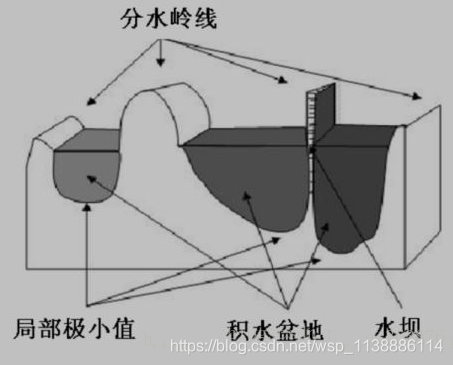
# 功能与函数介绍

分水岭算法是一种图像区域分割法，在分割的过程中，它会把跟临近像素间的相似性作为重要的参考依据，从而将在空间位置上相近并且灰度值相近的像素点互相连接起来构成一个封闭的轮廓，**封闭性是分水岭算法的一个重要特征**。  
其他[图像分割](https://so.csdn.net/so/search?q=%E5%9B%BE%E5%83%8F%E5%88%86%E5%89%B2&spm=1001.2101.3001.7020)方法，如阈值，边缘检测等都不会考虑像素在空间关系上的相似性和封闭性这一概念，彼此像素间互相独立，没有统一性。分水岭算法较其他分割方法更具有思想性，更符合人眼对图像的印象。  
  
任意的灰度图像可以被看做是地质学表面，高亮度的地方是山峰，低亮度的地方是山谷。给每个孤立的山谷（局部最小值）不同颜色的水（标签），当水涨起来，根据周围的山峰（梯度），不同的山谷也就是不同的颜色会开始合并，要避免这个，你可以在水要合并的地方建立障碍，直到所有山峰都被淹没。你所创建的障碍就是分割结果，这个就是分水岭的原理，但是这个方法会分割过度，因为有噪点，或者其他图像上的错误。所以[OpenCV](https://so.csdn.net/so/search?q=OpenCV&spm=1001.2101.3001.7020)实现了一个基于掩模的分水岭算法，你可以指定哪些是要合并的点，哪些不是，这是一个交互式的图像分割，我们要做的是给不同的标签。给我们知道是前景或者是目标用一种颜色加上标签，给我们知道是背景或者非目标加上另一个颜色，最后不知道是什么的区域标记为0. 然后使用分水岭算法。

## 1 cv2.distanceTransform(src, distanceType, maskSize)

距离变换的基本含义是计算一个图像中非零像素点到最近的零像素点的距离，也就是到零像素点的最短距离个最常见的距离变换算法就是通过连续的腐蚀操作来实现，腐蚀操作的停止条件是所有前景像素都被完全腐蚀。这样根据腐蚀的先后顺序，我们就得到各个前景像素点到前景中心呗Ⅵ像素点的距离。根据各个像素点的距离值，设置为不同的灰度值。这样就完成了二值图像的距离变换

参数：

distanceType – 距离类型：cv2.DIST\_L1, cv2.DIST\_L2 , cv2.DIST\_C

maskSize – 距离变换蒙版的大小：取值3，5 或 CV\_DIST\_MASK\_PRECISE（后一个选项仅由第一个函数支持）。在CV\_DIST\_L1或CV\_DIST\_C距离类型的情况下，参数被强制为3。

## 2 cv2. distanceTransformWithLabels(src, distanceType, maskSize, dst=None, labels=None, labelType=None)

labels – 可选输出2D标签数组（离散Voronoi图）: 它的类型为CV\_32SC1，大小与src相同

labelType – 构建标签数组的类型:  
若为 DIST\_LABEL\_CCOMP，则src中每个连接的零组件（以及最接近连接组件的所有非零像素）将被分配相同的标签。  
若为DIST\_LABEL\_PIXEL，那么每个零像素（以及最接近它的所有非零像素）都会获得自己的标签。

## 实例1.

|  |
| --- |
|  |
|  |
|  |

### 效果：

|  |
| --- |
|  |

## 一、 原理

### 1. 分水岭算法原理

* 任何一副灰度图像都可以被看成拓扑平面，**灰度值高**的区域可以被看成是**山峰**，**灰度值低**的区域可以被看成是**山谷**。我们向每一个山谷中灌不同颜色的水。随着水的位的升高，不同山谷的水就会相遇汇合，为了防止不同山谷的水汇合，我们需要在水汇合的地方构建起堤坝。不停地灌水，不停地构建堤坝知道所有的山峰都被水淹没。我们构建好的堤坝就是对**图像的分割**，这就是分水岭算法的背后原理。
* OpenCV采用了**基于掩模**的分水岭算法，在这种算法中我们要设置那些山谷点会汇合，那些不会。这是一种交互式的图像分割，我们要做的就是给我们已知的对象打上不同的标签。如果某个区域肯定是前景或对象，就使用某个颜色（或灰度值）标签标记它。如果某个区域肯定不是对象而是背景就使用另外一个颜色标签标记。而剩下的不能确定是前景还是背景的区域就用 0 标记，这就是我们的标签。然后实施分水岭算法。每一次灌水，我们的标签就会被更新，当两个不同颜色的标签相遇时就构建堤坝，直到将所有山峰淹没，最后我们得到的边界对象（堤坝）的值为 -1。

### 2. 距离变换

* 距离变换的基本含义是计算一个图像中非零像素点到最近的零像素点的距离，也就是到零像素点的最短距离。
* 最常见的距离变换算法就是通过连续的腐蚀操作来实现，腐蚀操作的停止条件是所有前景像素都被完全。
* 腐蚀。这样根据腐蚀的先后顺序，我们就得到各个前景像素点到前景中心骨架像素点的距离。
* 根据各个像素点的距离值，设置为不同的灰度值。这样就完成了二值图像的距离变换。

