# 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： 自动化4班**

**姓名： 班级序号： 同组人：**

**实验台号： 实验日期： 2024.5.17**

**教师签字：**

**实验三 图像分割与描述**

**一、算法原理**

**1、二值化图像**

根据图像局部区域的像素值动态调整阈值，以适应图像不均匀光照或对比度的情况。对图像中的每个像素，计算其周围邻域的平均值或加权平均值作为阈值进行二值化。

**2、形态学滤波**

形态学滤波的基本原理是使用结构元素（或内核）在图像上进行局部操作。具体来说，腐蚀操作可以缩小或消除物体边界，膨胀操作可以增加物体的大小或填充空洞，开运算可以消除小的噪点并保留物体的主要形状，闭运算可以填充物体的孔洞并平滑物体边界。形态学操作通常应用于二值化图像，以便更好地处理目标物体的形态和结构，同时减少图像中的噪声或不需要的细节。

**3、轮廓绘制**

将输入的图像转换为灰度图像，这一步是为了便于后续的边缘检测操作。灰度图像只包含亮度信息，而不考虑颜色，因此更适合用于边缘检测。使用Canny边缘检测算法对灰度图像进行处理，以便检测图像中的边缘。Canny边缘检测是一种经典的边缘检测算法，它可以有效地找到图像中的强边缘，同时抑制噪声和弱边缘。接下来，使用轮廓检测函数（如 cv2.findContours()）找到图像中的所有边缘轮廓。该函数会返回检测到的轮廓信息，包括每个轮廓的点集合和层级关系。创建一个新的图像副本用于绘制轮廓。在原始图像的副本上，使用 cv2.drawContours() 函数绘制检测到的轮廓。可以指定轮廓的颜色、线条粗细等参数来标识不同的轮廓。最后，显示带有轮廓的图像。绘制出的轮廓可以帮助分析图像中的物体形状、边界信息，以及后续的目标识别、分割等图像处理任务。

1. **重建边界**

首先，从图像中提取轮廓。通常，可以使用Canny边缘检测等方法得到图像中的边缘轮廓。每个轮廓由一系列的点组成，这些点描述了物体的边界。对轮廓进行傅里叶描述子的计算。傅里叶描述子是一种将轮廓表示为频域信号的方法，通过对轮廓的坐标进行傅里叶变换得到一系列复数值。选择一定数量的傅里叶描述子，通常是前几个描述子，以保留轮廓的主要特征。这些描述子可以用于重建轮廓，将频域表示转换回空域坐标。

使用傅里叶反变换（或逆傅里叶变换），将选定的傅里叶描述子转换为实际的轮廓点坐标。这样就得到了轮廓的重建结果。对重建的轮廓进行降采样，以减少点的数量并简化轮廓的表示。这样做可以在保留重要特征的同时，减少数据量和计算复杂度。

1. **图像分割方法**

**基于阈值的分割**：该方法使用全局阈值将图像的灰度值转换为二值图像。首先选择一个阈值，然后将图像中的像素值与该阈值进行比较，大于阈值的像素设置为一个值（通常为白色），小于阈值的像素设置为另一个值（通常为黑色）。这种方法适用于图像中目标物体与背景的灰度值差异明显的情况。

**自适应阈值分割**：这种方法根据图像局部区域的像素值动态地调整阈值。它先将图像分成多个小区域，然后针对每个区域计算一个局部阈值，从而实现更精确的二值化。自适应阈值分割对于光照不均匀或对比度变化较大的图像效果更好。

**边缘检测分割**：边缘检测是一种检测图像中边缘的技术，常用的边缘检测算法包括Canny边缘检测。Canny算法通过检测图像中的强边缘并抑制噪声来实现准确的边缘检测。检测到的边缘可以进一步处理，如基于梯度幅值进行阈值处理，或者应用分水岭算法进行图像分割，以实现更精细的物体分割和识别。

**聚类分割**：聚类分割利用聚类算法（如K-Means算法）将图像中的像素分成多个簇或类别。聚类分割通常需要将图像数据转换成聚类模型可以处理的形式，然后根据像素的特征相似性将其分组。这种方法适用于需要根据像素值或特征来进行区域划分和分割的应用场景。

