**武 汉 工 程 大 学**

**计算机科学与工程学院**

**《图像处理与机器视觉》实验报告[3]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业班级 | | 智能科学与技术1班 | 实验时间 | | **2022年11月9日** | | |
| 学生学号 | | 2005120203 | 实验地点 | | 机房3 | | |
| 学生姓名 | | 方成杰 | 指导教师 | | 闵锋 | | |
|  | | | | | | | |
| 实验项目 | | 数米粒实验 | | | | | |
| 实验类别 | | 综合性 | 实验学时 | | **4** | | |
| 实验目的及要求 | | 1. 掌握图像分割的基本原理和方法； 2. 通过形态学操作去除图像背景干扰；   3. 通过阈值分割将去背景图像转变为二值图像；  4. 找出米粒的连通域，数出米粒的数目；  5. 找出这些米粒中最大的面积和周长，用矩形框出来。 | | | | | |
| 成 绩 评 定 表 | | | | | | | |
| 类 别 | | 评 分 标 准 | | 分值 | | 得分 | 合 计 |
| 上机表现 | | 按时出勤、遵守纪律  认真完成各项实验内容 | | **30**分 | |  |  |
| 报告质量 | 程序代码规范、功能正确  填写内容完整、体现收获 | | | **70**分 | |  |
| **说明：**  **评阅教师：**  **日 期： 年月日** | | | | | | | |
| 实 验 内 容 | | | | | | | |
| 1. 实现图像的去背景，将原始图像，背景图像和去背景图像保存下来，与核心代码一起粘贴到此：   def main():  img = cv2.imread("rice.png", 0)  show\_img\_plt([img, rice(img.copy()), rice(img.copy())\*img], ["Origin", "OTSU Thresh", "Remove the background"])  def rice(img):  kernels = np.ones((5, 5), np.uint8)  # 腐蚀  img\_erode = cv2.erode(img, kernels, iterations=5)  # 膨胀  img\_dilation = cv2.dilate(img\_erode, kernels, iterations=5)  # 阴影修正  img = img - img\_dilation  thresh = OTSU(img)  dst = cv2.threshold(img, thresh, 255, cv2.THRESH\_BINARY)[1]  return dst  def OTSU(img):  maxCore = 0  threshold = 0  # img = histogram\_equalization(img)  H, W = img.shape  histogram = np.zeros(256)  for i in range(H):  for j in range(W):  histogram[img[i, j]] += 1  averageAll = np.sum(np.dot(histogram, np.array([n for n in range(256)])))  averageAll = averageAll / np.sum(np.array([n for n in range(256)]))  # 找令类间方差最大的k值  for i in range(1, 255):  # C1均值  mean1 = np.sum(np.dot(histogram[0:i], np.array([n for n in range(i)])))  mean1 = mean1 / np.sum(histogram[0:i])  # C2均值  mean2 = np.sum(np.dot(histogram[i:256], np.array([n for n in range(i, 256)])))  mean2 = mean2 / np.sum(histogram[i:256])  # 公式计算  score = sum(histogram[0:i]) \* ((averageAll - mean1) \*\* 2) + sum(histogram[i:256]) \* ((averageAll - mean2) \*\* 2)  if maxCore < score: # 记录最大值  maxCore = score  threshold = i  return threshold  2. 用迭代阈值方法进行图像分割，将最终的阈值和对应的分割图像，与核心代码一起粘贴到此：  def main():  img = cv2.imread("wit2.jpg", 0)  # 计算灰度平均值  initThreshold = np.mean(img)  # 阈值迭代  thresh = Iterate\_Thresh(img, initThreshold, 50)  dst = cv2.threshold(img, thresh, 255, cv2.THRESH\_BINARY)[1]  show\_img\_plt([img, dst], ["Origin", "Iterate Thresh"])  def Iterate\_Thresh(img, init\_threshold, max\_iter\_times=20, target=1):  mask1, mask2 = (img > init\_threshold), (img <= init\_threshold)  T1 = np.sum(mask1 \* img) / np.sum(mask1)  T2 = np.sum(mask2 \* img) / np.sum(mask2)  T = (T1 + T2) / 2  if abs(T - init\_threshold) < target or max\_iter\_times == 0:  return T  return Iterate\_Thresh(img, T, max\_iter\_times - 1)  3. 