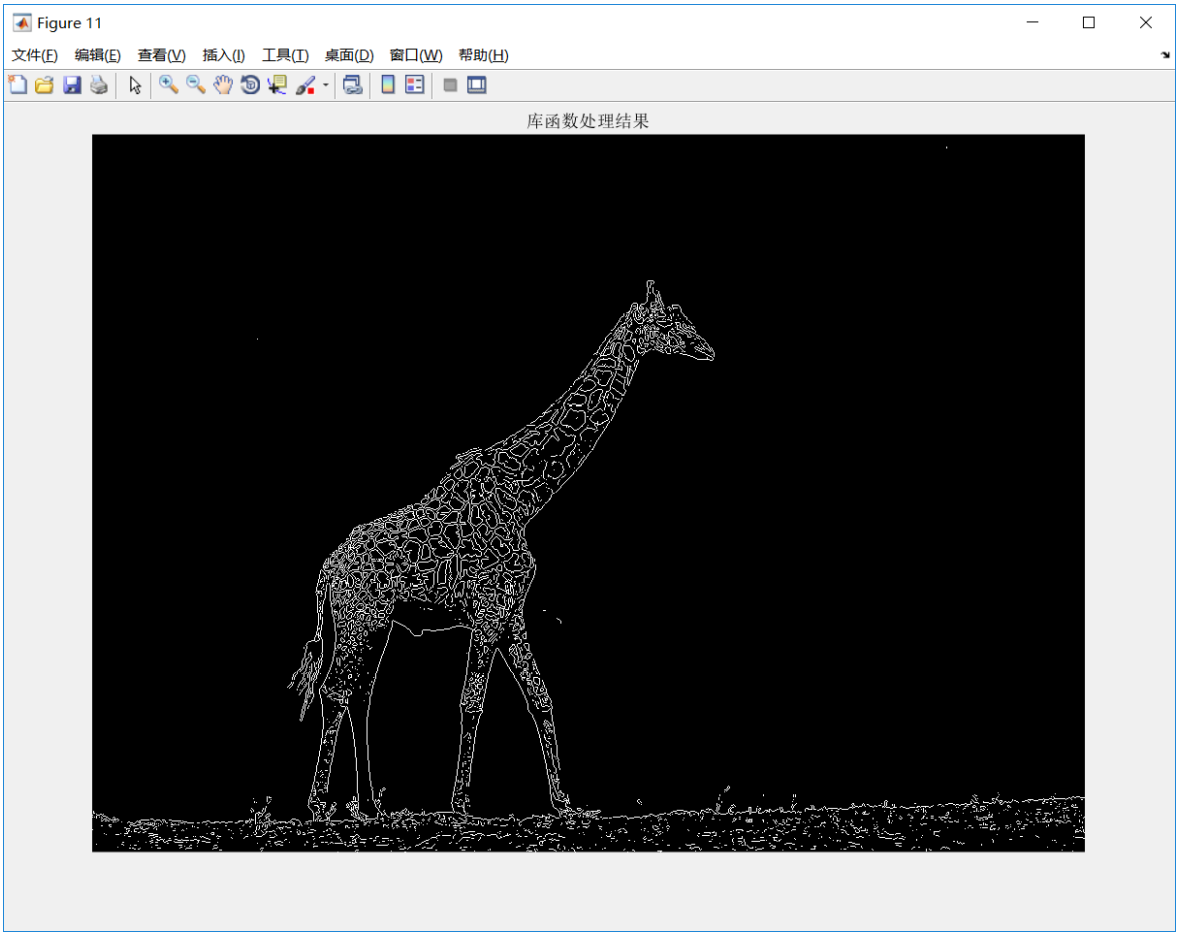


Figure 11
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

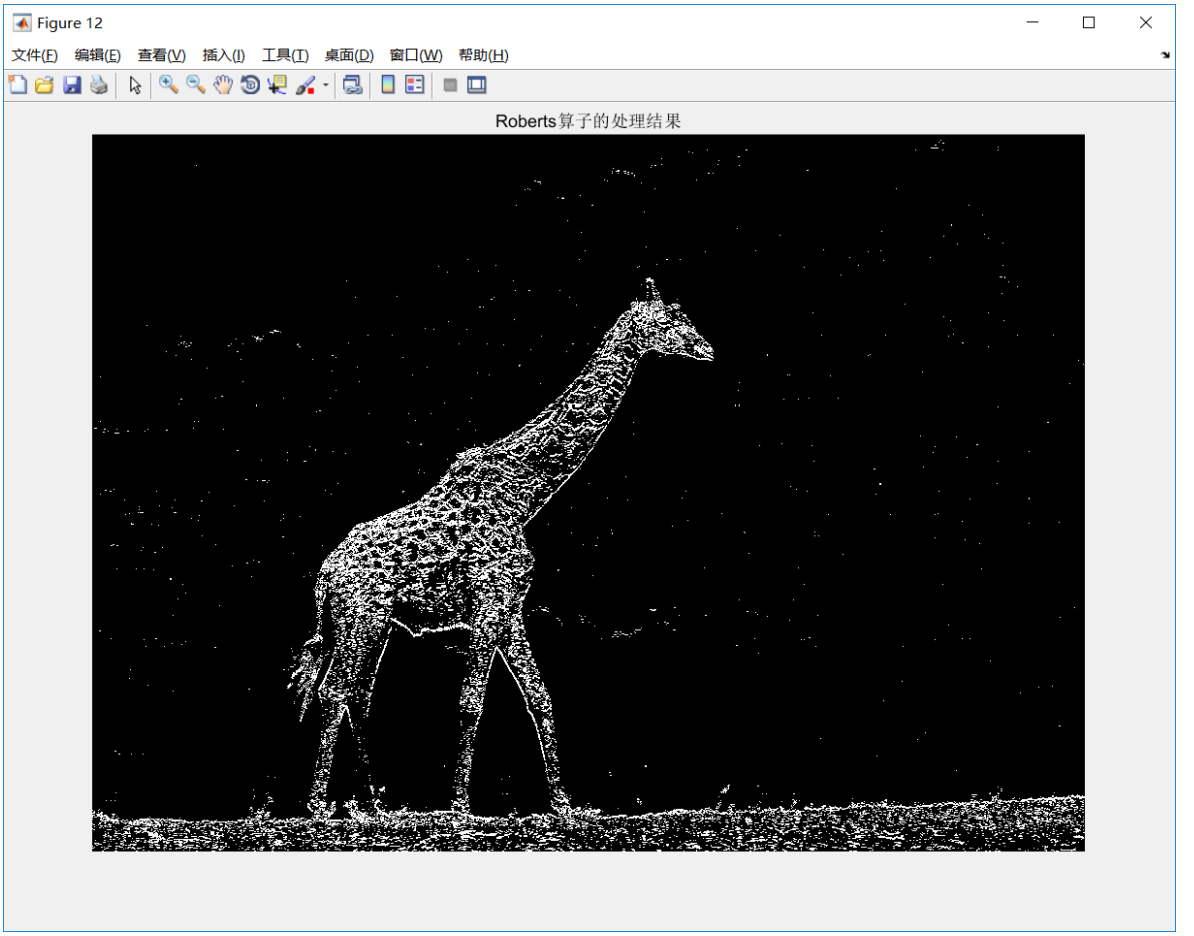


Figure 12
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

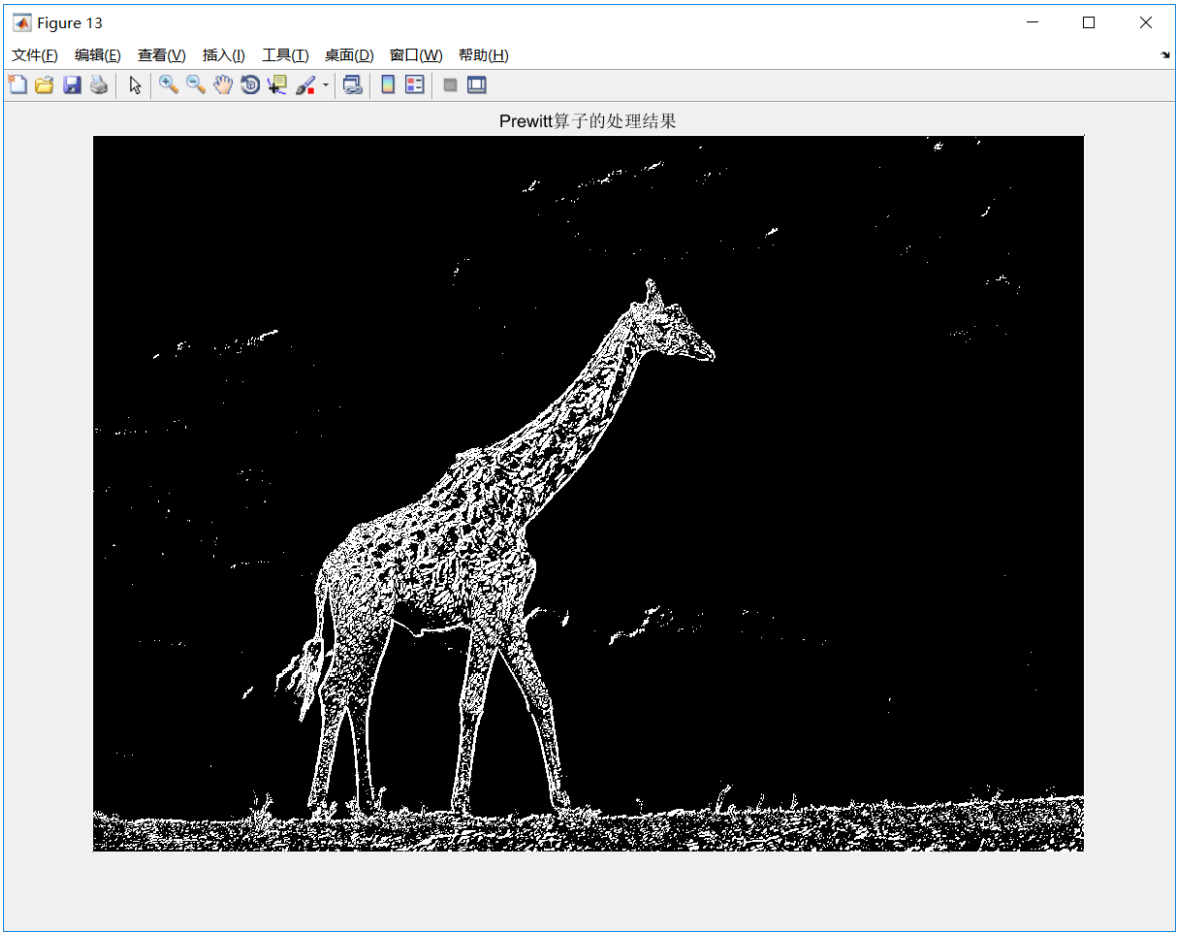


Figure 13
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

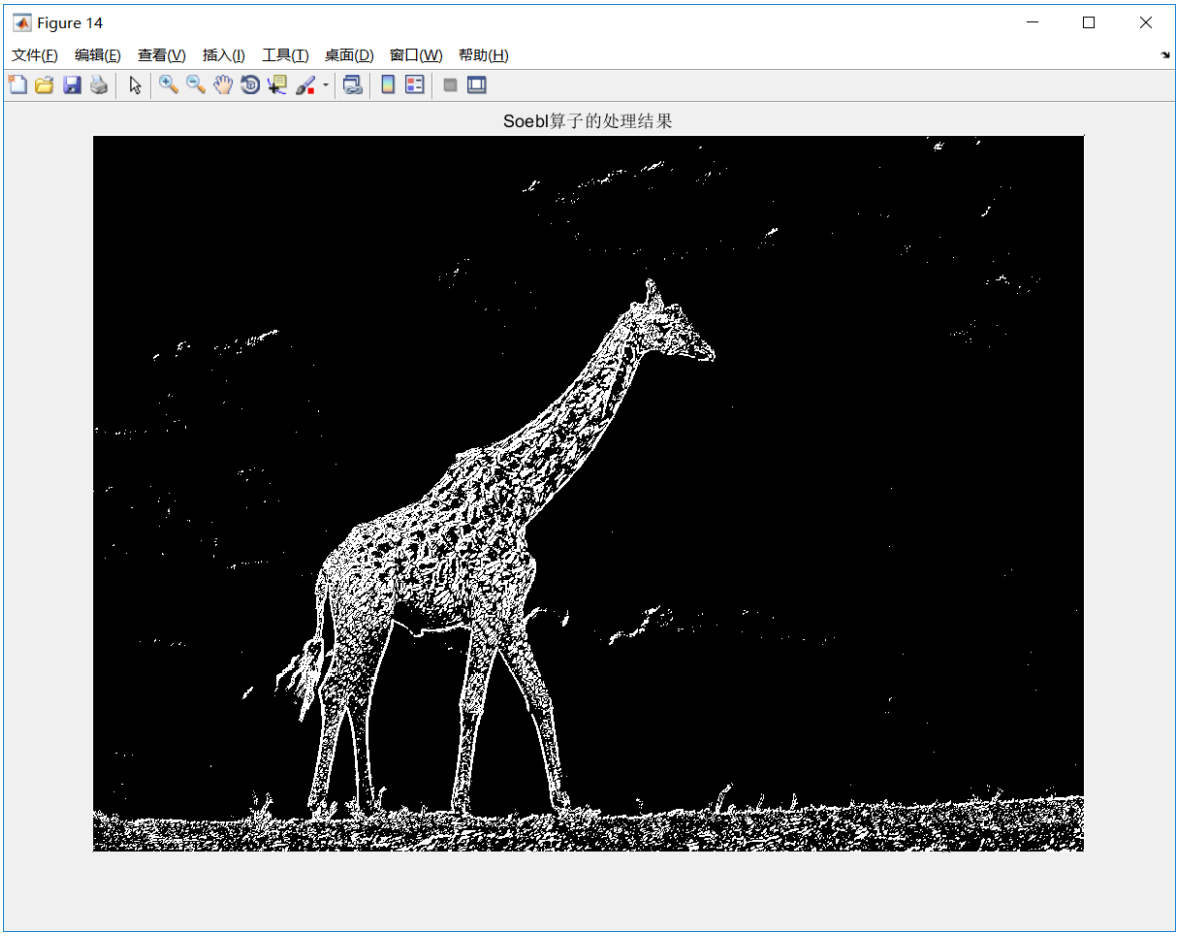


Figure 14
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

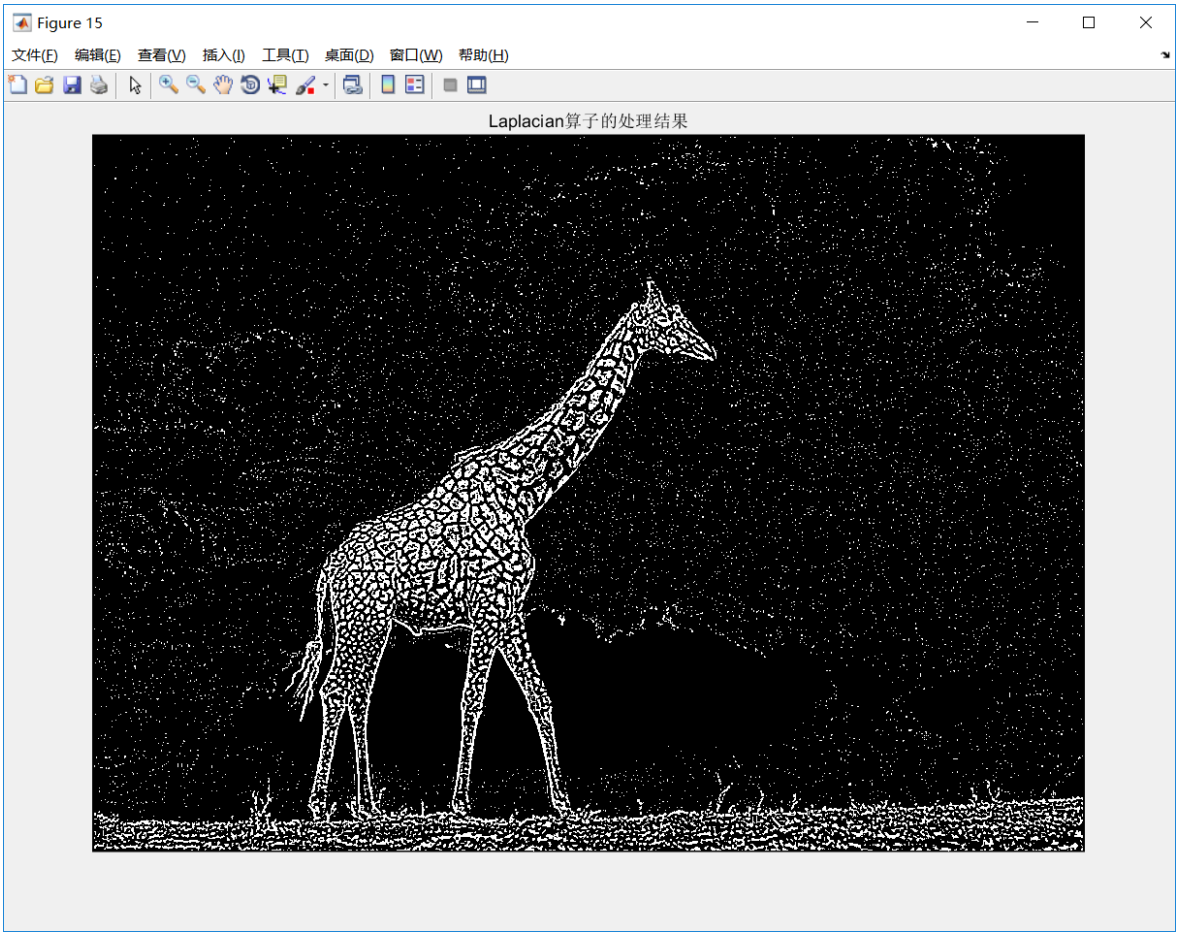


Figure 15
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

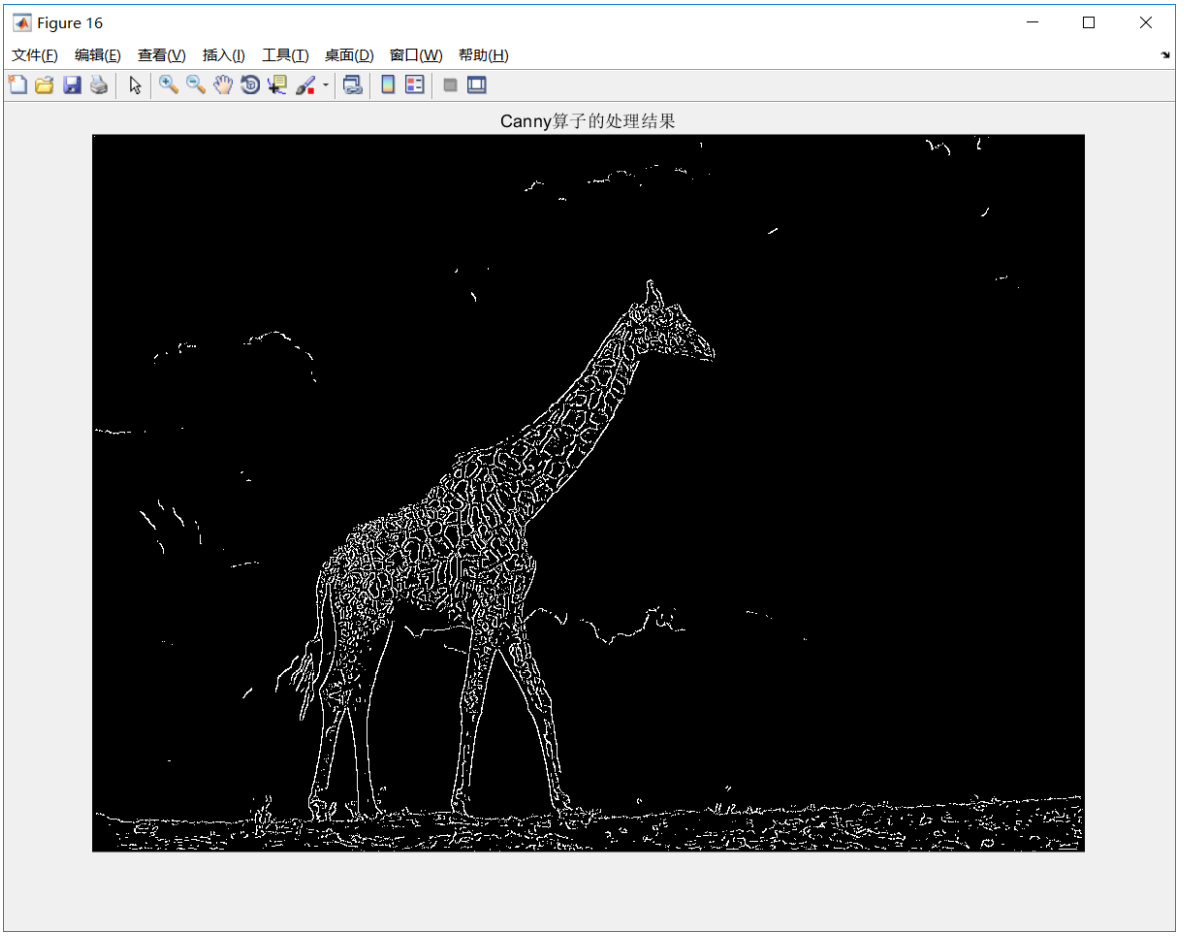


Figure 16
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

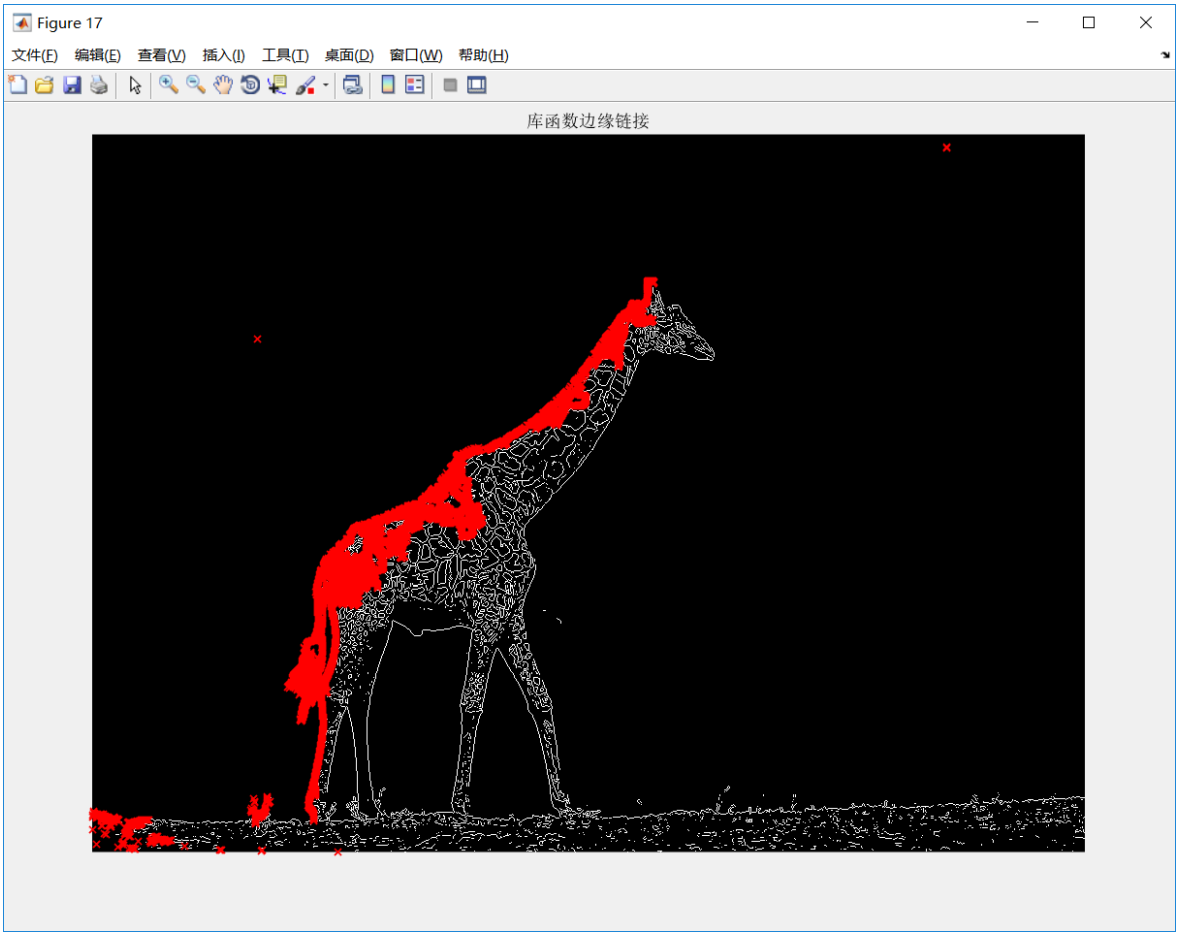


Figure 17
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

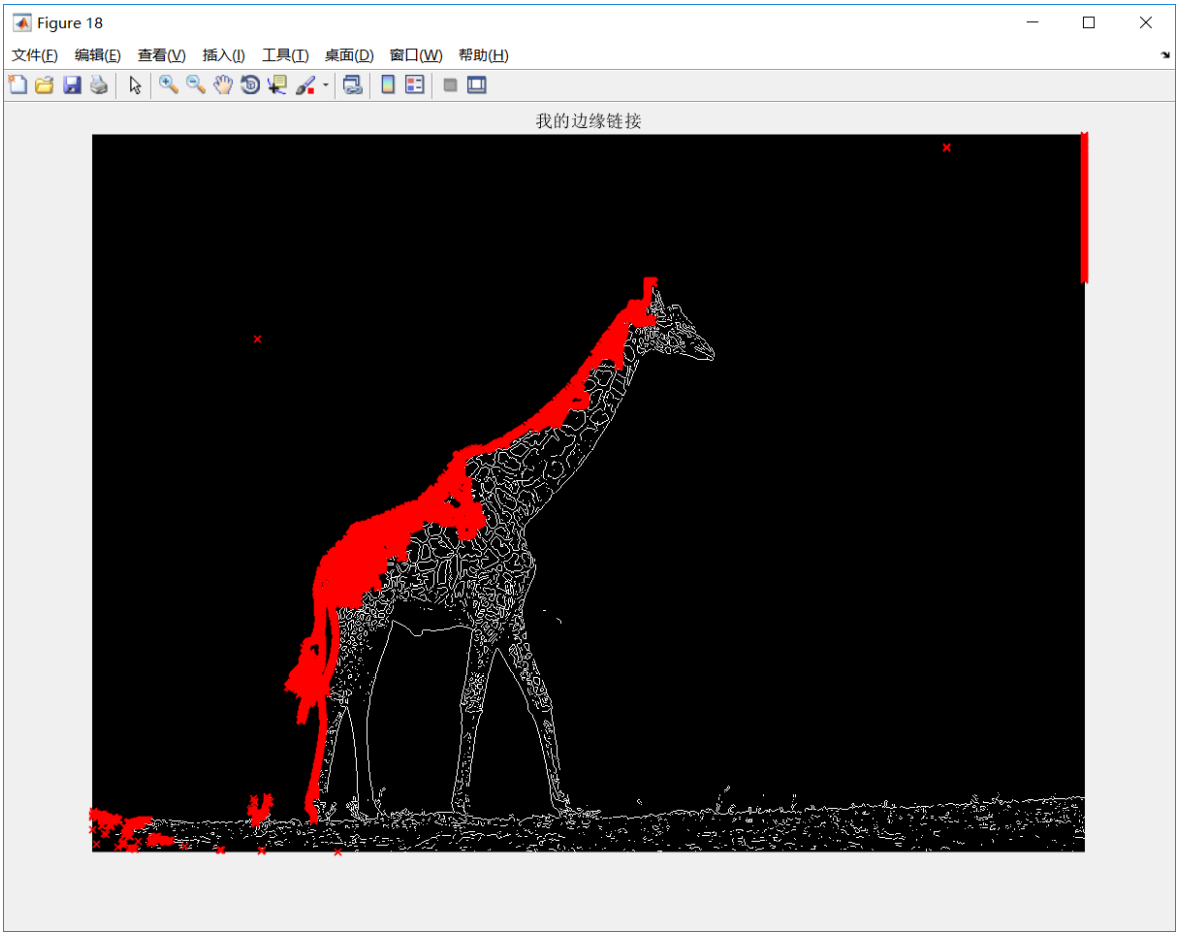
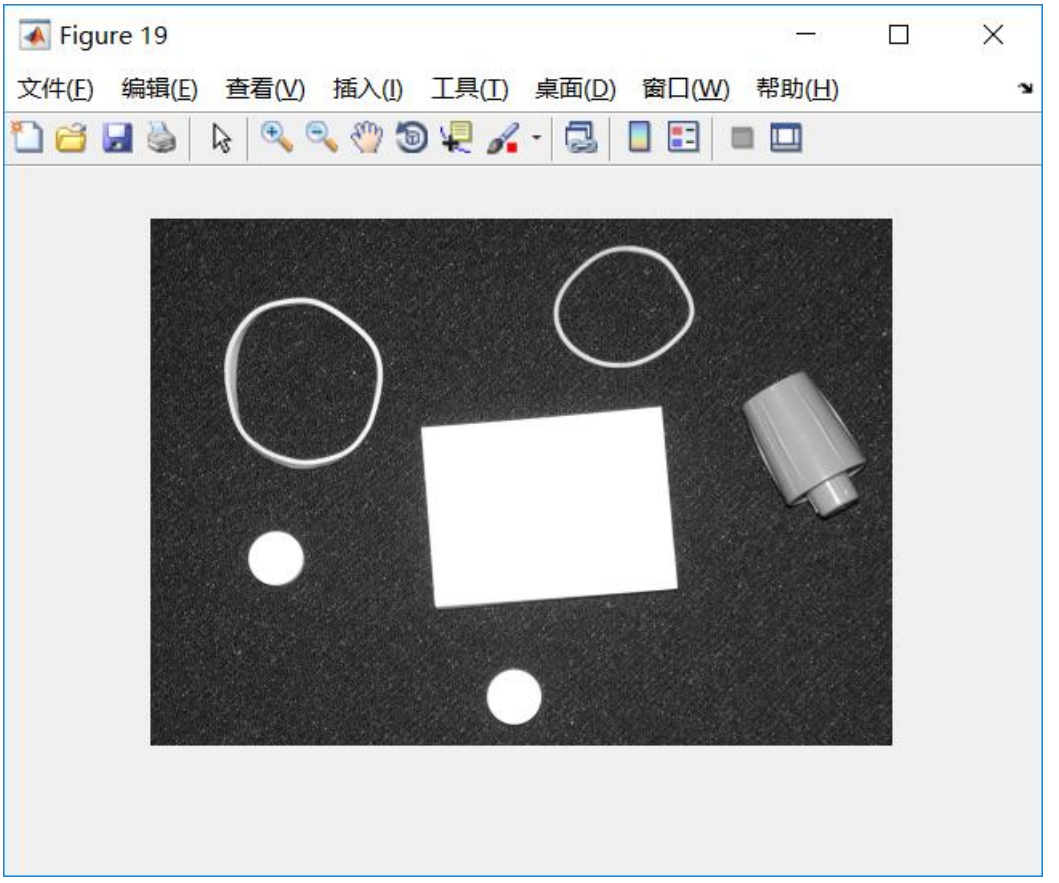


