

实验目的

掌握基本滤波算法的实现

实验内容

- ①算术均值滤波
- ②几何均值滤波
- ③逆谐波均值滤波
- ④修正阿尔法均值滤波
- ⑤中值滤波

算法设计

对于算术均值滤波，算法如下，其中 $m=n=3$

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t)$$

对于几何均值滤波，算如下，其中 $m=n=3$

$$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}}$$

对于逆谐波均值滤波，算法如下，其中参数值在下面每张图上说明

$$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{\varphi+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^{\varphi}}$$

对于修正阿尔法均值滤波，算法如下，其中参数值在下面每张图上说明

$$\hat{f}(x, y) = \frac{1}{mn - d} \sum_{(s,t) \in S_{xy}} g_r(s, t)$$

对于中值滤波，算法如下，其中模板大小为3×3

中值滤波根据给出的模板大小，讲模板内的像素值排序，以中间值为标准将模板内所有值改为中心值

代码块

```
# 算数均值滤波
from PIL import Image

def read_image(file_path):
    image = Image.open(file_path).convert("L")
    return image

def save_image(image, file_path):
    image.save(file_path)

def process_image(image, kernel_size):
    width, height = image.size
    filtered_pixels = []

    for y in range(height):
        for x in range(width):
            # 提取区域
            region = []
            # 从kernel_size 的最左边到最右边, +1是控制边界能到最右边 因为range是左闭右开
            for i in range(-kernel_size // 2, kernel_size // 2 + 1):
                for j in range(-kernel_size // 2, kernel_size // 2 + 1):
                    px, py = x + i, y + j
                    # 边界处理
                    if(px >= width or px < 0):
                        px = 0
                    if(py >= height or py < 0):
                        py = 0
                    # 获取像素值
                    region.append(image.getpixel((px, py)))

            # 计算算数均值并更新像素值
            mean_value = sum(region) // len(region)
            filtered_pixels.append(mean_value)

    # 创建新图像
    filtered_image = Image.new("L", (width, height))
    filtered_image.putdata(filtered_pixels)

    return filtered_image

def main():
    dir = "./img3/"
```

```

image_name = "img3.bmp"
input_image_path = dir+image_name
output_image_path = dir+"arithmetic.bmp"
kernel_size = 3

input_image = read_image(input_image_path)

# 应用算数均值滤波器
filtered_image = process_image(input_image, kernel_size)

# 显示原始图像和滤波后的图像
input_image.show(title="Original Image")
filtered_image.show(title="Filtered Image")

# 保存滤波后的图像
save_image(filtered_image, output_image_path)

if __name__ == "__main__":
    main()

```

```

# 几何均值滤波
from PIL import Image
from math import *

def read_image(file_path):
    image = Image.open(file_path).convert("L")
    return image

def save_image(image, file_path):
    image.save(file_path)

def process_image(image, kernel_size):
    width, height = image.size
    filtered_pixels = []

    for y in range(height):
        for x in range(width):
            # 提取区域
            region = []
            # 从kernel_size 的最左边到最右边, +1是控制边界能到最右边 因为range是左闭右开
            for i in range(-kernel_size // 2, kernel_size // 2 + 1):
                for j in range(-kernel_size // 2, kernel_size // 2 + 1):
                    px, py = x + i, y + j
                    # 边界处理
                    if(px>=width or px<0):
                        px=0
                    if(py>=height or py<0):
                        py = 0
                    # 获取像素值
                    region.append(image.getpixel((px, py)))

            # 计算几何均值并更新像素值
            geometric_mean_value = 1
            for pixel_value in region:
                geometric_mean_value *= pixel_value

```

```

        geometric_mean_value = int(geometric_mean_value ** (1 /
len(region)))
        filtered_pixels.append(geometric_mean_value)

# 创建新图像
filtered_image = Image.new("L", (width, height))
filtered_image.putdata(filtered_pixels)

return filtered_image

def main():
    dir = "./img4/"
    image_name = "img4.bmp"
    input_image_path = dir+image_name
    output_image_path = dir+"geometric.bmp"
    kernel_size = 3

    input_image = read_image(input_image_path)

    # 应用几何均值滤波器
    filtered_image = process_image(input_image, kernel_size)

    # 显示原始图像和滤波后的图像
    input_image.show(title="Original Image")
    filtered_image.show(title="Filtered Image")

    # 保存滤波后的图像
    save_image(filtered_image, output_image_path)

if __name__ == "__main__":
    main()

