**南 京 师 范 大 学**

**《数字图像处理》**

**期末作业报告**



|  |  |  |
| --- | --- | --- |
| **题目** | **：** | **Matlab图像处理** |
| **姓名** | **：** | **张烨妍** |
| **学号** | **：** | **19220711** |
| **任课教师** | **：** | **谢捷** |

[一：直方图处理 4](#_Toc185610615)

[1.1 gray\_histogram 方法：统计图像灰度直方图 4](#_Toc185610616)

[1.2 draw\_histogram 方法：绘制灰度直方图 5](#_Toc185610617)

[1.3直方图均衡化 (histogram\_equalization) 6](#_Toc185610618)

[1.4 BtnHistogramEqualizationPushed 7](#_Toc185610619)

[1.5 DropDownHistogramValueChanged 7](#_Toc185610620)

[1.6 KnobHistogramStretchingValueChanged 8](#_Toc185610621)

[1.7函数：compute\_histogram 9](#_Toc185610622)

[二：灰度化处理 10](#_Toc185610623)

[2.1线性灰度级增强 (linear\_enhance) 10](#_Toc185610624)

[2.2非线性对数变换 (log\_enhance) 11](#_Toc185610625)

[2．3非线性指数变换 (exp\_enhance)·· 11](#_Toc185610626)

[2.4 ImgEnhanceInputClicked 11](#_Toc185610627)

[2.5 SpinnerEnhanceTanValueChanged 12](#_Toc185610628)

[2.6 SpinnerEnhanceLogValueChanged 13](#_Toc185610629)

[2.7 BtnEnhanceExpButtonPushed 14](#_Toc185610630)

[三 缩放旋转处理 15](#_Toc185610631)

[3.1 图像缩放 (scale) 15](#_Toc185610632)

[3.2图像旋转 (rotate) 16](#_Toc185610633)

[3.3 ImgRSInputImageClicked 16](#_Toc185610634)

[3.4 SliderRSValueChanged 17](#_Toc185610635)

[四 噪声滤波处理 18](#_Toc185610636)

[4.1双边滤波 (filter\_double) 18](#_Toc185610637)

[4.2基于模糊技术的加权均值滤波 (filter\_mohu) 19](#_Toc185610638)

[4.3 DropdownFilterValueChanged 19](#_Toc185610639)

[4.4 NoiseFilter Tab的创建 20](#_Toc185610640)

[五 边缘检测与提取处理 21](#_Toc185610641)

[5.1 图像锐化函数：sharpen\_image 21](#_Toc185610642)

[5.2 Roberts 边缘检测：edge\_roberts 23](#_Toc185610643)

[5.3 edge\_prewitt 24](#_Toc185610644)

[5.4 edge\_sobel 25](#_Toc185610645)

[5.5 edge\_laplacian 26](#_Toc185610646)

[六 图像分割提取 27](#_Toc185610647)

[6.1 BtnFGOstuButtonPushed - Otsu 阈值法进行图像分割 28](#_Toc185610648)

[6.2 KnobFGtValueChanged - 使用指定阈值进行图像分割 29](#_Toc185610649)

[6.3 SpinnerFGkValueChanged - 使用 K-means 聚类进行图像分割 31](#_Toc185610650)

[6.4 CheckBoxFGreverseValueChanged - 控制蒙版的反转 31](#_Toc185610651)

[6.5 KnobFGhoughValueChanged - 霍夫变换进行图像分割 32](#_Toc185610652)

[6.6 BtnseedPushed - 区域生长法进行图像分割 33](#_Toc185610653)

[6.7 BtnactiveButtonPushed - 主动轮廓模型进行图像分割 33](#_Toc185610654)

[七 图像特征提取 35](#_Toc185610655)

[7.1 LBP特征提取按钮功能 (LBPButtonPushed) 35](#_Toc185610656)

[7.2 LBP 特征提取 36](#_Toc185610657)

[7.3 函数：getNeighborhood 37](#_Toc185610658)

[7.4 函数：calculateLBP 38](#_Toc185610659)

[7.5 HOG特征提取按钮功能 (HOGButtonPushed) 39](#_Toc185610660)

[7.6 HOG 特征提取 40](#_Toc185610661)

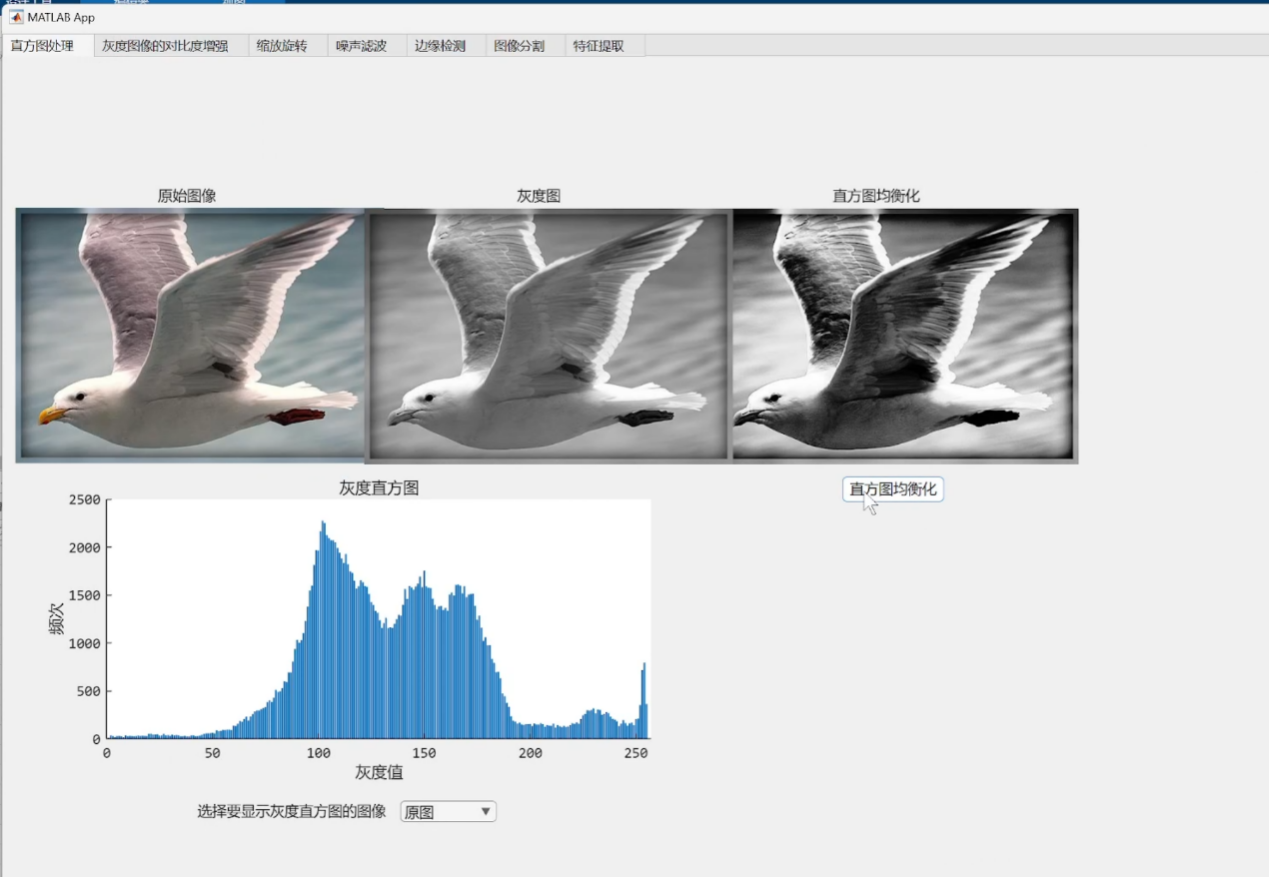
[7.7 函数：normalize\_hog 40](#_Toc185610662)

[八 图像处理大作业心得体会 41](#_Toc185610663)

[附： 42](#_Toc185610664)

[大作业位置链接： 42](#_Toc185610665)

# 一：直方图处理



## 1.1 gray\_histogram 方法：统计图像灰度直方图

function histogram = gray\_histogram(app, image)

histogram = zeros(256, 1); % 初始化灰度直方图数组，每个值对应0-255灰度级的像素计数

[height, width] = size(image); % 获取图像的高和宽

for x = 1:width

for y = 1:height

pixelValue = image(y, x) + 1; % 获取像素值并将其转换为索引 (1-256)

histogram(pixelValue) = histogram(pixelValue) + 1; % 对应灰度值计数加1

end

end

end

**代码详解**

1. **初始化灰度直方图数组**：
   1. histogram = zeros(256, 1); 创建一个长度为256的列向量，用于存储每个灰度级的像素数量。256个灰度级对应灰度值0到255。
2. **获取图像尺寸**：
   1. [height, width] = size(image); 通过 size 函数提取图像的高度和宽度，确保可以遍历每个像素。
3. **遍历每个像素**：
   1. 使用嵌套的 for 循环遍历图像的每一列（x）和每一行（y）。
4. **计算像素值对应的直方图索引**：
   1. pixelValue = image(y, x) + 1; MATLAB 的数组索引从1开始，而灰度值从0开始。因此，将像素值加1后作为数组索引。
5. **更新直方图**：
   1. histogram(pixelValue) = histogram(pixelValue) + 1; 增加对应灰度值的计数。

**算法原理**

这是计算灰度直方图的经典方法：

1. 图像是由像素组成的矩阵，每个像素的灰度值介于0到255之间。
2. 直方图统计的是每种灰度值的频次。

**设计思路**

1. 为确保直方图统计结果准确，需要遍历图像中的每一个像素，并根据其灰度值更新直方图。
2. 使用二维数组访问像素位置，符合直观的图像处理逻辑。

**正确性分析**

1. 对每个像素访问和计数操作的正确性依赖于以下几点：
   1. 索引转换正确：像素值加1后符合MATLAB索引规则。
   2. 初始化数组为零：避免累加之前的残留值。
2. 时间复杂度：O(width × height)，因为每个像素仅访问一次。