Figure 18
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.3 rubberband_cup.png



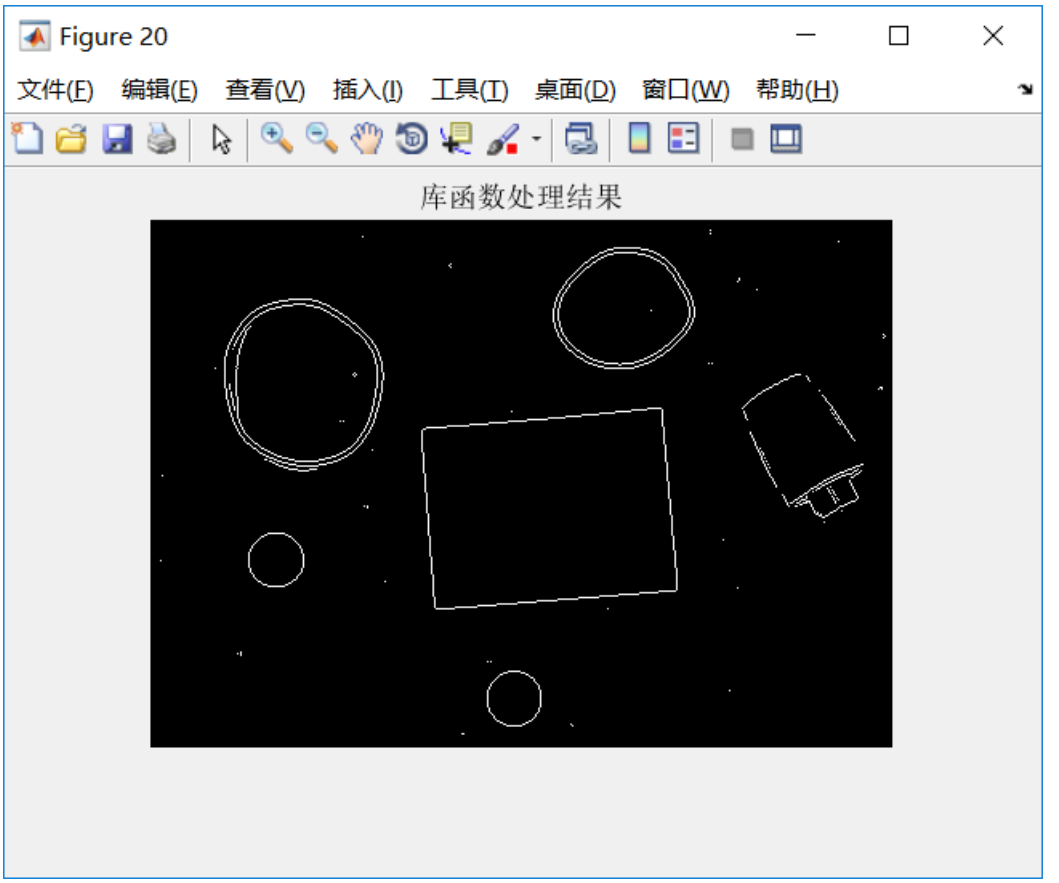


Figure 20
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

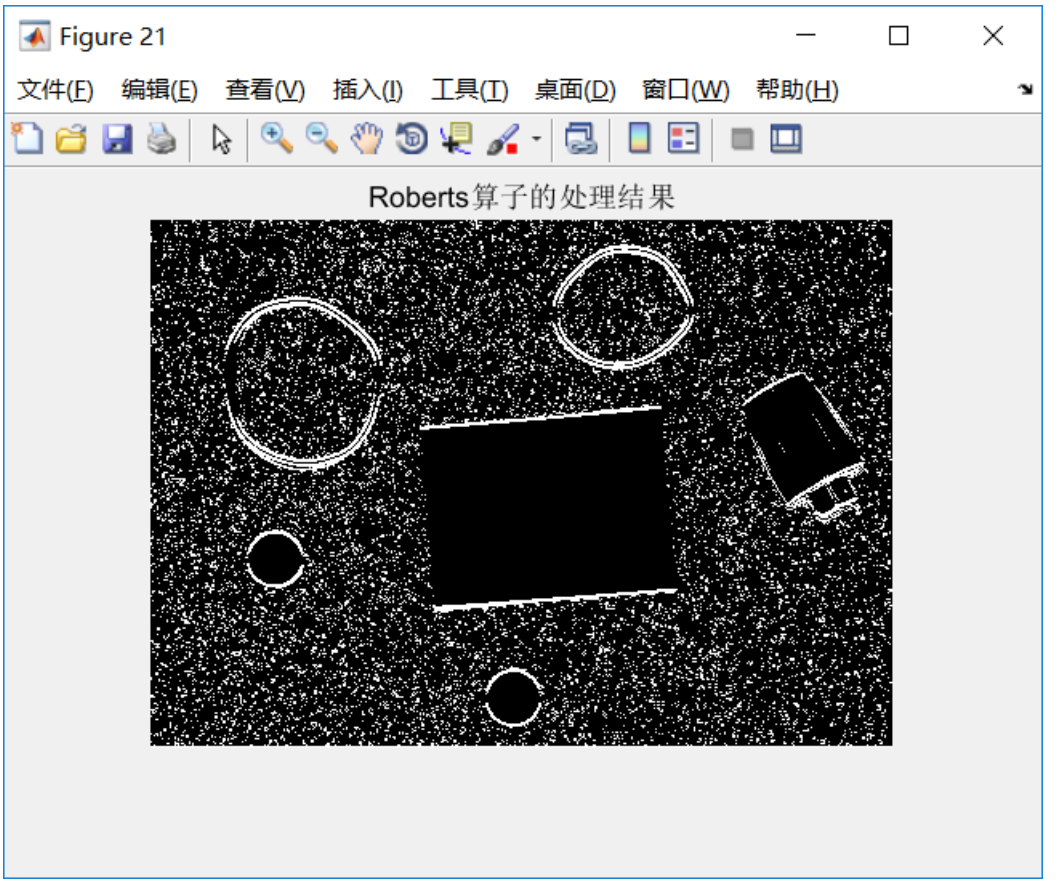


Figure 21
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

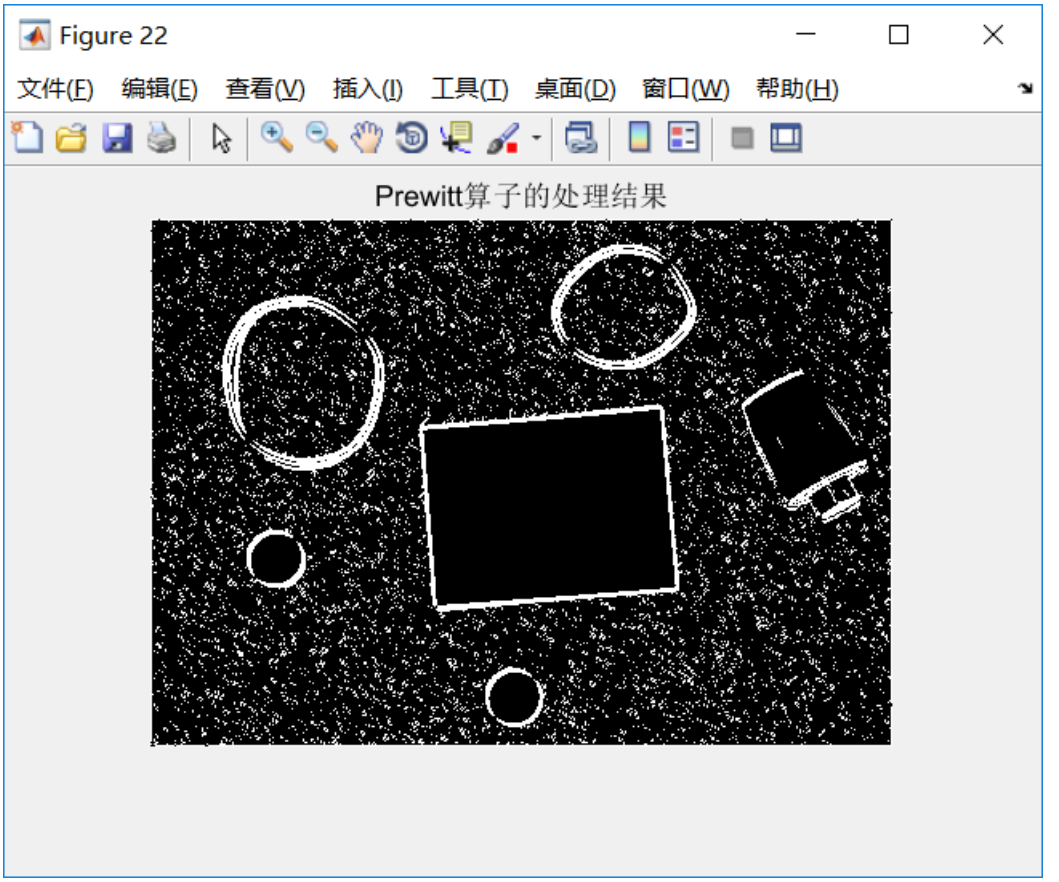


Figure 22
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

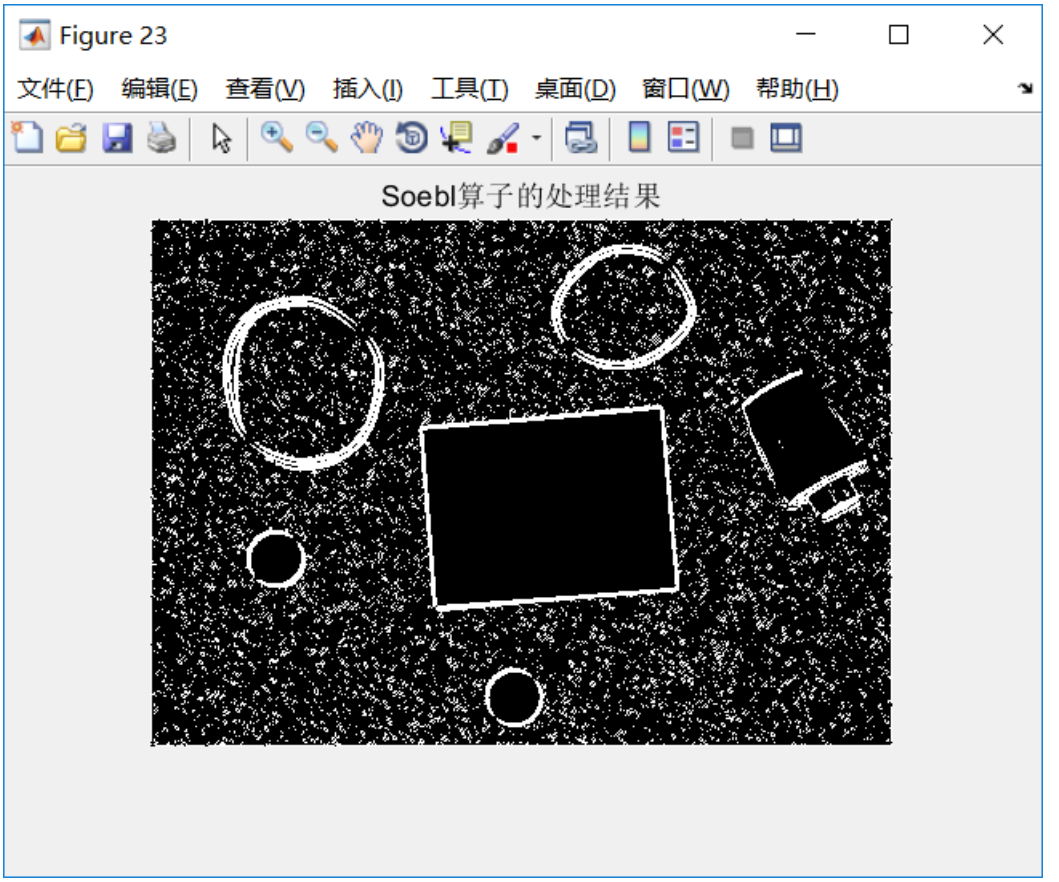


Figure 23
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

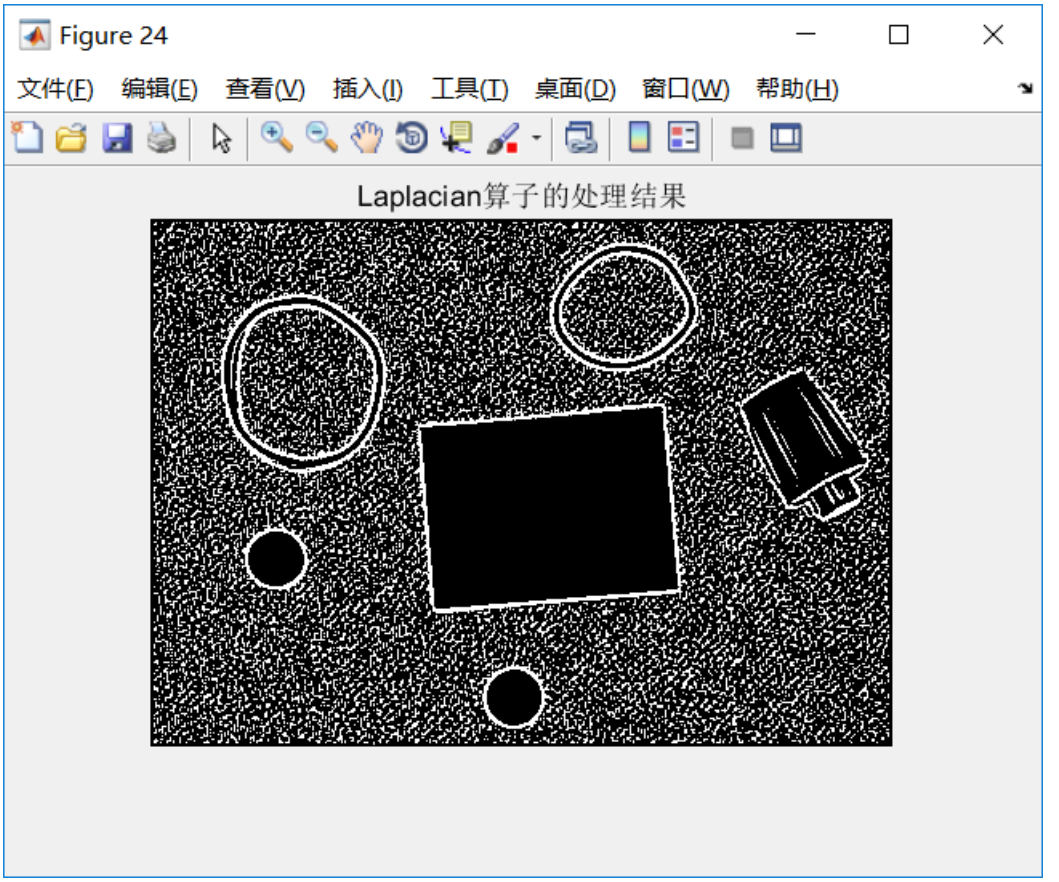


Figure 24
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

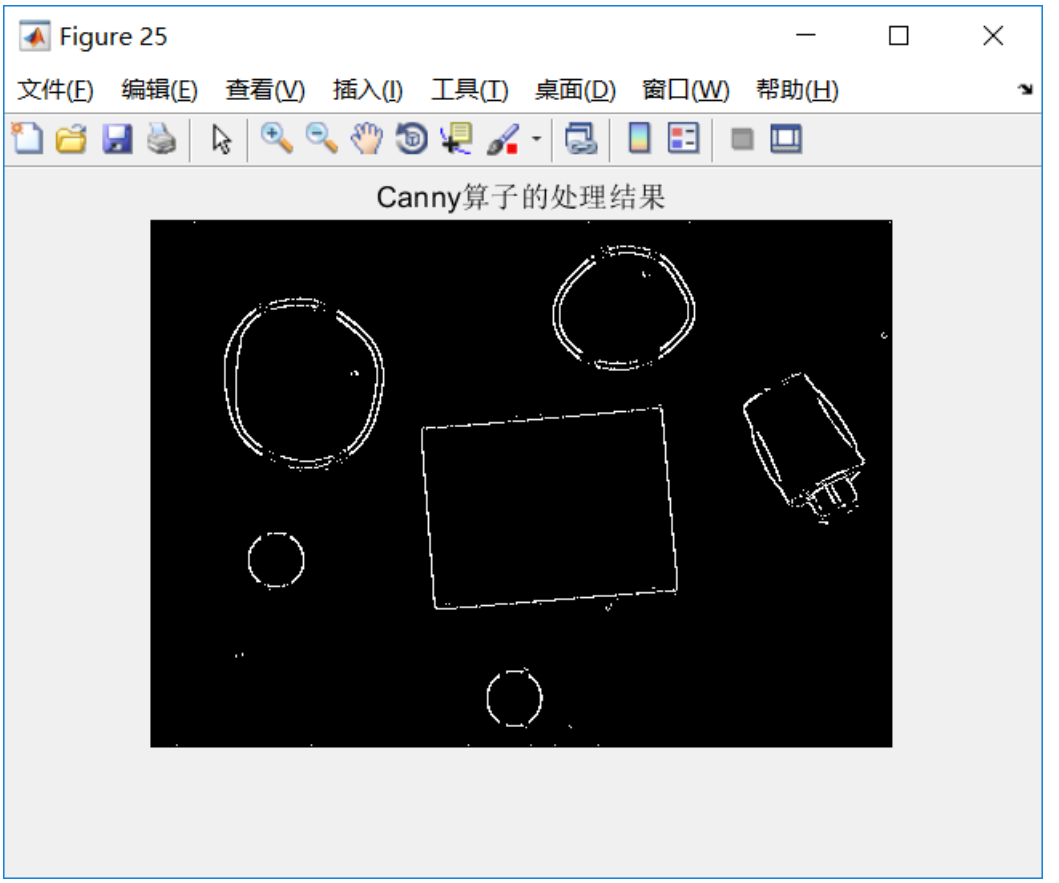


Figure 25
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

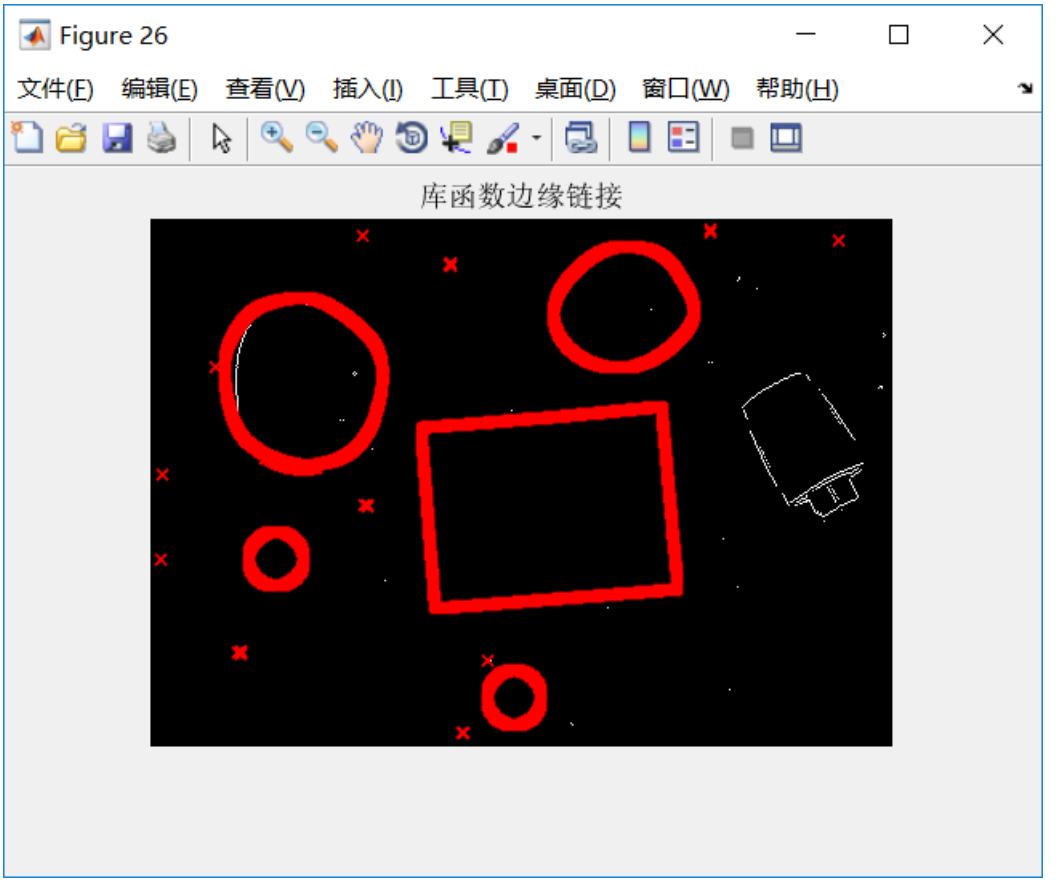


Figure 26
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

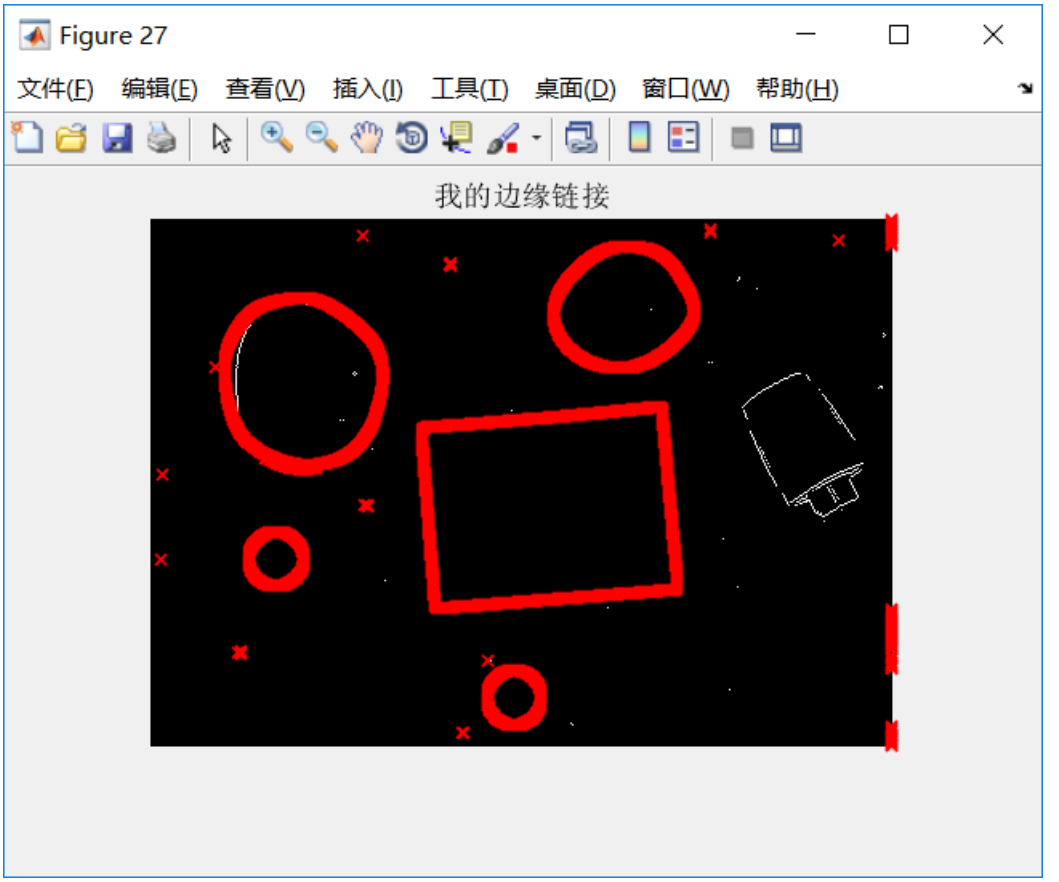
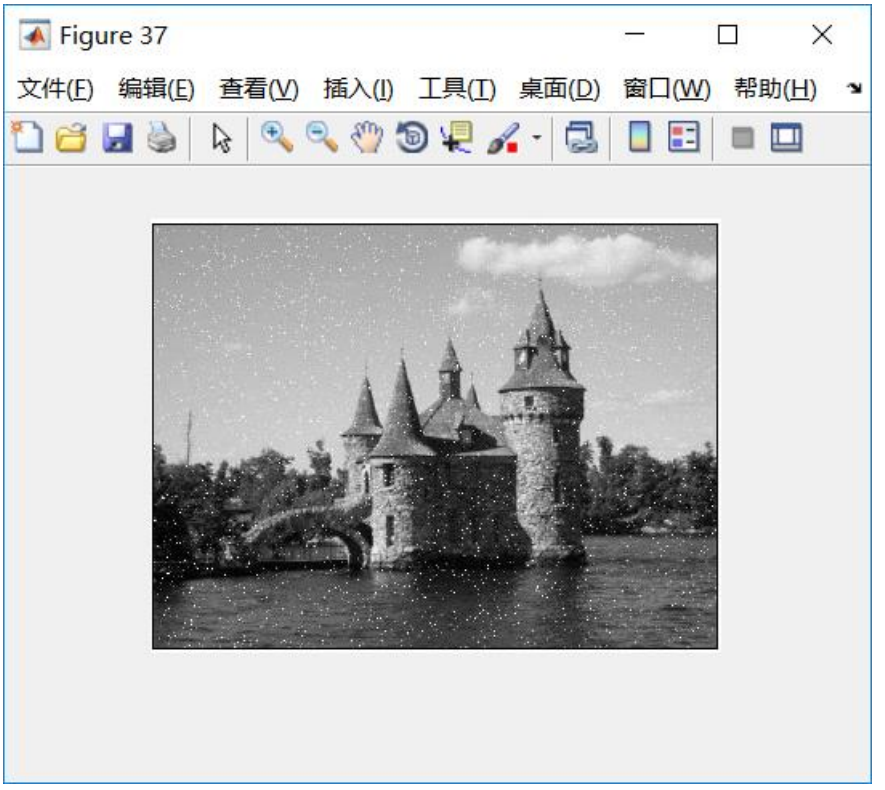
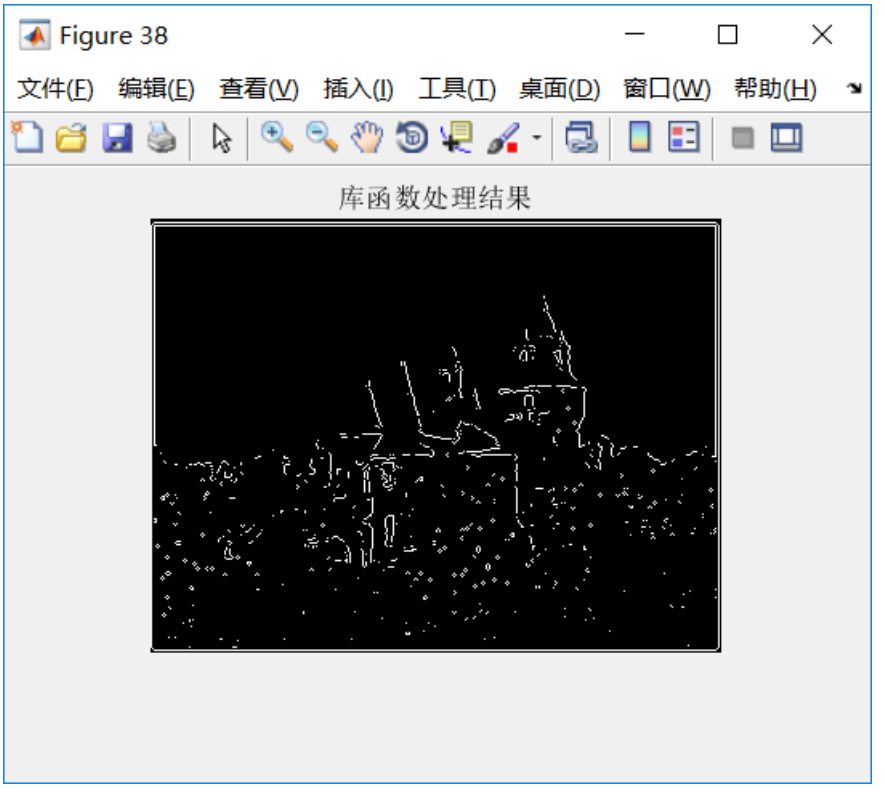


