《数字图像处理》课程

实验报告

姓名：

学号：

年级：

专业：

日期：

一、实验目的

结合数字图像处理中所学知识，利用图像滤波、直方图均衡的方法对图像进行处理，减少干扰并提取图像液面边缘。本实验完成了直方图均衡化、图像滤波以及边缘锐化等所有要求。本实验使用 python 实现。

二、实验环境

Python3.9.18，附加库 os、cv2、numpy、matplotlib.pyplot。
实验结果以附件中的保存图表和 Image.ipynb 呈现。

三、实验方法与结果

本实验分为搭建数据结构、读取数据、图像增强（求直方图、累积分布函数 CDF、直方图均衡化、空间滤波）以及边缘锐化提取、结果保存五部分。结果将在每部分末尾展示。

①数据结构

定义了 Image（）类来实现图像数据的统一读取、并行处理。以下是其大致结构。公有变量已在其中特地标出：

```
class Image():
    Tabnine | Edit | Test | Explain | Document
    def __init__(self):
    Tabnine | Edit | Test | Fix | Explain | Document
    def Getpath(self, path):
    Tabnine | Edit | Test | Fix | Explain | Document
    def read(self, path, bins = 256):
        self.path
        self.bins
        self.img
    Tabnine | Edit | Test | Explain | Document
    def Histogram(self):
        self.mat_hist
    Tabnine | Edit | Test | Explain | Document
    def GetCDF(self):
        self.cdf
    Tabnine | Edit | Test | Explain | Document
    def Equalization(self):
    Tabnine | Edit | Test | Fix | Explain | Document
    def filt(self, img, model):
    Tabnine | Edit | Test | Fix | Explain | Document
    def edgeExtract(self, img, t1, t2):
    Tabnine | Edit | Test | Fix | Explain | Document
    def show(self, img):
    Tabnine | Edit | Test | Fix | Explain | Document
    def save(self, img, path):
    Tabnine | Edit | Test | Fix | Explain | Document
    def undo(self):
```

图 1 Image()类的数据结构

②读取数据

__init__()函数实现了类的初始化。当初始化成功时，会打印提示符。

Getpath()函数针对某文件夹下有大批图像文件的情况，可读取该文件夹路径下的所有图像（.jpg）路径，并保存到一个数组中并返回：

```
def Getpath(self, path):
    """
    Get all .jpg files in the given path.
    Return the paths in a list.
    """
    path_list = []
    for root, dirs, files in os.walk(path):
        for file in files:
            if file.lower().endswith('.jpg'):
                path_list.append(os.path.join(root, file))
    return path_list
```

read()函数实现将图像以数组格式读取。必选参数为图像路径，可选参数为图像的灰度值格式（默认不填则为 256 灰度值图像格式）。另外，该函数提供了 self.path、self.bins、self.img、self.undo 等在类外也可调用的公有变量（public variables），可以方便查看图像的路径、像素值、灰度范围等基本信息。详细实现如下：

```
def read(self, path, bins = 256):
    """
    Read image into a matrix.
    self.path: path of the image, public variable.
    self.bins: number of bins for histogram, 256 defaultly. Public variable.
    self.img: matrix format of the image, public variable.
    self.undo_img: get the previous image.
    """
    self.path = path
    self.bins = bins
    self.img = cv2.imread(path, 0)
    self.undo_img = self.img
    if self.img.all() != None:
        print('Read succeeded!\n')
```

实现部分，利用该函数实现了源图像文件夹 data 下全部六张图像路径的一次性提取，并定义了一个 Image()类数组，每个元素为一个 Image()对象，分别存储其中一个图像，并针对该图像自身提供了诸多处理操作，如获取灰度直方图、获取累积分布函数、直方图均衡化、空间滤波、边缘锐化提取、撤销操作、窗口显示等，极大方便了图像的批量操作。

以下是该部分的应用实例，包含了类列表的初始化、路径批量获取以及图片批量导入。运行结果来自附件中的 Image.ipynb：

```python
lists = Getpath('data')
a = [Image() for i in range(6)]
for i in range(6):
    a[i].read(lists[i])
```

Python

```
Initialized successfully!

Initialized successfully!

Initialized successfully!

Initialized successfully!

Initialized successfully!

Initialized successfully!

Read succeeded!

Read succeeded!

Read succeeded!

Read succeeded!

Read succeeded!

Read succeeded!
```

