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：　陈　博

学　　号：　U202214123

目录

一、 实验目的

1. 理解傅里叶变换以及傅里叶系数的物理含义
2. 实现对图像的旋转、插值放大
3. 实现对图像的傅里叶变换

二、傅里叶变换的理解

1.傅里叶变换的物理含义

傅里叶变换是一种将信号从时域转换到频域的数学工具。它的物理含义是频率的角度来观察信号，将复杂的时间信号分解为一组简单的正弦波，这些正弦波的频率、幅度和相位合成了原始信号。傅里叶变换实现了从一个时间维度观察信号到从多个频率维度观察信号的转变，进而帮助我们分析信号的频谱，理解不同频率成分的贡献。通过傅里叶变换，我们可以解决一些在时域无法解决的问题，比如强化高频成分，或者对某个特定的频率成分进行滤波。

2.傅里叶系数的物理含义

傅里叶变换可以看成原信号在不同频率正弦分量（基向量）上的投影，而傅里叶系数则表示信号中每个频率成分的强度和相位，可以看作描述了信号在各个频率分量上的“相似度”。例如，傅里叶系数较大的频率成分说明信号在该频率上的表现突出，因此这些系数能够帮助我们识别和重构信号的特征。通过分析这些系数，我们可以深入了解信号的结构与性质。

三、实现图像的旋转、插值放大

1.图像的旋转

实现图像旋转的基本原理就是矩阵相乘，具体的公式如下：

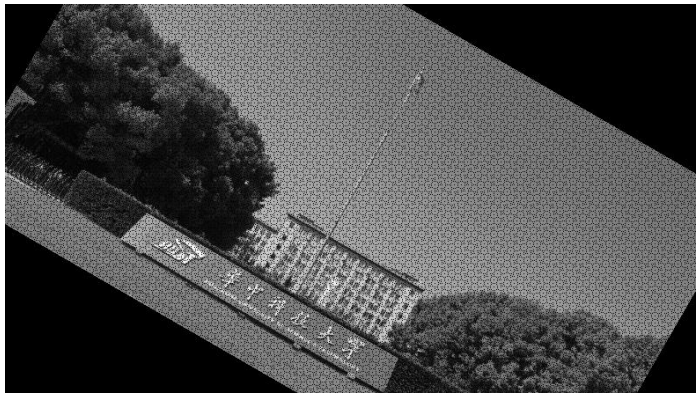
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} cos\,\theta & -sin\,\theta \\ sin\,\theta & cos\,\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

因此，通过编写程序，设定θ值，即可实现旋转，具体核心代码实现如下：

```
for i = 1:rows
```

```
    for j = 1:cols
        position = [i; j];
        previous_position = round(R*(position-center)+center);
        if (previous_position(1) >= 1 && previous_position(1) <= rows &&
previous_position(2) >= 1 && previous_position(2) <= cols)
            rotated_img(previous_position(1), previous_position(2)) =
img(i,j);
        end
    end
end
```

最后出来的结果如下图所示，



图像中出现很多噪声很多杂点，出现杂点的原因是从原图旋转后的像素位置在原图可能找不到。因此，我最终采用逆向思维，从目标图片反向旋转到原图进行像素查找。因为旋转矩阵的逆就是旋转矩阵的转置，得到以下公式：

