# 中国矿业大学信控学院

# 实验报告

**课程名称： 机器人技术试验 成 绩：**

**实验名称：图像分割与描述 班 级：**

**姓名： 班级序号： 同组人：**

**实验台号： 实验日期：**

**教师签字：**

**实验三 图像分割与描述**

**说明：1）内容包括以下部分，第一至三部分预习报告需要完成的，特别是没有提供的代码需要在第三部分给出；2）预习报告需要打印，做实验时签字。最终报告在预习报告的基础上完善添加后面部分；3）报告内容可以比要求提供的部分多。4）上交报告打印版时，电子档与相关代码也需雨课堂提交，上交电子版会使用作业内容重复系统进行检测；**

**一、算法原理**

**1、二值化图像**

二值化图像是一种只包含黑白两种像素值的图像类型，在二值化图像中，像素值只能为0或1，其中0代表黑色，1代表白色。

二值化图像的原理是将灰度图像转换为只包含黑白两种颜色的图像。在计算机中，灰度图像是由像素点组成的二维矩阵，每个像素点的值在0-255之间，代表了图像该点的亮度，值为0表示最暗，值为255表示最亮。二值化的过程就是将灰度图像中某个阈值以下的像素点值设置为0，其他像素点设置为1，即将图像从灰度空间转换为二进制空间。

**2、形态学滤波**

形态学滤波是一种基于形态学原理的滤波算法，它可以用于图像处理中的去噪、边缘检测、加强特征等操作。形态学滤波的基本思想是通过图像中像素点的形态变化，对输入图像进行平滑、增强边缘等处理。

形态学滤波算法的核心是结构元素，它是一个二值矩阵，用于描述某个形状的模板或模型。形态学滤波的过程就是将结构元素沿着图像的每个像素点移动并进行形状匹配计算，以便得到处理后的图像。

基本的形态学滤波操作有腐蚀和膨胀。腐蚀操作用于去除图像中噪声点或小区域，原理是对于结构元素覆盖的像素点，只有当所有像素都为1时，在结果图中相应区域才为1。膨胀操作用于增强图像边缘或扩大区域，原理是对于结构元素覆盖的像素点，只要有一个像素为1，那么结果图相应区域就为1。此外，还有开运算、闭运算和梯度操作等，它们可以用来平滑和增强图像细节。在进行形态学滤波前，需要先设置结构元素，以及形态学滤波的操作类型和次数等参数。

**3、轮廓绘制**

绘制轮廓是一种基于边缘检测和形态学处理的图像处理技术，用于提取图像中的特定边缘轮廓或形状。其主要原理是通过对图像进行边缘检测获取图像中物体的边缘，然后进行形态学处理使得边缘变得稳定，最终将稳定的轮廓信息绘制在原始图像上。

具体的绘制轮廓技术大致包括以下几个步骤：

1. 将彩色图像转换为灰度图像，并进行预处理，如平滑以去除噪声。

2. 进行边缘检测，常用的边缘检测算子包括Sobel算子、Canny算子等，其目的是在图像中找到物体的轮廓线段。

3. 对轮廓线段进行形态学处理，通常包括膨胀、腐蚀、开运算、闭运算等操作，以消除干扰，使得轮廓更加平滑、连续、稳定。

4. 对处理后的轮廓线段进行提取和绘制。提取轮廓线段的方法有多种，如边缘追踪法、Hough变换等，可以根据需要选择合适的方法。绘制轮廓时，可以通过将轮廓线段分割为一定长度的线段，然后使用线段匹配等技术将其拟合成一个完整的轮廓线段，最终将轮廓线段信息绘制在原始图像上，以实现轮廓绘制的效果。

**4、重建边界**

计算各区域边界点的傅里叶描述子并用四分之一点重建边界需要进行以下步骤：

①获取边界点

使用图像处理技术（例如Canny算子）从原始图像中提取边缘点。

②将图像分割为区域

将边缘点根据它们所在的区域进行分割并储存在不同的数组中。

③计算每个区域的傅里叶描述子

对于每个边界点数组，使用离散傅里叶变换（DFT）计算每个点的傅里叶系数。在使用DFT前，为了更准确地描述边界，需要将边界点按照其离散长度进行插值和平滑操作。然后将系数计算结果保存为一个向量，表示该区域的傅里叶描绘子。