## 1.2 draw\_histogram 方法：绘制灰度直方图

function draw\_histogram(app, histogram)

% 获取灰度直方图坐标轴组件

axesHandle = app.UIAxesGrayHistogramHistgram;

% 绘制直方图

bar(axesHandle, histogram);

end

**功能**

此方法接受一个灰度直方图数组，并在指定的 UIAxes 对象中绘制直方图。

**代码详解**

1. **获取目标坐标轴组件**：
   1. axesHandle = app.UIAxesGrayHistogramHistgram; 使用应用程序的 UIAxes 组件来作为绘图目标。
2. **绘制直方图**：
   1. bar(axesHandle, histogram); 使用 MATLAB 的 bar 函数绘制柱状图，将直方图数据以柱状的形式展示。

**算法原理**

1. 柱状图是一种直观的方式展示每个灰度级的像素频次。
2. MATLAB 的 bar 函数会根据输入数组自动生成柱状图。

**设计思路**

1. UI组件中的 UIAxes 是绘图的目标对象，分离数据处理和展示逻辑，便于代码维护和扩展。
2. 直方图作为柱状图展示，可以清晰地反映图像的灰度分布。

**正确性分析**

1. 确保 histogram 数据的格式为长度256的列向量，否则 bar 函数可能报错或输出异常。
2. UIAxes 必须在应用中已正确初始化，否则绘图会失败。

## 1.3直方图均衡化 (histogram\_equalization)

**功能：**

实现图像的直方图均衡化，用于增强图像的对比度，使图像灰度分布更加均匀。

**思路：**

1. **计算灰度直方图**：通过 imhist 获得图像各灰度值的像素数量。
2. **累积分布函数 (CDF)**：通过累加灰度值的频率，计算每个灰度级对应的累计概率。
3. **像素映射**：将每个像素重新映射到均衡化后的灰度值。
4. **归一化**：将灰度值缩放到 [0, 255]。

histgram = imhist(image); % 计算灰度直方图

[h, w] = size(image); % 图像的高和宽

equalized\_image = zeros(h, w, 'uint8'); % 初始化结果图像

% 累计概率分布函数（累加直方图值）

s = zeros(256);

s(1) = histgram(1);

for t = 2 : 256

s(t) = s(t - 1) + histgram(t);

end

% 像素映射，根据累计概率调整像素值

for x = 1: w

for y = 1: h

equalized\_image(y, x) = uint8(255 \* s(image(y, x) + 1) / (w \* h));

end

end

**算法原理：**

直方图均衡化的本质是对像素的灰度级重新分配，使图像整体对比度增强，尤其是在原始图像对比度较低时。

## 1.4 BtnHistogramEqualizationPushed

if ~isempty(app.IImgHistogramInput.ImageSource)

equalized\_image = histogram\_equalization(app, rgb2gray(app.IImgHistogramInput.ImageSource));

app.ImgHistogramEqualization.ImageSource = cat(3, equalized\_image, equalized\_image, equalized\_image);

else

disp('输入图像为空');

end

**功能**:

1. 对加载的图像进行直方图均衡化，并显示结果。

**思路**:

1. 检查图像输入源是否为空，确保操作的有效性。
2. 调用histogram\_equalization函数完成均衡化操作。
3. 将结果存储到界面组件的ImageSource，实现动态显示。

## 1.5 DropDownHistogramValueChanged

function DropDownHistogramValueChanged(app, event)

value = app.DropDownHistogram.Value;

if strcmp(value, '原图')

histogram = gray\_histogram(app, rgb2gray(app.IImgHistogramInput.ImageSource));

draw\_histogram(app, histogram);

elseif strcmp(value, '均衡化')

histogram = gray\_histogram(app, app.ImgHistogramEqualization.ImageSource);

draw\_histogram(app, histogram);

end

end

**解释：**

1. **功能描述：** 用户切换下拉菜单时，根据选择绘制原始图像或均衡化图像的直方图。
2. **实现逻辑：**
   1. 读取下拉菜单当前值 value。
   2. 根据值判断并调用 gray\_histogram 计算对应直方图，调用 draw\_histogram 绘制。

## 1.6 KnobHistogramStretchingValueChanged

function KnobHistogramStretchingValueChanged(app, event)

stretch\_intensity = app.KnobHistogramStretching.Value;

if ~isempty(app.ImgHistInput.ImageSource)

gray\_image = rgb2gray(app.ImgHistInput.ImageSource);

min\_val = double(min(gray\_image(:)));

max\_val = double(max(gray\_image(:)));

stretched\_image = imadjust(gray\_image, [min\_val max\_val] / 255, [0 stretch\_intensity / 100]);

app.ImgHistOutput.ImageSource = cat(3, stretched\_image, stretched\_image, stretched\_image);

else

disp('未加载图像');

end

end

**解释：**

* **功能描述：** 根据用户调整的 **直方图拉伸旋钮 (Histogram Stretching)** 值，对图像进行直方图拉伸处理。
* **实现逻辑：**
  + 将灰度图像的像素值范围线性拉伸到新的范围。
  + 使用 imadjust 执行拉伸操作，输入范围为 [min\_val, max\_val]。
  + 根据旋钮值调整输出范围的最大值。



## 1.7函数：compute\_histogram

**功能：** 计算一个 cell 内的方向梯度直方图。

function hist = compute\_histogram(~,mag, ang, bins)

% 将角度转换到 0-180 度区间

ang = mod(ang, 180);

% 创建方向梯度直方图

hist = zeros(1, bins);

% 计算每个像素的梯度方向对应的 bin

for i = 1:numel(ang)

bin = floor(ang(i) / (180 / bins)) + 1;

hist(bin) = hist(bin) + mag(i);

end

end

**详细解释：**

1. **算法原理：**
   1. 将每个像素的方向分配到对应的直方图 bin 中。
   2. 使用幅度值（mag）作为权重累加。
2. **关键点：**
   1. bins 决定直方图分辨率，通常为 9（每 bin 对应 20 度）。
   2. 归一化角度值确保方向范围正确。

# 二：灰度化处理



## 2.1线性灰度级增强 (linear\_enhance)

**功能：**

通过线性变换增强图像对比度，例如通过倍增灰度值来拉伸亮度范围。

**思路：**

1. 对每个像素灰度值乘以 tan\_alpha。
2. 将增强后的像素值限制在 [0, 255] 范围内。

new\_image = tan\_alpha \* double(image); % 应用线性变换

new\_image = uint8(max(0, min(255, new\_image))); % 限制灰度值在合法范围

**算法原理：**

线性变换简单有效，通过调整系数 tan\_alpha 来实现亮度的增强或减弱。

## 2.2非线性对数变换 (log\_enhance)

**功能：**

通过对数变换压缩灰度值范围，增强暗部细节。

**思路：**

1. 应用公式 new\_pixel = c \* log(1 + pixel)，其中 c 是缩放因子。
2. 对结果进行灰度值限制。

new\_image = c \* log(1 + double(image)); % 对数变换

new\_image = uint8(max(0, min(255, new\_image))); % 限制灰度值范围

**算法原理：**

对数变换是非线性的，对暗部灰度值变化更敏感，可以有效突出图像的暗部细节。

## 2．3非线性指数变换 (exp\_enhance)··

**功能：**

利用指数函数调整图像亮度或对比度。

**思路：**

1. 应用公式 new\_pixel = b^(c \* (pixel - a)) - 1，其中 a 为基准灰度，b 和 c 控制变换强度。
2. 结果灰度值限定在 [0, 255]。

new\_image = power(b, c \* (image - a)) - 1; % 指数变换

new\_image = uint8(max(0, min(255, new\_image))); % 限制灰度范围

**算法原理：**

指数变换可以压缩亮部或增强暗部，适合非均匀亮度分布的图像增强。

## 2.4 ImgEnhanceInputClicked

function ImgEnhanceInputClicked(app, event)

[filename, pathname] = uigetfile({'\*.jpg;\*.png;\*.bmp', 'Image Files (\*.jpg, \*.png, \*.bmp)'}, '选择图像文件');

if isequal(filename, 0) || isequal(pathname, 0)

return;

end

fullFilePath = fullfile(pathname, filename);

imgData = imread(fullFilePath);

app.ImgEnhanceInput.ImageSource = imgData;

gray\_image = rgb2gray(imgData);

app.ImgEnhanceGray.ImageSource = cat(3, gray\_image, gray\_image, gray\_image);

new\_image\_linear = linear\_enhance(app, rgb2gray(app.ImgEnhanceGray.ImageSource), app.SpinnerEnhanceTan.Value);

new\_image\_log = log\_enhance(app, rgb2gray(app.ImgEnhanceGray.ImageSource), app.SpinnerEnhanceLog.Value);

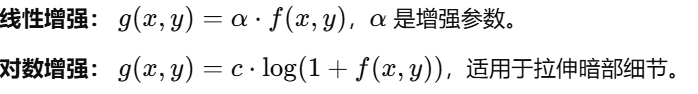
app.ImgEnhanceLinear.ImageSource = cat(3, new\_image\_linear, new\_image\_linear, new\_image\_linear);

app.ImgEnhanceLog.ImageSource = cat(3, new\_image\_log, new\_image\_log, new\_image\_log);

end

**解释：**

1. **功能描述：** 用户点击图像增强输入区域时，加载图片，生成灰度图，并执行线性增强和对数增强处理，分别显示结果。
2. **实现逻辑：**
   1. 检查是否选择了文件，避免无效路径。
   2. 将图片转换为灰度后调用 linear\_enhance 和 log\_enhance，分别处理。
   3. 拼接为三通道显示。
3. **核心算法：**



