# 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：图像增强 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：**

**实验二 图像增强**

1. **算法原理**

### Task 1: 灰度转换及灰度直方图计算

将彩色图像转换为灰度图是通过对彩色图像的像素进行加权平均来实现的，通常使用固定的加权系数。灰度直方图显示了灰度图中不同灰度级的像素频数，反映了图像的亮度分布情况。

### Task 2: 分段线性变换

分段线性变换对像素值在不同区间内进行不同的处理，本文中的变换是对大于100的像素值进行拉伸，使其范围更广，从而增强对比度。

### Task 3: 直方图均衡化

直方图均衡化通过重新分配图像的灰度值，使得结果图像的直方图均匀分布，从而增强图像的对比度。

### Task 4: 伪彩色处理

伪彩色处理将灰度图像转换为彩色图像，通过颜色映射增强视觉效果，使得图像中不同灰度级更易于区分。

### Task 5: 高斯噪声添加与平滑处理

添加高斯噪声通过在图像中加入服从正态分布的随机噪声来模拟现实中的噪声。高斯模糊则使用二维高斯核对图像进行卷积，实现图像的平滑处理和噪声的抑制。

### Task 6: 图像锐化与Sobel边缘检测

Sobel算子通过计算图像在水平方向和垂直方向的梯度来检测边缘。图像锐化则是通过将原图像与边缘信息线性组合，增强边缘部分的像素值，实现图像的锐化效果。

### Task 7: Canny边缘检测与形态学梯度

Canny边缘检测通过多级算法检测图像的边缘，包括噪声去除、梯度计算、非极大值抑制和双阈值检测。形态学梯度通过膨胀与腐蚀操作计算图像的边缘信息，进一步增强边缘效果。

### Task 8: 频域滤波

傅里叶变换将图像从空间域转换到频域，通过分析频域特性可以应用不同的滤波器。低通滤波器保留低频信息，去除高频噪声，而高通滤波器保留高频信息，突出图像细节。频域滤波通过掩码在频域中进行滤波操作，然后通过逆傅里叶变换将结果转换回空间域。

