1 Exercises

1.1 Color Spaces

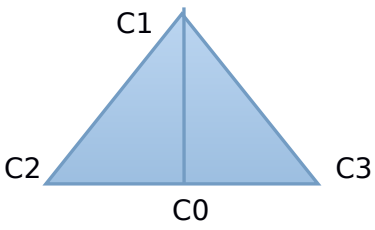
1. 现实中，HSI 模型更符合人描述和解释颜色的方式，还可以接触图像中的颜色和灰度信息的联系，而 RGB 模型是最通用的面向硬件的模型，通常应用于彩色监视器和彩色饰品摄像机。

2. HSI 模型中，原色间相隔 120 度，原色与二次色间隔 60 度，二次色之间也是相隔 120 度。对于给定图像的 HSI 模型，对其 R、G、B 原色加上 60 度的偏转，那么原图的显示就会变成二次色为基底的显示，即 CMY 模型显示。

1.2 Color Composition

因为色度图中连接任意两点的直线段定义了所有不同颜色的变化，这些颜色可以由这两种颜色的加性组合得到。又，给定色度图上不在同一直线上的 C1、C2、C3 三个颜色点，要证明三角域内的颜色都可以由给定的三个颜色组合得到，只需要证明三角域内任意一点可以由 C1、C2、C3 的线性组合得到。

证明：



由上图，设任意给出一点 C 在三角域内，过 C 点做 C1C 线段，与 C2C3 相交于 C0 点，则有，C 点颜色可由 C0 和 C1 两种颜色的加性组合得到

C = a*C0 + b*C1

同理可得，C0 = c*C2 + d*C3

综上所得，C = b*C1 + a*c*C2 + a*d*C3，C 是 C1C2C3 三个颜色的加性组合，得证。

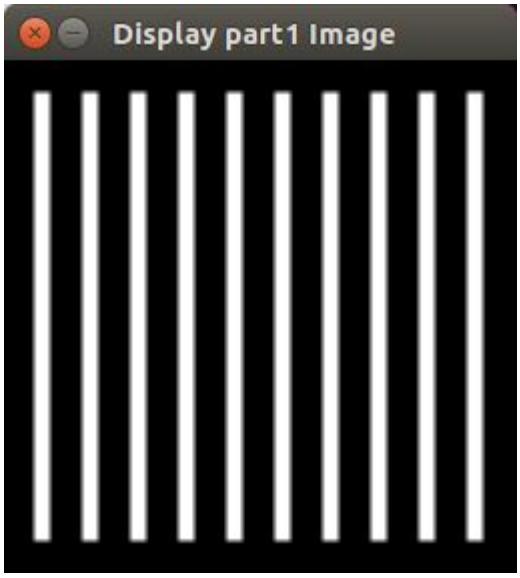
2 Programming Tasks

2.1 Pre-requirement

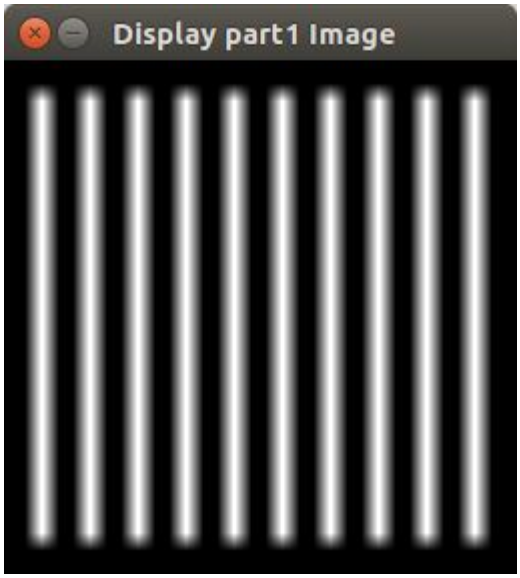
2.2 Image Filtering

1.

（1）3*3 arithmetic mean filters



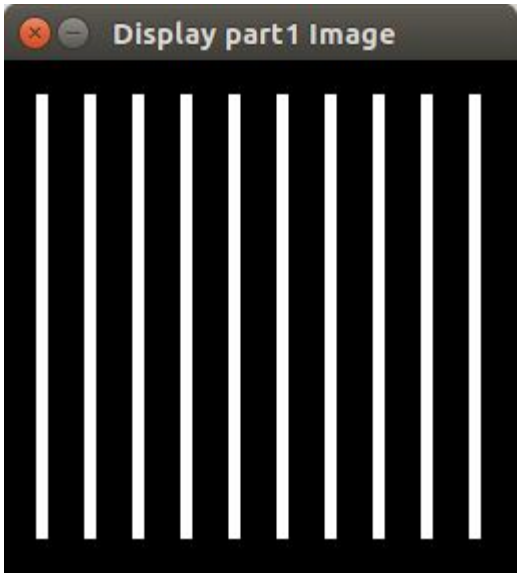
（2）9*9 arithmetic mean filters



由上图比较可以看出，算术均值滤波器模糊了图像，白色区域宽度明显增加

2.

（1）3*3 harmonic mean filters

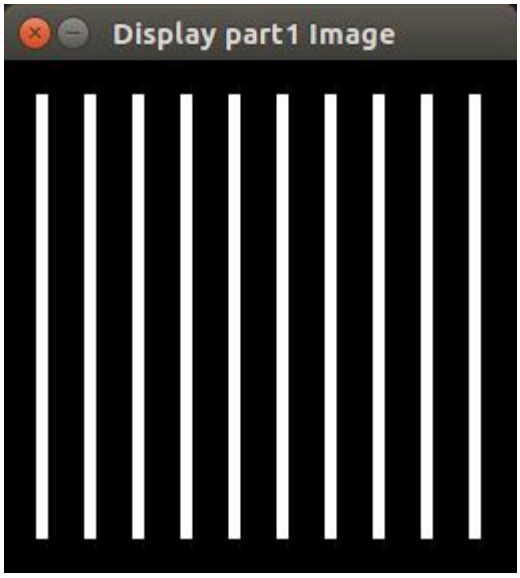


（2）9*9 harmonic mean filters

由上图比较可知，调和均值滤波器主要去除盐噪声（255），这里把白条作为盐噪声，3*3 大小时，白条明显变细了，到了 9*9 大小时，白条已经消失了。

3.

（1）3*3 Q=-1.5 contraharmonic mean filters



（2）9*9 Q=-1.5 contraharmonic mean filters

Q<0:主要用于消除盐噪声，对比上图，Q= -1 是为调和均值滤波，Q = -1.5 时与调和均值滤波器的处理效果相似。

2.3 Image Denoising

1. noise generator

高斯噪声生成：

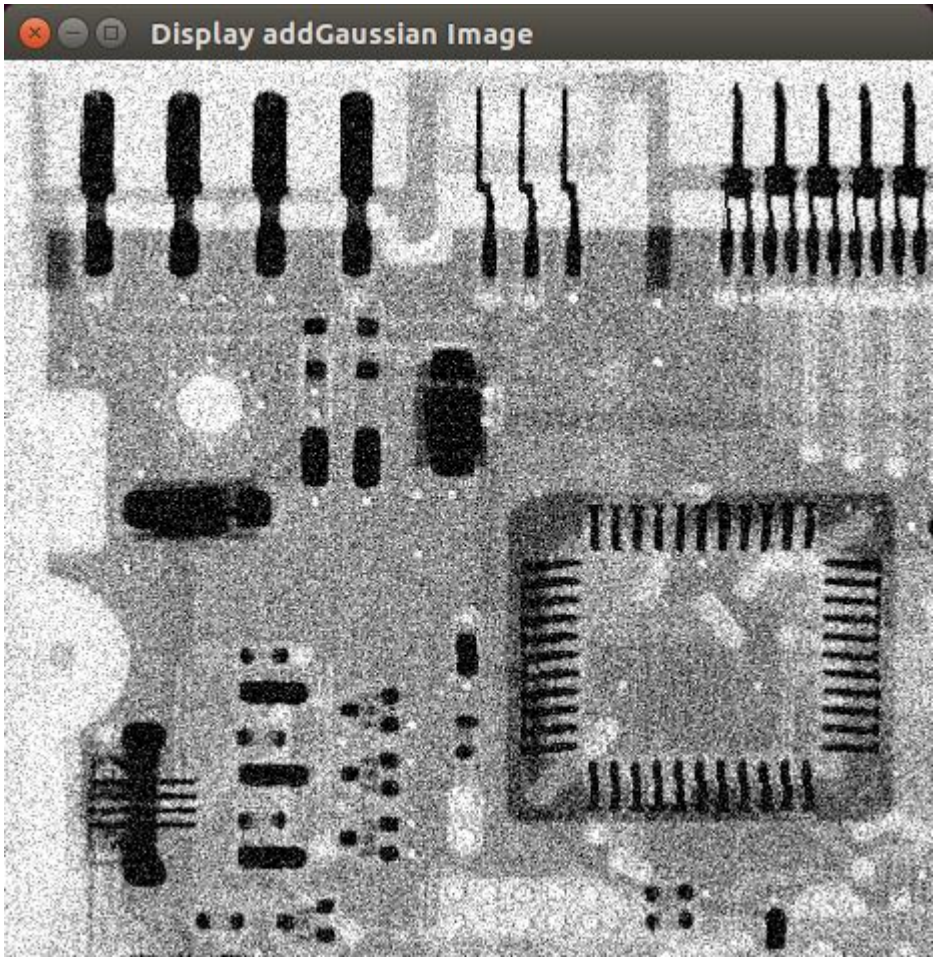
```
double Random() {
	double u1 = (double)(rand()/(double)RAND_MAX);
	double u2 = (double)(rand()/(double)RAND_MAX);
	double Z = sqrt(-2 * log(u2)) * cos(2 * PI * u1);
	return Z;
}

Mat addGaussian(Mat &input, int mean, int standard) {
	srand((unsigned)time(NULL));
	int row = input.rows;
	int col = input.cols;
	int channel = input.channels();
	Mat output = input.clone();
	for (int i = 0 ; i < row; i ++) {
		for (int j = 0; j < col; j ++) {
			double result = Random() * standard + mean;
			if (channel == 3) {
				for (int k = 0; k < 3; k ++)
					output.at<Vec3b>(i, j)[k] = saturate_cast<uchar>(result + output.at<Vec3b>(i, j)[k]);
			}
			else {
				output.ptr<uchar>(i)[j] = saturate_cast<uchar>(result + output.ptr<uchar>(i)[j]);
			}
		}
	}
	return output;
}
```

椒盐噪声生成：

```
Mat addSaltPepper(Mat &input, double probability, bool salt) {
	int row = input.rows;
	int col = input.cols;
	int channel = input.channels();
	Mat output = input.clone();
	int u1;
	int u2;
	int count = (int)(probability * row * col);
	while (count -- ){
		u1 = rand() % row;
		u2 = rand() % col;
		if (salt) {
			for (int k = 0; k < 3; k ++)
				output.at<Vec3b>(u1, u2)[k]= 255;
		}
		else {
			for (int k = 0; k < 3; k ++)
				output.at<Vec3b>(u1, u2)[k]= 0;
		}
	}
	return output;
}
```

