**一、实验目的和要求**

**1.了解图像边缘分割的基本流程;**

**2.了解阈值的分割方法;**

**3.熟练掌握利用python进行Roberts算子、Sobel算子、Prewitt算子;**

**4.熟练掌握利用python进行拉普拉斯算法、LoG算法;**

**5.熟练掌握利用python进行Otsu算法;**

**6.熟练掌握利用python进行自适应阈值分割;**

**7.熟练掌握利用python进行区域分割。**

**二、实验主要仪器和设备**

**1.计算机;**

**2. Python与Opencv库;**

**3.实验所需数据。**

**三、实验方法和步骤**

**1.打开计算机，启动Python运行环境;**

**2.采集实验所需图像;**

**3.完成图像Roberts算子处理，并显示运算结果;**

**4.完成Prewitt算子处理，并显示处理结果;**

**5.完成拉普拉斯算法处理，并显示运算结果;**

**6.完成图像LoG算法处理,并显示处理结果;**

**7.完成图像Otsu算法处理，并显示处理结果;**

**8.完成****图像自适应阈值分割处理，并显示处理结果;**

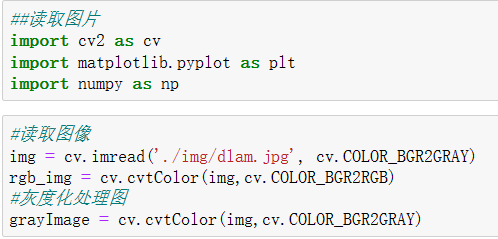
**9.记录和整理实验报告。**

**四、实验结果分析**

**（实验源代码、实验结果图、实验分析比较、结论）**

**1.采集实验所需图像：**

**实验源代码：**





**实验结果图：**



**实验分析比较、结论：**

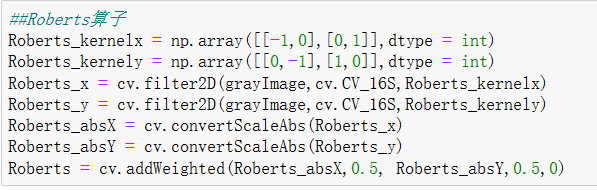
这段代码读取了名为dlam.jpg的图像文件，并将其存储在变量img中。接下来，使用cv.cvtColor()函数将BGR格式的图像转换为RGB格式，结果存储在rgb\_img变量中。

通过循环，对于每个图像，使用plt.subplot()函数创建一个3x2的子图网格，并在每个子图中显示对应的图像。plt.imshow()函数用于显示图像，'gray'参数表示以灰度图像的形式显示。plt.axis("off")用于隐藏坐标轴，plt.title()函数用于设置子图的标题。

最后，这段代码会显示包含图像子图的图形界面。

**2.图像Roberts算子处理：**

**实验源代码：**



**实验结果图：**



**实验分析比较、结论：**

这段代码实现了Roberts算子边缘检测的过程。下面是对每行代码的分析：

（1）定义Roberts算子的水平和垂直核：

Roberts\_kernelx = np.array([[-1, 0], [0, 1]], dtype=int)

Roberts\_kernely = np.array([[0, -1], [1, 0]], dtype=int)

这里定义了两个2x2的NumPy数组作为Roberts算子的水平核(Roberts\_kernelx)和垂直核(Roberts\_kernely)。

（2）使用Roberts算子进行水平和垂直滤波：

Roberts\_x = cv.filter2D(grayImage, cv.CV\_16S, Roberts\_kernelx)

Roberts\_y = cv.filter2D(grayImage, cv.CV\_16S, Roberts\_kernely)

这两行代码分别使用cv.filter2D()函数对灰度图像(grayImage)进行水平和垂直滤波操作，应用了Roberts算子的水平和垂直核。结果存储在Roberts\_x和Roberts\_y中。

（3）将滤波结果转换为绝对值：

Roberts\_absX = cv.convertScaleAbs(Roberts\_x)