Figure 27
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.4 noise.png





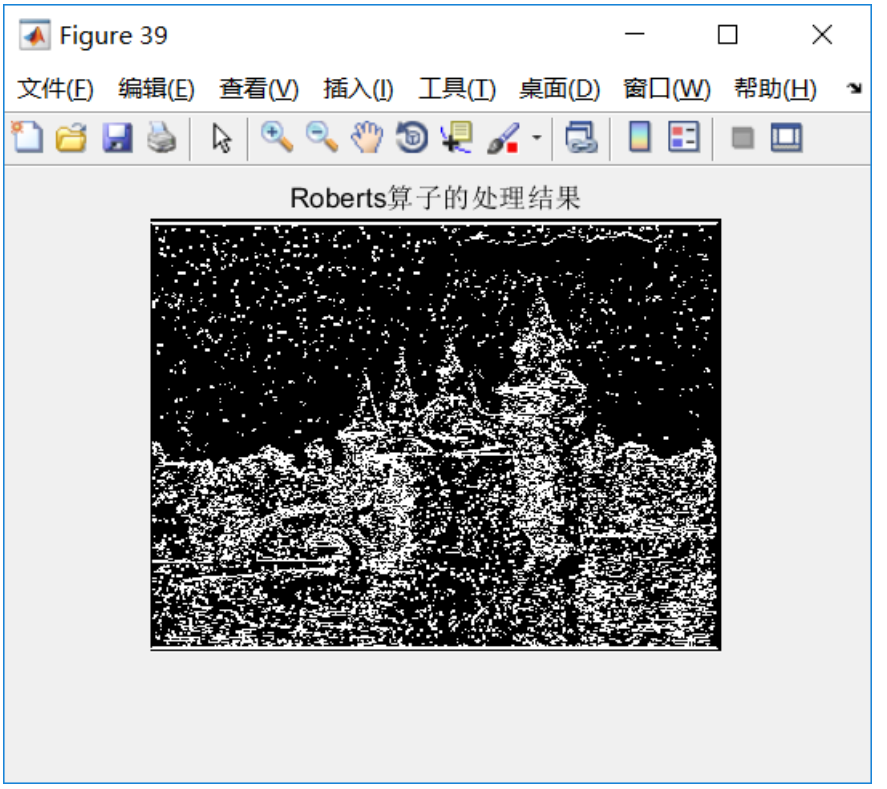


Figure 39
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

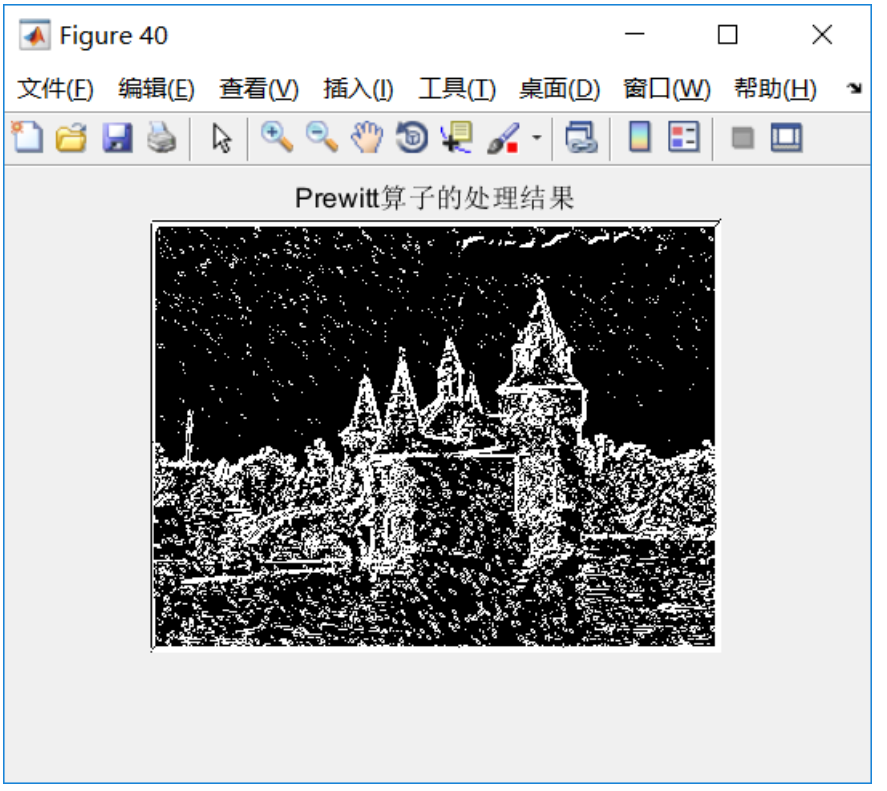


Figure 40
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

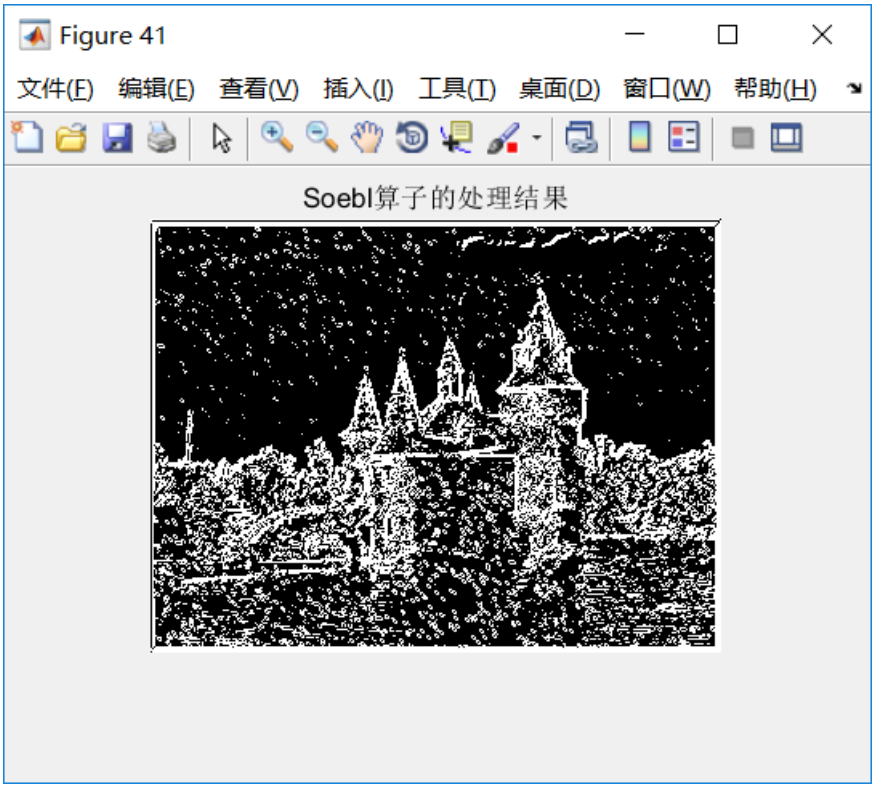


Figure 41
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

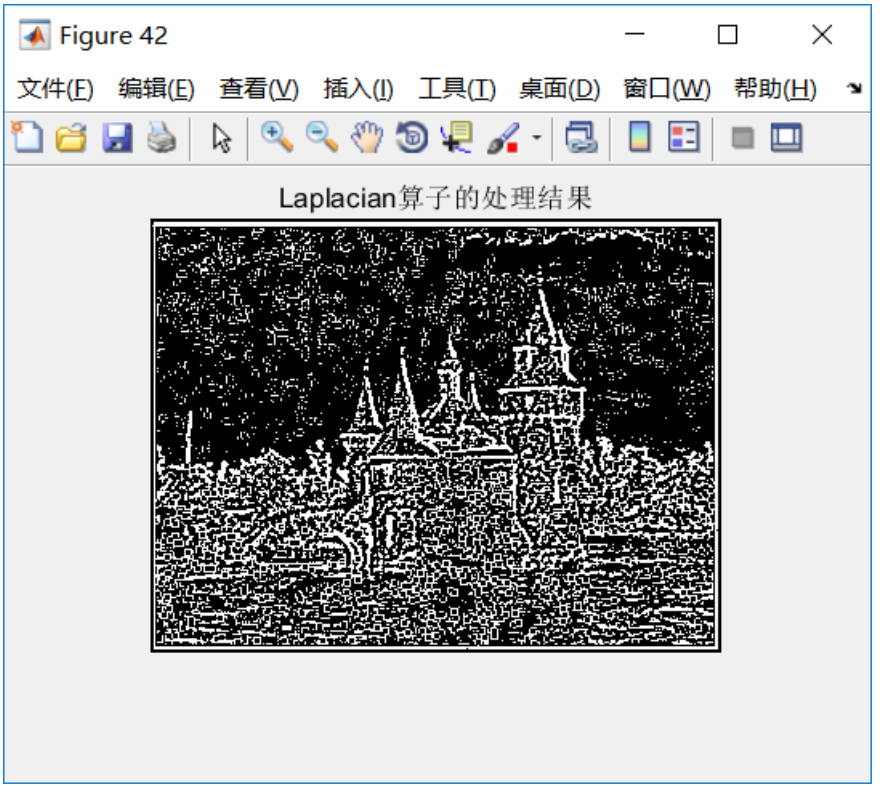


Figure 42
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

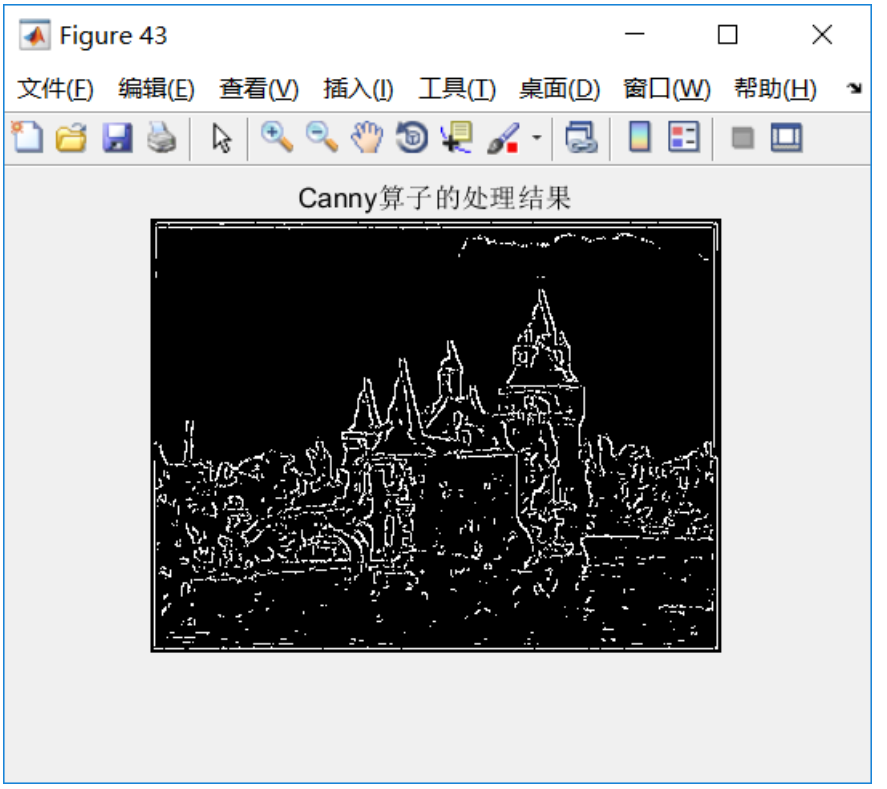


Figure 43
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

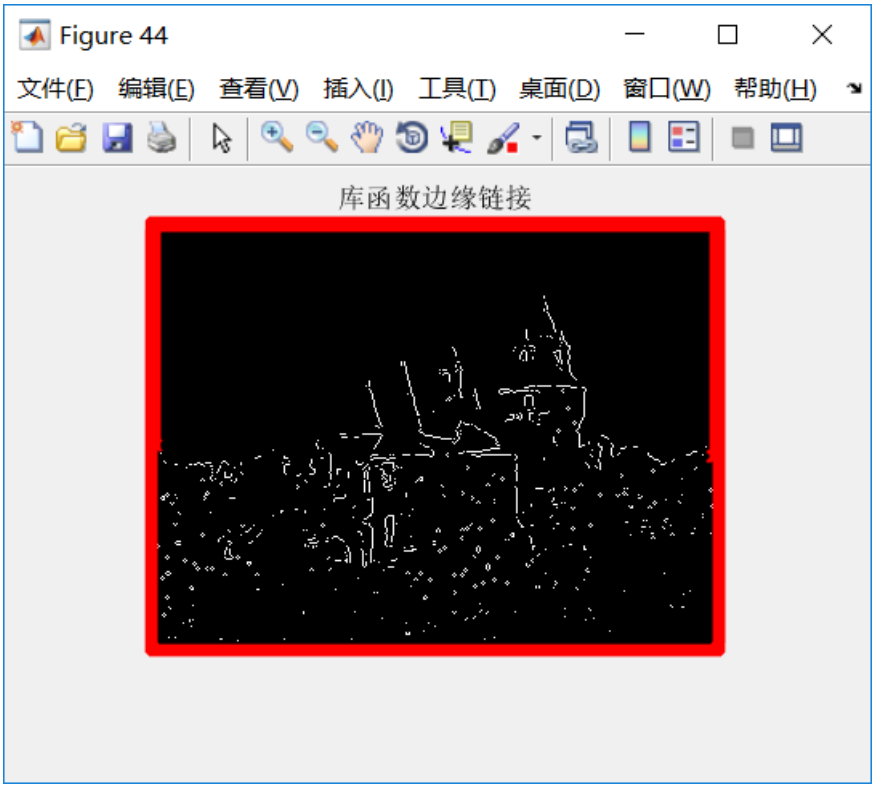


Figure 44
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

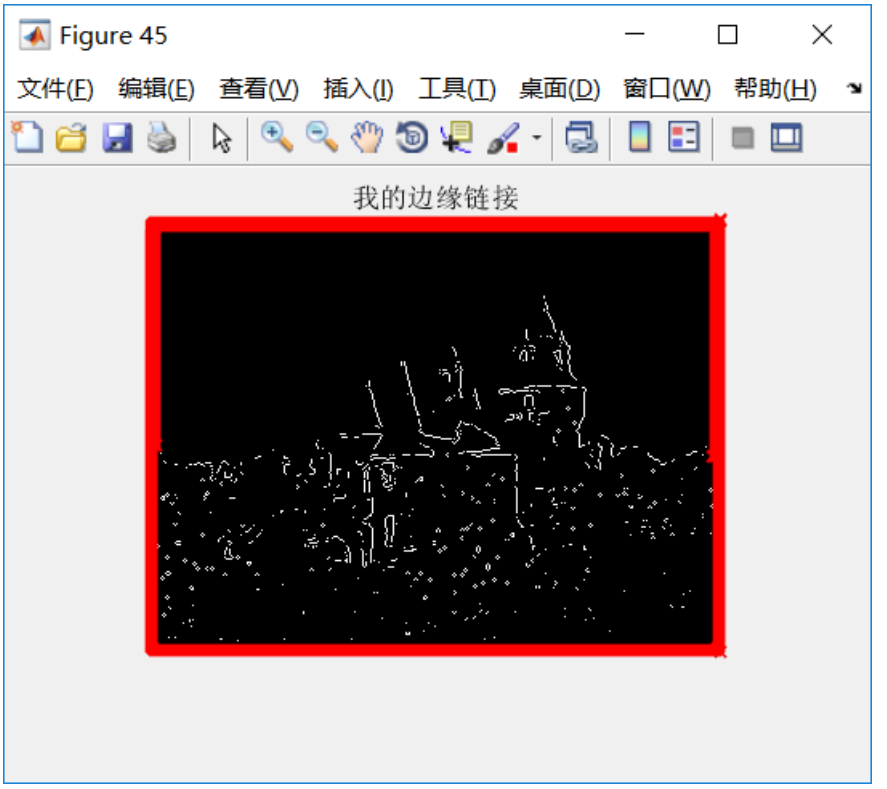
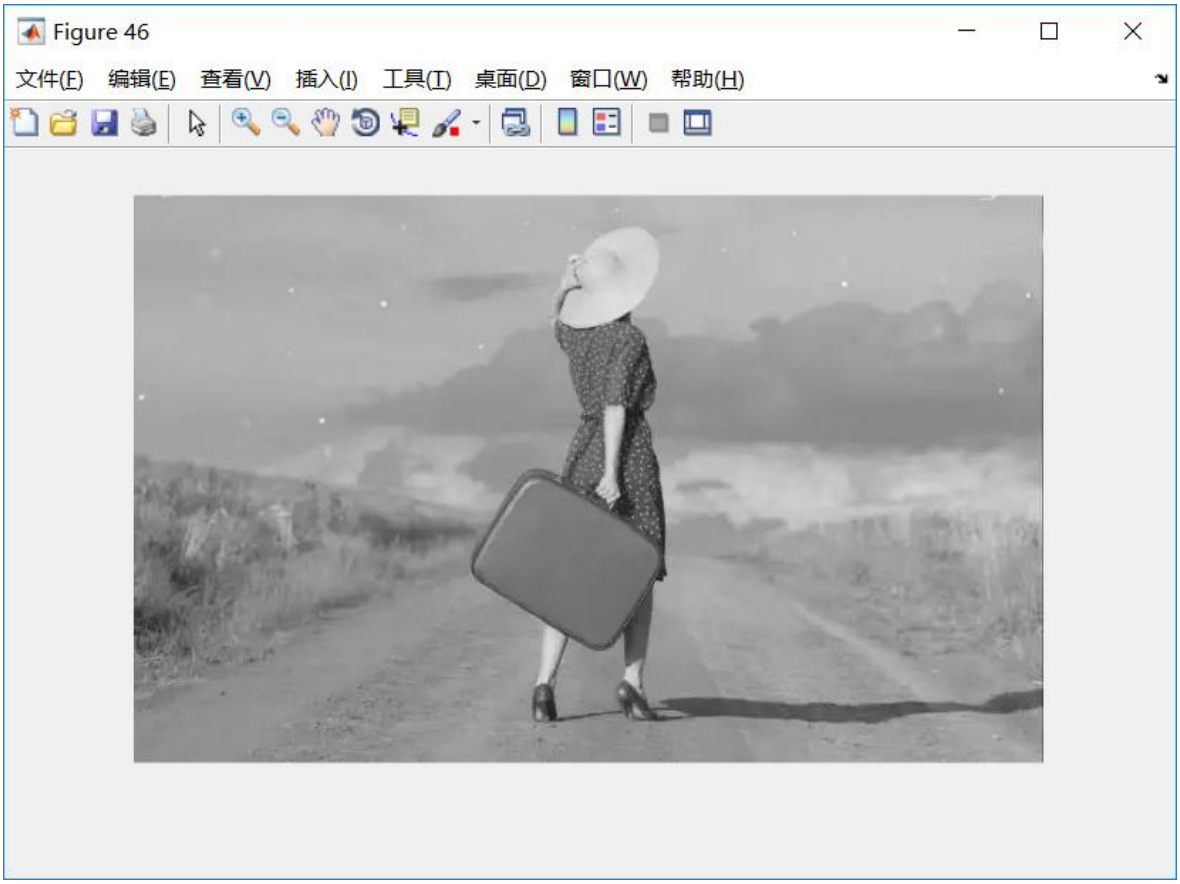