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

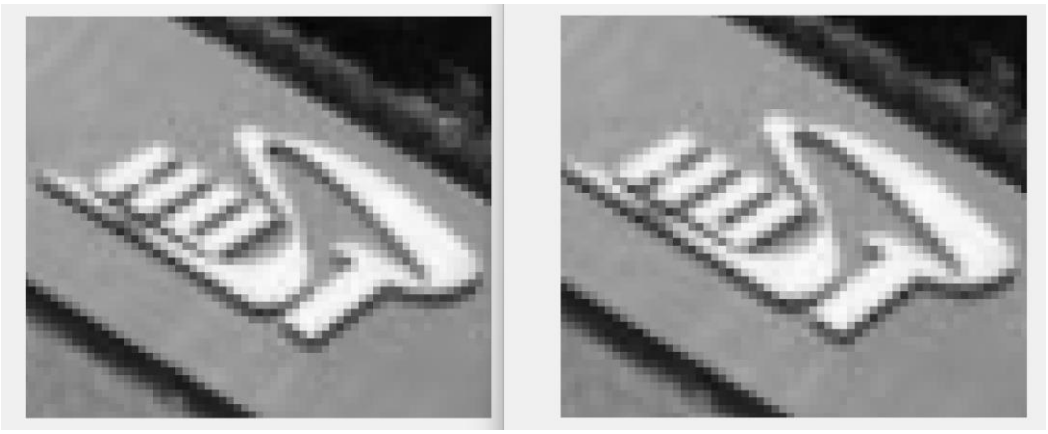
最后进行编程，得到最近邻插值和双线性插值的结果，并与 matlab 自带的函数做出来的结果进行比对（代码见附录）。

手写函数实现最近邻插值

Matlab 自带函数实现最近邻插值

手写函数实现双线性插值

Matlab 自带函数实现双线性插值



双线性插值时的对比图，Matlab 自带函数（左），手写函数（右）

最后实现的结果跟 Matlab 自带函数的效果存在差异，比如 HUST 中 S 上面那一弯，Matlab 拐弯处有一个黑点，而我自己写的却没有，但结果大体相同。

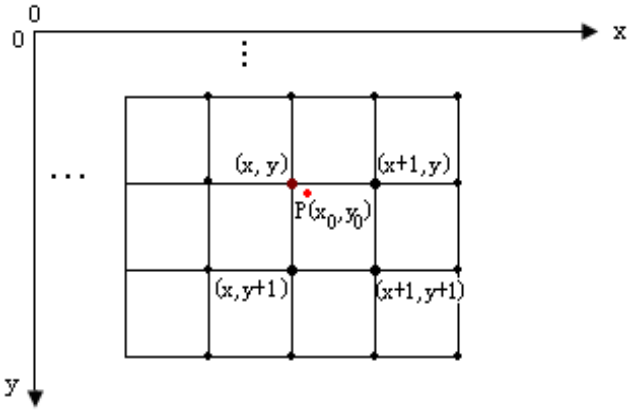
2.图像的插值放大

放大时主要是产生了放大前没有的像素点，需要插值处理，这里，基于最近邻插值和双线性插值将图片放大，并与 Matlab 自带的函数产生的效果进行对比。

A. 最近邻插值的过程：

1) 计算与点 P(x0，y0)临近的四个点;

2) 将与点 P(x0，y0)最近的整数坐标点(x，y)的灰度值取为 P(x0，y0) 点灰度近似值。



最近邻插值的原理图

核心代码如下：

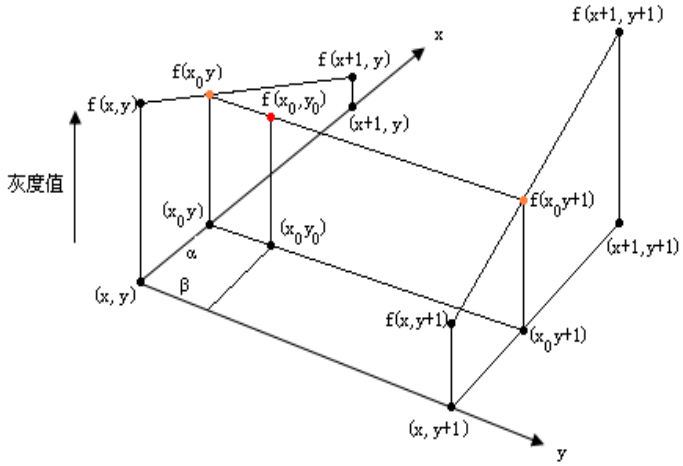
```
% 计算对应原图像的最近邻像素
orig_x = round(i / scale);
orig_y = round(j / scale);

% 防止越界
orig_x = max(1, min(orig_x, rows));
orig_y = max(1, min(orig_y, cols));
```

```
% 赋值给新图像
enlarged_img(i, j) = img(orig_x, orig_y);
```

