第三次作业：边缘检测和边缘连接

姓名：丁保荣　　**学号**：171860509

2019 年 12 月 2 日

目录

1 边缘检测

对于边缘检测，我一共实现了四种算法：基于 sobel 算子的边缘检测器、基于 prewitt 算子的边缘检测器、Marr-Hildreth 边缘检测器、Canny 边缘检测器。

1.1 基于 sobel 算子的边缘检测器

1.1.1 实现原理

基于 sobel 算子的边缘检测器的实现在 *sobel.m* 文件中，主要的过程是

1. 先对原图像 G 进行高斯滤波得到 G' (主要是为了减小噪声的影响)
2. 对 G' 进行 X 方向的 sobel 滤波，得到 G'_X
3. 对 G' 进行 Y 方向的 sobel 滤波，得到 G'_Y
4. 将 X 方向和 Y 方向的滤波结果进行叠加。得到 new_G
5. 根据阈值，对 new_G 进行二值化，得到输出图像

其中第 1 步中的高斯滤波可以根据图像有无噪声进行选择，如果图像没有噪声，可以选择不进行高斯滤波。

1.1.2 实现效果

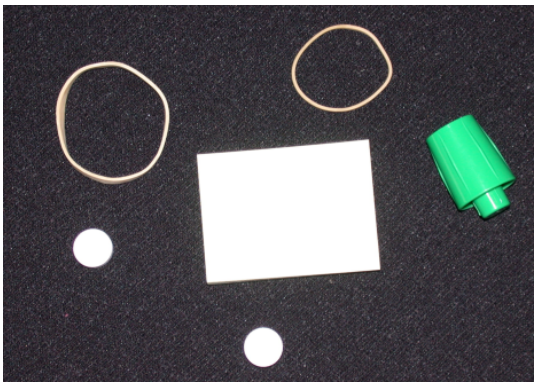

图 1: 原图

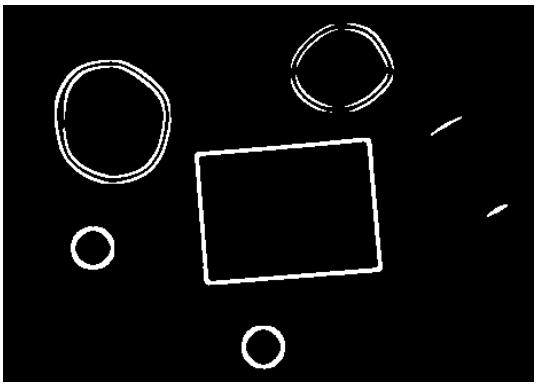

图 2: sobel 滤波后的

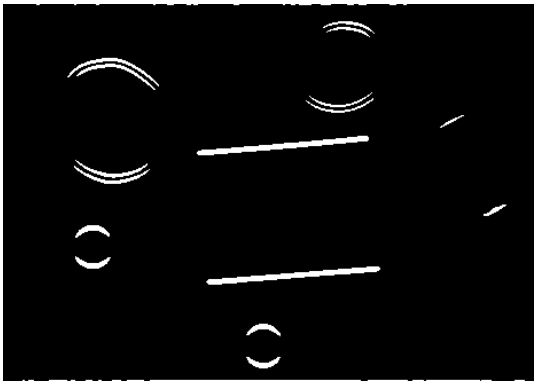

图 3: 只进行 X 方向滤波