## 2.5 SpinnerEnhanceTanValueChanged

function SpinnerEnhanceTanValueChanged(app, event)

value = app.SpinnerEnhanceTan.Value;

if ~isempty(app.ImgEnhanceGray.ImageSource)

enhanced\_image = linear\_enhance(app, rgb2gray(app.ImgEnhanceGray.ImageSource), value);

app.ImgEnhanceLinear.ImageSource = cat(3, enhanced\_image, enhanced\_image, enhanced\_image);

else

disp('未加载图像');

end

end

**解释：**

1. **功能描述：** 当用户调整 **线性增强系数 (Tan)** 的微调框值时，根据新的系数实时更新增强后的图像。
2. **实现逻辑：**
   1. 读取微调框的当前值 value，作为线性增强的系数。
   2. 检查输入图像是否存在，避免空引用错误。
   3. 调用 linear\_enhance 对灰度图像执行线性增强，将结果加载到对应组件。

## 2.6 SpinnerEnhanceLogValueChanged

function SpinnerEnhanceLogValueChanged(app, event)

value = app.SpinnerEnhanceLog.Value;

if ~isempty(app.ImgEnhanceGray.ImageSource)

enhanced\_image = log\_enhance(app, rgb2gray(app.ImgEnhanceGray.ImageSource), value);

app.ImgEnhanceLog.ImageSource = cat(3, enhanced\_image, enhanced\_image, enhanced\_image);

else

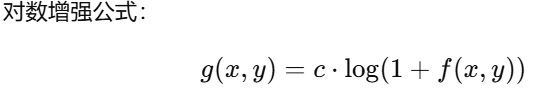
disp('未加载图像');

end

end

**解释：**

1. **功能描述：** 用户调整 **对数增强系数 (Log)** 的微调框值时，实时更新增强后的图像。
2. **实现逻辑：**
   1. 获取微调框当前值 value 作为对数增强参数。
   2. 确保图像已加载，调用 log\_enhance 对灰度图像执行增强操作。
   3. 将结果拼接为三通道显示。



## 2.7 BtnEnhanceExpButtonPushed

function BtnEnhanceExpButtonPushed(app, event)

if ~isempty(app.ImgEnhanceGray.ImageSource)

enhanced\_image = exp\_enhance(app, rgb2gray(app.ImgEnhanceGray.ImageSource));

app.ImgEnhanceExp.ImageSource = cat(3, enhanced\_image, enhanced\_image, enhanced\_image);

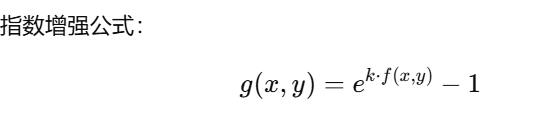
else

disp('未加载图像');

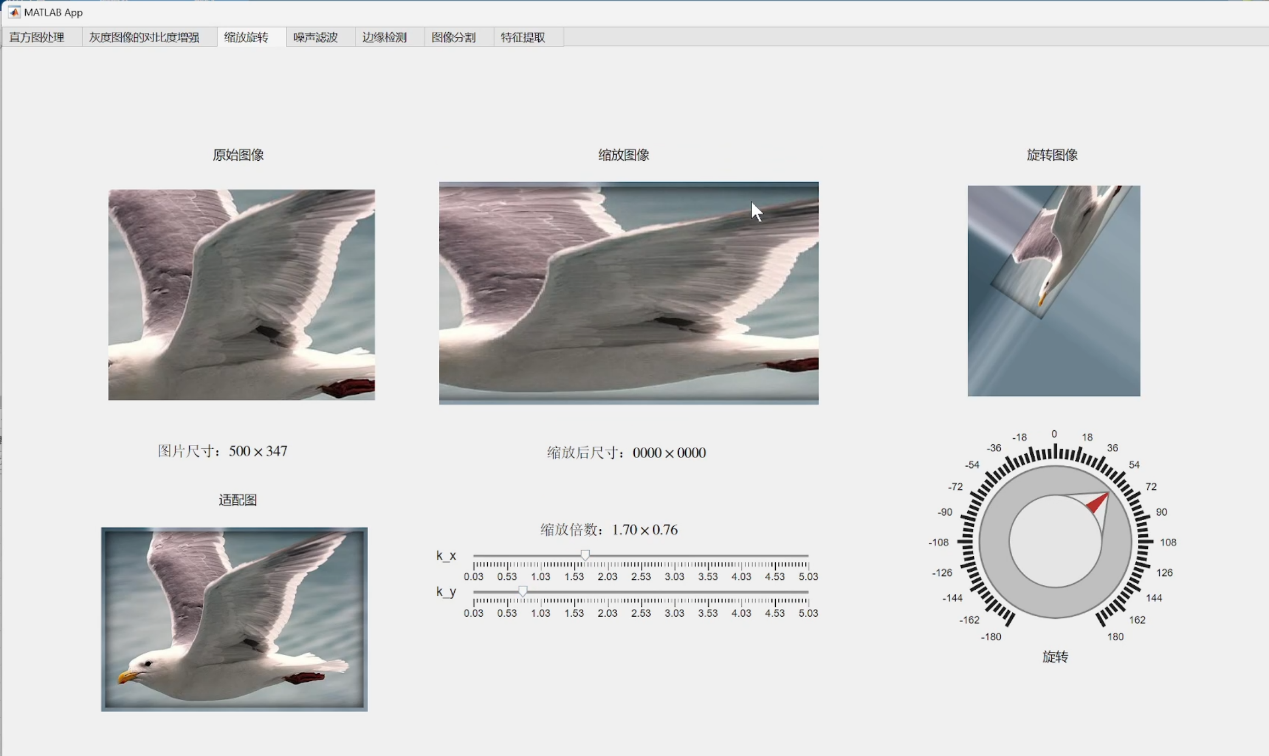
end

end

**解释：**

1. **功能描述：** 用户点击 **指数增强按钮 (Exp)** 时，执行增强操作并显示结果。
2. **实现逻辑：**
   1. 检查是否加载了灰度图像。
   2. 调用 exp\_enhance 实现指数增强。
   3. 显示处理后的三通道图像。
   4. 

# 三 缩放旋转处理



## 3.1 图像缩放 (scale)

**功能：**

对图像进行缩放，支持不同的纵横比例。

**思路：**

1. **初始化新图像大小**：根据比例 k\_x 和 k\_y 计算新尺寸。
2. **双线性插值**：
   1. 通过像素位置的分数部分计算权重。
   2. 使用四个邻近像素进行插值计算新像素值。

**代码解释：**

for x\_ = 1:new\_w

for y\_ = 1:new\_h

x = x\_ / k\_x;

y = y\_ / k\_y;

% 插值计算

f\_x\_y\_b = image(y, x, :) + b \* (image(y+1, x, :) - image(y, x, :));

f\_x\_a\_y\_b = f\_x\_y\_b + a \* (image(y, x+1, :) - f\_x\_y\_b);

new\_image(y\_, x\_, :) = f\_x\_a\_y\_b;

end

end

**算法原理：**

双线性插值通过线性加权周围像素值计算目标像素值，能有效平滑缩放图像。

## 3.2图像旋转 (rotate)

**功能：**

将图像绕中心点旋转指定角度。

**思路：**

1. **计算新图像尺寸**：根据旋转角度确定包围盒大小。
2. **逆向映射**：
   1. 对目标图像的每个像素，找到其在原图中的对应位置。
   2. 应用双线性插值计算像素值。

new\_x = cosd(theta) \* x - sind(theta) \* y + center\_x; % 坐标变换

new\_y = sind(theta) \* x + cosd(theta) \* y + center\_y;

算法原理：

通过坐标变换实现旋转，双线性插值保证旋转后的图像平滑过渡。

## 3.3 ImgRSInputImageClicked

function ImgRSInputImageClicked(app, event)

[filename, pathname] = uigetfile({'\*.jpg;\*.png;\*.bmp', 'Image Files (\*.jpg, \*.png, \*.bmp)'}, '选择图像文件');

if isequal(filename, 0) || isequal(pathname, 0)

return;

end

fullFilePath = fullfile(pathname, filename);

imgData = imread(fullFilePath);

app.ImgRSInputImage.ImageSource = imgData;

app.SliderRS.Value = 100

app.KnobRSrotate.Value = 0;

resized\_image = imresize(imgData, app.SliderRS.Value / 100);

app.ImgRSResize.ImageSource = resized\_image;

rotated\_image = imrotate(resized\_image, app.KnobRSrotate.Value);

app.ImgRSRotate.ImageSource = rotated\_image;

end

**解释：**

1. **功能描述：** 用户点击 **缩放与旋转输入图像区域 (RS)** 时，加载图片，并初始化滑块和旋钮的值，显示初始缩放与旋转结果。
2. **实现逻辑：**
   1. 使用文件选择器加载图片并显示在输入区域。
   2. 初始化滑块和旋钮值（缩放为 100%，旋转角度为 0°）。
   3. 调用 imresize 按滑块值进行缩放，调用 imrotate 按旋钮值旋转，显示处理结果。

## 3.4 SliderRSValueChanged

function SliderRSValueChanged(app, event)

scale = app.SliderRS.Value / 100;

resized\_image = imresize(app.ImgRSInputImage.ImageSource, scale);

app.ImgRSResize.ImageSource = resized\_image;

rotated\_image = imrotate(resized\_image, app.KnobRSrotate.Value);

