机器视觉第二次作业

1、自己构建一个边缘检测器（以 matlab、octave、C++均可）。注释每一步的含义。（可以参考书中习题 4.7 和 4.8，观察与 Canny 等边缘检测算子实现效果的差异）

通过 matlab 构建了 Sobel、Robert 和 Canny 算子进行比较，以下是对于代码的解释；

（1）Sobel 算子；

代码如图 1 所示，Sobel 算子基于梯度的计算，通过对图像进行水平和垂直方向的卷积操作，得到图像在这两个方向上的梯度信息。然后计算梯度幅值，使用幅值来表示图像中的边缘强度。Sobel 算子的优点是能够有效地检测边缘，但对于噪声的敏感度较高。

代码中设置了 sobelGx 和 sobelGy 用于计算水平和垂直梯度的 Sobel 滤波器。使用 conv2 函数将图像和 Sobel 算子进行卷积，得到图像的水平梯度 sobelGradientX 和垂直梯度 sobelGradientY。通过计算两个方向梯度的平方和的平方根，得到梯度幅值 sobelEdge。

```
% Sobel 算子
sobelGx = [-1 0 1; -2 0 2; -1 0 1];
sobelGy = [-1 -2 -1; 0 0 0; 1 2 1];
% 卷积计算梯度
sobelGradientX = conv2(double(grayImg), sobelGx, 'same');
sobelGradientY = conv2(double(grayImg), sobelGy, 'same');
% 计算梯度的幅值
sobelEdge = sqrt(sobelGradientX.^2 + sobelGradientY.^2);
% 显示结果
subplot(1,4,2);
imshow(sobelEdge, []);
title('Sobel');
```

图 1 Sobel 算子代码实现

（2）Robert 算子：

Robert 算子与 Sobel 算子类似，但其核心思想是通过一个更小的卷积核（2x2 矩阵）计算图像的局部梯度。由于其计算简单，适合于检测图像中较为细微的边缘，但对噪声的敏感度也较高。

代码中同样是设置了两个变量用于计算水平和垂直梯度的 Robert 算子，并将其与图像进行卷积得到图像在两个方向上的梯度，随后计算梯度幅值并输出成一个图像。

```
% Robert 算子
robertGx = [1 0; 0 -1];
robertGy = [0 1; -1 0];

% 卷积计算梯度
robertGradientX = conv2(double(grayImg), robertGx, 'same');
robertGradientY = conv2(double(grayImg), robertGy, 'same');

% 计算梯度的幅值
robertEdge = sqrt(robertGradientX.^2 + robertGradientY.^2);

% 显示结果
subplot(1,4,3);
imshow(robertEdge, []);
title('Robert');
```

图 2 Robert 算子代码实现

（3）Canny 算子：

Canny 算子则是一个更为复杂的边缘检测算法，包含多个步骤，旨在生成更精确的边缘图像。首先，Canny 算子使用高斯滤波器平滑图像，以减少噪声的影响；接着，计算图像的梯度并通过非最大抑制去除非边缘的像素；最后应用双阈值处理，将强边缘与弱边缘区分开来，并通过边缘连接保持图像的完整性。Canny 算子通过这些步骤能够在保留边缘信息的同时，抑制噪声并减少误检，通常被认为是最优的边缘检测方法。

```
%1. 高斯平滑
gaussianKernel = fspecial('gaussian', [5 5], 1); % 5x5
smoothedImg = conv2(double(grayImg), gaussianKernel, 'same');

%2. 计算梯度
gx = [-1 0 1; -2 0 2; -1 0 1];
gy = [-1 -2 -1; 0 0 0; 1 2 1];

gradientX = conv2(smoothedImg, gx, 'same');
gradientY = conv2(smoothedImg, gy, 'same');

% 计算梯度幅值和方向
gradientMagnitude = sqrt(gradientX.^2 + gradientY.^2);
gradientDirection = atan2(gradientY, gradientX) * 180 / pi;

%3. 非最大抑制
nonMaxSuppressed = edgeNonMaxSuppression(gradientMagnitude, gradientDirection);

%4. 双阈值处理
highThreshold = 0.1 * max(gradientMagnitude(:));
lowThreshold = 0.05 * highThreshold;

%5. 边缘连接
cannyEdge = edgeLinking(nonMaxSuppressed, highThreshold, lowThreshold);

%显示 Canny 边缘检测结果
subplot(1,4,4);
imshow(cannyEdge, []);
title('Canny');
```

```
%非最大抑制
function suppressedImg = edgeNonMaxSuppression(magnitude, direction)
    [rows, cols] = size(magnitude);
    suppressedImg = zeros(rows, cols);
    direction = mod(direction, 180); % 确保方向在 0 到 180 之间

    for i = 2:rows-1
        for j = 2:cols-1
            angle = direction(i,j);
            magnitudeValue = magnitude(i,j);

            % 0 - 180 degrees
            if (angle >= 0 && angle < 45) || (angle >= 135 && angle < 180)
                neighbor1 = magnitude(i, j+1);
                neighbor2 = magnitude(i, j-1);
            elseif (angle >= 45 && angle < 90) || (angle >= 90 && angle < 135)
                neighbor1 = magnitude(i+1, j);
                neighbor2 = magnitude(i-1, j);
            end

            % 非最大抑制
            if magnitudeValue >= neighbor1 && magnitudeValue >= neighbor2
                suppressedImg(i,j) = magnitudeValue;
            else
                suppressedImg(i,j) = 0;
            end
        end
    end
end
```

```
%边缘连接
function edgeImg = edgeLinking(magnitude, highThreshold, lowThreshold)
    [rows, cols] = size(magnitude);
    edgeImg = zeros(rows, cols);

    % 将大于高阈值的边缘标记为强边缘
    edgeImg(magnitude > highThreshold) = 1;

    % 对于小于低阈值的像素，设置为 0
    edgeImg(magnitude < lowThreshold) = 0;

    % 对于介于低和高阈值之间的像素，通过边缘连接判断
    for i = 2:rows-1
        for j = 2:cols-1
            if magnitude(i,j) >= lowThreshold && magnitude(i,j) <= highThreshold
                % 如果与强边缘相连，标记为边缘
                if any(edgeImg(i-1:i+1, j-1:j+1) == 1)
                    edgeImg(i,j) = 1;
                end
            end
        end
    end
end
```