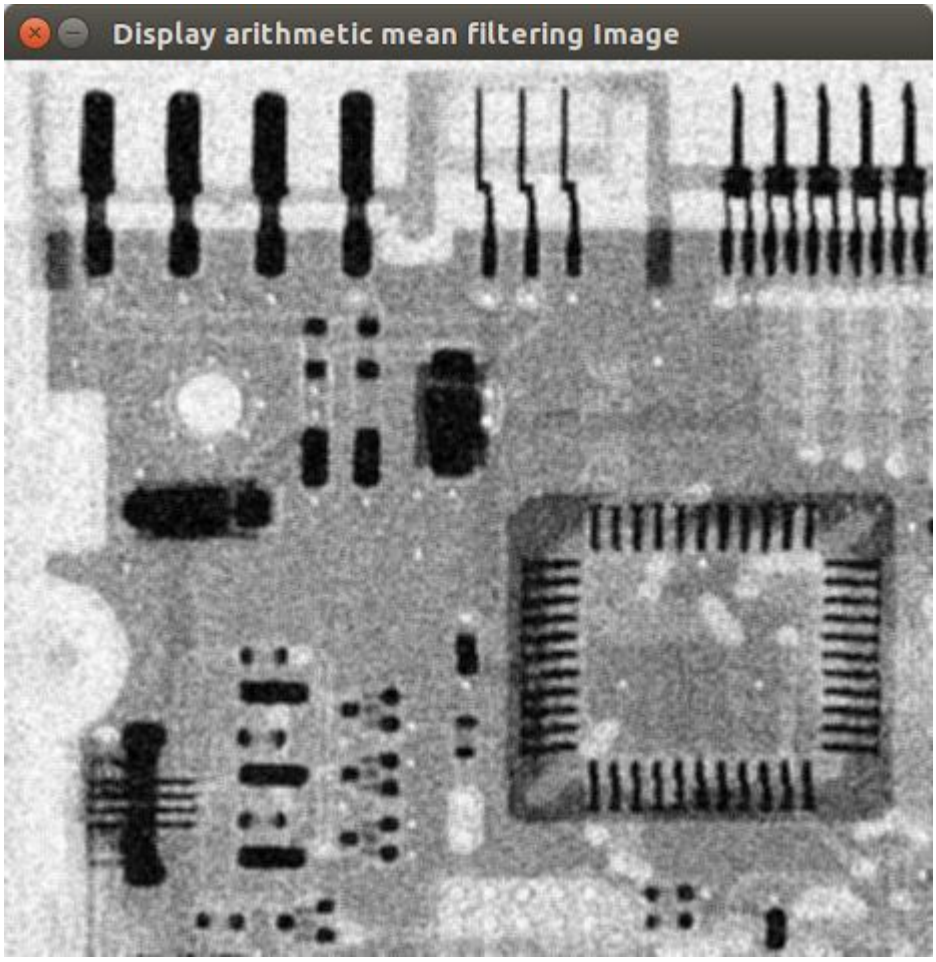
2. Add noise

(1)Add Gaussian noise

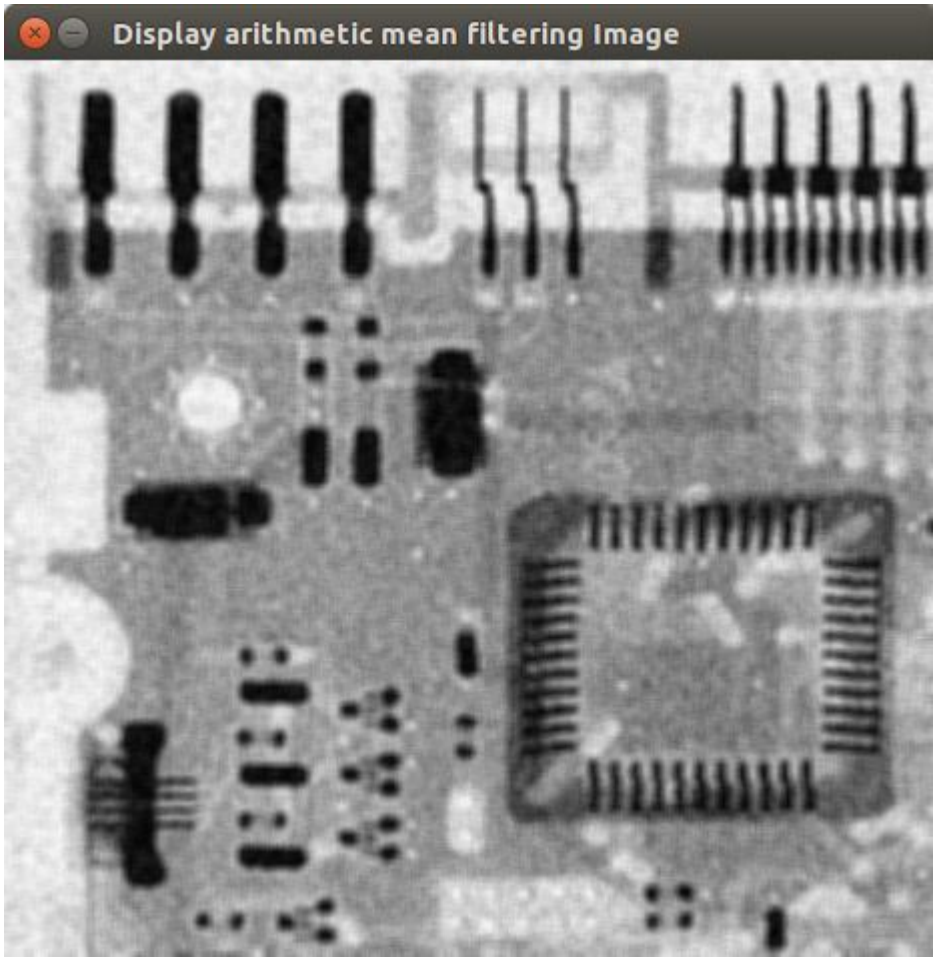


(2)arithmetic mean filtering

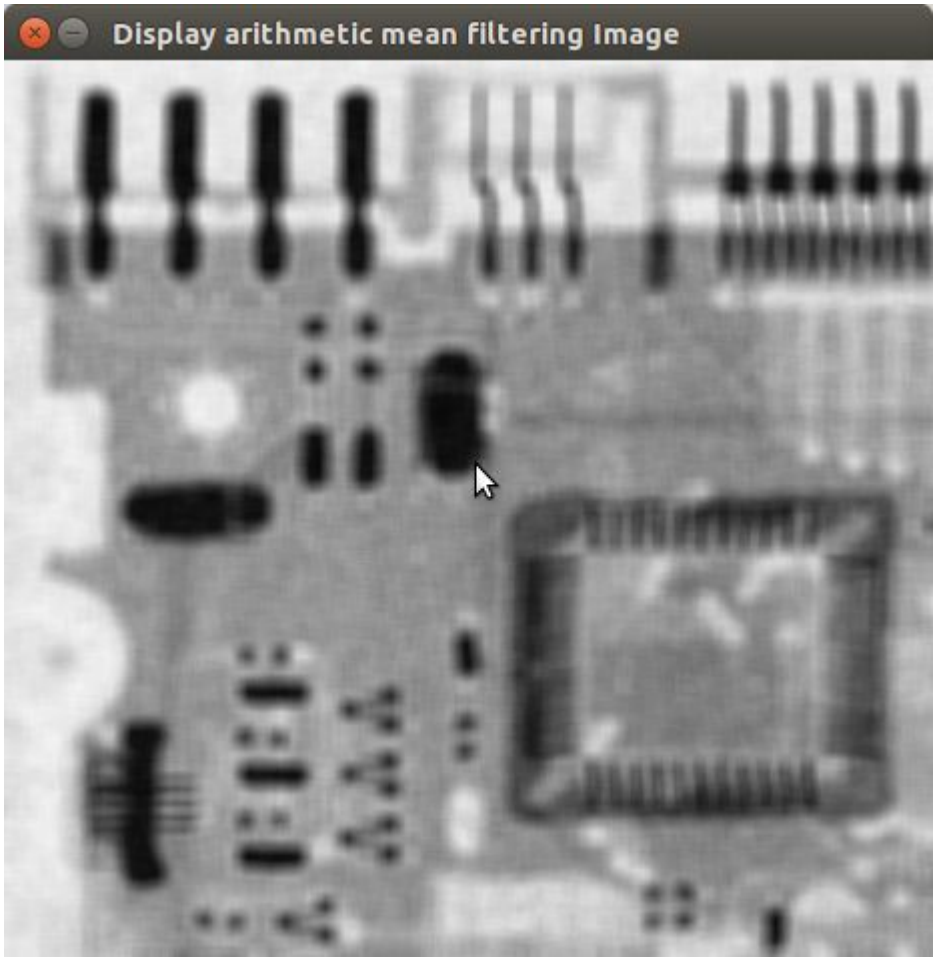
3*3 :