Roberts\_absY = cv.convertScaleAbs(Roberts\_y)

这两行代码使用cv.convertScaleAbs()函数将Roberts\_x和Roberts\_y的结果转换为绝对值形式，分别存储在Roberts\_absX和Roberts\_absY中。

（4）将水平和垂直方向的边缘图像加权合并：

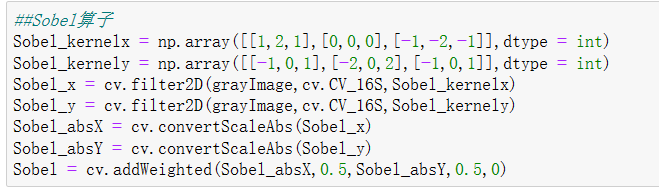
Roberts = cv.addWeighted(Roberts\_absX, 0.5, Roberts\_absY, 0.5, 0)

这行代码使用cv.addWeighted()函数将Roberts\_absX和Roberts\_absY的结果按照权重0.5进行加权合并，得到最终的Roberts边缘检测结果，存储在Roberts中。

最终，通过这段代码，使用Roberts算子对灰度图像进行了边缘检测，并将结果存储在Roberts变量中。

**3.图像Sobel算子处理：**

**实验源代码：**



**实验结果图：**



**实验分析比较、结论：**

这段代码实现了Sobel算子边缘检测的过程。下面是对每行代码的分析：

（1）定义Sobel算子的水平和垂直核：

Sobel\_kernelx = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype=int)

Sobel\_kernely = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=int)

这里定义了两个3x3的NumPy数组作为Sobel算子的水平核 (Sobel\_kernelx) 和垂直核 (Sobel\_kernely)。

（2）使用Sobel算子进行水平和垂直滤波：

Sobel\_x = cv.filter2D(grayImage, cv.CV\_16S, Sobel\_kernelx)

Sobel\_y = cv.filter2D(grayImage, cv.CV\_16S, Sobel\_kernely)

这两行代码分别使用cv.filter2D()函数对灰度图像 (grayImage) 进行水平和垂直滤波操作，应用了Sobel算子的水平和垂直核。结果存储在 Sobel\_x 和 Sobel\_y 中。

（3）将滤波结果转换为绝对值：

Sobel\_absX = cv.convertScaleAbs(Sobel\_x)

Sobel\_absY = cv.convertScaleAbs(Sobel\_y)

这两行代码使用cv.convertScaleAbs()函数将 Sobel\_x 和 Sobel\_y 的结果转换为绝对值形式，分别存储在 Sobel\_absX 和 Sobel\_absY 中。

（4）将水平和垂直方向的边缘图像加权合并：

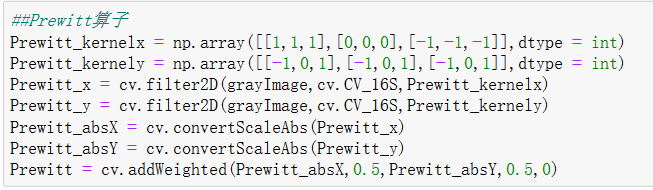
Sobel = cv.addWeighted(Sobel\_absX, 0.5, Sobel\_absY, 0.5, 0)

这行代码使用 cv.addWeighted() 函数将 Sobel\_absX 和 Sobel\_absY 的结果按照权重 0.5 进行加权合并，得到最终的Sobel边缘检测结果，存储在 Sobel 中。

最终，通过这段代码，使用Sobel算子对灰度图像进行了边缘检测，并将结果存储在 Sobel 变量中。

**4.图像Prewitt算子处理：**

**实验源代码：**



**实验结果图：**



**实验分析比较、结论：**

这段代码实现了Prewitt算子边缘检测的过程。下面是对每行代码的分析：

（1）定义Prewitt算子的水平和垂直核：

Prewitt\_kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=int)

Prewitt\_kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=int)