### 3. opencv有关函数的用法

代码语言：javascript

复制

cv2.distanceTransform(src, distanceType, maskSize, dst=None, dstType=None)

* src：输入二值图像
* distanceType：计算距离的方式
* maskSize：蒙板尺寸

代码语言：javascript

复制

cv2.connectedComponents(image, labels=None, connectivity=None, ltype=None)

* image：输入8位单通道图像
* labels：输出标签地图
* connectivity：连通性，默认8，还可以取4。
* Itype：输出标签类型 ，默认 CV\_32S， 还可以取CV\_16U。

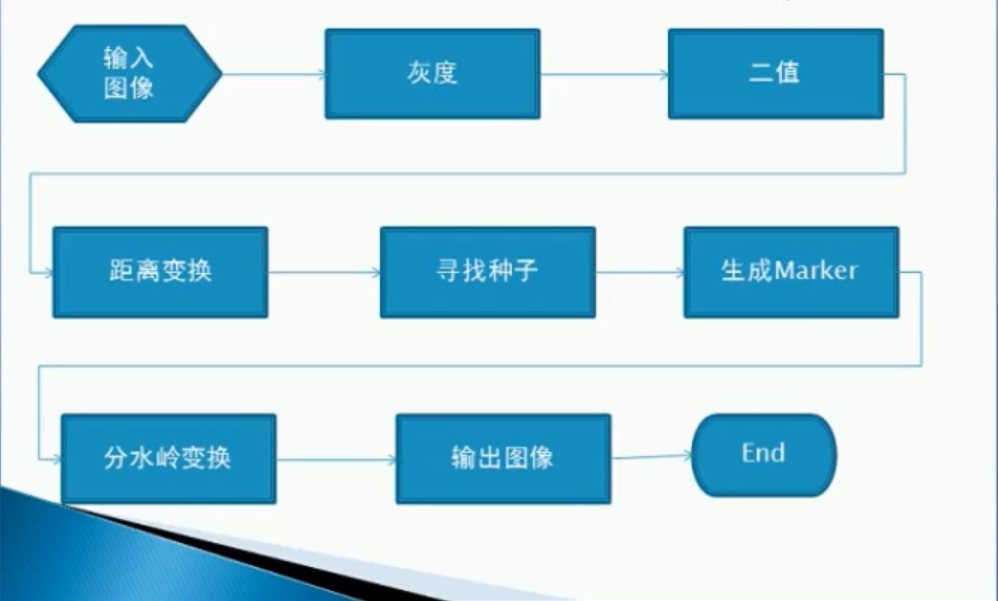
代码语言：javascript

复制

cv2.watershed(image, markers)

* image：输入图像
* markers：标记

## 二、基于距离的分水岭分割流程



* 输入图像，有噪声的话，先进行去噪。
* 转成灰度图像
* 二值化处理、形态学操作
* 距离变换
* 寻找种子、生成marker
* 实施分水岭算法、输出分割后的图像

## 实例2

|  |
| --- |
| *# -\*- coding: UTF-8 -\*-* **import** cv2 **as** cv **import** numpy **as** np   **def** watershed\_algorithm(image):  *# 边缘保留滤波EPF 去噪* blur = cv.pyrMeanShiftFiltering(image,sp=10,sr=100)  *# 转成灰度图像* gray = cv.cvtColor(blur, cv.COLOR\_BGR2GRAY)  *# 得到二值图像 自适应阈值* ret, binary = cv.threshold(gray, 0, 255, cv.THRESH\_BINARY | cv.THRESH\_OTSU)  *# cv.imshow('binary image', binary)   # 形态学操作 获取结构元素 开操作* kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3))  opening = cv.morphologyEx(binary, cv.MORPH\_OPEN, kernel=kernel, iterations=2)  *# 确定区域* sure\_bg = cv.dilate(opening, kernel, iterations=3)  *# cv.imshow('mor-opt', sure\_bg)   # 距离变换* dist = cv.distanceTransform(opening, cv.DIST\_L2, 3)  dist\_out = cv.normalize(dist, 0, 1.0, cv.NORM\_MINMAX)  *# cv.imshow('distance-', dist\_out \* 50)* ret, surface = cv.threshold(dist\_out, dist\_out.max() \* 0.55, 255, cv.THRESH\_BINARY)  *# cv.imshow('surface-markers', surface)* surface\_fg = np.uint8(surface) *# 转成8位整型* unkonown = cv.subtract(sure\_bg, surface\_fg) *# 找到位置区域  # Marker labelling* ret, markers = cv.connectedComponents(surface\_fg) *# 连通区域* print(ret)   *# 分水岭变换  # Add one to all labels so that sure background is not 0, but 1* markers = markers + 1  *# Now, mark the region of unknown with zero* markers[unkonown == 255] = 0  *# 实施分水岭算法了。标签图像将会被修改，边界区域的标记将变为 -1* markers = cv.watershed(image, markers=markers)  image[markers == -1] = [0, 0, 255] *# 被标记的区域 设为红色* cv.imshow(**'result'**, image)   src = cv.imread(**'../mydata/silver\_coins.jpeg'**) src = cv.resize(src, **None**, fx=0.5, fy=0.5) cv.imshow(**'input image'**, src) watershed\_algorithm(src) cv.waitKey(0) cv.destroyAllWindows() |

### 效果：

|  |
| --- |
|  |