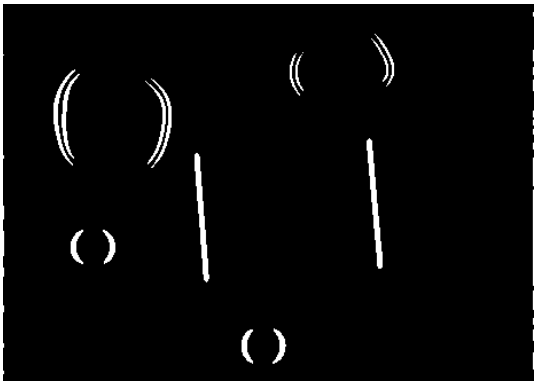

图 4: 只进行 Y 方向滤波

如果只进行 X 方向或 Y 方向的滤波，会发现只有一个方向的边缘比较明显，另一个方向的边缘并不是很明显。

下面是其他一些图像经过 sobel 滤波后的结果：

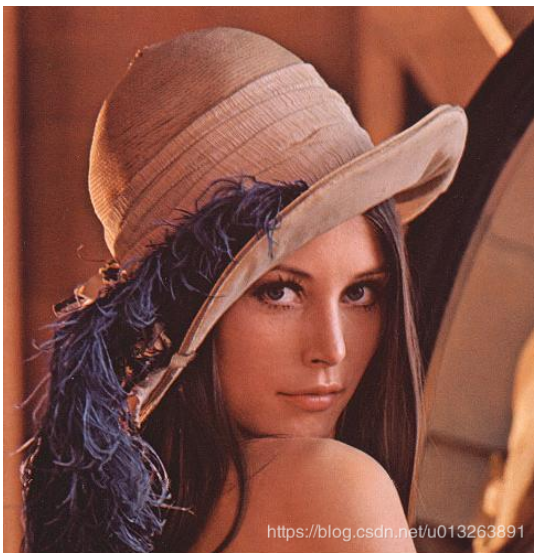

图 5: 原图

图 6: sobel 滤波后的

图 7: 原图

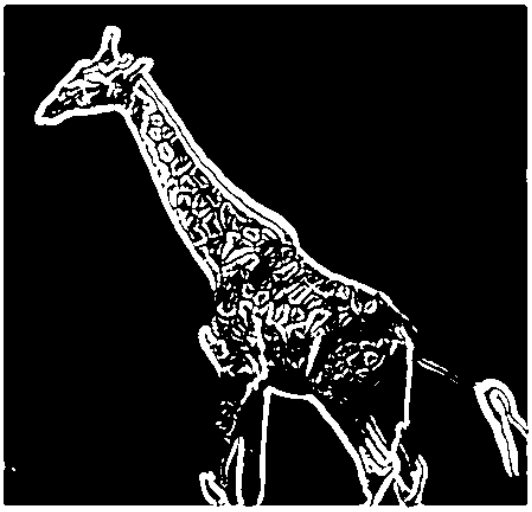

图 8: sobel 滤波后的

图 9: 原图

图 10: sobel 滤波后的

图 11: 原图

图 12: sobel 滤波后的

图 13: 原图

图 14: sobel 滤波后的

1.2 基于 prewitt 算子的边缘检测器

1.2.1 实现原理

基于 prewitt 算子的边缘检测器的实现在 *prewitt.m* 文件中，实现方法与 sobel 算子类似，主要的过程是

1. 先对原图像 G 进行高斯滤波得到 G' (主要是为了减小噪声的影响)
2. 对 G' 进行 X 方向的 prewitt 滤波，得到 G'_X
3. 对 G' 进行 Y 方向的 prewitt 滤波，得到 G'_Y
4. 将 X 方向和 Y 方向的滤波结果进行叠加。得到 new_G
5. 根据阈值，对 new_G 进行二值化，得到输出图像

