数字图像处理实验报告

人工智能 2204 班

杨鲲

U202215264

实验目的

实验一：对图像（自行转换为灰度图）展开（1）顺时针旋转 30 度；（2）基于最近邻和双线性插值将图像分别放大 2 倍和 4 倍

实验二：对以上图像（自行转换为灰度图）展开傅里叶变换，提取傅里叶变换图像（将频率原点移至图像中心）

实验内容

实验一

代码展示：

```
% 主程序

input_image = imread('1.bmp');  % 读取图像

gray_image = rgb2gray(input_image);  % 转换为灰度图



% 旋转图像

rotated_image = rotate_image(gray_image, 30);

imwrite(rotated_image, 'rotated_image.jpg');
```

```
% 最近邻插值放大 2 倍和 4 倍
enlarged_image_2x_nn = nearest_neighbor_enlarge(gray_image, 2);
imwrite(enlarged_image_2x_nn, 'enlarged_image_2x_nn.jpg');

enlarged_image_4x_nn = nearest_neighbor_enlarge(gray_image, 4);
imwrite(enlarged_image_4x_nn, 'enlarged_image_4x_nn.jpg');

% 双线性插值放大 2 倍和 4 倍
enlarged_image_2x_bilinear = bilinear_enlarge(gray_image, 2);
imwrite(enlarged_image_2x_bilinear, 'enlarged_image_2x_bilinear.jpg');

enlarged_image_4x_bilinear = bilinear_enlarge(gray_image, 4);
imwrite(enlarged_image_4x_bilinear, 'enlarged_image_4x_bilinear.jpg');

disp('处理完成，图像已保存。');

% 旋转图像函数
function rotated_image = rotate_image(input_image, angle)
    theta = angle * (pi / 180);
    [h, w] = size(input_image);
    new_h = ceil(abs(h * cos(theta)) + abs(w * sin(theta)));
    new_w = ceil(abs(h * sin(theta)) + abs(w * cos(theta)));
```

```
    rotated_image = zeros(new_h, new_w);


    center_x = w / 2;
    center_y = h / 2;
    new_center_x = new_w / 2;
    new_center_y = new_h / 2;


    for x = 1:new_w
        for y = 1:new_h
            old_x = (x - new_center_x) * cos(-theta) - (y - new_center_y) *
sin(-theta) + center_x;
            old_y = (x - new_center_x) * sin(-theta) + (y - new_center_y) *
cos(-theta) + center_y;


            if old_x >= 1 && old_x <= w && old_y >= 1 && old_y <= h
                rotated_image(y, x) = input_image(round(old_y),
round(old_x));
            end
        end
    end
```

```
    rotated_image = uint8(rotated_image);
end

% 最近邻插值放大函数
function enlarged_image = nearest_neighbor_enlarge(input_image, scale_factor)
    [h, w] = size(input_image);
    new_h = h * scale_factor;
    new_w = w * scale_factor;

    enlarged_image = zeros(new_h, new_w);

    for x = 1:new_w
        for y = 1:new_h
            old_x = round(x / scale_factor);
            old_y = round(y / scale_factor);

            if old_x >= 1 && old_x <= w && old_y >= 1 && old_y <= h
                enlarged_image(y, x) = input_image(old_y, old_x);
            end
        end
    end
```

```
    enlarged_image = uint8(enlarged_image);
end

% 双线性插值放大函数
function enlarged_image = bilinear_enlarge(input_image, scale_factor)
    [h, w] = size(input_image);
    new_h = h * scale_factor;
    new_w = w * scale_factor;

    enlarged_image = zeros(new_h, new_w);

    for x = 1:new_w
        for y = 1:new_h
            old_x = (x - 1) / scale_factor + 1;
            old_y = (y - 1) / scale_factor + 1;

            left = floor(old_x);
            right = ceil(old_x);
            top = floor(old_y);
            bottom = ceil(old_y);
```

```
                if left >= 1 && right <= w && top >= 1 && bottom <= h
                    a = old_x - left;
                    b = old_y - top;

                    top_left = input_image(top, left);
                    top_right = input_image(top, right);
                    bottom_left = input_image(bottom, left);
                    bottom_right = input_image(bottom, right);

                    top_interpolated = top_left * (1 - a) + top_right * a;
                    bottom_interpolated = bottom_left * (1 - a) +
bottom_right * a;
                    enlarged_image(y, x) = top_interpolated * (1 - b) +
bottom_interpolated * b;
                end
            end
        end

        enlarged_image = uint8(enlarged_image);
end
```

代码解释：

此实验中

- rotated_image.jpg：旋转后的图像
- enlarged_image_2x_nn.jpg：使用最近邻插值放大 2 倍后的图像
- enlarged_image_4x_nn.jpg：使用最近邻插值放大 4 倍后的图像
- enlarged_image_2x_bilinear.jpg：使用双线性插值放大 2 倍后的图像
- enlarged_image_4x_bilinear.jpg：使用双线性插值放大 4 倍后的图像