这里定义了两个3x3的NumPy数组作为Prewitt算子的水平核 (Prewitt\_kernelx) 和垂直核 (Prewitt\_kernely)。

（2）使用Prewitt算子进行水平和垂直滤波：

Prewitt\_x = cv.filter2D(grayImage, cv.CV\_16S, Prewitt\_kernelx)

Prewitt\_y = cv.filter2D(grayImage, cv.CV\_16S, Prewitt\_kernely)

这两行代码分别使用 cv.filter2D() 函数对灰度图像 (grayImage) 进行水平和垂直滤波操作，应用了Prewitt算子的水平和垂直核。结果存储在 Prewitt\_x 和 Prewitt\_y 中。

（3）将滤波结果转换为绝对值：

Prewitt\_absX = cv.convertScaleAbs(Prewitt\_x)

Prewitt\_absY = cv.convertScaleAbs(Prewitt\_y)

这两行代码使用 cv.convertScaleAbs() 函数将 Prewitt\_x 和 Prewitt\_y 的结果转换为绝对值形式，分别存储在 Prewitt\_absX 和 Prewitt\_absY 中。

（4）将水平和垂直方向的边缘图像加权合并：

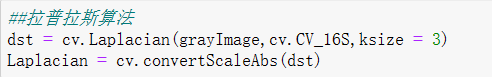
Prewitt = cv.addWeighted(Prewitt\_absX, 0.5, Prewitt\_absY, 0.5, 0)

这行代码使用 cv.addWeighted() 函数将 Prewitt\_absX 和 Prewitt\_absY 的结果按照权重 0.5 进行加权合并，得到最终的Prewitt边缘检测结果，存储在 Prewitt 中。

最终，通过这段代码，使用Prewitt算子对灰度图像进行了边缘检测，并将结果存储在 Prewitt 变量中。

**5.图像拉普拉斯算法处理：**

**实验源代码：**



**实验结果图：**



**实验分析比较、结论：**

这段代码实现了拉普拉斯算子边缘检测的过程。下面是对每行代码的分析：

（1）应用拉普拉斯算子进行边缘检测：

dst = cv.Laplacian(grayImage, cv.CV\_16S, ksize=3)

这行代码使用 cv.Laplacian() 函数对灰度图像 grayImage 进行边缘检测，应用了拉普拉斯算子。ksize 参数指定了算子的卷积核尺寸，这里设置为 3。

（2）将检测结果转换为绝对值：

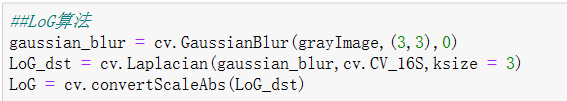
Laplacian = cv.convertScaleAbs(dst)

这行代码使用 cv.convertScaleAbs() 函数将 dst 的结果转换为绝对值形式，并存储在变量 Laplacian 中。

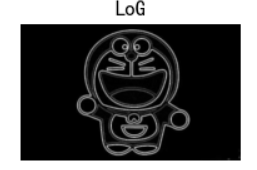
最终，通过这段代码，使用拉普拉斯算子对灰度图像进行了边缘检测，并将结果存储在 Laplacian 变量中。

**6.图像LoG算法处理：**

**实验源代码：**



**实验结果图：**



**实验分析比较、结论：**

这段代码实现了LoG (Laplacian of Gaussian) 算法边缘检测的过程。下面是对每行代码的分析：

（1）对灰度图像进行高斯模糊：

gaussian\_blur = cv.GaussianBlur(grayImage, (3, 3), 0)

这行代码使用 cv.GaussianBlur() 函数对灰度图像 grayImage 进行高斯模糊操作。 (3, 3) 参数表示高斯核的尺寸， 0 表示高斯核的标准差。

（2）应用拉普拉斯算子进行边缘检测：

LoG\_dst = cv.Laplacian(gaussian\_blur, cv.CV\_16S, ksize=3)