图 2 图像的批量读取

③图像增强

Histogram()函数实现了对对象内存储图像的灰度直方图的建构。该函数分为两部分，第一部分是对灰度直方图矩阵的获取（256 长度，每个元素对应该灰度值下像素的数目），第二部分是绘制灰度直方图。第一部分获取的直方图矩阵，存储到了一个公有变量 self.mat_hist 中。

```python
def Histogram(self):
    """
    Plot histogram of the image.
    self.mat_hist: matrix format of the histogram, public variable.
    """
    self.mat_hist, bins = np.histogram(self.img.ravel(), self.bins, [0,self.bins])
    hist = plt.hist(self.img.ravel(), bins = self.bins)
```

GetCDF()函数获取该图像的累积分布函数 CDF，并存储到公有变量 self.cdf 中：

```python
def GetCDF(self):
    """
    CDF is the cumulative distribution function of an image.
    Returns an array.
    """
    area = multiply(self.img)
    self.mat_hist, bins = np.histogram(self.img.ravel(), self.bins, [0,self.bins])
```

```
        self.cdf = [0] * self.bins
        for s in range(self.bins):
            self.cdf[s] = np.sum(self.mat_hist[:s+1]) / area
        return self.cdf
```

Equalization()函数是基于上述两个函数搭建的，实现了图像灰度直方图的标准化。与 Histogram()函数一样，该函数也有两部分：第一部分存储了标准化后的图像矩阵，第二部分展示了均衡化前后图像的对比，显示在窗格里：

```
    def Equalization(self):
        """
        Equalize the image and show them contrarily.
        self.res: the equalized image of matrix format. Public variable.
        self.cdf_norm: the normalized CDF. Only for plotting.
        """
        cdf = self.GetCDF()
        cdf_norm = cdf
        for i in range(self.bins):
            cdf_norm[i] = cdf[i] * max(self.mat_hist)

        equ = cv2.equalizeHist(self.img)
        self.res = np.hstack((self.img, equ))

        plt.plot(cdf_norm, color = 'b')
        plt.hist(self.res.flatten(), 256, [0, 256], color = 'r')
        plt.xlim([0,256])
        plt.legend(('CDF','Equalized histogram'), loc = 'upper left')
        plt.show()

        cv2.imshow('img',self.res)
        cv2.waitKey()
        cv2.destroyAllWindows()
```

filt()函数根据输入参数 model 取值不同，采取不同的空间滤波方式，对输入图像（不仅局限于原图像）进行不同角度的增强。可选的方式包括高斯滤波、均值滤波、中值滤波、拉普拉斯锐化、Sobel 锐化、Canny 锐化以及傅里叶变换求频谱。其中，高斯滤波作为最常用的滤波方式，被设定为默认滤波。该函数包括诸多可选参数，如作用域 d、高斯滤波参数 sigma、Canny 锐化阈值 t1、t2、Sobel 锐化的阶数 dx、dy 等等，根据滤波方式的不同，输入不同的参数而不发生冲突：

```
    def filt(self, img, model = 'gaussian', d = 3, sigma=2, t1 = 100, t2 = 200, dx = 0, dy = 0):
        """
        Apply different filters to the given image.
        Gaussian filter defaultly.
        img: matrix format, necessary parameters.
        model: filter model, string format, 'gaussian', 'laplacian', 'canny', 'average', 'median', 'sobel',
'fourier'.
```

```
    d: an even int. Range of the filter.
    sigma: standard deviation only for Gaussian filter.
    t1, t2: thresholds for Canny filter. t1 < t2.
    dx, dy: parameters for Sobel filter. Rate of the filter.
    """
    if model == 'gaussian':
        flt = cv2.GaussianBlur(img, (d, d), sigmaX=sigma, sigmaY=sigma)
    elif model == 'laplacian':
        flt = cv2.Laplacian(img, cv2.CV_64F)
        flt = cv2.convertScaleAbs(flt)
    elif model == 'canny':
        flt = cv2.Canny(img, threshold1 = t1, threshold2 = t2)
    elif model == 'average':
        flt = cv2.blur(img, (d, d))
    elif model =='median':
        flt = cv2.medianBlur(img, d)
    elif model == 'sobel':
        if dx * dy :
            flt = cv2.Sobel(img, cv2.CV_64F, dx, dy, ksize=d)
        else:
            print('Error! No parameters given to dx or dy! Set dx = dy = 1 defaultly!')
            sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0)
            sobelx = cv2.convertScaleAbs(sobelx)

            sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1)
            sobely = cv2.convertScaleAbs(sobely)
            flt = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)
    elif model == 'fourier':
        dft = np.fft.fft2(img)
        fshift = np.fft.fftshift(dft)
        flt = np.log(np.abs(fshift))
    return flt
```