Figure 45
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.5 noise2.png



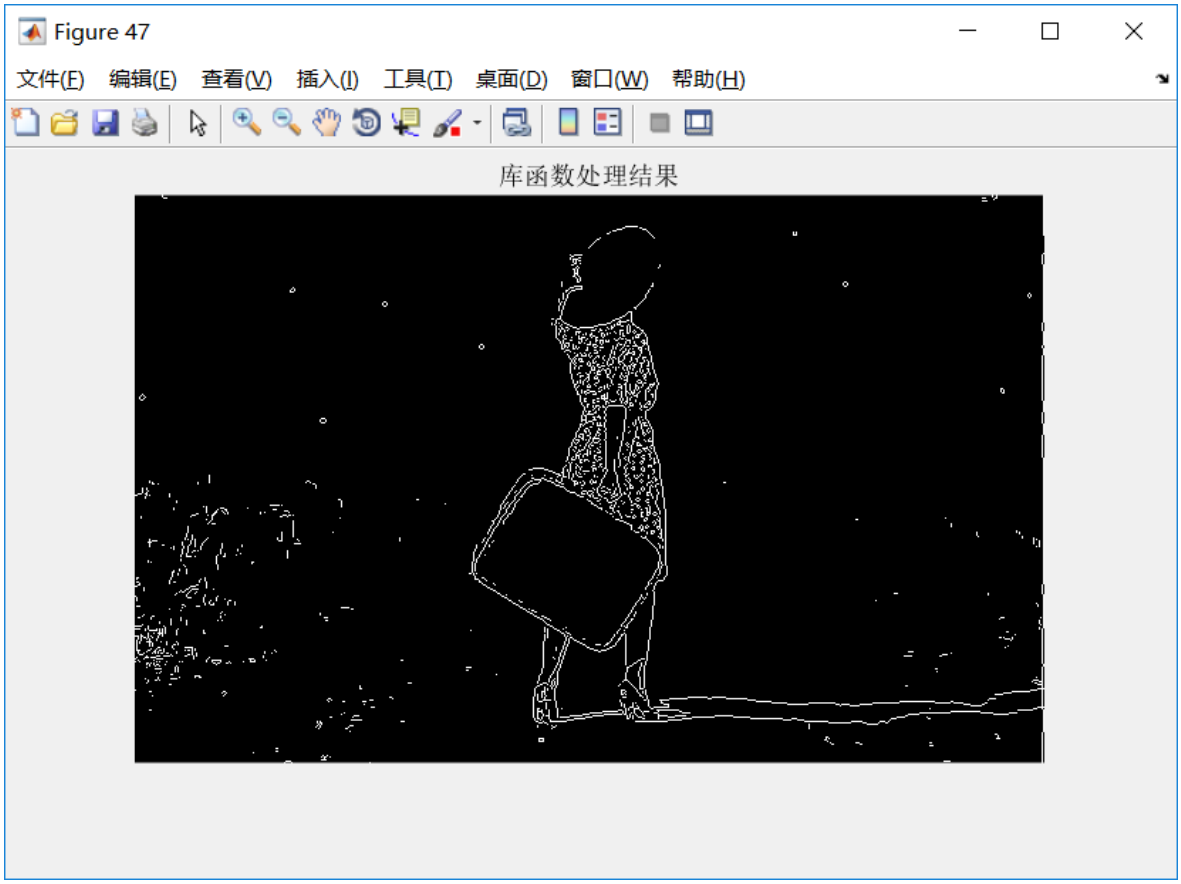


Figure 47
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

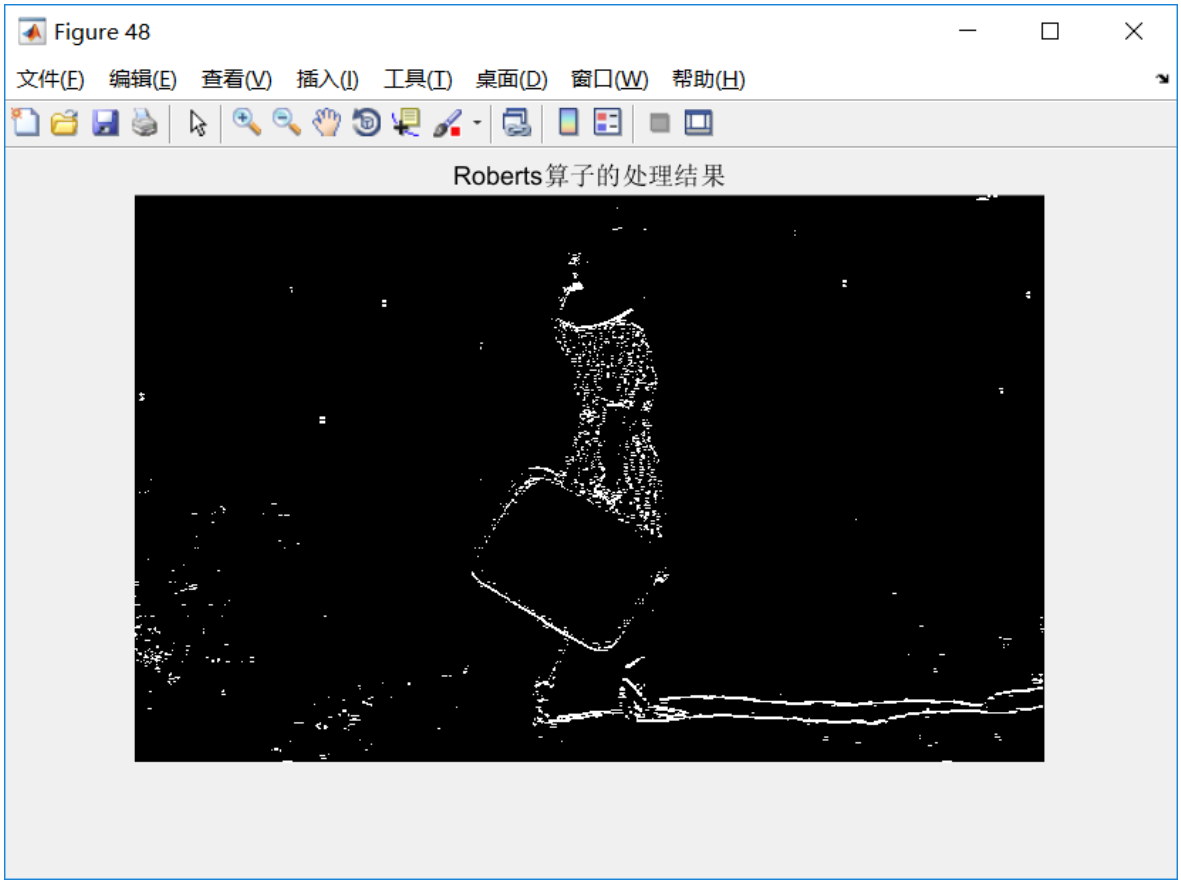


Figure 48
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

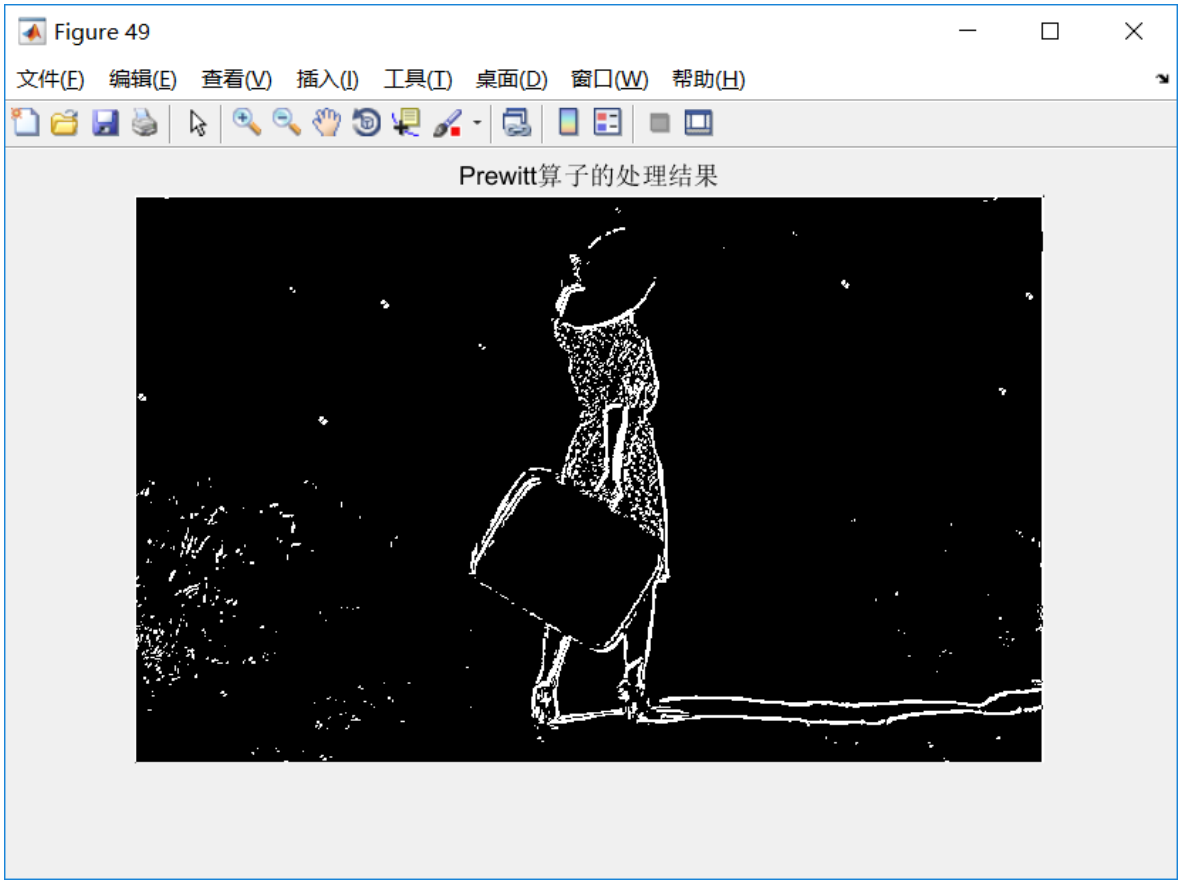


Figure 49
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

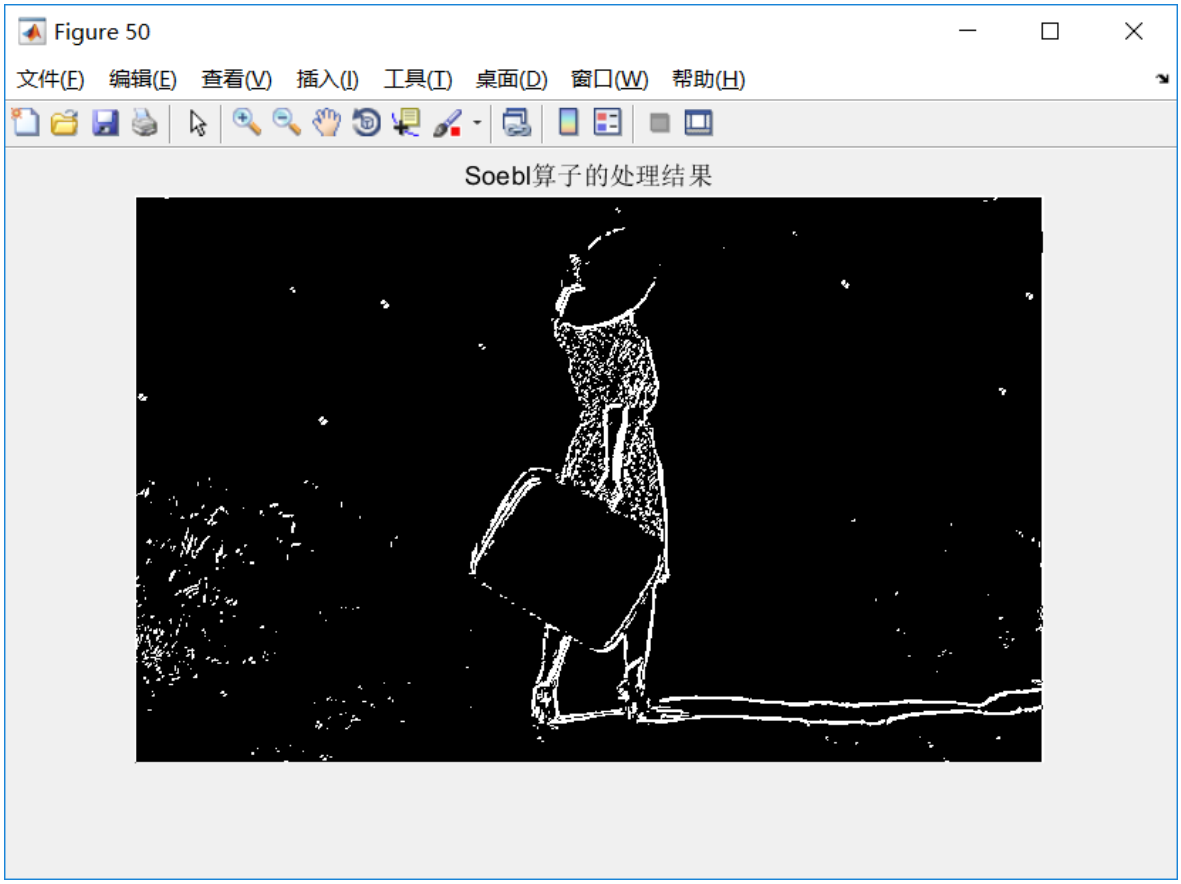


Figure 50
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

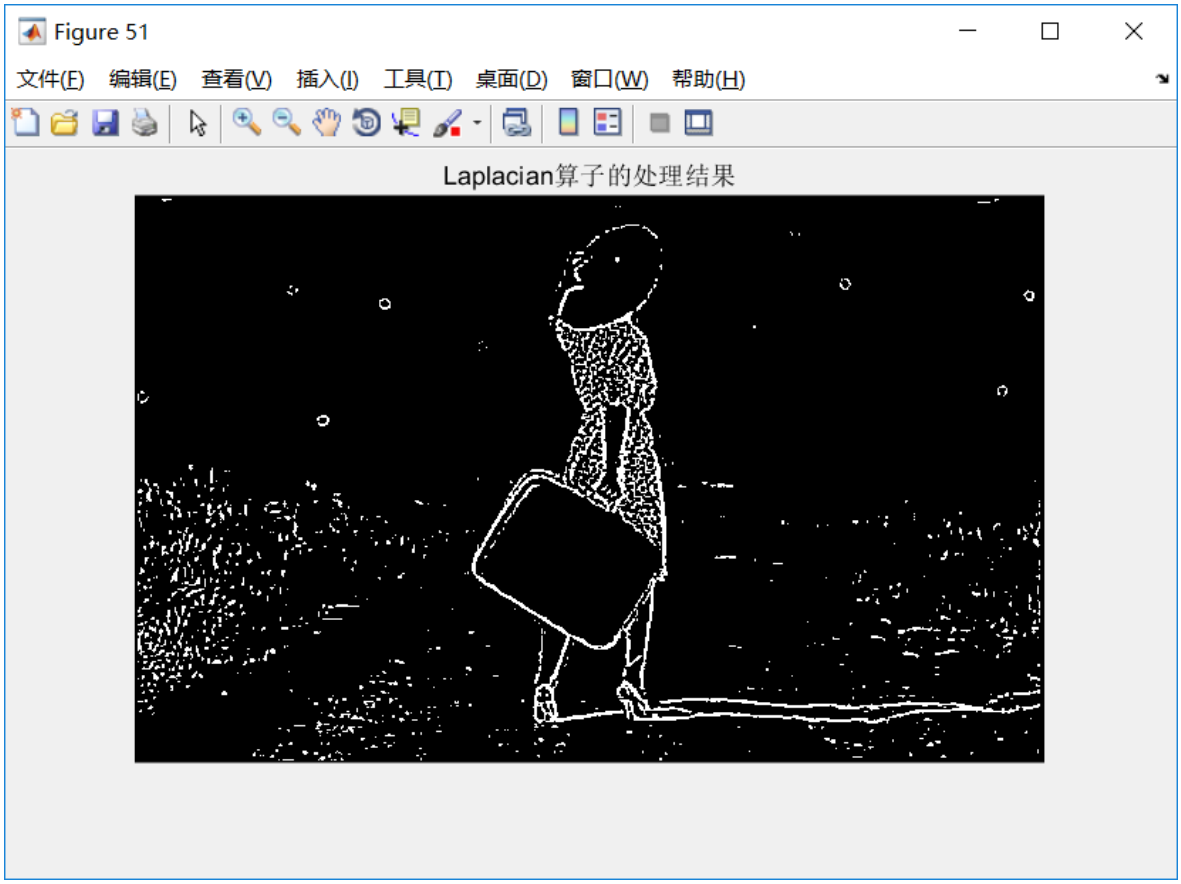


Figure 51
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

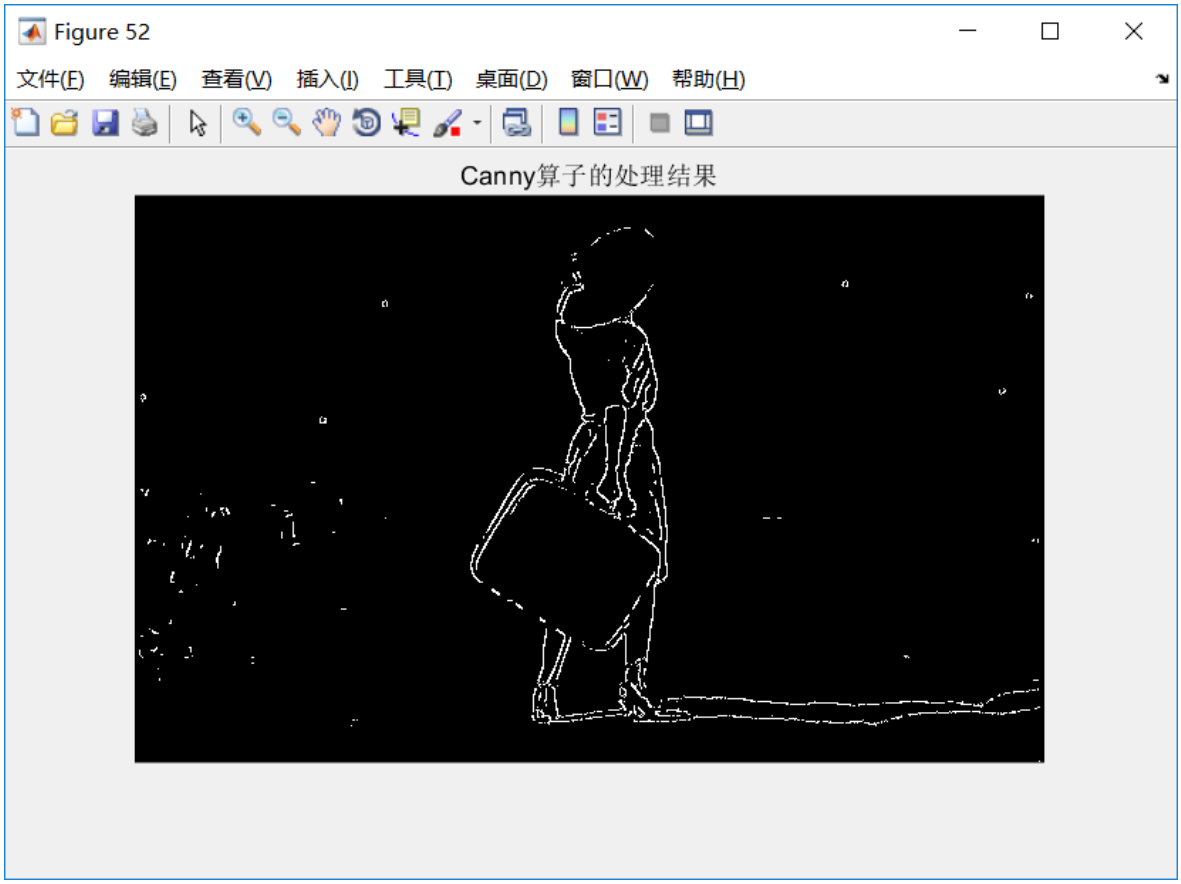


Figure 52
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

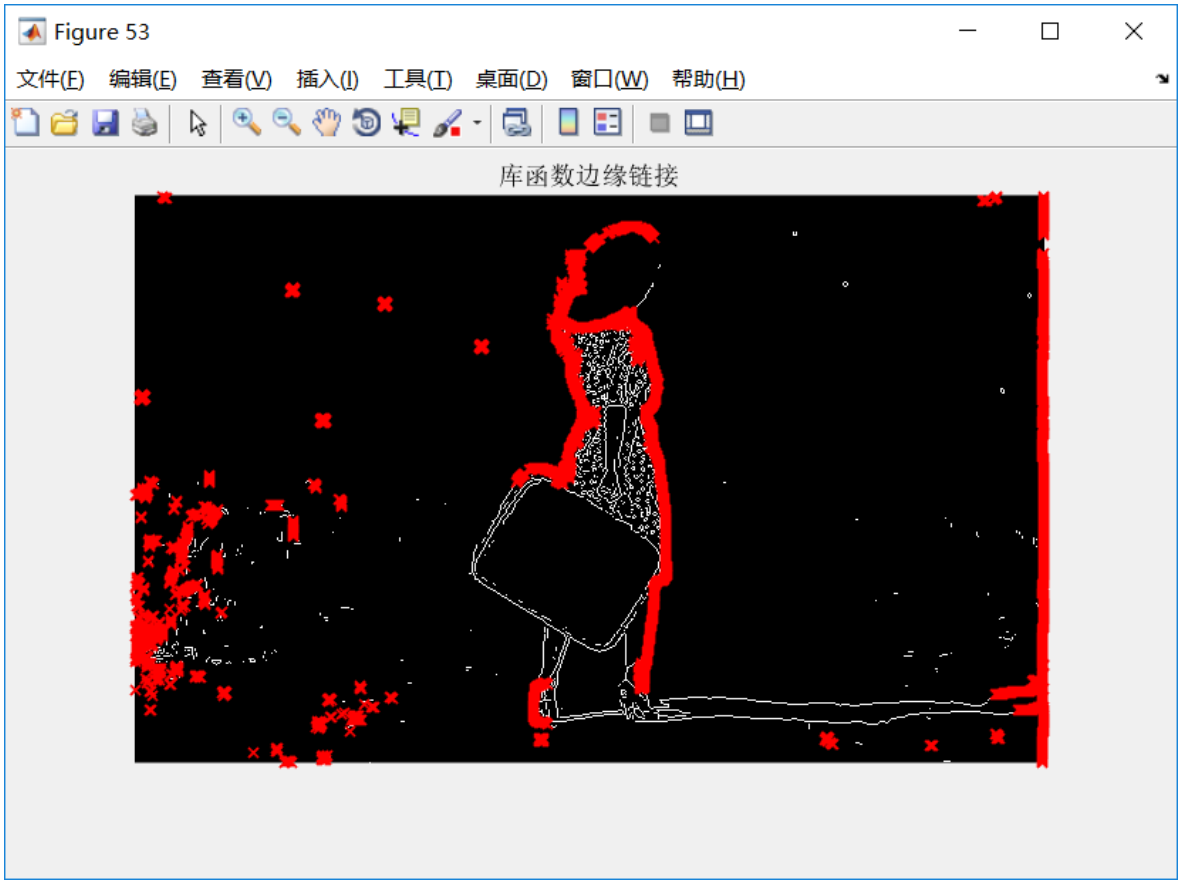


Figure 53
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

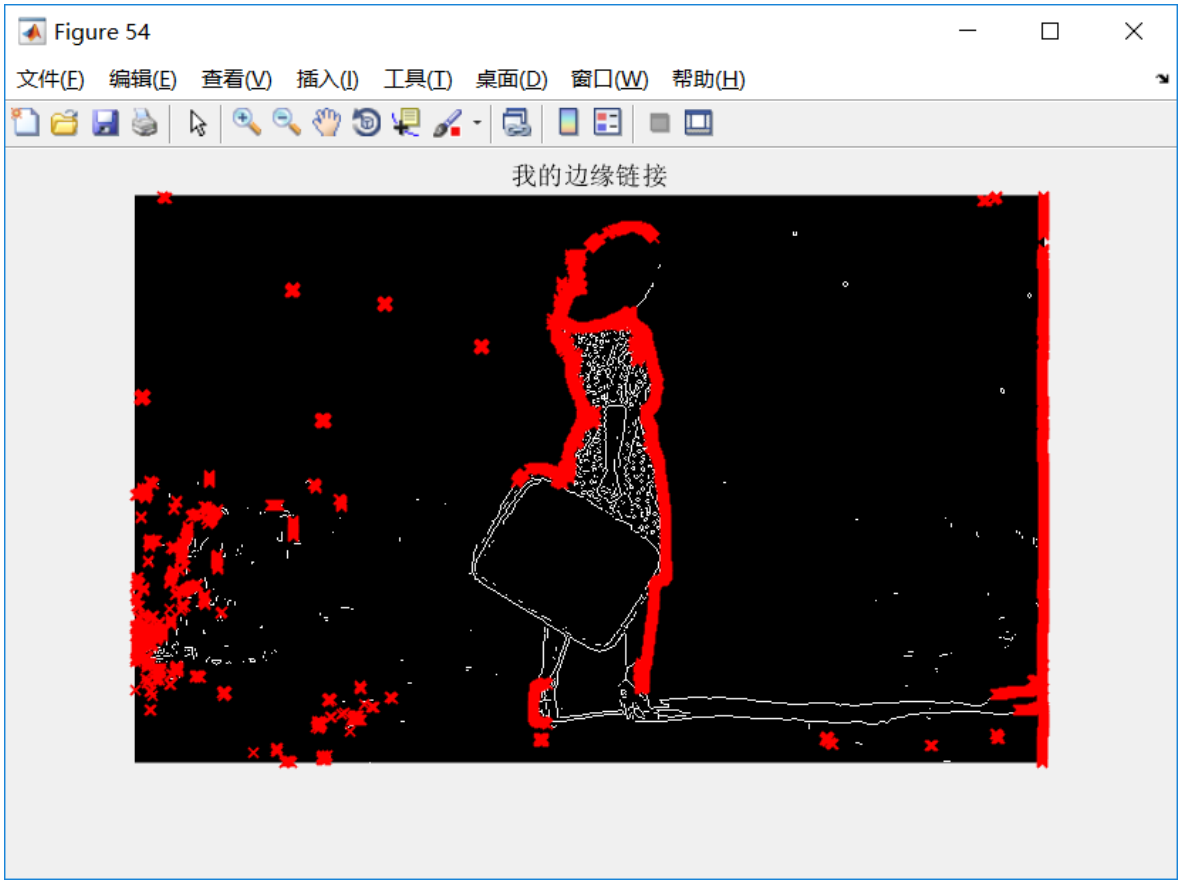
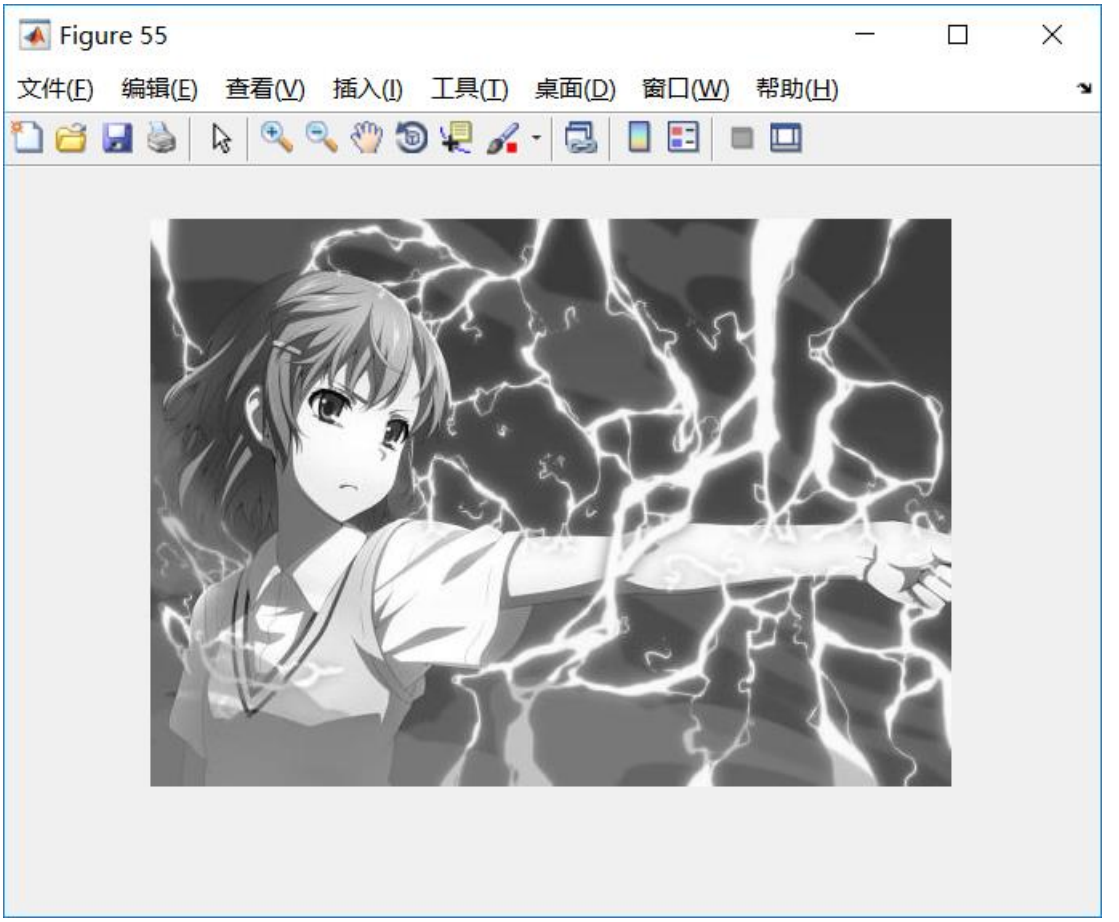