B. 双线性插值的过程

根据点 P(x0，y0)的四个相邻点的灰度值，通过两次插值计算出灰度值f(x0，y0)。



双线性插值原理示意图

$$f(x_0,y_0)=f(x,y)+\alpha[f(x+1,y)-f(x,y)]+\beta[f(x,y+1)-f(x,y)]+\alpha\beta[f(x+1,y+1)+f(x,y)-f(x,y+1)-f(x+1,y)]$$

核心代码如下：

```
% 计算原图中的坐标
x = i / scale;
y = j / scale;

% 获取最近的四个像素的坐标
x1 = floor(x);
x2 = ceil(x);
y1 = floor(y);
```

```
y2 = ceil(y);

% 防止越界
x1 = max(1, min(x1, rows));
x2 = max(1, min(x2, rows));
y1 = max(1, min(y1, cols));
y2 = max(1, min(y2, cols));

% 计算四个像素值
Q11 = double(img(x1, y1));
Q12 = double(img(x1, y2));
Q21 = double(img(x2, y1));
Q22 = double(img(x2, y2));

% 计算插值权重
weight_x2 = x - x1;     %a
weight_x1 = 1 - weight_x2;%1-a
weight_y2 = y - y1;     %b
weight_y1 = 1 - weight_y2;%1-b
```

```
% 进行双线性插值计算
enlarged_img(i, j) = uint8(weight_x1 * (weight_y1 * Q11
+ weight_y2 * Q12) + ...
                          weight_x2 * (weight_y1 * Q21 +
weight_y2 * Q22));
```