④进行主成分分析（PCA）降维

每个区域得到一个傅里叶描述子向量，每个向量的维度往往非常高，需要执行主成分分析（PCA）来压缩向量维度。

⑤重建边界

重建边界需要先将傅里叶描绘子转换成傅里叶系数。为了减少计算量，在系数中只保留前几项，然后进行反傅里叶变换以生成求得的区域边界。最后，将所有区域的重建结果拼接在一起，即可得到原始图像的边界粗略轮廓。初步生成的粗略轮廓上只有很少的点（原始边缘点的四分之一），需要进行进一步处理，可以使用插值方法从这些点中生成更精确的曲线。

1. **GrabCut分割图像**

GrabCut主要利用高斯混合模型和最小割理论进行分割，其原理可以简单概括如下：

①初始化：将图像分成两个部分，一个前景部分和一个背景部分，一般使用矩形或椭圆来选择前景部分，随后根据所选部分进行初始化分割。

②高斯混合模型：使用高斯混合模型对前景和背景进行建模，即通过计算颜色空间中每个像素点的概率分布，来对图像中的颜色信息进行建模，并将其划分为两个别，即前景和背景。

③迭代最优化：开始迭代寻找最优的分割。在迭代中，GRABCUT会根据当前分割图，更新前景和背景的高斯混合模型，并重新计算每个像素属于前景或者背景的概率。

④最小割分割：GRABCUT算法通过对两个高斯混合模型的交集进行最小割操作，得到一个图像的割图。具体来说，GRABCUT算法会将高斯混合模型看作图像二分图的两部分，并将每个像素作为节点，根据其属于前景或背景的概率作为节点的权值，建立一个边权图。通过在这个图上求解最小割问题，将图像分成前景和背景两个部分。

⑤重复迭代：重复以上步骤直到满足终止条件，如迭代次数达到预设次数或分割结果满足特定条件为。

1. **编程思路与程序流程**
2. 二值化图像

以8位深度，1通道的方式，即cv2.imread('plane.jpg', 0)读取图像，调用cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)即可完成对图像的二值化。

1. 形态学滤波

对二值化后的图像调用cv2.morphologyEx(src, op, kernel)即可完成图像的各种形态学变化，此函数参数中的‘op’即是对图像形态学变化方式的选择，可选项如下：

cv2.MORPH\_OPEN 开运算 ,先腐蚀后膨胀的过程。开运算能够除去孤立的小点, 毛刺和小桥,而总的位置和形状不变

cv2.MORPH\_CLOSE 闭运算,先膨胀后腐蚀的过程。闭运算能够填平小孔,弥合小裂缝, 而总的位置和形状不变。

cv2.MORPH\_GRADIENT 形态学梯度,能够突出团块的边缘,保留物体的边缘轮廓。

cv2.MORPH\_TOPHAT 顶帽,将突出比原轮廓亮的部分。

cv2.MORPH\_BLACKHAT 黑帽,将突出比原轮廓暗的部分。

1. 绘制轮廓

对灰度图像调用cv2.threshold(image1, 127, 255, cv2.THRESH\_BINARY\_INV)进行反二值化操作，然后调用cv2.findContours(BWimage,cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)即可完成对图像轮廓的寻找，之后再调用cv2.drawContours(image, contours, -1, (0, 0, 255), 1)即可完成对图像轮廓的绘制。

1. GrabCut分割图像

首先裁剪图像，用一个矩形或椭圆框选一个包含对象的正方形，作为图像分割的初始区域。然后确定前景和背景种子像素，用画笔工具手动画出一部分前景像素和背景像素，作为种子像素。之后使用高斯混合模型建模前景和背景的颜色分布，计算每个像素属于前景和背景的概率，再使用最小割算法分割每个像素是否属于前景或背景，更新对应的斯混合模型。最后循环执行步骤3和4，直到迭代满足终止条件

1. **算法各部分主要函数代码以及功能注释**

**1.#二值化图像**

import cv2

image = cv2.imread('plane.jpg', 0)

"""

cv2.threshold()

功能：对图像进行阈值处理

调用格式：

cv2.threshold (src, thresh, maxval, type)

src 源图片,必须是单通道

thresh 阈值,取值范围0～255

maxval 填充色,取值范围0～255

type 阈值类型,主要是以下5种

cv2.THRESH\_BINARY 二进制阈值化,非黑即白

cv2.THRESH\_BINARY\_INV 反二进制阈值化,非白即黑

cv2.THRESH\_TRUNC 截断阈值化 ,大于阈值设为阈值

cv2.THRESH\_TOZERO 阈值化为0 ,小于阈值设为0

cv2.THRESH\_TOZERO\_INV 反阈值化为0 ,大于阈值设为0

"""

ret,BWimage = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)#对图像进行二值化

cv2.imwrite('erzhihua.jpg',BWimage)

cv2.namedWindow('result', cv2.WINDOW\_NORMAL)

cv2.imshow('result', BWimage)

cv2.waitKey(0)

cv2.destroyAllWindows()

