華中科技大學

数字图像处理实验报告

灰度图像的旋转与放大
图像傅里叶变换

院　　系：人工智能与自动化学院

班　　级：人工智能 2204 班

姓　　名：陆慧敏

学　　号：U202215199

1. 作业要求

1. 编写 MATLAB 程序对名为“实验图像. bmp”的图像转换为灰度图

2. 将灰度图顺时针旋转 30 度

3. 基于最近邻和双线性插值将图像分别放大 2 倍和 4 倍

要求：不使用内置函数，自行编写函数实现几何变换和插值计算过程。

2. 编程工具

MATLAB R2021a

3. 实验过程

3.1 灰度图

首先，使用 MATLAB 中的 imread 函数读取 RGB 彩色图片。接着，由于不使用内置函数，这里使用了加权平均法来进行灰度图转化，将红、绿、蓝通道值按比例相加。权重分别是 0. 2989（红）、0. 5870（绿）和 0. 1140（蓝）。这些权重是基于人眼对不同颜色敏感度的不同得出的较为合适的值。

原图和灰度图对比如下：

图 1 原图与灰度图对比图

3.2 顺时针旋转 30°

首先，定义旋转角度为 30°，并转化为弧度制，并计算出旋转矩阵 R，这是二维平面

中的标准旋转矩阵，用于旋转坐标系。接着，通过三角函数计算旋转后图像所需的最小矩形包络框，得到旋转后的图像高度 new_m 与宽度 new_n。获取新图像中心点后，通过双重循环遍历旋转后图像的每个像素位置 (i, j)，并将新图像的每个像素与原图像的相应像素进行匹配，通过旋转矩阵的逆变换，将新图像坐标（i, j）逆旋转回原图像中的坐标（orig_x，orig_y）。然后使用最近邻插值法，将原图像中与 orig_x 和 orig_y 最接近的像素值赋值给新图像。

完成上述操作后，最终可以获得顺时针旋转 30° 的图片，并进行可视化操作，得到图像如下：

图 2 顺时针旋转 30° 的灰度图（不使用内置函数）

搜索 MATLAB 库函数，可得旋转图像函数为 imrotate。使用函数对图像顺时针旋转 30 度，得到如下的旋转结果：

图 3 顺时针旋转 30°的灰度图（使用内置函数）

通过对比可得，运用近邻法插值得到的旋转后图片与 MATLAB 库函数得到的旋转后图片效果基本一致。

3.3 近邻插值法

首先，定义放大倍数 scale，再通过 size 函数读取图像矩阵的大小。接着，取放大后的行数和列数均为原图像的 scale 倍，创建放大后的图像矩阵。

开始遍历放大后的图像，计算新图像中像素(i,j)对应的原图像的像素位置，将缩小后的坐标四舍五入到最接近的整数，得到坐标位置$(orig_x, orig_y)$。将原图像中 orig_x，orig_y 位置的像素值赋值给新图像中 （i,j） 位置，通过最近邻插值，将原图像中的像素值直接复制到放大后的图像中。

分别设置 scale 为 2 和 4，进行可视化，可以分别得到如下效果图：

图 4 放大两倍后的灰度图（不使用内置函数）

图 5 放大四倍后的灰度图（不使用内置函数）

搜索放大图像的 MATLAB 库函数，可得函数为 imresize。分别使用函数对图像放大 2 倍和 4 倍，得到如下的放大结果：

图 6 放大两倍后的灰度图（使用内置函数）

图 7 放大四倍后灰度图（使用内置函数）

通过对比可得，运用近邻插值法得到的放大后图片与 MATLAB 库函数得到的放大后图片效果基本一致，并且可以看出放大倍数越大，使用近邻插值法在边缘部分处理的缺点更加明显。

3.4 双线性插值法

首先，创建一个大小为 new_m × new_n 的空白图像，用于存储放大后的图像。通过两个嵌套的 for 循环，遍历放大后图像的每一个像素位置 (i, j)，接着计算该放大图像像素对应的原图像中的浮点坐标 （orig_x, orig_y），并使用 floor 和 ceil 函数找到 orig_x 和 orig_y 周围的四个整数坐标（四个像素点），再使用线性插值公式将计算出的像素值赋值给新图像坐标。