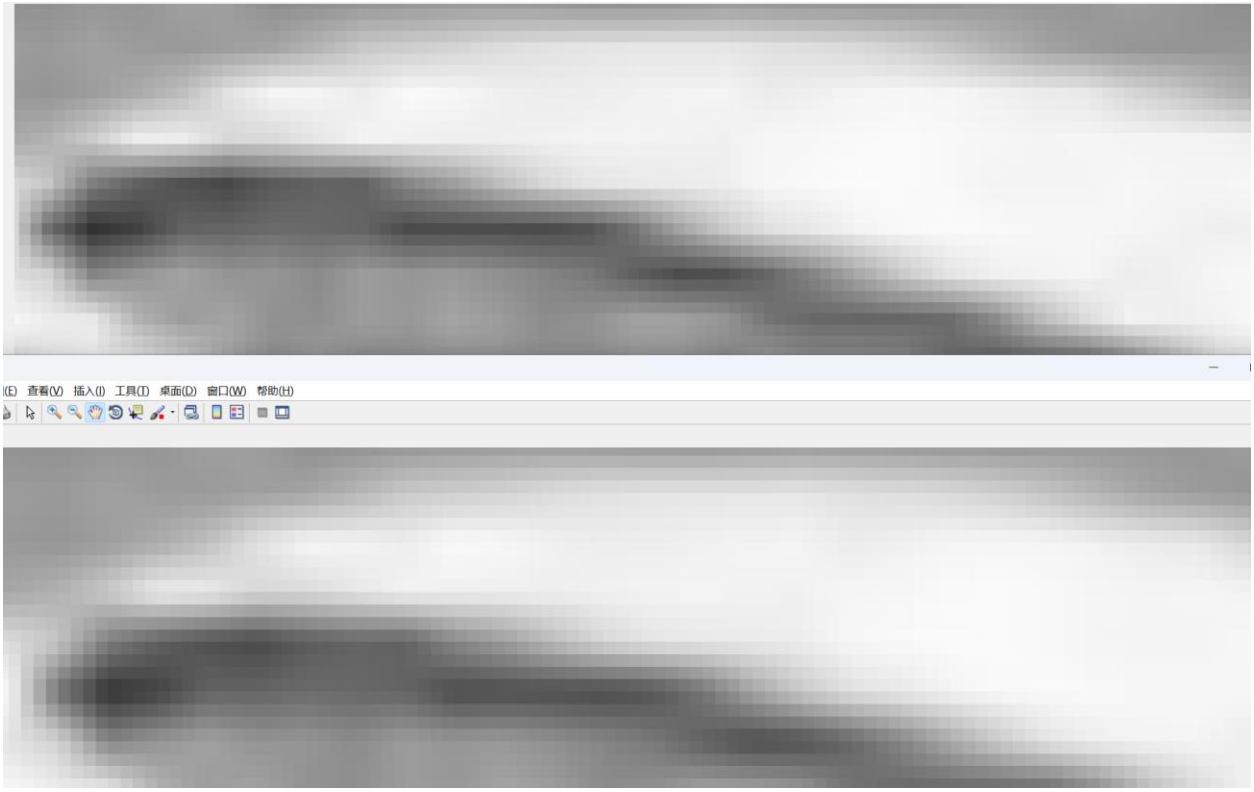
最后的实验结果如下：

手写函数放大结果

Matlab 自带函数函数放大结果

由于这里放到 word 中还存在放大，因此着重比对一下细节：

最近邻插值放大 2 倍时的细节对比，上图是手搓，下图是 Matlab 自带函数



双线性插值放大 4 倍时的细节对比，上图是手搓，下图是 Matlab 自带函数

可见，实现效果跟 Matlab 自带函数相比还是差不多的，同时，也明显展示出来双线性插值带来的更强大的平滑。

四、实现图像的傅里叶变换

1.实验原理

我们对图像进行二维傅里叶变换，从而观察图像中各个频率分量的分布情况，具体公式如下：

$$\boldsymbol{F(u,v) = \frac{1}{NN}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) exp\left(-j2\pi\left(\frac{ux}{N}+\frac{vy}{N}\right)\right)}$$

直接进行编程需要用到四层 for 循环，计算成本过大，因此，我们使用傅里叶变换的可分离性，将一次二维傅里叶变换变成两次一维傅里叶变换，减小计算量：

编程时采用模块化思维，将一维傅里叶变换封装成函数：

$$F(x,v) = \frac{1}{N}\sum_{y=0}^{N-1} f(x,y)\, exp[-\mathrm{j}2\pi vy/N]$$

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1} F(x,v)\, exp[-j2\pi ux/N]$$

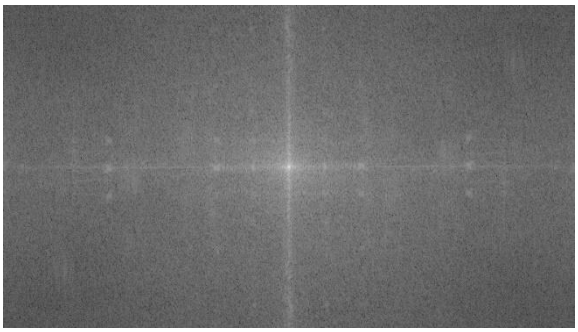
```
function F = my_fft1( x )
    N = length(x);
    F = zeros(1,N);
    n = 1:N;
    for k = 1:N
        F(k) = sum(x .* exp(-1i * 2 * pi * (k-1) * n/N));
    end
end
```