### 编程思路与程序流程

### Task 1: 图像灰度化和阈值分割

1. **图像灰度化：**

使用 cv2.imread(image\_url, 0) 读取图像，以灰度模式加载图像数据。

1. **阈值分割：**

调用 cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY) 对灰度图像进行阈值分割。参数 127 是阈值，大于阈值的像素设置为 255（白色），小于阈值的像素设置为 0（黑色）。返回阈值化后的二值图像 BWimage。

### Task 2: 形态学滤波

1. **读取并二值化图像：**

与 Task 1 中相同的步骤，读取图像并进行阈值化得到 BWimage。

1. **形态学滤波操作：**

定义形态学滤波的核 kernel。分别应用不同的形态学操作（开运算、闭运算、形态梯度、顶帽运算、黑帽运算）。

### Task 3: 绘制图像边缘线

1. **灰度化和边缘检测：**

将彩色图像转换为灰度图像。使用 Canny 边缘检测算法 cv2.Canny() 找到图像边缘。

1. **轮廓查找与绘制：**

使用 cv2.findContours() 查找图像中的轮廓。在原始图像上使用 cv2.drawContours() 绘制轮廓线。

### Task 4: 计算傅里叶描述子和重建边界

1. **计算傅里叶描述子：**

提取轮廓点并将其转换为复数表示。使用 np.fft.fft() 计算傅里叶描述子。

1. **重建边界：**

使用傅里叶逆变换 np.fft.ifft() 对描述子进行重建。对重建的边界点进行下采样，以减少点的数量。使用 cv2.drawContours() 绘制重建后的边界。

### Task 5: 图像分割方法

1. **基于阈值的分割：**

使用 cv2.threshold() 对图像进行全局阈值分割。

1. **自适应阈值分割：**

使用 cv2.adaptiveThreshold() 对图像进行自适应阈值分割。

1. **边缘检测分割：**

使用 Canny 边缘检测进行边缘检测。对梯度幅值进行阈值处理和使用分水岭算法进行图像分割。

### Task 6: 图像分割和可视化

**图像分割：**

使用 K-Means 聚类对图像进行分割。将图像数据重塑为聚类模型可以处理的形式。使用 KMeans() 拟合模型并获取分割标签。使用 plt.imshow() 绘制分割后的图像。

1. **算法各部分主要函数代码以及功能注释**

*# 1.将Image灰度化为gray对其进行阈值分割转换为BW；*image = cv2.imread(image\_url)  
ret,BWimage = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)  
cv2.namedWindow('result', cv2.WINDOW\_NORMAL)  
cv2.imshow('result', BWimage)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

*# 2.形态学滤波*image = cv2.imread(image\_url, 0)  
ret, BWimage = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)  
kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))  
LBimage1 = cv2.morphologyEx(BWimage, cv2.MORPH\_OPEN, kernel)  
LBimage2 = cv2.morphologyEx(BWimage, cv2.MORPH\_CLOSE, kernel)  
LBimage3 = cv2.morphologyEx(BWimage, cv2.MORPH\_GRADIENT, kernel)  
LBimage4 = cv2.morphologyEx(BWimage, cv2.MORPH\_TOPHAT, kernel)  
LBimage5 = cv2.morphologyEx(BWimage, cv2.MORPH\_BLACKHAT, kernel)  
A = cv2.hconcat([LBimage1, LBimage2, LBimage3 ])  
B = cv2.hconcat([LBimage4, LBimage5, BWimage])  
C =cv2.vconcat([A, B])  
cv2.namedWindow('result', cv2.WINDOW\_NORMAL)  
cv2.imshow('result', C)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

*# 3. 绘制边缘线*image = cv2.imread(image\_url)  
*# 将图像转换为灰度图像*gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
  
*# 使用Canny边缘检测*edges = cv2.Canny(gray\_image, 300, 800)  
  