5*5：

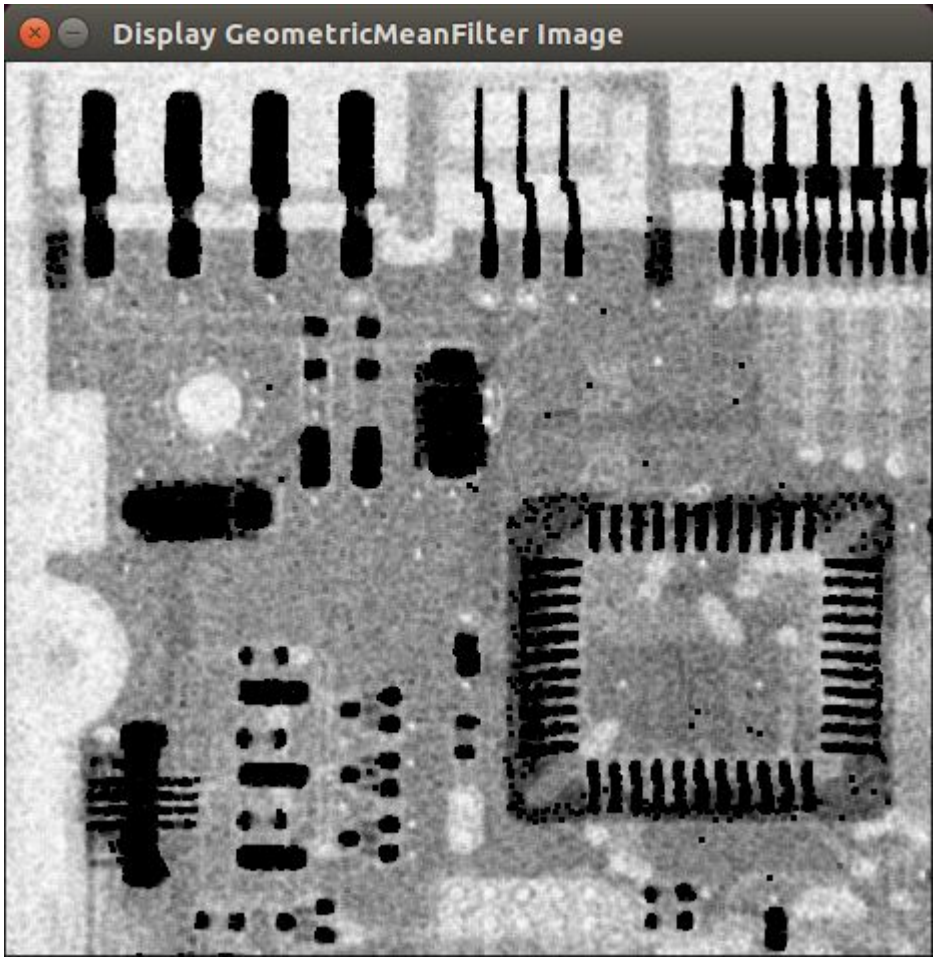


9*9：

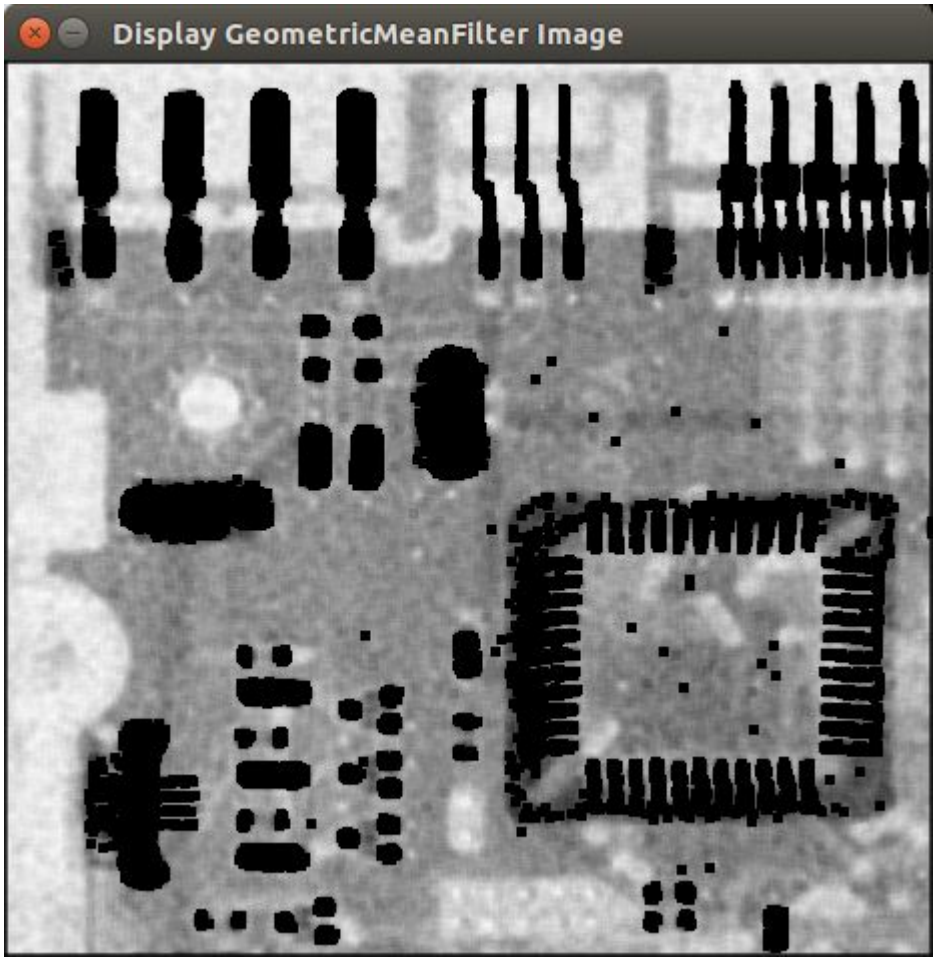


(3)Geometric mean filtering

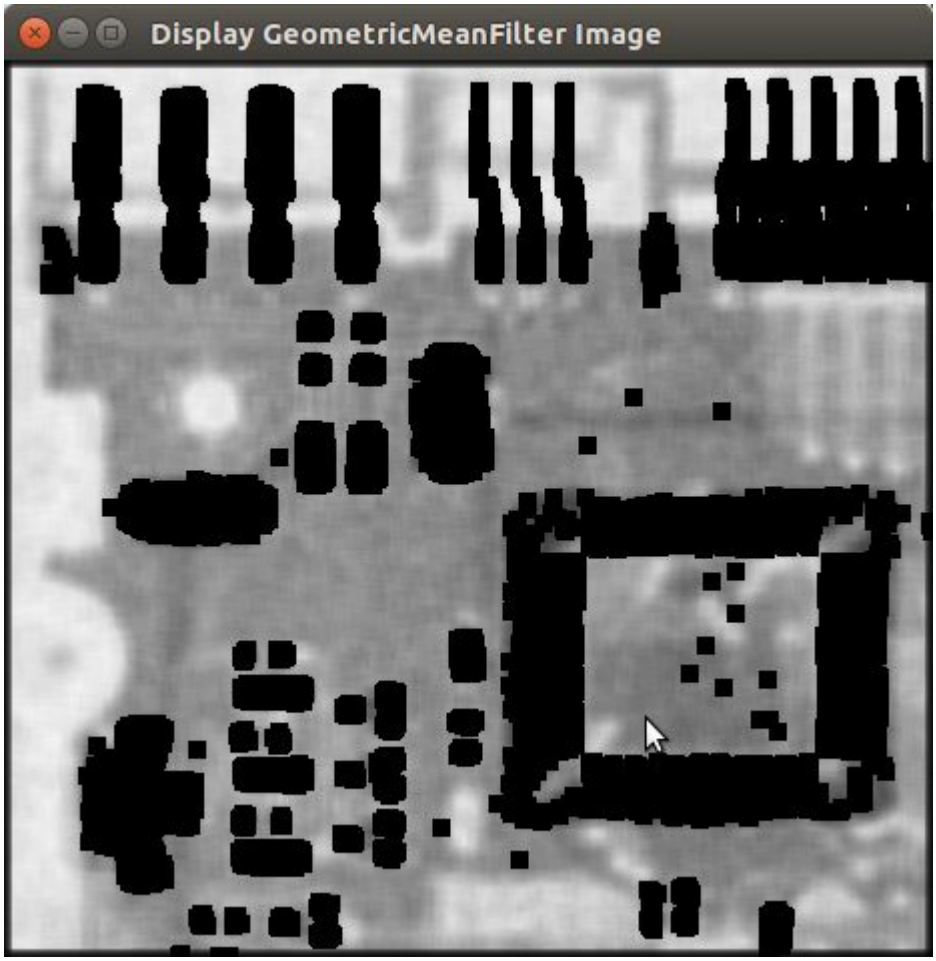
3*3 :



5*5：

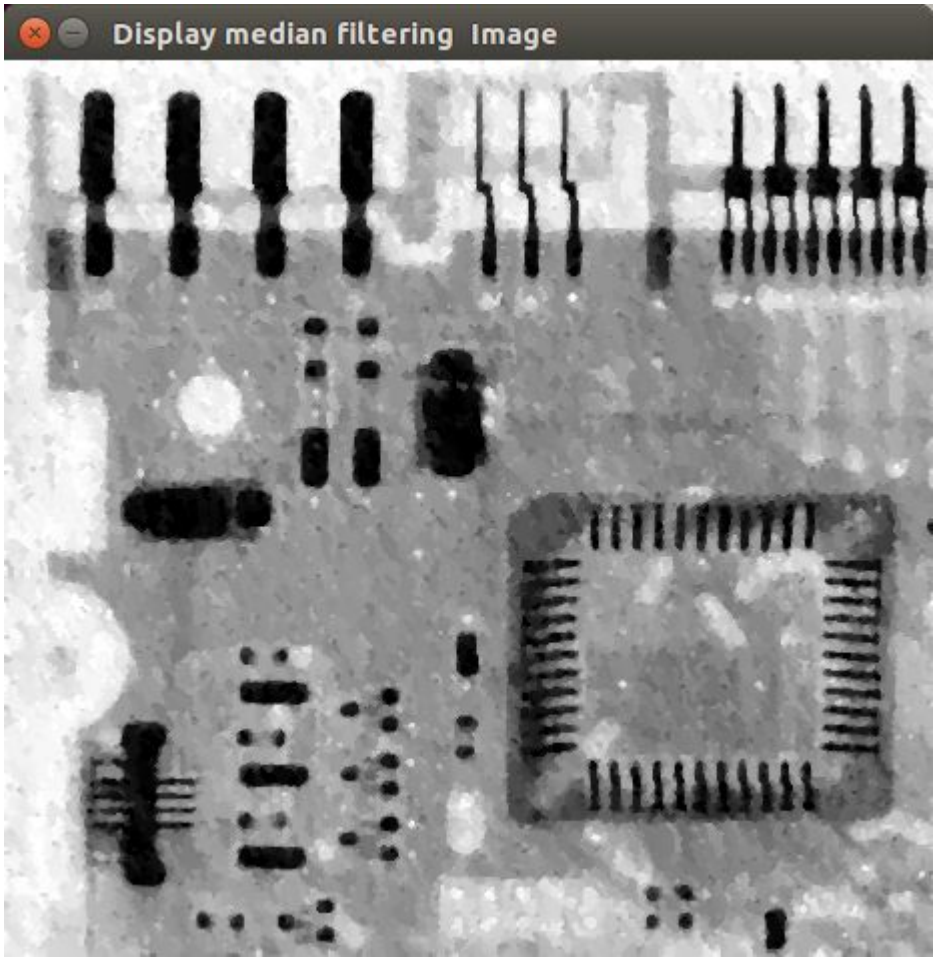


9*9：

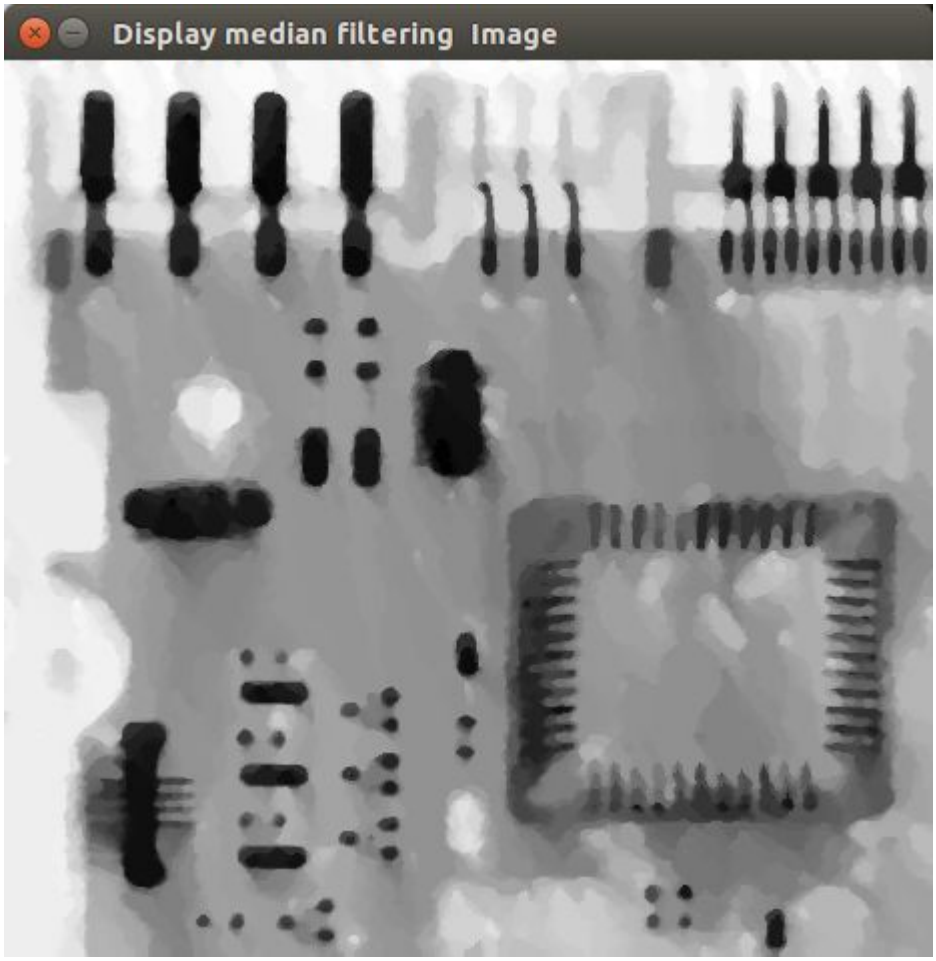


(4)median filtering

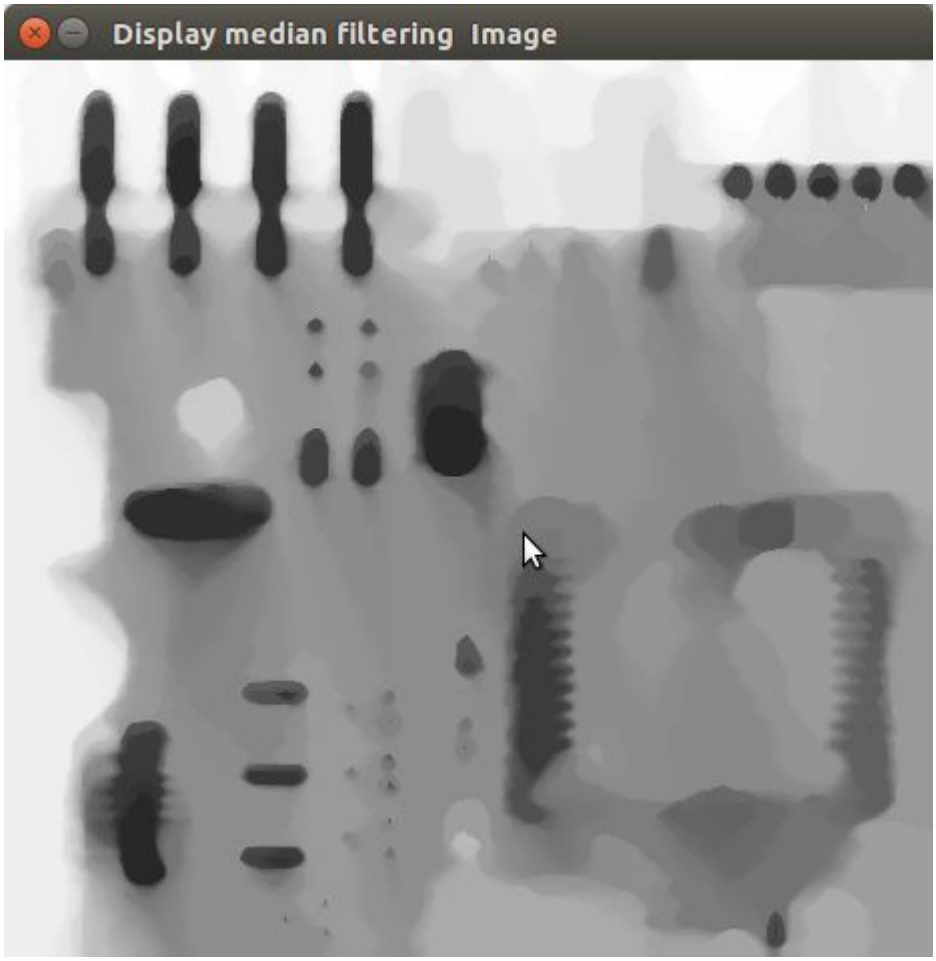
3*3 :