分别将 scale 设置为 2 和 4，进行可视化，可以分别得到如下效果图：

图 8 放大两倍后的灰度图（不使用内置函数）

图 9 放大四倍后的灰度图（不使用内置函数）

搜索放大图像的 MATLAB 库函数，可得函数为 imresize。分别使用函数对图像放大 2 倍和 4 倍，得到如下的放大结果：

图 10 放大两倍后的灰度图（使用内置函数）

图 11 放大四倍后的灰度图（使用内置函数）

通过对比可得，运用双线性插值法得到的放大后图片与 MATLAB 库函数得到的放大后图片效果基本一致，并且可以看出双线性插值法在边缘部分的效果要优于最近邻插值法。

3.5 傅里叶变换

首先，读取图像并转换为灰度图像，然后进行离散傅里叶变换，考虑到二维离散傅里叶变换会有四层 for 循环，对于 MATLAB 来说运行较为耗时，所以在这里将而离散傅里叶变换拆分为两个一维离散傅里叶变换，先对每一行进行离散傅里叶变换再对每一列进行离散傅里叶变换。完成离散傅里叶变换后需要将频率中心移到图像中心，方便更直观得观察。交换四个象限得到频谱中心化后的图像。

可视化结果如下：

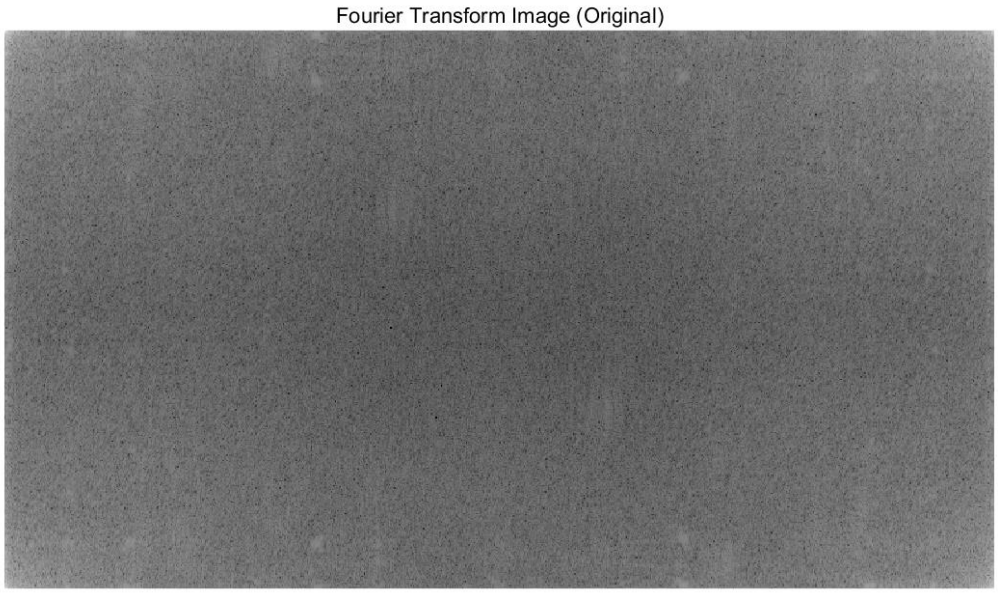

图 12 未中心化的傅里叶变换图像（不使用内置函数）

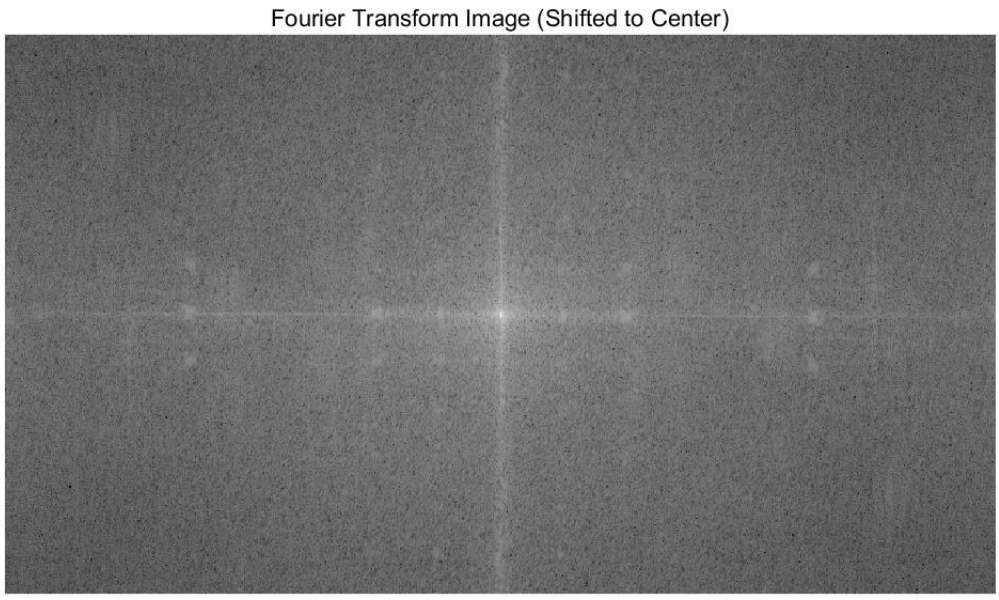

图 13 中心化后的傅里叶变换图像（不使用内置函数）

搜索放大图像的 MATLAB 库函数，可得函数为 fft2 和 fftshift。使用函数对图像进行离散傅里叶变换和频谱中心化，得到可视化图像如下：

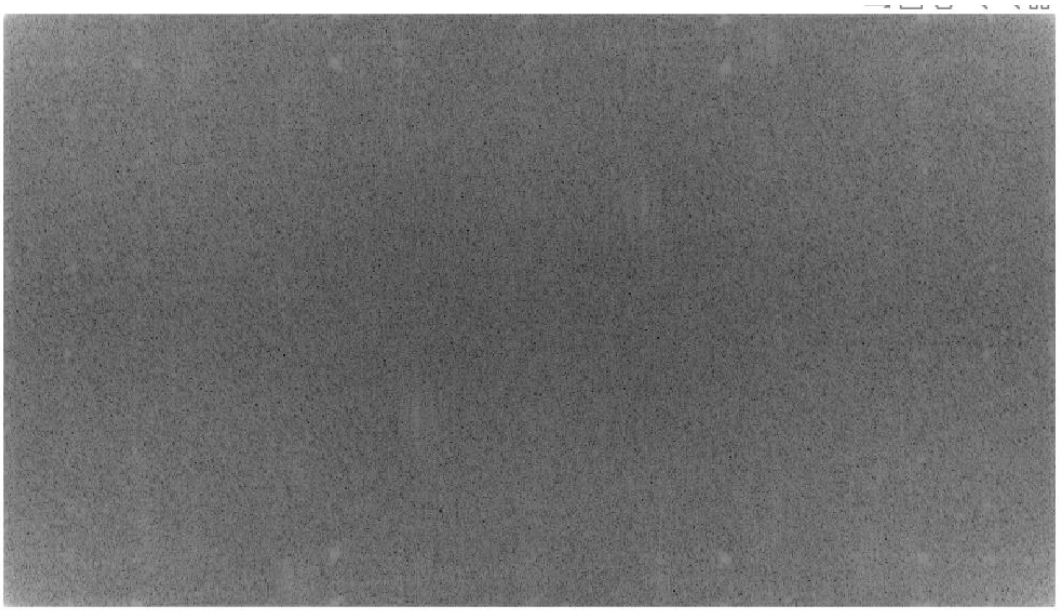

图 14 未中心化的傅里叶变换图像（使用内置函数）

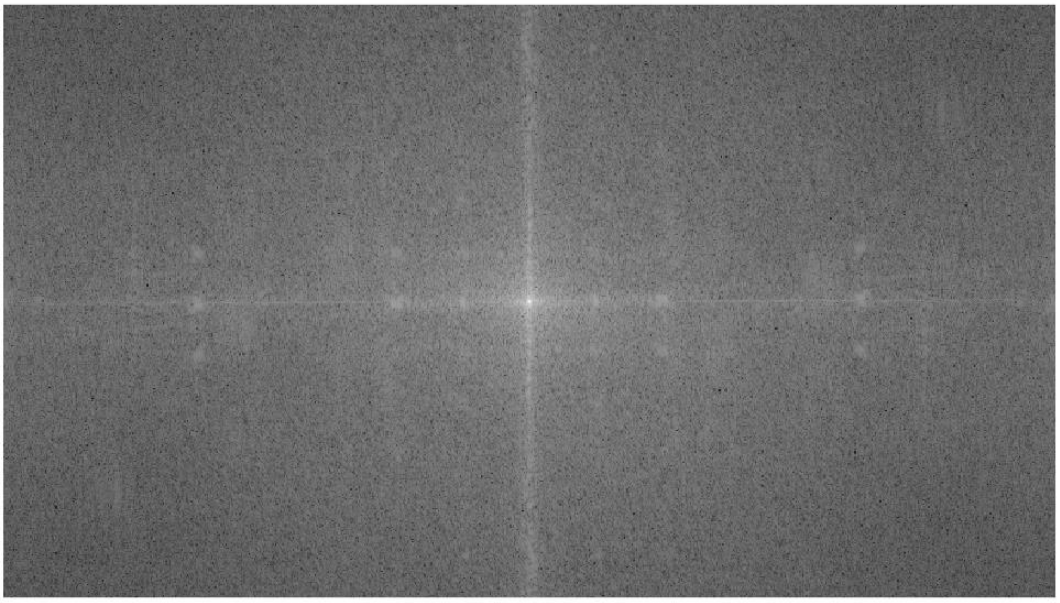

图 15 中心化后的傅里叶变换图像（使用内置函数）

4. 傅里叶变换和傅里叶系数的物理含义

傅里叶变换将一个复杂的信号分解成不同频率的正弦波和余弦波的叠加，它揭示了信号中不同频率成分的分布。高频分量反映了信号中的快速变化或细节，低频分量代表信号的平滑变化或整体趋势。

傅里叶系数是傅里叶变换的输出结果，它代表了不同频率成分的权重或贡献大小。傅里叶系数告诉我们在信号中每个特定频率的正弦波或余弦波的幅值和相位。傅里叶系数的幅度反映了该频率分量在原始信号中的强度或贡献，傅里叶系数的相位表示该频率分量的相位偏移。

5. 心得体会

在完成这两个作业任务的过程中，我加深了对图像处理的基本方法和理论的理解。旋转图像的过程让我熟悉了几何变换的原理，特别是通过旋转矩阵进行坐标转换的方式。通过逆变换找到原始图像的对应像素并赋值给旋转后的图像，这一过程让我意识到旋转时需要谨慎处理边界情况，特别是当图像旋转后尺寸变大时。在图像放大的任务中，使用最近邻和双线性插值法让我直观感受到两种插值方法的差异。最近邻插值虽然简单高效，但会带来“块状”效果，图像显得较为生硬，边缘部分效果不太好；而双线性插值通过周围像素的加权计算，生成的图像更加平滑，更接近原图的细节。这让我深刻体会到不同插值方法的适用场景以及它们在视觉效果上的影响。