主要函数包括旋转图像函数（rotated_image）、最近邻插值放大函数（nearest_neighbor_enlarge）和双线性插值放大函数（bilinear_enlarge）

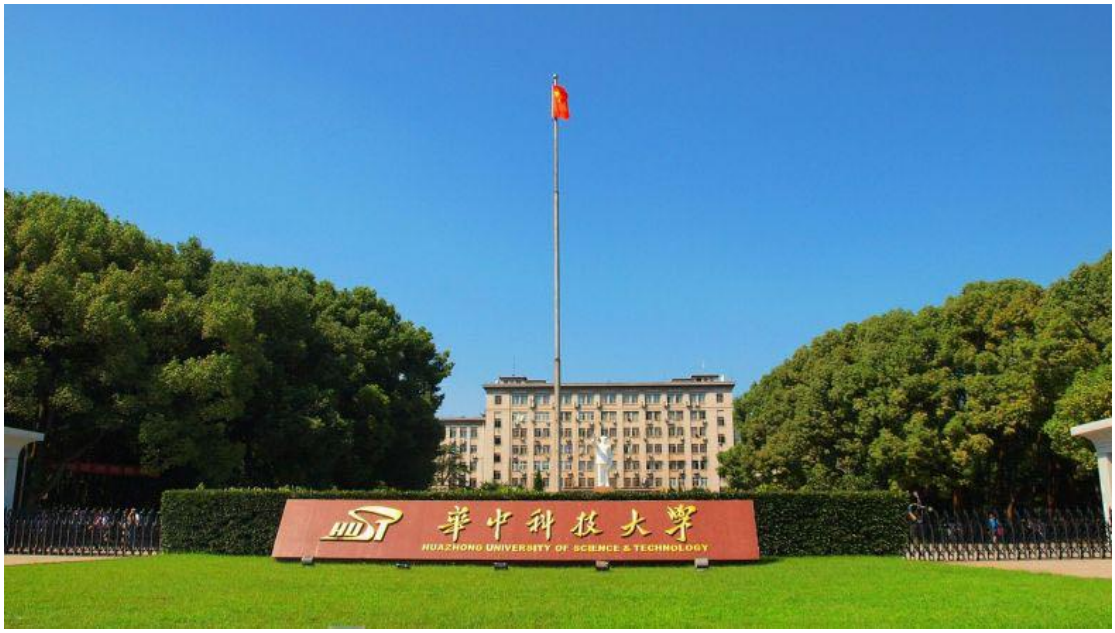
其中旋转图像函数需要将角度转换为弧度，获取输入图像尺寸，计算旋转后的图像尺寸，对于输出后的图像矩阵，需要四舍五入来计算每个像素点的像素，最后将图像转换为 8 位；

对于最近邻插值放大函数，需要获取输入、输出图像的尺寸，对于输出图像矩阵，需要计算最近邻的原像素位置以及进行边界检查，最后将图像转换为 8 位；

对于双线性插值放大函数，需要获取输入、输出图像的尺寸，计算原像素的位置和插值位置，对于边界检查，需要进行双线性插值，最后将图像转换为 8 位。

输出结果：

原图像（785*442 像素）



顺时针旋转 30 度（901*776 像素）

最近邻放大 2 倍（1570*884 像素）

最近邻放大 4 倍（3140*1768 像素）

双线性插值放大 2 倍（1570*884 像素）

双线性插值放大 4 倍（3140*1768 像素）

实验二

代码展示

```
img = imread('1.bmp');

grayImg = rgb2gray(img);
```

```
% 将图像转换为双精度
grayImg = double(grayImg);

% 获取图像的尺寸
[M, N] = size(grayImg);

% 初始化傅里叶变换结果
F = zeros(M, N);

% 计算二维傅里叶变换
for u = 1:M
    for v = 1:N
        % 计算傅里叶变换的每个频率
        for x = 1:M
            for y = 1:N
                F(u, v) = F(u, v) + grayImg(x, y) * exp(-2 * pi * 1i * ((u - 1)
* (x - 1) / M + (v - 1) * (y - 1) / N));
            end
        end
    end
end
```

```
% 将频率原点移至图像中心
F_shifted = fftshift(F);


% 计算幅度谱并取对数（加 1 避免对数为负）
magnitude = log(abs(F_shifted) + 1);


% 显示结果
figure;
subplot(1, 2, 1);
imshow(grayImg, []);
title('原始灰度图');


subplot(1, 2, 2);
imshow(magnitude, []);
title('傅里叶变换幅度谱');
```

代码解释：

对于傅里叶变换，这里需要初始化一个与原图像大小相同的复数矩阵 F 存储变换结果，然后使用双重循环计算出每个频率 F(u, v) 的值。

对于频率原点移动，这里使用 fftshift 函数将频率原点移至图像中心。

输出结果：

傅里叶变换图像