5*5：



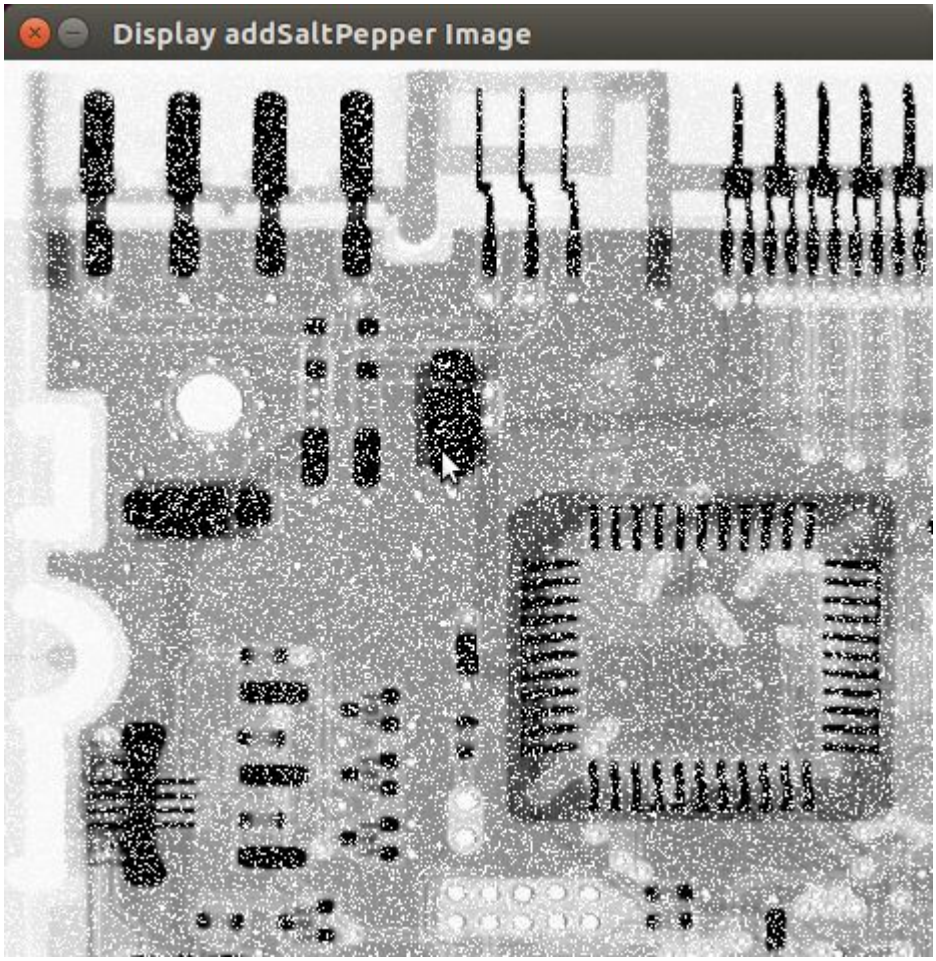
9*9：



（5）一些比较，比较上图可知，3*3 的窗口大小，处理过后仍然可以看到噪声，5*5 的大小会好一点，噪声相对减少，虽然模糊但基本上能看出原图，到了 9*9 的大小，就会变得“面目全非”了，已经几乎看不出原图了。故而像比较之下可以选择 5*5 左右大小做为最适窗口大小。

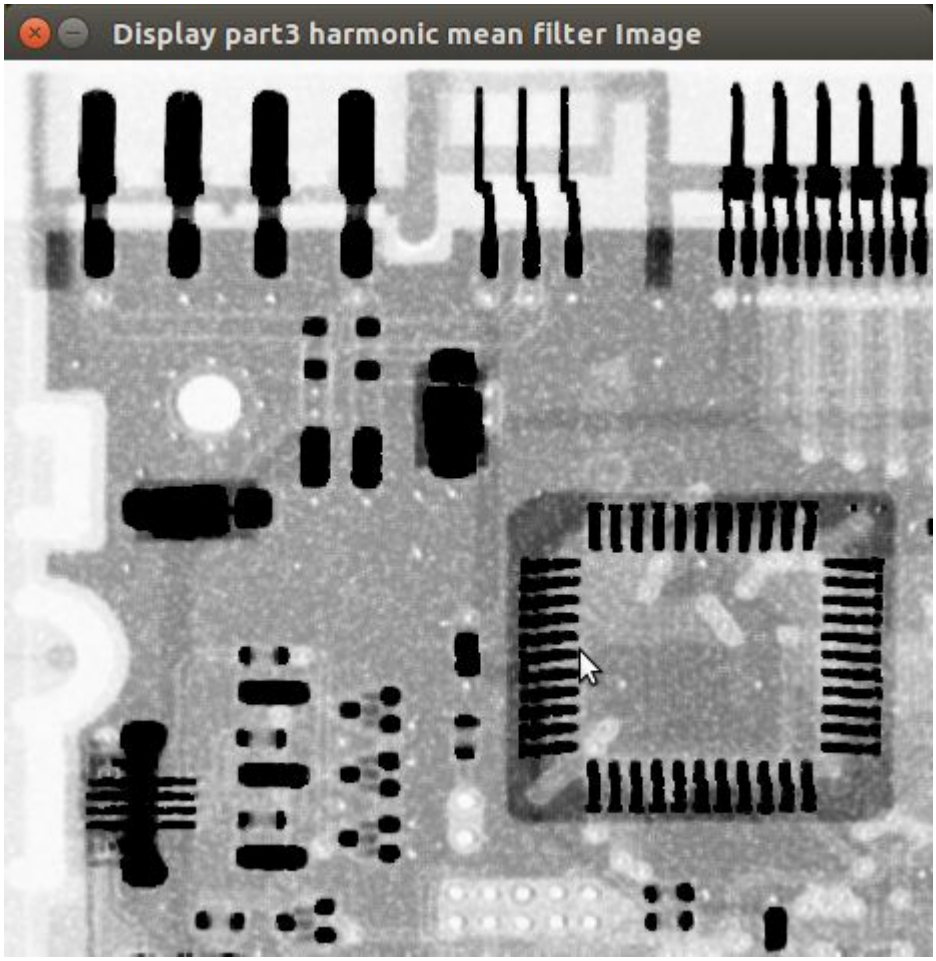
3. Add salt noise

(1)Add salt noise by setting its probabilities to 0.2

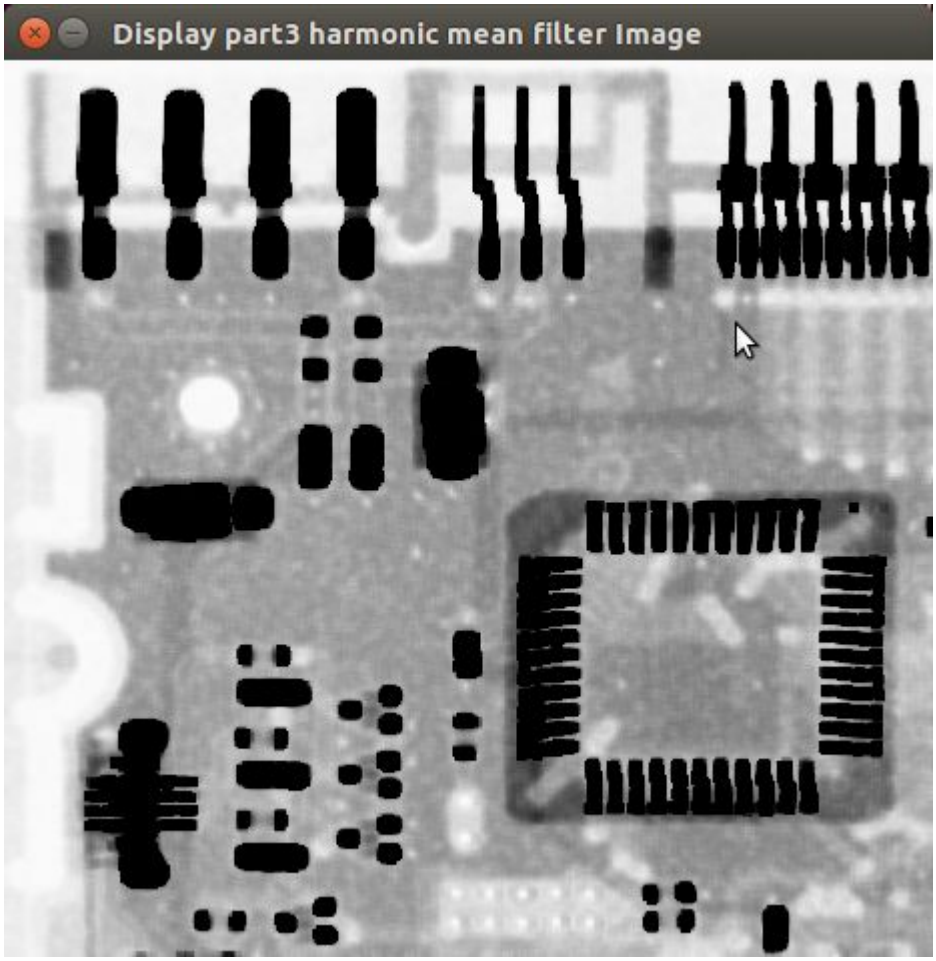


(2)harmonic mean filtering

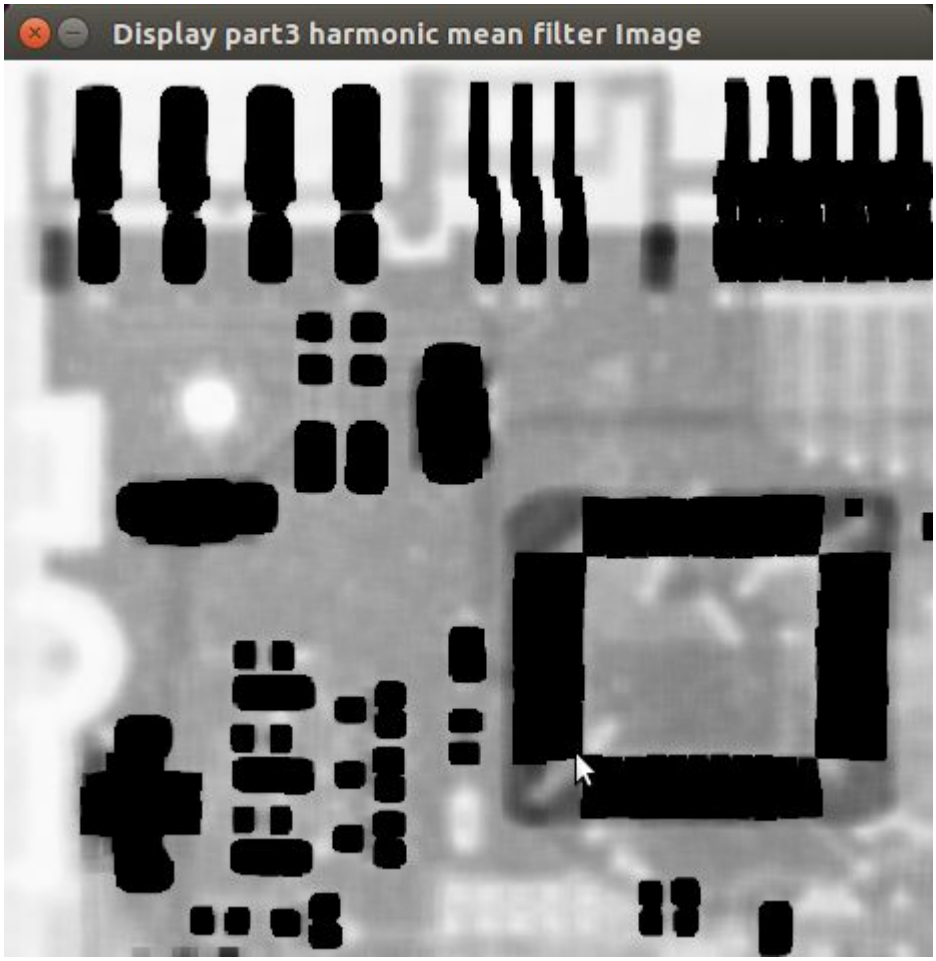
3*3:



5*5:



9*9:

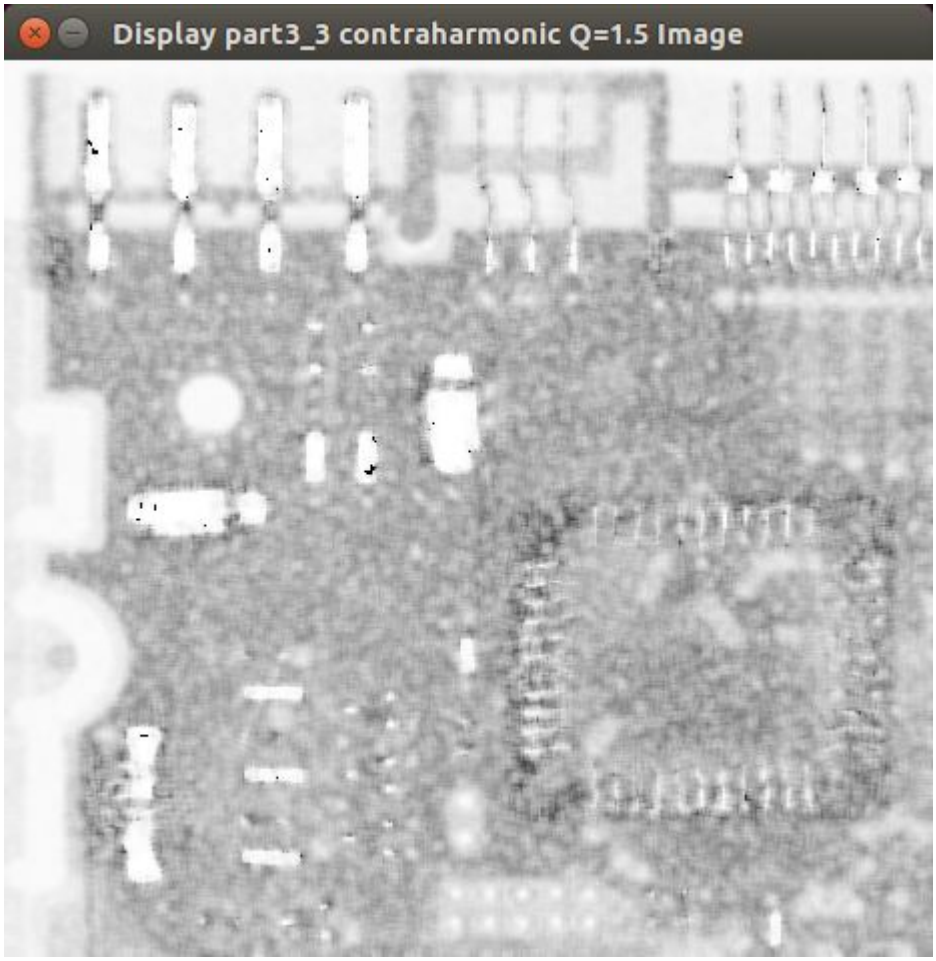


(3)Contraharmonic mean filtering Q > 0

取值 Q=1.5

3*3：

5*5：

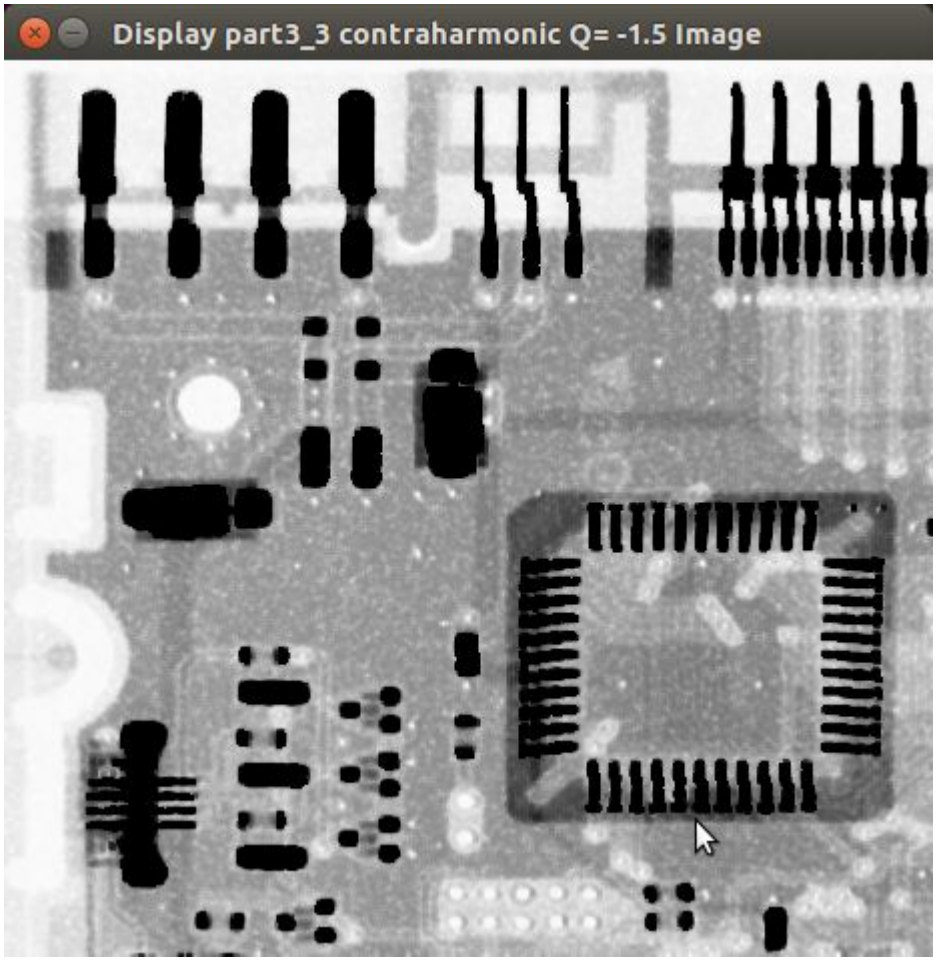


9*9：

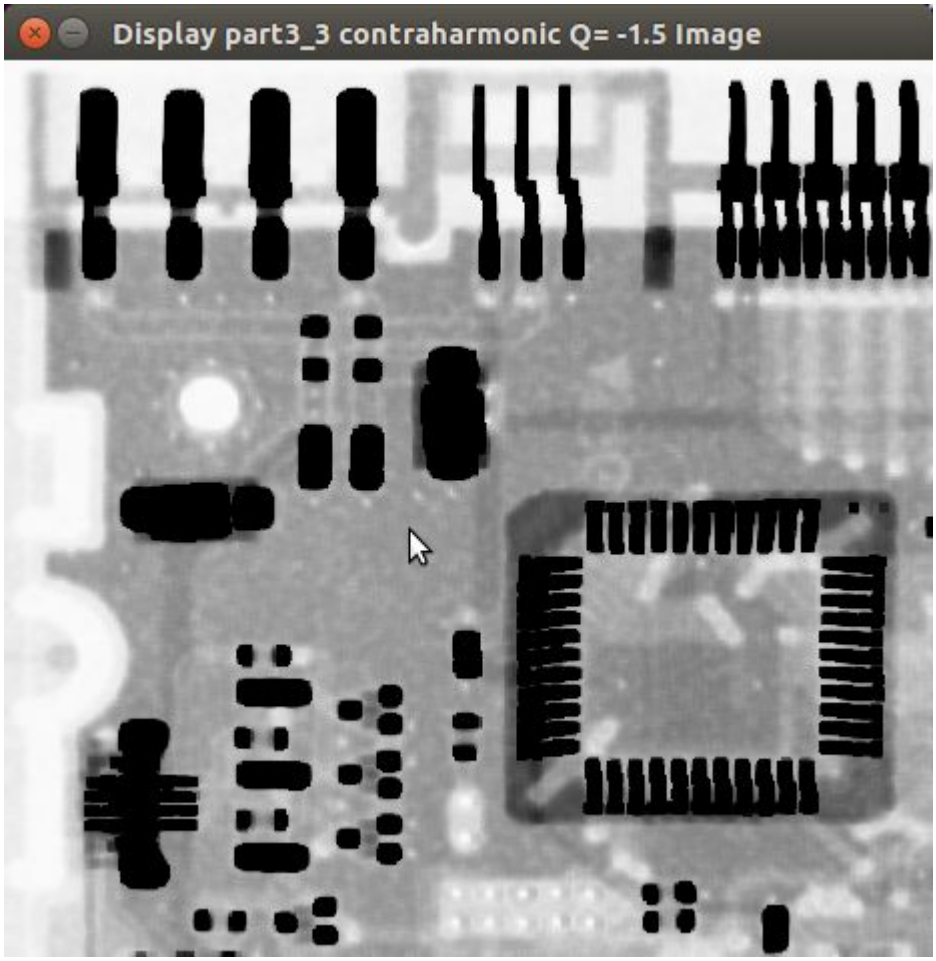
(4)Contraharmonic mean filtering Q < 0

取值 Q = -1.5

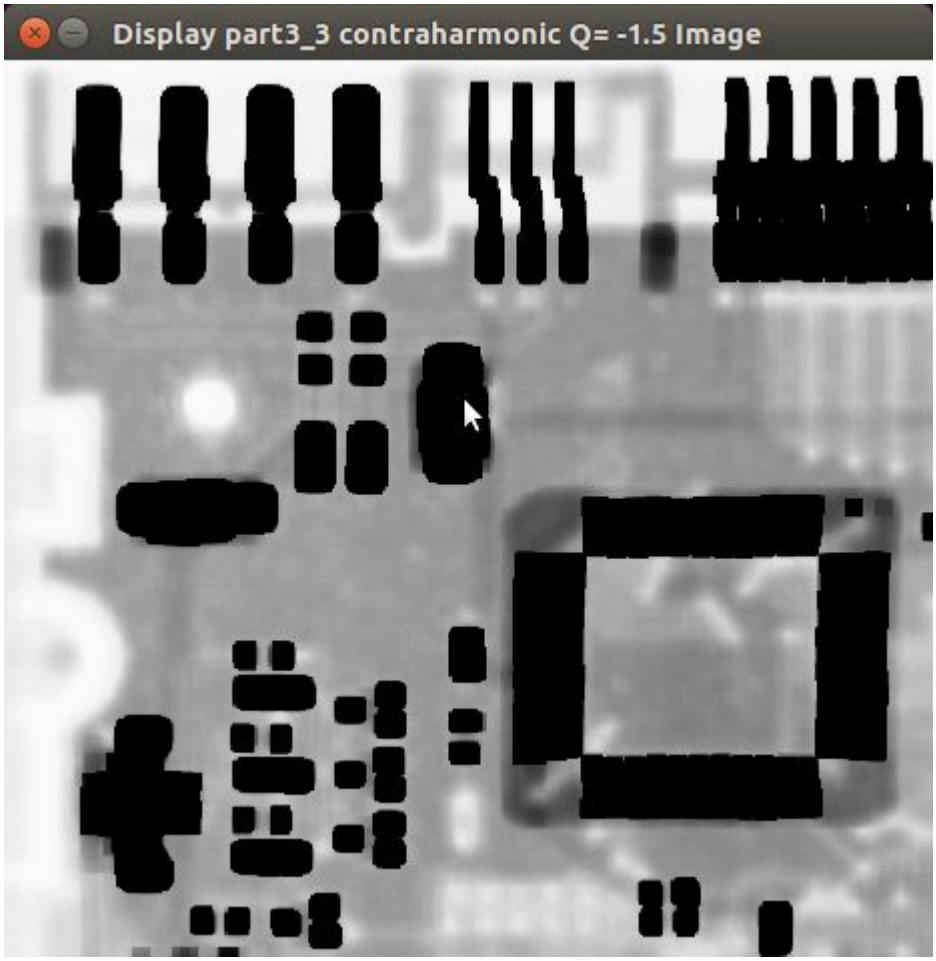
3*3：



5*5：



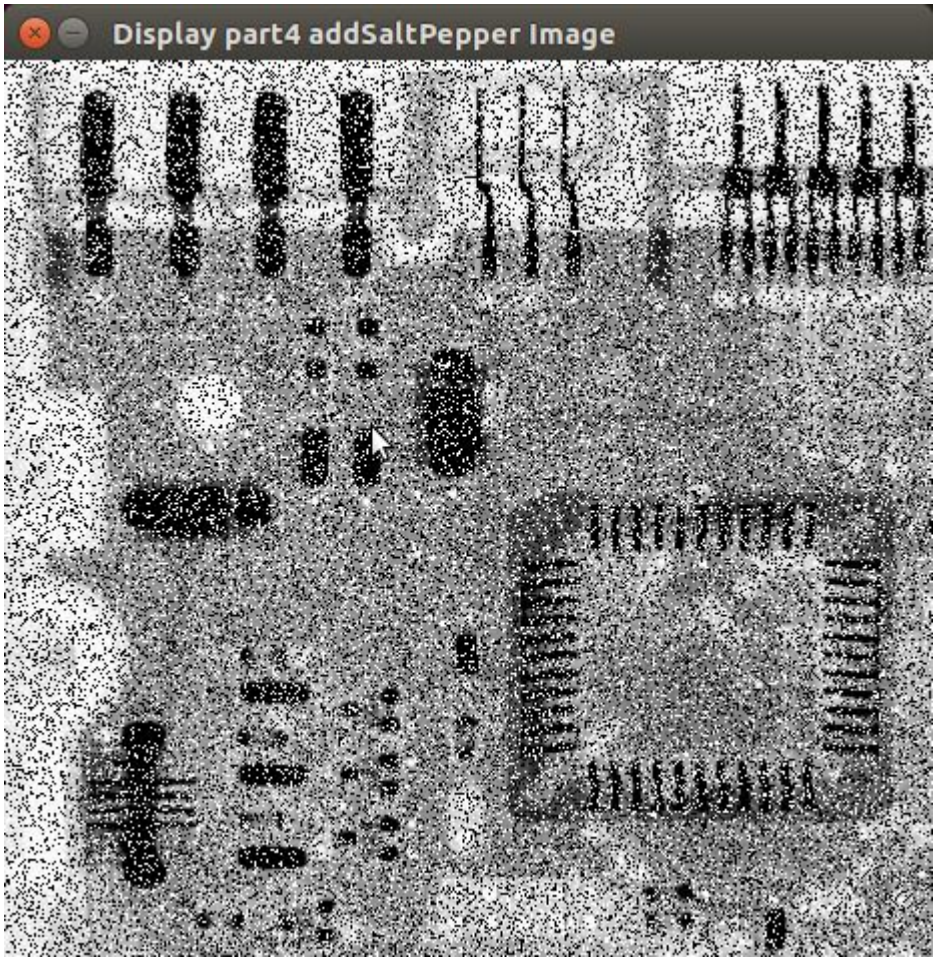
9*9：



（5）一些思考：