1. **编程思路与程序流程**

1.**灰度转换及灰度直方图**：

将彩色图像转换为灰度图，并计算和绘制其灰度直方图。

2.**分段线性变换**：

定义一个分段线性变换函数，对灰度图像应用该变换，并显示原图与变换后的图像。

3.**直方图均衡化**：

对灰度图像应用直方图均衡化，并显示原图与均衡化后的图像。

4.**伪彩色处理**：

对灰度图像应用伪彩色处理，并显示原图与伪彩色处理后的图像。

5.**高斯噪声添加与平滑处理**：

向灰度图像添加高斯噪声，并使用高斯模糊进行平滑处理，显示原图、噪声图和平滑处理后的图像。

6.**图像锐化与边缘检测**：

使用Sobel算子检测边缘，并结合边缘信息进行图像锐化，显示原图与锐化后的图像。使用Canny边缘检测和形态学梯度进一步进行图像锐化，并显示各步骤的图像。

7.**图像锐化与边缘检测**：

使用Sobel算子检测边缘，并结合边缘信息进行图像锐化，显示原图与锐化后的图像。使用Canny边缘检测和形态学梯度进一步进行图像锐化，并显示各步骤的图像。

8.**频域滤波**：

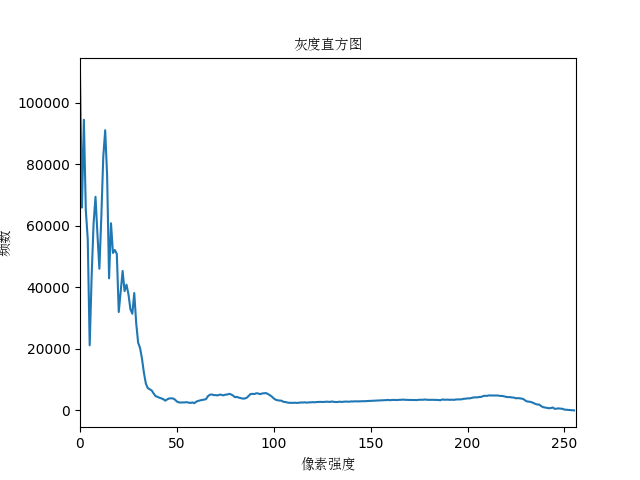
对灰度图像进行傅里叶变换，并应用低通和高通滤波器，最后将滤波结果转换回空间域，显示原图、低通滤波图和高通滤波图。

1. **算法各部分主要函数代码以及功能注释**

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
image\_url = "lotus.jpg"  
image = cv2.imread(image\_url)  
grayImage = cv2.imread(image\_url,0)  
  
  
def pltImageShow(image1,name1,image2,name2):  
 plt.figure(figsize=(12, 6))  
 plt.subplot(1, 2, 1)  
 plt.imshow(image1, cmap='gray')  
 plt.title(name1)  
 plt.axis('off')  
 plt.subplot(1, 2, 2)  
 plt.imshow(image2, cmap='gray')  
 plt.title(name2)  
 plt.axis('off')  
 plt.tight\_layout()  
 plt.show()  
  
  
  
*# 将图像转换为灰度图*gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
  
*# 显示灰度图像*plt.imshow(gray\_image, cmap='gray')  
plt.title('灰度图像',fontproperties="SimSun")  
plt.axis('off') *# 不显示坐标轴*plt.show()  
  
*# 计算灰度直方图*histogram = cv2.calcHist([gray\_image], [0], None, [256], [0, 256])  
  
*# 绘制灰度直方图*plt.figure()  
plt.title('灰度直方图',fontproperties="SimSun")  
plt.xlabel('像素强度',fontproperties="SimSun")  
plt.ylabel('频数',fontproperties="SimSun")  
plt.plot(histogram)  
plt.xlim([0, 256]) *# 设置 x 轴范围为 [0, 256]（256个灰度级）*plt.show()  
  
  
*# Task2*def piecewise\_linear\_transform(pixel\_value):  
 *# 定义分段线性变换函数* if pixel\_value < 100:  
 return pixel\_value *# 小于100的部分保持不变* else:  
 return int(255 \* (pixel\_value - 100) / (255 - 100))  
  
transformed\_image = np.vectorize(piecewise\_linear\_transform)(grayImage) *# 应用变换*pltImageShow(grayImage,'Original Image', transformed\_image,'Transformed Image') *# 显示结果  
  
  
# Task3*equalizedImage = cv2.equalizeHist(grayImage) *# 应用直方图均衡化*pltImageShow(grayImage,'Original Gray Image',equalizedImage,'Equalized Image')  
  
*# Task4*pseudocolorImage = cv2.applyColorMap(grayImage, cv2.COLORMAP\_JET)  
pltImageShow(grayImage,'Gray Image',pseudocolorImage,'Pseudocolor Image')  
  
*# Task5*mean = 0 *# 添加高斯噪声*sigma = 25 *# 调整噪声强度*noisyImage = grayImage + np.random.normal(mean, sigma, grayImage.shape)  
noisyImage = np.clip(noisyImage, 0, 255) *# 将像素值限制在0到255之间*noisyImage = noisyImage.astype(np.uint8)  
  
*# 使用高斯模糊进行平滑处理*blurredImage = cv2.GaussianBlur(noisyImage, (5, 5), 0) *# 调整核大小和标准差*plt.figure(figsize=(12, 6))  
plt.subplot(1, 3, 1)  
plt.imshow(grayImage, cmap='gray')  
plt.title('Original Gray Image')  
plt.axis('off')  
plt.subplot(1, 3, 2)  
plt.imshow(noisyImage, cmap='gray')  
plt.title('Noisy Image')  
plt.axis('off')  
plt.subplot(1, 3, 3)  
plt.imshow(blurredImage, cmap='gray')  
plt.title('Blurred Image')  
plt.axis('off')  
plt.tight\_layout()  
plt.show()  
  
