《数字图像处理》实验报告

姓名：孔馨怡 学号：22122128

实验 1

一. 任务 1

a) 核心代码及讲解：

- 问题概要：

  我将问题分成一下几个步骤：

1. 提取图片并处理
2. 分离每个字符所在区域并适当存储每个字符的区域图片
3. 目标文字的输入以及目标图片的生成
4. 美化部分（美化背景颜色、文字间隔等）

- 代码分段讲解：

提取图片并处理

```
import cv2
import numpy as np

# 读取&转灰度
image = cv2.imread('letter.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 应用阈值，将图像转换为二值图像
_, thresh = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY_INV)
```

Tips：如果没有转化为二值图像则无法使用 findContours 函数提取轮廓

分离每个字符所在区域并适当存储每个字符的区域图片

```
# ----------------------------提取 char 轮廓--------------------------------
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 创建一个空白图像来绘制轮廓（选择与原始图像相同大小）
contour_image = np.ones_like(image) * 255  # 白色背景图像
```

```python
# 查看轮廓提取情况
cv2.drawContours(contour_image, contours, -1, (0, 0, 255), 2)
cv2.imshow('Contours', contour_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
# ------------------------------根据轮廓截取字符图像并存储--------------------------
characters = {}
max_width = 0
max_height = 0
# 遍历每个轮廓并裁剪出对应的图像区域
for i, contour in enumerate(contours):
    # 获取轮廓的边界框
    x, y, w, h = cv2.boundingRect(contours[i])

    # 过滤掉太小的轮廓
    if w < 10 or h < 10:
        continue

    # 从原始图像中裁剪出轮廓区域
    cropped_image = image[y:y + h, x:x + w]
    # 显示裁剪后的图像
    cv2.imshow(f'Cropped Image {i + 1}', cropped_image)
    print("输入该字符是哪个？（输小写字母和阿拉伯数字） (回车跳过此字符): ")
    key = cv2.waitKey(0)  # 等待用户按键

    # 判断用户输入的键
    if key == 13:  # 如果按下回车键，跳过当前字符
        cv2.destroyWindow(f'Cropped Image {i + 1}')
        continue

    max_width = max(max_width, cropped_image.shape[1])  # 求最大宽度
    max_height = max(max_height, cropped_image.shape[0])  # 求最大高度
    # 将用户输入的键作为索引键存储裁剪的图像
    characters[chr(key)] = cropped_image
    cv2.destroyWindow(f'Cropped Image {i + 1}')

# 查看存储效果，这里我用的是A 字母 稍微看一下效果就可以了
cv2.imshow('A', characters['a'])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Tips：轮廓提取不够准确要滤去较小的错误片段，且没有识别功能所以手动输入 char 的值并作为索引保存，这样的话想用哪个 char 更加方便。需要注意的是 此处裁剪的按照轮廓的矩形区域，裁剪更完整的字体

目标文字的输入以及目标图片的生成

```
# ----------------------生成新的图片----------------------------
# 目标文字设置
id = input('输入学号：')  # 用户输入目标字符
name = input('输入名字拼音：')

# 设置字符间隔和行间距
char_spacing = 5  # 字符之间的间隔
line_spacing = 5  # 行间距

# 计算每个字符的宽度之和 目的是为了使整张图片的宽度更加美观合理
def calculate_total_width(string):
    total_width = 0
    for char in string:
        if char in characters:
            total_width += characters[char].shape[1]  # 累加字符宽度
        else:
            print(char, '字符未找到')
    return total_width

# 计算新图像的总宽度和高度
total_width = max(calculate_total_width(id),calculate_total_width(name)) + (len(id) -
1) * char_spacing + char_spacing * 4
total_height = max_height * 2 + line_spacing * 2
print('total_width:', total_width)
print('total_height:', total_height)

# 选择原图中的一个区域作为背景颜色的参考
reference_region = image[0:10, 0:10]  # 选取原图的左上角 10x10 区域（该区域为背景原色）
background_color = np.mean(reference_region, axis=(0, 1)).astype(int)  # 计算平均颜色

# 确保背景颜色为uint8
background_color = np.clip(background_color, 0, 255).astype(np.uint8)

# 创建一个空白的背景图像（使用提取的背景颜色）
new_image = np.ones((total_height, total_width, 3), dtype=np.uint8) * background_color

# ---------------------------------拼接第一行学号---------------------------------
current_x = char_spacing  # 从字符间隔开始
current_y = char_spacing  # 从字符间隔开始
for char in id:
    if char in characters:
        char_image = characters[char]  # 获取字符图像
        char_height = char_image.shape[0]
```

```
        char_width = char_image.shape[1]

        # 将字符图像粘贴到新图像的指定位置
        if char_width > (total_width - current_x):
            char_image = cv2.resize(char_image, (total_width - current_x, char_height))

        new_image[current_y:current_y + char_height, current_x:current_x + char_width] =
char_image
        current_x += char_width + char_spacing  # 更新x坐标 加上字符间距
    else:
        print(char, '字符未找到')

# ------------------------------拼接第二行名字------------------------------
current_x = char_spacing  # 重置x坐标
current_y += max_height + line_spacing  # 更新y坐标 加上行间距

for char in name:
    if char in characters:
        char_image = characters[char]  # 获取字符图像
        char_height = char_image.shape[0]
        char_width = char_image.shape[1]

        # 将字符图像粘贴到新图像的指定位置
        if char_width > (total_width - current_x):
            char_image = cv2.resize(char_image, (total_width - current_x, char_height))

        new_image[current_y:current_y + char_height, current_x:current_x + char_width] =
char_image
        current_x += char_width + char_spacing  # 更新x坐标 加上字符间距
    else:
        print(char, '字符未找到')

# 显示拼接完成的图像
cv2.imshow('New Image', new_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 保存结果
cv2.imwrite('output_image.png', new_image)
```

Tips：将目标文字设置为输入的模式，更加通用。并且在输入之后计算目标字符串的所涉及到的字符宽度之和，结合字符高度和设置的间距来开创一个画布并且设置背景颜色

为后续粘贴字符图片做准备。而后顺序粘贴每个小字符区域图片到画布上，并且更新坐标以继续粘贴。

美化部分（美化背景颜色、文字间隔等）

此部分在上述代码中用高亮显示，可查看，美化内容概括如下：

- 为行和字符之间设置间距
- 为空白的画布设置颜色背景，保证整个图片不突兀 更统一

  这里是选取原图片的某个区域来计算平均颜色来运用
- 在粘贴图片的时候不强制将字符图片大小调整到一致，而是获取实际宽度来一一调整粘贴，这样更加美观

b） **实验结果截图**

显示框截图如下：

生成的原图如下：

c） **实验小结**

基本思想：不想死板地将每个要用的字母截取并粘贴，所以选择识别轮廓后存储所有的字符图片，这样更加通用

结果分析：在最一开始实现功能以后，为了效果更加完美 添加了很多很多的细节和美化 使得最后的成果非常美观 且通用快捷