《数字图像处理》实验报告

姓名：孔馨怡 学号：22122128

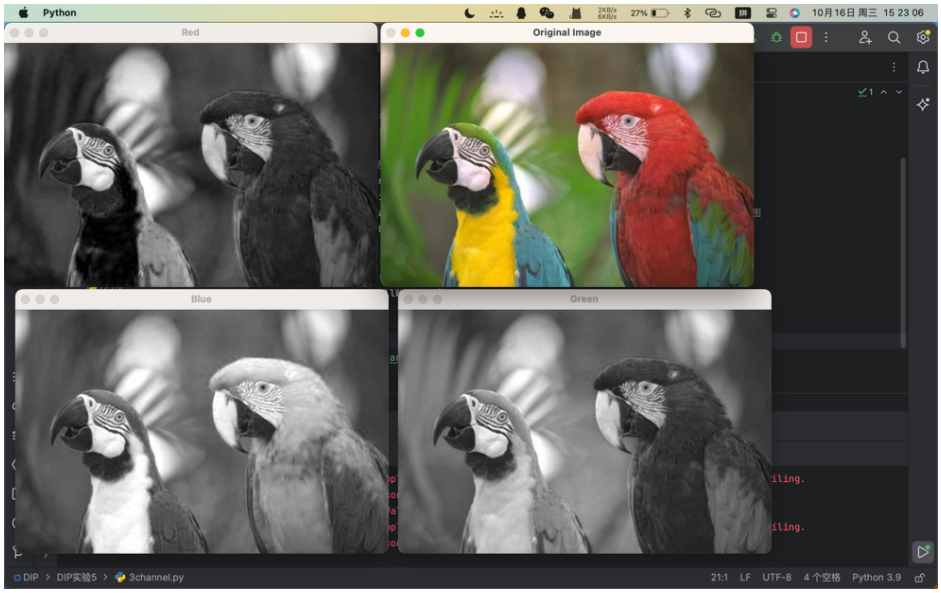
实验 5

一. 任务 1：将 araras.jpg 的三通道拆开用灰度图像显示

a) 核心代码：

```python
def channel_to_gray(filename):
    image = cv2.imread(filename, cv2.IMREAD_COLOR)

    # RGB 通道的名字
    channels = ['Red', 'Green', 'Blue']

    # 遍历 RGB 通道
    for i, channel in enumerate(channels):
        temp = image[:, :, i]  # 获取每个通道的图像
        gray_image = cv2.cvtColor(temp, cv2.COLOR_GRAY2BGR)  # 将单通道转为3 通道灰度图
        cv2.imshow(channel, gray_image)  # 使用通道名字作为窗口名

    cv2.imshow('Original Image', image)  # 显示原图
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return
```

b) 实验结果截图



c) 实验小结

基本思想：通过拆分彩色图像的 RGB 通道，分别提取并显示每个通道的灰度图像，以观察各通道对图像的影响。

结果分析：通过展示红、绿、蓝通道的灰度图，可以直观地看到每个通道的亮度分布及其对图像整体色彩的贡献。就像平时用 p 图软件调色的时候调整某一颜色的感觉。

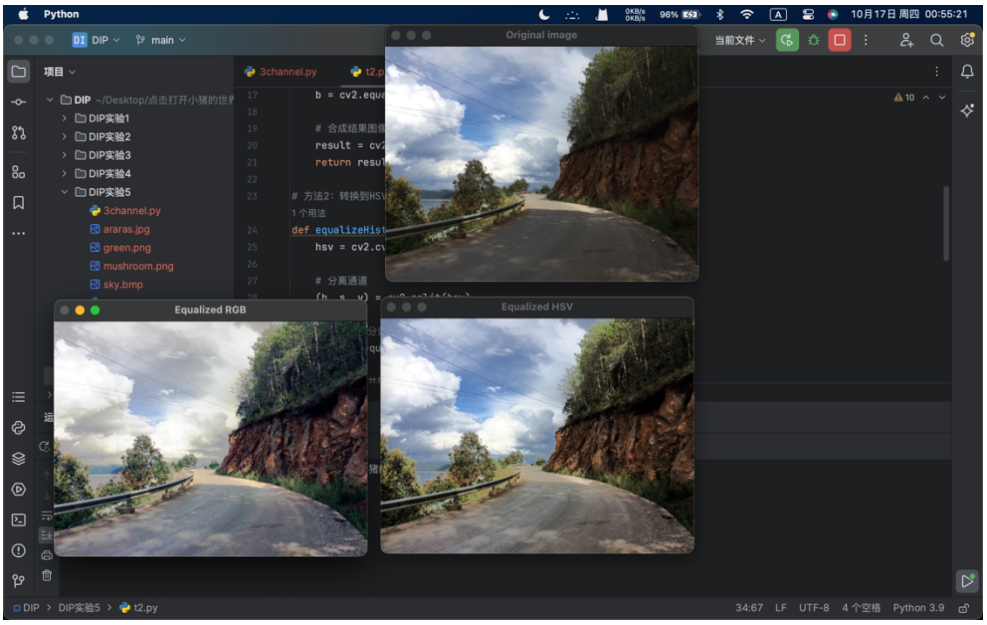
二. 任务 2：实现彩色图像的直方图均衡化（可以用 OpenCV 自带函数）两个方法