其中第 1 步中的高斯滤波可以根据图像有无噪声进行选择，如果图像没有噪声，可以选择不进行高斯滤波。

1.2.2 实现效果

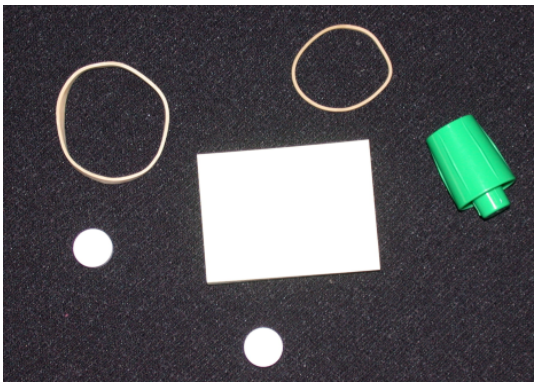

图 15: 原图

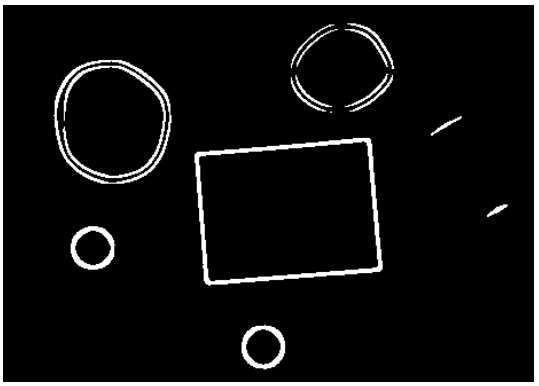

图 16: prewitt 滤波后的

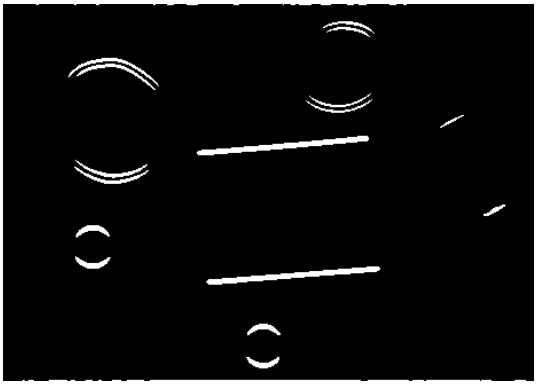

图 17: 只进行 X 方向滤波

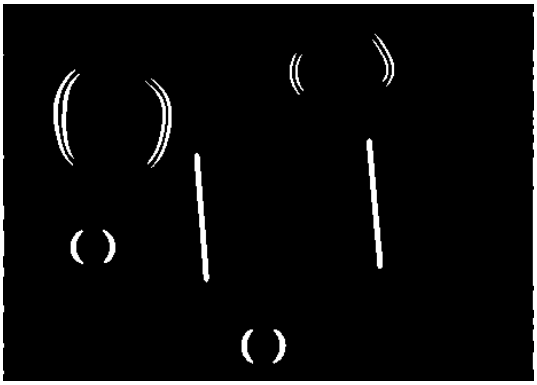

图 18: 只进行 Y 方向滤波

与 sobel 算子类似，如果只进行 X 方向或 Y 方向的滤波，会发现只有一个方向的边缘比较明显，另一个方向的边缘并不是很明显。

下面是其他一些图像经过 prewitt 滤波后的结果：

图 19: 原图

图 20: prewitt 滤波后的

图 21: 原图

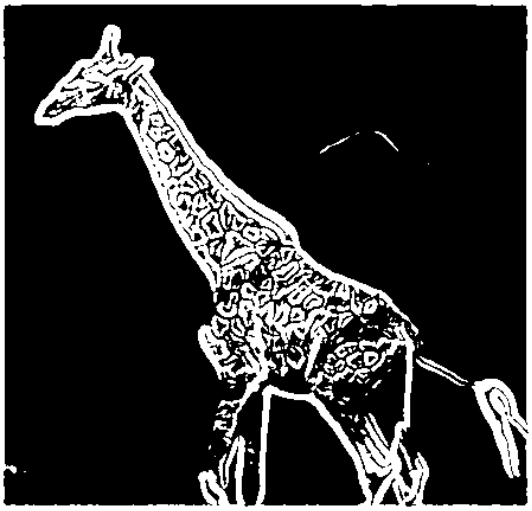

图 22: prewitt 滤波后的

图 23: 原图

图 24: prewitt 滤波后的

图 25: 原图

图 26: prewitt 滤波后的

图 27: 原图

图 28: prewitt 滤波后的

1.3 prewitt 算子和 sobel 算子的比较

X 方向的 prewitt 算子:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

X 方向的 sobel 算子:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

在矩阵表示上，我们发现 sobel 算子和 prewitt 算子的差异很小，所以我们看他们效果也差不多。因为没有非最大抑制，所以 sobel 算子和 prewitt 算子产生的图像边缘都比较粗

