区域生长法的随机种子设定及优化方法

区域增长法是一种基于像素相似性的图像分割方法，其基本思想是从种子点开始，将与种子点相邻的像素逐个加入到同一区域中，直到所有像素都被分配到某个区域为止。这种方法适用于具有相似颜色或纹理的区域。

一、实现步骤

使用区域增长法进行图像分割的具体步骤如下：

- 选择种子点：首先需要选择一个或多个**种子点**，作为区域增长的起点。
- 确定相似性准则：在区域增长过程中，需要确定像素之间的相似性准则，以便将相似的像素加入到同一区域中。常用的相似性准则包括**像素灰度值之差**、像素颜色之差、像素纹理特征等。

  基于像素灰度值差异的绝对值判断两个像素点是否相似。当两个像素点的灰度值差异小于设定的阈值时，认为这两个像素点相似；当灰度值差异大于等于阈值时，则认为这两个像素点不相似。
- 确定生长准则：在区域增长过程中，需要确定像素加入区域的生长准则，以便控制区域的生长方向和速度。常用的生长准则包括像素与区域的相似性、像素与种子点的距离、像素与区域边界的距离等。

  从种子点开始，将与种子点相似的像素点合并到一个区域中，直到该区域中的像素点都与种子点相似为止。具体来说，将种子点标记为前景区域，并将其加入队列中。然后，遍历队列中的像素点，对于每个像素点，遍历它周围的8个像素点，计算当前像素点与种子点之间的灰度值差异。如果灰度值差异小于阈值并且该像素点未被标记为前景区域，则将其标记为前景区域，并将其加入队列中，以便下一次迭代遍历。重复此过程，直到队列为空。
- 区域增长：从种子点开始**遍历整张图**，按照相似性准则和生长准则，逐个将与当前区域相邻的像素加入到同一区域中，直到所有像素都被分配到某个区域为止。
- 后处理：对分割结果进行后处理，包括**去噪**、填补空洞、分割边界平滑等。

二、区域生长处理代码及结果

OpenCV

OpenCV（Open Source Computer Vision Library，开源计算机视觉库）是一个开源的计算机视觉和机器学习软件库，它主要用于实时图像处理、计算机视觉、目标识别、人脸识别、机器学习等领域。

OpenCV最初是由英特尔公司开发，现在已成为一个由全球贡献者共同维护的开源项目。它支持多种编程语言，包括C++、Python、Java、MATLAB等，并提供了丰富的图像处理和计算机视觉算法库。

OpenCV提供了许多图像处理和计算机视觉算法，例如：

- 图像读取、显示、保存等基本操作
- 图像预处理、滤波、形态学操作、边缘检测等图像处理操作
- 特征提取和描述、特征匹配、目标检测和跟踪等计算机视觉操作
- 机器学习算法库，例如支持向量机、随机森林等