*# 寻找边缘的轮廓*contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
*# 创建一个新的图像副本用于绘制边缘线*image\_with\_edges = image.copy()  
  
*# 在原始图像上绘制红色边缘线*cv2.drawContours(image\_with\_edges, contours, -1, (0, 0, 255), 2)  
  
*# 显示带有边缘线的图像*cv2.imshow('Image with Red Edges', image\_with\_edges)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

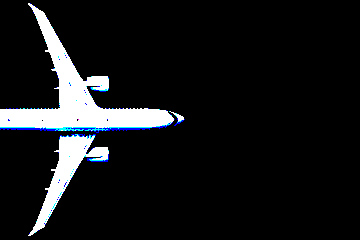
*# Task4 计算各区域边界点的傅里叶描绘子并用四分之一点重建边界;*def calculate\_fourier\_descriptors(contour, num\_descriptors):  
 *# 提取边界点的复数表示* complex\_contour = contour[:, 0, 0] + 1j \* contour[:, 0, 1]  
  
 *# 计算傅里叶描述子* descriptors = np.fft.fft(complex\_contour)  
  
 *# 仅保留指定数量的描述子* descriptors = descriptors[:num\_descriptors]  
  
 return descriptors  
  
def reconstruct\_contour(descriptors, num\_points):  
 *# 使用傅里叶描述子重建边界* reconstructed\_contour = np.fft.ifft(descriptors).real  
  
 *# 将复数表示转换为点坐标* reconstructed\_contour = np.array([[int(np.round(pt.real)), int(np.round(pt.imag))] for pt in reconstructed\_contour])  
  
 *# 确保重建的边界点数量足够大，避免步长为零的情况* if len(reconstructed\_contour) < num\_points:  
 num\_points = len(reconstructed\_contour) *# 使用重建边界的全部点  
  
 # 下采样边界点* step = len(reconstructed\_contour) // num\_points  
 reconstructed\_contour = reconstructed\_contour[::step]  
  
 return reconstructed\_contour  
  
*# 读取图像*image = cv2.imread(image\_url)  
  
*# 转换为灰度图像并执行Canny边缘检测*gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
edges = cv2.Canny(gray\_image, 300, 800)  
  
*# 寻找边缘的轮廓*contours, \_ = cv2.findContours(edges, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
*# 提取第一个轮廓的边界点*contour = contours[0]  
  
*# 计算傅里叶描述子（取前10个描述子）*num\_descriptors = 10  
descriptors = calculate\_fourier\_descriptors(contour, num\_descriptors)  
  
*# 重建边界（取50个重建点）*num\_reconstructed\_points = 50  
reconstructed\_contour = reconstruct\_contour(descriptors, num\_reconstructed\_points)  
  
*# 在图像上绘制重建的边界*reconstructed\_image = image.copy()  
cv2.drawContours(reconstructed\_image, [reconstructed\_contour], -1, (0, 0, 255), 2)  
  
*# 显示带有重建边界的图像*cv2.imshow('Reconstructed Contour', reconstructed\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

*# Task5 分割方法  
# 读取图像并灰度化*image = cv2.imread(image\_url, cv2.IMREAD\_GRAYSCALE)  
  
*# 方法一：基于阈值的分割*\_, thresh1 = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)  
  
*# 方法二：自适应阈值分割*thresh2 = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 11, 2)  
  
*# 方法三：边缘检测分割*edges = cv2.Canny(gray\_image, 300, 800)  
  
*# 使用Sobel算子计算梯度*gradient\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=5)  
gradient\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=5)  
  
*# 计算梯度幅值和方向*gradient\_magnitude = np.sqrt(gradient\_x\*\*2 + gradient\_y\*\*2)  
gradient\_direction = np.arctan2(gradient\_y, gradient\_x)  
  
*# 对梯度幅值进行阈值处理*\_, sure\_fg = cv2.threshold(gradient\_magnitude, 0.7 \* gradient\_magnitude.max(), 255, 0)  
  