*# task6*sobel\_x = cv2.Sobel(grayImage, cv2.CV\_64F, 1, 0, ksize=3) *# 水平方向*sobel\_y = cv2.Sobel(grayImage, cv2.CV\_64F, 0, 1, ksize=3) *# 垂直方向*sobel\_magnitude = np.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2) *# 计算梯度幅值*sobel\_magnitude = np.uint8(sobel\_magnitude / np.max(sobel\_magnitude) \* 255) *# 将结果转换为8位图像格式*alpha = 1.5 *# 原图像的权重*beta = -0.5 *# 边缘增强部分的权重*sharpened\_image = cv2.addWeighted(grayImage, alpha, sobel\_magnitude, beta, 0)  
pltImageShow(grayImage,'Original Image',sharpened\_image,'Sharpened Image')  
  
  
*# Task7*low\_threshold = 50 *# 设定低阈值和高阈值*high\_threshold = 150  
  
edges = cv2.Canny(grayImage, low\_threshold, high\_threshold) *# 使用Canny算子检测边缘*kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3)) *# 定义一个3x3的内核*morph\_gradient = cv2.morphologyEx(grayImage, cv2.MORPH\_GRADIENT, kernel) *# 计算形态学梯度  
  
# 进行图像锐化，alpha和beta是调节锐化强度的参数*sharpened\_image = cv2.addWeighted(grayImage, 1.0, edges, 1.0, 0)  
sharpened\_image = cv2.addWeighted(sharpened\_image, 1.0, morph\_gradient, 1.0, 0)  
  
*# 使用matplotlib显示结果*plt.figure(figsize=(10, 7))  
  
plt.subplot(2, 2, 1)  
plt.title('Original Image')  
plt.imshow(grayImage, cmap='gray')  
  
plt.subplot(2, 2, 2)  
plt.title('Canny Edges')  
plt.imshow(edges, cmap='gray')  
  
plt.subplot(2, 2, 3)  
plt.title('Morphological Gradient')  
plt.imshow(morph\_gradient, cmap='gray')  
  
plt.subplot(2, 2, 4)  
plt.title('Sharpened Image')  
plt.imshow(sharpened\_image, cmap='gray')  
  
plt.show()  
  
*# Task8  
# 对图像进行傅里叶变换*dft = cv2.dft(np.float32(grayImage), flags=cv2.DFT\_COMPLEX\_OUTPUT)  
dft\_shift = np.fft.fftshift(dft)  
  
*# 获取图像的大小*rows, cols = grayImage.shape  
crow, ccol = rows // 2 , cols // 2  
  
*# 创建低通滤波器掩码*mask = np.zeros((rows, cols, 2), np.uint8)  
r = 30 *# 半径*center = [crow, ccol]  
x, y = np.ogrid[:rows, :cols]  
mask\_area = (x - center[0]) \*\* 2 + (y - center[1]) \*\* 2 <= r\*r  
mask[mask\_area] = 1  
  
*# 创建高通滤波器掩码*mask\_high = 1 - mask  
  
*# 应用低通滤波器*fshift\_low = dft\_shift \* mask  
  
*# 应用高通滤波器*fshift\_high = dft\_shift \* mask\_high  
  
*# 低通滤波结果的逆傅里叶变换*f\_ishift\_low = np.fft.ifftshift(fshift\_low)  
img\_back\_low = cv2.idft(f\_ishift\_low)  
img\_back\_low = cv2.magnitude(img\_back\_low[:, :, 0], img\_back\_low[:, :, 1])  
  