这样极大减小编程工作量。

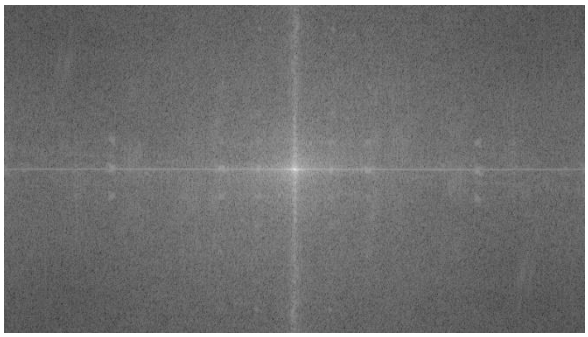
对于将频率原点移至图像中心，我们使用如下公式：

$$f(x,y)(-1)^{x+y} \Leftrightarrow F\left(u-\frac{N}{2}, v-\frac{N}{2}\right)$$

（代码见附录）最后的实验结果如下：



Matlab 自带函数实现的傅里叶变换



手写函数实现的傅里叶变换

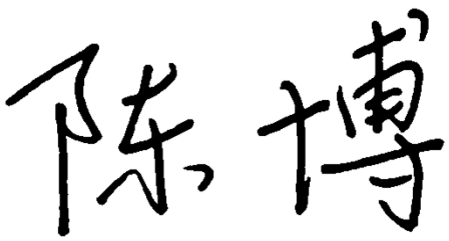
可见，原图中既含有低频成分，又含有高频成分。

一方面，大部分区域是表示低频成分的平滑区域，其中心较亮表明图中低频成分占主导地位。

另一方面，高频成分所占据的比例也含有不少，它们代表了细节和边缘信息，这主要包含了图中南一楼的窗户，校门栅栏等地方。

同时，我也比较了 MATLAB 自带库中傅里叶变换的结果与我们手动编程的结果，最后得到的矩阵大体相似，误差控制在 0.0003 以内。

陈博_人工智能 2204 班_U202214123，电子签名：



五、附录

1.旋转图像代码

```
i=imread('D:\Matlab_project\image_rotating\实验图像.bmp'); %读取图像
figure,imshow(i); %显示图像
```

```
imwrite(i,'实验图像.bmp'); %存储图像

j=rgb2gray(i); %rgb 图像转为灰度图像
figure,imshow(j); %显示灰度图像
imwrite(j,'grey.bmp'); %存储灰度图像

% matlab 自带旋转
j = imread('grey.bmp');
rotated_img_builtin = imrotate(j, -30, 'bilinear');
figure, imshow(rotated_img_builtin);
imwrite(rotated_img_builtin,
'rotated_builtin_bilinear.bmp');


% 调用自定义函数进行旋转并显示图像
j = imread('grey.bmp');  % 读取灰度图像
rotated_img = rotate_image(j, -30);  % 旋转 30 度
figure, imshow(rotated_img);  % 显示旋转后的图像
imwrite(rotated_img,
'rotated_custom_initial.bmp');  % 存储旋转后的图像
```

函数：

```
function rotated_img = rotate_image_bilinear(img,
angle)
    % 将角度转换为弧度
    angle_rad = deg2rad(angle);
    R = [cos(angle_rad), -sin(angle_rad);
sin(angle_rad), cos(angle_rad)];
    R = R';

    % 获取原图像尺寸
    [rows, cols] = size(img);

    % 计算新图像的尺寸
    new_rows = ceil(abs(rows * cos(angle_rad)) +
abs(cols * sin(angle_rad)))+1;
    new_cols = ceil(abs(rows * sin(angle_rad)) +
abs(cols * cos(angle_rad)))+1;

    % 创建空白的新图像
    rotated_img = zeros(new_rows, new_cols,
'uint8');
```

```
    % 找到图像中心点
    center_row = floor(rows / 2);
    center_col = floor(cols / 2);
    center = [center_row; center_col];
    new_center_row = floor(new_rows / 2);
    new_center_col = floor(new_cols / 2);
    new_center = [new_center_row; new_center_col];

%     for i = 1:rows
%         for j = 1:cols
%             position = [i; j];
%             previous_position = round(R*(position-
center)+center);
%             if (previous_position(1) >= 1 &&
previous_position(1) <= rows &&
previous_position(2) >= 1 && previous_position(2)
<= cols)
%                 rotated_img(previous_position(1),
previous_position(2)) = img(i,j);
```

```
%             end
%         end
%     end

%     % 对新图像进行最近邻插值
%     for i = 1:new_rows
%         for j = 1:new_cols
%             position = [i; j];
%             previous_position = round(R*(position-
new_center)+center);
%             if (previous_position(1) >= 1 &&
previous_position(1) <= rows &&
previous_position(2) >= 1 && previous_position(2)
<= cols)
%                 rotated_img(i, j) =
img(previous_position(1), previous_position(2));
%             end
%         end
%     end
```

```
    % 对新图像进行双线性插值
    for i = 1:new_rows
        for j = 1:new_cols
            position = [i; j];
            previous_position = (R*(position-
new_center)+center);
            f_previous_position =
floor(previous_position);
            ab = previous_position -
f_previous_position;
            a = ab(1);
            b = ab(2);
            m = f_previous_position(1);
            n = f_previous_position(2);

            if (previous_position(1) >= 2 &&
previous_position(1) <= rows-1 &&
previous_position(2) >= 2 && previous_position(2)
<= cols-1)
```

```
                rotated_img(i, j) = (1-a)*(1-
b)*img(m,n) + a*(1-b)*img(m+1,n)...
                                        + (1-
a)*b*img(m,n+1) + a*b*img(m+1,n+1);
            end
        end
    end
end
```