图 3 Canny 算子代码实现

分别使用三种算子对同一张图片进行测试，如图 4 所示，在测试中发现，Sobel、Robert 和 Canny 算子各有特点。Sobel 算子使用 3x3 卷积核，适合检测较强的边缘，对噪声有一定容忍度，但在细节捕捉上较弱。Robert 算子使用更小的 2x2 卷积核，能够捕捉细小的边缘，但对噪声较为敏感，适合噪声较少且边缘锐利的图像。Canny 算子则通过多步骤处理（如高斯平滑、梯度计算和非最大抑制），能够精确地检测边缘并有效去噪，适用于高精度要求的应用，虽然计算复杂且处理速度较慢。简而言之，Sobel 适用于简单场景，Robert 适合细节边缘，Canny 在精度和噪声抑制上最强。

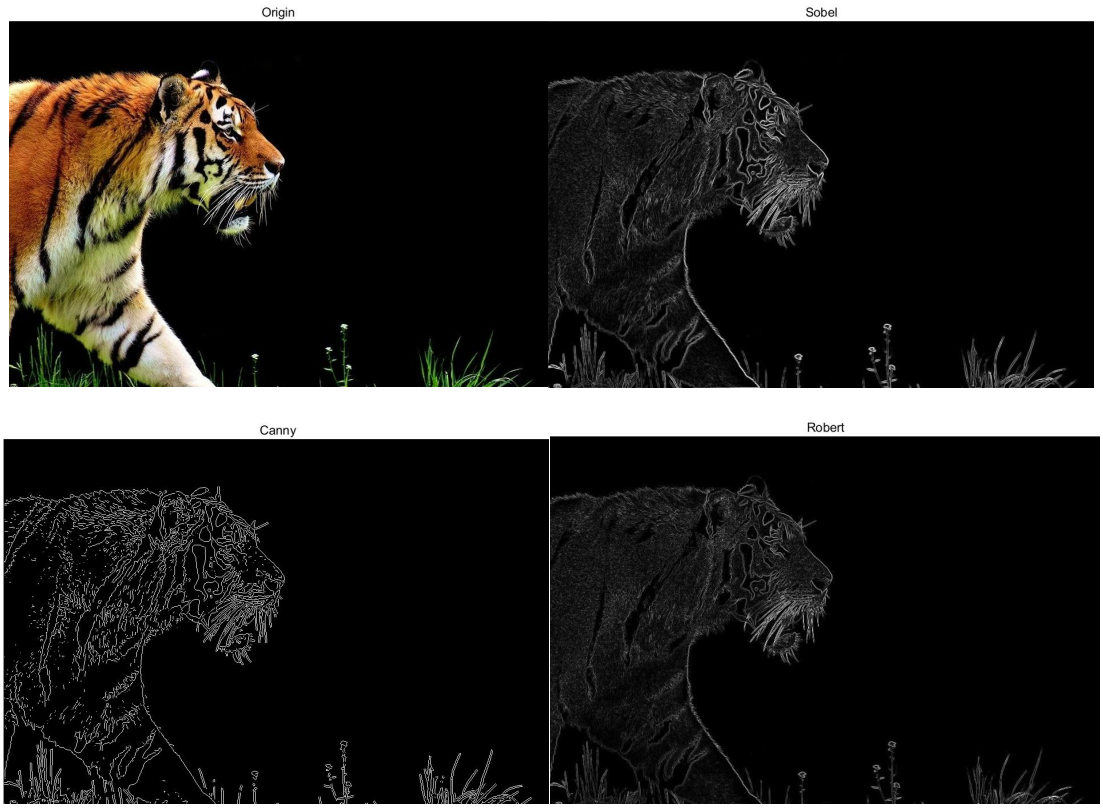

图 4 测试结果

2、阅读论文：Seam Carving for Content-Aware Image Resizing ，Shai & Avidan, ACM SIGGRAPH 2007。

这篇文章主要介绍了一种名为 Seam Carving 的图像处理技术用于实现内容感知的图像缩放。文章提出了一种方法，通过移除或插入图像中的“缝合线”（即在能量函数下由低能量像素构成的路径），实现图像尺寸的灵活调整（包括缩小和放大），同时尽可能保留图像的视觉重要内容。以下通过三个部分进行文章中的内容介绍。

（1）文章背景与 idea

随着显示设备的多样化，如手机、平板和 PC，对图像尺寸适配的需求日益增加。传统的图像缩放（Scaling）和裁剪（Cropping）方法在调整图像尺寸时，通常忽略内容的重要性，导致信息丢失或图像失真。而近年来，一些基于显著性检测的内容调整方法试图解决这一问题，但仍然存在局限性。所以，文章提出了一种名为 Seam Carving 的图像处理技术，通过计算图像中的“低能量路径”（即缝合线），实现内容感知的动态缩放。该方法能够在调整图像尺寸的过程中，智

能识别和保护图像的重要部分，如显著区域或用户指定的内容，从而在视觉效果上更好地满足多设备显示需求。

（2）实现 idea 的方法