Figure 54
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.6 1.bmp



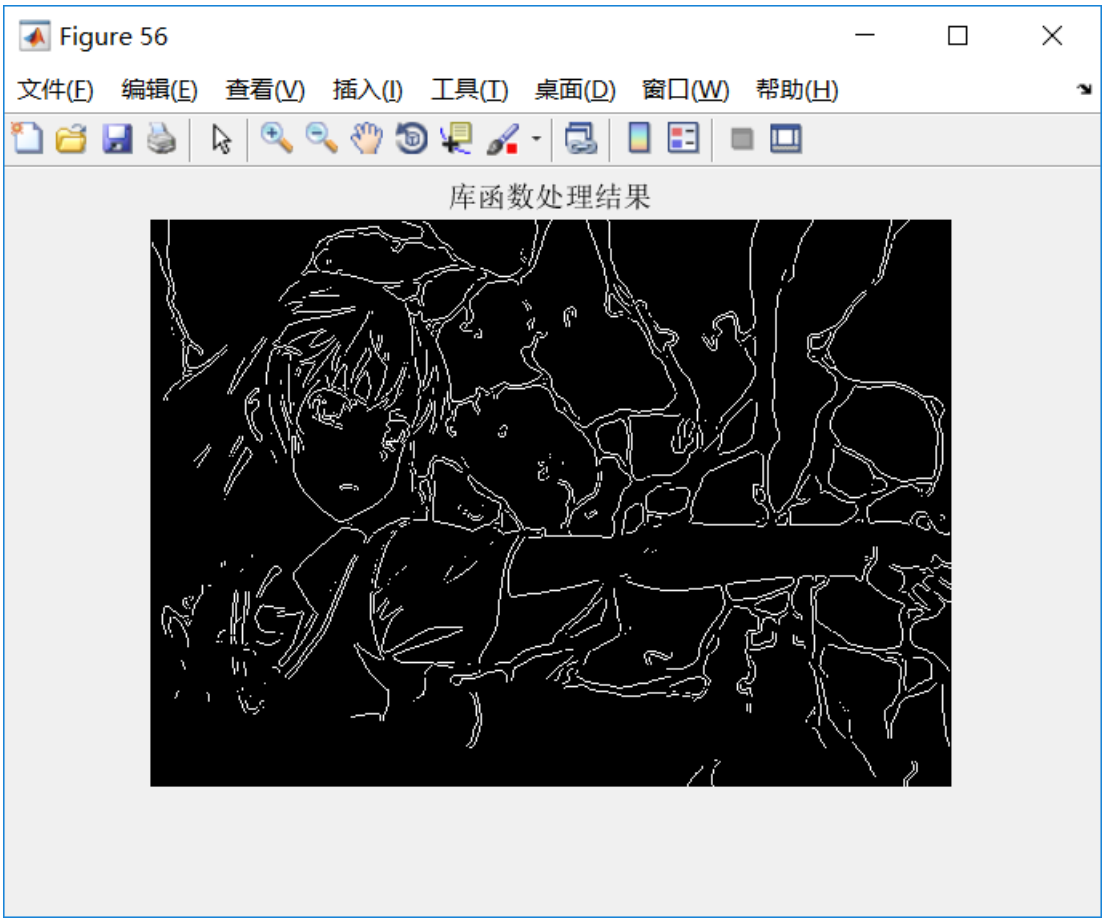


Figure 56
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

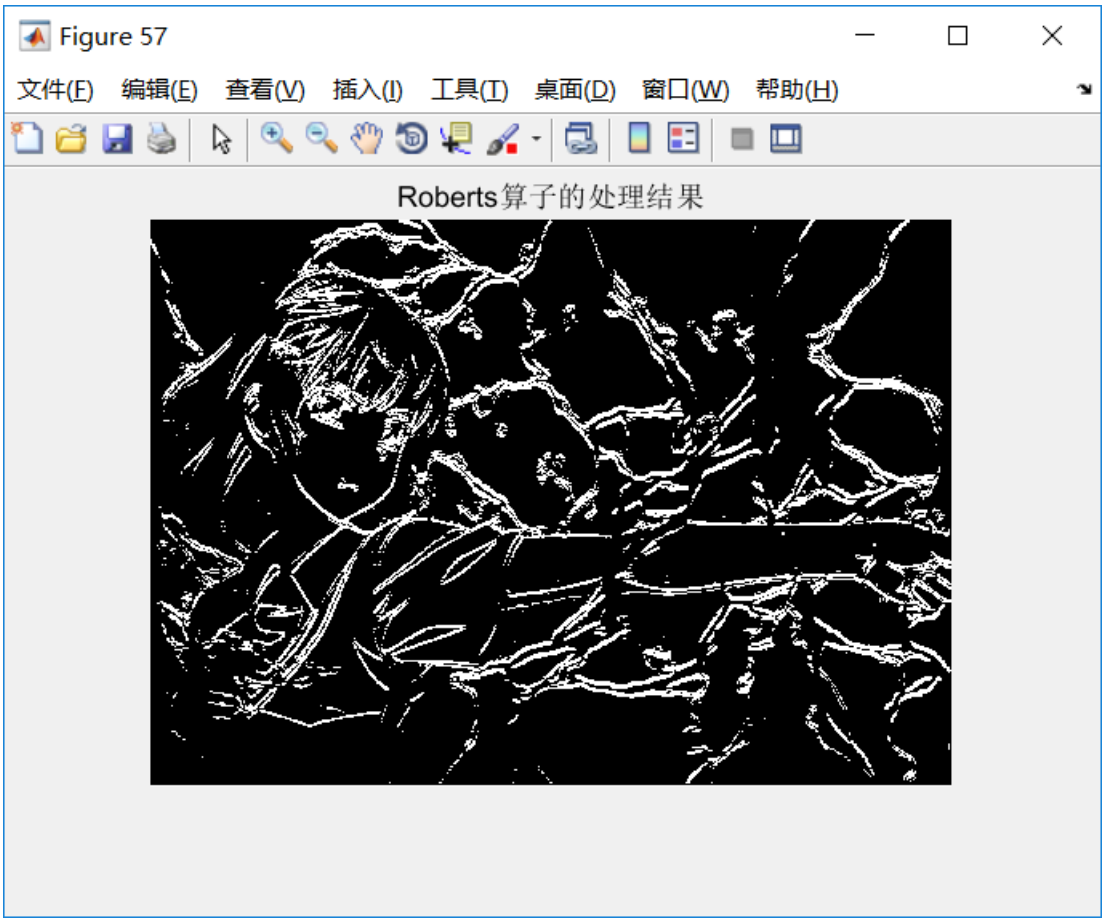


Figure 57
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

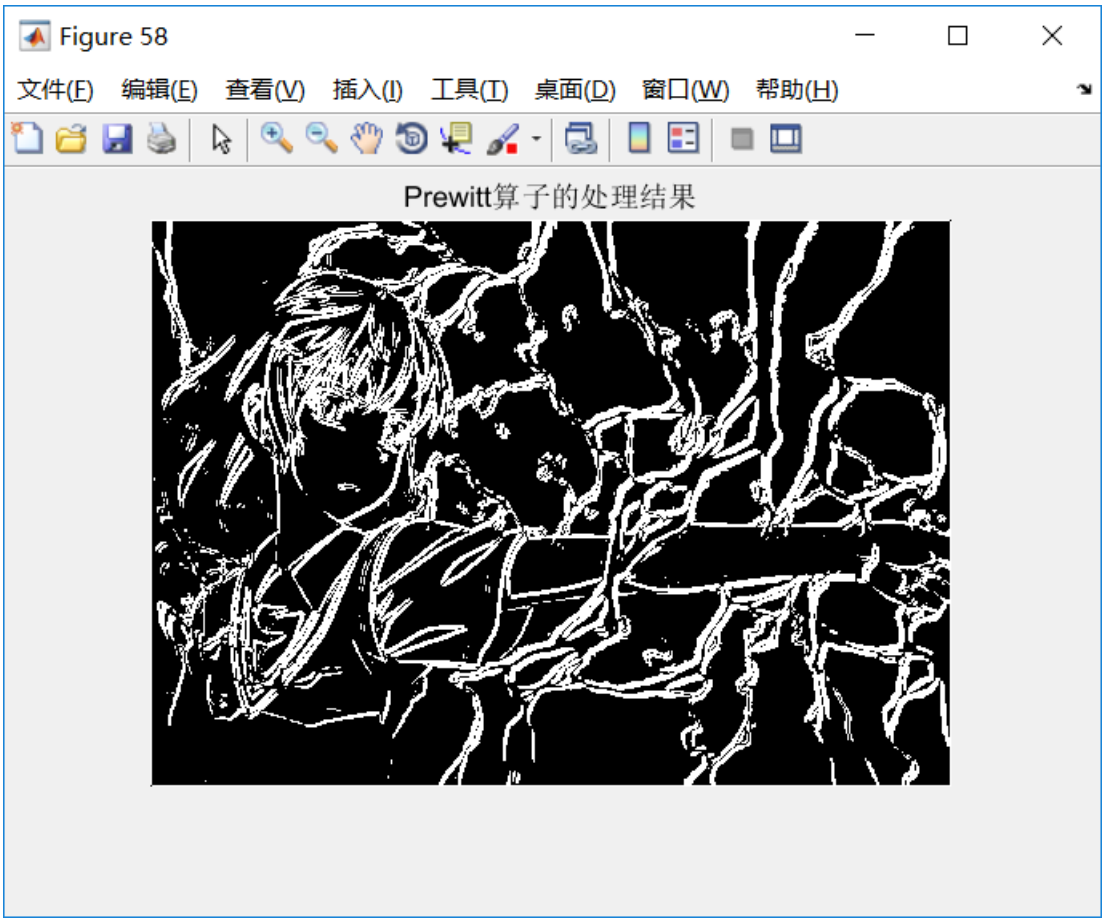


Figure 58
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

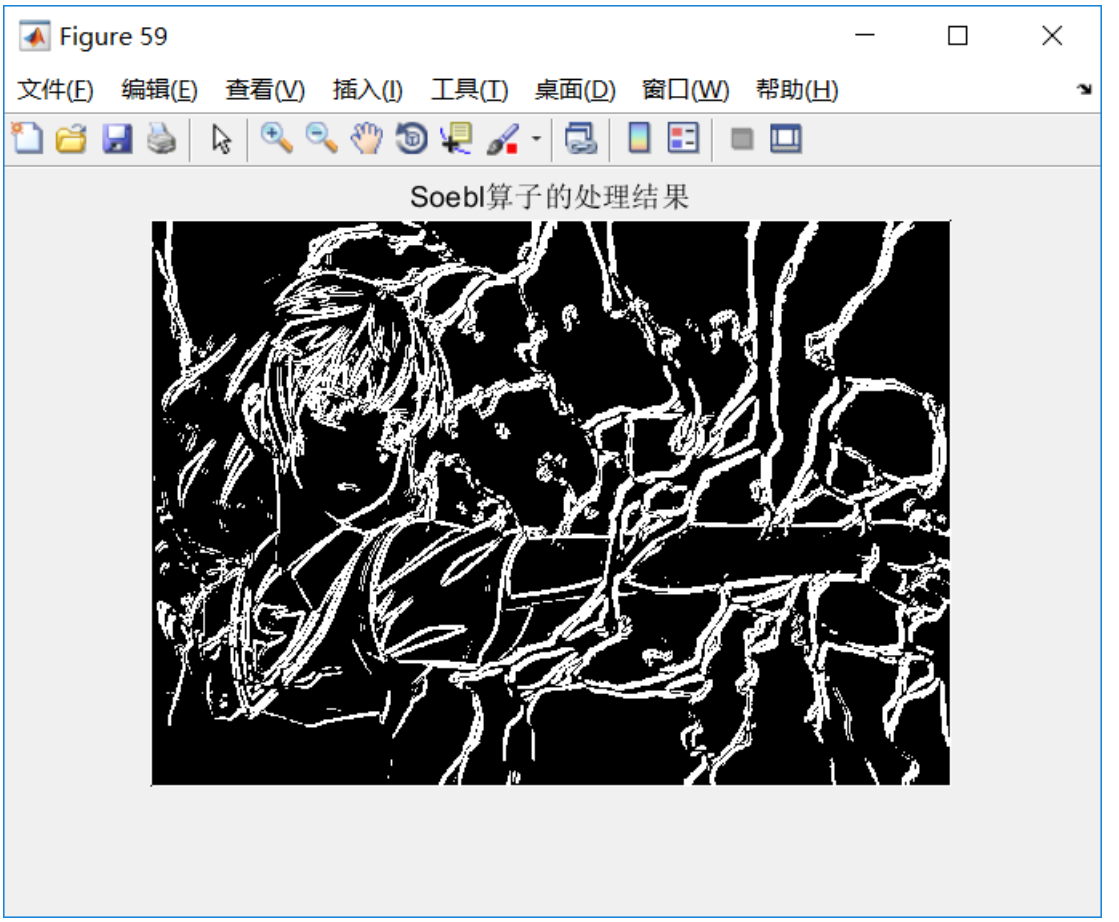


Figure 59
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

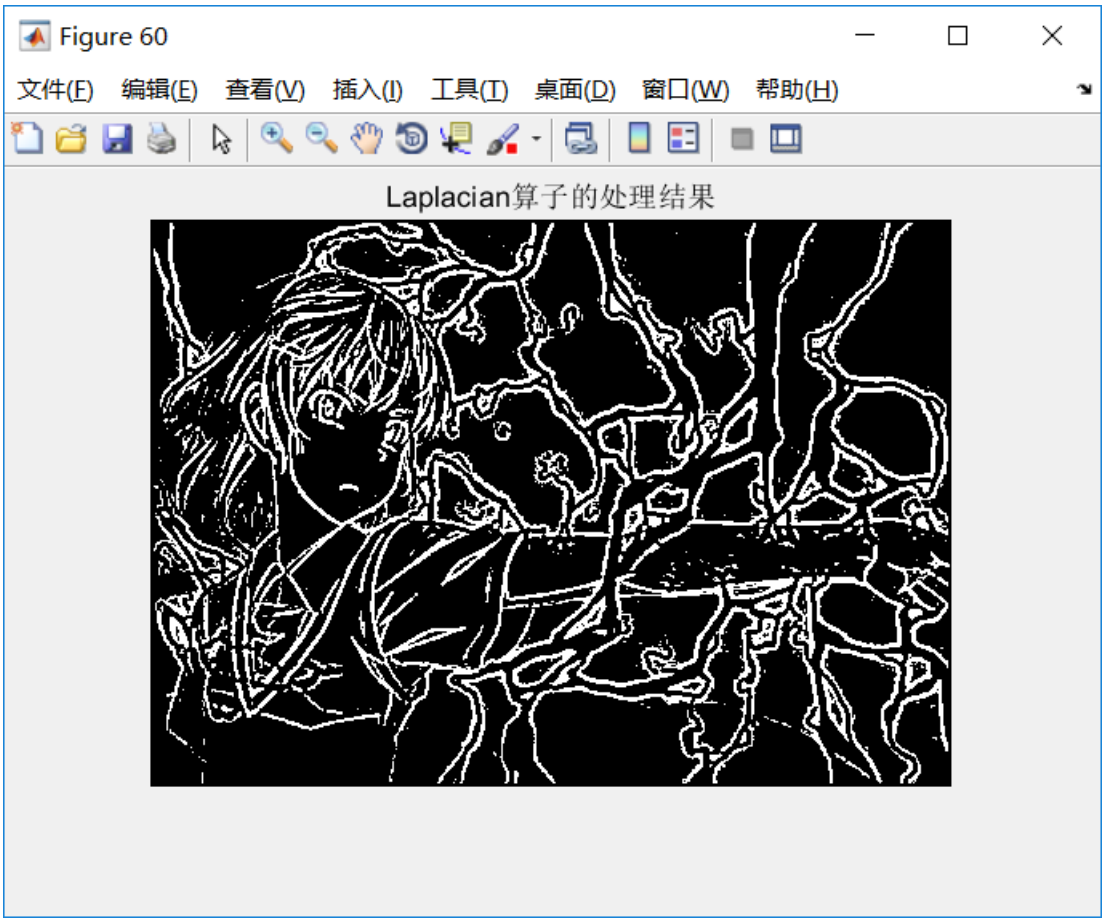


Figure 60
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

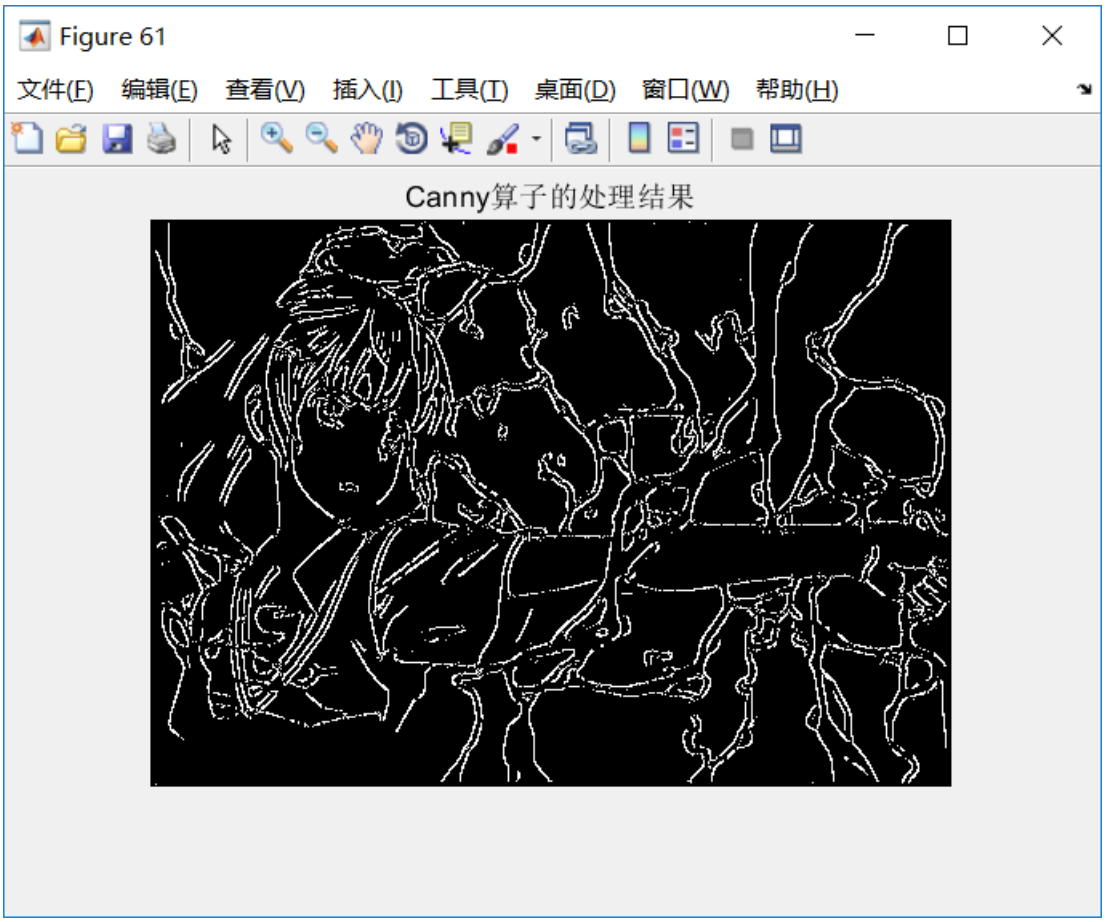


Figure 61
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

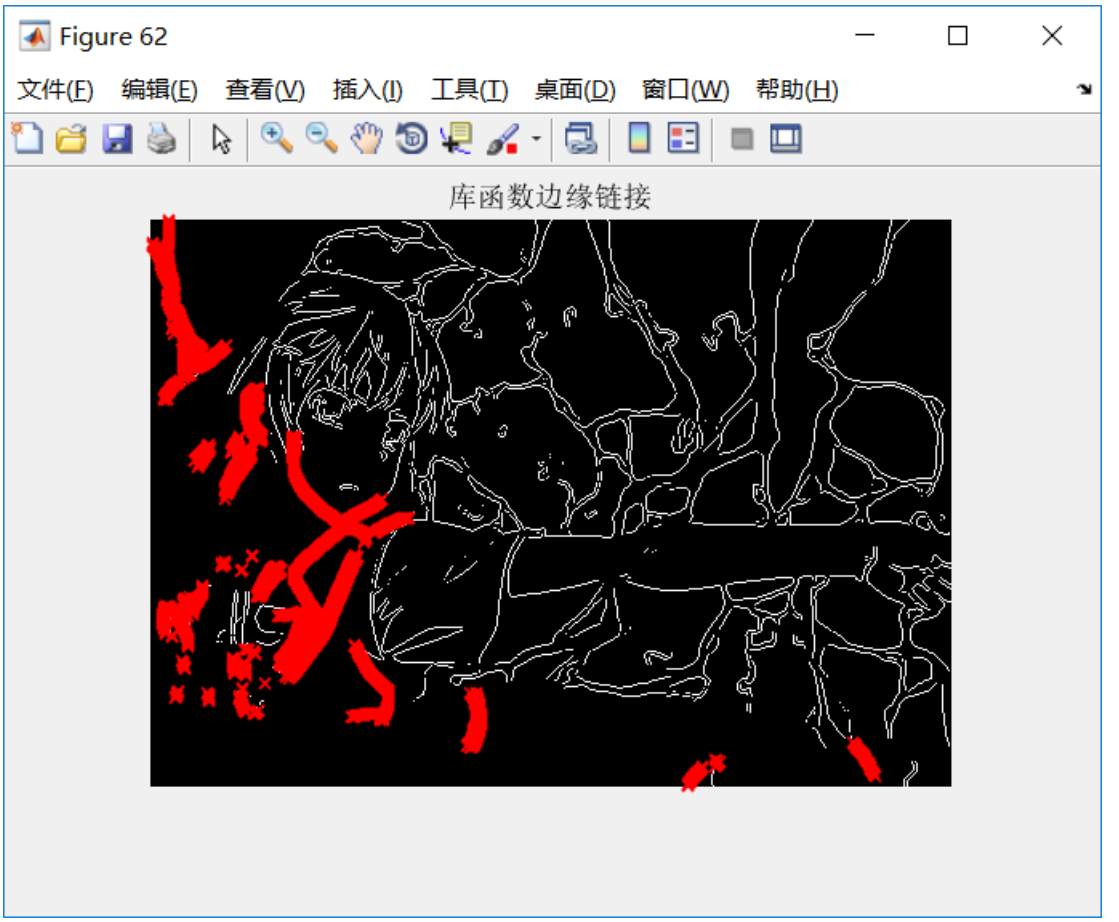


Figure 62
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

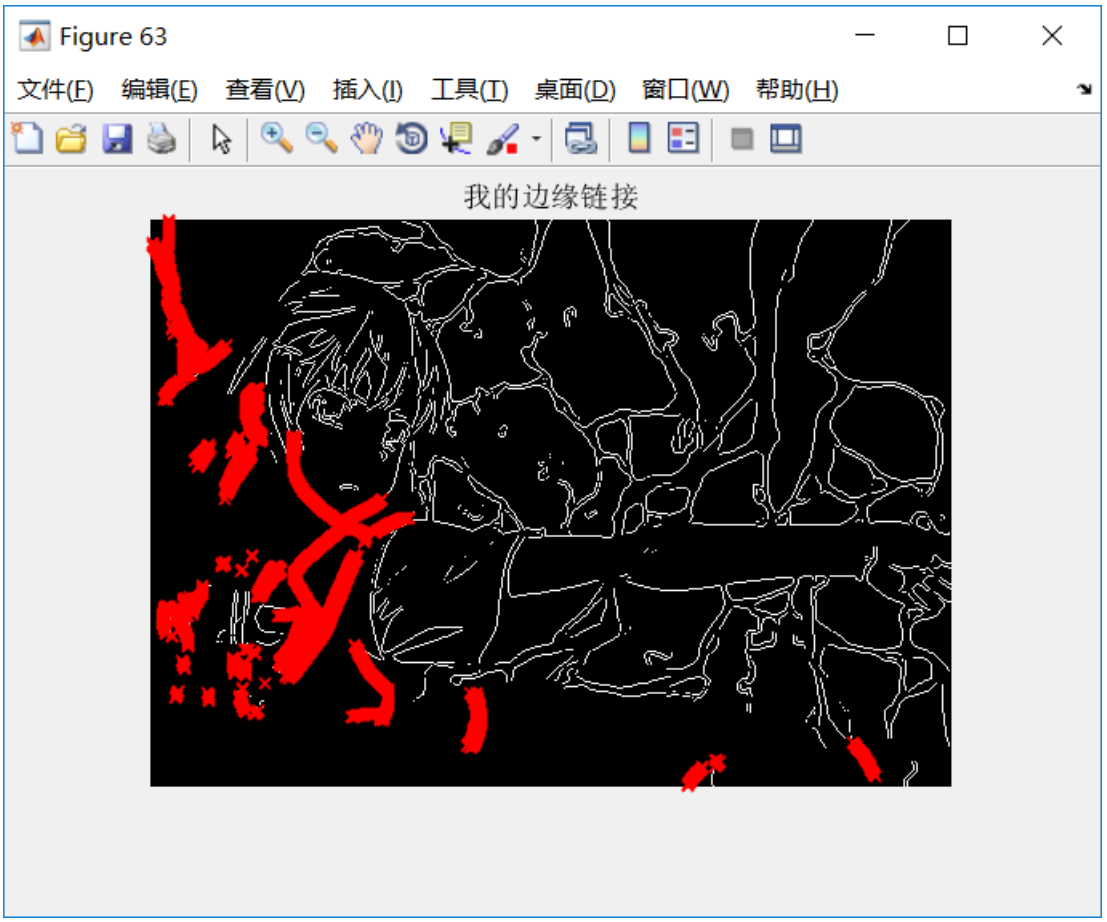
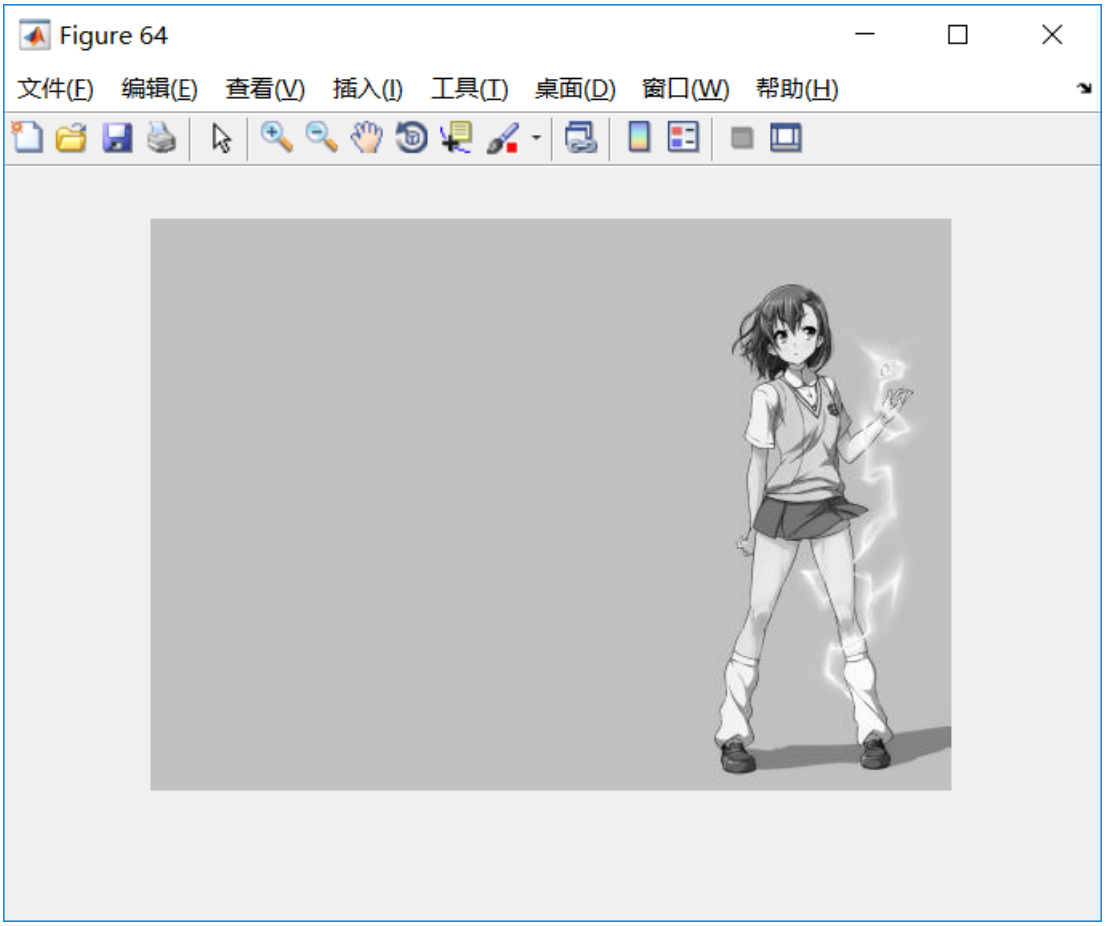


