华中科技大学

数字图像处理实验报告

院系　　人工智能与自动化学院

专业班级　　人工智能 2204 班

学号　　U202214123

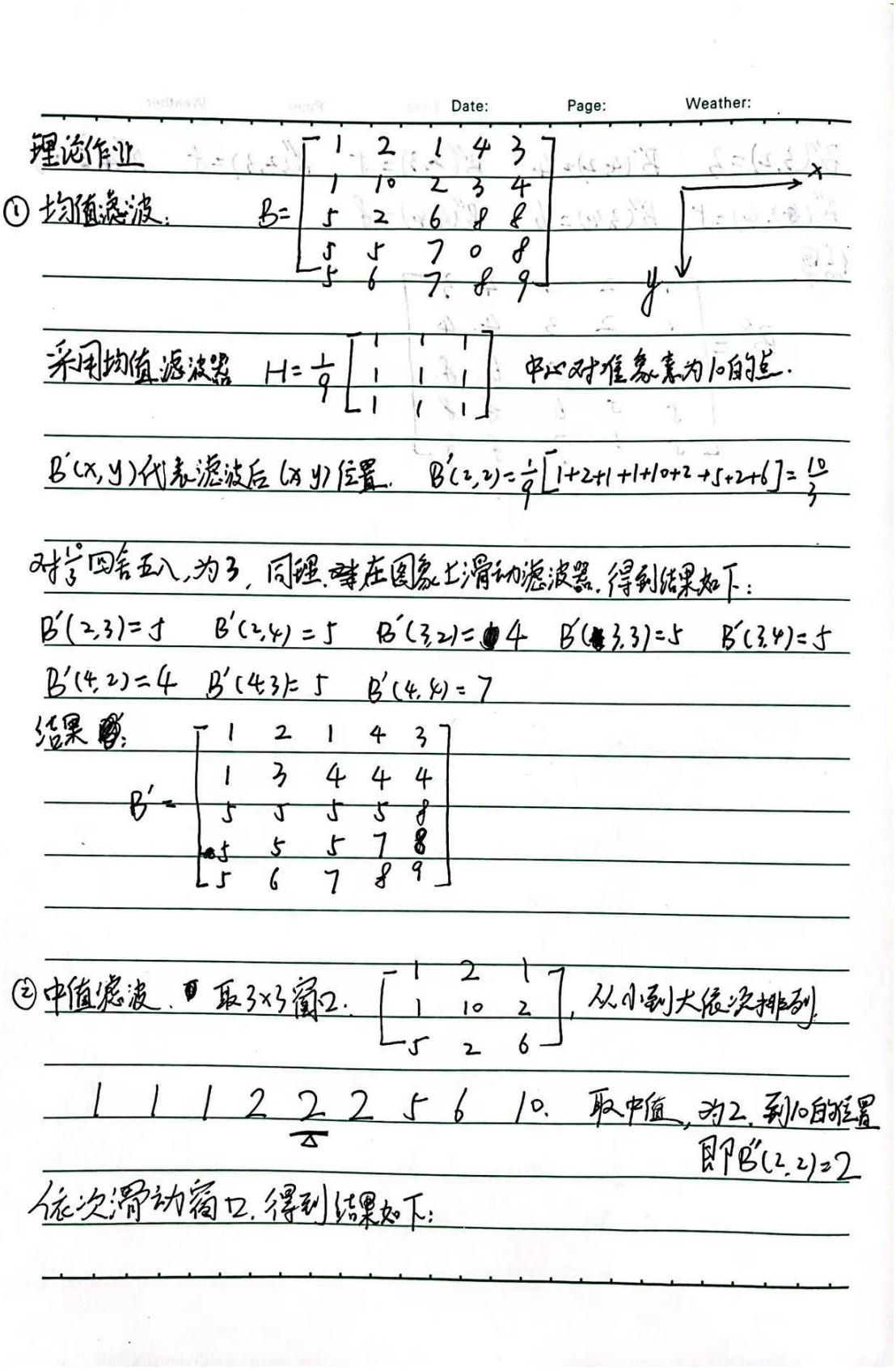
姓名　　陈博

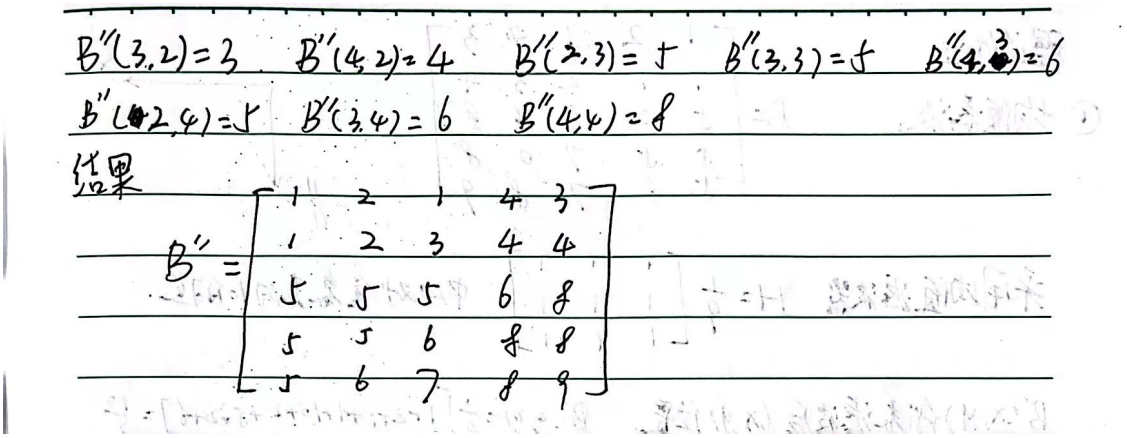
指导老师　　肖阳

提交时间　　2024 年 10 月 20 日

目录

1 理论作业


Date:
Page:
Weather:
理论作业
① 均值滤波:
$B=\begin{bmatrix}1&2&1&4&3\\1&10&2&3&4\\5&2&6&8&8\\5&5&7&0&8\\5&6&7&8&9\end{bmatrix}$
x
y
采用均值滤波器 $H=\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ 中心对准象素为10的点.
$B'(x,y)$代表滤波后(x,y)位置. $B'(2,2)=\frac{1}{9}[1+2+1+1+10+2+5+2+6]=\frac{10}{3}$
对$\frac{10}{3}$四舍五入,为3,同理,对在图象上滑动滤波器,得到结果如下:
$B'(2,3)=5$ $B'(2,4)=5$ $B'(3,2)=4$ $B'(3,3)=5$ $B'(3,4)=5$
$B'(4,2)=4$ $B'(4,3)=5$ $B'(4,4)=7$
结果:
$B'=\begin{bmatrix}1&2&1&4&3\\1&3&4&4&4\\5&5&5&5&8\\5&5&5&7&8\\5&6&7&8&9\end{bmatrix}$
② 中值滤波. 取3×3窗口. $\begin{bmatrix}1&2&1\\1&10&2\\5&2&6\end{bmatrix}$, 从小到大依次排列
1 1 1 2 2 2 5 6 10. 取中值,为2. 到10的位置
即$B''(2,2)=2$
依次滑动窗口,得到结果如下:



2 实验一：灰度图与离散傅里叶变换

2.1 实验原理

灰度图的统计，即为遍历图像中所有像素，统计各个灰度值对应像素点的个数

离散傅里叶变换（Discrete Fourier Transform, DFT）是一种用于将信号从时域转换到频域的方法。对于二维图像，DFT 可以将图像从空间域转换到频率域。离散傅里叶变换的数学表达式为：

