# 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：图像色彩、代数、几何变换 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：**

**实验一 图像色彩、代数、几何变换**

**一、算法原理**

Task1

复制了原始图像,然后将绿色通道的值赋给新图像的蓝色通道,将红色通道的值赋给新图像的绿色通道

Task2

使用OpenCV库中的cv2.cvtColor()函数将彩色图像转换为灰度图像。灰度化的过程实际上是一种维度缩减,将原始图像的三个颜色通道(RGB)合并为单一的灰度通道。线性组合考虑了人眼对不同颜色通道的敏感程度,以产生最佳的灰度效果。

Task3

使用了cv2.getRotationMatrix2D()函数计算出旋转矩阵,cv2.warpAffine()函数将矩阵应用于原始图像,实现图像旋转。对于缩放操作使用cv2.resize()函数

Task4

进行基本的像素级别运算,包括图像拼接(np.concatenate())、加法(cv2.add())、减法(cv2.subtract())、乘法(cv2.multiply())和除法(cv2.divide())。对于除法运算,在除数上加上一个很小的浮点数,以避免除以零的情况发生。

Task6

实现了RGB到HSI颜色空间的转换,并对HSI空间中的色相(H)、饱和度(S)和强度(I)分量进行了变换,最后将变换后的HSI图像转换回RGB空间。HSI颜色空间更加接近人类对颜色的感知方式,其中色相描述了颜色本身,饱和度描述了颜色的纯度,强度描述了颜色的亮度。通过分别调整这三个分量,可以实现各种各样的颜色效果变换,在图像增强、风格迁移等领域有着重要应用。

1. **编程思路与程序流程**

**任务1: 交换 RGB 通道**

1. 首先读取一张名为 peppers.jpg 的图像。
2. 创建一个 pltImageShow 函数,用于显示原始图像和处理后的图像。
3. 使用 image.copy() 创建一个新的图像副本 swapped\_image。
4. 将原始图像的绿色通道复制到新图像的蓝色通道。
5. 将原始图像的红色通道复制到新图像的绿色通道。

**任务2: 转换为灰度图像**

1. 使用 cv2.cvtColor 函数将原始图像转换为灰度图像。
2. 调用 pltImageShow 函数,显示原始图像和灰度图像。

**任务3: 旋转和缩放灰度图像**

1. 定义一个 rotate\_image 函数,使用 cv2.getRotationMatrix2D 和 cv2.warpAffine 函数实现图像旋转。
2. 定义一个 resize\_image 函数,使用 cv2.resize 函数实现图像缩放。
3. 将灰度图像旋转 30 度,使用最近邻插值法,并显示结果。
4. 将灰度图像缩放 1.5 倍,使用立方插值法,并显示结果。

**任务4: 图像算术运算**

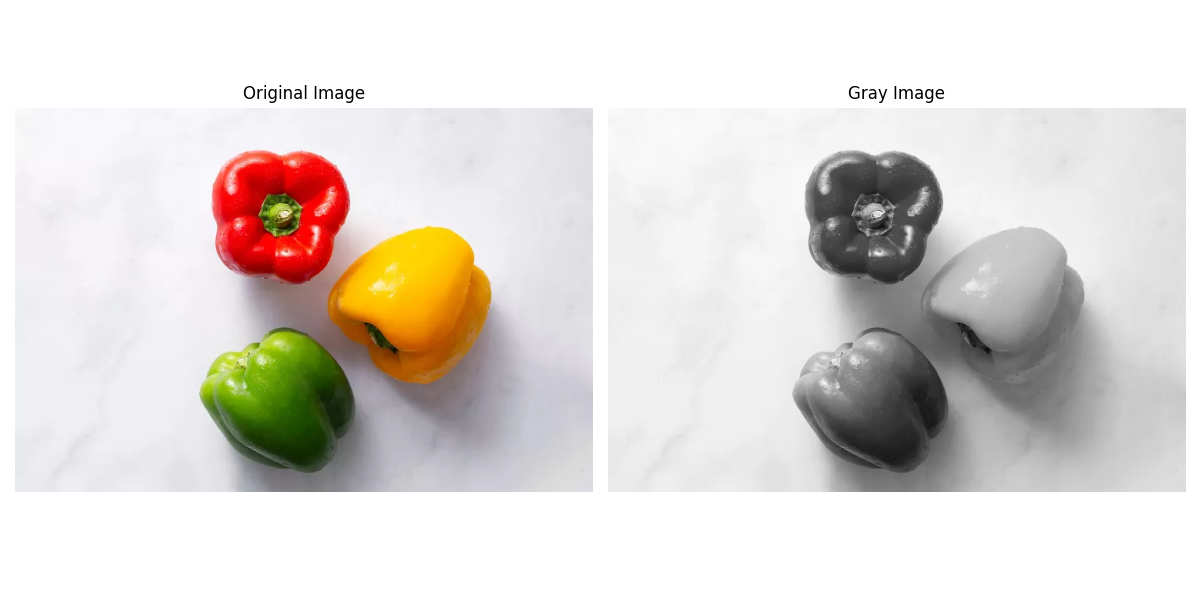
1. 读取另一张名为 lotus.jpg 的图像。
2. 确保两幅图像具有相同的尺寸。
3. 将两幅图像进行横向拼接,使用 np.concatenate 函数。
4. 对两幅图像进行像素级加法,使用 cv2.add 函数。
5. 对两幅图像进行像素级减法,使用 cv2.subtract 函数。
6. 对两幅图像进行像素级乘法,使用 cv2.multiply 函数。
7. 对两幅图像进行像素级除法,使用 cv2.divide 函数,避免除以零的情况。
8. 显示原始图像、拼接图像、加法结果、减法结果、乘法结果和除法结果。