**2.#形态学滤波**

import cv2

image = cv2.imread('plane.jpg', 0)

ret, BWimage = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))

"""

cv2.morphologyEx()

功能：进行各类形态学的变化

调用格式：

cv2.morphologyEx(src, op, kernel)

第二个参数主要有5种选择

cv2.MORPH\_OPEN 开运算 ,先腐蚀后膨胀的过程。开运算能够除去孤立的小点,毛刺和小桥,而总的位置和形状不变

cv2.MORPH\_CLOSE 闭运算,先膨胀后腐蚀的过程。闭运算能够填平小孔,弥合小裂缝,而总的位置和形状不变。

cv2.MORPH\_GRADIENT 形态学梯度,能够突出团块的边缘,保留物体的边缘轮廓。

cv2.MORPH\_TOPHAT 顶帽,将突出比原轮廓亮的部分。

cv2.MORPH\_BLACKHAT 黑帽,将突出比原轮廓暗的部分。

"""

LBimage1 = cv2.morphologyEx(BWimage, cv2.MORPH\_OPEN, kernel)#开运算

LBimage2 = cv2.morphologyEx(BWimage, cv2.MORPH\_CLOSE, kernel)#闭运算

LBimage3 = cv2.morphologyEx(BWimage, cv2.MORPH\_GRADIENT, kernel)#形态学梯度

LBimage4 = cv2.morphologyEx(BWimage, cv2.MORPH\_TOPHAT, kernel)#顶帽运算

LBimage5 = cv2.morphologyEx(BWimage, cv2.MORPH\_BLACKHAT, kernel)#黑帽运算

A = cv2.hconcat([LBimage1, LBimage2, LBimage3 ])#水平方向合并图像

B = cv2.hconcat([LBimage4, LBimage5, BWimage])#水平方向合并图像

C =cv2.vconcat([A, B])#竖直方向合并图像

cv2.imwrite('xingtaixue.jpg',C)

cv2.namedWindow('result', cv2.WINDOW\_NORMAL)

cv2.imshow('result', C)

cv2.waitKey(0)

cv2.destroyAllWindows()

**3.#绘制轮廓**

import cv2

image = cv2.imread('plane.jpg')

image1 = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

ret, BWimage = cv2.threshold(image1, 127, 255, cv2.THRESH\_BINARY\_INV)

"""

cv2.findContours()

功能：寻找图像轮廓

调用格式：

cv2.findContours(image,mode,method)

mode表示轮廓检索模式,有4种选择

RETR\_EXTERNAL 只检测外轮廓

RETR\_LIST 检测的轮廓不建立等级关系

RETR\_CCOMP 建立两个等级的轮廓,上面的一层为外边界,里面的一层为内孔的边界信息。如果内孔内还有一个连通物体,这个物体的边界也在顶层。

RETR\_TREE 建立一个等级树结构的轮廓

method表示轮廓逼近方法,有3种选择

cv2.CHAIN\_APPROX\_NONE 存储轮廓所有点的信息,相邻两个轮廓点在图象上也是相邻的；

cv2.CHAIN\_APPROX\_SIMPLE 压缩水平方向,垂直方向,对角线方向的元素,只保留该方向的终点坐标；

cv2.CHAIN\_APPROX\_TC89\_L1 使用teh-Chinl chain 近似算法保存轮廓信息。

"""

contours, hierarchy = cv2.findContours(BWimage, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)#寻找轮廓

"""

cv2.drawContours()

功能：绘制图像轮廓

调用格式：

cv2.drawContours(image,contours,contourIdx,color, thickness=None, lineType=None)