$$F(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/N+vy/N)},$$

其中，$f(x,y)$ 是图像在空间域的像素值，$F(u,v)$ 是图像在频率域的表示，M 和 N 分别是图像的宽度和高度。

逆离散傅里叶变换（Inverse Discrete Fourier Transform, IDFT）用于将图像从频率域转换回空间域，其数学表达式为：

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

通过傅里叶变换，我们可以分析图像的频率成分，从而进行各种图像处理操作，如滤波、去噪等。

对于将频率原点移至图像中心，我们使用如下公式：

$$f(x,y)(-1)^{x+y} \leftrightarrow F(u-N/2, v-N/2)$$

2.2 核心代码编写

以下是将彩色图像转换为灰度图并统计灰度直方图的核心 MATLAB 代码（完整代码在附录）：

```
% 将彩色图像转换为灰度图像
for i = 1:height
    for j = 1:width
        % 计算每个像素点的灰度值，使用常用的RGB转灰度公式
        gray_img(i, j) = 0.299 * double(img(i, j, 1))
        + 0.587 * double(img(i, j, 2))
        + 0.114 * double(img(i, j, 3));
```

```
    end
end

gray_img = uint8(gray_img);

% 统计灰度直方图
for i = 1:height
    for j = 1:width
        % 获取当前像素的灰度值
        gray_value = gray_img(i, j);
        % 增加对应灰度值的计数
        histogram(gray_value + 1) = histogram(gray_value + 1) + 1;
    end
end
```