2.放大图像代码

```
i=imread('D:\Matlab_project\image_magnify\grey.bmp'
); %读取图像
figure,imshow(i); %显示图像


% % MATLAB 自带函数
% enlarged_img_nearest = imresize(i, 4, 'nearest');
%
% % 显示并保存放大后的图像
% figure, imshow(enlarged_img_nearest);
```

```
% imwrite(enlarged_img_nearest,
'enlarged_nearest_builtin_4.bmp');


enlarged_img_bilinear = imresize(i, 4, 'bilinear');

% 显示并保存放大后的图像
figure, imshow(enlarged_img_bilinear);
imwrite(enlarged_img_bilinear,
'enlarged_bilinear_builtin.bmp');


% % 手搓函数
% enlarged_img_nearest =
nearest_neighbor_enlarge(i, 4);  % 放大 2 倍
% figure, imshow(enlarged_img_nearest);  % 显示放大
后的图像
% imwrite(enlarged_img_nearest,
'enlarged_nearest_4.bmp');  % 保存放大后的图像
```

```
enlarged_img_bilinear = bilinear_enlarge(i, 4);  % 放大 4 倍
figure, imshow(enlarged_img_bilinear);  % 显示放大后的图像
imwrite(enlarged_img_bilinear, 'enlarged_bilinear.bmp');  % 保存放大后的图像
```

函数：

```
function enlarged_img = nearest_neighbor_enlarge(img, scale)
    [rows, cols] = size(img);

    % 计算放大后图像的尺寸
    new_rows = rows * scale;
    new_cols = cols * scale;

    % 创建空白的放大图像
    enlarged_img = zeros(new_rows, new_cols, 'uint8');

    % 最近邻插值
```

```
    for i = 1:new_rows
        for j = 1:new_cols
            % 计算对应原图像的最近邻像素
            orig_x = round(i / scale);
            orig_y = round(j / scale);

            % 防止越界
            orig_x = max(1, min(orig_x, rows));
            orig_y = max(1, min(orig_y, cols));

            % 赋值给新图像
            enlarged_img(i, j) = img(orig_x,
orig_y);
        end
    end
end


function enlarged_img = bilinear_enlarge(img,
scale)
    % 获取原图像尺寸
```

```
    [rows, cols] = size(img);

    % 计算放大后图像的尺寸
    new_rows = rows * scale;
    new_cols = cols * scale;

    % 创建空白的放大图像
    enlarged_img = zeros(new_rows, new_cols,
'uint8');

    % 双线性插值
    for i = 1:new_rows
        for j = 1:new_cols
            % 计算原图中的坐标
            x = i / scale;
            y = j / scale;

            % 获取最近的四个像素的坐标
            x1 = floor(x);
            x2 = ceil(x);
```

```
        y1 = floor(y);
        y2 = ceil(y);

        % 防止越界
        x1 = max(1, min(x1, rows));
        x2 = max(1, min(x2, rows));
        y1 = max(1, min(y1, cols));
        y2 = max(1, min(y2, cols));

        % 计算四个像素值
        Q11 = double(img(x1, y1));
        Q12 = double(img(x1, y2));
        Q21 = double(img(x2, y1));
        Q22 = double(img(x2, y2));

        % 计算插值权重
        weight_x2 = x - x1;     %a
        weight_x1 = 1 - weight_x2;%1-a
        weight_y2 = y - y1;     %b
        weight_y1 = 1 - weight_y2;%1-b
```

```
            % 进行双线性插值计算
            enlarged_img(i, j) = uint8(weight_x1 *
(weight_y1 * Q11 + weight_y2 * Q12) + ...
                                       weight_x2 *
(weight_y1 * Q21 + weight_y2 * Q22));
        end
    end
end
```