*# 高通滤波结果的逆傅里叶变换*f\_ishift\_high = np.fft.ifftshift(fshift\_high)  
img\_back\_high = cv2.idft(f\_ishift\_high)  
img\_back\_high = cv2.magnitude(img\_back\_high[:, :, 0], img\_back\_high[:, :, 1])  
  
*# 使用matplotlib显示结果*plt.figure(figsize=(10, 7))  
  
plt.subplot(2, 2, 1)  
plt.title('Original Image')  
plt.imshow(grayImage, cmap='gray')  
  
plt.subplot(2, 2, 2)  
plt.title('Low Pass Filtered Image')  
plt.imshow(img\_back\_low, cmap='gray')  
  
plt.subplot(2, 2, 3)  
plt.title('High Pass Filtered Image')  
plt.imshow(img\_back\_high, cmap='gray')  
  
plt.show()

**四、调试过程与运行结果分析**



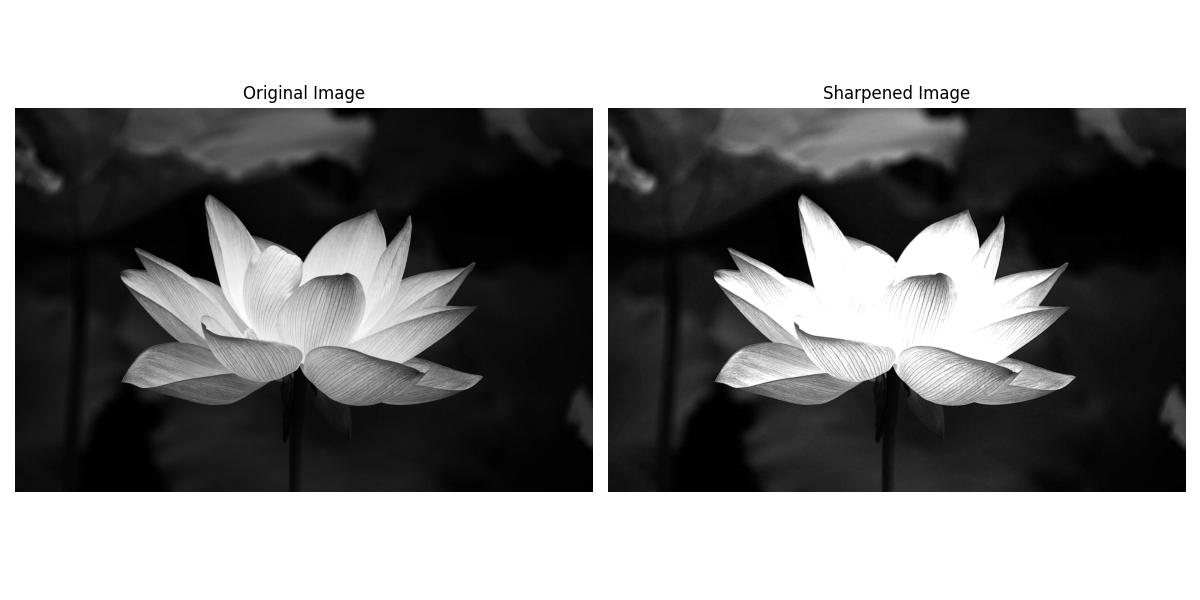


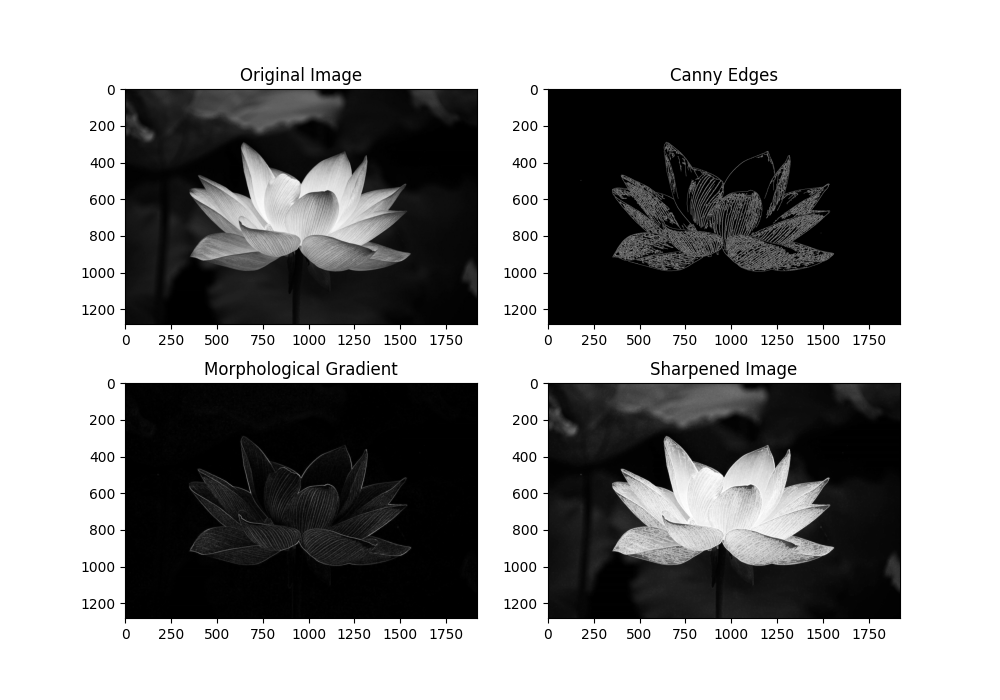


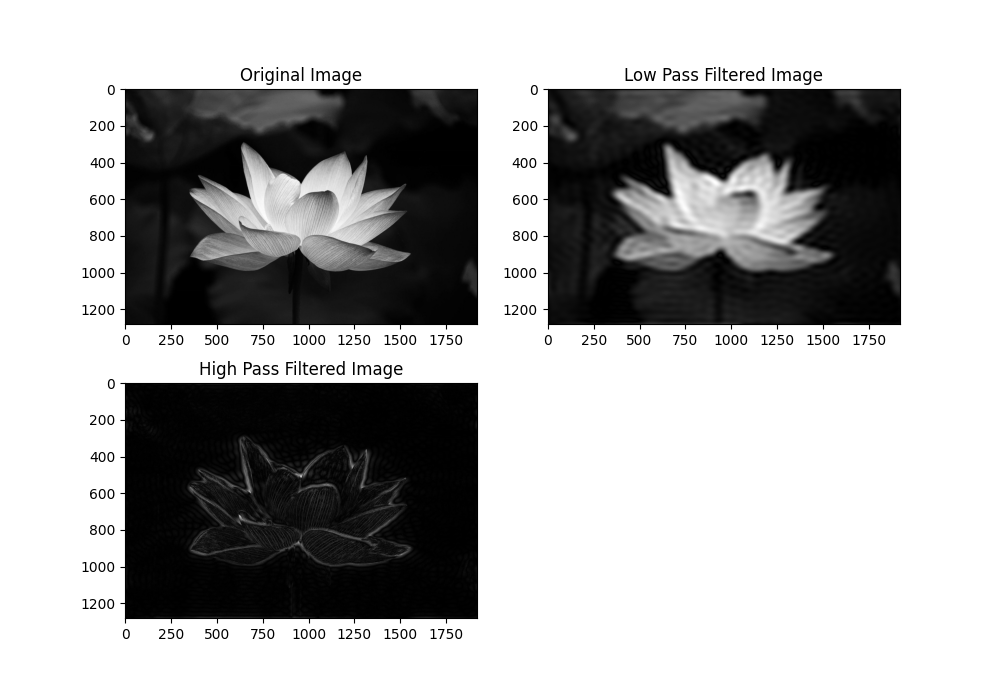












**六、实验小结**