第一个参数image表示目标图像,

第二个参数contours表示输入的轮廓组,每一组轮廓由点vector构成,

第三个参数contourIdx指明画第几个轮廓,如果该参数为负值,则画全部轮廓,

第四个参数color为轮廓的颜色,

第五个参数thickness为轮廓的线宽,如果为负值或CV\_FILLED表示填充轮廓内部,

第六个参数lineType为线型,

"""

cv2.drawContours(image, contours, -1, (0, 0, 255), 1)

cv2.imwrite('lunkuo.jpg',image)

cv2.namedWindow('result', cv2.WINDOW\_NORMAL)

cv2.imshow('result', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**4. #重建的图像**

import cv2

import numpy as np

gray = cv2.imread('plane.jpg',0)#读取单通道图像，即灰度图像

image = cv2.imread('plane.jpg')#读取三通道图像，即彩色图像

ret, BWimage = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY\_INV)#对灰度图像进行反二值化

contours, hierarchy = cv2.findContours(BWimage, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE) #寻找图像的外轮廓

contour\_array = contours[0]#记录图像的第一个轮廓

contours\_complex = np.empty(contour\_array.shape[:-1], dtype=complex)#建立一个数组

contours\_complex.real = contour\_array[:,:,0]

contours\_complex.imag = contour\_array[:,:,1]

for i in range(1,len(contours\_complex.real),4):

contours\_complex.real[i,0] = contour\_array[i,:,0]

for j in range(1,len(contours\_complex.imag),4):

contours\_complex.imag[j,0] = contour\_array[j,:,1]

fourier\_result = np.fft.fft(contours\_complex)#傅里叶描绘子转换成傅里叶系数

contour\_reconstruct = np.fft.ifft(fourier\_result)#进行反傅里叶变换以生成求得的区域边界

contour\_reconstruct = np.array([contour\_reconstruct.real,contour\_reconstruct.imag])

contour\_reconstruct = np.transpose(contour\_reconstruct)

contour\_reconstruct = np.expand\_dims(contour\_reconstruct, axis = 1)

cv2.drawContours(image,contour\_reconstruct.astype(int),-1, (0, 0, 255), 1)

cv2.imwrite('chongjian.jpg',image)

cv2.namedWindow("contour\_reconstruct",0)

cv2.imshow("contour\_reconstruct", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**5.#相关分割**

import cv2

import numpy as np

img = cv2.imread('plane1.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)#阈值分割

edges = cv2.Canny(gray, 100, 200)

contours, hierarchy = cv2.findContours(edges, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)# 基于边缘的分割

contour\_img = np.zeros\_like(img)

cv2.drawContours(contour\_img, contours, -1, (0, 0, 255), 2)

#GrabCut分割

mask = np.zeros(img.shape[:2], np.uint8)# 定义掩膜

# 定义背景和前景模型

bgdModel = np.zeros((1, 65), np.float64)

fgdModel = np.zeros((1, 65), np.float64)

# 定义矩形ROI

rect = (40, 40, img.shape[1]-100, img.shape[0]-300)

# 进行GrabCut算法分割

cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC\_INIT\_WITH\_RECT)

# 重新定义掩膜

mask2 = np.where((mask==2)|(mask==0), 0, 1).astype('uint8')

# 进行掩膜操作

res = cv2.bitwise\_and(img, img, mask=mask2)

# 显示原图和分割后的图像

cv2.imshow('Original Image', img)

cv2.imshow('Threshold Segmentation', thresh)

cv2.imshow('Edge-based Segmentation', contour\_img)

cv2.imshow('GrabCut Segmentation', res)

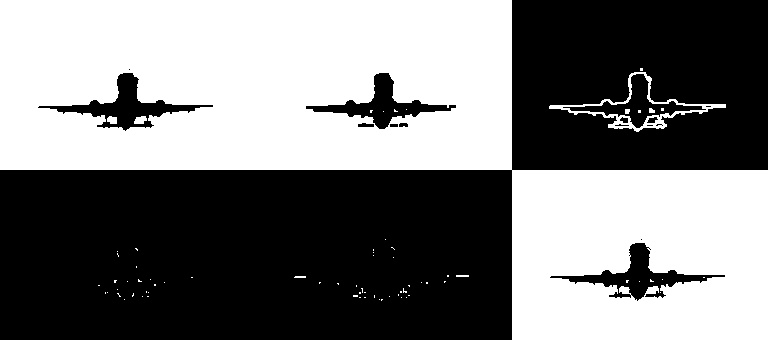
cv2.waitKey(0)

**四、调试过程与运行结果分析**

**1.按照序号一个个分析结果，并写出调试中遇到问题是怎么解决的**



二值化图像



各种形态学变化后的图像



绘制图像轮廓 重建后的图像



阈值分割图像



边缘分割图像



GrabCut分割图像

**六、实验小结**