以下是进行图像离散傅里叶变换的核心 MATLAB 代码（完整代码在附录）：

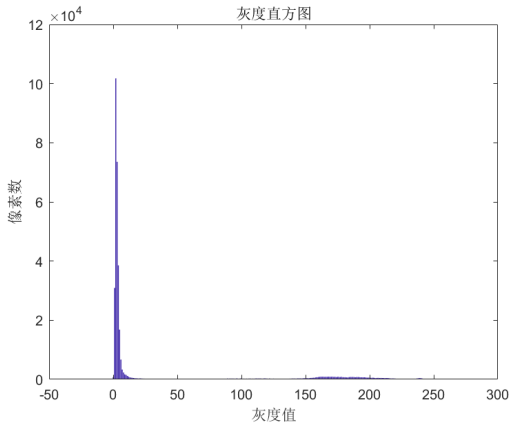
```
function F = my_fft1( x )
    % my_fft1 手动实现离散傅里叶变换 (DFT)
    % 输入:
    %   x - 输入信号向量
    % 输出:
    %   F - 计算得到的傅里叶变换结果向量

    % 开始计算DFT
    for k = 1:N % 对于输出向量F中的每一个元素
        % 计算第k个频率分量的值
        % 利用复指数 e^(-j*2*pi*(k-1)*n/N) 与输入信号 x 相乘，然后求和
        F(k) = sum(x .* exp(-1i * 2 * pi * (k-1) * n/N));
    end
end

function F_shifted = my_fft2(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));

end
```

2.3 实验结果

通过观察灰度图1a，我们发现这张照片整体水平偏暗，同时，在灰度直方图1b中，灰度级集中分布在25 以下的部分，在离散傅里叶变换频谱图2a中，图像大部分频率分布在低频附近，高频成分较少，这可能是由于照片整体上灰度较暗，我们接下来使用直方图均衡化与同态滤波两种方法处理这张图片，期望使照片整体灰度分布均匀一些。

(a) 灰度图



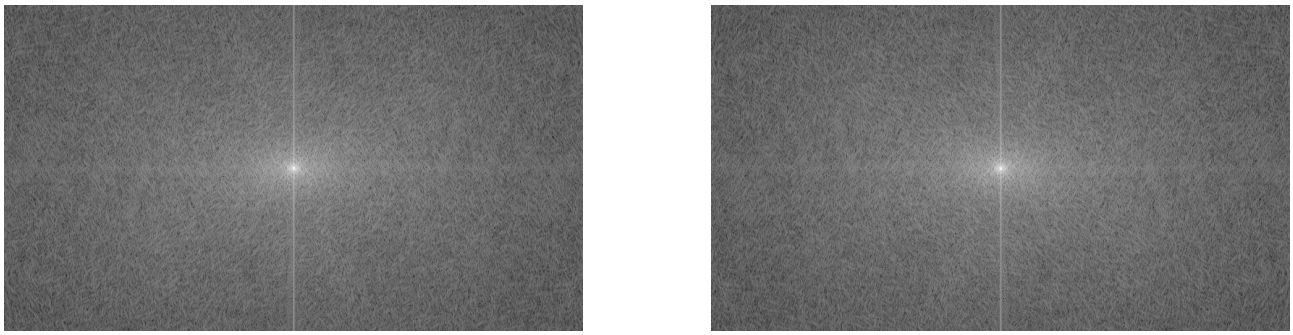
(b) 灰度直方图

图 1: 灰度图及其直方图

3 实验二：直方图均衡化

3.1 实验原理

直方图均衡化是一种增强图像对比度的技术，它通过对图像的灰度直方图进行非线性变换来改善图像的整体视觉效果。直方图均衡化的基本思想是将图像的灰度级重新映射到新的范围，使得每个灰度级出现的概率更加均匀，从而提高图像的对比度。