*# 膨胀操作*sure\_fg = np.uint8(sure\_fg)  
kernel = np.ones((3, 3), np.uint8)  
sure\_fg = cv2.dilate(sure\_fg, kernel, iterations=3)  
  
*# 背景区域的确定*unknown = cv2.subtract(sure\_fg, thresh1)  
  
*# 标记初始化*markers, num\_markers = ndimage.label(sure\_fg)  
markers[unknown == 255] = num\_markers + 1  
  
*# 分水岭算法*markers = cv2.watershed(cv2.cvtColor(image, cv2.COLOR\_GRAY2RGB), markers)  
  
*# 转换成CV\_8UC3格式*image\_rgb = cv2.cvtColor(image, cv2.COLOR\_GRAY2RGB)  
  
*# 在分水岭结果上标记边界*image\_rgb[markers == -1] = [255, 0, 0]  
  
*# 绘制四个子图*plt.figure(figsize=(12, 8))  
  
plt.subplot(2, 2, 1)  
plt.imshow(thresh1, cmap='gray')  
plt.title('Threshold Binary')  
  
plt.subplot(2, 2, 2)  
plt.imshow(thresh2, cmap='gray')  
plt.title('Adaptive Threshold')  
  
plt.subplot(2, 2, 3)  
plt.imshow(edges, cmap='gray')  
plt.title('Edge Detection')  
  
plt.subplot(2, 2, 4)  
plt.imshow(image\_rgb)  
plt.title('Watershed Segmentation')  
  
plt.show()

*# Task 6*def segment\_image(image, num\_segments):  
 *# 获取图像形状* height, width, \_ = image.shape  
  
 *# 将图像数据重塑为 (num\_pixels, num\_features) 的数组* image\_data = image.reshape((-1, 3)) *# 对于RGB图像，3代表三个通道  
  
 # 初始化K均值模型* kmeans = KMeans(n\_clusters=num\_segments, random\_state=0)  
  
 *# 在图像数据上拟合K均值模型* kmeans.fit(image\_data)  
  
 *# 获取每个像素的分割标签* segmentation = kmeans.labels\_  
  
 *# 将分割结果重塑回图像形状* segmented\_image = segmentation.reshape((height, width))  
  
 return segmented\_image  
  
*# 3. 可视化分割结果*def visualize\_segmentation(segmented\_image):  
 plt.figure(figsize=(8, 8))  
 plt.imshow(segmented\_image, cmap='viridis')  
 plt.axis('off')  
 plt.title('Segmented Image')  
 plt.show()  
  
image = cv2.imread(image\_url)  
image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
  
*# 设置分割数*num\_segments = 2  
  
*# 对图像进行分割*segmented\_image = segment\_image(image, num\_segments)  
  
*# 可视化分割结果*visualize\_segmentation(segmented\_image)