```

```

# 逆谐波均值滤波
from PIL import Image
import math

def read_image(file_path):
    image = Image.open(file_path).convert("L")
    return image

def save_image(image, file_path):
    image.save(file_path)

def process_image(image, kernel_size, Q):
    width, height = image.size
    filtered_pixels = []

    for y in range(height):
        for x in range(width):
            # 提取区域
            region = []
            # 从kernel_size 的最左边到最右边, +1是控制边界能到最右边 因为range是左闭右开
            for i in range(-kernel_size // 2, kernel_size // 2 + 1):
                for j in range(-kernel_size // 2, kernel_size // 2 + 1):
                    px, py = x + i, y + j

```

```

        # 边界处理
        if(px>=width or px<0):
            px=0
        if(py>=height or py<0):
            py = 0
        # 获取像素值
        region.append(image.getpixel((px, py)))

        # 计算逆谐波均值并更新像素值
        up = 0
        down = 0
        for item in region:
            if(Q<0):
                up+=math.pow(1/item,abs(Q+1)) if item !=0 else 0
                down+=math.pow(1/item,abs(Q)) if item !=0 else 0
            else:
                up+=math.pow(item,(Q+1))
                down+=math.pow(item,Q)
        median_value = int(up//down) if down !=0 else 0
        filtered_pixels.append(median_value)

    # 创建新图像
    filtered_image = Image.new("L", (width, height))
    filtered_image.putdata(filtered_pixels)

    return filtered_image

def main():
    kernel_size = 3
    Q = 2
    dir = "./img4/"
    image_name = "img4.bmp"
    input_image_path = dir+image_name
    output_image_path = dir+"contraharmonic_mean_Q="+str(Q)+"_.bmp"

    input_image = read_image(input_image_path)

    # 应用逆谐波均值滤波器
    filtered_image = process_image(input_image, kernel_size,Q)

    # 显示原始图像和滤波后的图像
    input_image.show(title="Original Image")
    filtered_image.show(title="Filtered Image")

    # 保存滤波后的图像
    save_image(filtered_image, output_image_path)

if __name__ == "__main__":
    main()

```

```

# 修正阿尔法均值滤波
from PIL import Image

def read_image(file_path):
    image = Image.open(file_path).convert("L")

```

```

    return image

def save_image(image,file_path):
    image.save(file_path)

def process_image(image,d,alpha):
    width, height = image.size
    filtered_image = Image.new("L", (width,height))

    for x in range(d,width-d):
        for y in range(d,height-d):
            neighbors = []

            for i in range(-d,d+1):
                for j in range(-d, d+1):
                    pixel_value = image.getpixel((x+i,y+i))
                    neighbors.append(pixel_value)

            # 对neighbors内的元素排序
            for item in range(len(neighbors)):
                key = neighbors[item]
                j = item-1
                while j>=0 and key<neighbors[j]:
                    neighbors[j+1] = neighbors[j]
                    j-=1
                neighbors[j+1]=key

            # 去除最大最小值
            neighbors = neighbors[alpha:-alpha]
            mean_value = sum(neighbors)//len(neighbors)
            filtered_image.putpixel((x,y),mean_value)

    return filtered_image

def main():
    d = 3
    alpha = 5
    dir = "./img4/"
    image_name = "img4.bmp"
    input_image_path = dir+image_name
    output_image_path = dir+"alpha_mean_alpha="+str(alpha)+"_d="+str(d)+".bmp"

    input_image = read_image(input_image_path)

    # 应用修正alpha均值滤波器
    filtered_image = process_image(input_image, d,alpha)

    # 显示原始图像和滤波后的图像
    input_image.show(title="Original Image")
    filtered_image.show(title="Filtered Image")

    # 保存滤波后的图像
    save_image(filtered_image, output_image_path)

if __name__ == "__main__":