(a) 手搓函数实现　　(b) Matlab 自带函数实现

图 2: 离散傅里叶变换结果图

应用到离散灰度级，设一幅图像的像素总数为 n，分别为 L 个灰度级，n_k 表示第 k 个灰度级出现的像素数，则变换函数的离散形式可表示为：

$$t_k = T(s_k) = \sum_{i=0}^{k} p_s(s_i) = \sum_{i=0}^{k} \frac{n_i}{n}$$

其中 $0 \leq s_k \leq 1$，$k = 0, 1, \cdots, L-1$。

直方图均衡化的过程通常包含以下几个步骤：

1. 计算图像的累积分布函数 (CDF)，即灰度直方图的累计概率分布：

$$CDF(k) = \sum_{i=0}^{k} \frac{n_i}{n}$$

2. 对于每一个灰度级 s_k，计算对应的输出灰度级 t_k：

$$t_k = CDF(s_k) \cdot (2^d - 1)$$

其中 d 是输出图像的位深度，通常是 8 位，即 $2^d = 256$。

3. 进行灰度映射，将输入图像中的每个像素值 s_k 替换成对应的输出灰度级 t_k。

这个过程可以通过以下公式实现：

$$t = T(s) = \text{round}(CDF(s) \cdot (2^d - 1))$$

其中 round(·) 函数用于将浮点数四舍五入到最接近的整数。

4. 最后，将映射后的图像作为输出图像。

直方图均衡化能够有效地提升图像的对比度，特别是在图像的灰度级分布不均匀的情况下。这种方法适用于大多数自然图像，因为它们的灰度直方图通常具有长尾分布的特点。

3.2 核心代码编写

以下是进行图像均衡化的核心 MATLAB 代码（完整代码在附录）：

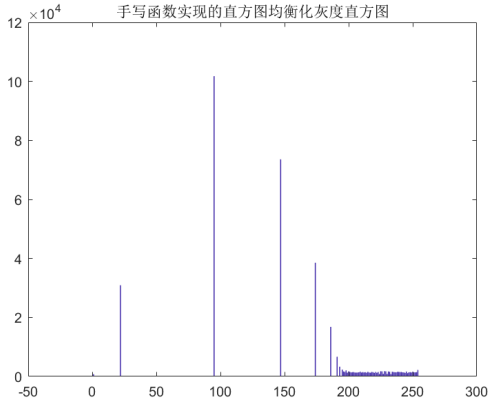
```
function [eq_img, eq_hist] = hist_eq_manual(img)
    % hist_eq_manual 手写实现直方图均衡化
    % 输入：
    %    img - 输入的彩色图像
```

```
    % 输出:
    %   eq_img - 直方图均衡化后的图像
    %   eq_hist - 新图像的灰度直方图

    % 统计灰度直方图
    for i = 1:height
        for j = 1:width
            % 获取当前像素的灰度值
            gray_value = gray_img(i, j);
            % 增加对应灰度值的计数
            hist(gray_value + 1) = hist(gray_value + 1) + 1;
        end
    end

    % 计算累积分布函数 (CDF)
    cdf = cumsum(hist); % 计算累积和
    cdf = cdf / max(cdf); % 归一化累积分布函数

    % 进行灰度映射
    eq_img = uint8(255 * cdf(gray_img + 1));
    % 将输入图像中的每个像素值替换成对应的输出灰度级

    % 计算均衡化后的图像的灰度直方图
    eq_hist = zeros(1, 256);
    % 创建一个长度为256的零向量, 用于存储均衡化后的图像的灰度直方图
    for i = 1:height
        for j = 1:width
            % 获取当前像素的灰度值
            eq_gray_value = eq_img(i, j);
            % 增加对应灰度值的计数
            eq_hist(eq_gray_value + 1) = eq_hist(eq_gray_value + 1) + 1;
        end
    end
end
```

3.3 实验结果

最后结果表明，使用内置函数得到的图像4a要比自己写的函数得到的图像3a要偏暗一些，对比灰度直方图，可以发现使用内置函数时4b，灰度级数量较少，整体偏亮，可能是由于内置函数进行了对直方图均衡化进行了优化。