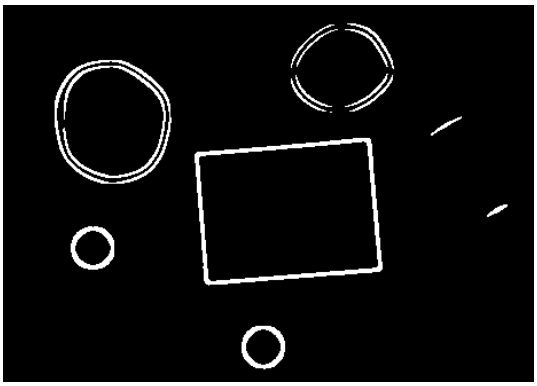

图 29: sobel 滤波后的

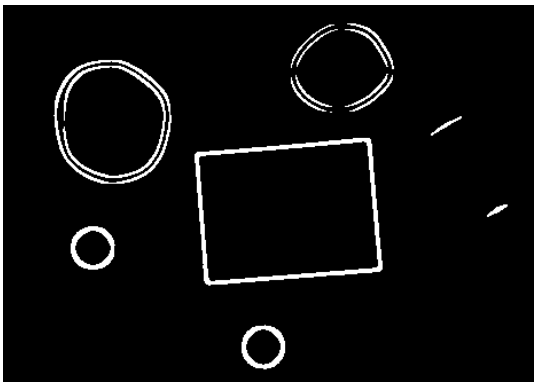

图 30: prewitt 滤波后的

图 31: sobel 滤波后的

图 32: prewitt 滤波后的

1.4 Marr-Hildreth 边缘检测器

1.4.1 实现原理

Marr-Hildreth 边缘检测器的实现在 $MarrHildreth.m$ 文件中，实现方法与 sobel 算子类似，主要的过程是

1. 用 $n \times n$ 的高斯低通滤波器平滑图像 G 得到 G'(n 是大于等于 6σ 的最小奇数)
2. 计算 G' 的拉普拉斯 G'_L
3. 寻找 G'_L 的零交叉 (任意方向的两个邻居符号相反且差值大于一定的阈值)

Marr-Hildreth 主要需要设置的参数有 n、阈值系数 (th)，都和输入图像有一定的关系。其中我对不同图像设置不同的阈值系数 (th) 以达到最好的效果。如果阈值比例设置得太低，会有很多噪点，但阈值比例设置得太高，又可能会使得边缘断开。

1.4.2 实现效果

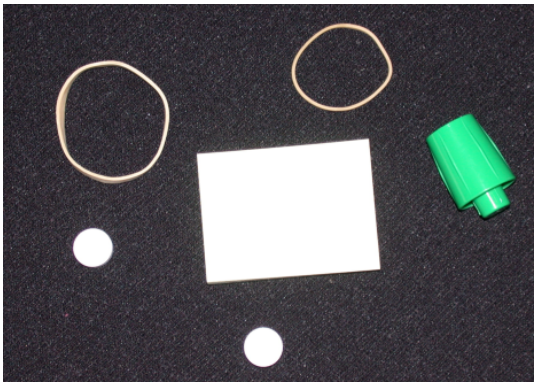

图 33: 原图

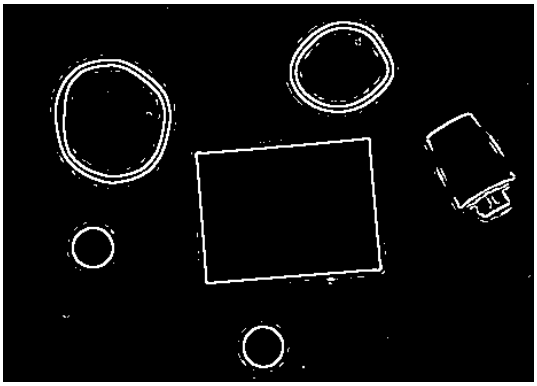

图 34: MH 处理后的 ($th = 0.35$)