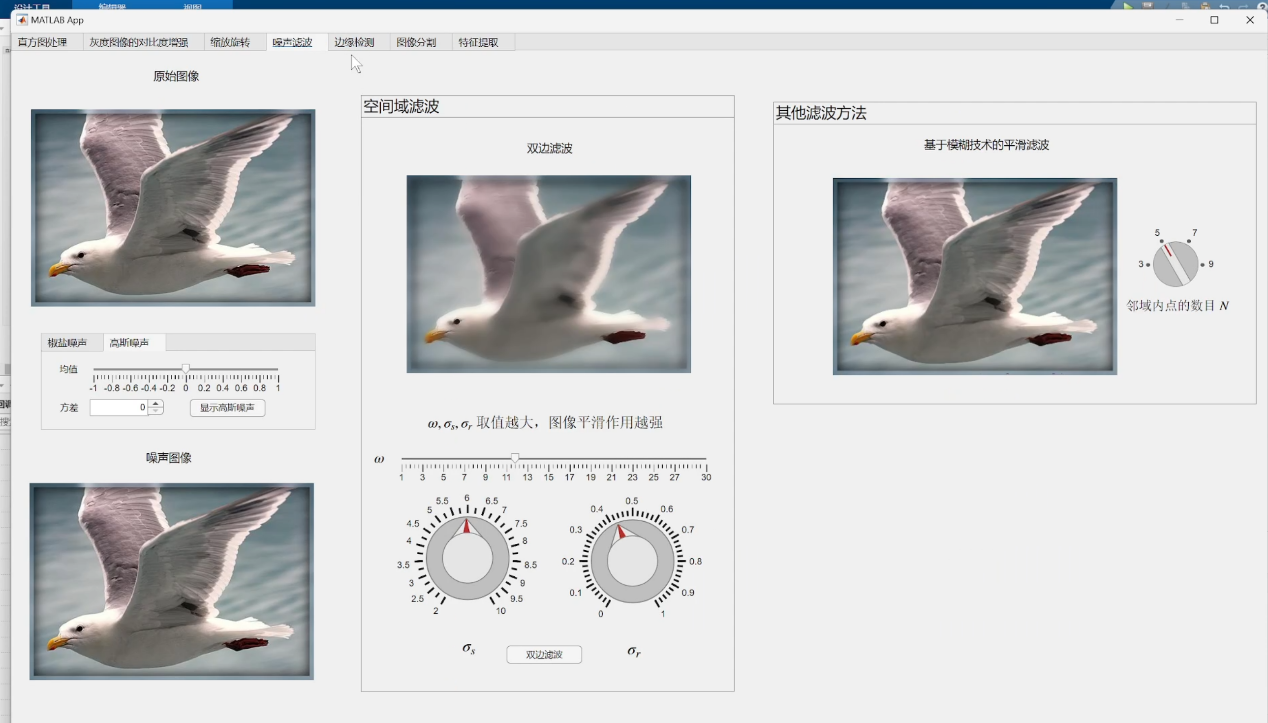
app.ImgRSRotate.ImageSource = rotated\_image;

end

**解释：**

1. **功能描述：** 用户调整 **缩放滑块 (RS)** 的值时，根据缩放比例实时更新缩放和旋转后的图像。
2. **实现逻辑：**
   1. 读取滑块的值 scale，按比例缩放图片。
   2. 根据旋钮当前值，结合缩放后的图像，计算旋转后的结果。
3. **核心算法：**
   1. **缩放：** 使用 MATLAB 的 imresize 函数，基于双线性插值进行大小调整。
   2. **旋转：** 使用 imrotate 函数，根据中心点旋转图像。

# 四 噪声滤波处理



## 4.1双边滤波 (filter\_double)

**功能：**

保留图像边缘细节的同时去除噪声。

**思路：**

1. 计算空间权重和灰度权重。
2. 对像素邻域加权求和。
3. 使用双边权值调整像素值。

Gr = exp(-(sum((I - image(i, j, :)).^2, 3)) / (2 \* sigma\_r^2)); % 灰度权重

W = Gr .\* Gs; % 总权重

算法原理：

结合空间距离和灰度差的权重，平滑噪声同时保留边缘。

## 4.2基于模糊技术的加权均值滤波 (filter\_mohu)

**功能：**

基于像素间灰度差的模糊权重进行滤波。

**思路：**

1. 计算像素差的平方和，确定模糊参数 beta。
2. 根据模糊权重计算加权均值更新像素值。

mu = exp(-d / beta); % 模糊权重

sum2 = sum2 + mu / beta \* image(m, n); % 加权均值分子

算法原理：

模糊滤波动态调整权重，适合复杂纹理场景的去噪。

## 4.3 DropdownFilterValueChanged

function DropdownFilterValueChanged(app, event)

selected\_filter = app.DropdownFilter.Value;

if ~isempty(app.ImgFilterInput.ImageSource)

switch selected\_filter

case '均值滤波'

filtered\_image = imfilter(rgb2gray(app.ImgFilterInput.ImageSource), fspecial('average', [3 3]));

case '中值滤波'

filtered\_image = medfilt2(rgb2gray(app.ImgFilterInput.ImageSource), [3 3]);

case '高斯滤波'

filtered\_image = imgaussfilt(rgb2gray(app.ImgFilterInput.ImageSource), 0.5);

otherwise

disp('未知滤波器类型');

return;

end

app.ImgFilterOutput.ImageSource = cat(3, filtered\_image, filtered\_image, filtered\_image);

else

disp('未加载图像');

end

end

**解释：**

1. **功能描述：** 用户选择不同滤波器时，对图像进行相应滤波操作并显示结果。
2. **实现逻辑：**
   1. 获取下拉菜单选择的滤波器类型 selected\_filter。
   2. 检查输入图像是否已加载。
   3. 根据选择调用对应的滤波函数：
      1. **均值滤波：** 使用 imfilter 和 fspecial 函数生成滤波器核。
      2. **中值滤波：** 使用 medfilt2 函数对噪声抑制效果较好的滤波。
      3. **高斯滤波：** 使用 imgaussfilt 对图像进行平滑处理。

## 4.4 NoiseFilter Tab的创建

这一部分主要实现噪声滤波，包括椒盐噪声与高斯噪声滤波。

**创建椒盐噪声控制滑块**

app.SliderNFsaltD = uislider(app.Tab);

app.SliderNFsaltD.Limits = [0 1];

app.SliderNFsaltD.ValueChangedFcn = createCallbackFcn(app, @SliderNFsaltDValueChanged, true);

app.SliderNFsaltD.Position = [83 57 257 3];

uislider：用于控制椒盐噪声的密度，值范围为0到1。通过ValueChangedFcn触发回调函数SliderNFsaltDValueChanged来更新噪声图像。

**高斯噪声控制**

app.SliderNFgm = uislider(app.Tab\_2);

app.SliderNFgm.Limits = [-1 1];

app.SliderNFgm.Position = [70 78 244 3];

uislider：控制高斯噪声的均值，范围为-1到1。

**高斯噪声方差输入框**

app.SpinnerNFgVar = uispinner(app.Tab\_2);

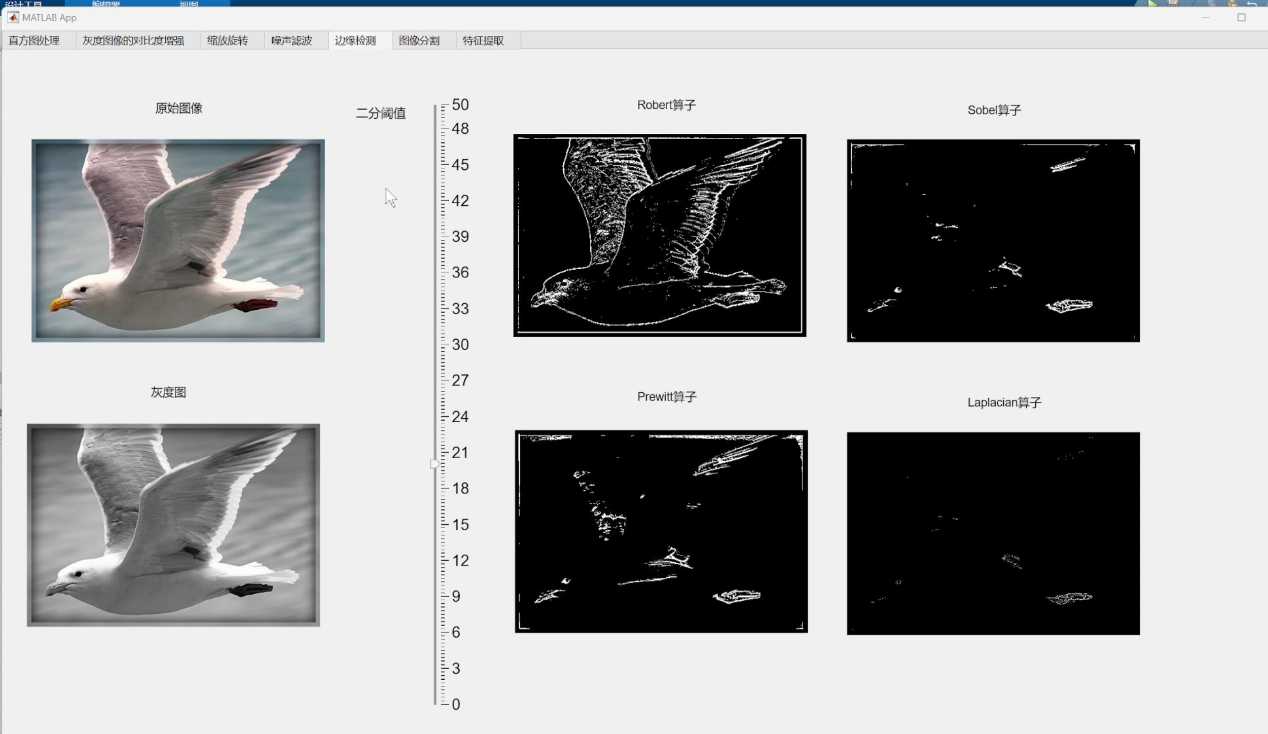
app.SpinnerNFgVar.Step = 0.01;

app.SpinnerNFgVar.Limits = [0 2];

app.SpinnerNFgVar.Position = [65 18 100 22];

uispinner：用于输入方差值，控制高斯噪声的方差。

# 五 边缘检测与提取处理



## 5.1 图像锐化函数：sharpen\_image

**功能**

对输入图像进行锐化处理，支持多种算子（roberts, sobel, prewitt, laplacian）的应用。

function sharpImage = sharpen\_image(~, image, op)

image = im2double(image); % 将图像转换为双精度，以支持小数运算

if strcmp(op, 'roberts')

H1 = [1 0; 0 -1];