针对 Sobel 滤波设置了空参数提醒。当未设置参数时，使用默认参数而不报错致使程序中断。图像增强部分的应用实例如下：

（1）直方图均衡化部分

实现了六张图的直方图均衡化并作图：

```
for i in range(6):
    a[i].Equalization()
```

Python

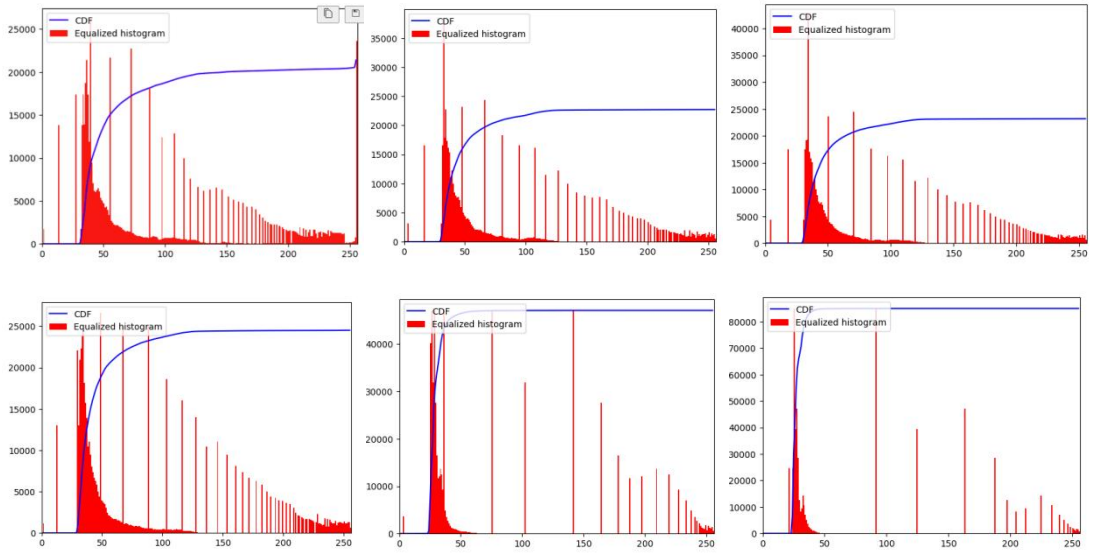


图 3 均衡化后的 CDF 和直方图

应特别指出，上面图中的 CDF 曲线是经过缩放处理的。否则 CDF 曲线最大值不超过 1，与均衡化后的直方图共同作图时走势不明显。

均衡化前后的图像对比：

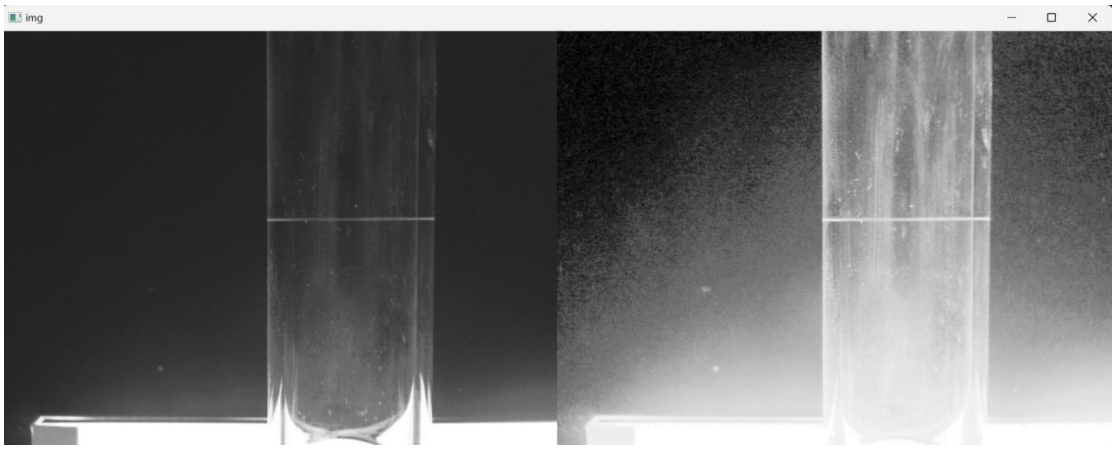
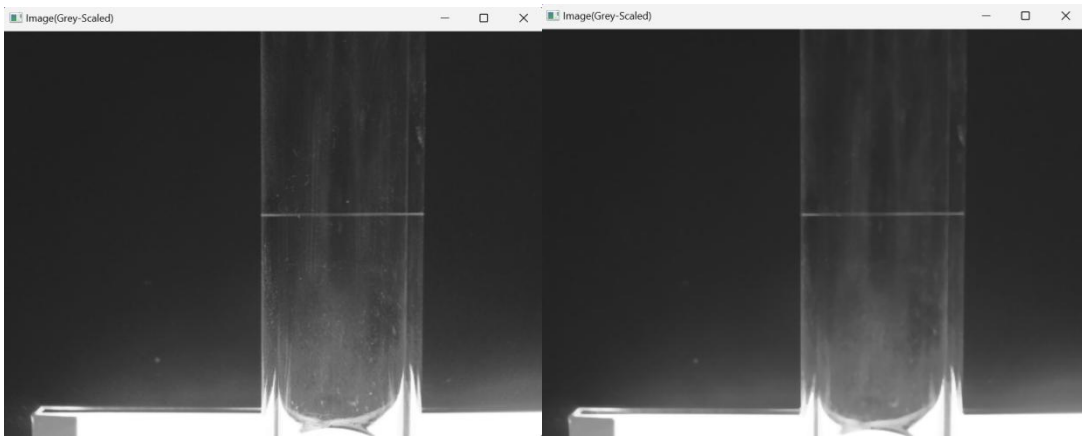