我发现设置不同阈值系数对于图像的效果影响还是很大的：如下面两张图的阈值比例就设置得不太好。

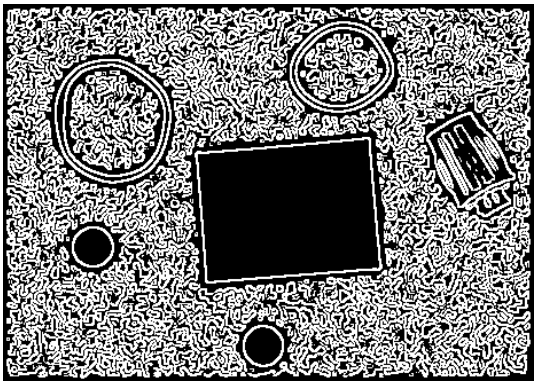

图 35: MH 处理后的 ($th = 0.04$)

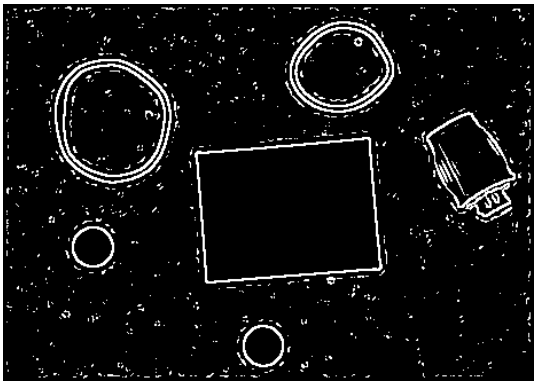

图 36: MH 处理后的 ($th = 0.2$)

下面展示一下其他图片用 Marr-Hildreth 边缘检测器的效果:

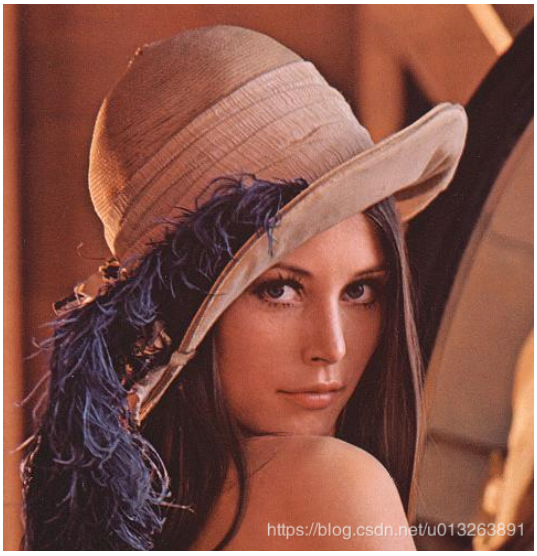

图 37: 原图

图 38: MH 处理后的 ($th = 0.3$)

图 39: 原图

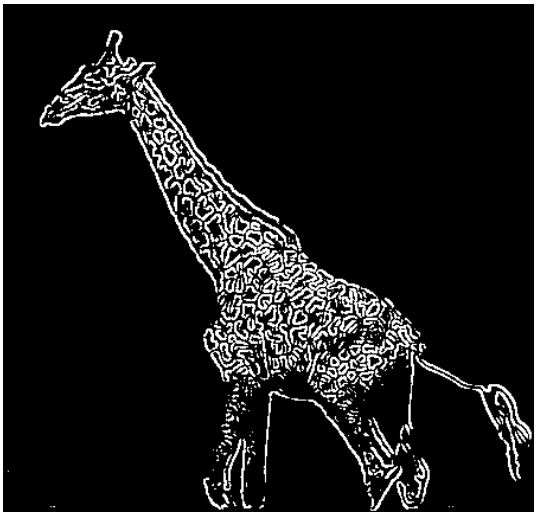

图 40: MH 处理后的 ($th = 0.35$)

图 41: 原图

图 42: MH 处理后的 ($th = 0.2$)

图 43: 原图

图 44: MH 处理后的 ($th = 0.15$)

图 45: 原图

图 46: MH 处理后的 ($th = 0.055$)

1.5 Canny 边缘检测器

1.5.1 实现原理

Canny 边缘检测器的实现在 *Canny.m* 文件中，实现方法与 sobel 算子类似，主要的过程是

1. 用 $n \times n$ 的高斯低通滤波器平滑图像 G 得到 G'(n 是大于等于 6σ 的最小奇数)

2. 计算 G' 的梯度大小 $grad$ 和方向 $alpha$，
3. 非最大化抑制：我是对四个方向做非最大化抑制，如果该方向的邻居有比当前点大的抑制当前点
4. 滞后阈值和联通性分析：对于每一个大于 TH 的点，设置为 1，再对它的八连通点进行分析，如果大于 TL，则也设置为 1.