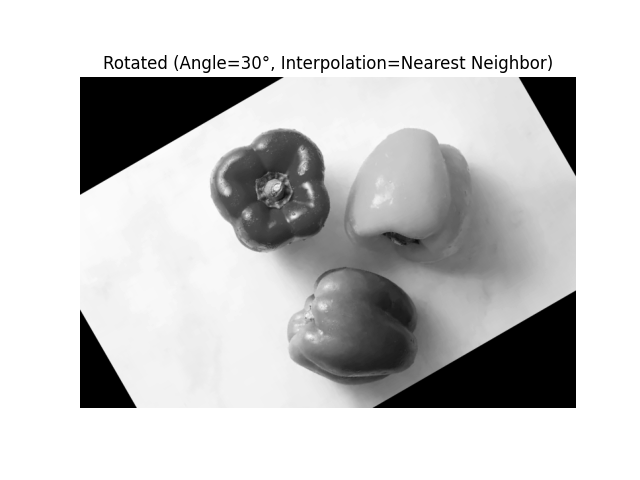
**任务6: HSI 颜色空间变换**

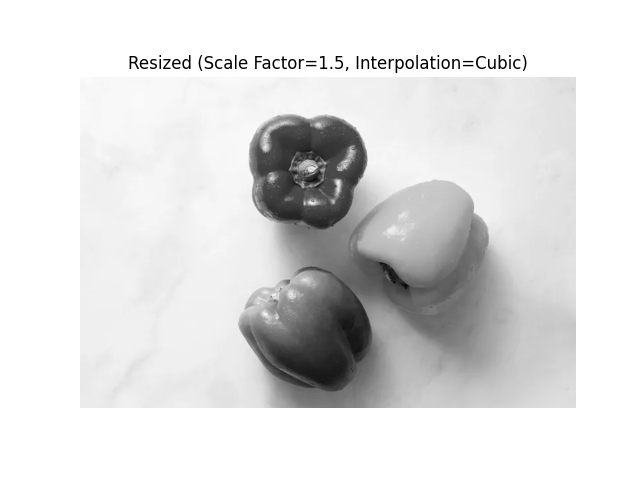
1. 定义一个 rgb\_to\_hsi 函数,将 RGB 图像转换为 HSI 颜色空间。
2. 定义一个 hsi\_to\_rgb 函数,将 HSI 图像转换回 RGB 颜色空间。
3. 将原始图像转换为 HSI 颜色空间。
4. 对 HSI 颜色空间中的各个通道进行变换:调整色相、增加饱和度和减小强度。
5. 将变换后的 HSI 图像转换回 RGB 颜色空间。
6. **算法各部分主要函数代码以及功能注释**
7. import cv2  
   import numpy as np  
   import matplotlib.pyplot as plt  
     
   image = cv2.imread('peppers.jpg')  
     
   def pltImageShow(image1, name1, image2, name2):  
    plt.figure(figsize=(12, 6))  
    plt.subplot(1, 2, 1)  
    plt.imshow(cv2.cvtColor(image1, cv2.COLOR\_BGR2RGB))  
    plt.title(name1)  
    plt.axis('off')  
    plt.subplot(1, 2, 2)  
    plt.imshow(cv2.cvtColor(image2, cv2.COLOR\_BGR2RGB))  
    plt.title(name2)  
    plt.axis('off')  
    plt.tight\_layout()  
    plt.show()  
     