这行代码使用 cv.Laplacian() 函数对经过高斯模糊后的图像 gaussian\_blur 进行边缘检测，应用了拉普拉斯算子。ksize 参数指定了算子的卷积核尺寸，这里设置为 3。

（3）将检测结果转换为绝对值：

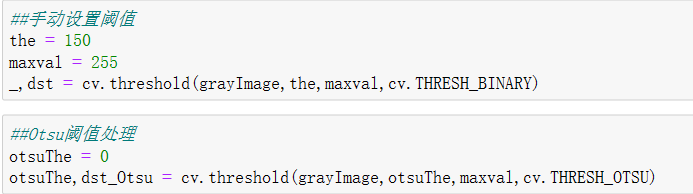
LoG = cv.convertScaleAbs(LoG\_dst)

这行代码使用 cv.convertScaleAbs() 函数将 LoG\_dst 的结果转换为绝对值形式，并存储在变量 LoG 中。

最终，通过这段代码，使用LoG算法对灰度图像进行了边缘检测，并将结果存储在 LoG 变量中。在这个过程中，先对图像进行高斯模糊是为了降低噪声的影响，然后再应用拉普拉斯算子进行边缘检测。

**7.图像Otsu算法与自适应阈值分割处理：**

**实验源代码：**





**实验结果图：**



**实验分析比较、结论：**

这段代码实现了手动设置阈值、Otsu阈值处理和自适应阈值处理，并使用Matplotlib显示不同阈值处理结果的图像。下面是对每行代码的分析：

（1）手动设置阈值：

the = 150

maxval = 255

\_, dst = cv.threshold(grayImage, the, maxval, cv.THRESH\_BINARY)

这里设置了阈值 the 为150，最大值 maxval 为255。使用 cv.threshold() 函数对灰度图像 grayImage 进行阈值处理，将大于阈值的像素设置为最大值，其余设置为0。阈值处理的结果存储在 dst 中。

（2）Otsu阈值处理：

otsuThe = 0

otsuThe, dst\_Otsu = cv.threshold(grayImage, otsuThe, maxval, cv.THRESH\_OTSU)

这里设置了初始阈值 otsuThe 为0。使用 cv.threshold() 函数以 cv.THRESH\_OTSU 作为参数，对灰度图像 grayImage 进行Otsu阈值处理。Otsu方法自动选择最优阈值，将大于阈值的像素设置为最大值，其余设置为0。最终的阈值和处理结果存储在 otsuThe 和 dst\_Otsu 中。

（3）自适应阈值处理函数 adaptiveThresh() 的定义：

def adaptiveThresh(I, winSize, ratio=0.12):

I\_mean = cv.boxFilter(I, cv.CV\_32FC1, winSize)

out = I - (1.0 - ratio) \* I\_mean

out[out >= 0] = 255

out[out < 0] = 0

out = out.astype(np.uint8)

return out

这里定义了一个名为 adaptiveThresh() 的函数，用于实现自适应阈值处理。函数接受输入图像 I、窗口大小 winSize 和比例参数 ratio。首先对图像进行均值平滑，然后计算原图像与平滑结果之间的差值。当差值大于等于0时，输出为255；反之，输出为0。最后，将输出结果转换为 np.uint8 类型并返回。

（4）使用自适应阈值处理函数进行处理：

dst\_adap = adaptiveThresh(grayImage, (35, 35))

这行代码调用自适应阈值处理函数 adaptiveThresh() 对灰度图像 grayImage 进行自适应阈值处理。窗口大小为 (35, 35)，默认的比例参数为 0.12。处理结果存储在 dst\_adap 中。