```

```
main()

# 中值滤波
from PIL import Image

def read_image(file_path):
    image = Image.open(file_path).convert("L")
    return image

def save_image(image, file_path):
    image.save(file_path)

def process_image(image, kernel_size):
    width, height = image.size
    filtered_pixels = []

    for y in range(height):
        for x in range(width):
            # 提取区域
            region = []
            # 从kernel_size的最左边到最右边, +1是控制边界能到最右边, 因为range是左闭右开
            for i in range(-kernel_size // 2, kernel_size // 2 + 1):
                for j in range(-kernel_size // 2, kernel_size // 2 + 1):
                    px, py = x + i, y + j
                    # 边界处理
                    if px >= width or px < 0:
                        px = 0
                    if py >= height or py < 0:
                        py = 0
                    # 获取像素值
                    region.append(image.getpixel((px, py)))

            # 手动实现排序
            for i in range(len(region)):
                for j in range(0, len(region) - i - 1):
                    if region[j] > region[j + 1]:
                        region[j], region[j + 1] = region[j + 1], region[j]

            # 计算中值并更新像素值
            median_value = region[len(region) // 2]
            filtered_pixels.append(median_value)

    # 创建新图像
    filtered_image = Image.new("L", (width, height))
    filtered_image.putdata(filtered_pixels)

    return filtered_image

def main():
    dir = "./img4/"
    image_name = "img4.bmp"
    input_image_path = dir + image_name
    output_image_path = dir + "midvalue.bmp"
    kernel_size = 3
```

```
input_image = read_image(input_image_path)

# 应用中值滤波器
filtered_image = process_image(input_image, kernel_size)

# 显示原始图像和滤波后的图像
input_image.show(title="Original Image")
filtered_image.show(title="Filtered Image")

# 保存滤波后的图像
save_image(filtered_image, output_image_path)

if __name__ == "__main__":
    main()
```