     
   *# Task1*swapped\_image = image.copy()  
   swapped\_image[:, :, 0] = image[:, :, 1] *# 将绿色通道复制到蓝色通道*swapped\_image[:, :, 1] = image[:, :, 0] *# 将红色通道复制到绿色通道  
     
   # 显示原图和处理后的图像*pltImageShow(image, 'Original Image', swapped\_image, 'Swapped Red-Green Channels')  
     
   *# Task2*gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
   pltImageShow(image,'Original Image',gray\_image,'Gray Image')  
     
     
   *# task3  
     
   # 图像旋转函数*def rotate\_image(image, angle, interpolation=cv2.INTER\_LINEAR):  
    h, w = image.shape  
    *# 计算旋转中心* center = (w // 2, h // 2)  
    *# 获取旋转矩阵* M = cv2.getRotationMatrix2D(center, angle, 1.0)  
    *# 应用仿射变换* rotated\_image = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER\_LINEAR, borderMode=cv2.BORDER\_CONSTANT)  
    return rotated\_image  
     
   *# 图像放大函数*def resize\_image(image, scale\_factor, interpolation=cv2.INTER\_LINEAR):  
    h, w = image.shape  
    *# 计算新的图像尺寸* new\_h = int(h \* scale\_factor)  
    new\_w = int(w \* scale\_factor)  
    *# 使用指定的插值方法进行图像放大* resized\_image = cv2.resize(image, (new\_w, new\_h), interpolation=interpolation)  
    return resized\_image  
     
   *# 旋转灰度图像并显示*angle = 30  
   rotated\_gray\_image = rotate\_image(gray\_image, angle, interpolation=cv2.INTER\_NEAREST)  
   plt.imshow(rotated\_gray\_image, cmap='gray')  
   plt.title(f'Rotated (Angle={angle}°, Interpolation=Nearest Neighbor)')  
   plt.axis('off')  
   plt.show()  
     
   *# 缩放灰度图像并显示*scale\_factor = 1.5  
   resized\_gray\_image = resize\_image(gray\_image, scale\_factor, interpolation=cv2.INTER\_CUBIC)  
   plt.imshow(resized\_gray\_image, cmap='gray')  
   plt.title(f'Resized (Scale Factor={scale\_factor}, Interpolation=Cubic)')  
   plt.axis('off')  
   plt.show()  
     
     
   *# Task4  
     
   # 读取图像1和图像2*image1 = cv2.imread('peppers.jpg')  
   image2 = cv2.imread('lotus.jpg')  
     
   *# 确保两幅图像具有相同的尺寸*h, w, \_ = image1.shape  
   image2 = cv2.resize(image2, (w, h))  
     
   *# 图像拼接（横向）*concatenated\_image = np.concatenate((image1, image2), axis=1)  
     
   *# 图像加法（按通道）*added\_image = cv2.add(image1, image2)  
     
   *# 图像减法（按通道）*subtracted\_image = cv2.subtract(image1, image2)  
     
   *# 图像乘法（按通道）*multiplied\_image = cv2.multiply(image1, image2)  
     
   *# 图像除法（按通道）  
   # 避免溢出*image1\_float = image1.astype(np.float32)  
   image2\_float = image2.astype(np.float32)  
   divided\_image = cv2.divide(image1\_float, image2\_float + np.finfo(float).eps) *# 避免除以0  
     
   # 显示结果*plt.figure(figsize=(16, 12))  
     
   plt.subplot(2, 3, 1)  
   plt.imshow(cv2.cvtColor(image1, cv2.COLOR\_BGR2RGB))  
   plt.title('Image 1')  
   plt.axis('off')  
     
   plt.subplot(2, 3, 2)  
   plt.imshow(cv2.cvtColor(image2, cv2.COLOR\_BGR2RGB))  
   plt.title('Image 2')  
   plt.axis('off')  
     
   plt.subplot(2, 3, 3)  
   plt.imshow(cv2.cvtColor(concatenated\_image, cv2.COLOR\_BGR2RGB))  
   plt.title('Concatenated Images')  
   plt.axis('off')  
     
   plt.subplot(2, 3, 4)  
   plt.imshow(cv2.cvtColor(added\_image, cv2.COLOR\_BGR2RGB))  
   plt.title('Added Images')  
   plt.axis('off')  
     
   plt.subplot(2, 3, 5)  
   plt.imshow(cv2.cvtColor(subtracted\_image, cv2.COLOR\_BGR2RGB))  
   plt.title('Subtracted Images')  
   plt.axis('off')  
     
   plt.subplot(2, 3, 6)  
   plt.imshow(cv2.cvtColor(multiplied\_image, cv2.COLOR\_BGR2RGB))  
   plt.title('Multiplied Images')  
   plt.axis('off')  
     
   plt.tight\_layout()  
   plt.show()  
     