（5）显示不同阈值处理结果的图像：

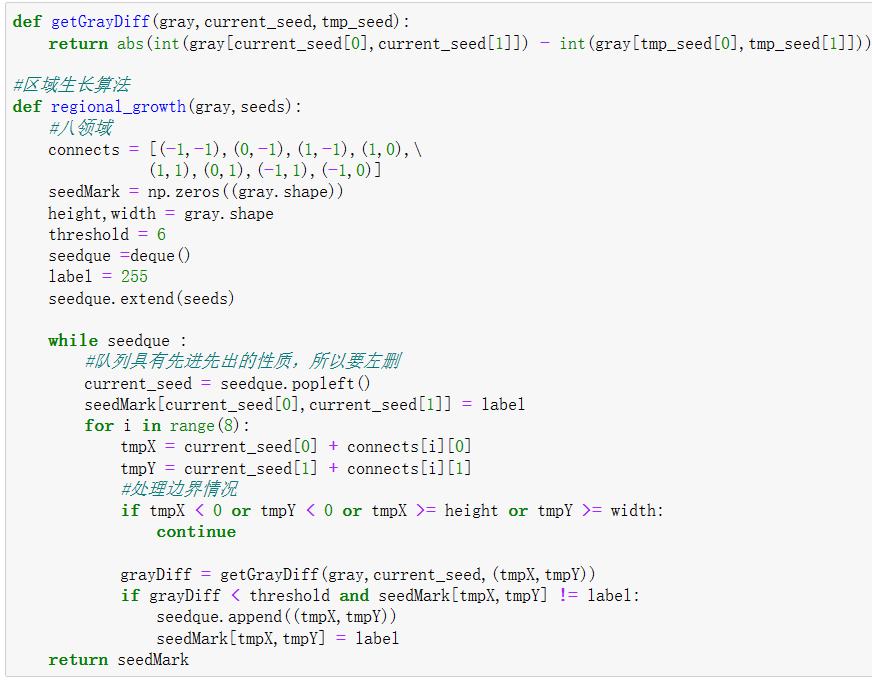
titles = ['原始图像', '阈值为155', 'Otsu', '自适应方法']