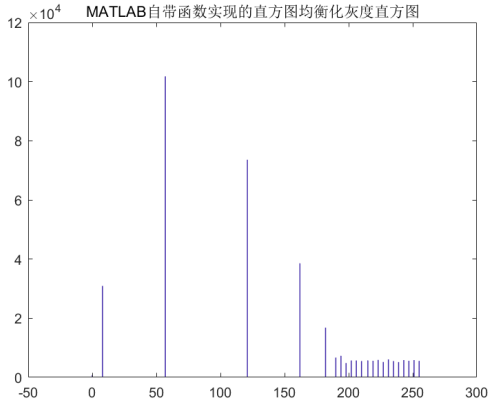
(a) 均衡化结果



(b) 灰度直方图

图 3: 手写函数实现直方图均衡化以及结果图像的灰度直方图

(a) 均衡化结果



(b) 灰度直方图

图 4: Matlab 自带函数实现直方图均衡化以及结果图像的灰度直方图

4 实验三：同态滤波

4.1 实验原理

同态滤波是一种基于频域的图像处理技术，它可以对图像进行亮度和对比度的分离与调整。通过傅里叶变换和逆傅里叶变换，同态滤波器可以在保持图像细节的同时改变其亮度和对比度。同态滤波的基本流程如下：

1. 首先，对图像进行对数变换，将图像的动态范围压缩到有限范围内。
2. 接着，对经过对数变换的图像进行傅里叶变换，将其转换到频域。
3. 在频域上应用特定的滤波器 $H(u,v)$，例如高通滤波器，以增强图像的高频成分。
4. 最后，通过逆傅里叶变换将处理过的频域信号转换回时域，再通过指数运算恢复图像的亮度和对比度。

4.1.1 图像模型

图像 $f(x,y)$ 可以分解成两个部分：反射率 $r(x,y)$ 和光照 $i(x,y)$ 的乘积：

$$f(x,y) = r(x,y)i(x,y)$$

4.1.2 对数变换

为了简化后续操作，我们首先对图像进行对数变换：

$$\ln f(x,y) = \ln i(x,y) + \ln r(x,y)$$

4.1.3 傅里叶变换

接下来，对对数变换后的图像进行傅里叶变换：

$$F(u,v) = I(u,v) + R(u,v)$$

其中，$F(u,v)$ 是对数变换后的图像在频域上的表示，$I(u,v)$ 和 $R(u,v)$ 分别代表反射率和光照在频域上的表示。

4.1.4 滤波器

同态滤波的滤波器函数有如下形式：

$$H(u,v) = (\gamma_H - \gamma_L)[1 - e^{-(D(u,v)^2/D_0^2)}] + \gamma_L \tag{2}$$

其中，γ_H, γ_L 分别问高频增益和低频增益，需要自己调整，常数 c 控制函数坡度的锐利程度。上式中，D_0 是一个正常数，$D(u,v)$ 则是频率域中点 (u,v) 与频率矩形中心的距离，即

$$D(u,v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2} \tag{3}$$

在频域上应用滤波器 $H(u,v)$ 处理 $F(u,v)$:

$$H(u,v)F(u,v) = H(u,v)I(u,v) + H(u,v)R(u,v)$$

4.1.5 逆傅里叶变换和指数运算

最后，通过逆傅里叶变换将处理过的频域信号转换回时域，并通过指数运算恢复图像的亮度和对比度：

$$g(x,y) = \exp(H(u,v)F(u,v))$$

4.2 核心代码编写

以下是进行同态滤波的核心 MATLAB 代码（完整代码在附录）：

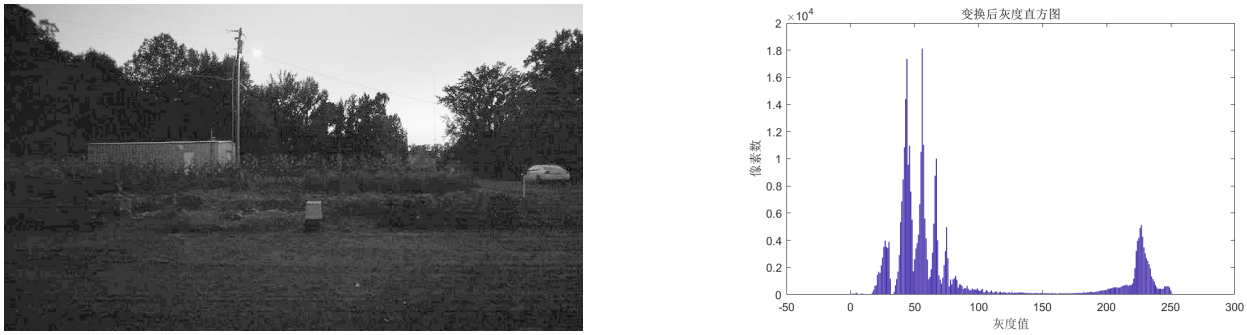
```
function [img_after, hist_output] = homo_f(image)
% 参数声明
rH = 0.05; % 高频增益
```

```
rL = 0.03; % 低频增益
c = 0.04;  % 控制函数坡度的锐利程度，介于 rH 和 rL 之间
D0 = 0.2;  % 截止频率

% 平移图像
% 通过交替符号来实现图像的平移，这有助于后续的傅里叶变换
img_py = zeros(M, N);
for i = 1:M
    for j = 1:N
        if mod(i + j, 2) == 0
            img_py(i, j) = img_log(i, j);
        else
            img_py(i, j) = -1 * img_log(i, j);
        end
    end
end

% 对平移后的图像进行傅里叶变换
% 傅里叶变换将图像从空间域转换到频率域
img_py_fft = fft2(img_py);

% 定义同态滤波函数（高斯滤波）
% 构建一个高斯滤波器，用于增强高频部分并抑制低频部分
H = zeros(M, N);
r = rH - rL;
D = D0^2; % 设置参数
m_mid = floor(M / 2); % 中心点坐标
n_mid = floor(N / 2);

for i = 1:M
    for j = 1:N
        dis = ((i - m_mid)^2 + (j - n_mid)^2); % 计算每个点到中心点的距离
        H(i, j) = r * (1 - exp((-c) * (dis / D))) + rL; % 高斯同态滤波函数
    end
end

% 应用滤波器
% 将滤波器应用于频率域中的图像
imgtemp = img_py_fft .* H;

% 开始反变换
% 将处理后的频率域图像转换回空间域
imgtemp = abs(real(ifft2(imgtemp)));
```

```
    % 反对数变换
    % 将图像从对数域转换回原始域
    imgtemp = exp(imgtemp) - 1;

    % 归一化处理
    % 将图像数据归一化到 0-255 的范围内
    max_num = max(imgtemp(:));
    min_num = min(imgtemp(:));
    range = max_num - min_num;
    img_after = uint8(255 * (imgtemp - min_num) / range);
end
```

4.3 实验结果

实验结果如图5所示，同态滤波很好的将原始图像的灰度直方图1b进行展开，有了更多的灰度级，图像也更加清晰，原本较暗的地方也显现了出来。



(a) 同态滤波结果　　(b) 灰度直方图

图 5: 同态滤波结果图像以及结果图像的灰度直方图

5 两种处理图像方法的比较

5.1 直方图均衡化：

图像4a的整体亮度有所提高，暗部细节得到了更好的展现，对比度有一定的提高，但是整体色彩偏冷色调。

通过观察灰度直方图4b，可以看出均衡化后图像的灰度分布变得更加均匀，峰值出现在中间区域，表明图像的动态范围得到了扩展。

5.2 同态滤波

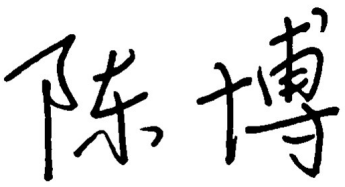
相比于直方图均衡化，同态滤波5a保留了更多的自然色彩，同时显著提高了图像的对比度，使得图像看起来更加清晰。

通过观察灰度直方图，可以看出滤波后图像的灰度分布5b的对比度得到了明显的增强，有一定的聚类效果，高亮度和低亮度的区域被分开，在没有合并灰度级的条件下保留了更多的自然色彩。

5.3 总结

总的来说，直方图均衡化和同态滤波都能改善图像的视觉效果，但它们的方法不同。直方图均衡化主要是通过对图像的全局统计特性进行调整，使得图像的灰度分布更加均匀，但灰度级的减少，使图像减少了一部分自然色彩，有些连接的地方没有那么自然；而同态滤波则利用了图像的频率信息，通过滤除噪声和增强高频成分来提升图像质量，但我们在滤波时保留了一部分低频分量，所以图片整体看起来还是比较平滑的，视觉效果比直方图均衡化更好。