     
   *# Task 6*def rgb\_to\_hsi(rgb\_image):  
    *# 将 RGB 图像转换为浮点数类型* rgb\_image = rgb\_image.astype(np.float32) / 255.0  
     
    *# 提取 RGB 通道* r, g, b = rgb\_image[:, :, 0], rgb\_image[:, :, 1], rgb\_image[:, :, 2]  
     
    *# 计算强度（Intensity）* intensity = (r + g + b) / 3.0  
     
    *# 计算饱和度（Saturation）* min\_val = np.minimum.reduce([r, g, b])  
    saturation = 1 - min\_val / intensity  
    saturation = np.nan\_to\_num(saturation) *# 处理除以零产生的 NaN  
     
    # 计算色相（Hue）* numerator = 0.5 \* ((r - g) + (r - b))  
    denominator = np.sqrt((r - g) \*\* 2 + (r - b) \* (g - b))  
    hue = np.arccos(np.clip(numerator / denominator, -1, 1))  
     
    *# 根据 B 分量大于 G 分量的情况调整色相值* hue[b > g] = 2 \* np.pi - hue[b > g]  
     
    *# 将强度、饱和度和色相值限制在合适的范围内* intensity = np.clip(intensity, 0, 1)  
    saturation = np.clip(saturation, 0, 1)  
    hue = np.clip(hue / (2 \* np.pi), 0, 1)  
     
    *# 将 HSI 通道合并为一个图像* hsi\_image = np.stack([hue, saturation, intensity], axis=-1)  
     
    return hsi\_image  
     
   def hsi\_to\_rgb(hsi\_image):  
    *# 提取 HSI 通道* hue, saturation, intensity = hsi\_image[:, :, 0], hsi\_image[:, :, 1], hsi\_image[:, :, 2]  
     
    *# 将色相转换为弧度* hue = hue \* 2 \* np.pi  
     
    *# 计算 RGB 分量* r, g, b = np.zeros\_like(hue), np.zeros\_like(hue), np.zeros\_like(hue)  
     
    *# 第一批计算* for i in range(hue.shape[0]):  
    for j in range(hue.shape[1]):  
    if hue[i, j] < 2 \* np.pi / 3:  
    b[i, j] = intensity[i, j] \* (1 - saturation[i, j])  
    r[i, j] = intensity[i, j] \* (1 + (saturation[i, j] \* np.cos(hue[i, j]) / np.cos(np.pi / 3 - hue[i, j])))  
    g[i, j] = 3 \* intensity[i, j] - (r[i, j] + b[i, j])  
    elif hue[i, j] < 4 \* np.pi / 3:  
    hue[i, j] -= 2 \* np.pi / 3  
    r[i, j] = intensity[i, j] \* (1 - saturation[i, j])  
    g[i, j] = intensity[i, j] \* (1 + (saturation[i, j] \* np.cos(hue[i, j]) / np.cos(np.pi / 3 - hue[i, j])))  
    b[i, j] = 3 \* intensity[i, j] - (r[i, j] + g[i, j])  
    else:  
    hue[i, j] -= 4 \* np.pi / 3  
    g[i, j] = intensity[i, j] \* (1 - saturation[i, j])  
    b[i, j] = intensity[i, j] \* (1 + (saturation[i, j] \* np.cos(hue[i, j]) / np.cos(np.pi / 3 - hue[i, j])))  
    r[i, j] = 3 \* intensity[i, j] - (g[i, j] + b[i, j])  
     
    *# 将 RGB 分量合并为一个图像* rgb\_image = np.stack([r, g, b], axis=-1)  
     
    *# 将浮点数值转换为 0-255 范围内的整数* rgb\_image = (rgb\_image \* 255).astype(np.uint8)  
     
    return rgb\_image  
     
   *# 将 RGB 图像转换为 HSI 空间*hsi\_image = rgb\_to\_hsi(image)  
     