用OTSU方法进行图像分割，将最终的阈值和对应的分割图像，与核心代码一起粘贴到此：  def OTSU(img):  maxCore = 0  threshold = 0  H, W = img.shape  histogram = np.zeros(256)  for i in range(H):  for j in range(W):  histogram[img[i, j]] += 1  averageAll = np.sum(np.dot(histogram, np.array([n for n in range(256)])))  averageAll = averageAll / np.sum(np.array([n for n in range(256)]))  # 找令类间方差最大的k值  for i in range(1, 255):  # C1均值  mean1 = np.sum(np.dot(histogram[0:i], np.array([n for n in range(i)])))  mean1 = mean1 / np.sum(histogram[0:i])  # C2均值  mean2 = np.sum(np.dot(histogram[i:256], np.array([n for n in range(i, 256)])))  mean2 = mean2 / np.sum(histogram[i:256])  # 公式计算  score = sum(histogram[0:i]) \* ((averageAll - mean1) \*\* 2) + sum(histogram[i:256]) \* ((averageAll - mean2) \*\* 2)  # 记录最大值  if maxCore < score:   maxCore = score  threshold = i  return threshold  4. 实现数米粒功能，将米粒的颗数，最大米粒的面积，周长，在图像中的位置坐标输出，与核心代码一起粘贴到此：  def rice(img, img\_color):  kernels = np.ones((5, 5), np.uint8)  # 腐蚀  img\_erode = cv2.erode(img, kernels, iterations=5)  # 膨胀  img\_dilation = cv2.dilate(img\_erode, kernels, iterations=5)  # 阴影修正  img = img - img\_dilation  thresh = OTSU(img)  dst = cv2.threshold(img, thresh, 255, cv2.THRESH\_BINARY)[1]  contours, hierarchy = cv2.findContours(dst, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  cv2.drawContours(img\_color, contours, -1, (0, 0, 255), 1)  max\_perimeter = 0  max\_area = 0  print("米粒数:" + str(len(contours)))  for i in contours:  max\_area = max(cv2.contourArea(i), max\_area)  max\_perimeter = max(cv2.arcLength(i, closed=True), max\_perimeter)  print("最大面积:" + str(max\_area))  print("最大周长:" + str(max\_perimeter))  return img\_color   1. 在黑色背景下用白色画出所有米粒的轮廓，并将最大面积米粒的轮廓用红色显示，将结果图像与核心代码一起粘贴到此：   def rice(img, img\_color):  black = np.zeros(img\_color.shape, np.uint8)  kernels = np.ones((5, 5), np.uint8)  # 腐蚀  img\_erode = cv2.erode(img, kernels, iterations=5)  # 膨胀  img\_dilation = cv2.dilate(img\_erode, kernels, iterations=5)  # 阴影修正  img = img - img\_dilation  thresh = OTSU(img)  dst = cv2.threshold(img, thresh, 255, cv2.THRESH\_BINARY)[1]  contours, hierarchy = cv2.findContours(dst, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)  cv2.drawContours(black, contours, -1, (255, 255, 255), 1)  max\_area = 0  max\_index = 0  for i in range(len(contours)):  if cv2.contourArea(contours[i]) > max\_area:  max\_index = i  max\_area = max(cv2.contourArea(contours[i]), max\_area)  cv2.drawContours(black, contours[max\_index], -1, (0, 0, 255), 1)  return black   1. 分析去背景和不同分割方法对数米粒结果的影响。   迭代阈值方法在当目标和背景相关的直方图模式之间存在一个非常清晰的波谷时十分有效。  OTSU运用则概率论的方法，完全基于图像的直方图得出最佳阈值。也因为这点，直方图很少依赖目标和背景的相对大小，小目标会受到直方图大波峰的抑制，或当光照不均匀，会对暗区域识别出错等，此时进行阴影修正能得到更好的效果，上面的实验我普遍采用的方法就是基于开操作的修正。 | | | | | | | |