H2 = [0 1; -1 0]; % Roberts算子模板

elseif strcmp(op, 'sobel')

H1 = [-1 -2 -1; 0 0 0; 1 2 1];

H2 = [-1 0 1; -2 0 2; -1 0 1]; % Sobel算子模板

elseif strcmp(op, 'prewitt')

H1 = [-1 -1 -1; 0 0 0; 1 1 1];

H2 = [-1 0 1; -1 0 1; -1 0 1]; % Prewitt算子模板

elseif strcmp(op, 'laplacian')

H = [0 -1 0; -1 5 -1; 0 -1 0]; % Laplacian锐化模板

sharpImage = imfilter(image, H); % 直接进行卷积

return

end

R1 = imfilter(image, H1); % 应用第一个模板

R2 = imfilter(image, H2); % 应用第二个模板

edgeImage = abs(R1) + abs(R2); % 合成梯度图像

sharpImage = image + edgeImage; % 原图加上边缘增强后的图像

end

**实现思路**

1. **输入图像归一化**： 使用 im2double 将图像转换为 [0,1] 的范围，便于后续的算子操作和梯度叠加。
2. **选择算子模板**： 根据用户指定的操作（op），选择合适的滤波器模板（Roberts、Sobel、Prewitt、Laplacian）。
3. **滤波操作**：
   1. 对于 Roberts、Sobel 和 Prewitt 算子：
      * 使用两个滤波器模板 H1 和 H2 分别计算图像的梯度分量。
      * 通过绝对值和相加得到梯度幅值图像（edgeImage）。
   2. 对于 Laplacian 算子：
      * 直接通过模板 H 和 imfilter 进行卷积操作得到锐化图像。
4. **叠加梯度**： 将梯度图像与原始图像叠加，提升边缘对比度，从而完成锐化。

**实现原理**

1. **算子选择**：通过参数op指定锐化方法，支持四种常见的锐化算子。每种算子的模板定义了其特定的滤波规则。
   * **Roberts**：通过对像素梯度的计算突出边缘。
   * **Sobel 和 Prewitt**：更复杂的梯度算子，能提取垂直和水平边缘信息。
   * **Laplacian**：二阶导数算子，强化图像的局部对比度。
2. **锐化操作**：
   * 对于Roberts、Sobel、Prewitt：通过两个模板（H1 和 H2）分别计算不同方向的梯度，将梯度绝对值相加后增强原图。
   * 对于Laplacian：使用固定模板进行卷积，直接得到锐化后的图像。
3. **正确性**：利用imfilter函数进行卷积运算，确保滤波过程高效且准确。边缘增强通过梯度累加实现。

**总结**

该方法灵活性高，适合多种应用场景，但不同算子在噪声敏感性和边缘强化程度上表现不同。

## 5.2 Roberts 边缘检测：edge\_roberts

**功能**

通过Roberts算子进行边缘检测，并基于阈值对检测结果进行二值化。

function new\_image = edge\_roberts(~, f, T)

[m, n] = size(f); % 获取图像尺寸

new\_image = zeros(m, n); % 初始化新图像

for i = 2:m-1

for j = 2:n-1

% 计算Roberts算子对应的梯度

new\_image(i, j) = abs(f(i+1, j+1) - f(i, j)) + abs(f(i, j+1) - f(i+1, j));

% 应用阈值处理

if new\_image(i, j) < T

new\_image(i, j) = 0;

else

new\_image(i, j) = 255;

end

end

end

new\_image = uint8(new\_image); % 转换为uint8格式

end

**实现思路**

1. **初始化**： 创建一个与输入图像尺寸相同的空矩阵 new\_image，用于存储边缘检测结果。
2. **梯度计算**： 对每个像素（非边界区域），计算其相邻像素的 Roberts 梯度分量，分别在对角方向求差。
3. **阈值分割**：
   * 如果梯度值小于阈值 T，认为该像素为非边缘，设置为 0。
   * 否则，将该像素设置为 255，标记为边缘。
4. **类型转换**： 将结果图像转换为 uint8 格式，便于显示。

**实现原理**

1. **梯度计算**：
   * Roberts算子通过对对角线方向的像素梯度进行计算，突出图像中显著的变化区域。
2. **二值化处理**：
   * 通过阈值T将检测结果二值化，进一步提取显著边缘。
3. **正确性**：通过循环逐像素计算，适用于小型图像处理，但效率较低。

## 5.3 edge\_prewitt

**功能说明**

实现 Prewitt 算子图像边缘检测，通过水平方向和垂直方向的梯度计算图像的边缘强度，并通过阈值分割得到二值化的边缘图像。

function new\_image = edge\_prewitt(~, f, T)

% Prewitt 算子模板

Px = [-1 -1 -1; 0 0 0; 1 1 1]; % 垂直方向模板

Py = [-1 0 1; -1 0 1; -1 0 1]; % 水平方向模板

% 水平与垂直方向梯度计算

gx = imfilter(double(f), Px);

gy = imfilter(double(f), Py);

% 梯度幅值计算

G = abs(gx) + abs(gy);

% 阈值分割

new\_image = uint8((G >= T) \* 255);

end

**实现思路**

1. **Prewitt算子模板**： 使用两个固定模板 Px 和 Py 分别计算垂直和水平方向的梯度。
2. **梯度计算**：
   1. 使用 imfilter 函数对输入图像进行卷积操作，计算水平和垂直方向的梯度图像。
   2. 求取梯度幅值图像 G，即两个方向梯度绝对值的相加。
3. **阈值分割**：
   1. 设置一个阈值 T，将梯度值小于 T 的像素置为 0，大于等于 T 的像素置为 255，生成二值化图像。
4. **类型转换**： 将结果图像转换为 uint8 格式，用于显示。

**算法原理**

1. **Prewitt算子**：
   1. 基于图像的一阶导数，适合计算图像边缘的方向梯度。
   2. 水平方向模板检测水平边缘，垂直方向模板检测垂直边缘。

**正确性分析**

1. 模板定义正确，符合 Prewitt 算子的标准。
2. 使用 imfilter 函数实现梯度计算，效率较高。
3. 阈值分割简洁明了，二值化结果符合要求。

## 5.4 edge\_sobel

**功能说明**

实现 Sobel 算子图像边缘检测，通过增强的水平方向和垂直方向的梯度计算图像的边缘强度，并通过阈值分割得到二值化的边缘图像。

function new\_image = edge\_sobel(~, f, T)

% Sobel 算子模板

Sx = [-1 -2 -1; 0 0 0; 1 2 1]; % 垂直方向模板

Sy = [-1 0 1; -2 0 2; -1 0 1]; % 水平方向模板

% 水平与垂直方向梯度计算

gx = imfilter(double(f), Sx);

gy = imfilter(double(f), Sy);

% 梯度幅值计算

G = abs(gx) + abs(gy);

% 阈值分割

new\_image = uint8((G >= T) \* 255);

end

**实现思路**

1. **Sobel算子模板**： 使用两个增强模板 Sx 和 Sy 分别计算垂直和水平方向的梯度。
2. **梯度计算**：
   1. 使用 imfilter 函数对输入图像进行卷积操作，计算水平和垂直方向的梯度图像。
   2. 求取梯度幅值图像 G，即两个方向梯度绝对值的相加。
3. **阈值分割**：
   1. 设置一个阈值 T，将梯度值小于 T 的像素置为 0，大于等于 T 的像素置为 255，生成二值化图像。
4. **类型转换**： 将结果图像转换为 uint8 格式，用于显示。

**算法原理**

1. **Sobel算子**：
   1. 是 Prewitt 算子的改进版本，使用中心加权的梯度计算方式，能更好地平滑噪声。
   2. 适合检测图像中的显著边缘，同时抑制细微噪声。

**正确性分析**

1. 模板定义正确，符合 Sobel 算子的标准。
2. 使用 imfilter 函数实现梯度计算，逻辑正确。
3. 阈值分割简洁明了，二值化结果符合预期。

## 5.5 edge\_laplacian

**功能说明**

实现 Laplacian 算子图像边缘检测，通过二阶导数的计算得到图像的边缘强度，并通过阈值分割生成二值化边缘图像。

function new\_image = edge\_laplacian(~, f, T)

% Laplacian 算子模板

H = [0 -1 0; -1 4 -1; 0 -1 0];

% 使用 Laplacian 模板进行卷积

G = abs(imfilter(double(f), H));

% 阈值分割

new\_image = uint8((G >= T) \* 255);

end

**实现思路**

1. **Laplacian算子模板**： 使用一个固定模板 H 计算图像的二阶梯度。
2. **卷积计算**：
   1. 使用 imfilter 函数对输入图像与模板 H 进行卷积操作。
   2. 计算绝对值梯度幅值图像 G。
3. **阈值分割**：
   1. 设置一个阈值 T，将梯度值小于 T 的像素置为 0，大于等于 T 的像素置为 255，生成二值化图像。
4. **类型转换**： 将结果图像转换为 uint8 格式，用于显示。