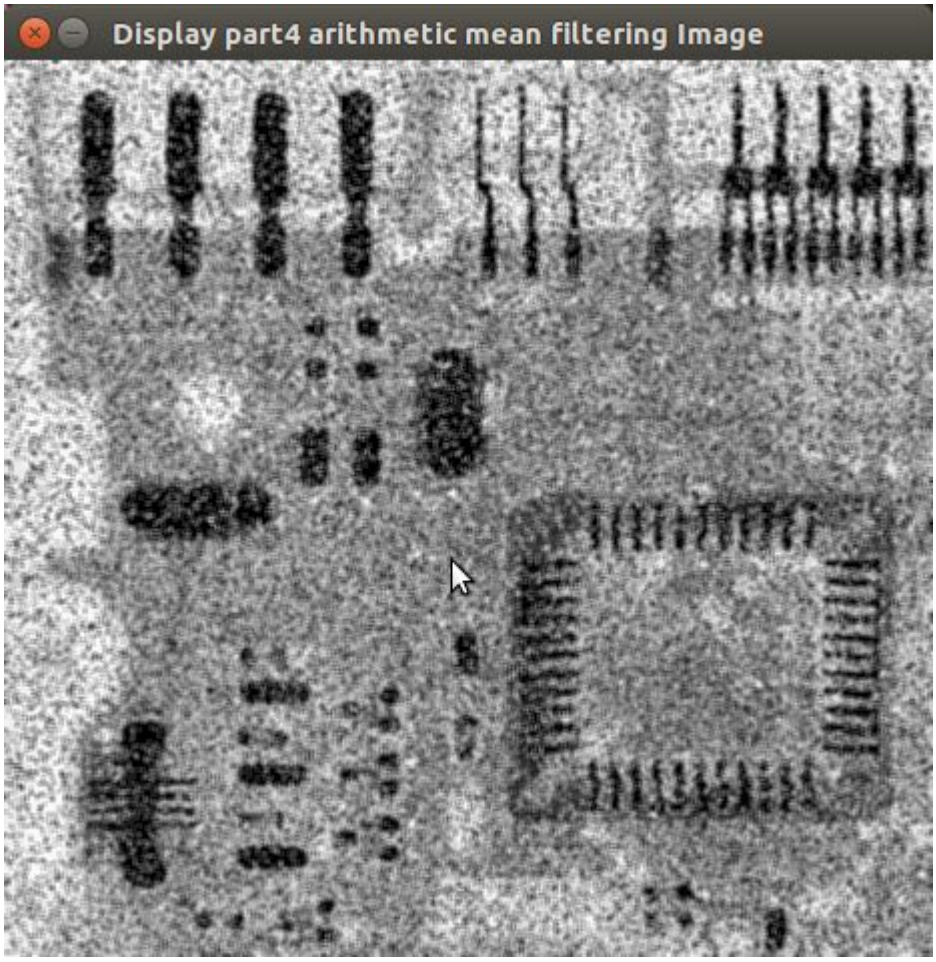
由上图比较可知，在 Q < 0 的时候对噪声的处理效果最好。椒盐噪声是属于脉冲噪声，而调和均值滤波（Q = -1）对盐粒噪声、高斯噪声的处理效果比较好，Q > 0 时的反调和均值滤波器适合对胡椒噪声处理，所以不能确定噪声类型，选错了 Q 值的话，就会造成破坏性的后果，图片愈加“面目全非”。

4. Add salt-and-pepper noise

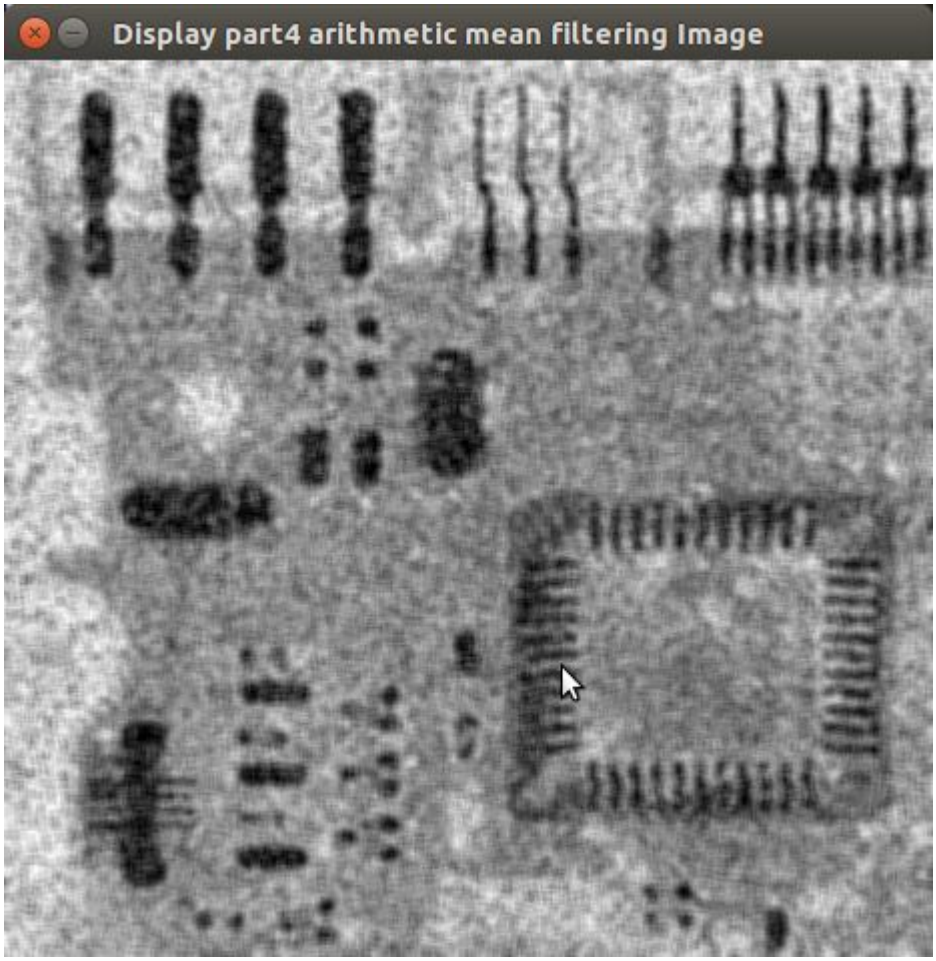


(1) arithmetic mean filtering

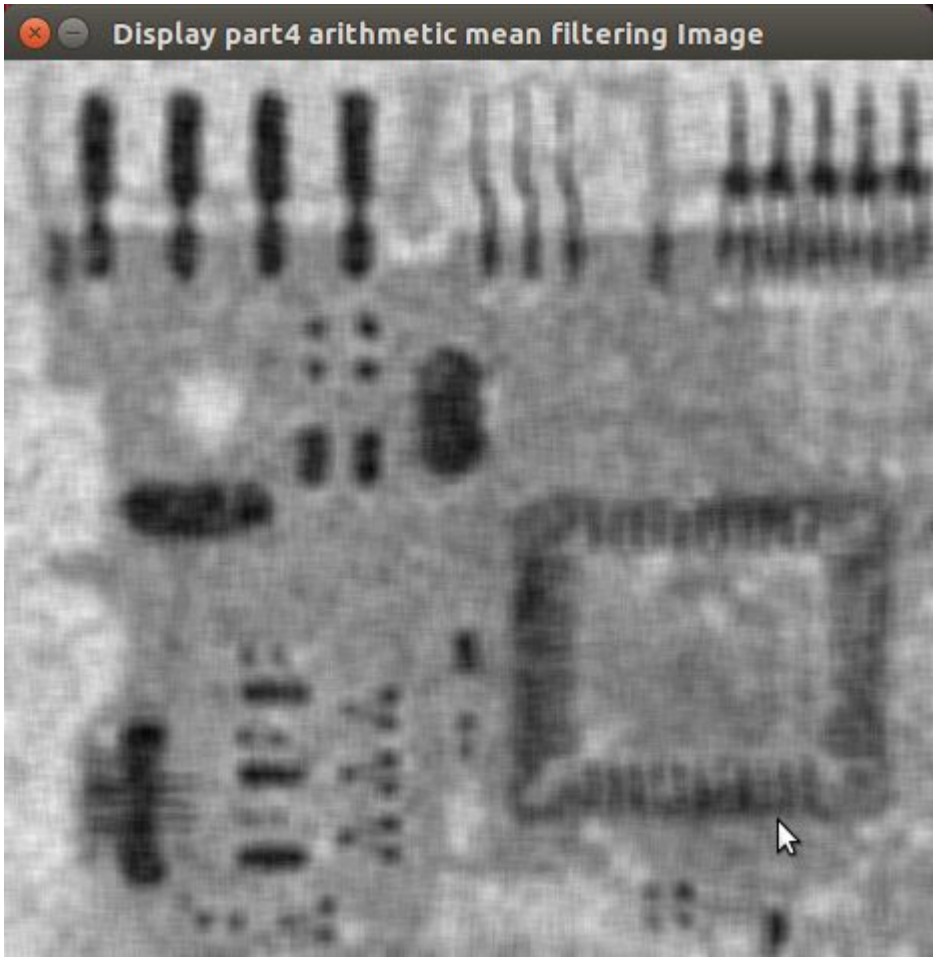
3*3 :



5*5：

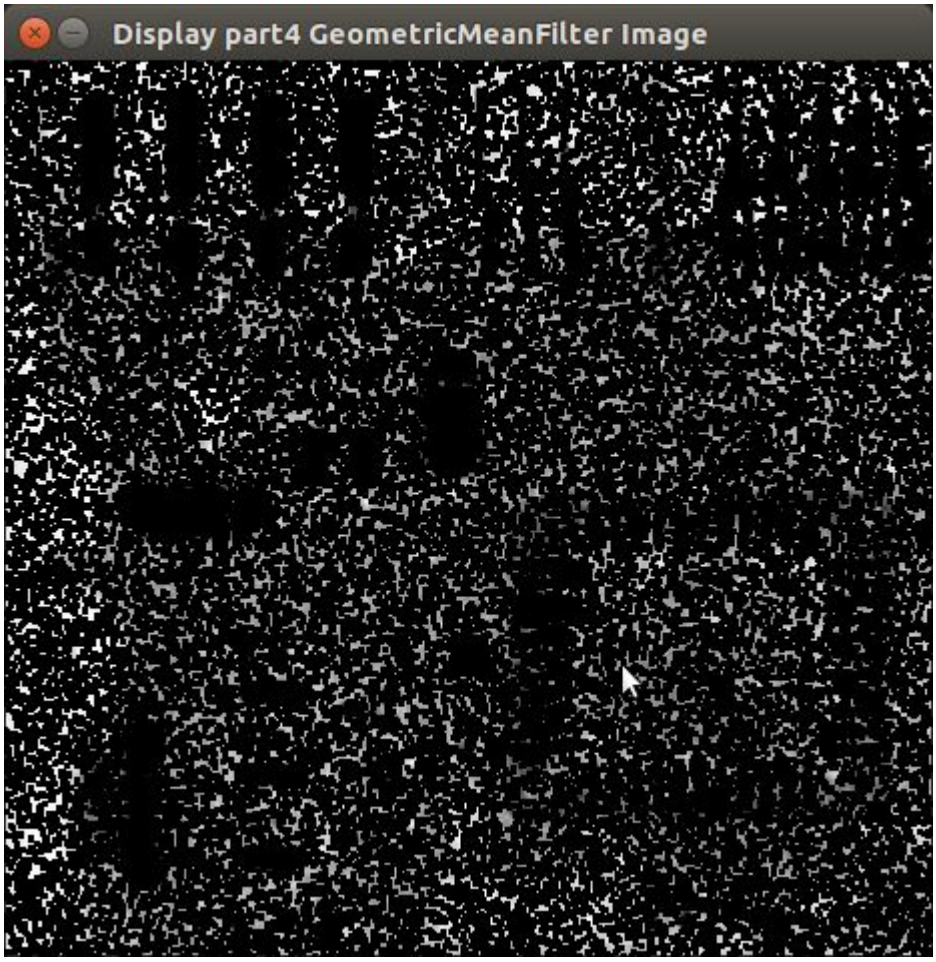


9*9 :