6 电子签名



陈博-人工智能 2204 班-U202214123，电子签名：

A 附录 A

A.1 实验一代码

```
%读取图像
img=imread('D:\Matlab_project\image_fourier\color.jpg');

% 获取图像的尺寸
[height, width, ~] = size(img); % 获取图像的高度、宽度和通道数

% 初始化灰度图像
gray_img = zeros(height, width); % 创建一个与原图像相同大小的零矩阵

% 将彩色图像转换为灰度图像
for i = 1:height
    for j = 1:width
        % 计算每个像素点的灰度值，使用常用的RGB转灰度公式
        gray_img(i, j) = 0.299 * double(img(i, j, 1))
        + 0.587 * double(img(i, j, 2))
        + 0.114 * double(img(i, j, 3));
    end
end

gray_img = uint8(gray_img);
```

```
imwrite(gray_img,'gray_img.jpg')
% 显示灰度图像
figure;
imshow(gray_img); % 将灰度图像的数据类型转换为uint8以便显示
title('灰度图'); % 添加标题

% 初始化灰度直方图数组
histogram = zeros(1, 256);
% 创建一个长度为256的零向量，用于存储每个灰度值的像素数量

% 统计灰度直方图
for i = 1:height
    for j = 1:width
        % 获取当前像素的灰度值
        gray_value = gray_img(i, j);
        % 增加对应灰度值的计数
        histogram(gray_value + 1) = histogram(gray_value + 1) + 1;
    end
end

% 绘制灰度直方图
figure;
bar([0:255], histogram);
% stem(0:255, histogram, 'filled');
% 绘制直方图，使用stem函数绘制填充的柱状图
title('灰度直方图'); % 添加标题
xlabel('灰度值'); % 添加X轴标签
ylabel('像素数'); % 添加Y轴标签

my_fft2(gray_img)

% 对图像进行二维傅里叶变换
F = fft2(double(gray_img));

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

% 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.jpg');

function F = my_fft1( x )
    % my_fft1 手动实现离散傅里叶变换 (DFT)
    % 输入:
    %   x - 输入信号向量
    % 输出:
    %   F - 计算得到的傅里叶变换结果向量

    N = length(x); % 获取输入信号的长度
    F = zeros(1,N); % 初始化输出傅里叶变换结果向量, 全部设置为0
    n = 1:N; % 创建一个从1到N的向量, 用于后续计算中的索引

    % 开始计算DFT
    for k = 1:N % 对于输出向量F中的每一个元素
        % 计算第k个频率分量的值
        % 利用复指数 e^(-j*2*pi*(k-1)*n/N) 与输入信号 x 相乘, 然后求和
        F(k) = sum(x .* exp(-1i * 2 * pi * (k-1) * n/N));
    end
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.jpg');
end
```

