工件提取大作业报告

1. 基本功能实现

本次大作业实现了图像背景噪声去除、工件分离。

在实现图像背景噪声去除功能时，我采用了两种不同的方法：一种是频域滤波去噪；另一种是利用空域中图像特点去噪。

在实现工件分离时，我采用了 canny 算子求图像边缘，再由区域生长与边缘结合分离出三个工件。

2. 设计思路

2.1 背景噪声去除

2.1.1 频域方法

观察图像可以发现，背景噪声主要来自构成网格的横线和竖线。由二维傅里叶变换规律可以知道这些横竖线对应的频域部分分别是竖、横部分。

我选择的方法需要经历一下几个过程：

1. DFT 变换

变换结果如下：

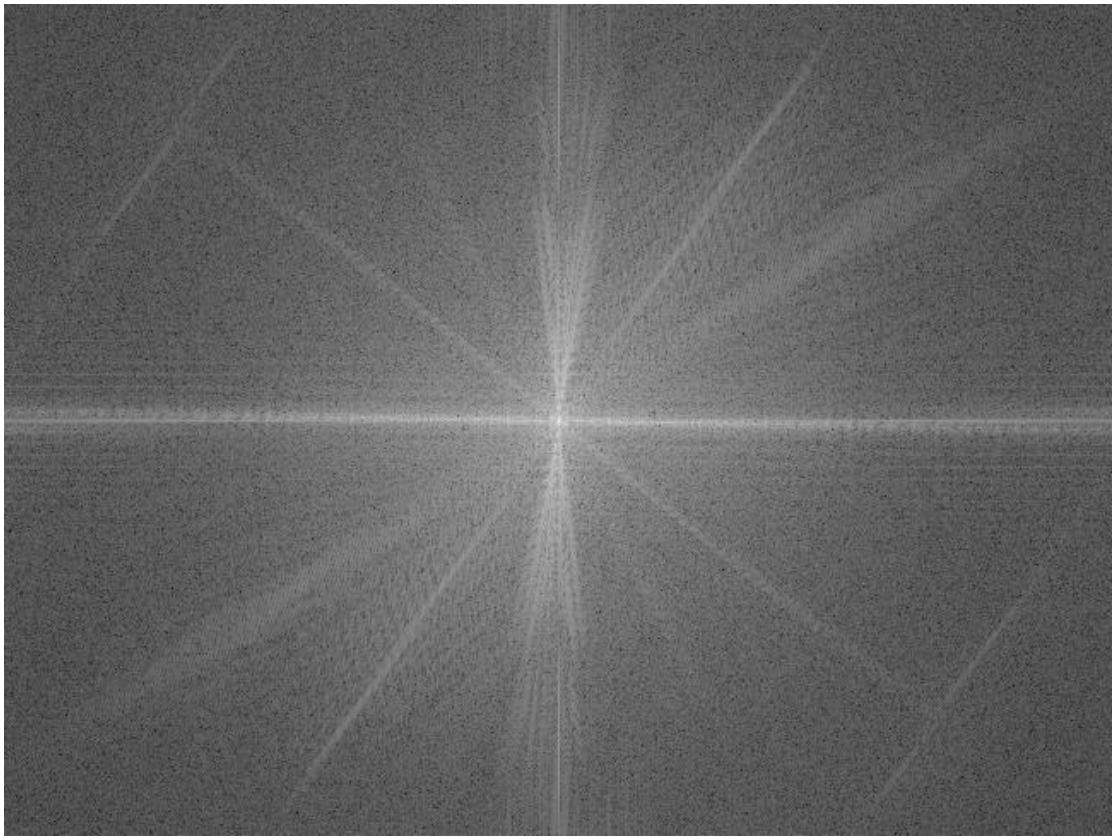


图 1　原图像经过傅里叶变换后的结果（居中后）