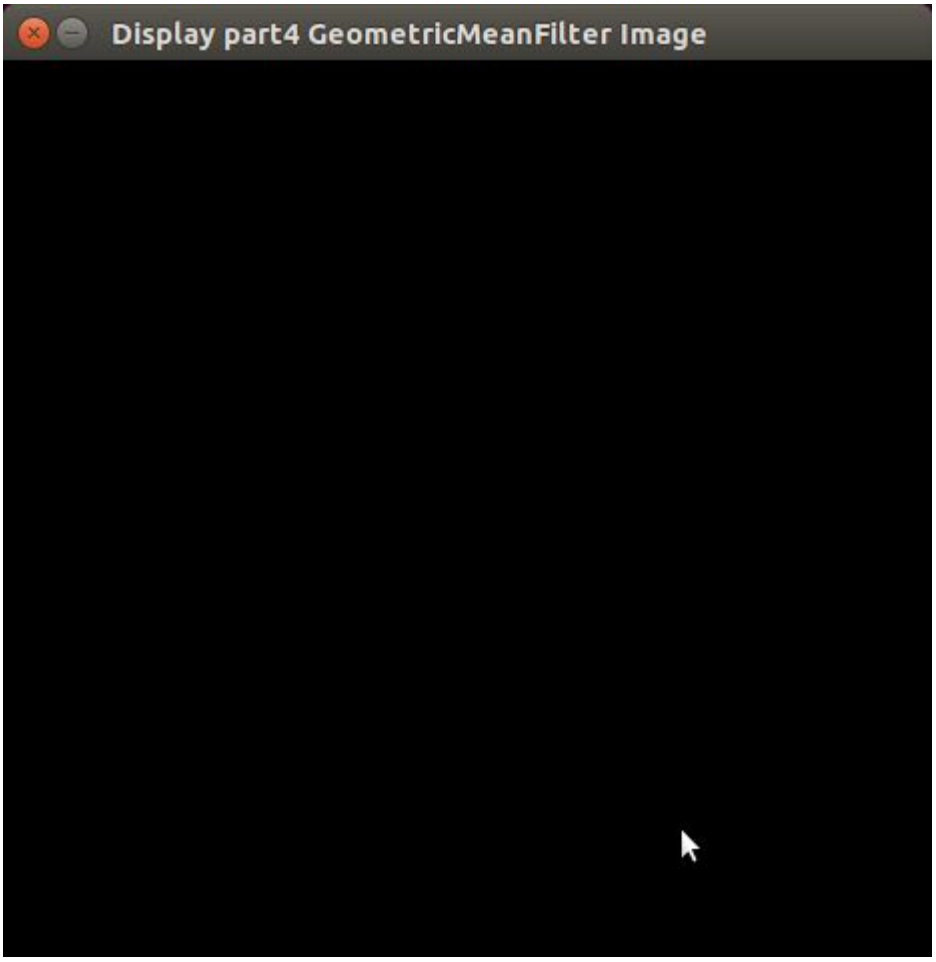
(2) geometric mean filtering

3*3 :



5*5：

9*9 :

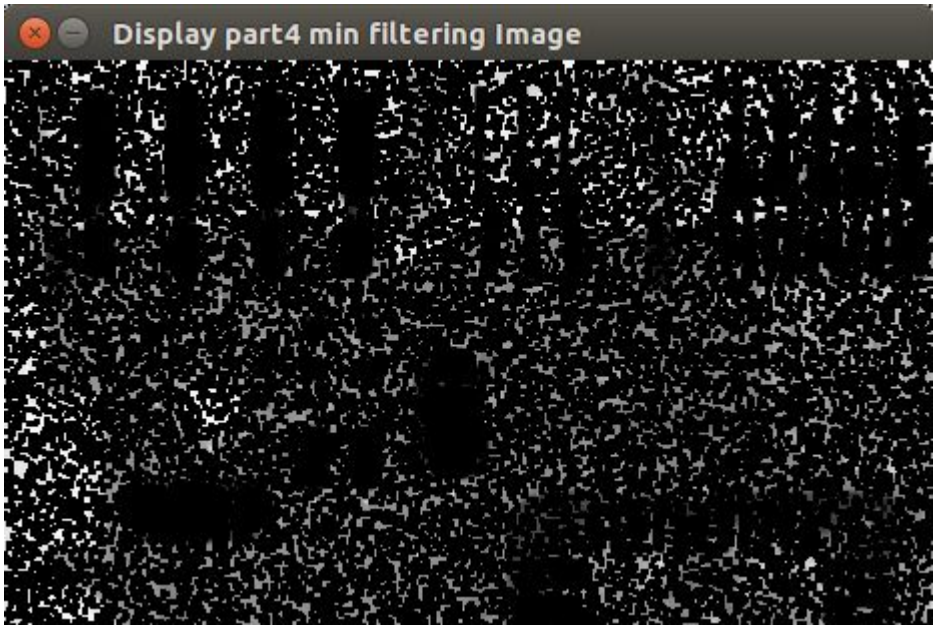


(3) max filtering

3*3 :

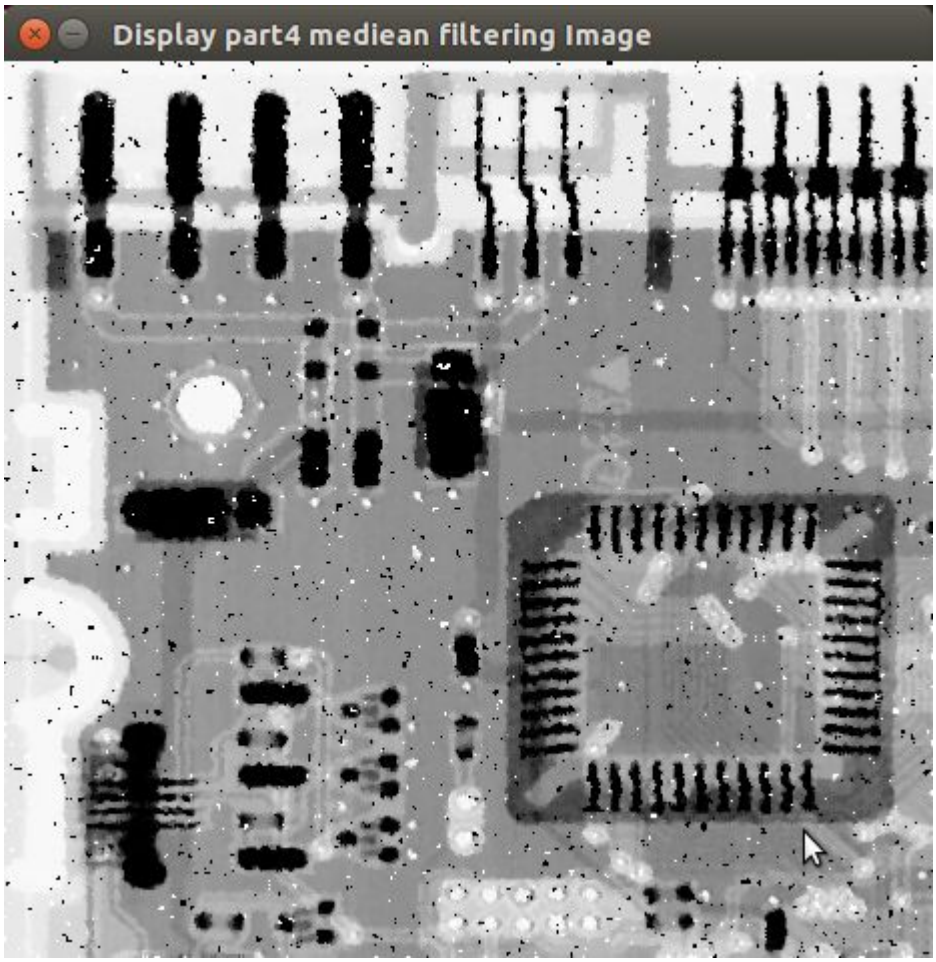
(4) min filtering

3*3 :



（5）median filtering

3*3：



（6）明显，中值滤波的效果远比最值滤波好。椒盐噪声是由于某些原因图像中产生一些灰度值接近 0 和接近 255 的点，这些点的值往往和周围的像素值相差很大，所以任意一个区域内，最大值和最小值很可能就是噪声点，而中值滤波器可以去除区域内极值的影响，故而有比较好的去噪效果。

5. 实现方式：

（1）高斯噪声生成

生成正太分布的随机数：

```
double Random()
```

然后， 把随机生成数乘上 standard 再加上 mean，然后加到原图中去，

输出

```
Mat addGaussian(Mat &input, int mean, int standard)
```

（2）椒盐噪声生成：

输入图像，probability 表示噪声的概率，salt 为 true 表示生成盐噪声，反之生成胡椒噪声。随机取 probability*row*col 个像素设为 255 或者 0.

```
Mat addSaltPepper(Mat &input, double probability, bool salt) {
```

（3）filter1-》算术均值、谐波均值、逆谐波均值（因为写完之后发现算法结构相似，就把它们整合到一起了）其中 q 为逆谐波均值中的 Q 值，size 表示滤波器的大小，geometric 为真表示做几何均值滤波（累乘），反之做逆谐波滤波（累加），逆谐波均值的 Q 值不同是可以分别时谐波均值、算术均值等等。

```
//算术均值、谐波均值、逆谐波均值
Mat filter1(Mat &input, double q, int size, bool geometric) {
```

（4）filter2 -》中值滤波，最大值、最小值滤波（同样也是由于算法结构相似，整合到一起）size 表示滤波器的大小，choice==0 表示最小值滤波，choice==1 表示中值滤波，choice==2 表示最大值滤波。

```
//中值滤波器 最大值、最小值
//mediean -> choice == 1;
//max -> choice == 2;
// min -> choice == 0;
Mat filter2(Mat &input, int size, int choice) {
```

（5）所有的滤波器都要考虑 input 图像的通道数，task_1.png 为灰度图，通道数为一，task_2.png 默认读取通道数为 3，对与不同通道数的图像处理相似，只不过 3 通道需要多考虑一次循环。

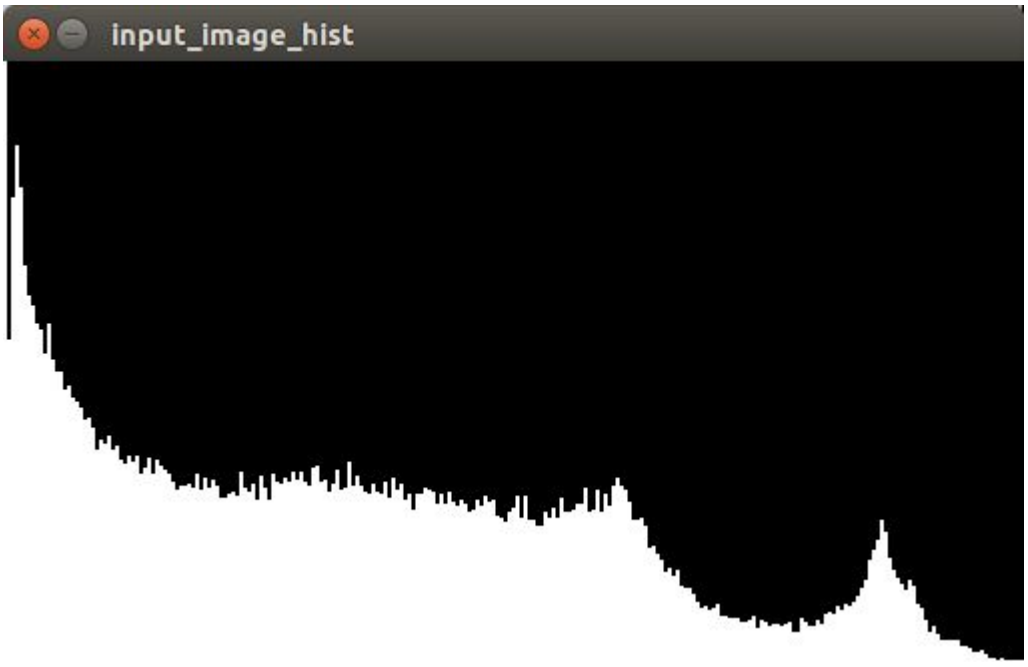
2.4 Histogram Equalization on Color Images