实验结果

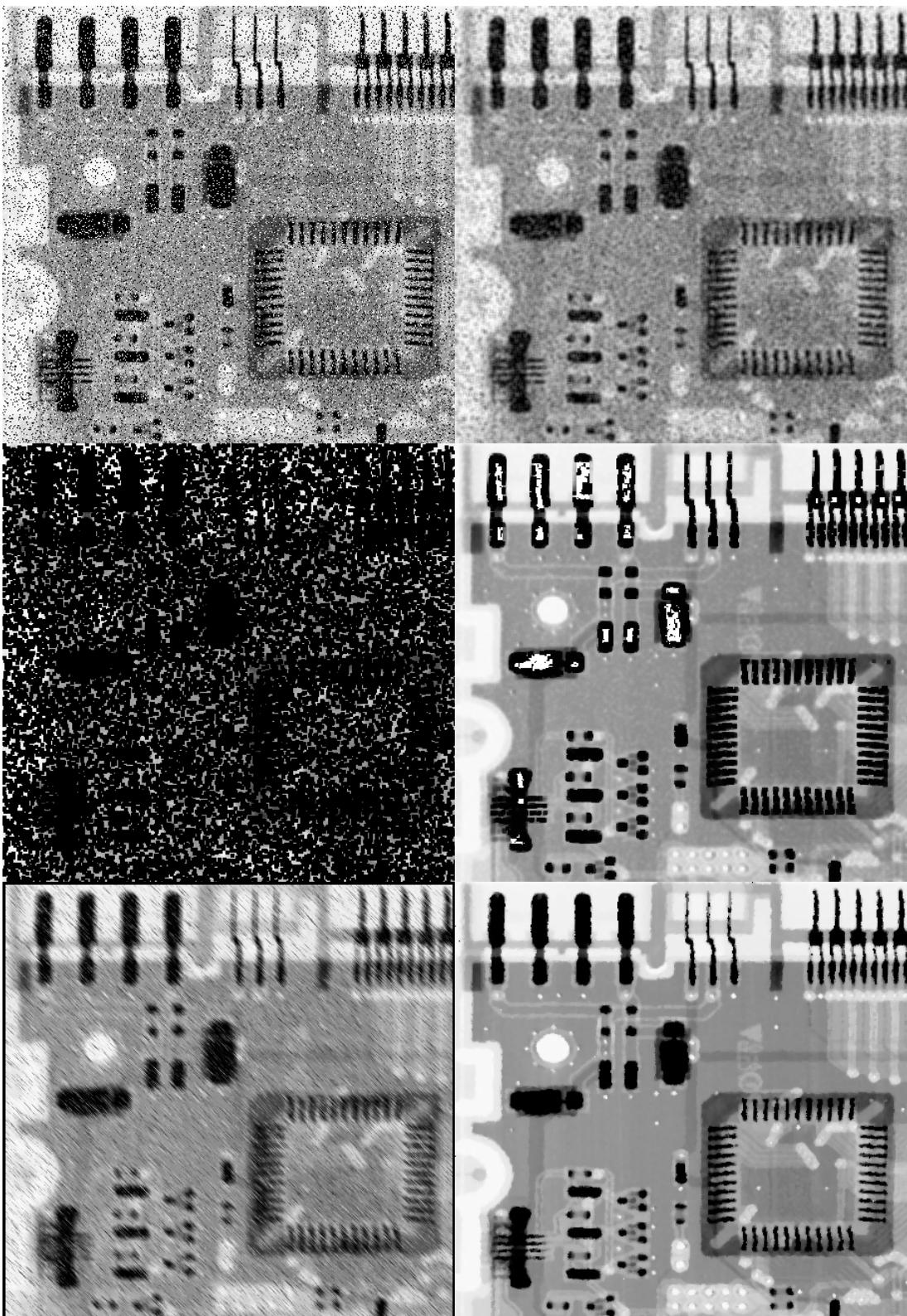
图片顺序：

第一排依次是原图、算数

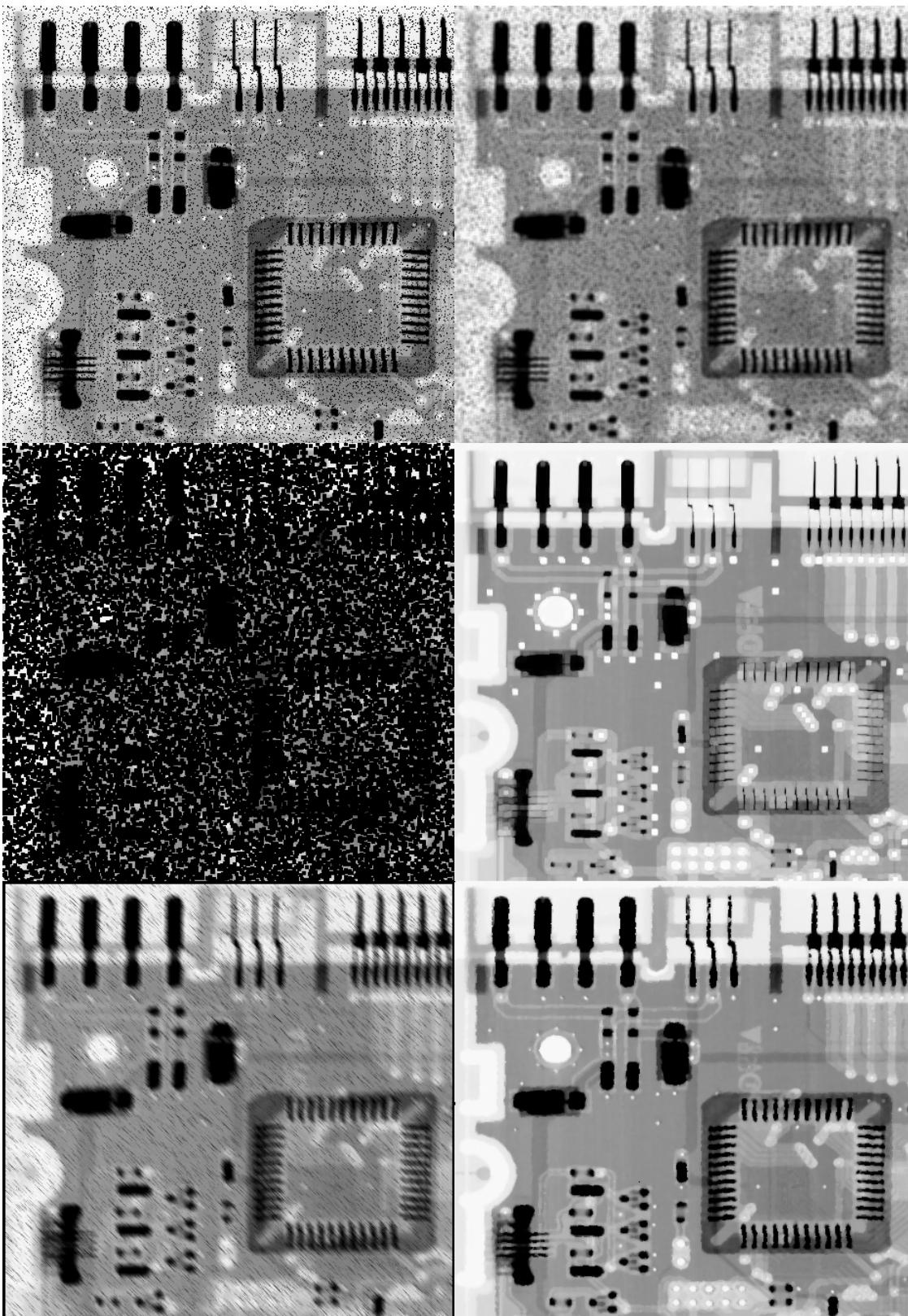
第二排依次是几何、逆谐波