- 深度学习算法库，例如卷积神经网络、循环神经网络等

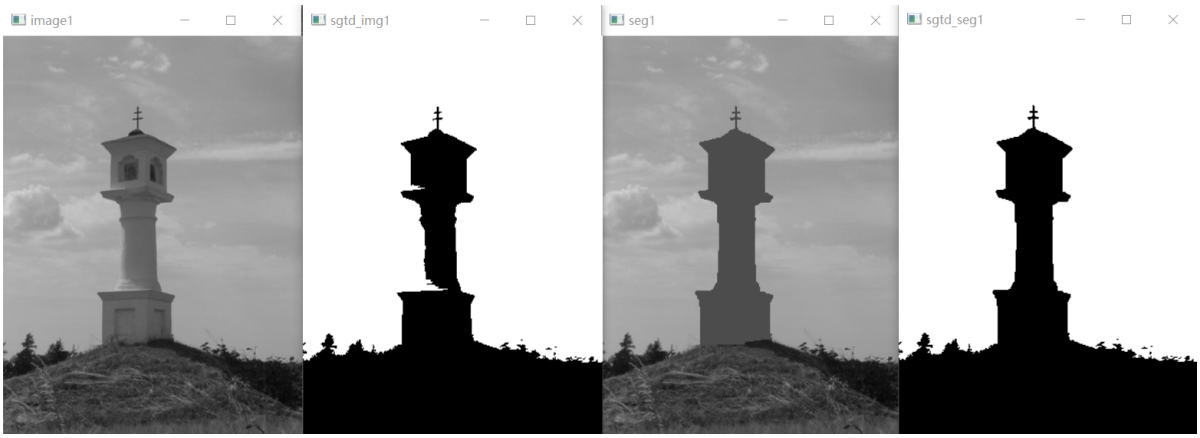
首先是区域增长的自编函数代码，其中针对不同的图片可以调整的参数是 **threshold 阈值**。

```
import cv2
import numpy as np

def region_growing(img, seed): # 分割函数需要的参数是图像数组和种子点位置
    # 阈值设定，根据图像决定
    threshold = 66
    # 获取种子点的像素值
    seed_value = img[seed[0], seed[1]]
    # 获取图像大小（行数和列数）
    rows, cols = img.shape[:2]
    # 根据图像大小创建一个大小相同的全零数组，用来记录已经被分割的像素点
    segmented = np.zeros_like(img)
    # 将种子点标记为255作为前景区域，同时表示已经被分割
    segmented[seed[0], seed[1]] = 255
    # 创建一个队列，用于储存未分割的像素点
    queue = []
    # 将种子点加入队列
    queue.append(seed)

    # 遍历队列中的像素点
    while queue:
        # 取出队头的像素点
        current_point = queue.pop(0)
        # 遍历种子点周围的8个像素点
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i == 0 and j == 0:   # 跳过当前点
                    continue
                # 检查周围8个像素点是否超出图像边界，如果超出则跳过当前点，不再继续分割
                if current_point[0] + i < 0 or current_point[0] + i >= rows or current_point[1] + j < 0 or \
                        current_point[1] + j >= cols:
                    continue
                # 计算当前像素点和周围点的像素灰度值差异的绝对值
                diff = abs(int(img[current_point[0] + i, current_point[1] + j]) - int(seed_value))
                # 若绝对值超出设定阈值，则认为两个像素点不相似
                # 若绝对值小于设定阈值，则认为两个像素点相似，此时若该像素点还未被标记则将其设置为前景区域，记为已分割
                if diff < threshold and segmented[current_point[0] + i, current_point[1] + j] == 0:
                    segmented[current_point[0] + i, current_point[1] + j] = 255
                    # 将当前点加入队列，继续迭代遍历
                    queue.append((current_point[0] + i, current_point[1] + j))
    # 返回已分的图像数组
    return segmented
```

下面可以根据自编的函数对示例图片进行分割，这段代码中可以调整的是**种子点的位置选取**，可以通过图片大小以及种子位置的灰度值和 threshold 阈值来进行判断和调整。

```python
# 读取图片
img1 = cv2.imread('Image1.png')
seg1 = cv2.imread('Seg1.png')

# 查看图片大小（行数、列数、通道数）
height1, width1, channels1 = img1.shape
print('Image size: {} x {} pixels'.format(width1, height1))

# 将图像转化为灰度图像
gray_img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray_seg1 = cv2.cvtColor(seg1, cv2.COLOR_BGR2GRAY)

# 选取种子点并查看种子点的灰度值，一定程度上决定了阈值的设置
x1, y1 = 160, 20
pixel_value = img1[y1, x1]
print('Pixel value at ({}, {}): {}'.format(x1, y1, pixel_value))
seed1 = (x1, y1)

# 进行区域增长
sgtd_img1 = region_growing(gray_img1, seed1)
sgtd_seg1 = region_growing(gray_seg1, seed1)

# 计算分割效果
Jaccard_index_1 = jaccard_index(sgtd_img1, sgtd_seg1)
print("Jaccard指数为：", Jaccard_index_1)
Dice_index_1 = dice_coefficient(sgtd_img1, sgtd_seg1)
print("Dice指数为：", Dice_index_1)
index_1 = (Jaccard_index_1 + Dice_index_1)/2
print("平均分割效果为：", index_1)

# 显示分割结果
cv2.imshow('image1', img1)
cv2.imshow('seg1', seg1)
cv2.imshow('sgtd_img1', sgtd_img1)
cv2.imshow('sgtd_seg1', sgtd_seg1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

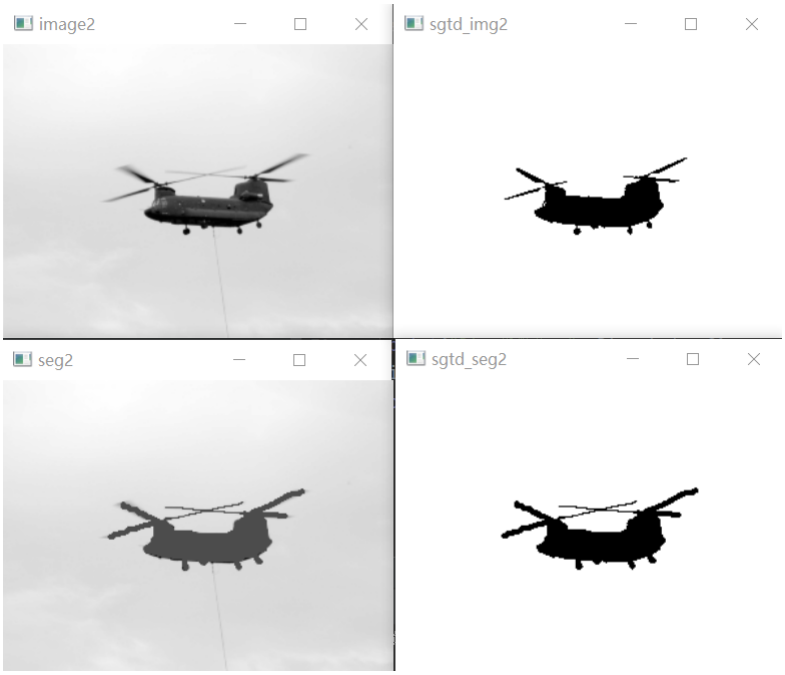
img1