**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



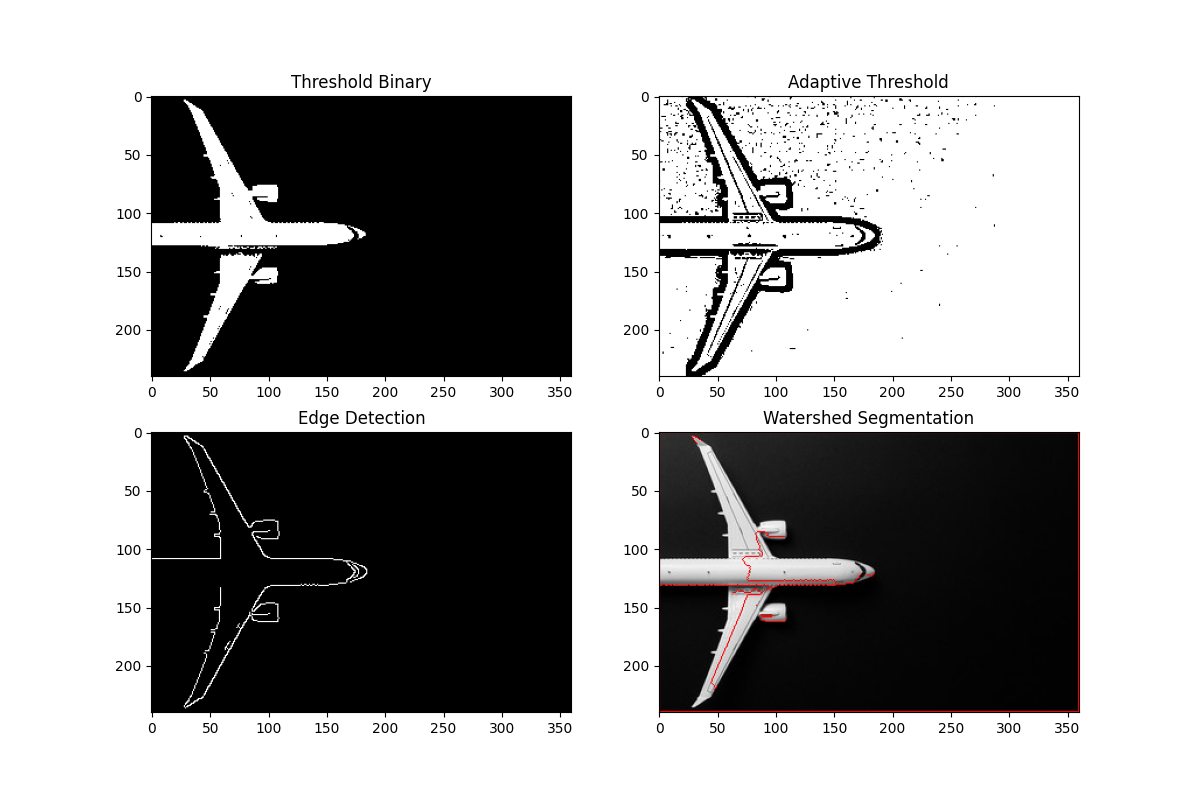
绘制图像轮廓 重建后的图像

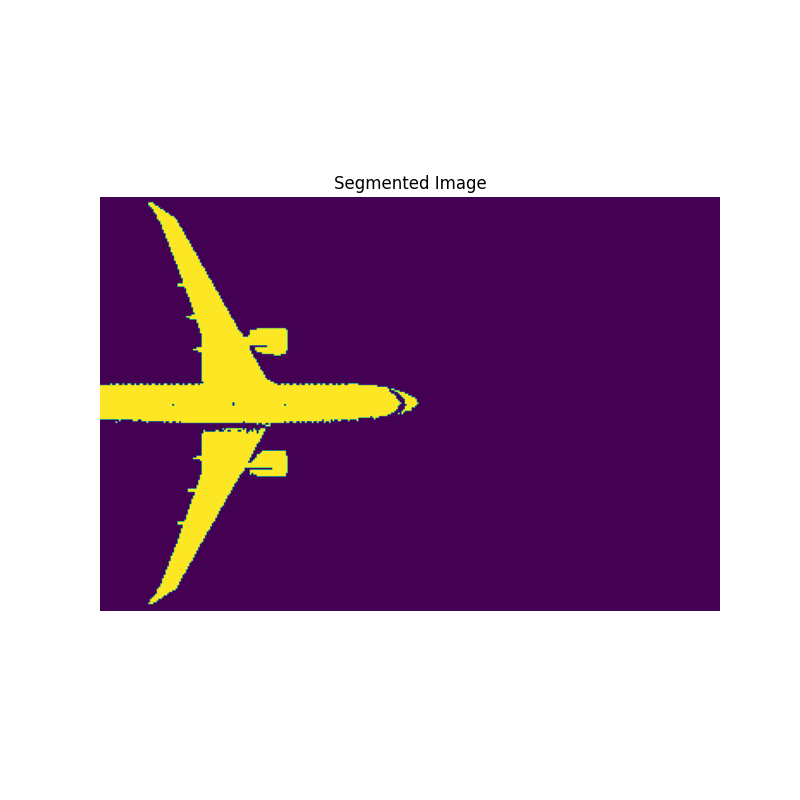


阈值分割图像



边缘分割图像





**六、实验小结**