Canny 边缘检测器需要设置的主要有两个参数：高阈值 (TH) 和低阈值 (TL).

1.5.2 实现效果

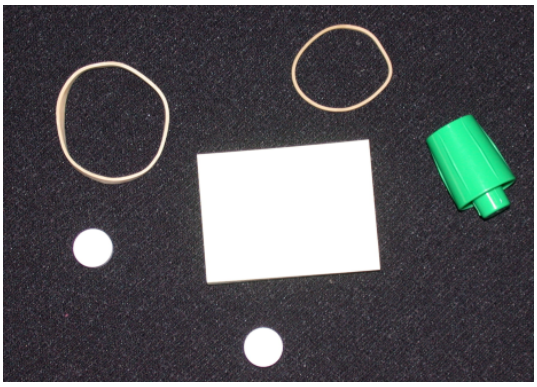

图 47: 原图

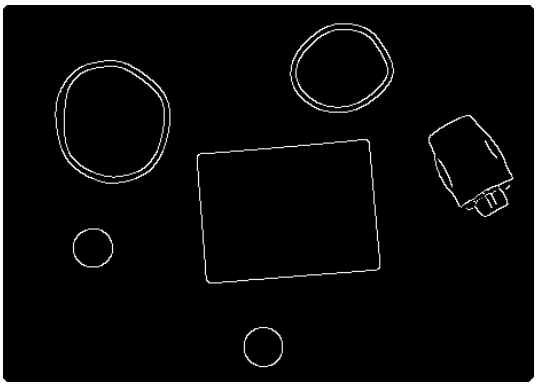

图 48: Canny 处理后的 ($TH = 0.02, TL = 0.01$)

图 49: 原图

图 50: Canny 处理后的 ($TH = 0.025, TL = 0.01$)

图 51: 原图

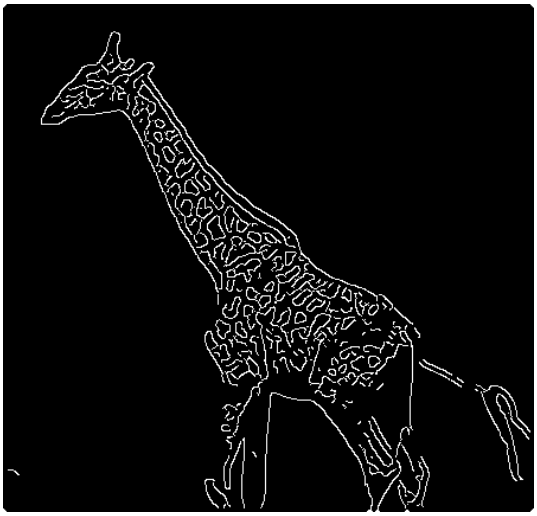

图 52: Canny 处理后的 ($TH = 0.035, TL = 0.015$)

图 53: 原图

图 54: Canny 处理后的 ($TH = 0.015, TL = 0.006$)

图 55: 原图

图 56: Canny 处理后的 ($TH = 0.02, TL = 0.01$)

图 57: 原图

图 58: Canny 处理后的 ($TH = 0.015, TL = 0.005$)