图 4 左：原图； 右：均衡化后图像

（2）**滤波部分**

以下是利用各种滤波器对图 img1524.png 进行的试验：



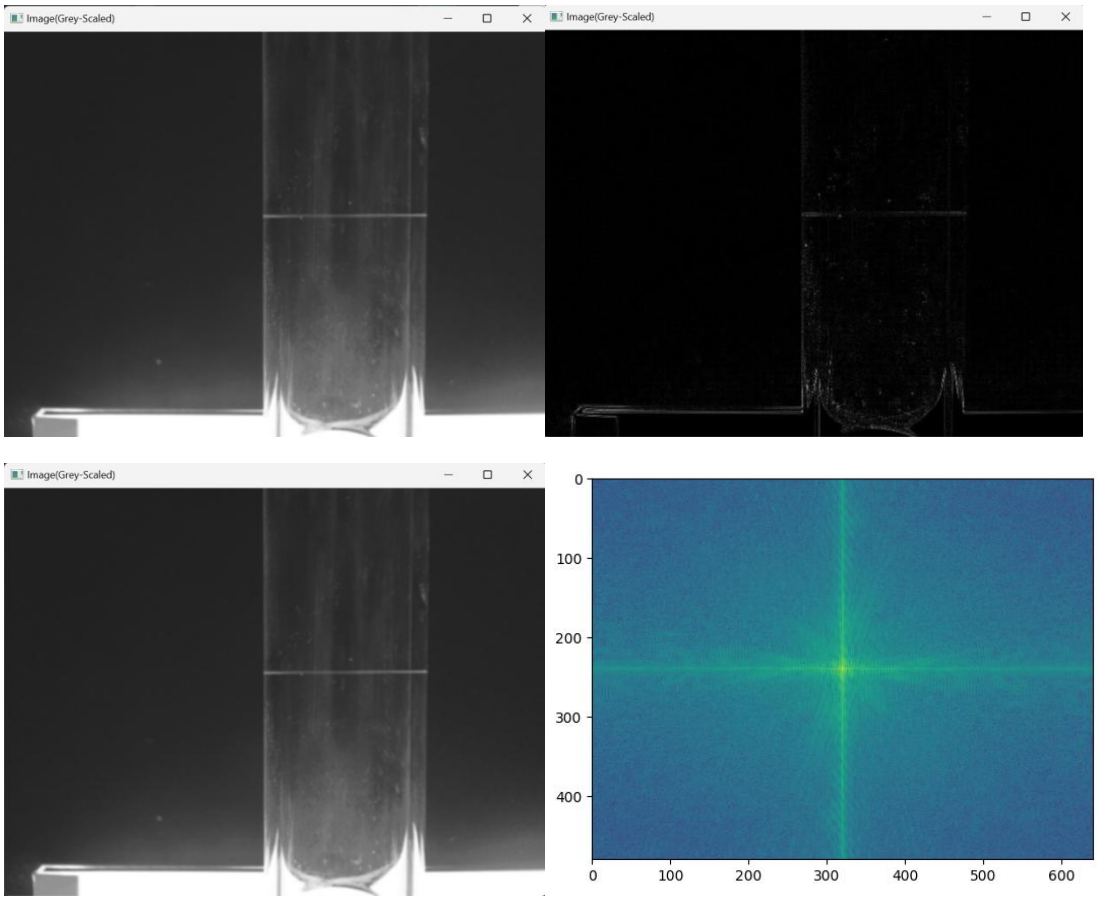


图 5：上下、左右依次为原图、高斯、中值、拉普拉斯、均值、频谱结果

对各种参数取值的试验发现，滤波处理对后续边缘提取的准确度影响极大。一方面，亮光条件下试管壁上出现小液滴痕迹（类似于椒盐噪声）。这类亮光条件下拍摄的照片另外使用中值滤波进行特殊处理效果较好（如下图）：

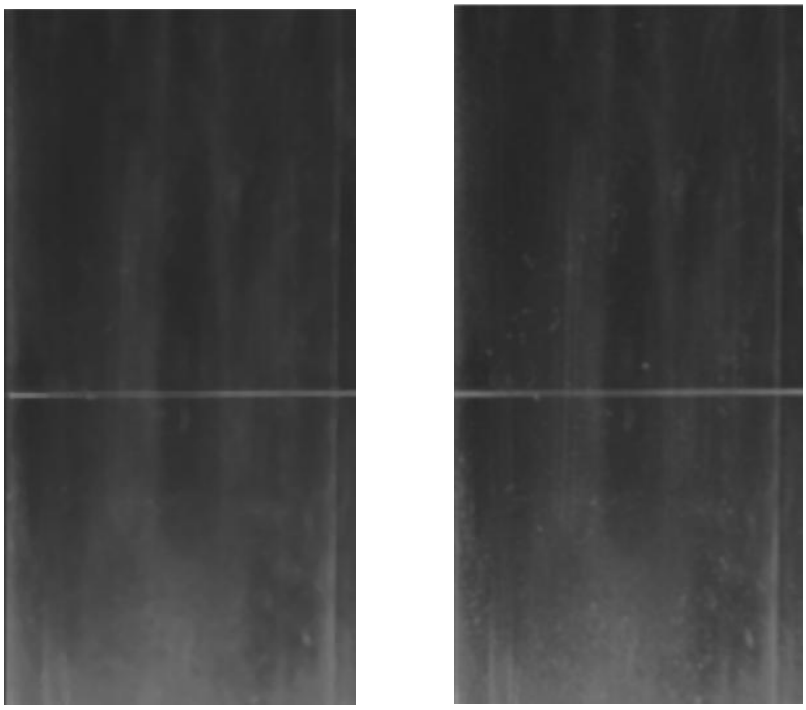


图 6 左：高斯+中值 右：仅高斯

另一方面，对暗光下拍摄的图像，液面与周围环境对比不明显，直方图均衡化后对比度增强亦不明显，应使用高斯滤波将液面与两侧黑暗尽可能区分开：

```python
filted, edges = [], []
for i in range(6):
    filted.append(a[i].filt(a[i].img, 'gaussian', d = 5, sigma = 2.5))
    edges.append(a[i].edgeExtract(filted[i], 10, 50))
```

针对后续的边缘提取，经过多次试验，得出以下结论：

1. 为将暗光条件下的边界清晰提取出来，需采取 sigma 较小的高斯滤波器进行处理。然而，在处理亮暗混合的图像集时，这样的滤波器同时也会使亮光条件下的图像产生纹理样的噪点（如下图 7）。因而，将 sigma 值适当放大，并取大一些的滤波核，采用 sigma = 2.5, d = 5 的高斯滤波器进行滤波。