傅里叶变换部分让我认识到图像的频率域表示是如何揭示图像的细节信息的。通过将频率原点移动至图像中心，我能更清晰地看到低频和高频信息的分布。

总的来说，完成这些实验不仅强化了我对图像处理算法的掌握，也让我更加注重代码实现时的细节处理和算法效率。

附　件

1、图像放大和旋转程序（不使用内置函数）

```
% 读取图像并转换为灰度图
img = imread('C:\Users\28323\Desktop\第一次作业\实验图像.bmp');
[m, n, c] = size(img);


% 如果是 RGB 图像，进行手动灰度转换
if c == 3
    gray_img = 0.2989 * double(img(:,:,1)) + 0.5870 * double(img(:,:,2)) + 0.1140
* double(img(:,:,3));
else
    gray_img = double(img); % 已经是灰度图
end


% 将灰度图数据类型转换为 double
gray_img = double(gray_img);
```

```
% 定义旋转角度 (30 度转换为弧度)
theta = -30 * pi / 180;


% 获取旋转矩阵
R = [cos(theta) -sin(theta); sin(theta) cos(theta)];


% 获取原图像尺寸
[m, n] = size(gray_img);
```

```
% 计算旋转后新图像的尺寸，考虑边界
new_m = round(abs(m*cos(theta)) + abs(n*sin(theta)));
new_n = round(abs(n*cos(theta)) + abs(m*sin(theta)));


% 创建一个空白的新图像
rotated_img = zeros(new_m, new_n);


% 获取中心点
cx = m / 2;
cy = n / 2;
new_cx = new_m / 2;
new_cy = new_n / 2;


% 遍历新图像的每个像素，找到其对应的原图像的像素
for i = 1:new_m
    for j = 1:new_n
        % 逆变换到原图像坐标
        original_coords = R \ ([i; j] - [new_cx; new_cy]) + [cx; cy];
        orig_x = original_coords(1);
        orig_y = original_coords(2);

        % 使用最近邻插值找到最近的像素
        if orig_x >= 1 && orig_x <= m && orig_y >= 1 && orig_y <= n
            rotated_img(i, j) = gray_img(round(orig_x), round(orig_y));
        end
    end
end


% 显示旋转后的图像
imshow(uint8(rotated_img));
```

```
% 定义放大倍数（这里以 2 倍为例）
scale = 4;


% 获取原始图像尺寸
```

```
[m, n] = size(gray_img);


% 定义放大后的图像尺寸
new_m = round(m * scale);
new_n = round(n * scale);


% 创建空白的新图像
resized_img_nearest = zeros(new_m, new_n);


% 遍历新图像的每个像素，找到其对应的原图像的像素
for i = 1:new_m
    for j = 1:new_n
        % 对应的原图像坐标
        orig_x = round(i / scale);
        orig_y = round(j / scale);

        % 确保坐标不超出边界
        orig_x = max(min(orig_x, m), 1);
        orig_y = max(min(orig_y, n), 1);

        % 赋值最近的像素
        resized_img_nearest(i, j) = gray_img(orig_x, orig_y);
    end
end


% 显示最近邻插值放大后的图像
imshow(uint8(resized_img_nearest));
```

```
% 创建一个空白的新图像用于双线性插值
resized_img_bilinear = zeros(new_m, new_n);


% 遍历新图像的每个像素
for i = 1:new_m
    for j = 1:new_n
        % 计算对应的原图坐标（浮点数）
        orig_x = (i - 0.5) / scale + 0.5;
        orig_y = (j - 0.5) / scale + 0.5;
```

```
        % 找到其周围四个像素的坐标
        x1 = floor(orig_x); x2 = ceil(orig_x);
        y1 = floor(orig_y); y2 = ceil(orig_y);

        % 防止坐标超出边界
        x1 = max(min(x1, m), 1);
        x2 = max(min(x2, m), 1);
        y1 = max(min(y1, n), 1);
        y2 = max(min(y2, n), 1);

        % 计算插值权重
        dx = orig_x - x1;
        dy = orig_y - y1;

        % 双线性插值公式
        pixel_value = (1 - dx) * (1 - dy) * gray_img(x1, y1) + ...
                      dx * (1 - dy) * gray_img(x2, y1) + ...
                      (1 - dx) * dy * gray_img(x1, y2) + ...
                      dx * dy * gray_img(x2, y2);

        % 将计算出的像素值赋值给新图像
        resized_img_bilinear(i, j) = pixel_value;
    end
end


% 显示双线性插值放大后的图像
imshow(uint8(resized_img_bilinear));
```

2、图像放大和旋转程序（使用内置函数）

```
% 读取图像
img = imread('C:\Users\28323\Desktop\第一次作业\实验图像.bmp');


% 将图像转换为灰度图
gray_img = rgb2gray(img);


% 顺时针旋转 30 度（使用'imrotate'函数）
```

```
rotated_img = imrotate(gray_img, -30, 'bilinear', 'loose');


% 显示旋转后的图像
figure;
imshow(rotated_img);


% 使用最近邻插值法放大图像 2 倍
scaled_img = imresize(gray_img, 2, 'nearest');


%使用双线性插值法放大图像 2 倍
scaled_img2 = imresize(gray_img, 2, 'bilinear');


% 显示放大后的图像
figure;
imshow(scaled_img);
imshow(scaled_img2);
```