第三排依次是修正阿尔法、中值

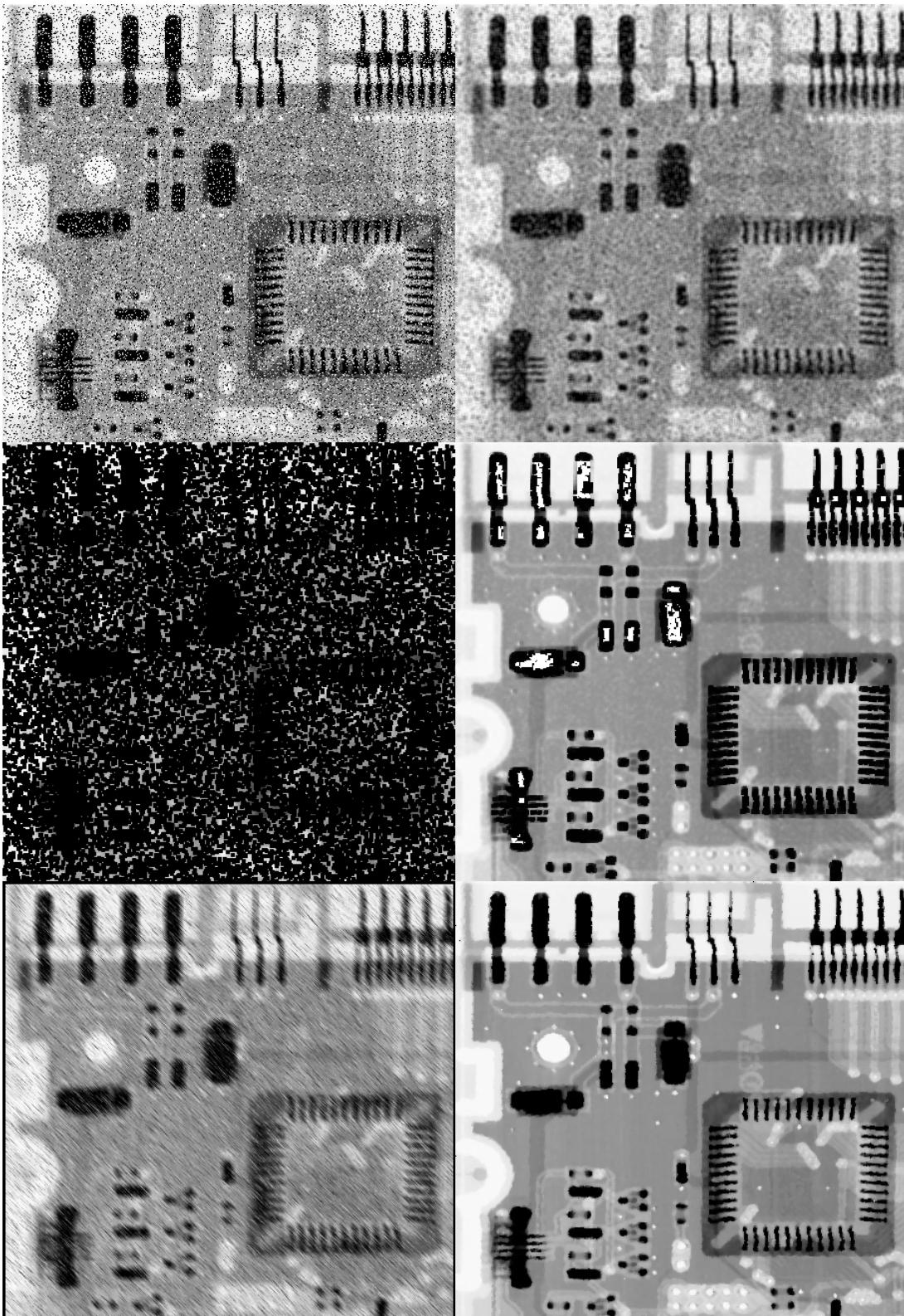
对于图片2-1来说，实验结果如下，其中逆谐波均值滤波 $Q=-2$ ，阿尔法修正中 $d = 3$, $\alpha=5$