1. Use the function “equalize hist” that you have written in HW2 to process the R, G, B channels separately

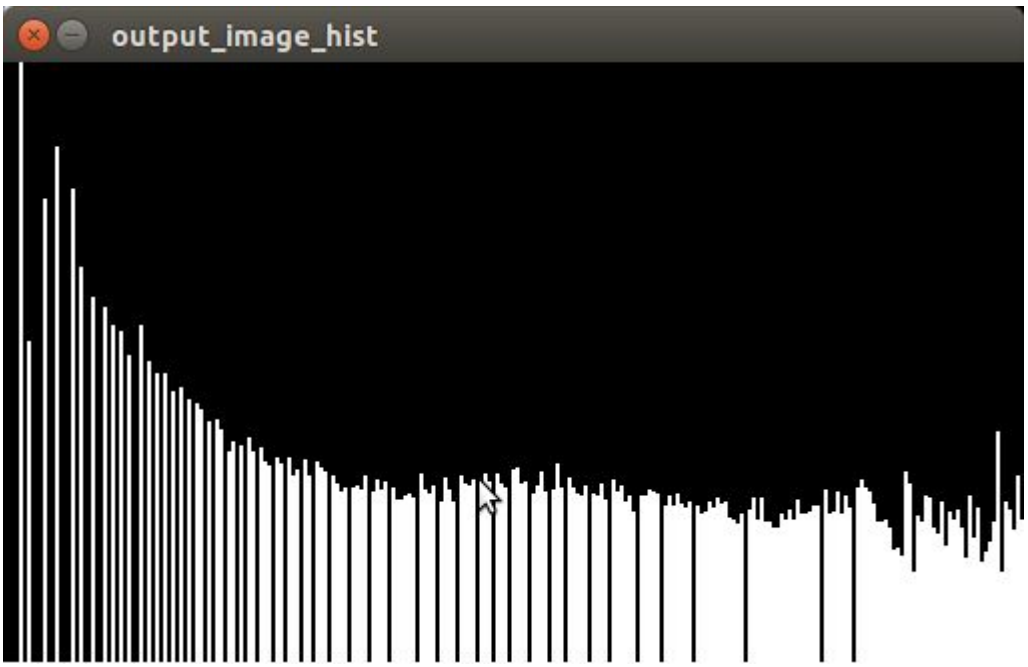
a) 把 R 通道转为单通道灰度图：

i. 灰度图

ii. 原直方图



iii. 均衡化后直方图

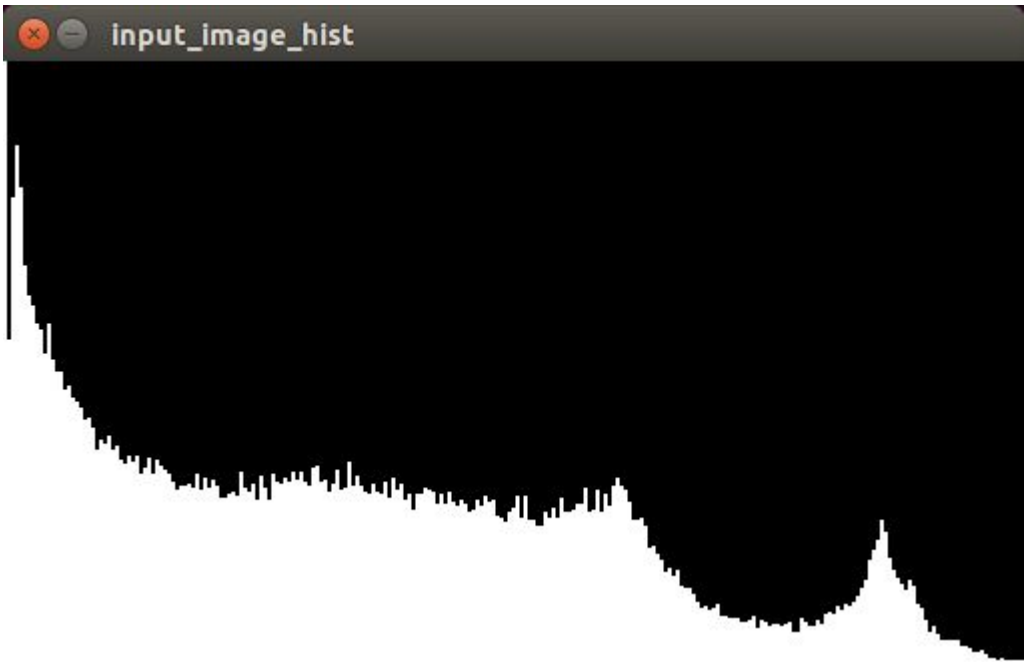


iv. 结果图

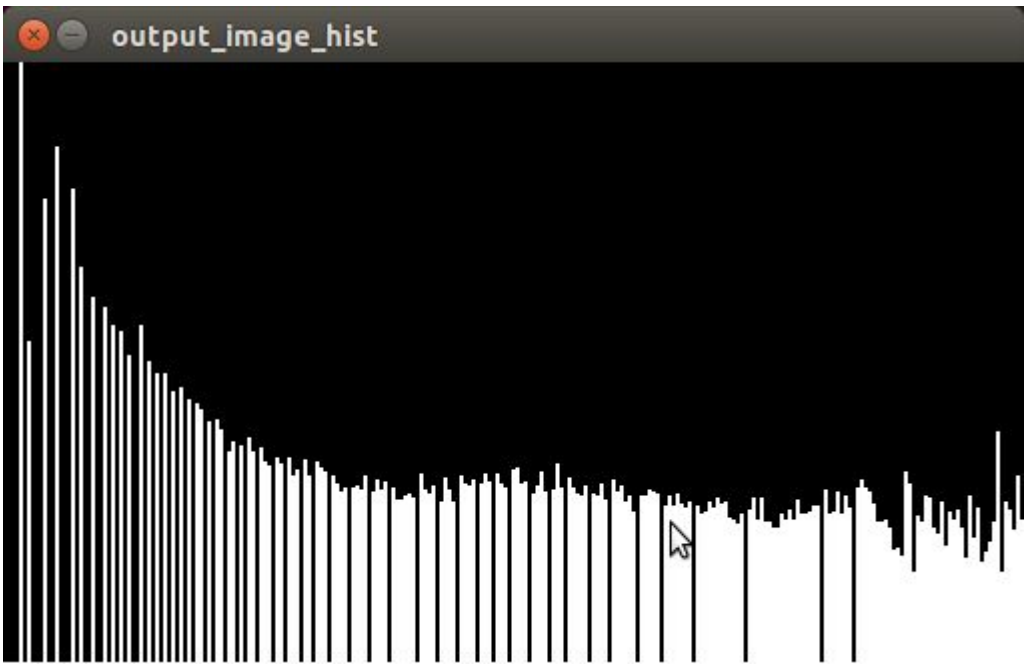
b) 把 G 通道转为单通道灰度图：

i. 灰度图

ii. 原直方图



iii. 均衡化后直方图

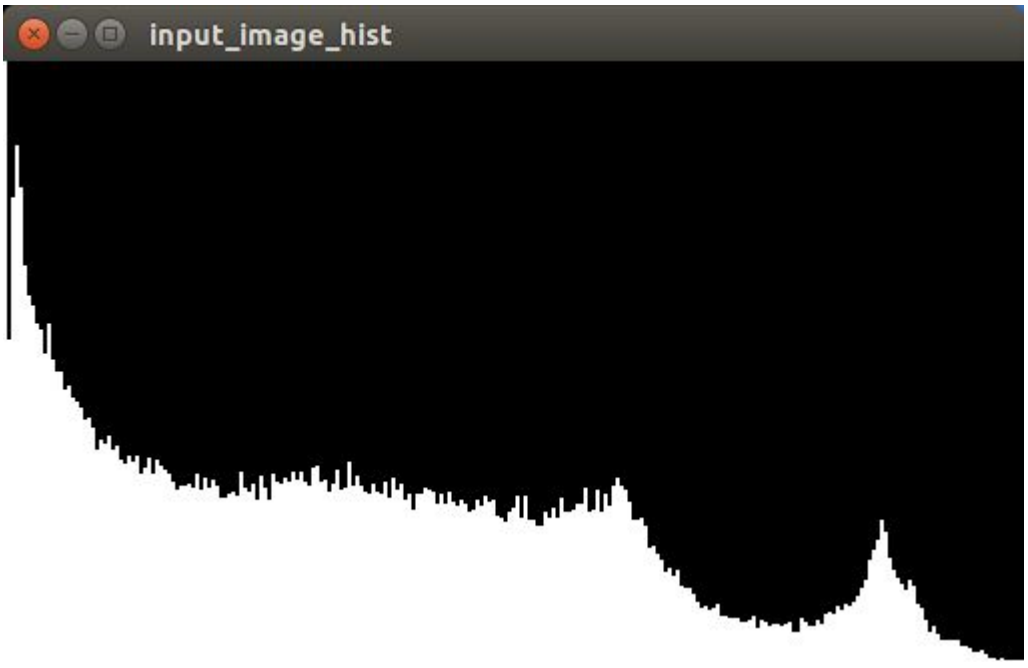


iv. 结果图

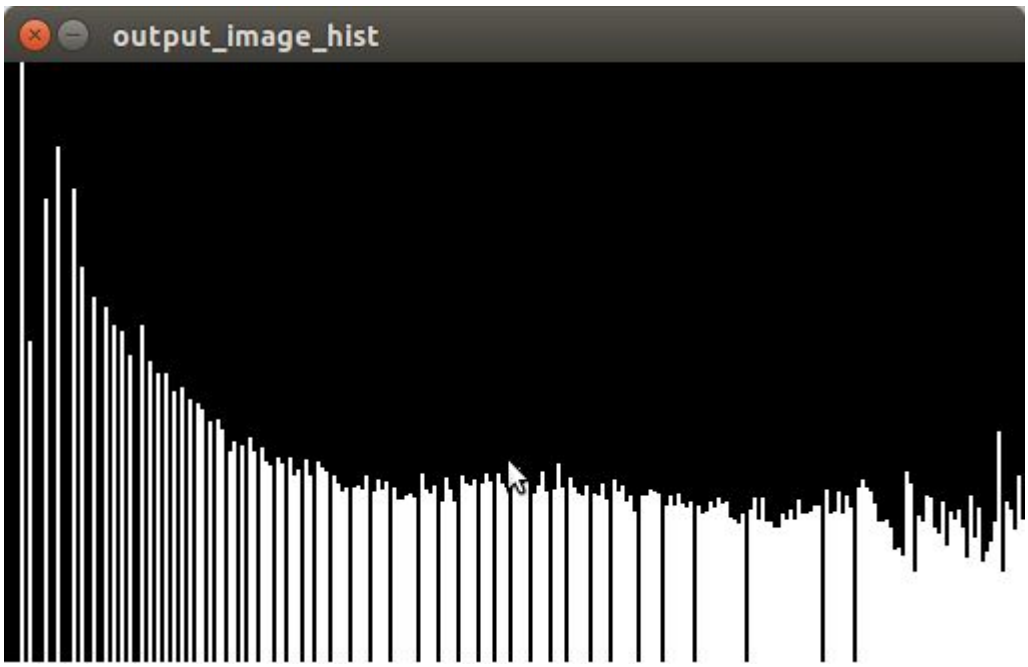
c) 把 B 通道转为单通道灰度图：

i. 灰度图

ii. 原直方图



iii. 均衡化后直方图



iv. 结果图

原图：

处理后的图：

2. Calculate the histogram on each channel separately, and then compute an average histogram from these three histograms.

做法：把 RGB 三通道分开成三幅单通道灰度图，然后分别映射到一个

（R+G+B）/ 3 的直方图上，处理，然后再把结果图合并成三通道图形呈现。

（1）R 处理后灰度图为：

（2）G 处理后灰度图为：

（3）B 处理后灰度图为：

平均图（即强度图）：

最后合成：

3. Compare and explain the differences in the results produced by the above two applications within 1 page

两者比较：

处理过后的图像，直观上感觉差别不大，比较之下 rebuild1 和 rebuild2 的图像相较于原图，色调、饱和度都有改变，而 rebuild1 和 rebuild2 相似。按照公式来理解，（R+G+B）/ 3 事实上就是图像的强度值，而在 HSI 模型中，I 是最接近原图的，它包含了很多原图的细节，而把三个通道的值分别映射到同一强度图上，则是一种增强图像的方法，使得图像三个通道的像素值分布更加均衡，所以最后的结果与三个通道分别进行直方图均衡化的结果相似。