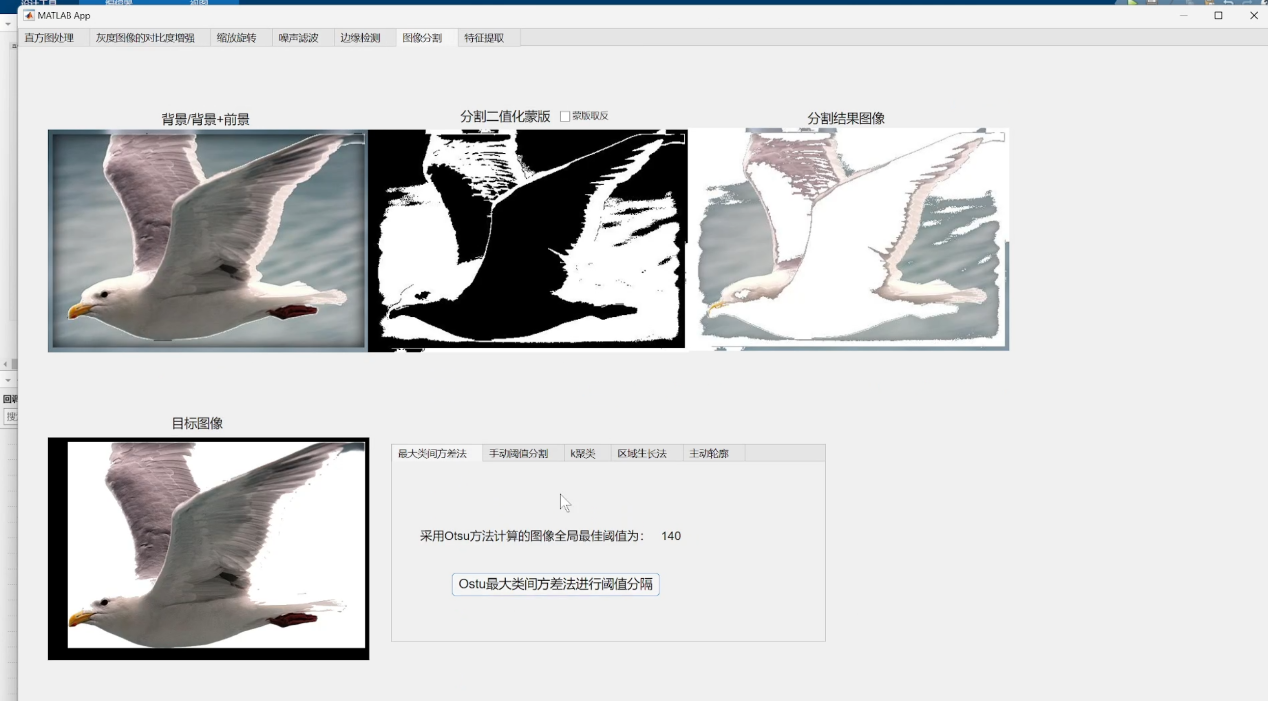
**算法原理**

1. **Laplacian算子**：
   1. 基于二阶导数，用于检测图像中的边缘和细节。
   2. 对噪声敏感，适合用在经过平滑处理的图像上。

**正确性分析**

1. 模板定义正确，符合 Laplacian 算子的标准。
2. 使用 imfilter 函数实现卷积计算，逻辑清晰。
3. 阈值分割直观，二值化结果能突出边缘信息。

# 六 图像分割提取



## 6.1 BtnFGOstuButtonPushed - Otsu 阈值法进行图像分割

function BtnFGOstuButtonPushed(app, event)

% 转换为灰度图像

imgData = app.ImgFGinput.ImageSource;

grayImage = rgb2gray(imgData);

% 使用Ostu最大类间方差法，进行阈值分割

threshold = graythresh(grayImage); % 计算 Otsu 阈值

app.LabelFGOstu.Text = threshold \* 255 + ""; % 显示阈值

% 将图像二值化

binaryImage = imbinarize(grayImage, threshold);

% 对二值图像进行后处理，去除小区域

binaryImage = bwareaopen(binaryImage, 1000);

% 创建彩色蒙版图像

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage); % 黑色背景

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage); % 白色背景

end

% 将彩色蒙版叠加到原始图片上

segmentedResult = imgData;

indices = colorMask; % 获取满足条件的像素索引

segmentedResult(indices) = 255; % 将这些像素设置为白色

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage); % 扩展为 RGB 图像

binaryImage = uint8(binaryImage) \* 255;

app.ImgFGbin.ImageSource = binaryImage;

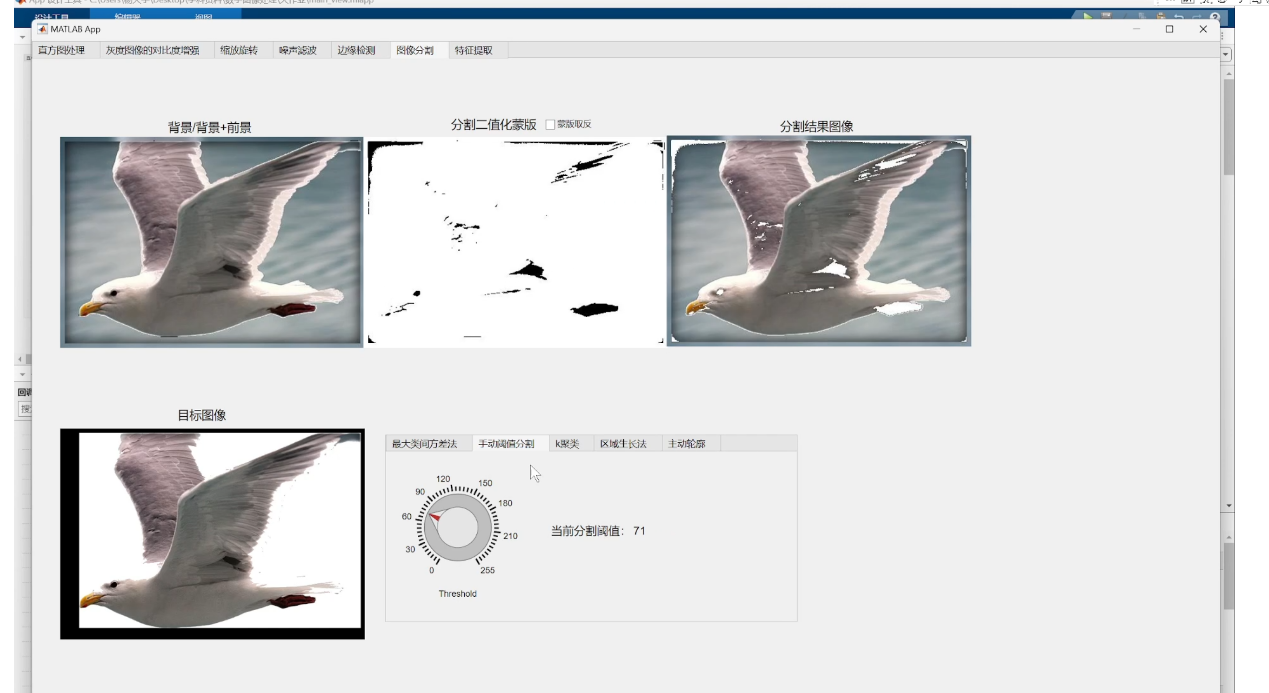
app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 将输入图像转换为灰度图像。
2. 使用 Otsu 最大类间方差法自动计算合适的阈值。
3. 根据该阈值对图像进行二值化。
4. 后处理：去除面积小于 1000 像素的区域。
5. 生成二值图像的彩色蒙版，并叠加回原图，最终显示处理后的图像。

## 6.2 KnobFGtValueChanged - 使用指定阈值进行图像分割



function KnobFGtValueChanged(app, event)

threshold = app.KnobFGt.Value; % 获取阈值

imgData = app.ImgFGinput.ImageSource;

grayImage = rgb2gray(imgData);

app.LabelFGt.Text = round(threshold) + ""; % 显示当前阈值

% 使用自定义阈值进行二值化

binaryImage = imbinarize(grayImage, threshold / 255);

% 后处理

binaryImage = bwareaopen(binaryImage, 1000);

% 创建彩色蒙版并叠加到原图

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage);

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage);

end

segmentedResult = imgData;

indices = colorMask;

segmentedResult(indices) = 255;

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage);

binaryImage = uint8(binaryImage) \* 255;

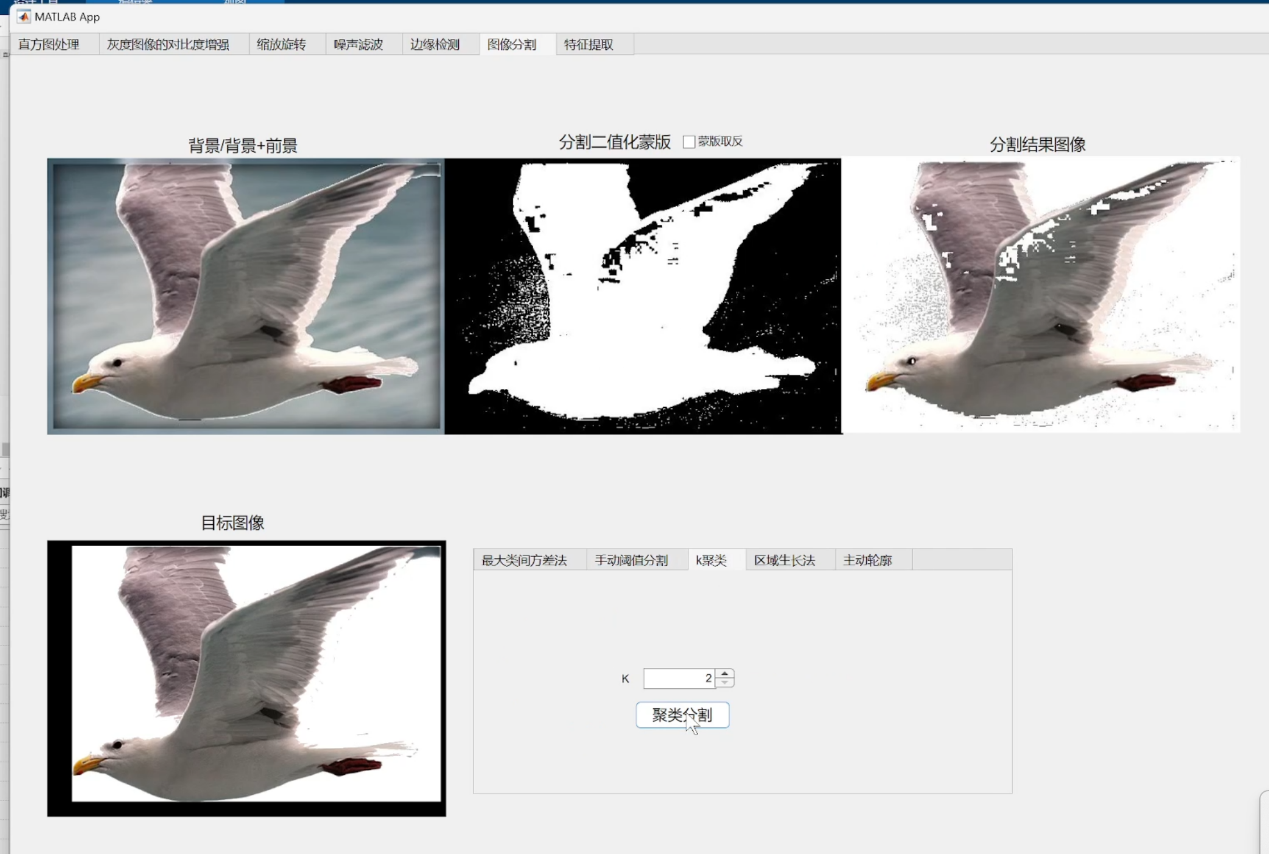
app.ImgFGbin.ImageSource = binaryImage;

app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 获取用户通过旋钮 KnobFGt 控制的阈值。
2. 使用该阈值对灰度图像进行二值化，并进行后处理。
3. 将二值化结果叠加到原始图像上，最后显示处理结果。