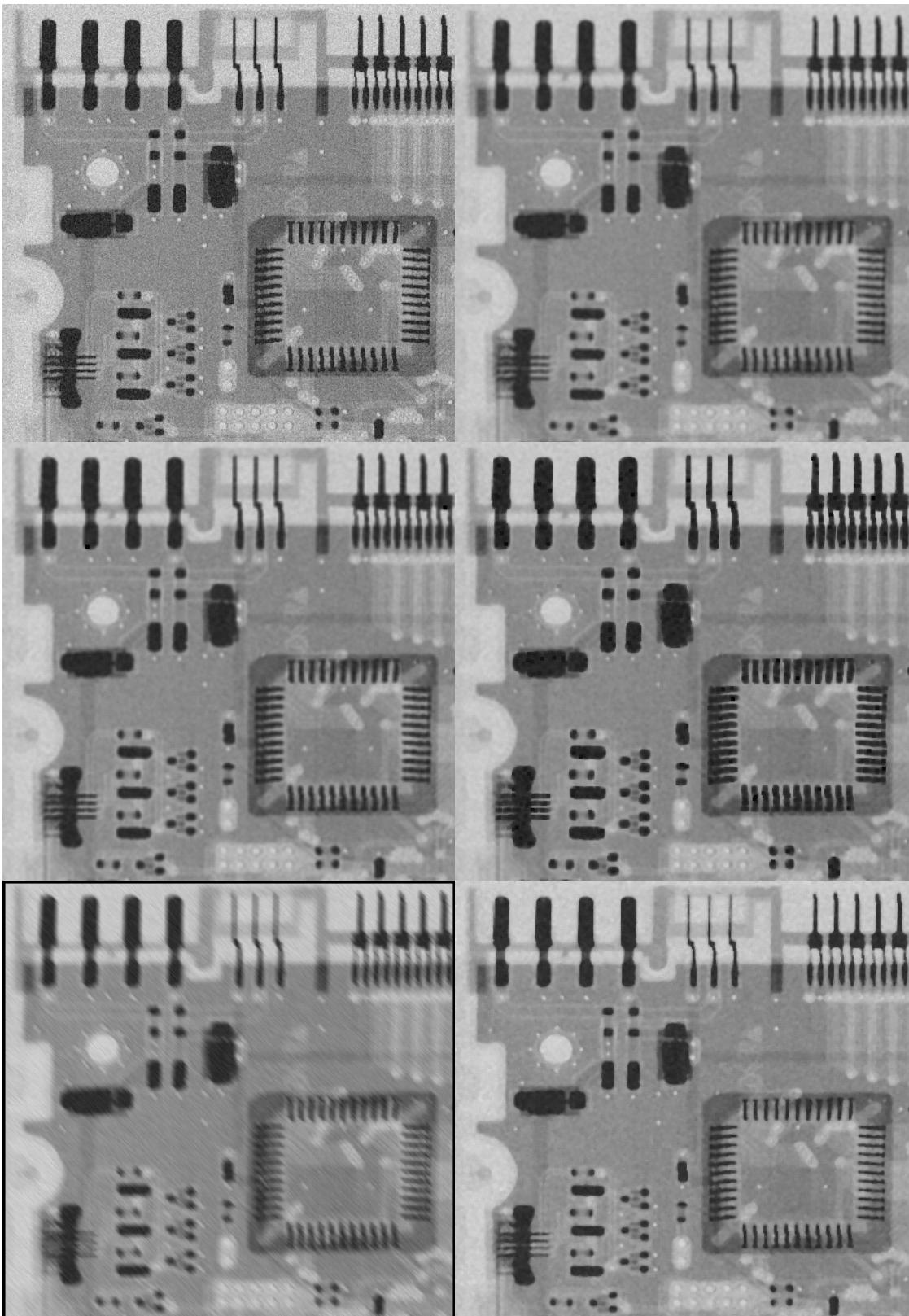
对于图片2-2来说，实验结果如下，其中逆谐波均值滤波 $Q=40$ ，阿尔法修正中 $d=3$, $\alpha=5$