其中，傅里叶变换图中水平和竖直部分对应了空域中的网格线。

2.　自定义滤波器保留频域中噪声部分

先使用高通滤波器去除低频部分，然后使用特定滤波器得到水平和竖直部分。

频域中的低频部分在空域中对应元件，所以去除低频相当于在空域中先去掉元件，这样做的目的是尽量保护元件不受频域滤波的影响。

高通滤波后结果如下：

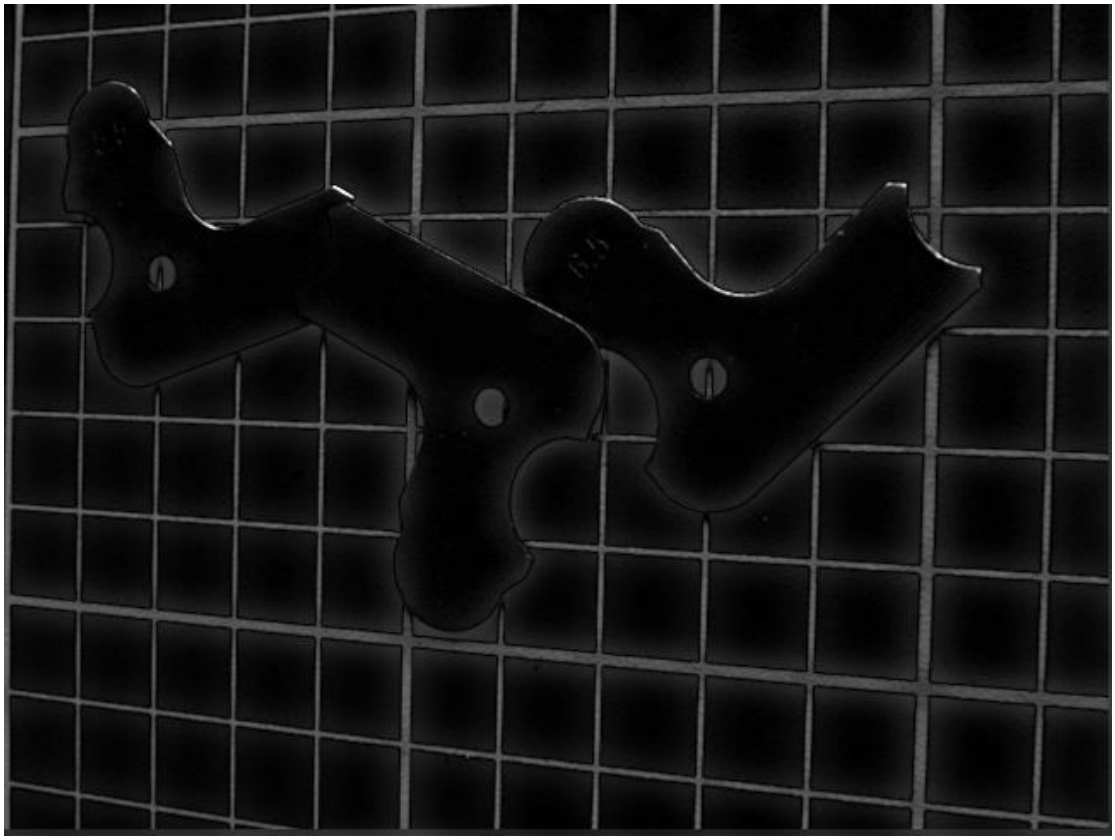


图 2　高通滤波后空域

使用特定形状滤波器，形状类似花瓣，以此来得到水平和竖直的频域部分，其在空域对应为网格线。

特定形状滤波后如下:

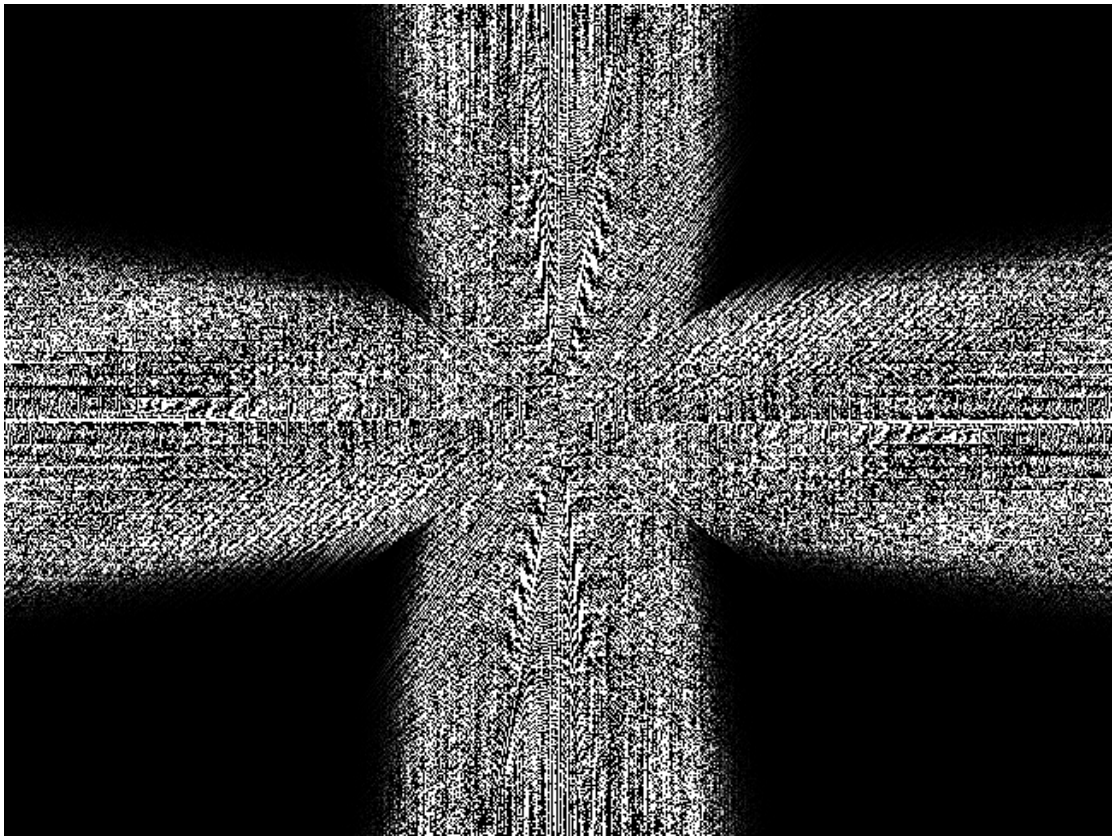


图 3　特定形状滤波器滤波后（只显示了实部）

高通滤波和特定形状滤波之后的空域结果:

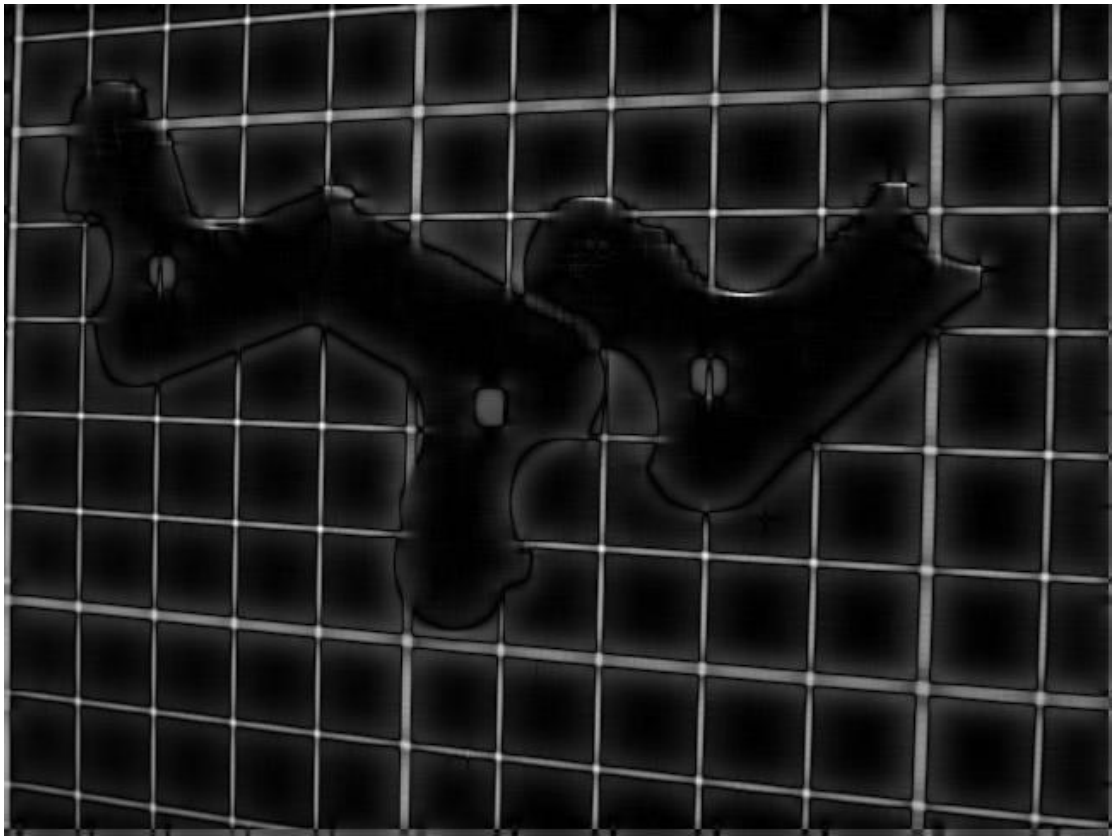


图 4　滤波之后得到网格线

3. 空域中减去网格线

在空域中用原图减去网格线，剩下部分主要是元件，但是元件也有部分丢失。

结果如下:

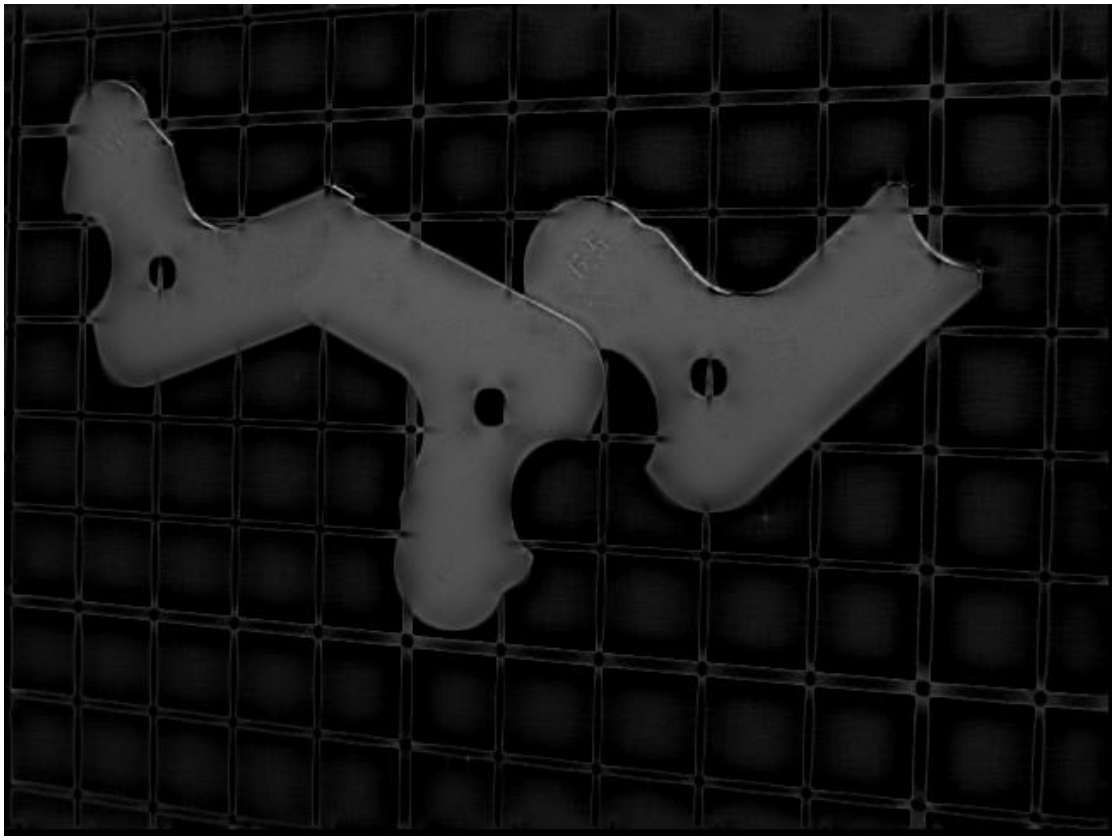


图 5　减去网格线之后

4.　二值化

二值化的目的是彻底将网格线和元件分割开来。

结果如下:

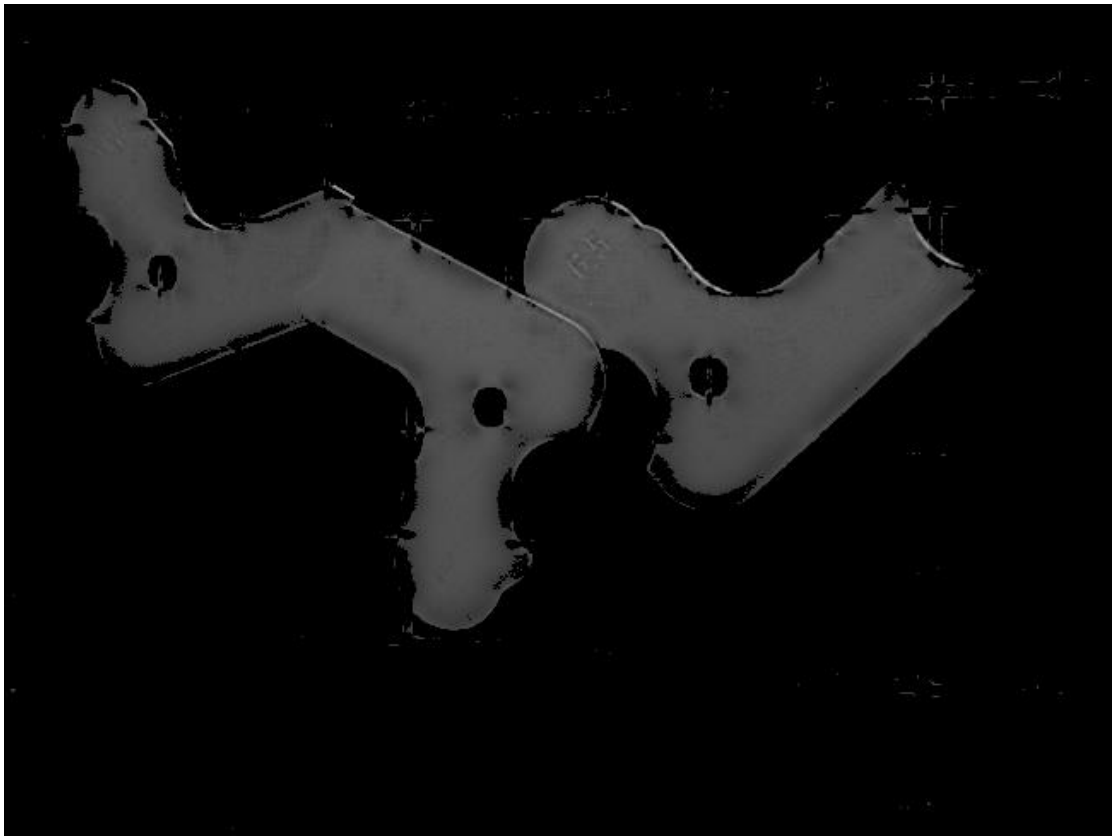


图 6　二值化后结果

可以看到，二值化之后大部分的网格已经被去掉，但是有部分的工件也被去掉了。

5.　还原像素值

将二值化后像素值不为 0 的像素变为原图对应位置的像素值。频域中的滤波对工件还是有一定影响，改变了其对应位置处的像素值，所以这里需要将像素值还原。

还原像素值之后：

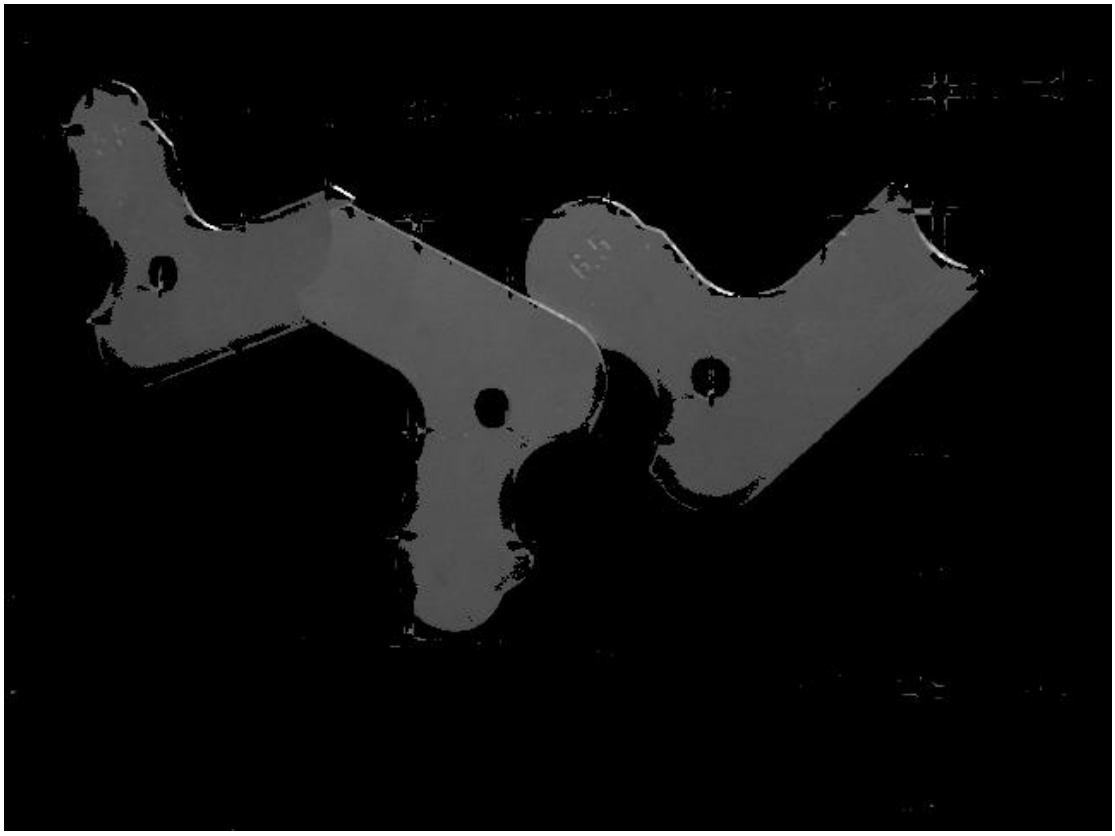


图 7　还原像素值后

6.　填补丢失元件

一系列的处理后，元件出现部分丢失，所以需要将丢失的部分补齐。

将原图通过开操作后得到大致元件，用得到的元件补齐频域处理后丢失的部分。

开操作后：