2. 若将亮暗条件下拍摄的图片分开进行处理，则可考虑选取更小的 sigma，因为 sigma 的值与暗光下提取的准确度相关性较大（如下图 8）。

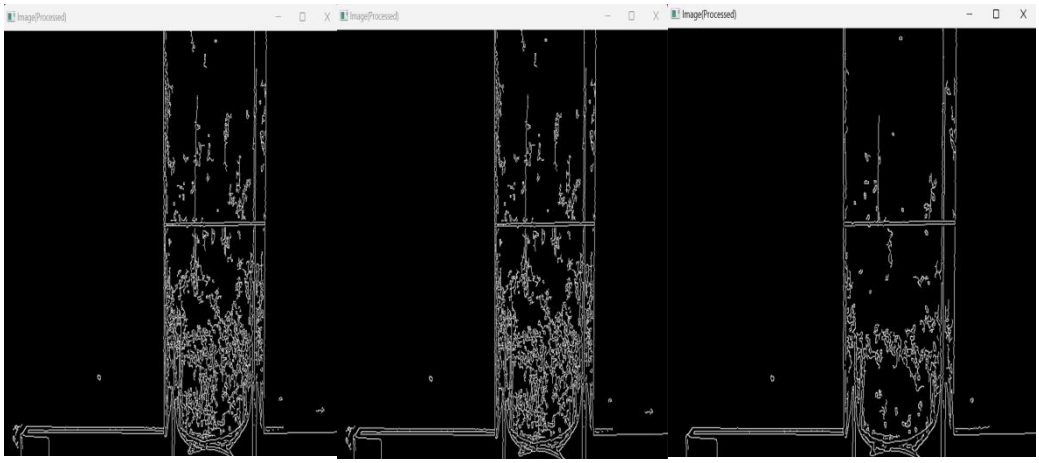


图 7 分别来自(s, d) = (3, 0.5), (5, 0.5), (3, 10)，对亮光下图的处理

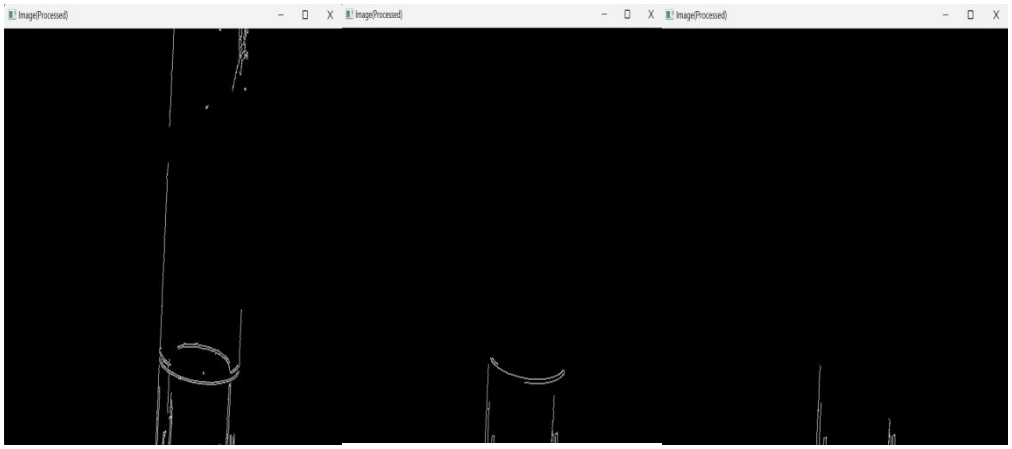


图 8 分别来自(s, d) = (5, 0.5), (5, 5), (5, 10)，对暗光下图的处理

当 sigma 极小（sigma = 0.25）时，会出现类似于亮光条件下的纹理样噪音（如下图 9）：针对暗光处理取 sigma 极小时出现的纹理样的噪声，可以使用滤波核较小（d = 3）的中值滤波，明显改善出现的此类噪声，如下图。