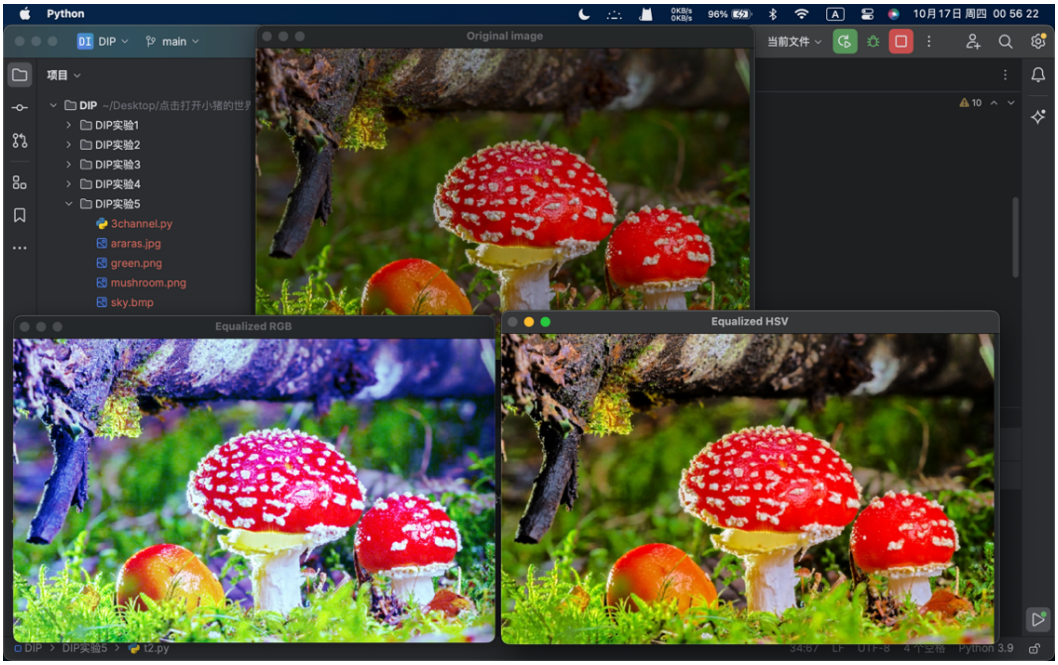
a) 核心代码：

```
# 方法 1：对 RGB 通道分别做直方图均衡化再合成
def equalizeHist_rgb(image):
    # 拆分 RGB 三通道
    (b,g,r) = cv2.split(image)

    # 分别做直方图均衡化
    r = cv2.equalizeHist(r)
    g = cv2.equalizeHist(g)
    b = cv2.equalizeHist(b)

    # 合成结果图像
    result = cv2.merge((b,g,r))
    return result

# 方法 2：转换到 HSV 空间，仅对亮度分量 V 用直方图均衡化，再转换回 RGB
def equalizeHist_hsv(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # 分离通道
    (h, s, v) = cv2.split(hsv)

    # 仅对亮度分量 V 用直方图均衡化
    v = cv2.equalizeHist(v)

    # 合成图像并转回成 RGB
    result = cv2.cvtColor(cv2.merge((h, s, v)), cv2.COLOR_HSV2BGR)
    return result

# 测试函数
def test_image(filename):
    image = cv2.imread(filename, cv2.IMREAD_COLOR)
    rgb = equalizeHist_rgb(image)
    hsv = equalizeHist_hsv(image)
    cv2.imshow("Equalized RGB", rgb)
```

```
    cv2.imshow("Equalized HSV", hsv)
    cv2.imshow("Original image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return
```

b) 实验结果截图





a) 实验小结（代码思想和关键步骤在以上的代码注释中已用高亮标出，可结合代码查看）

基本思想：两种方法的一个基本步骤：先分离轨道，对需要的轨道做直方图均衡化，合并轨道，有需要的要进行转变（eg：HSV to RGB）大概这样一个过程之后可以比较出结果。