1.6 Marr-Hildreth 和 Canny 边缘检测器总结

在 MH 中，阈值比例 th 的选取对结果的影响是比较大的。同样的，在 Canny 中，低阈值 TL 和高阈值 TH 的选取对结果的影响也是比较大的。

因为有非最大抑制，所以 Canny 的边缘较细。而 MH 的边缘较粗，但比 sobel 算子和 prewitt 算子的细。

Canny 在边缘不间断上做的要比 MH 好。如下面两张图的对比：

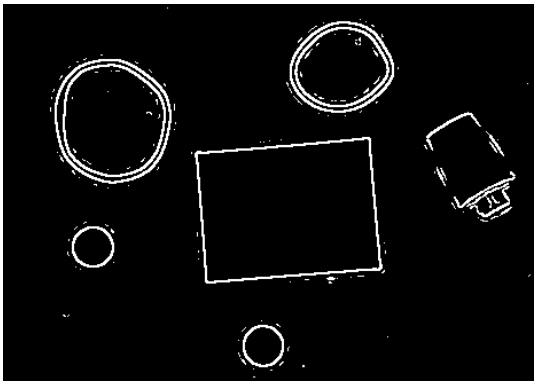

图 59: MH 处理后的 ($th = 0.35$)

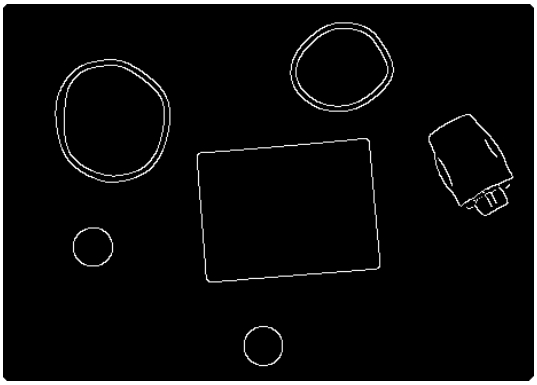

图 60: Canny 处理后的 ($TH = 0.02, TL = 0.01$)

2 边缘连接

对于边缘连接，我使用 Moore-Neighbor Tracing 算法。用 imtool 来定界，确定一个点后，使用该算法就能得出一个轮廓。

2.1 Moore-Neighbor Tracing

2.1.1 实现原理

Moore-Neighbor Tracing 算法的实现主要在 *my_edgelinking.m* 中，主要的算法流程是：

1. 设输入图像为 G, 输入的起始点为 $s(row, col)$
2. 设 c 为 $N(s)$ 顺时针方向中的一个点。($N(s)$ 表示 s 的邻域集合)
3. 将 s 加入到 B 中
4. 当不满足终止条件时，反复执行下列步骤，否则返回 B
5. 如果 $G(c) = 1$, 那么
   - 令 $p = c$
   - 将 c 加入 B 中
   - 更新方向向量
6. 方向向量递增，选择 $N(p)$ 中相应方向的点。

   终止条件可以设置为到达 s 多次，或者以相同的方向进入 s.

2.1.2 实现效果

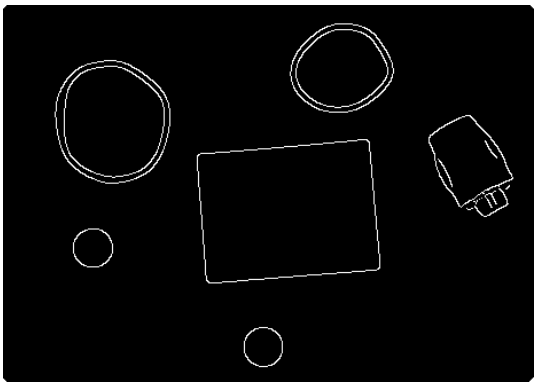

图 61: Canny 处理后

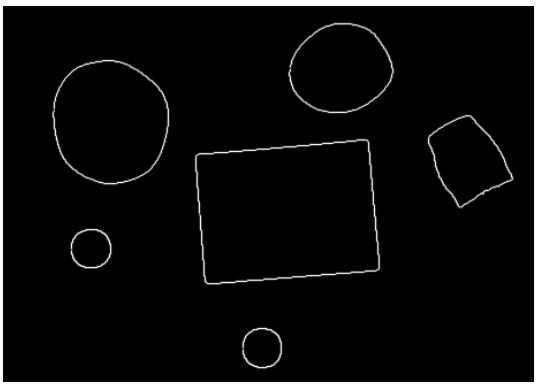

图 62: 进一步边缘连接后