## 6.3 SpinnerFGkValueChanged - 使用 K-means 聚类进行图像分割

function SpinnerFGkValueChanged(app, event)

k = app.SpinnerFGk.Value; % 获取聚类的数量

imgData = app.ImgFGinput.ImageSource;

binaryImage = clusterImage(app, imgData, k); % 使用聚类分割图像

% 创建彩色蒙版并叠加到原图

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage);

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage);

end

segmentedResult = imgData;

indices = colorMask;

segmentedResult(indices) = 255;

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage);

binaryImage = uint8(binaryImage) \* 255;

app.ImgFGbin.ImageSource = binaryImage;

app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 使用 K-means 聚类对图像进行分割。
2. clusterImage 是一个用户定义的函数，负责基于 K-means 聚类算法对图像进行处理。
3. 创建并叠加彩色蒙版到原图，显示分割结果。

## 6.4 CheckBoxFGreverseValueChanged - 控制蒙版的反转

function CheckBoxFGreverseValueChanged(app, event)

value = app.CheckBoxFGreverse.Value;

app.FGreverse = value; % 记录反转设置

end

功能：

控制是否反转蒙版（决定图像的显示是黑色背景还是白色背景）。

## 6.5 KnobFGhoughValueChanged - 霍夫变换进行图像分割

function KnobFGhoughValueChanged(app, event)

threshold = app.KnobFGhough.Value;

imgData = app.ImgFGinput.ImageSource;

app.LabelFGhough.Text = round(threshold) + ""; % 显示阈值

% 使用霍夫变换分割图像

binaryImage = houghSegmentation(app, imgData, threshold);

% 创建彩色蒙版并叠加到原图

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage);

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage);

end

segmentedResult = imgData;

indices = colorMask;

segmentedResult(indices) = 255;

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage);

binaryImage = uint8(binaryImage) \* 255;

app.ImgFGbin.ImageSource = binaryImage;

app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 使用霍夫变换算法对图像进行分割。
2. 更新分割结果的显示，二值化后的图像叠加到原图上。

## 6.6 BtnseedPushed - 区域生长法进行图像分割

function BtnseedPushed(app, event)

binaryImage = interactiveRegionGrowing(app, app.ImgFGinput.ImageSource); % 交互式区域生长分割

imgData = app.ImgFGinput.ImageSource;

% 创建彩色蒙版并叠加到原图

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage);

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage);

end

segmentedResult = imgData;

indices = colorMask;

segmentedResult(indices) = 255;

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage);

binaryImage = uint8(binaryImage) \* 255;

app.ImgFGbin.ImageSource = binaryImage;

app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 使用交互式区域生长方法进行图像分割。
2. interactiveRegionGrowing 是用户定义的函数，用于实现区域生长。
3. 最终将二值图像和原图叠加显示。

## 6.7 BtnactiveButtonPushed - 主动轮廓模型进行图像分割

function BtnactiveButtonPushed(app, event)

imgData = app.ImgFGinput.ImageSource;

I = rgb2gray(imgData);

% 显示原始图像

imshow(I);

title('Original Image');

r = drawpolygon; % 手动绘制初始轮廓

mask = createMask(r); % 创建初始轮廓的蒙版

imshow(mask);

title('Initial Contour Location');

% 使用主动轮廓模型进行分割

binaryImage = activecontour(I, mask, 1500, 'edge');

rgbImage = cat(3, binaryImage, binaryImage, binaryImage); % 转换为RGB图像

rgbImage = uint8(rgbImage \* 255);

app.ImgFGbin.ImageSource = rgbImage;

% 创建彩色蒙版并叠加到原图

if app.FGreverse == false

colorMask = ~cat(3, binaryImage, binaryImage, binaryImage);

else

colorMask = cat(3, binaryImage, binaryImage, binaryImage);

end

segmentedResult = imgData;

indices = colorMask;

segmentedResult(indices) = 255;

% 更新图像显示

binaryImage = cat(3, binaryImage, binaryImage, binaryImage);

binaryImage = uint8(binaryImage) \* 255;

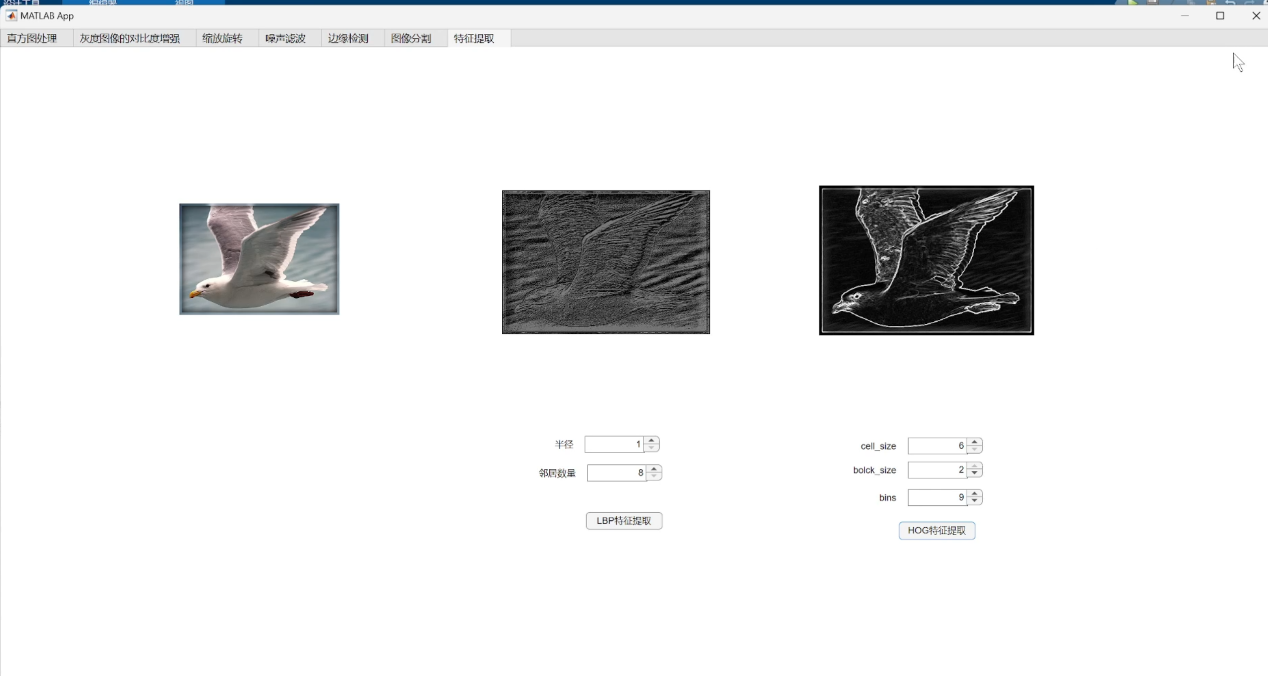
app.ImgFGres.ImageSource = segmentedResult;

end

**功能**：

1. 使用主动轮廓模型（Active Contour Model）进行图像分割。
2. 用户绘制初始轮廓，主动轮廓模型在此基础上进行图像分割。
3. 结果显示在图像上，最后输出分割结果。

# 七 图像特征提取



## 7.1 LBP特征提取按钮功能 (LBPButtonPushed)

function LBPButtonPushed(app, event)

% 进行LBP特征提取

radius = app.SpinnerNFgVar\_2.Value; % 获取半径值

neighbors = app.SpinnerNFgVar\_3.Value; % 获取邻居数目

img = imread(app.Original.ImageSource); % 读取原始图像

imgData = rgb2gray(img); % 将图像转换为灰度图

[~, lbp\_image] = lbp\_histogram(app, imgData, radius, neighbors); % 调用LBP特征提取函数

% lbp\_image\_rgb = repmat(lbp\_image, [1, 1, 3]); % 另一种创建RGB图像的方式

lbp\_image\_rgb = cat(3, lbp\_image, lbp\_image, lbp\_image); % 将LBP图像扩展为RGB图像

% 设置图像源

app.Image.ImageSource = uint8(lbp\_image\_rgb); % 显示处理后的图像

end

**代码解析：**

1. **读取和处理图像**：
   1. 代码通过imread函数读取原始图像。
   2. 将图像转换为灰度图，采用rgb2gray函数，灰度图像是后续特征提取的基础。
2. **LBP特征提取**：
   1. lbp\_histogram是一个调用的函数，负责计算局部二值模式（LBP）。LBP是一种纹理特征，用于图像分类、识别等任务。
   2. radius和neighbors分别控制LBP操作中的半径和邻域大小。
3. **生成RGB图像**：
   1. lbp\_image是处理后的单通道图像（灰度），cat(3, lbp\_image, lbp\_image, lbp\_image)将其扩展为三通道RGB图像，用于显示。
4. **显示结果**：
   1. 通过app.Image.ImageSource将生成的RGB图像展示在UI上。