Figure 63
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.7 2.bmp



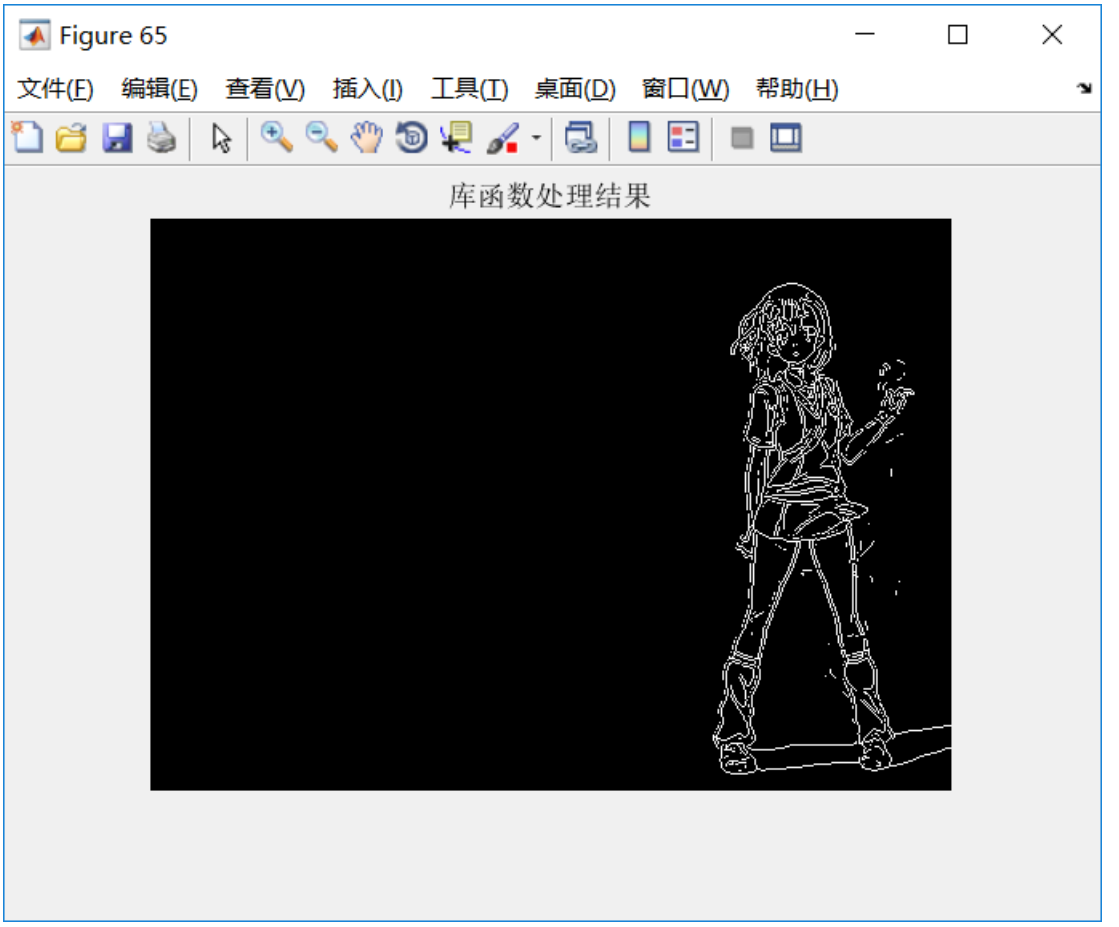


Figure 65
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

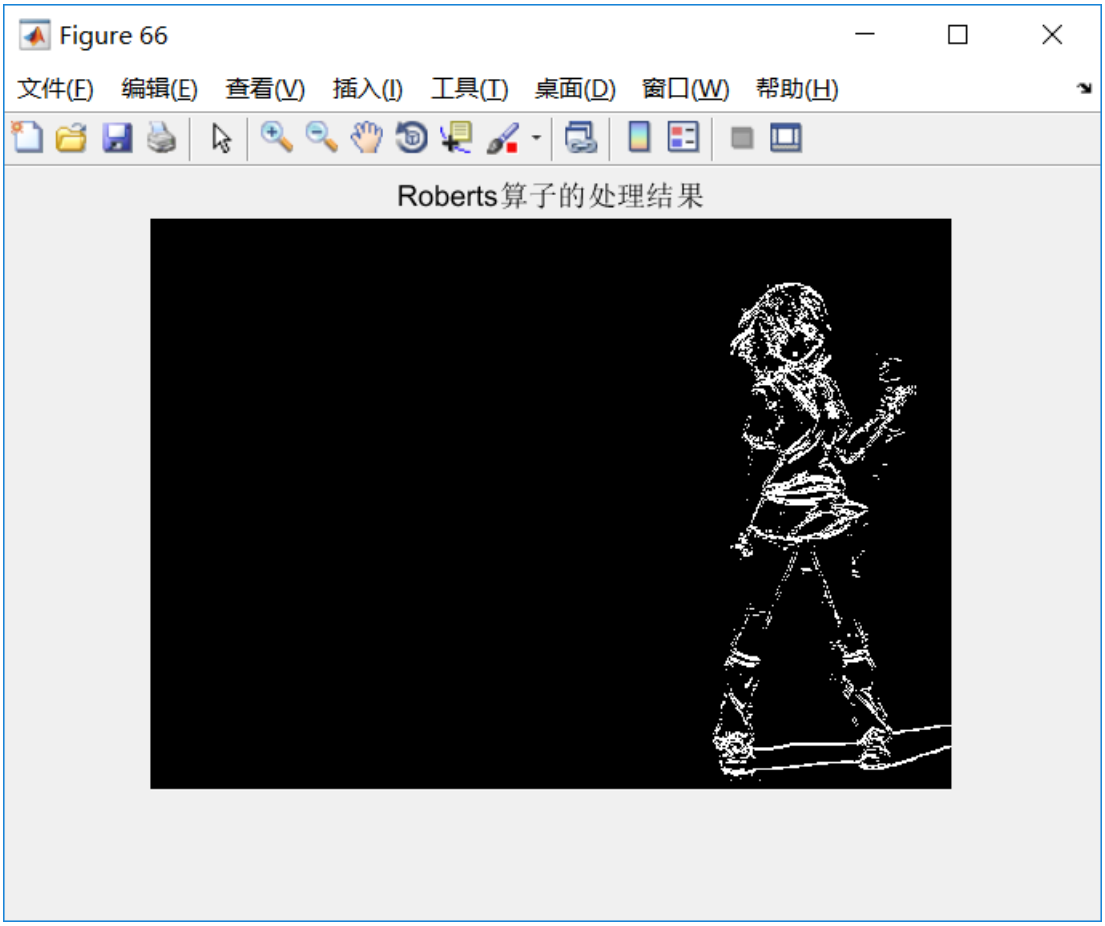


Figure 66
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

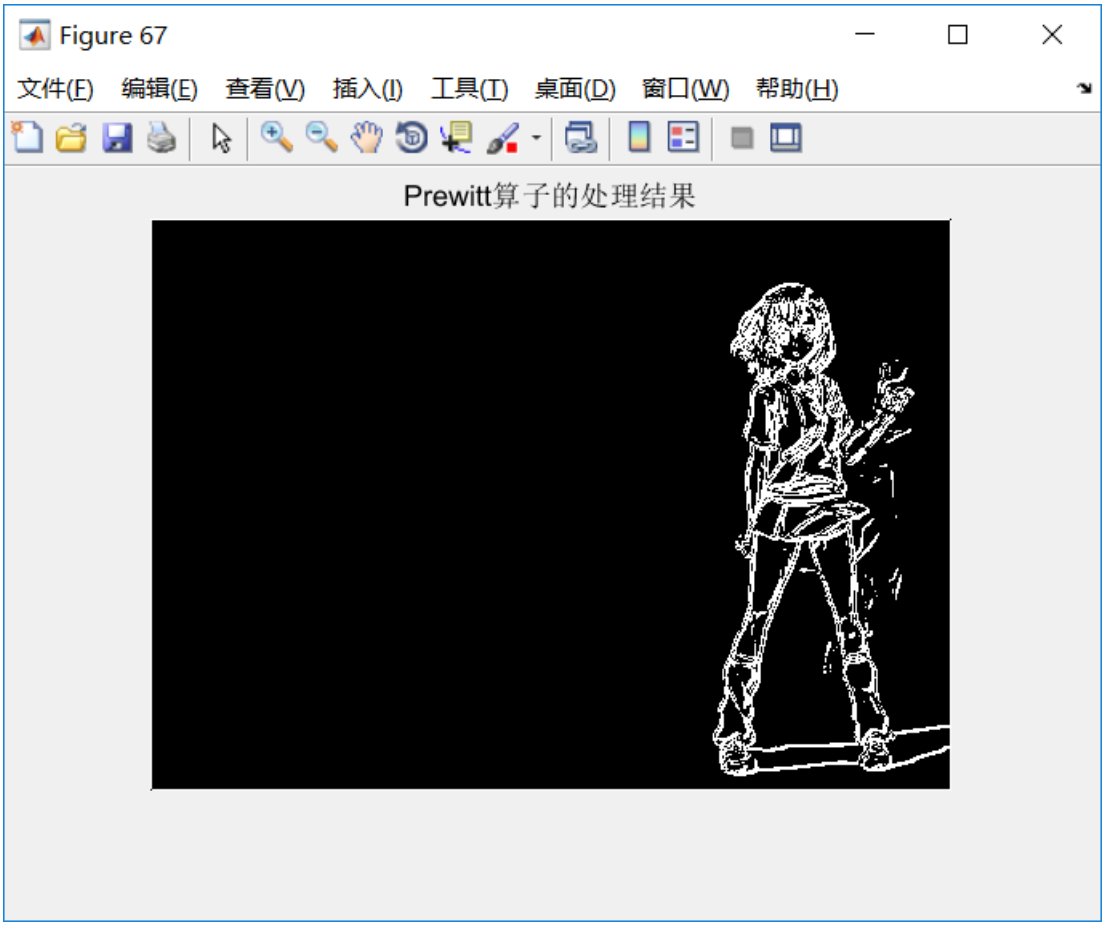


Figure 67
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

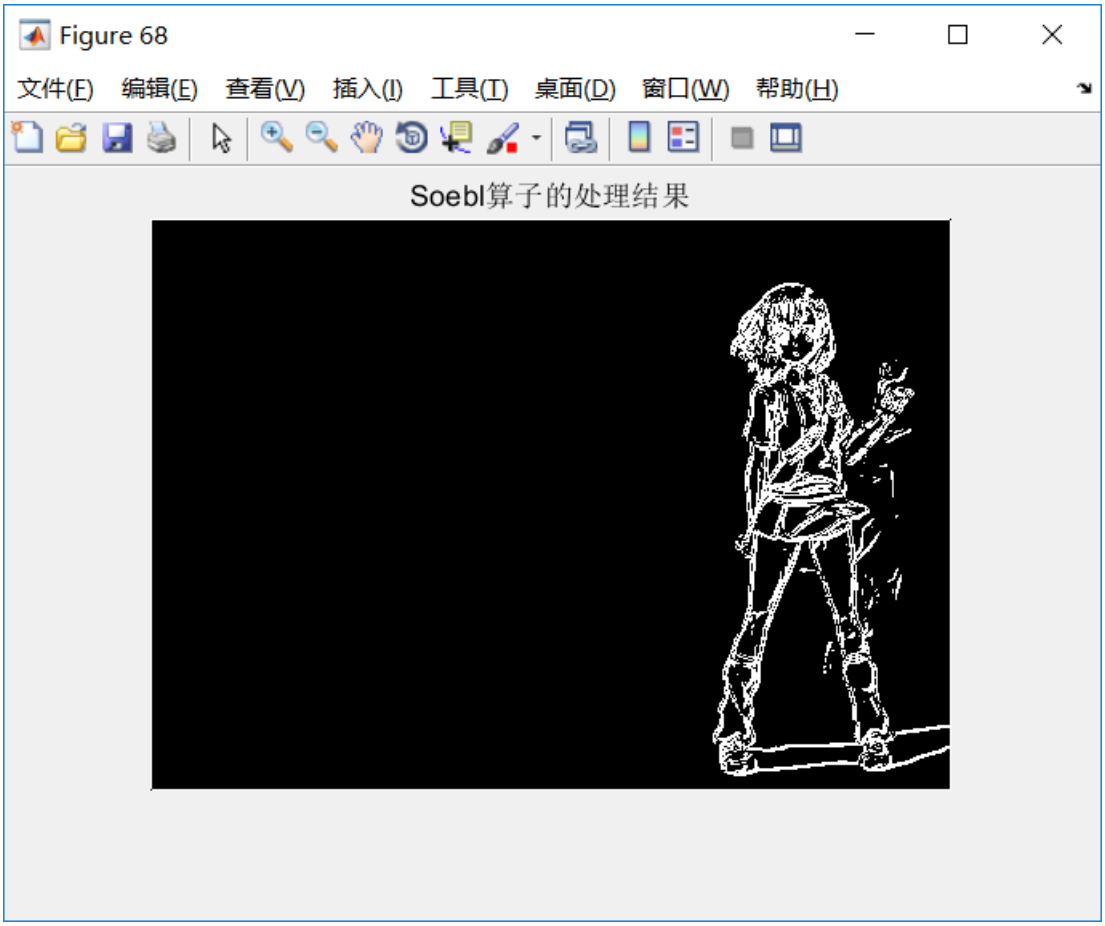


Figure 68
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

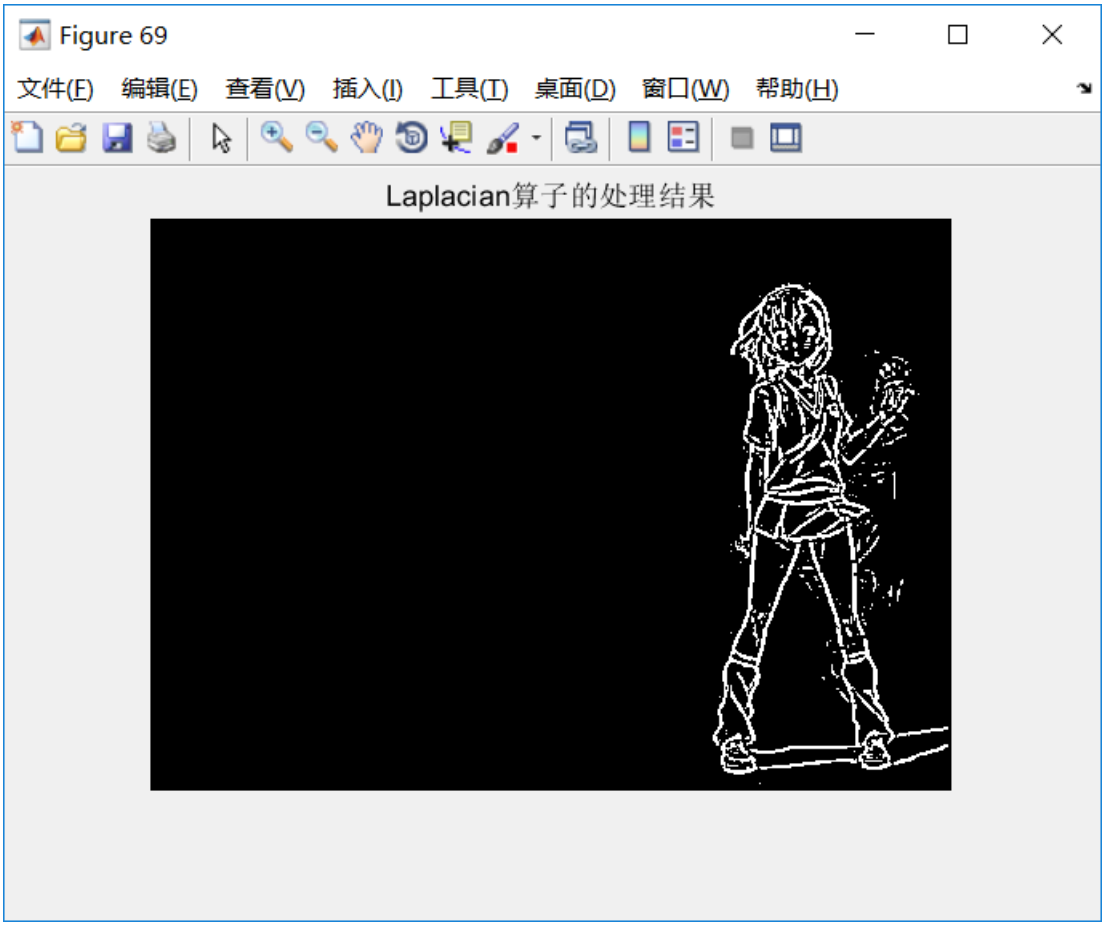


Figure 69
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

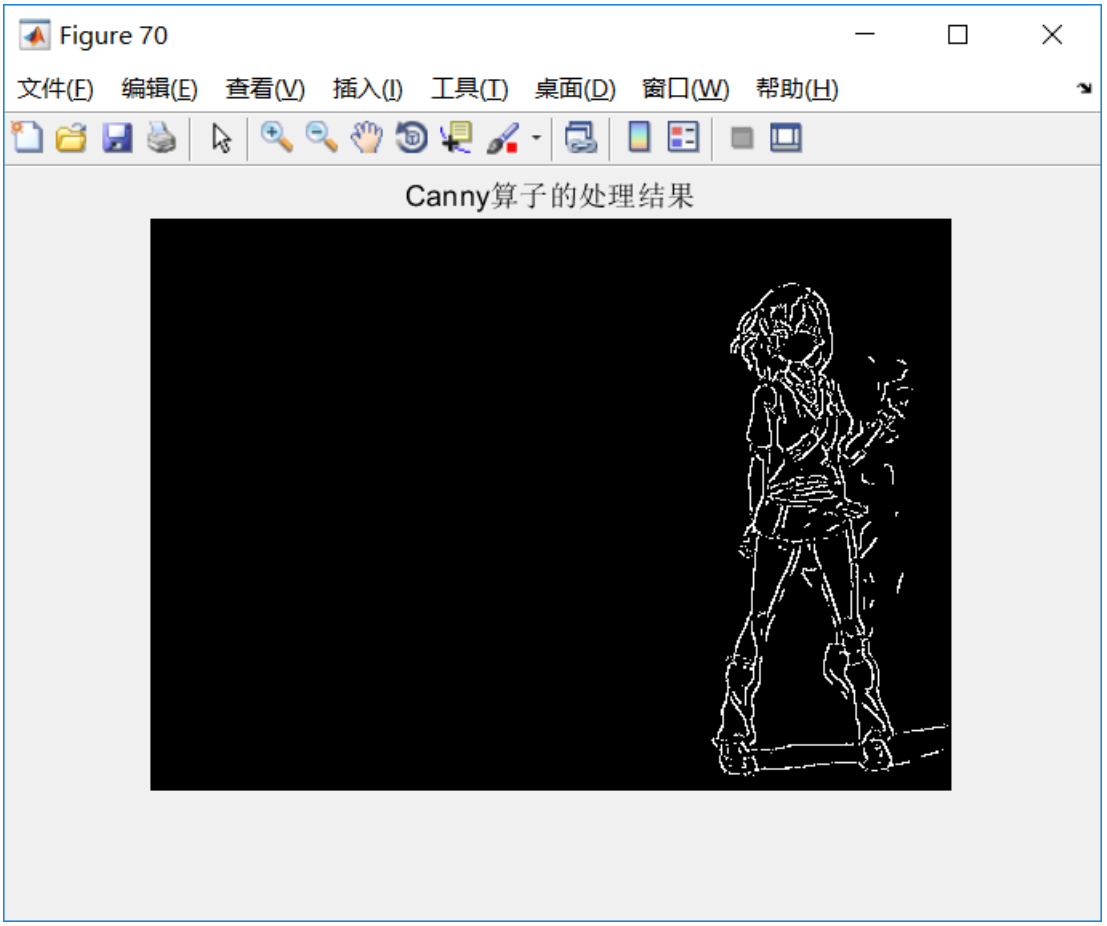


Figure 70
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

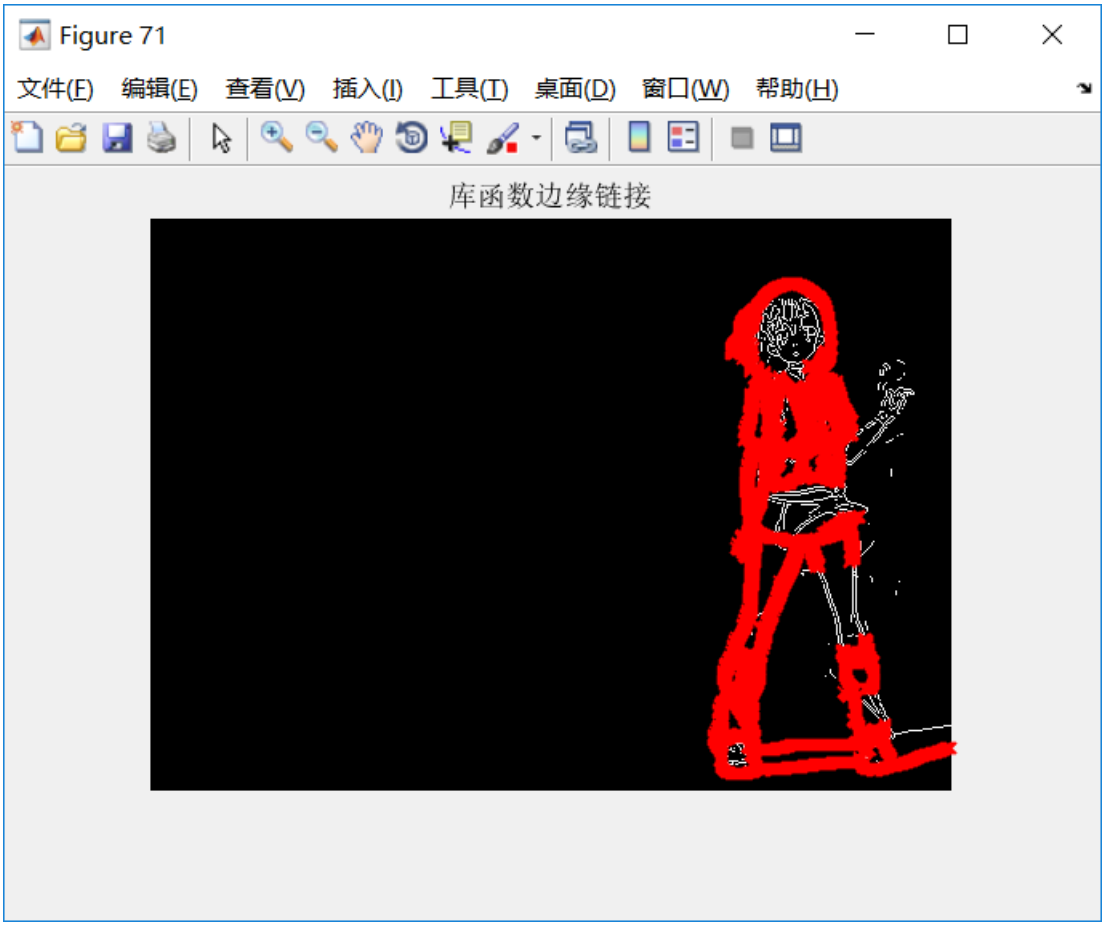


Figure 71
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

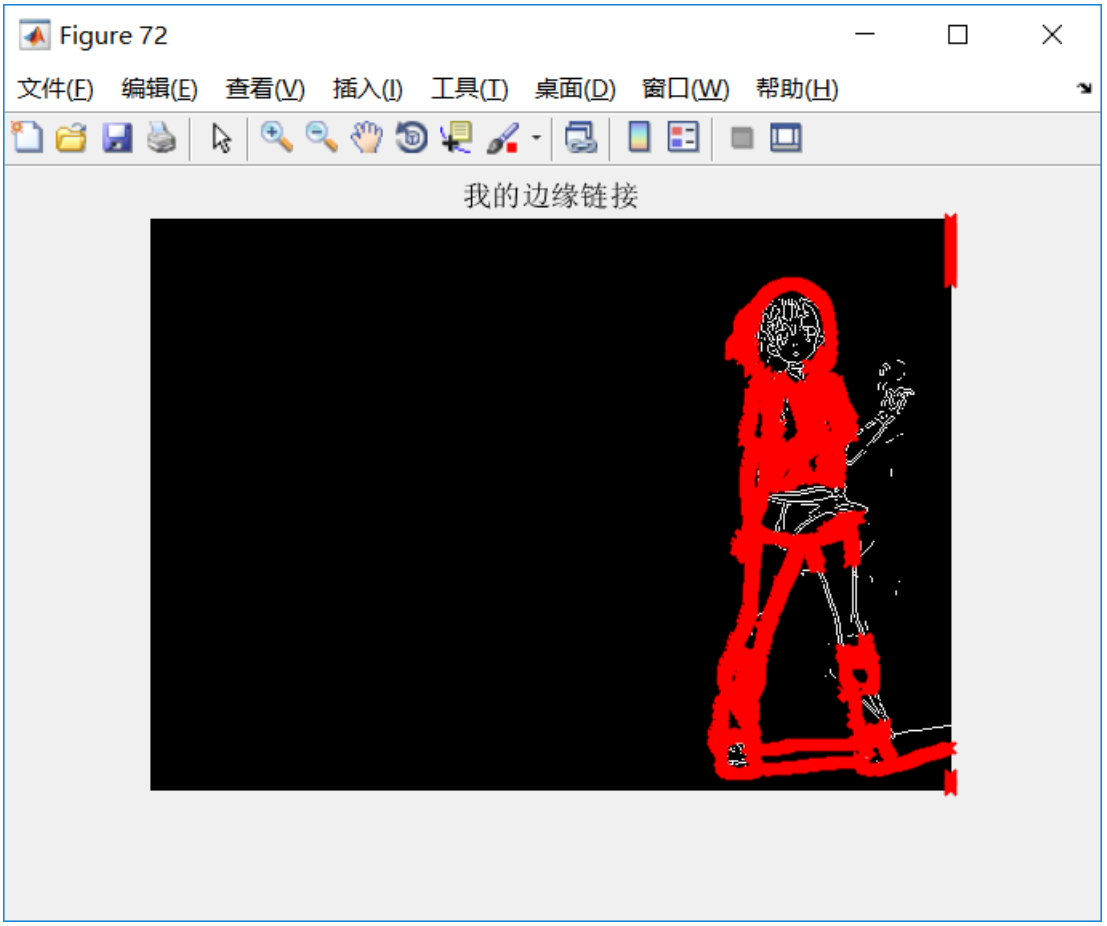
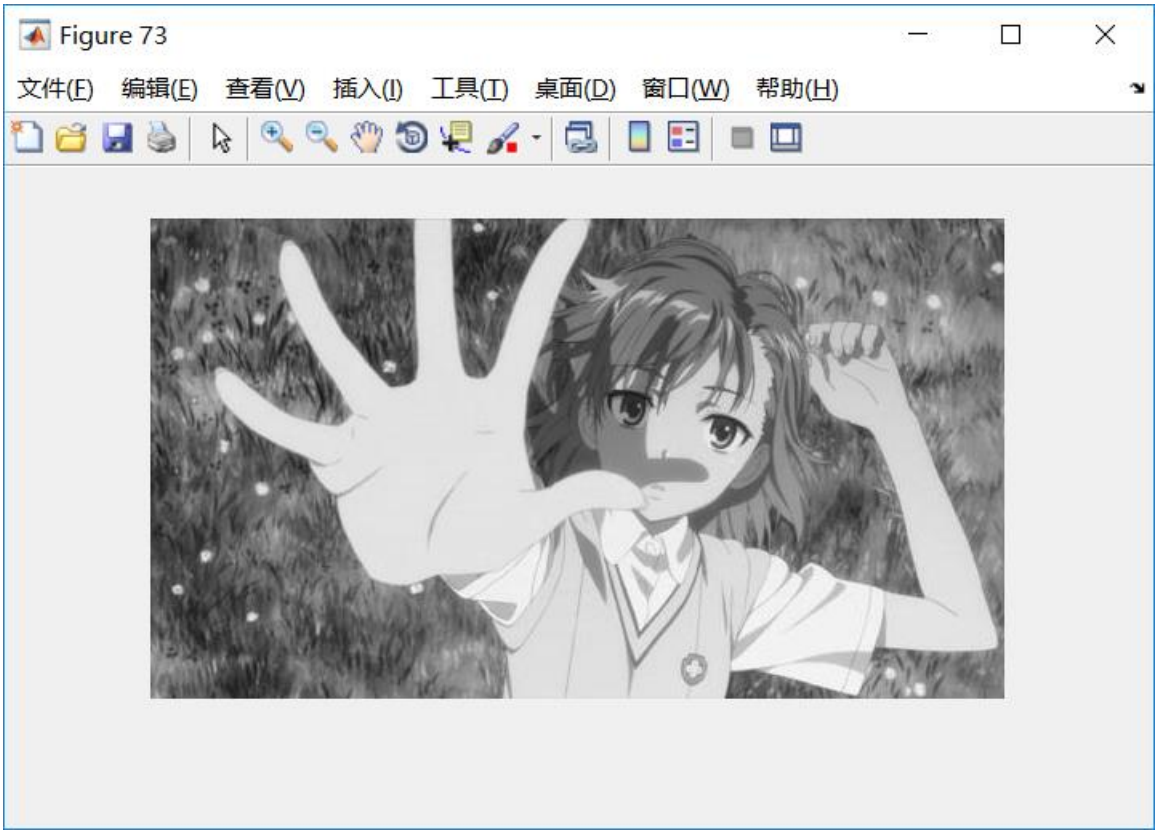