3.**傅里叶变换代码**

```
i=imread('D:\Matlab_project\image_fourier\grey.bmp'
); %读取图像
my_fft2(i)

% 对图像进行二维傅里叶变换
F = fft2(double(i));

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

```
% 取傅里叶变换的幅度并取对数，以便更好地显示
F_magnitude = abs(F_shifted);
F_magnitude_log = log(1 + F_magnitude);  % 取对数避免动态范围过大

% 显示傅里叶变换后的图像
figure, imshow(F_magnitude_log, []);  % 使用空数组[] 自动缩放显示

% 保存傅里叶变换后的图像
imwrite(mat2gray(F_magnitude_log), 'fft_image_matlab.bmp');
```

函数：

```
function F = my_fft1( x )
    N = length(x);
    F = zeros(1,N);
    n = 1:N;
    for k = 1:N
        F(k) = sum(x .* exp(-1i * 2 * pi * (k-1) * n/N));
```

```
    end
end
```

```
% function F_shifted = my_fft2(img)
%
%     %img = double(img);
%     % 获取图像大小
%     [rows, cols] = size(img);
%
%     % 构建平移矩阵，将频率原点移至中心
%     % 利用傅里叶变换的平移性，构建 (-1)^(x+y) 的矩阵
%     for x = 1:rows
%         for y = 1:cols
%             img(x, y) = img(x, y) * (-1)^(x + y);
%         end
%     end
%
%     tmp = 0+1i;
%
%     % 利用傅里叶变换的可分离性，分别对行和列做 1 维傅里叶变换
```

```
%     % 对每一行做 1 维傅里叶变换
%     F_row = zeros(rows, cols);
%     for x = 1:rows
%         F_row(x, :) = fft(img(x, :));
% %         for v = 1:cols
% %             sum = 0;
% %             for y = 1:cols
% %                 sum = sum + img(x,y)*exp(-
tmp*2*pi*v*y/cols);
% %             end
% %             sum = sum / cols;
% %             F_row(x,v) = sum;
% %         end
%     end
%
%     % 对每一列做 1 维傅里叶变换
%     F_shifted = zeros(rows, cols);
%     for y = 1:cols
%         F_shifted(:, y) = fft(F_row(:, y));
%     end
```

```
%
%     % 使用 fftshift 将结果移到中心
%     F_shifted = fftshift(F_shifted);
%
%     % 输出傅里叶变换后的结果
%     figure, imshow(log(1 + abs(F_shifted)),
[]); % 取对数并显示
% end
%

function F_shifted = my_fft2(img)

    figure;
    imshow(uint8(img));

    % 获取图像大小
    [rows, cols] = size(img);

     % 将图像转换为双精度类型
    img = double(img);
```

```
    % 构建平移矩阵，将频率原点移至中心
    % 利用傅里叶变换的平移性，构建 (-1)^(x+y) 的矩阵
    for x = 1:rows
        for y = 1:cols
            img(x, y) = img(x, y) * (-1)^(x + y);
        end
    end



    % 利用傅里叶变换的可分离性，分别对行和列做 1 维傅里叶变换
    % 对每一行做 1 维傅里叶变换
    F_row = zeros(rows, cols);
    for x = 1:rows
        F_row(x,:) = my_fft1(img(x, :));
    end

    % 对每一列做 1 维傅里叶变换
```

```
    F_shifted = zeros(rows, cols);
    for y = 1:cols
        F_shifted(:,y) = my_fft1(F_row(:,y)');
    end

    F_frequence = log(1 + abs(F_shifted));

    figure;
    imshow(F_frequence, []);
    imwrite(mat2gray(F_frequence),
'fft_image_byhand.bmp');
end
```

陈博_人工智能 2204 班_U202214123，电子签名：