```
Image1
threshold = 66
Image size: 300 x 400 pixels
Pixel value at (160, 20): [177 177 177]
Jaccard指数为： 0.9915752891125891
Dice指数为： 0.9957698255579558
平均分割效果为： 0.9936725573352725
```

可以看出对于 img1，在种子点取（160, 20），threshold 取 58 时计算出的分割效果较好，是因为输入图像和理想分割图像分割结果较为相似，但实际上我们的目的是需要分割出特定的物品，该分割进行区域增长后把灯塔下面的土地也包括进来，这是由于土地灰度值较高，比起和背景天空来说和物品本身更为接近。但是由于灯塔本身也是有亮面和暗面且区分度较大，因此如果降低阈值会导致分割不出完整的灯塔。

img2

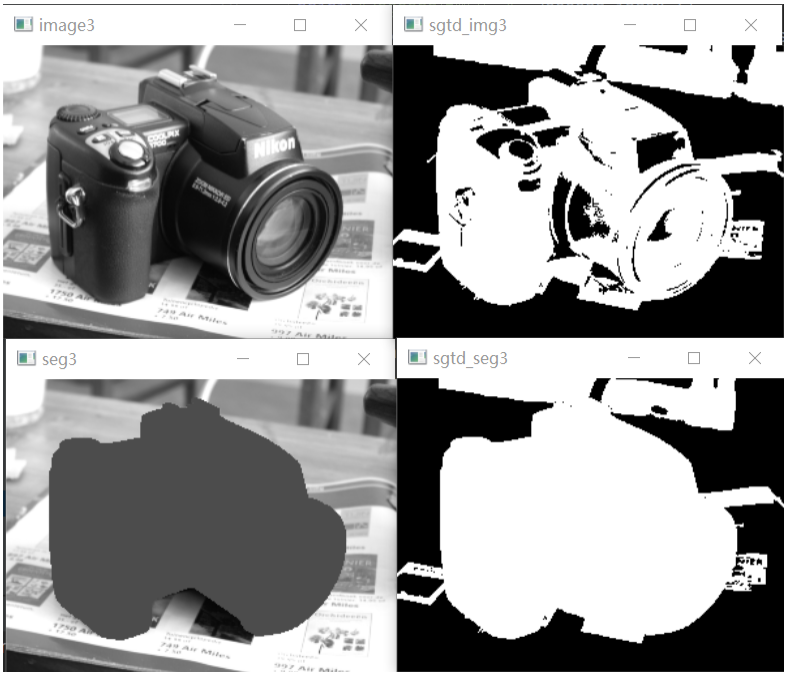