Figure 72
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.8 3.bmp



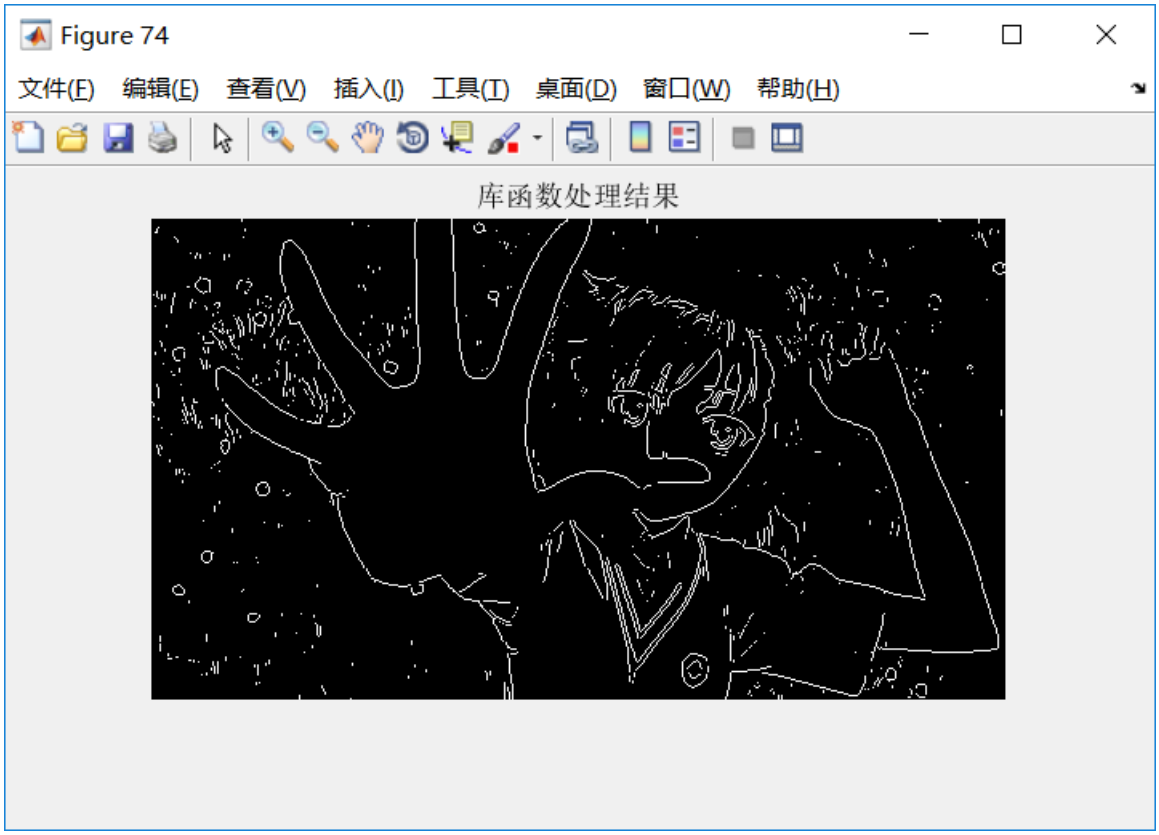


Figure 74
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数处理结果

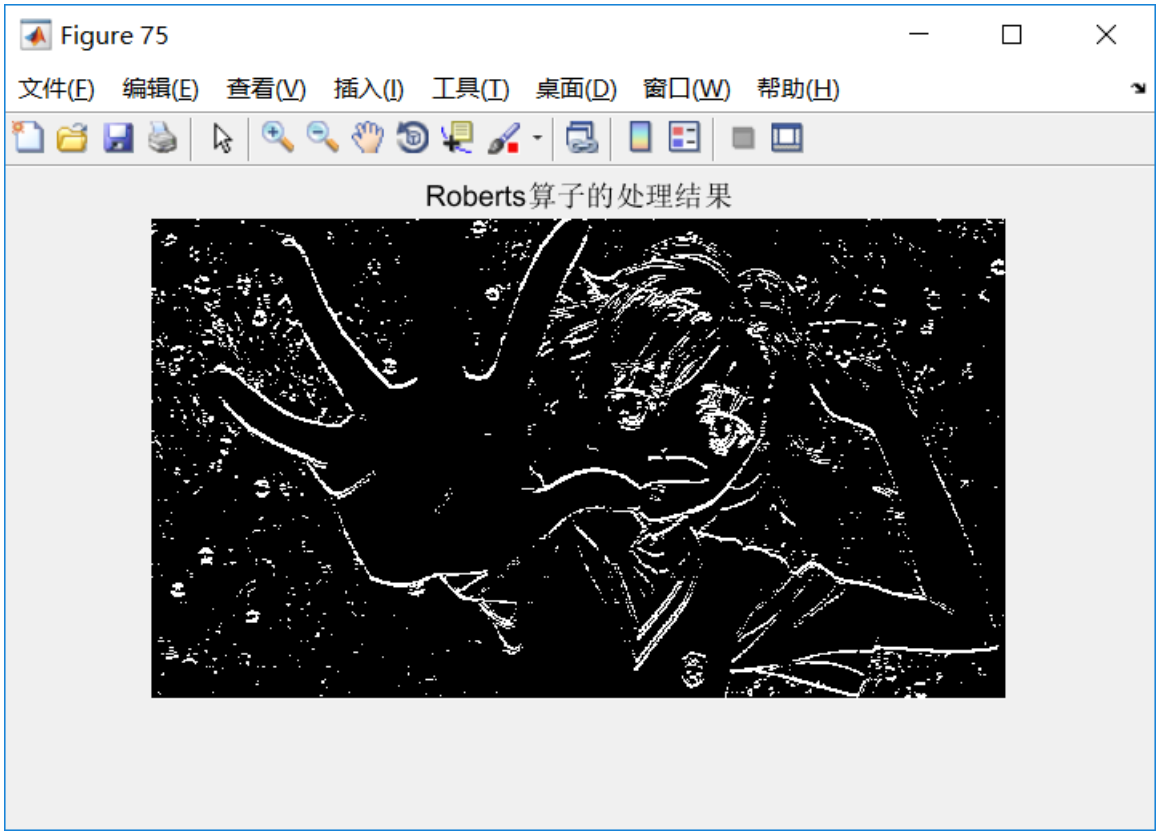


Figure 75
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

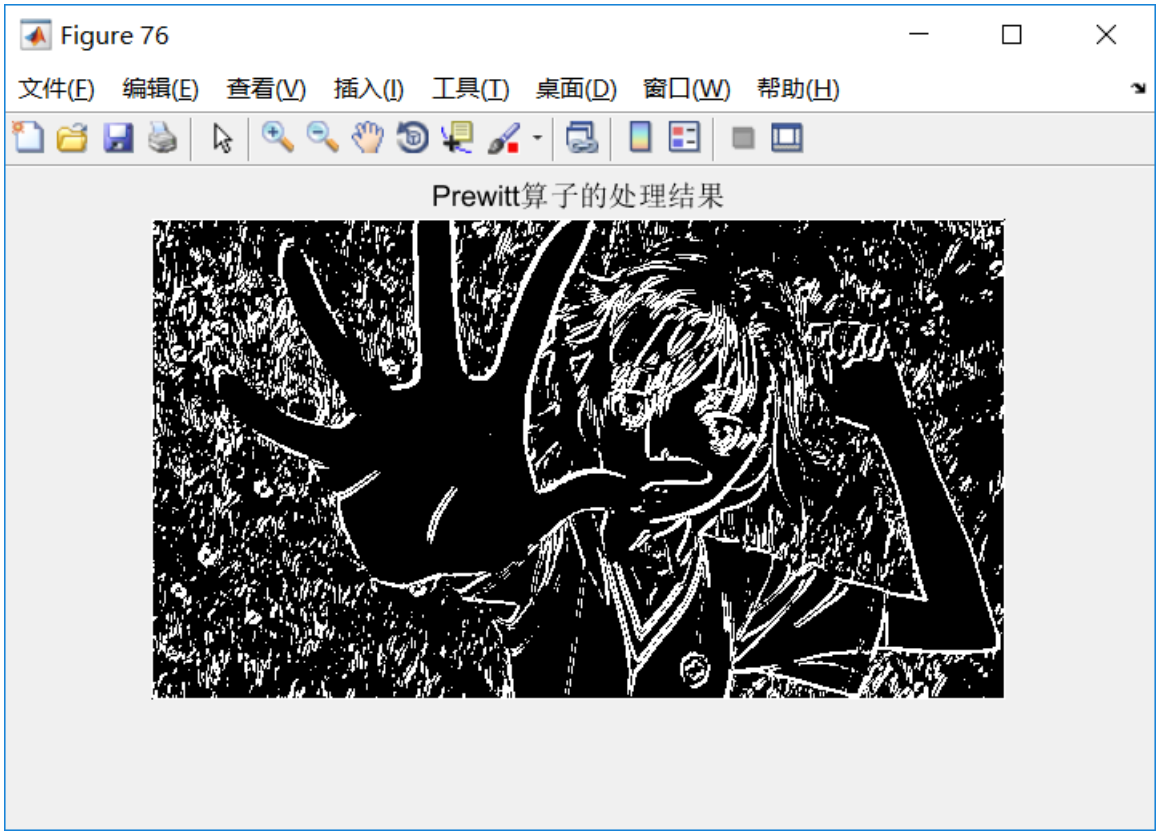


Figure 76
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

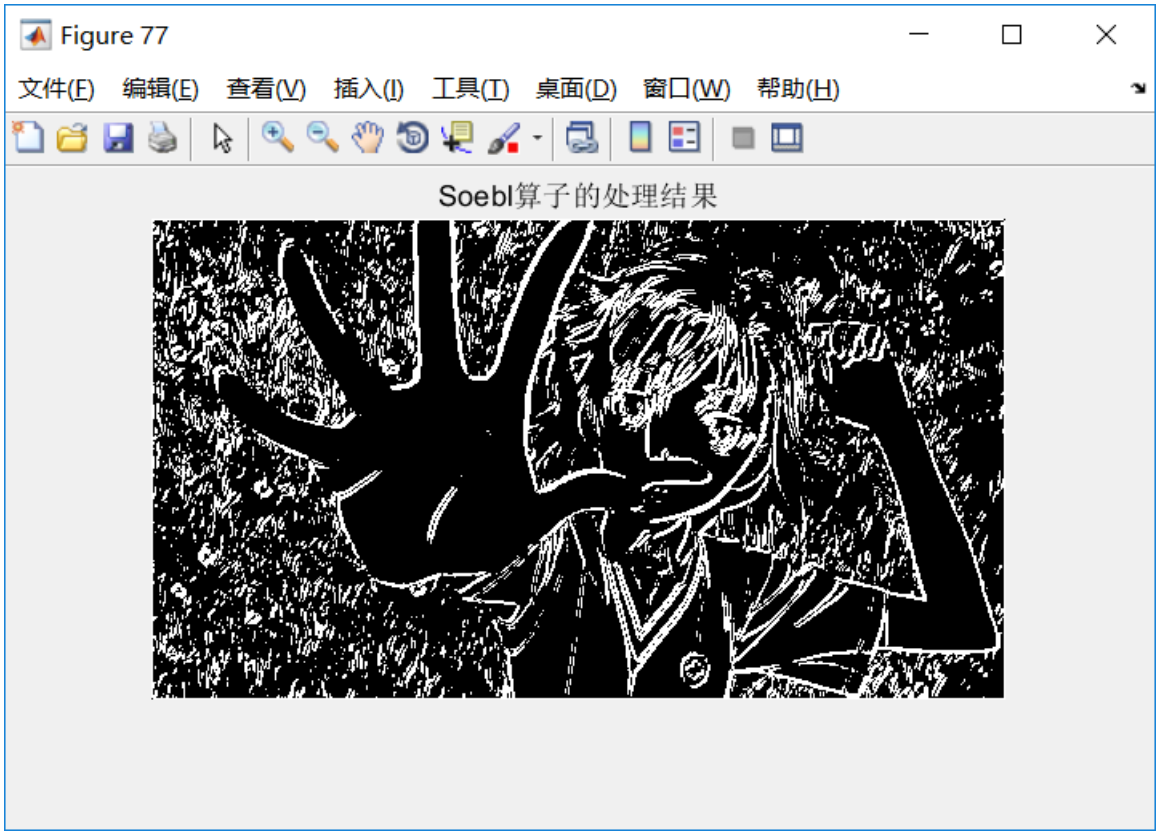


Figure 77
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

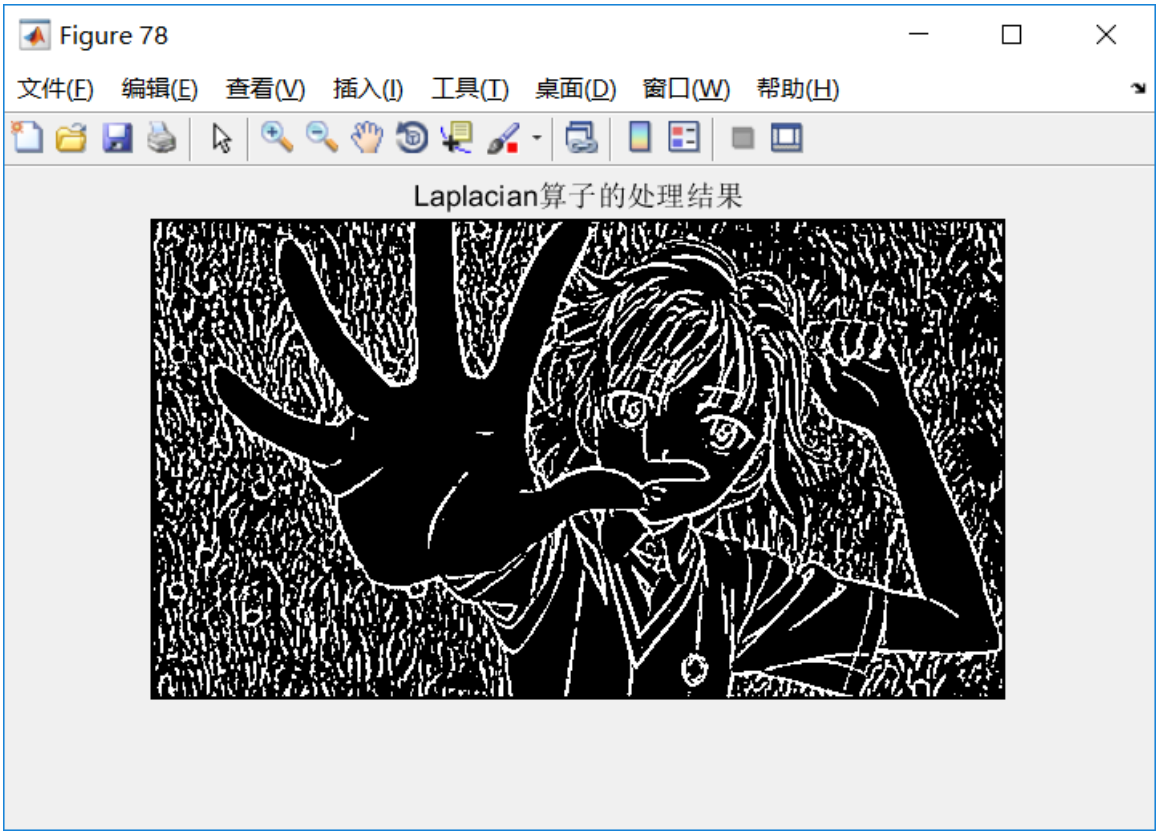


Figure 78
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

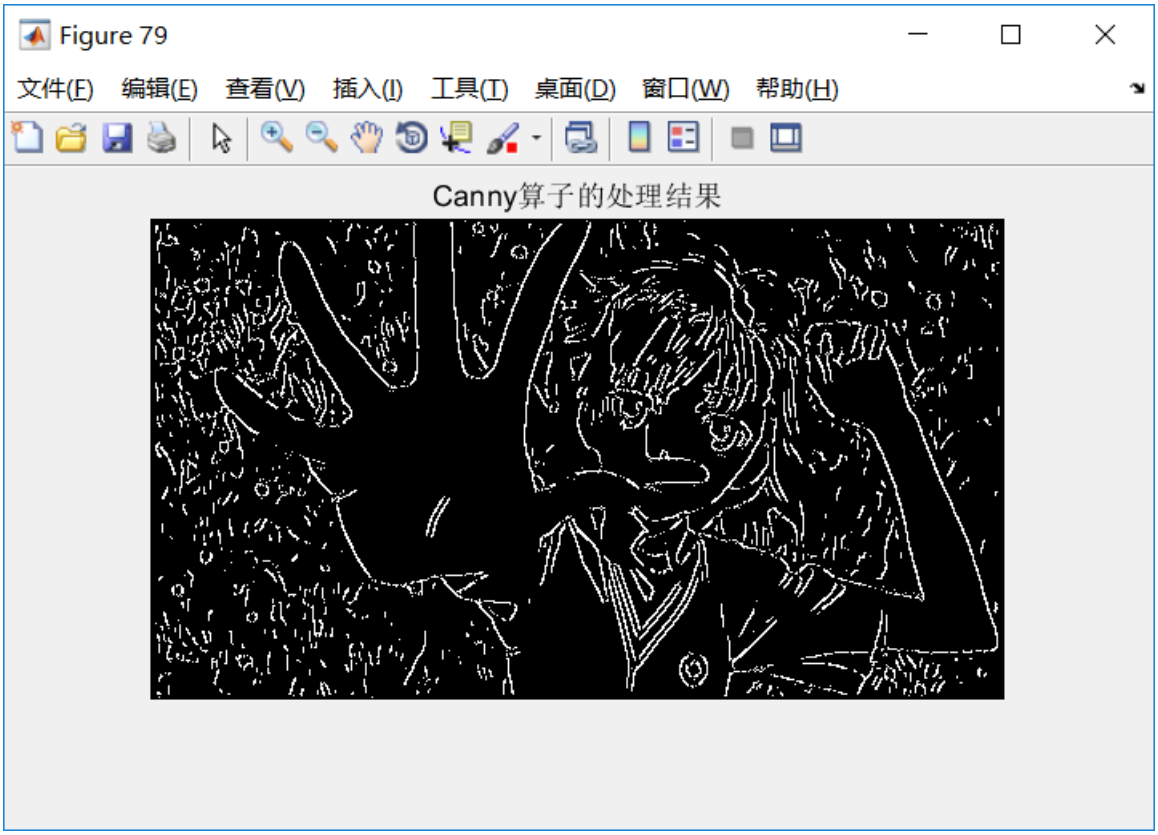


Figure 79
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Canny算子的处理结果

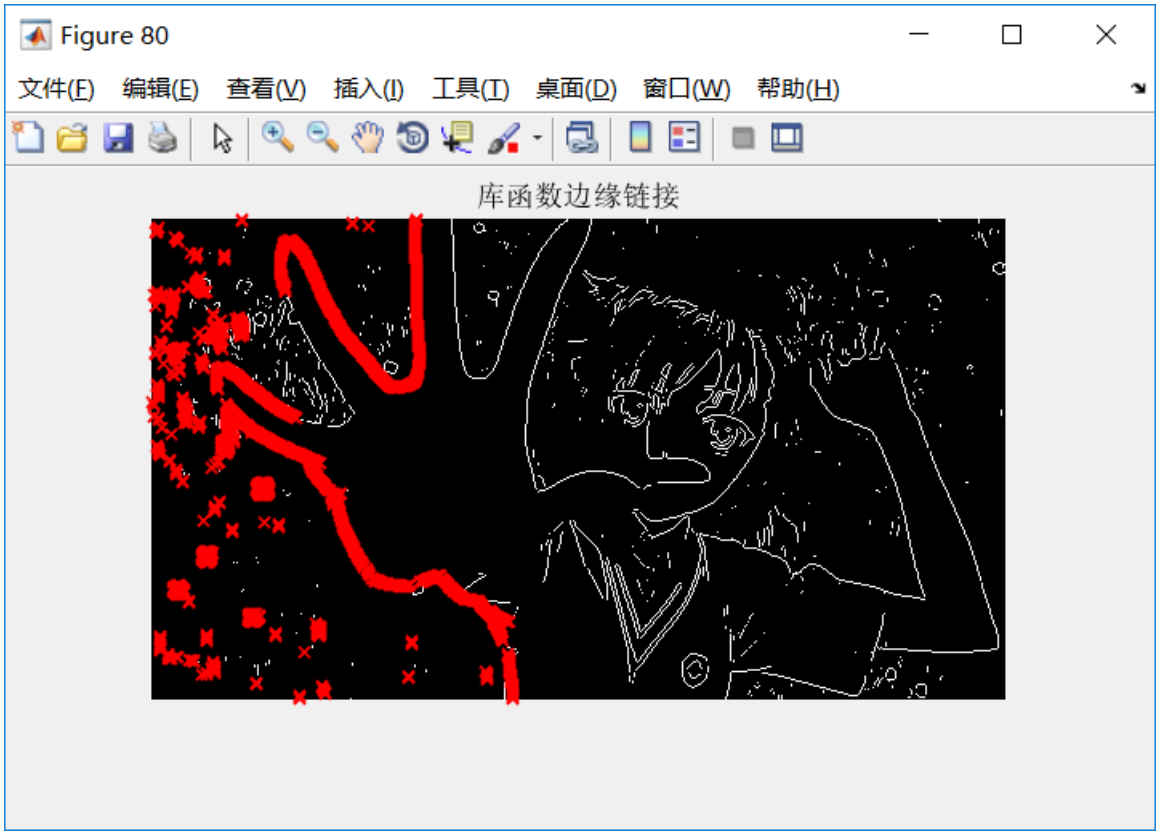


Figure 80
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
库函数边缘链接

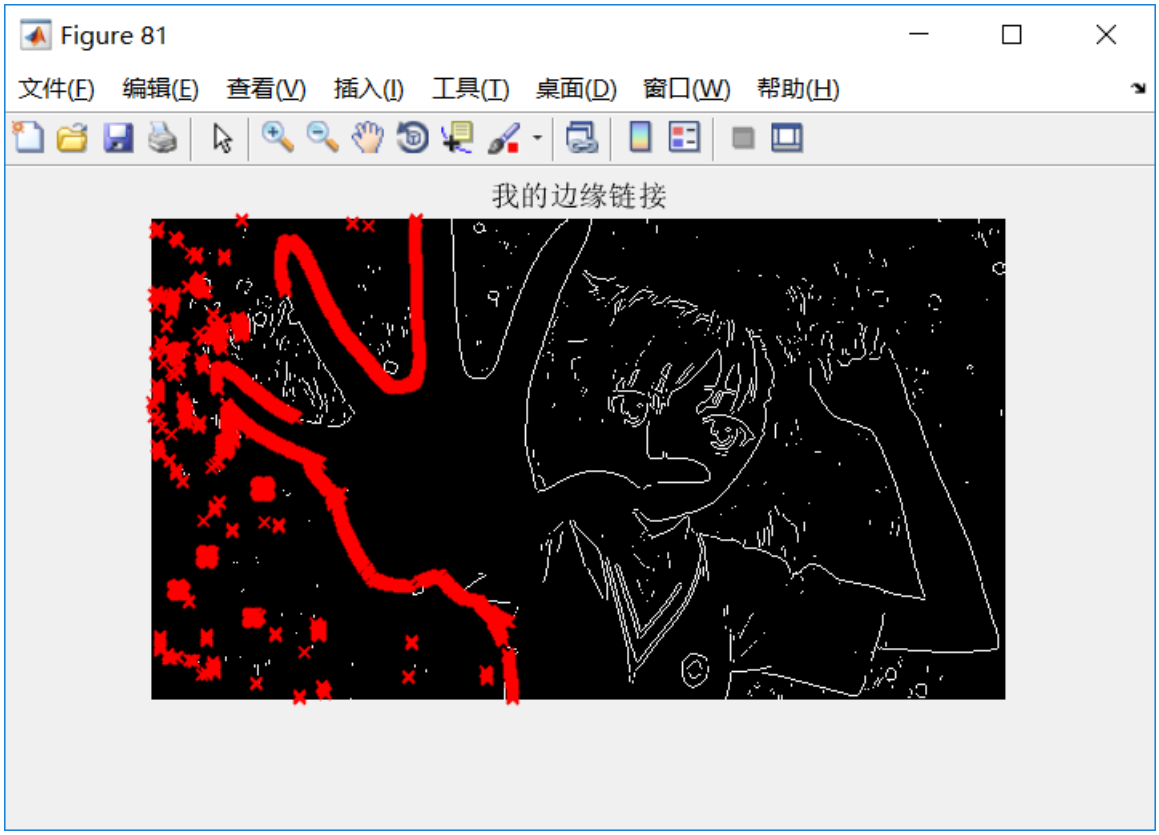
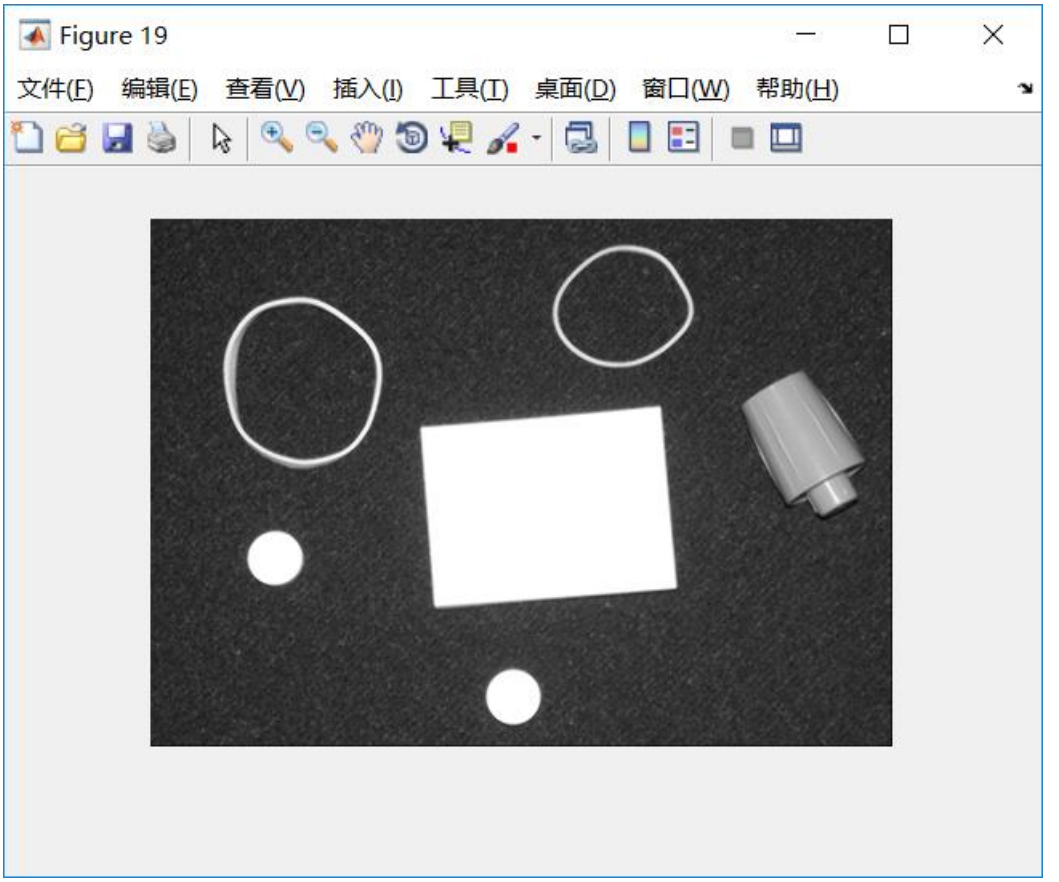


Figure 81
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
我的边缘链接

2.2.9　注意到 rubberband_cup.png、noise.png 和 noise2.png 三张图片的 Roberts、Prewitt、Sobel 和 Laplacian 算法出现了噪点爆炸的问题，我对这几张图片进行了中值滤波后再进行了测试。之所以选择中值滤波，是基于观察得出的，这几张图片的早点都是随机出现的白色噪点。新的测试结果如下：