**LBP算法原理：**

1. \*\*局部二值模式（LBP）\*\*是一种纹理描述符。它通过计算像素与其邻域像素的比较值，生成一个二进制数值来表示局部纹理。
2. radius控制邻域的范围，neighbors控制邻域内像素的数量，常见的设置为8邻域、1像素半径。

## 7.2 LBP 特征提取

**函数：lbp\_histogram**

**功能：** 提取图像的局部二值模式（LBP）特征图，并统计特征直方图。

function [histogram, lbp\_image] = lbp\_histogram(app,image, radius, neighbors)

% 初始化 LBP 图像和直方图

[height, width] = size(image);

lbp\_image = zeros(height, width);

histogram = zeros(2^neighbors, 1);

% 遍历每个像素

for x = 1 + radius : width - radius

for y = 1 + radius : height - radius

% 获取邻域像素

neighborhood = getNeighborhood(app,image, x, y, radius, neighbors);

% 计算 LBP 值

lbp\_value = calculateLBP(app,neighborhood);

% 更新 LBP 图像和直方图

lbp\_image(y, x) = lbp\_value;

histogram(lbp\_value + 1) = histogram(lbp\_value + 1) + 1;

end

end

end

**详细解释：**

1. **输入参数：**
   1. image：输入灰度图像。
   2. radius：计算邻域时的半径。
   3. neighbors：邻域中采样的点数。
2. **核心思路：**
   1. 对每个像素，计算其邻域像素值与中心像素值的关系。
   2. 通过将大于等于中心像素值的点置为 1，小于的置为 0，得到二进制模式。
   3. 将二进制模式转换为十进制值作为 LBP 特征。
3. **正确性分析：**
   1. 直方图数组大小为 2^neighbors，表示所有可能的 LBP 模式。
   2. 计算过程中，确保了像素边界 radius 外的像素不被访问，避免数组越界。

## 7.3 函数：getNeighborhood

**功能：** 获取指定像素的邻域像素值。

function neighborhood = getNeighborhood(~,image, x, y, radius, neighbors)

angle\_step = 2 \* pi / neighbors;

neighborhood = zeros(1, neighbors);

for i = 1:neighbors

angle = angle\_step \* (i - 1);

offset\_x = round(radius \* cos(angle));

offset\_y = round(radius \* sin(angle));

neighborhood(i) = image(y + offset\_y, x + offset\_x);

end

end

**详细解释：**

1. **算法原理：**
   1. 使用极坐标计算邻域点的位置，radius 表示距离，angle\_step 控制均匀分布的角度。
2. **关键点：**
   1. 利用 cos 和 sin 计算偏移量，并通过四舍五入确保索引为整数。
   2. 邻域像素值存储在 neighborhood 中。
3. **正确性分析：**
   1. 通过均匀采样确保了邻域点的分布一致性。
   2. 适用于任意采样点数（如 8 或 16）。

## 7.4 函数：calculateLBP

**功能：** 根据邻域像素计算 LBP 值。

function lbp\_value = calculateLBP(~,neighborhood)

center\_pixel = neighborhood(1);

lbp\_value = 0;

for i = 2:length(neighborhood)

if neighborhood(i) >= center\_pixel

lbp\_value = lbp\_value + 2^(i - 2);

end

end

end

1. **算法原理：**
   1. 以中心像素值为基准，将邻域像素值与其比较。
   2. 使用二进制编码方式，将满足条件的像素赋值为 1，否则为 0。
2. **关键点：**
   1. 循环遍历邻域点，利用位权加权（2^(i-2)）计算 LBP 值。
3. **正确性分析：**
   1. 保证每个邻域点的编码唯一且无丢失。
   2. 避免循环边界错误。

## 7.5 HOG特征提取按钮功能 (HOGButtonPushed)

function HOGButtonPushed(app, event)

cell\_size = app.SpinnerNFgVar\_4.Value; % 获取单元格大小

block\_size = app.SpinnerNFgVar\_5.Value; % 获取块大小

bins = app.SpinnerNFgVar\_6.Value; % 获取直方图的bin数

img = imread(app.Original.ImageSource); % 读取原始图像

imgData = rgb2gray(img); % 转换为灰度图

imgData = double(imgData); % 转换为double类型，便于后续处理

% 调用HOG特征提取函数

[~, hog\_image] = hog\_feature(app, imgData, cell\_size, block\_size, bins);

% 将HOG特征图像转换为RGB显示

hog\_image\_rgb = cat(3, hog\_image, hog\_image, hog\_image);

app.Image\_3.ImageSource = uint8(hog\_image\_rgb); % 显示HOG特征图像

end

1. **获取HOG参数**：
   1. 通过UI控件获取HOG特征提取的参数：cell\_size、block\_size和bins。
2. **图像处理**：
   1. 同样地，图像转换为灰度图，并转换为double类型，以便进行数值计算。
3. **调用HOG特征提取函数**：
   1. hog\_feature函数负责提取图像的HOG特征。HOG（Histogram of Oriented Gradients）特征广泛用于图像分类、物体检测等领域。
   2. HOG特征提取的核心思想是：通过计算图像局部区域的梯度方向直方图来描述图像的形状特征。
4. **显示HOG图像**：
   1. 将处理后的HOG图像转换为RGB格式，并通过app.Image\_3.ImageSource显示在UI中。

**HOG算法原理：**

**HOG**通过对图像局部区域计算梯度，并将其转化为方向直方图来描述图像特征。通过对这些局部特征进行归一化处理，能够有效捕捉图像的形状特征。

## 7.6 HOG 特征提取

**函数：hog\_feature**

**功能：** 提取图像的 HOG 特征和对应的可视化图像。

function [hog\_features, hog\_image] = hog\_feature(app,image, cell\_size, block\_size, bins)

% 将输入图像转换为双精度

image = double(image);

% 计算梯度（水平和垂直方向）

[Gx, Gy] = imgradientxy(image);

% 计算梯度幅度和方向

[mag, ang] = imgradient(Gx, Gy);

**详细解释：**

1. **梯度计算：**
   1. 使用 imgradientxy 计算水平（Gx）和垂直（Gy）梯度。
   2. 利用 imgradient 进一步得到梯度幅度（mag）和方向（ang）。
2. **可视化意义：**
   1. 梯度方向和幅度可用于区分边缘和纹理信息。
3. **后续步骤：**
   1. 将图像分为小块，分别计算每块的梯度方向直方图。

## 7.7 函数：normalize\_hog

**功能：** 对 HOG 特征进行块级归一化。

function hog\_features = normalize\_hog(~,hog\_features, block\_size, num\_cells\_x, num\_cells\_y)

num\_blocks\_x = num\_cells\_x - block\_size + 1;

num\_blocks\_y = num\_cells\_y - block\_size + 1;

normalized\_hog\_features = [];

for i = 1:num\_blocks\_y

for j = 1:num\_blocks\_x

block\_start\_idx = (i-1) \* num\_cells\_x + j;

block\_end\_idx = block\_start\_idx + block\_size \* block\_size - 1;

block = hog\_features(block\_start\_idx:block\_end\_idx);

% L2 归一化

norm\_factor = norm(block, 2);

if norm\_factor > 0

block = block / norm\_factor;

end

normalized\_hog\_features = [normalized\_hog\_features, block];

end

end

hog\_features = normalized\_hog\_features;

end

**详细解释：**

1. **归一化目的：**
   1. 降低光照变化的影响，使特征更鲁棒。
2. **L2 正则化：**
   1. 使用欧几里得范数将块特征值归一化。
3. **块级划分：**
   1. 将相邻 cell 组合成块（block），并进行归一化。

# 八 图像处理大作业心得体会

本次大作业围绕图像处理的多个功能展开，我通过 MATLAB 编写了完整的程序，并设计了交互式 GUI 界面，用户可以通过按钮或菜单实现功能选择并查看运行结果。所有核心功能均通过自己编写的代码实现，深刻体验到了从理论到实践的全过程。这次实践让我对图像处理的基本理论和算法实现有了更加深刻的认识。

首先，我通过编写代码实现了直方图均衡化和匹配、对比度增强、图像几何变换等基本功能。例如，在直方图均衡化部分，我直接计算图像灰度的累计分布函数，并通过映射完成直方图的均衡化，而不是简单调用 MATLAB 自带的函数。对于对比度增强，我分别实现了线性和非线性变换，其中对数和指数变换的实现让我更加理解这些方法对灰度动态范围的调整原理。图像的缩放和旋转变换则通过构建变换矩阵并手动插值完成，充分锻炼了我对基础图像处理技术的理解和动手能力。

在高级功能方面，我完成了边缘提取与特征提取等任务。其中，边缘提取部分使用了 Roberts、Prewitt、Sobel 和拉普拉斯算子，我手动实现了这些算法的卷积操作，而不是依赖 MATLAB 内置的 edge 函数。此外，在特征提取中，我编写了 LBP 和 HOG 的代码，通过手动划分图像块、计算梯度方向直方图等操作，真正理解了这些特征提取方法的核心原理。这些功能让我更加熟悉算法设计和实现的细节，也锻炼了我的逻辑思维能力。

最后，在加分项机器学习部分，我仅尝试使用传统方法（如支持向量机）对提取的特征进行分类，由于时间有限，仅对小型数据集进行了实验，但这让我初步了解了如何将特征与模型结合，完成简单的图像分类任务。

总体而言，本次大作业让我在理论与实践结合中获益匪浅，尤其是在自主实现算法的过程中，我对图像处理的基本原理有了更深刻的理解。同时，我也认识到自己的不足之处，例如界面设计还需优化，代码结构可以进一步模块化。未来，我希望在代码优化、模型应用等方面不断提升自己，为更复杂的图像处理任务打下坚实基础。

附：Github账号链接：<https://github.com/Chrismayan>

大作业位置链接：<https://github.com/Chrismayan/zyy19220711>