结果分析：RGB 通道均衡化效果明显提升了色彩的鲜明度，而 HSV 方法则保留了颜色的饱和度，使得图像在均衡后更自然。

三. 任务 3：将 green 图片中的人物抠出，并融合到 tree 图片中，力求融合结果自然

a) 核心代码：

```
# 扣人物的函数
def extract_person(green_filename):
    # 提取照片
    image = cv2.imread(green_filename, cv2.IMREAD_COLOR)
    # 转化为hsv
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # 定义绿色的HSV 阈值范围
    # H（Hue）色调：表示颜色的基本色调，范围为 0 到 180，绿色的色调大概在 35 到 85 之间。
    lower_green = np.array([35, 40, 40])
    # 这是定义绿色的最低 HSV 值。35 表示色调的最小值，对应于绿色的下限。
    # 40 表示饱和度的最小值，表示提取饱和度至少为40 的绿色。40 表示亮度的最小值，提取亮度至少为40 的绿色部分。
    upper_green = np.array([85, 255, 255])
    # 这是定义绿色的最高HSV 值。85 表示色调的最大值，对应于绿色的上限。
    # 255 表示饱和度的最大值，表示可以提取饱和度为255 的绿色。255 表示亮度的最大值，表示可以提取最亮的绿色。

    # 提取绿色区域
    mask_green = cv2.inRange(hsv, lower_green, upper_green)

    # 和原图取反 留下人物
    mask_person = cv2.bitwise_not(mask_green)

    # 按照mask 在原图上把人物扣下来
    person = cv2.bitwise_and(image, image, mask=mask_person)
    return person, mask_green

# 把人加到背景上的函数
def merge_with_tree(background_filename, person, mask_green):
    # 提取
    background = cv2.imread(background_filename, cv2.IMREAD_COLOR)

    # 准备背景图，将背景区域提取出来，用来放置提取出来的人物
    rows1, cols1, _ = person.shape
    rows2, cols2, _ = background.shape
    # 计算要放置人物图像的起始行坐标
    y_start = rows2 - rows1

    # 提取一块person 大小的区域出来
    roi = background[y_start:rows2, 100:100 + cols1]
```

```
    # 把这个区域原图上人物区域扣掉
    roi = cv2.bitwise_and(roi, roi, mask=mask_green)
    # 然后加上人物的图像
    roi = cv2.add(person, roi)

    # 把原图这个区域替换为我们加了人物的
    background[y_start:rows2, 100:100 + cols1] = roi
    return background

# 主函数
def main():
    green_image_path = 'green.png'
    tree_image_path = 'tree.jpg'
    # 提取人物
    person, mask_background = extract_person(green_image_path)
    # 将人物融合到树的背景上
    result = merge_with_tree(tree_image_path, person, mask_background)
    # 显示结果
    cv2.imshow("Merged Image", result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

b) 实验结果截图

a) 实验小结（代码思想和关键步骤在以上的代码注释中已用高亮标出，可结合代码查看）

基本思想：

提取人物的函数（extract_person）：首先读取绿色背景图像并转换为 HSV 颜色空间，利用定义的 HSV 阈值范围提取绿色区域。（这里要注意具体的一个调整，不是完全的一个绿色，范围要给好）通过反转掩膜（mask），保留人物部分，最后使用按位与操作从原图中扣出人物。

融合人物与背景的函数（merge_with_tree）：在读取背景图像后，计算人物放置的位置，并提取该区域（设计 ROI 区域）。使用绿色掩膜将背景中的相应区域清除，然后将扣出的人物叠加到该区域，最终形成自然的融合效果（如果未取出就融合效果不自然）。

以上操作几乎都是用 `cv2.bitwise_not，cv2.bitwise_and` 等取反和合并的操作实现的。

结果分析：

成功地将人物自然地融入背景，背景与人物之间的融合较为自然，可以调整 ROI 区域来调整融合人物进背景的一个大小和区域。