Seam Carving 通过以下方法实现内容感知的图像缩放。首先，定义能量函数，采用图像梯度幅值来表示像素的重要性，其中低能量像素更容易被移除。然后，通过动态规划算法计算缝合线，即一条能量最小的 8 连接路径，垂直缝合线从图像顶部到底部，水平缝合线从图像左侧到右侧。通过依次移除缝合线可以实现图像的缩小，而插入缝合线时，则复制原图像中的缝合线并插入人工像素，以实现图像的放大。此外，为提高算法的灵活性，还可以引入交互式用户输入（如标记保护区域）指导缝合线计算，并使用其他能量函数（如视觉显著性或熵）来优化效果。

（3）文中的方法的缺陷

虽然该方法自动处理了大部分图像，但并非所有图像都能获得理想效果。为了改进处理结果，可以引入更高层次的提示信息，这些信息可以是手动的，也可以是自动的。比如可以使用一些检测器或者约束来优化结果。然而，仍然存在一些无法仅通过高层次信息解决的问题。具体来说，有两种情况限制了 Seam Carving 的应用：首先，当图像过于“密集”，即图像中没有明显的“次要”区域时，任何基于内容的缩放策略都会失败。其次，即使图像没有过于密集，图像内容的布局也可能会限制缝合线的选择。例如，如果图像内容布局使得缝合线不得不穿越重要部分，那么无论如何调整，缝合线都无法绕过这些区域，从而影响最终效果。

所以，Seam Carving 作为一种创新的内容感知图像缩放技术，极大地改善了传统图像处理方法在缩放过程中的局限性。通过动态计算图像中的“缝合线”，该方法能够灵活地调整图像尺寸，无论是缩小还是放大，同时尽可能保留图像的视觉重要内容。Seam Carving 的应用领域非常广泛，它不仅可以用来改变图像的宽高比，如将图像适配到不同设备的屏幕尺寸，还能进行图像内容的增强，例如放大图像中的特定区域而不会造成视觉上的失真。并且 Seam Carving 还可以用来移除图像中的不需要部分，如删除图像中的物体或背景而不会影响周围重要内容，从而应用于图像修复和合成等任务。

尽管该方法在许多场景中表现出色，仍存在一定的局限性，特别是在图像过于“密集”或内容布局复杂时，缝合线的计算可能会受到限制。为了克服这些问题，文章提出通过结合高层次的提示信息（如自动检测器、用户交互或约束）来进一步优化结果，提升算法的适应性。这一技术的创新不仅为图像处理提供了新的解决方案，也为多设备、多场景下的动态图像调整提供了潜在的应用前景。因此，Seam Carving 为图像处理、内容创作以及用户交互提供了新的可能性，在未来的数字媒体和多屏互动应用中，将发挥更加重要的作用。