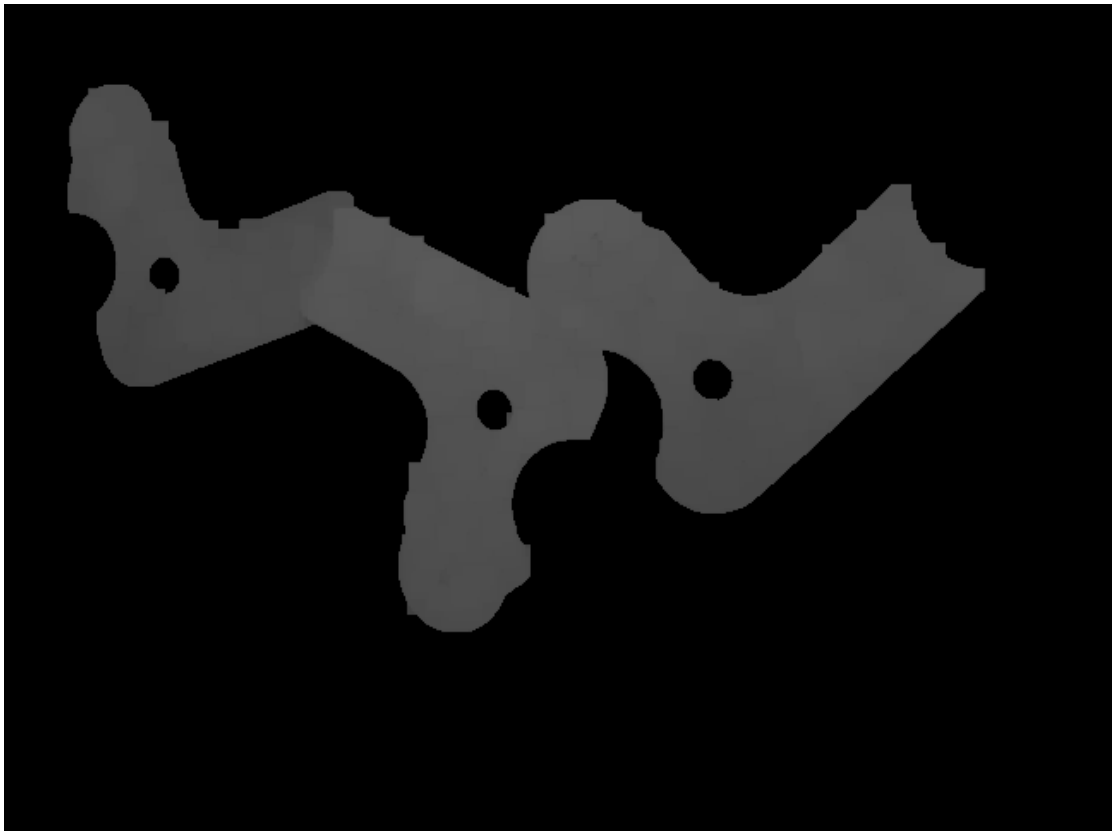


图 8　原图开操作后

填补丢失元件部分:

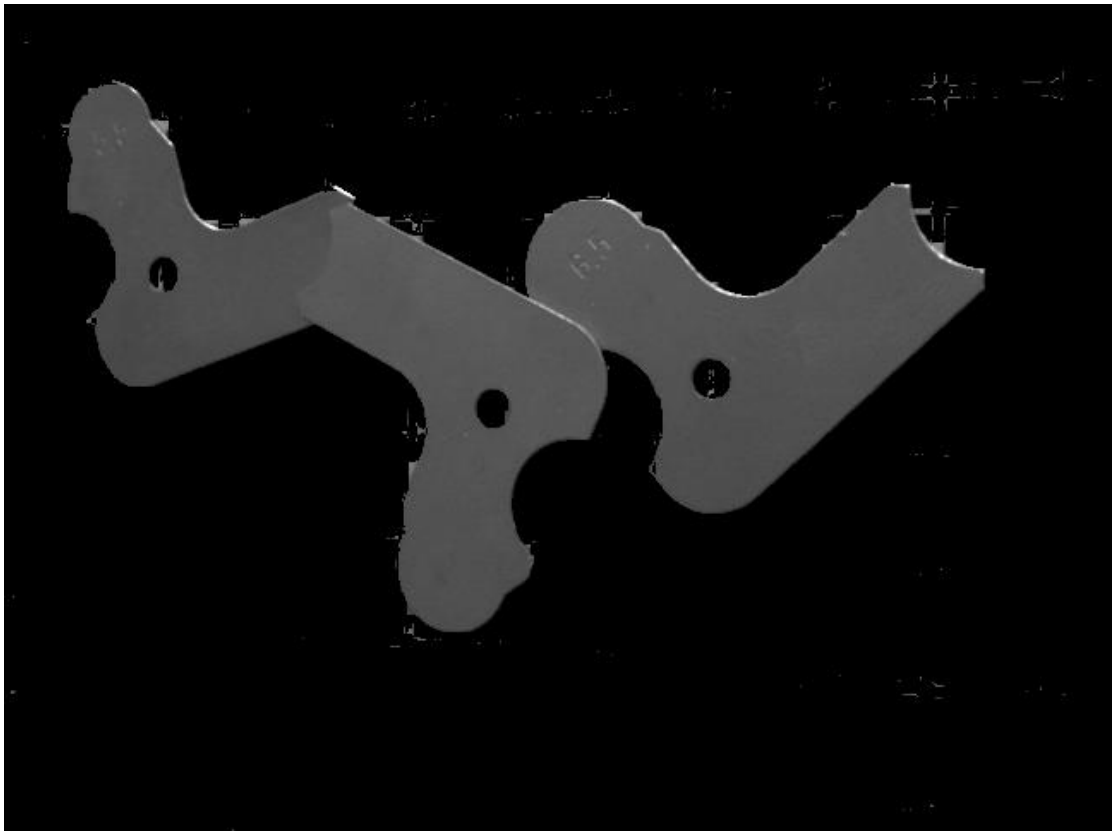


图 9　填补丢失元件

7. 去除剩余噪声

在频域中滤波始终无法将噪声完全去除，所以仍然需要进行空域操作。

空域中的操作与空域中去除网格噪声的方法一致：遍历图像的每一个点，以该点为中心构建一个 m*m 的正方形，统计其中像素值大于 60（元件包含像素点的像素值为 60 以上）的像素点数目，若该数目大于某一阈值（总数的 35%）则可以判定该点为元件上的点，否则，判定为元件外的点。

通过此方法去除剩余少量噪声，得到下图:

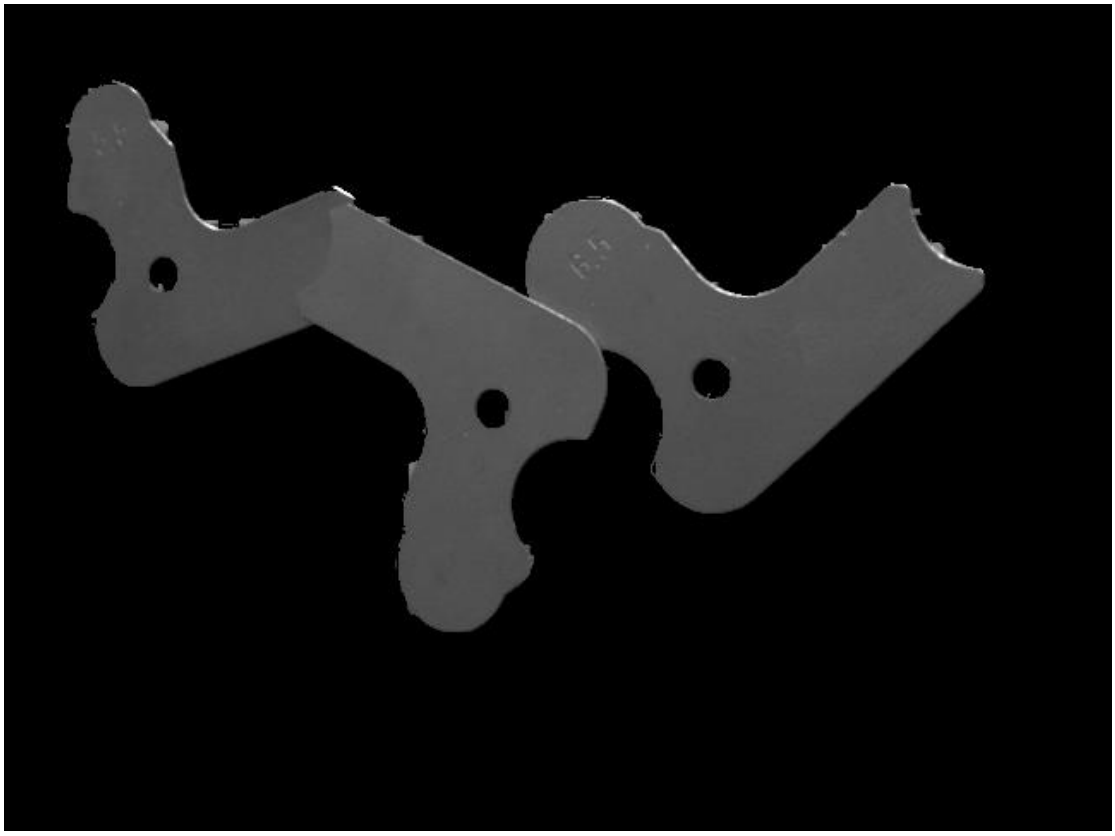


图 10　最终结果

2.1.2　空域方法

在空域观察噪声特点：呈线状，宽度不大。

所以以噪声点为中心建立一个 m*m 的正方形，只要 m 值合适，得到的正方形中包含像素值大于 60 的点必然不会太多。而在元件中的点，相应正方形中像素值大于 60 的点会占绝大多数（边缘部分除外）。