2.2.9.1　Rubberband_cup.png



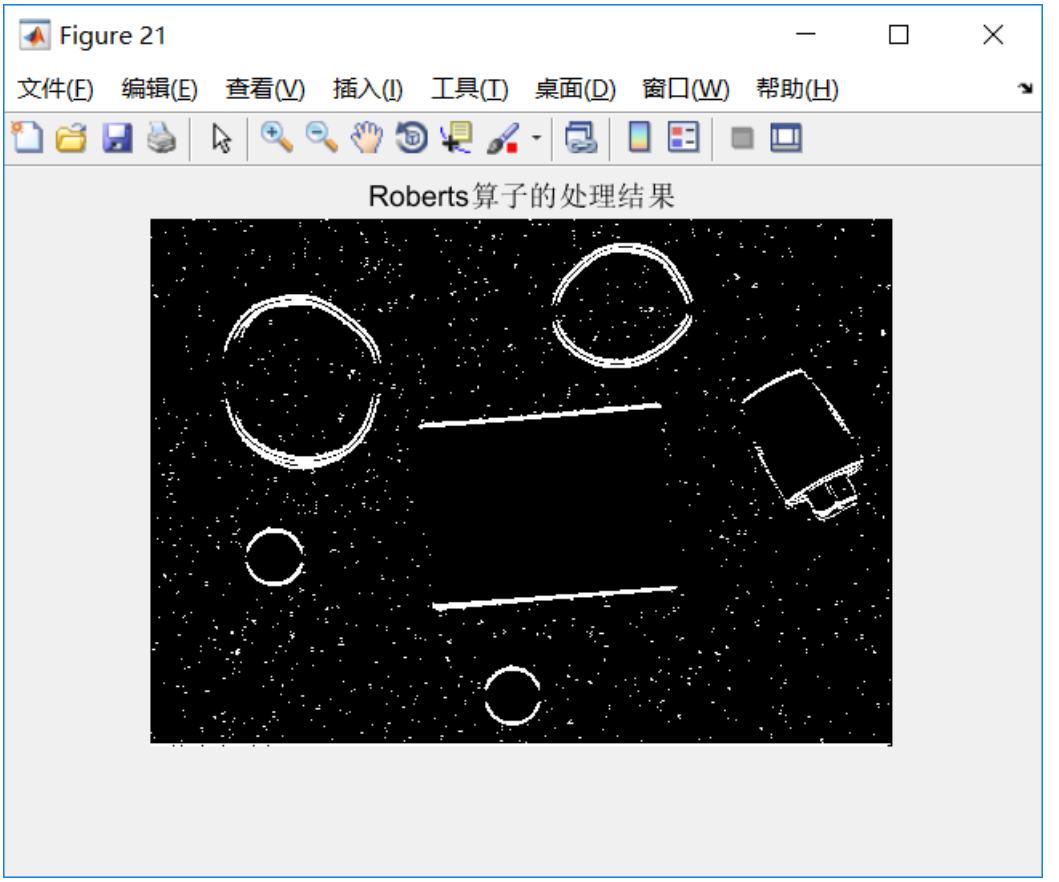


Figure 21
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

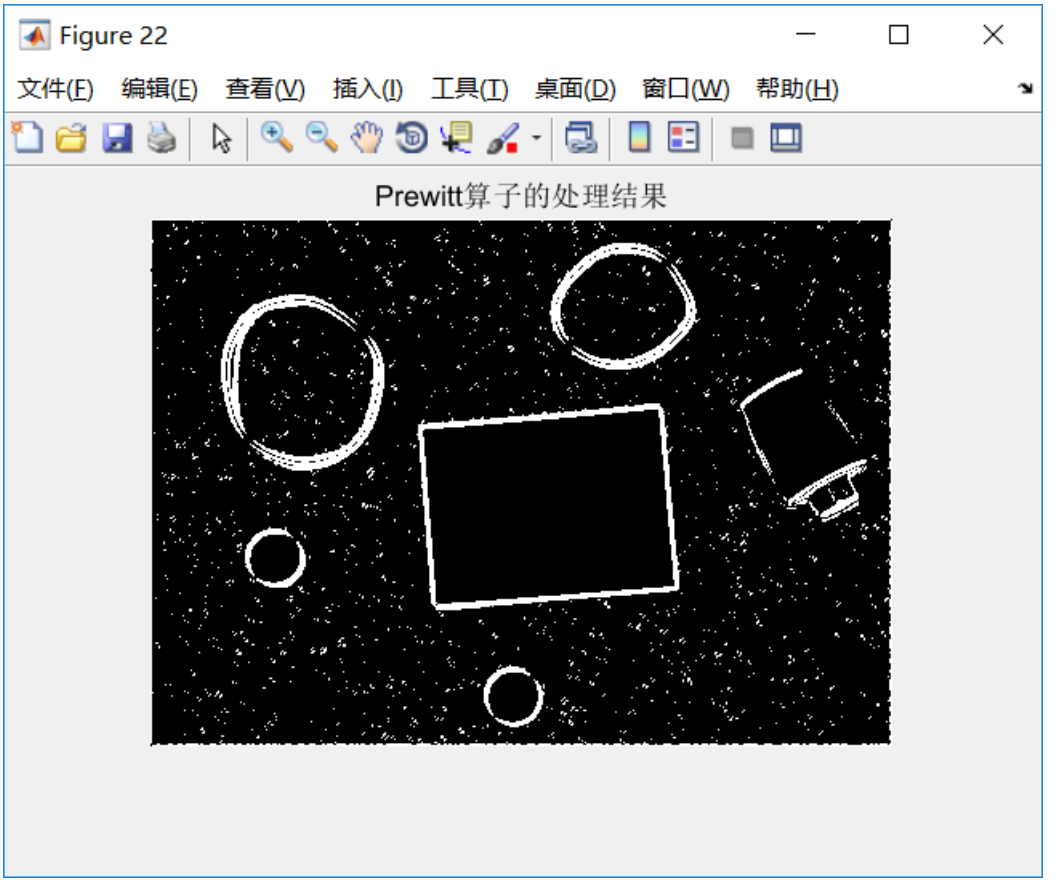


Figure 22
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

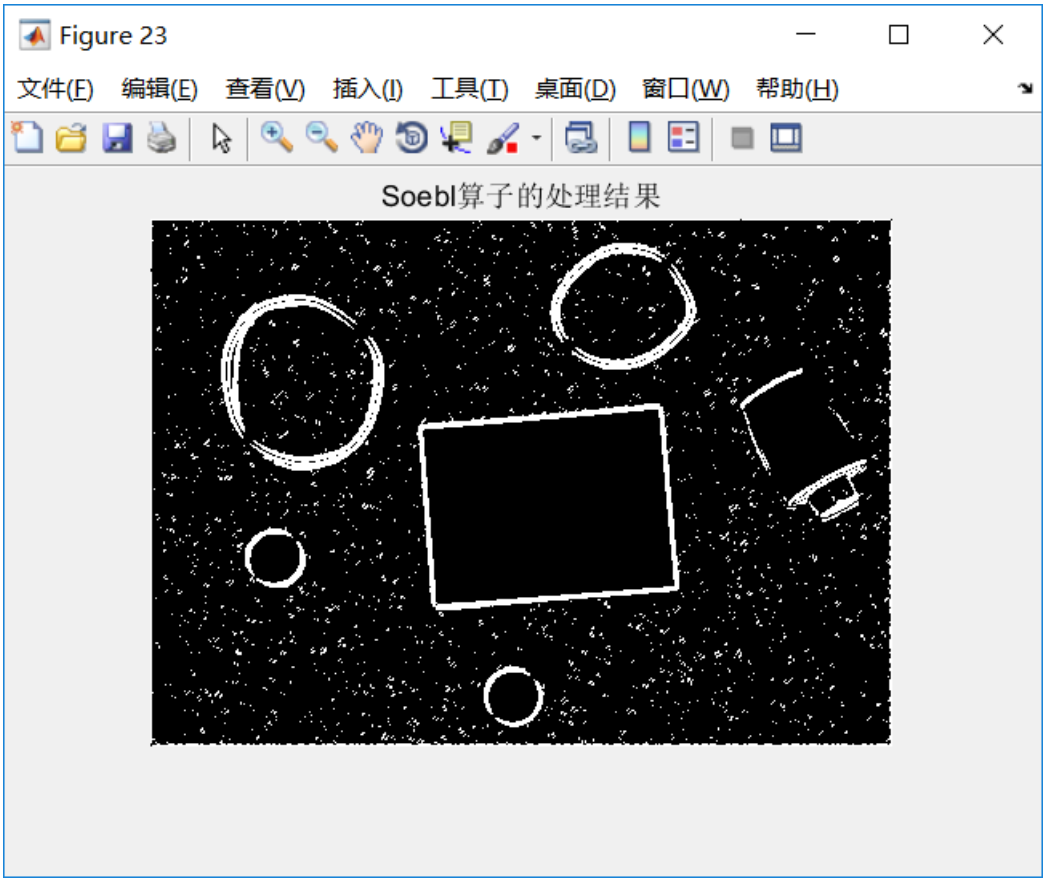


Figure 23
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

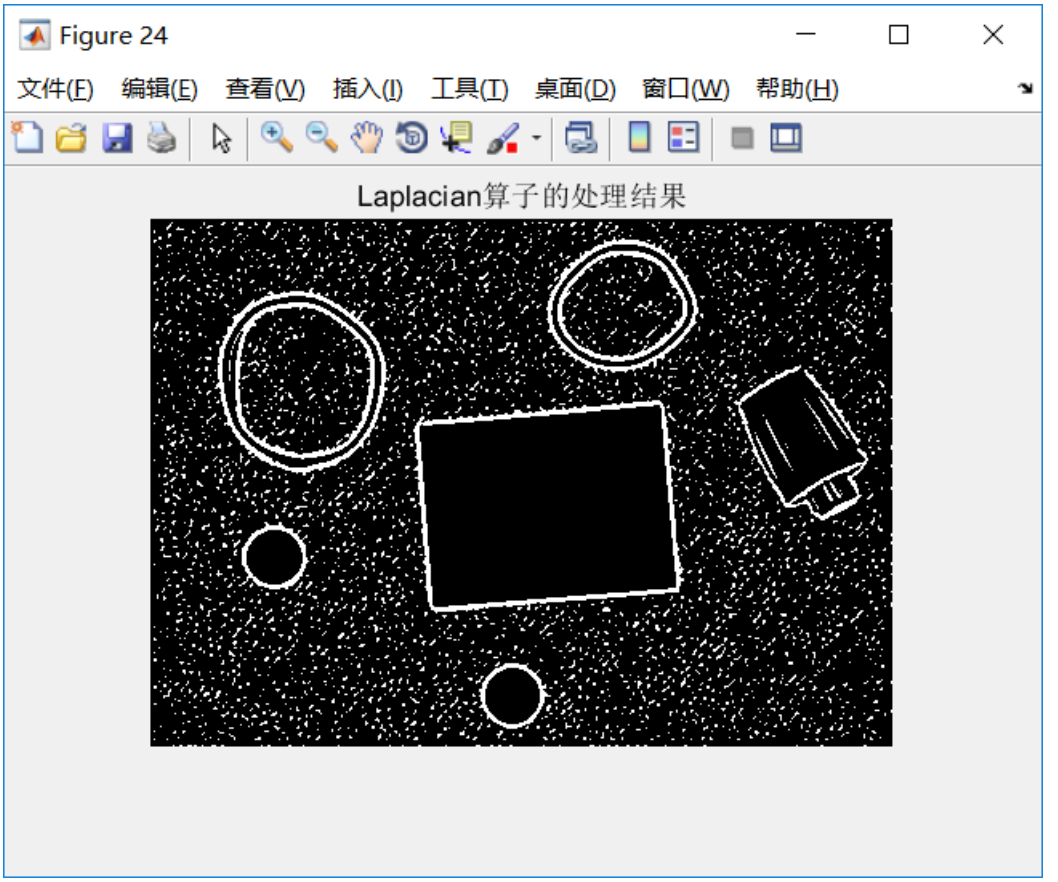
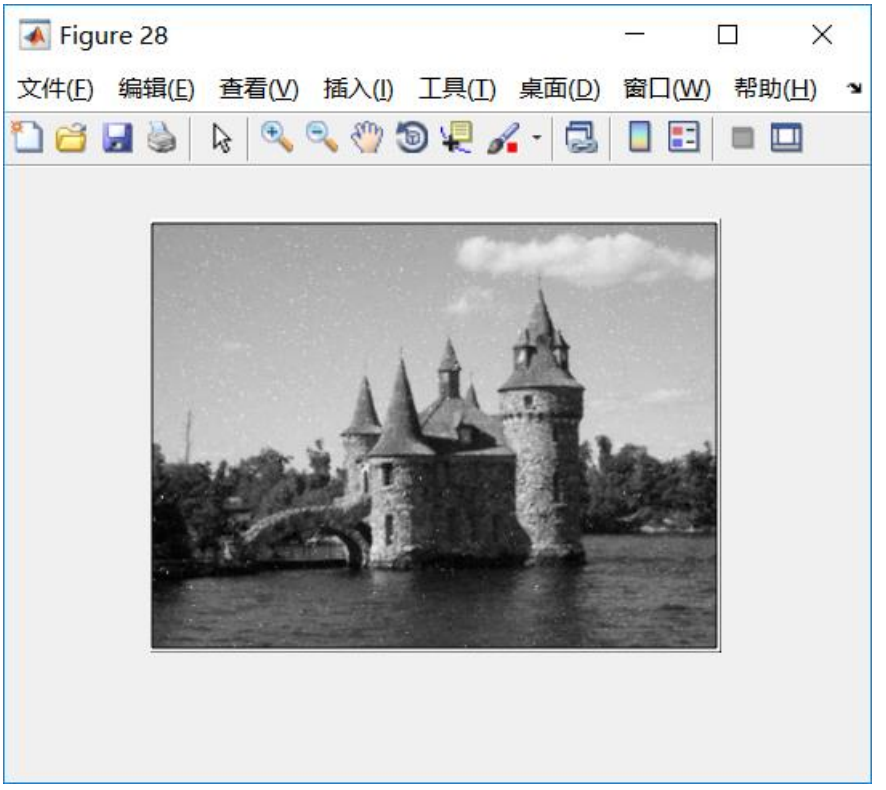
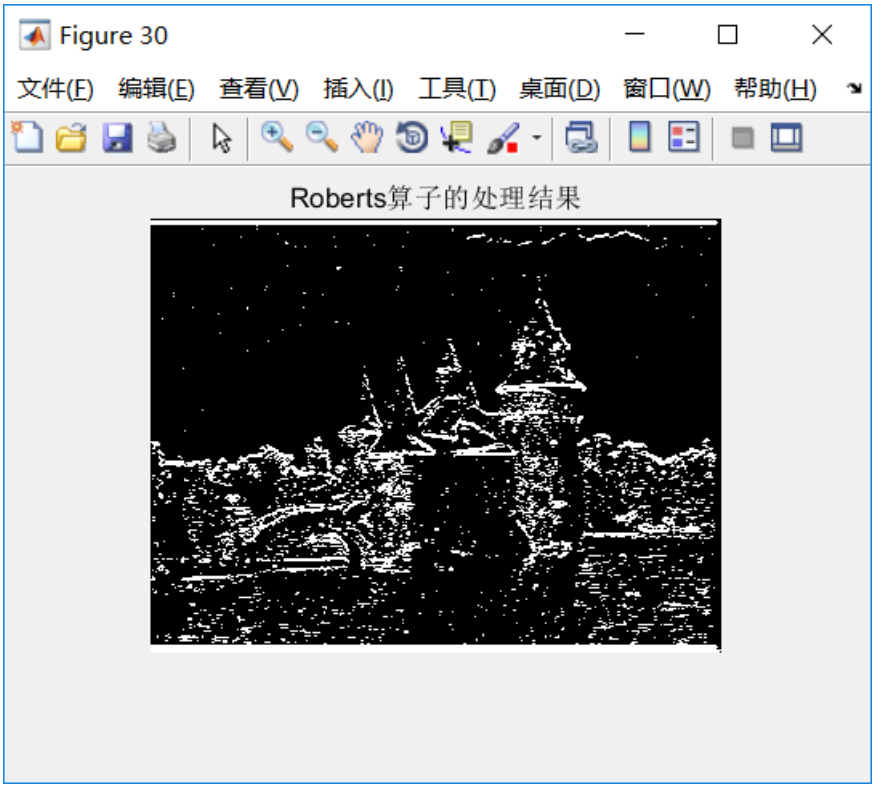


Figure 24
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

2.2.9.2 noise.png





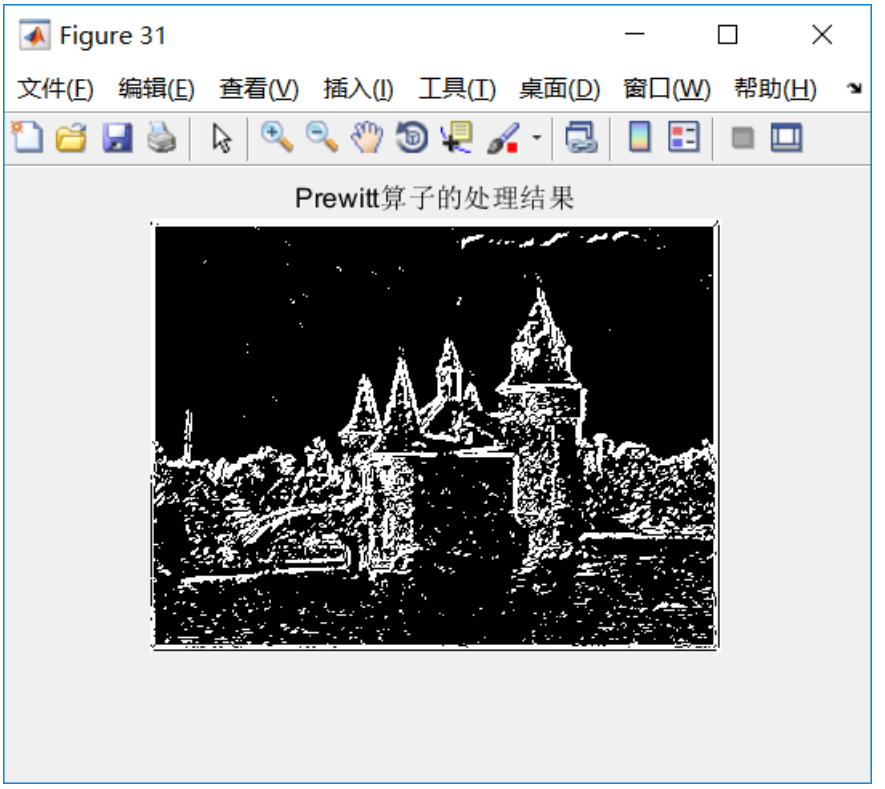


Figure 31
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

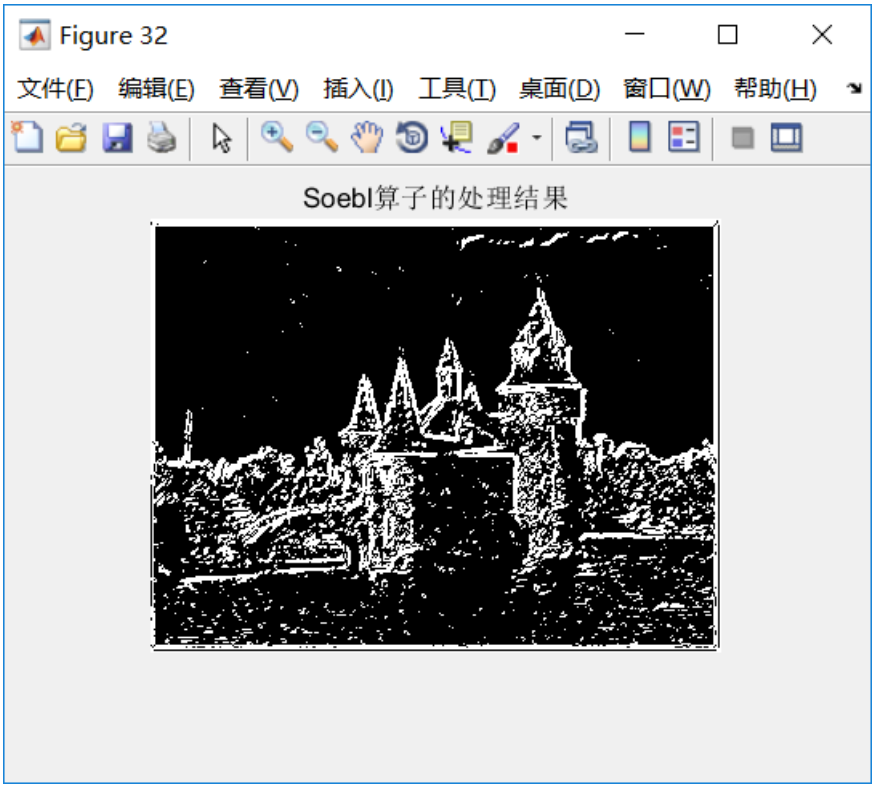


Figure 32
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果

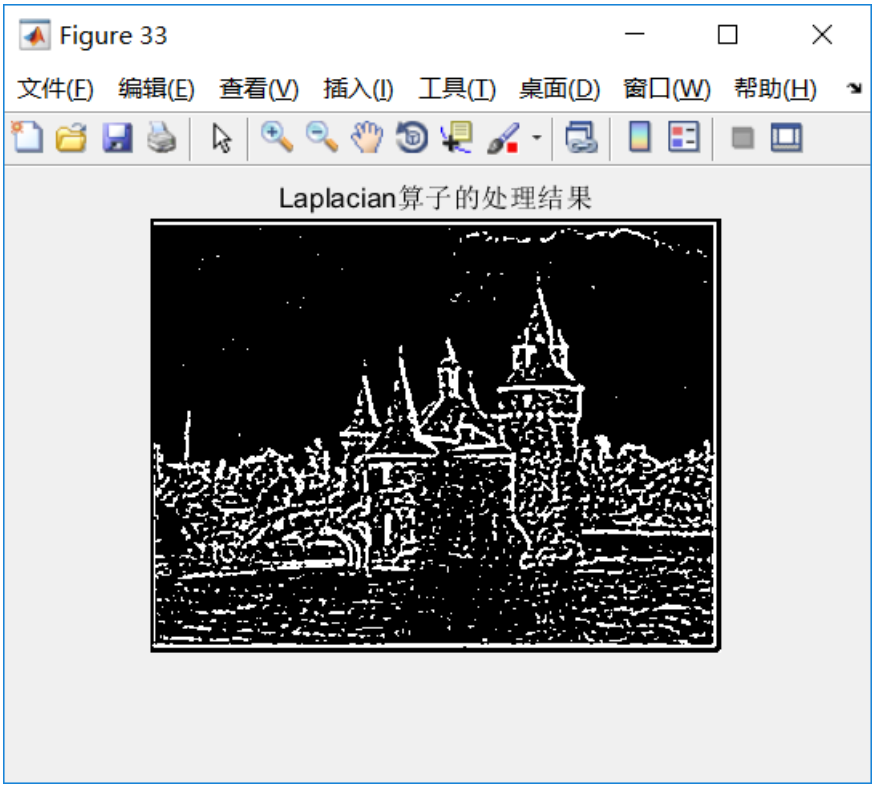
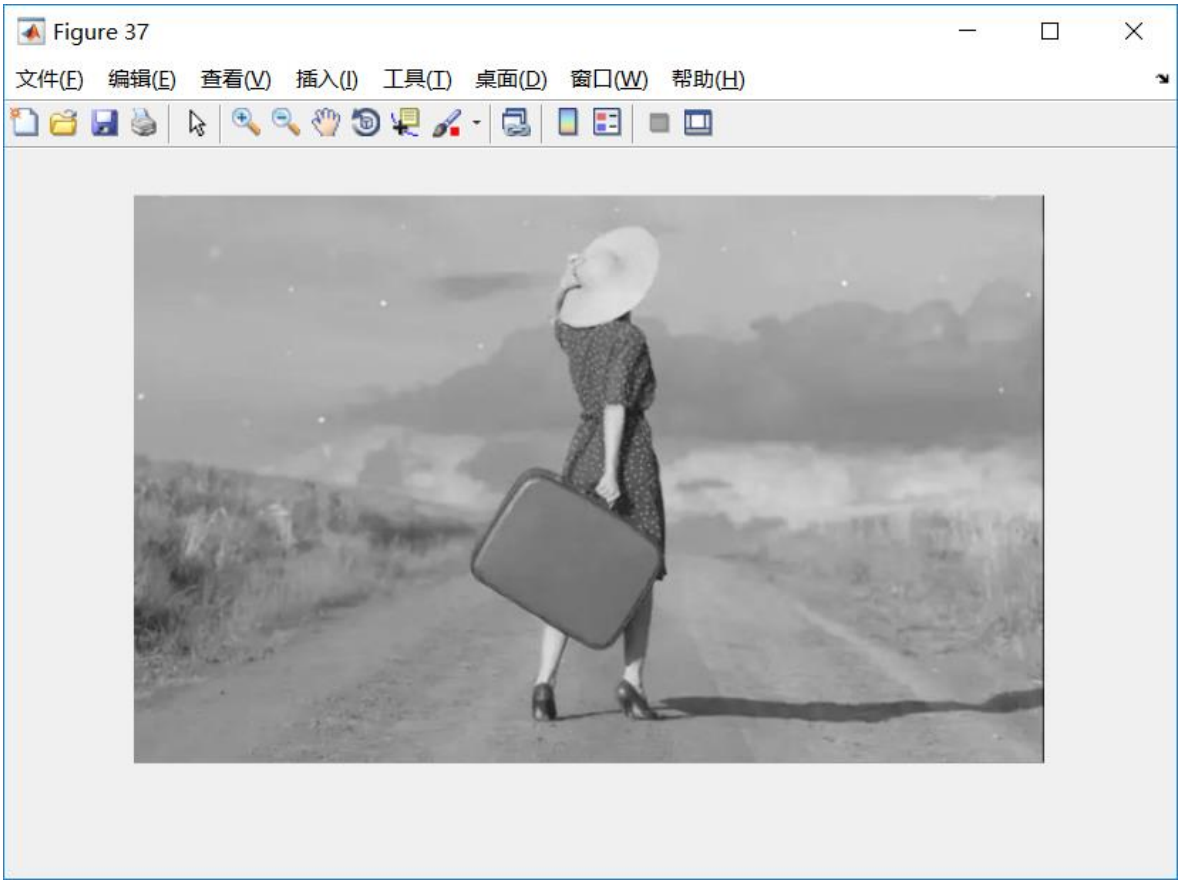


Figure 33
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Laplacian算子的处理结果

2.2.9.3 noise2.png



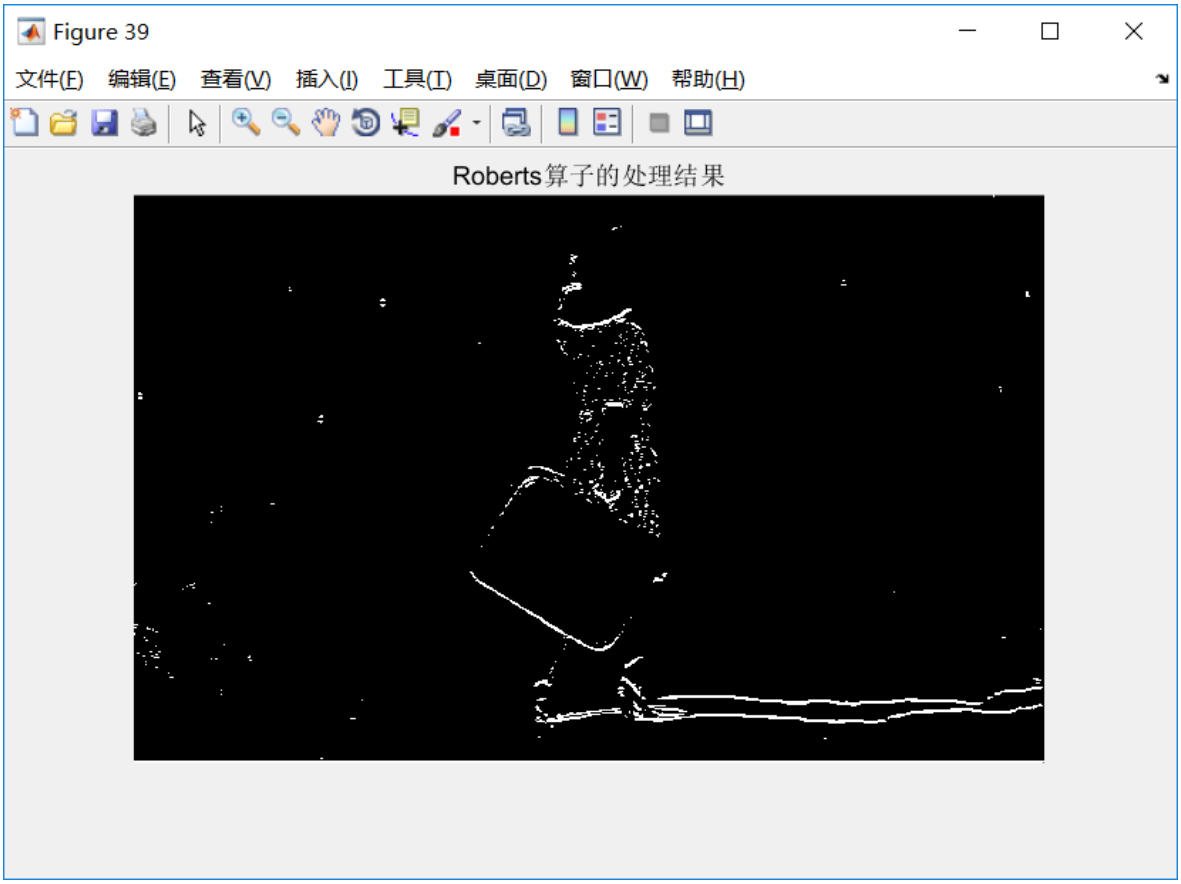


Figure 39
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Roberts算子的处理结果

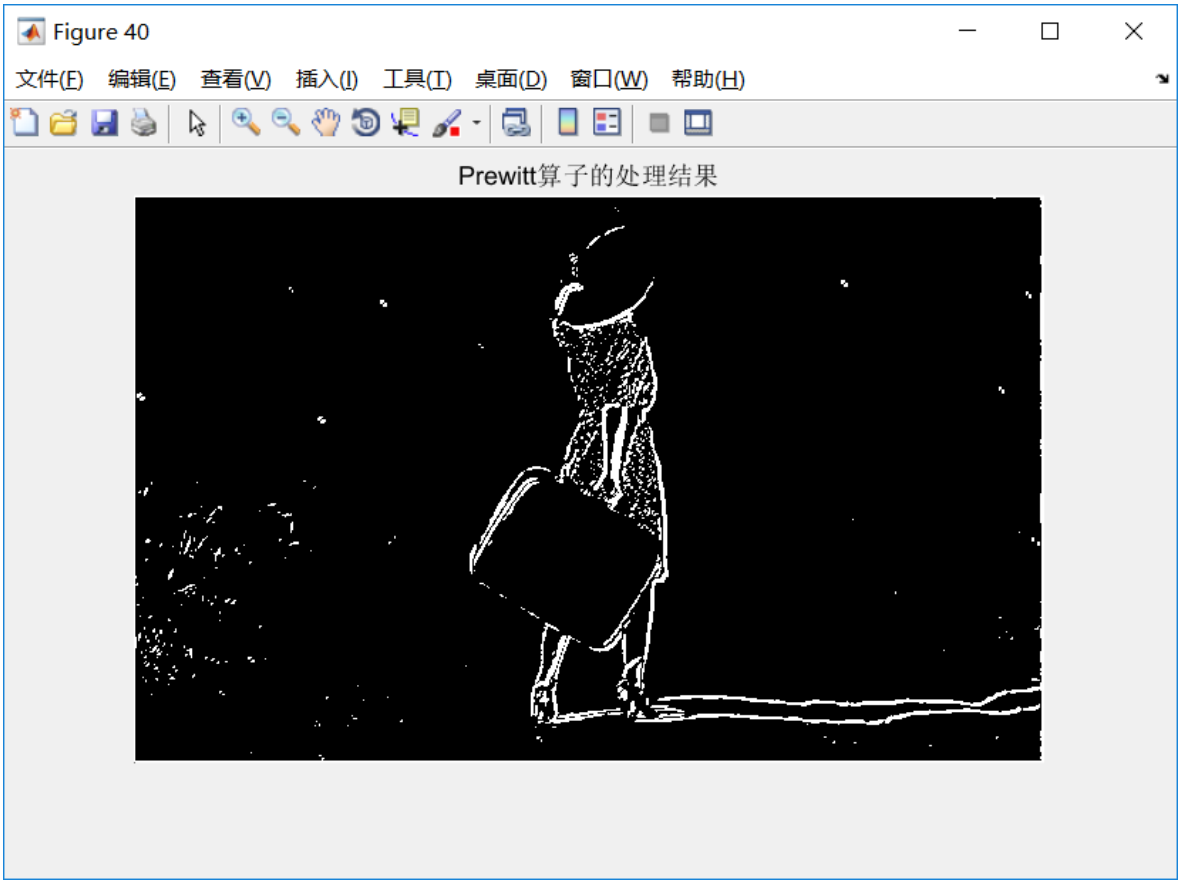


Figure 40
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Prewitt算子的处理结果

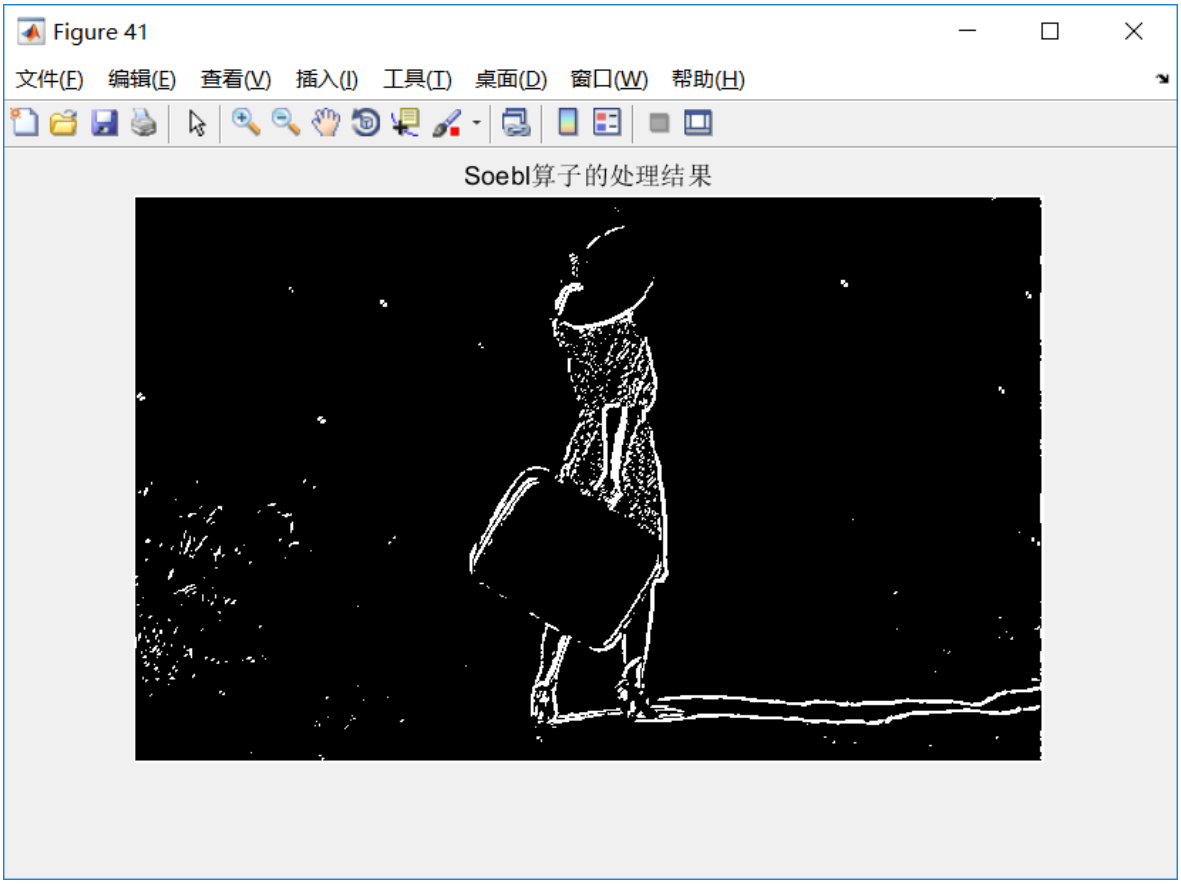
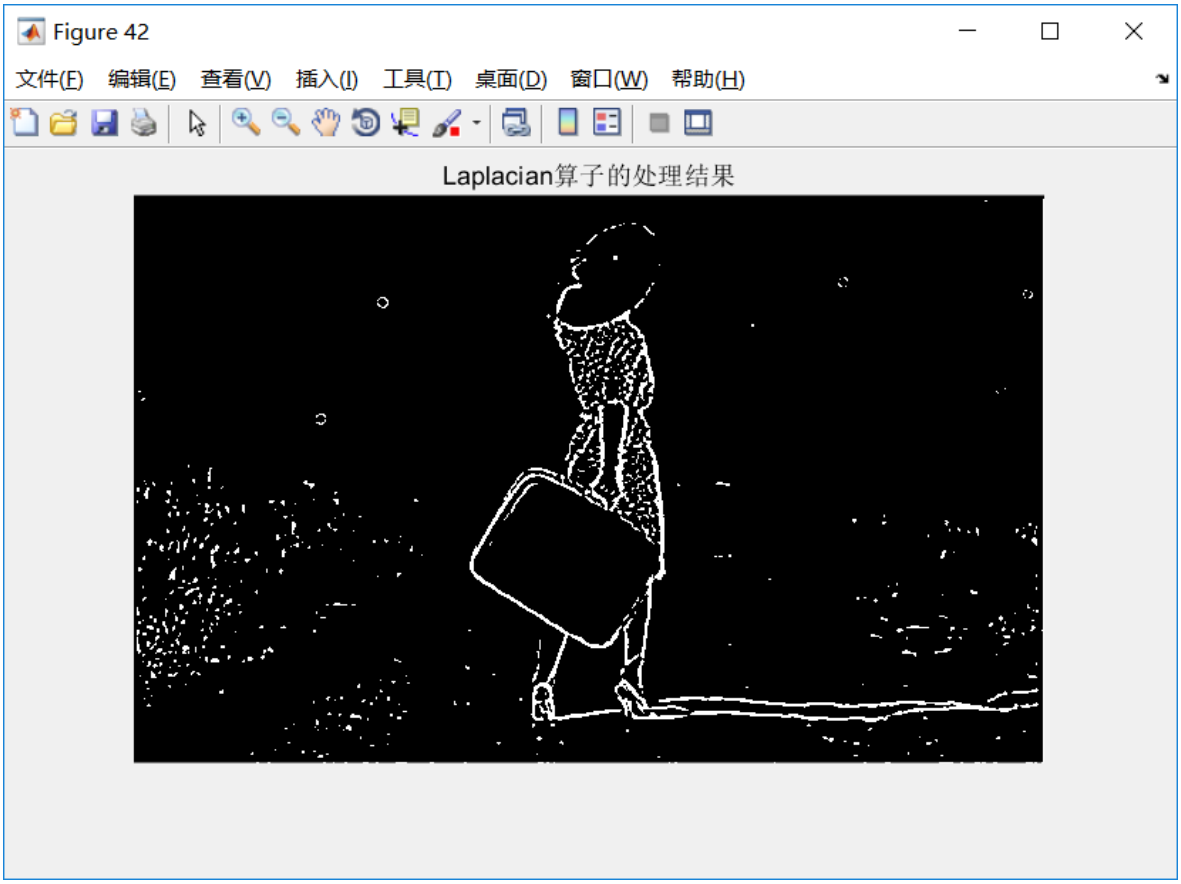


Figure 41
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Soebl算子的处理结果



3 结论

从上述实验结果可以看出，Roberts、Prewitt 和 Sobel 三种算子由于过于简单，因此结果也十分简陋，而 Laplacian 算子由于对噪声过于敏感，在部分测试图像中产生了噪点爆炸的问题，但同时，Laplacian 算子的边缘识别结果最为精细和完整，Canny 号称是应用最为广泛的算子，从结果中也可以得知，它对噪声有着较好的抑制作用，边缘识别结果虽然没有 Laplacian 算子那么完整但是在做到了抑制噪点的情况下，识别结果最接近 matlab 自带库函数。

而边缘链接算法，显然库函数与我的思想基本相同，都是使用了简单的启发式算法，寻找 8-邻域连通的连通分支。