```
Image2
threshold = 66
Image size: 300 x 225 pixels
Pixel value at (150, 145): [218 218 218]
Jaccard指数为： 0.9918807112867375
Dice指数为： 0.9959238077525147
平均分割效果为： 0.9939022595196261
```

img3

在针对 img3 进行调整的时候我们将区域增长函数的阈值提高到了75，种子点设置在x3, y3 = 99, 100

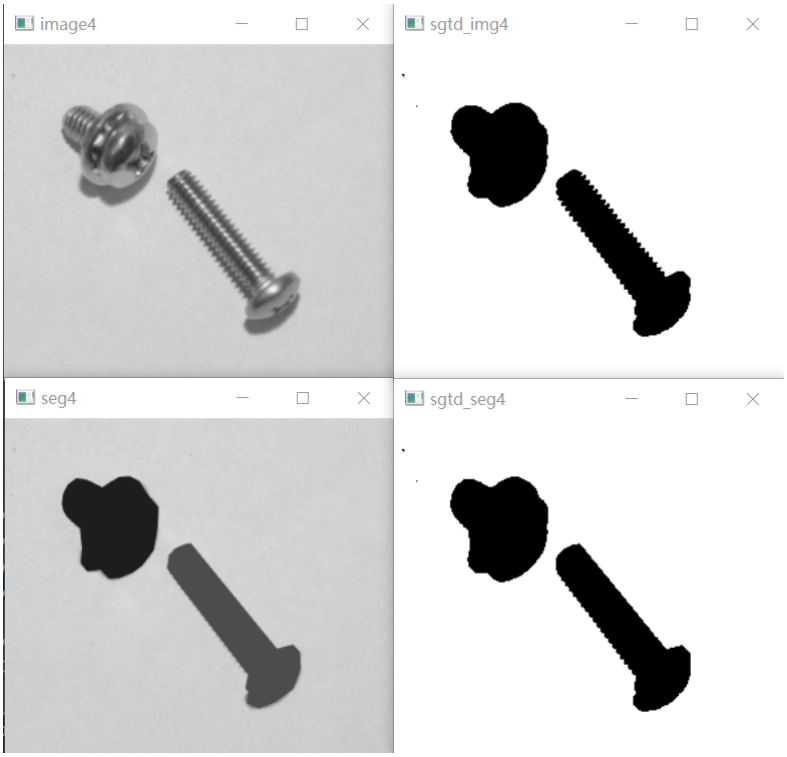
这是由于该图片需要分割的物品即使转化成灰度图像依旧可以看出形状较为不规则、在不同的面有反光的效果，因此需要较高的阈值覆盖反光区域。




```
Image3
threshold = 75
Image size: 300 x 225 pixels
Pixel value at (99, 100): [63 63 63]
Jaccard指数为： 0.8157473331526693
Dice指数为： 0.8985251617980272
平均分割效果为： 0.8571362474753483
```

img4

在针对 img3 进行调整的时候我们将区域增长函数的阈值降低到20，种子点设置在x4, y4 = 150, 145




```
Image4
threshold = 20
Image size: 300 x 258 pixels
Pixel value at (150, 145): [84 84 84]
Jaccard指数为： 0.9970338983050847
Dice指数为： 0.9985147464460005
平均分割效果为： 0.9977743223755426
```

三、优化方式

根据具体实现步骤中每一步查看哪些步骤可进行优化。

种子点选取

根据画布大小以及物品的位置选取。

前景区域和后景区域的设置。

区域生长阈值选取

看整体的明暗对比度。对比度强的可以设置高一些的阈值，对比度弱的不能设置太高的阈值。

根据种子点的灰度值进行阈值选择。（其实 OpenCV 库里有自动选择的代码）

图像预处理/后处理

去噪/分割边界平滑，通常使用均值滤波或者高斯滤波。

```python
# 定义高斯滤波
def gaussian_filter(img, ksize=3, sigma=1.5):
    return cv2.GaussianBlur(img, (ksize, ksize), sigma)

# 定义均值滤波
def median_filter(img, ksize=3):
    return cv2.medianBlur(img, ksize)
```

预处理：在进行区域增长之前进行去噪。

后处理：在得到区域增长结果之后进行边界平滑。

高斯滤波	(0,0)	(1,0)	(0,1)	(1,1)
img1	0.9937	0.9931	0.9914	0.9910
img2	0.9939	0.9940	0.9944	0.9944
img3	0.8571	0.7837	0.8831	0.8242
img4	0.9978	0.9955	0.9958	0.9945

均值滤波	(0,0)	(1,0)	(0,1)	(1,1)
img1	0.9937	0.9937	0.9937	0.9937
img2	0.9939	0.9945	0.9945	0.9945
img3	0.8571	0.8634	0.8635	0.8659
img4	0.9978	0.9976	0.9979	0.9979

	(0,0)	(G,M)	(M,G)
img1	0.9937	0.9931	0.9920
img2	0.9939	0.9943	0.9947
img3	0.8571	0.7732	0.8829
img4	0.9978	0.9957	0.9962