对于图片2-3来说实验结果如下，其中逆谐波均值滤波 $Q=-2$ ，阿尔法修正中 $d=3$, $\alpha=3$



对于图片2-4来说实验结果如下，其中逆谐波均值滤波 $Q=-2$ ，阿尔法修正中 $d=3$ ， $\alpha=5$



实验总结

中值滤波

在本次实验设计到的4个原图中，不论是从清晰度还是噪声去除上，所有图片都是中值滤波比其他所有不同的均值滤波效果要好。因为中值滤波选取一个模块内中间值作为该模块的标准值，而被噪声污染的像素值一般体现为很大或者很小的像素值，所以中值滤波能够完全规避很大或者很小的噪声的影响，而均值滤波虽然是在尽可能规避，但是因为其不抛弃噪声的像素，多少会受到影响。

算数均值滤波

对于算数均值滤波来说，算数均值滤波是通过计算一个卷积核内的算术均值来去除噪声，虽然能够在一定程度上去除噪声的影响，但是这也会将细节信息在算数平均的过程中抹除，尤其是对于边缘信息来说，会使得边缘更加模糊。

几何均值滤波

对于几何均值滤波，先幂次再开放的算法在对普通的卷积核内的像素的影响和算术均值滤波的影响相差无几，但是对于边缘信息来说，因为边缘变化是一个陡峭的变化，一边非常小甚至接近于0，那么结果也是0，这样就会使得边缘信息得以保留。

逆谐波均值滤波

对于逆谐波均值滤波，有一个很重要的参数Q，当Q=0时，等价于算数均值滤波，当Q>0时，对椒噪声有较好的去除效果，所以对于图2-2，选择Q=40的值来作为参数对图片降噪。**当Q=2时，确实得到了较好的结果，但是当Q=4时，结果比Q=2还要好，当我继续尝试增大Q值，比如Q=40，发现效果并没有比Q=4好很多，只是又亮了一些，所以并不值得**

而对于图2-3来说，当Q取负值的效果就比Q取正值的效果好很多，因为当Q取正值的时候对盐噪的去除效果较好，同样**当Q=-2时，确实得到了较好的结果，但是当Q=-4时，结果比Q=2还要好，当我继续尝试增大Q值，比如Q=-40，发现效果并没有比Q=-4好很多，只是又暗了一些，清晰度并没有很大的变化，所以并不值得**

对于图2-4来说，并没有很明显的椒盐噪声，所以降噪的效果并不明显，只是Q>0的时候图像更加亮，而Q<0的时候图像更加暗

修正阿尔法均值滤波

对于所有的修正阿尔法均值滤波，算法中对卷积核内的像素值也做了“掐头去尾”的处理，但是还是对剩下的像素进行了平均求值，这也会将一些被污染程度轻的因素影响最终结果，而且与算数均值滤波一样，还会损失边缘信息，是图片变得模糊。

总的来说，阿尔法修正均值滤波使得图片更加模糊，但是边界比算数均值明显，而且图片像“打了马赛克”