3、图像傅里叶变换程序（不使用内置函数）

```
FourierTransformOptimized()
```

```
function FourierTransformOptimized()
    % 读取图像并转换为灰度图像
    img = imread('C:\Users\28323\Desktop\第一次作业\实验图像.bmp');
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    img = double(img);  % 将图像转换为 double 类型，便于后续计算
    [M, N] = size(img);


    % 显示原始灰度图像
    figure;
    imshow(uint8(img));
    title('Original Grayscale Image');
```

```
% 一维傅里叶变换的手动实现（向量化优化）
function F = fft1D(x)
    N = length(x);
    F = zeros(1, N);
    n = 0:N-1;  % 创建索引向量
    for k = 1:N
        % 使用向量化的傅里叶变换公式计算
        F(k) = sum(x .* exp(-1i * 2 * pi * (k-1) * n / N));
    end
end


% 对每一行进行一维傅里叶变换
F_row = zeros(M, N);
for i = 1:M
    F_row(i, :) = fft1D(img(i, :));
end


% 对每一列进行一维傅里叶变换
F = zeros(M, N);
for j = 1:N
    F(:, j) = fft1D(F_row(:, j)');
end


% 显示傅里叶变换图像
figure;
imshow(log(abs(F) + 1), []);  % 显示傅里叶变换的幅度谱
title('Fourier Transform Image (Original)');


% 将频率原点移至图像中心
function F_shifted = my_fftshift(F)
    [M, N] = size(F);


    % 计算中心点
    half_M = floor(M / 2);
    half_N = floor(N / 2);
```

```
    % 创建一个新的矩阵用于存储结果
    F_shifted = zeros(M, N-1);

    % 处理偶数和奇数情况
    if mod(M, 2) == 0
        row_idx1 = 1:half_M;
        row_idx2 = half_M+1:M;
    else
        row_idx1 = 1:half_M;
        row_idx2 = half_M+2:M;
    end

    if mod(N-1, 2) == 0
        col_idx1 = 1:half_N;
        col_idx2 = half_N+1:N-1;
    else
        col_idx1 = 1:half_N;
        col_idx2 = half_N+2:N-1;
    end

    % 左上角 (Q1) 和右下角 (Q4) 交换
    F_shifted(row_idx1, col_idx1) = F(row_idx2, col_idx2);
    F_shifted(row_idx2, col_idx2) = F(row_idx1, col_idx1);

    % 右上角 (Q2) 和左下角 (Q3) 交换
    F_shifted(row_idx1, col_idx2) = F(row_idx2, col_idx1);
    F_shifted(row_idx2, col_idx1) = F(row_idx1, col_idx2);

    % 如果 M 或 N 是奇数，处理中心行/列
    if mod(M, 2) == 1
        F_shifted(half_M+1, :) = F(half_M+1, :);
    end
    if mod(N-1, 2) == 1
        F_shifted(:, half_N+1) = F(:, half_N+1);
    end
end
```

```
    F_shifted = my_fftshift(F)

    % 计算傅里叶变换结果的幅值
    F_magnitude = abs(F_shifted);
    F_log = log(1 + F_magnitude);


    % 显示中心化后的傅里叶变换图像
    figure;
    imshow(mat2gray(F_log));  % 显示中心化后的幅度谱
    title('Fourier Transform Image (Shifted to Center)');
end
```

4、图像傅里叶变换程序（使用内置函数）

```
% 读取图像
img = imread('C:\Users\28323\Desktop\第一次作业\实验图像.bmp');


% 将图像转换为灰度图
gray_img = rgb2gray(img);


% 对图像进行傅里叶变换
F = fft2(double(gray_img));


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


% 计算傅里叶变换的幅值
F_magnitude = abs(F_shifted);


% 对数变换增强可视化效果
F_log = log(1 + F_magnitude);


% 显示原始灰度图像
figure;
imshow(uint8(gray_img));
```

```
 % 显示傅里叶变换图像
figure;
imshow(log(abs(F) + 1), []);  % 显示傅里叶变换的幅度谱

% 显示傅里叶变换后的结果
figure;
imshow(mat2gray(F_log));  % 归一化以便显示
```

电子签名：