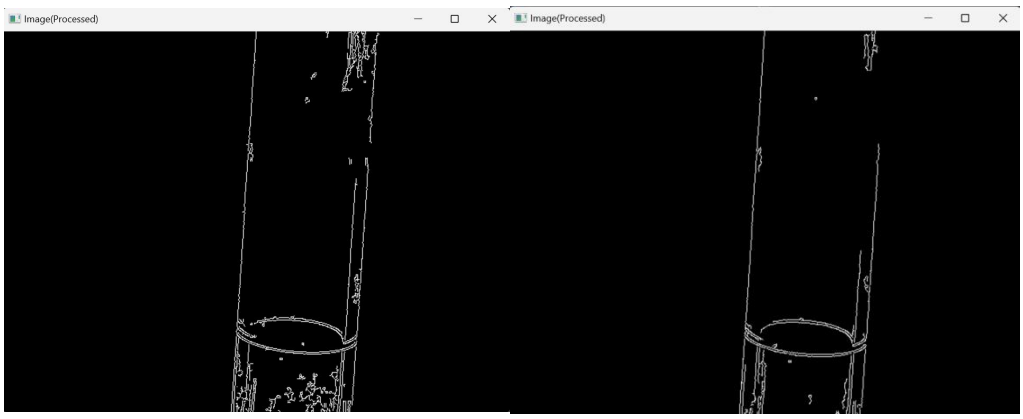


图 9 左：高斯滤波，右：中值+高斯滤波，对暗光图像处理，sigma 取极小（0.25）

即，这几张图片滤波的原则是：

（1）亮/暗拍摄的图像尽量分开处理；

（2）亮光下拍摄的图片使用中值滤波+高斯滤波的混合处理；

（3）暗光下拍摄的图片使用较小 sigma（2.5 以下）的高斯滤波处理，d 取值不重要；当使用极小 sigma（0.5 以下）时，应额外使用中值滤波去纹理噪声；

（4）若亮暗图片混合处理，则用较小 sigma（1 至 2.5）、较大 d（d = 5）处理，其效果不如亮暗分开处理。

④边缘锐化和结果保存

采取 t1 = 10, t2 = 50 对滤波后的图像进行边缘锐化和提取。

另外定义了几个方便显示、保存的函数：

```
def edgeExtract(self, img, threshold1 = 10, threshold2 = 50):
    """
    Normally works after filtering. Extract the edge from image.
    t1, t2: thresholds for Canny filter. t1 < t2.
    Smaller parameters help to extract more detailed edges.
    """
    edges = cv2.Canny(img, threshold1, threshold2)
    return edges


def show(self, img):
    "Show image in a window."
    cv2.imshow('Image(Grey-Scaled)', img)
    cv2.waitKey()
    cv2.destroyAllWindows()


def save(self, img, path):
    "Save the processed image into the given path."
    cv2.imwrite(path, img)


def undo(self):
    "Undo the last operation."
    return self.undo_img
```

提取后的结果如下：

图 10 边缘提取结果

可见液面轮廓均被完整提取出来。以上图片均保存在附件“提取的边缘”文件夹下。

四、问题与反思

（1）数据结构可以进一步简化。

处理大批量文件时，使用对象数组无疑带来了诸多便利。然而在进行滤波后图像的嵌套处理时，代码就变得非常繁琐：

```
filted, edges = [], []
for i in range(4):
    filted.append(a[i].filt(a[i].filt(a[i].img, 'median', d = 5), 'gaussian', d = 5, sigma = 2))
    show(filted[i])
```

这是对 4 张照片做批量处理，先使用中值滤波后使用高斯滤波的操作。可以针对这些批量、嵌套处理图像的操作，增加更简洁的表达。

（2）边缘提取后增加裁剪标记

提取边缘后得到的图像是整张图像。可以考虑增加自动识别椭圆部分然后裁剪图像的类内函数，得到感兴趣的液面区域。

（3）对现成结论作应用。

通过试验知道，对于亮暗光条件下拍摄的图像，最好应该分开处理。以此为根据，引入 self.brightness 属性，以区分输入图像的亮暗环境，据此分开处理。

（4）需要更多数据完善结论。

实验结论是根据六张图片得出的。需要更多的样本来验证结论。

五、附件：

电子版压缩包。含 output 文件夹（内有子文件夹：例子图标（各图像直方图）、滤波后图像（img1524.jpg 施加各种增强处理的效果）、提取的边缘（六张图片经处理后提取的边缘））、data 文件夹（原图像）、实验报告 pdf、源代码 Image.py、Notebook 运行结果 Image.ipynb 以及运行脚本 script.txt。