利用这点差别，通过多次去线操作，最终也可以得到比较理想的工件提取结果。

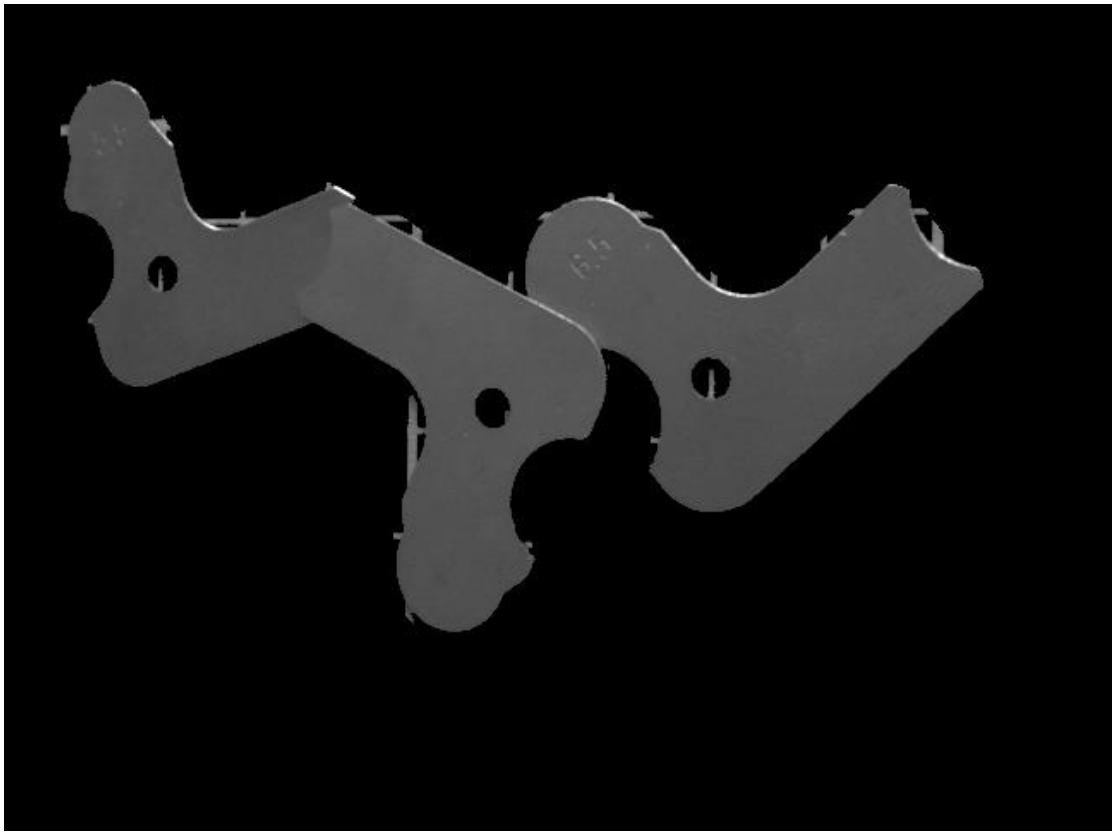


图 11　5 次去线操作结果

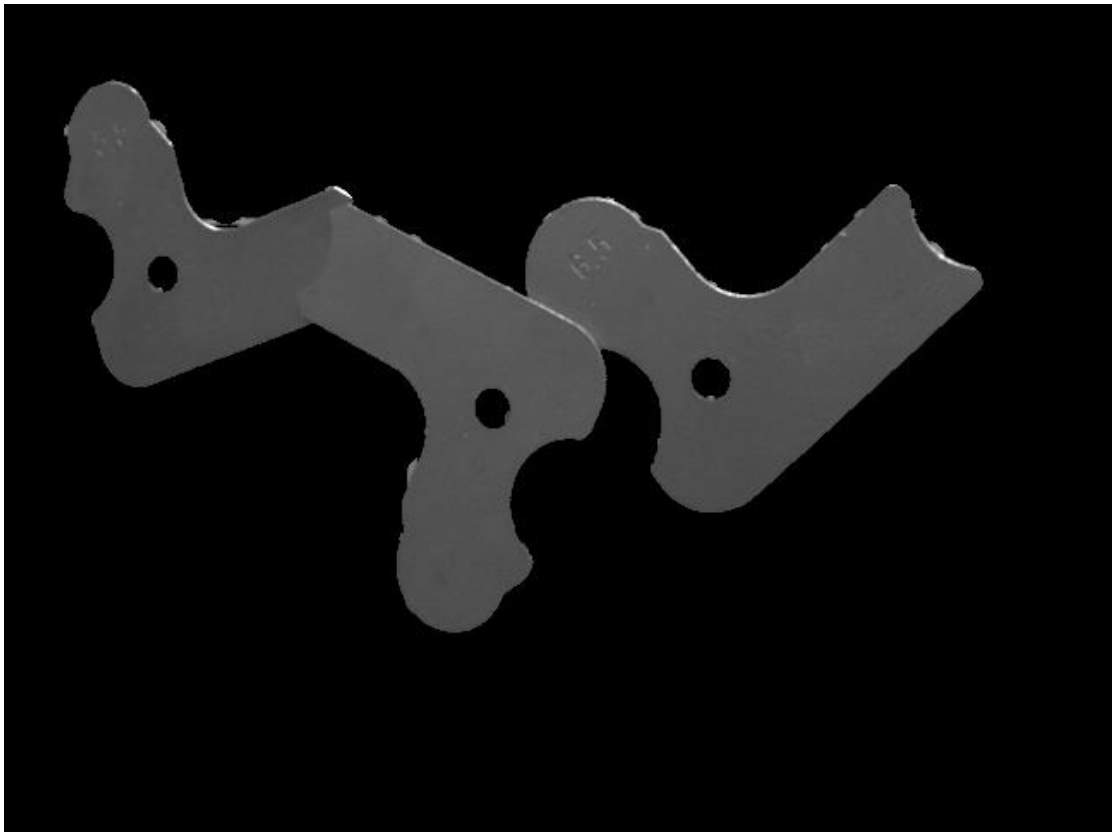


图 12　40 次去线操作结果

2.2　元件分离

元件的提取需要 canny 算子先得到边缘，再通过区域生长确定每个元件的空间位置，最后一步还原操作，根据像素点的空间位置还原为原图中对应像素点。

将去除噪声后的元件图分离需要以下几个步骤:

1.　两次 canny 操作

两次 canny 操作的目的是为了得到边缘比较小且能够区分三个元件。

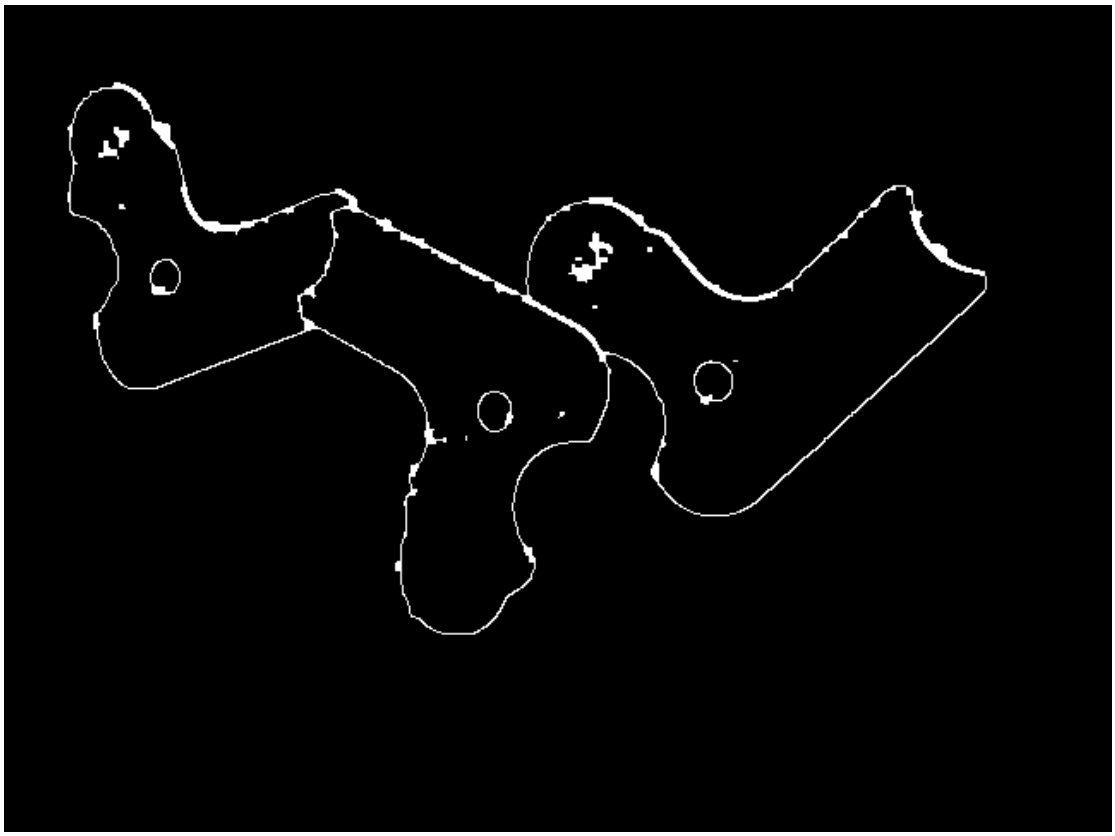


图 13　边缘粗糙

图 13 的边缘比较粗糙不利于图像信息保存，但是可以区分元件 1（左边）和元件 2（中间），因为两个部分中间被边缘阻断，区域生长时不会生成一起。

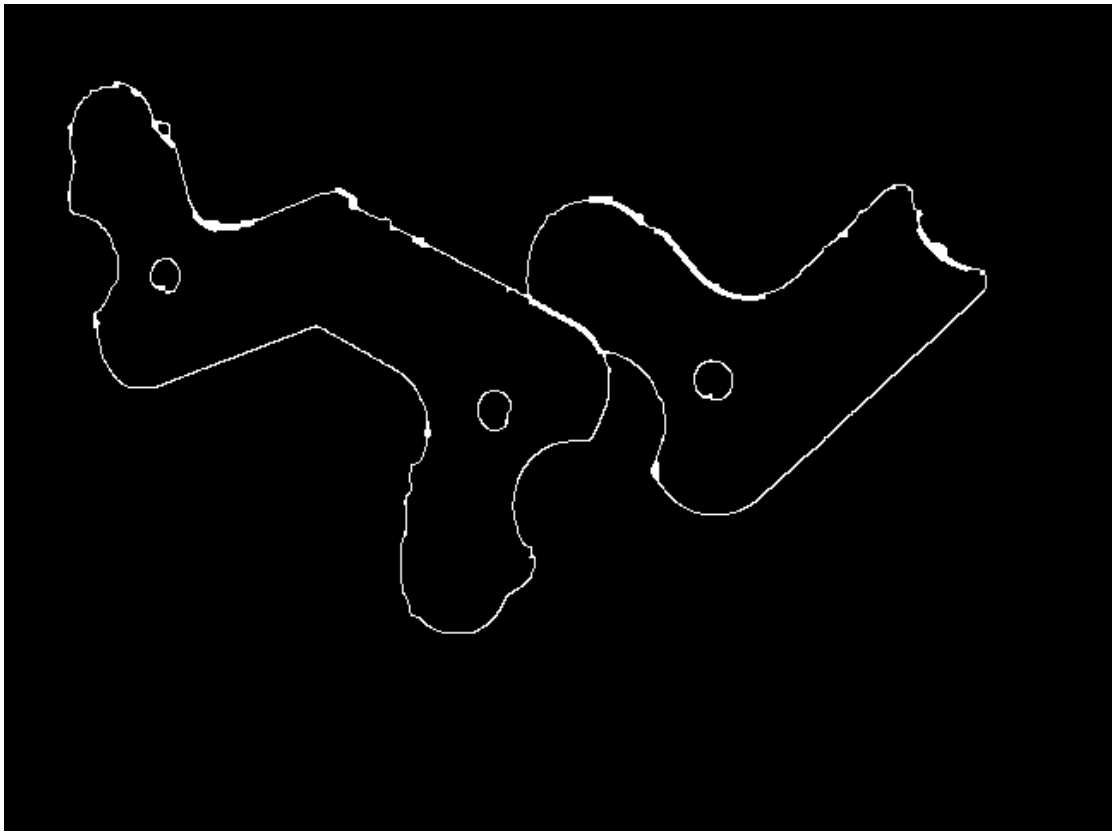


图 14　边缘细致

图 14 的边缘比较细致，看着比较纯净。但是元件 1 和元件 2 没有被区分处出来，所以需要图 13 和图 14 一起才可以得到比较好的元件空间位置。

2.　区域生长

区域生长可以将一个一个连通区分割开来，这恰好符合本次作业的要求。

区域生长分割出来的三个元件。

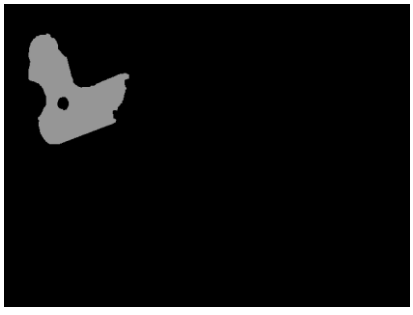


图 15　元件 1 边缘

图 16　元件 2 边缘

图 17　元件 3 边缘

3.　像素值还原

根据空间位置对应关系，由边缘映射到原图，便可以得到元件提取图。

图 18　元件 1

图 19　元件 2

图 20　元件 3

3.　基本原理

区域生长:

区域生长是图像分割技术的一种。区域生长的基本思想是将具有相似性的像素集合起来构成区域。首先对每个需要分割的区域找出一个种子像素作为生长的起点，然后将种子像素周围邻域中与种子有相同或相似性质的像素（根据事先确定的生长或相似准则来确定）合并到种子像素所在的区域中。而新的像素继续作为种子向四周生长，直到再没有满足条件的像素可以包括进来，一个区域就生长而成了。

本次作业中区域生长是根据闭合边缘确定三个区域，其具体步骤如下：

1. 手动选取元件 1 中某个点(x_0, y_0)作为生长起点。将其像素值赋为 200。并标记为已处理。

2. 以(x_0, y_0)为中心，考虑其上下左右四个点是否满足生长条件（像素值与(x_0, y_0)处像素值相等，均为 0），若满足则将其压入堆栈中。

3. 从堆栈中取出一个像素，把它当作(x_0, y_0)返回到步骤 2;

4. 当栈为空时，代表一个连通区域被选出来了。即一个元件的空间位置被选出来了。

5. 手动选取元件 2 中一点，重复步骤 2-4。

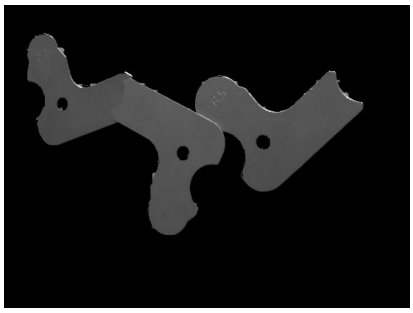
6. 手动选取元件 3 中一点，重读步骤 2-4。

4. 两种方法的比较

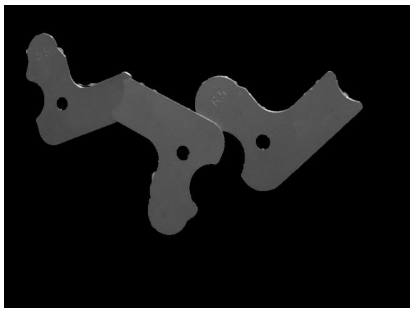
频域去噪：利用了噪声的周期性特点，过程比较复杂，但是计算量不是很大。

空域去噪：利用了噪声的空间特征（呈线状），过程比较简单，但是计算量比较大。

最终效果的比较：



频域去噪



空域去噪

从最终效果来看，空域去噪在边缘毛刺方面更好；在保持原本形状方面频域更好一些（比如元件的尖角处频域处理更好）。

5. 遇到的问题以及解决

本次作业有一个问题多次遇到，那就是访问 Mat 某点的像素值时总在<uchar>和<float>处搞错，主要原因是调用系统函数时，返回的有可能是归一化后的 Mat，这时候再用<uchar>就会出现一项不到的错误，而自己还很难发现。

解决方案:

统一模板，都用<uchar>或者<float>，可以使用类型转换来实现。