   *# 对 HSI 空间中的各通道进行变换*hsi\_image[:, :, 0] += 0.1 *# 调整色相*hsi\_image[:, :, 1] \*= 1.5 *# 增加饱和度*hsi\_image[:, :, 2] \*= 0.8 *# 减小强度  
     
   # 将变换后的 HSI 图像转换回 RGB 空间*transformed\_image = hsi\_to\_rgb(hsi\_image)  
   pltImageShow(image,"Original",transformed\_image,"Transformed HSI Image")

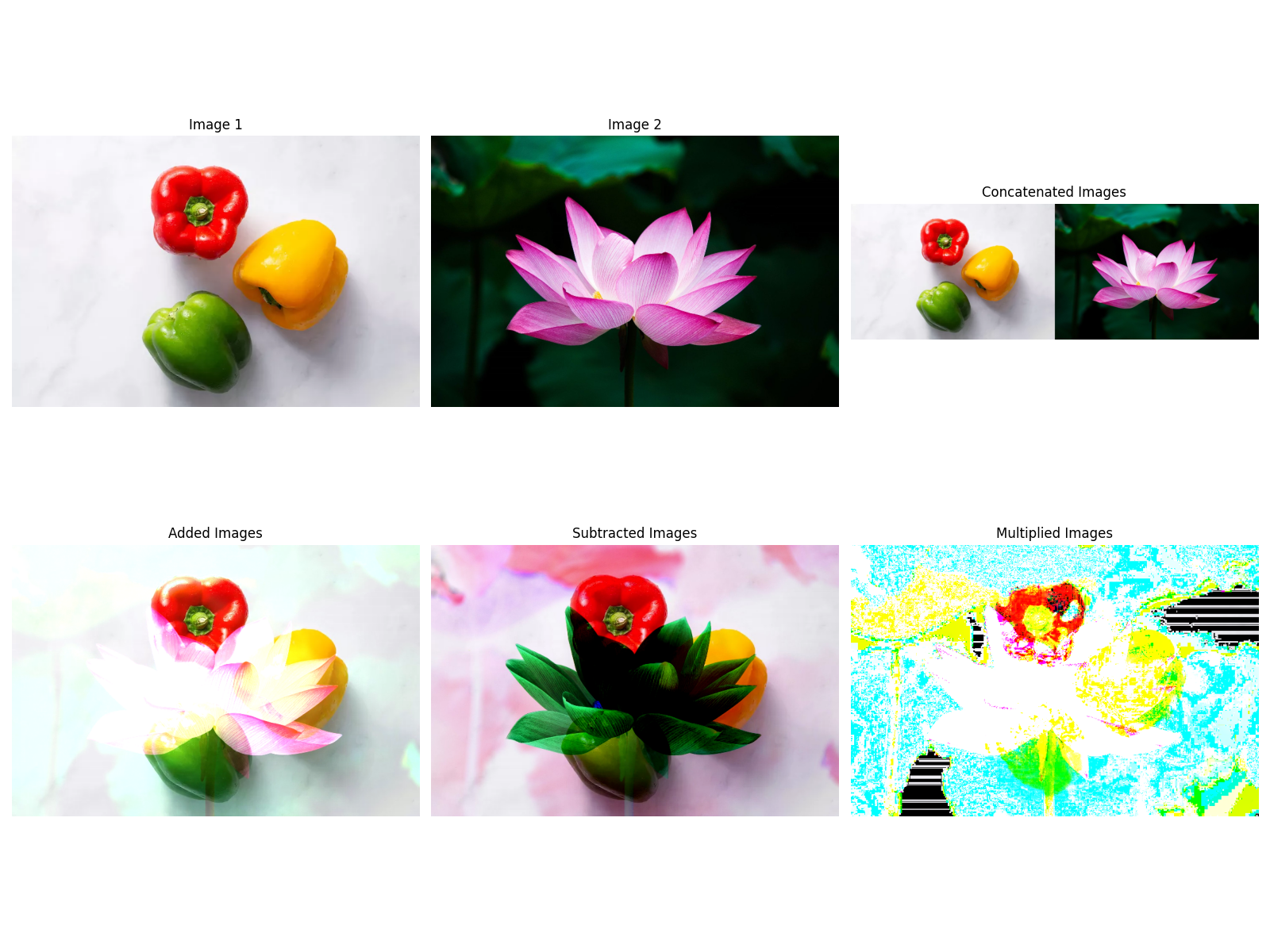
**四、调试过程与运行结果分析**













**六、实验小结**