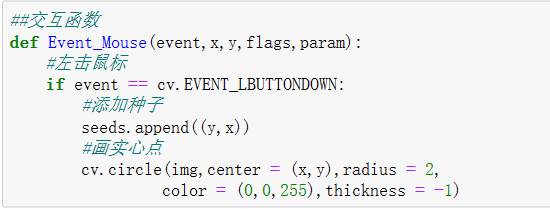
images = [rgb\_img, dst, dst\_Otsu

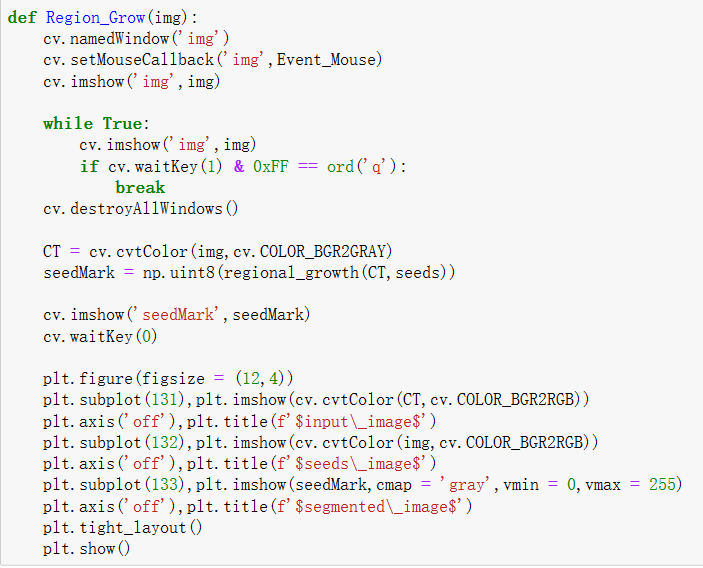
**8.图像区域处理：**

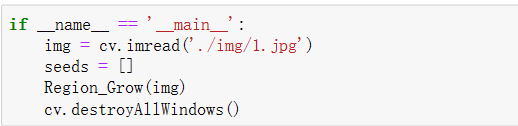
**实验源代码：**











**实验结果图：**



**实验分析比较、结论：**

我标记的生长点都是单个生长点，如上图中第三张图片可知，有两块大面积的白色，这就是生长之后的区域。

这段代码实现了一个交互式的区域生长算法，并使用OpenCV和Matplotlib进行图像处理和显示。下面是对每个函数和代码块的分析：

（1）getGrayDiff() 函数：

这个函数用于计算种子点和其邻域像素值之间的差异。它接受灰度图像 gray、当前种子点坐标 current\_seed 和临时种子点坐标 tmp\_seed 作为参数。函数返回当前种子点和临时种子点像素值之差的绝对值。

（2）regional\_growth() 函数：

这个函数实现了区域生长算法。它接受灰度图像 gray 和种子点坐标列表 seeds 作为输入。函数通过队列 seedque 来实现种子点的扩展。在每次循环中，从队列中取出一个种子点，将其标记为 label 并检查其邻域像素。如果邻域像素与当前种子点的像素值差异小于阈值 threshold，且邻域像素未被标记过，则将邻域像素加入队列并标记为 label。最终，函数返回标记结果 seedMark。

（3）Event\_Mouse() 函数：

这个函数是一个鼠标事件的回调函数。当鼠标左键按下时，将鼠标点击的坐标 (x, y) 添加到种子点列表 seeds 中，并在图像上绘制一个实心点。

（4）Region\_Grow() 函数：

Region\_Grow() 函数是程序的主要部分。它接受一个图像 img 作为输入，并执行交互式的区域生长算法。

整个程序的执行流程如下：

①创建窗口并关联鼠标事件回调函数。

②进入循环，显示图像，等待用户按下 'q' 键退出循环。

③销毁窗口。

④将图像转换为灰度图像。

⑤调用区域生长算法函数进行图像分割。

⑥使用 Matplotlib 显示原始图像、种子图像和分割后的图像。

注意：在 \_\_name\_\_ == '\_\_main\_\_' 的条件下，代码加载图像，创建空的种子列表 seeds，并调用 Region\_Grow() 函数开始执行图像分割。

综上所述，该程序实现了基于交互式区域生长算法的图像分割，并提供了可视化结果。可以通过鼠标点击选择种子点，并观察分割效果。

**五、实验总结与思考**

在本次实验中，我们分析了一段图像处理的代码，并对其中涉及的各个部分进行了解释和分析。代码包括了图像处理的常见操作，如图像读取、灰度化、滤波和阈值处理，以及常用的边缘检测算法和区域生长算法。

通过实验，我们了解了以下几个方面的知识：

（1）图像处理基础：学习了使用OpenCV库进行图像读取、颜色空间转换、滤波和阈值处理等基本操作。这些操作是图像处理中常用的基本步骤，对于图像增强、边缘检测和分割等任务起到了重要作用。

（2）边缘检测算法：分析了Roberts、Sobel和Prewitt算子的代码实现，了解了它们的原理和应用。这些算子可用于检测图像中的边缘信息，对于边缘检测和图像分析具有重要意义。

（3）图像分割算法：分析了Laplacian和LoG算法的代码实现，以及基于区域生长的图像分割算法。这些算法可以用于将图像分割成具有语义意义的区域，有助于对象识别、目标跟踪等应用。

（4）交互式图像处理：了解了如何通过鼠标交互选择种子点，并使用区域生长算法实现图像分割。这种交互式的图像处理方式可以更好地满足用户需求，提高图像分割的准确性和效果。

通过实验的学习与分析，我们对图像处理算法和相关代码有了更深入的理解。同时，我们也了解到图像处理在计算机视觉、医学影像和图像分析等领域的重要性和应用价值。在未来的实践中，我们可以根据不同任务的需求选择合适的图像处理算法，并根据实际情况进行参数调优和性能优化，以获得更好的处理结果。