A.2 实验二代码

```
% 读取彩色图像
img = imread('color.jpg'); % 读取彩色图像文件

% 手写函数实现直方图均衡化
[eq_img_manual, eq_hist_manual] = hist_eq_manual(img);

% 使用MATLAB自带函数实现直方图均衡化
[eq_img_builtin, eq_hist_builtin] = hist_eq_builtin(img);

% 显示结果
figure;

eq_img_manual = uint8(eq_img_manual);
imshow(eq_img_manual);
imwrite(eq_img_manual,'eq_img_manual.jpg');
title('手写函数实现的直方图均衡化图像');

bar([0:255], eq_hist_manual);
title('手写函数实现的直方图均衡化灰度直方图');
```

```
eq_img_builtin=uint8(eq_img_builtin);
imshow(eq_img_builtin);
imwrite(eq_img_builtin,'eq_img_builtin.jpg');
title('MATLAB自带函数实现的直方图均衡化图像');

bar([0:255], eq_hist_builtin);
title('MATLAB自带函数实现的直方图均衡化灰度直方图');

function [eq_img, eq_hist] = hist_eq_manual(img)
    % hist_eq_manual 手写实现直方图均衡化
    % 输入:
    %   img - 输入的彩色图像
    % 输出:
    %   eq_img - 直方图均衡化后的图像
    %   eq_hist - 新图像的灰度直方图

    % 将彩色图像转换为灰度图像
    % gray_img = rgb2gray(img); % 使用内置函数将彩色图像转换为灰度图像

    % 获取图像的尺寸
    [height, width, ~] = size(img); % 获取图像的高度、宽度和通道数

    % 初始化灰度图像
    gray_img = zeros(height, width); % 创建一个与原图像相同大小的零矩阵

    % 将彩色图像转换为灰度图像
    for i = 1:height
        for j = 1:width
            % 计算每个像素点的灰度值，使用常用的RGB转灰度公式
            gray_img(i, j) = 0.299 * double(img(i, j, 1))
            + 0.587 * double(img(i, j, 2))
            + 0.114 * double(img(i, j, 3));
        end
    end

    gray_img = uint8(gray_img);

    % 获取图像的尺寸
    [height, width] = size(gray_img); % 获取图像的高度和宽度

    % 初始化灰度直方图数组
    hist = zeros(1, 256);
    % 创建一个长度为256的零向量，用于存储每个灰度值的像素数量
```

```
    % 统计灰度直方图
    for i = 1:height
        for j = 1:width
            % 获取当前像素的灰度值
            gray_value = gray_img(i, j);
            % 增加对应灰度值的计数
            hist(gray_value + 1) = hist(gray_value + 1) + 1;
        end
    end

    % 计算累积分布函数 (CDF)
    cdf = cumsum(hist); % 计算累积和
    cdf = cdf / max(cdf); % 归一化累积分布函数

    % 进行灰度映射
    eq_img = uint8(255 * cdf(gray_img + 1));
    % 将输入图像中的每个像素值替换成对应的输出灰度级

    % 计算均衡化后的图像的灰度直方图
    eq_hist = zeros(1, 256);
    % 创建一个长度为256的零向量，用于存储均衡化后的图像的灰度直方图
    for i = 1:height
        for j = 1:width
            % 获取当前像素的灰度值
            eq_gray_value = eq_img(i, j);
            % 增加对应灰度值的计数
            eq_hist(eq_gray_value + 1) = eq_hist(eq_gray_value + 1) + 1;
        end
    end
end

function [eq_img_builtin, eq_hist_builtin] = hist_eq_builtin(img)
    % hist_eq_builtin 使用MATLAB自带函数实现直方图均衡化
    % 输入:
    %   img - 输入的彩色图像
    % 输出:
    %   eq_img_builtin - 直方图均衡化后的图像
    %   eq_hist_builtin - 新图像的灰度直方图

    % 将彩色图像转换为灰度图像
    gray_img = rgb2gray(img); % 使用内置函数将彩色图像转换为灰度图像
```

```
    % 使用MATLAB自带的直方图均衡化函数
    eq_img_builtin = histeq(gray_img); % 应用直方图均衡化

    % 计算均衡化后的图像的灰度直方图
    eq_hist_builtin = imhist(eq_img_builtin); % 计算灰度直方图
end
```

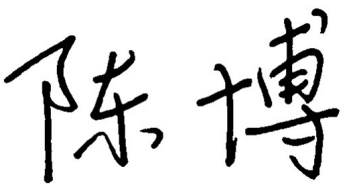
A.3 实验三代码

```
img = imread('color.jpg');

% 获取图像的尺寸
[height, width, ~] = size(img); % 获取图像的高度、宽度和通道数

% 初始化灰度图像
gray_img = zeros(height, width); % 创建一个与原图像相同大小的零矩阵

% 将彩色图像转换为灰度图像
for i = 1:height
    for j = 1:width
        % 计算每个像素点的灰度值，使用常用的RGB转灰度公式
        gray_img(i, j) = 0.299 * double(img(i, j, 1))
        + 0.587 * double(img(i, j, 2))
        + 0.114 * double(img(i, j, 3));
    end
end

gray_img = uint8(gray_img);

homo_f(gray_img);

function [img_after, hist_output] = homo_f(image)
    % 参数声明
    rH = 0.05; % 高频增益
    rL = 0.03; % 低频增益
    c = 0.04;  % 控制函数坡度的锐利程度，介于 rH 和 rL 之间
    D0 = 0.2;  % 截止频率

    % 获取图像尺寸
    [M, N] = size(image);

    % 对图像进行对数变换
    % 这一步是为了将图像的动态范围压缩，使得亮度变化更加均匀
```

```
img_log = log(double(image) + 1);

% 平移图像
% 通过交替符号来实现图像的平移，这有助于后续的傅里叶变换
img_py = zeros(M, N);
for i = 1:M
    for j = 1:N
        if mod(i + j, 2) == 0
            img_py(i, j) = img_log(i, j);
        else
            img_py(i, j) = -1 * img_log(i, j);
        end
    end
end

% 对平移后的图像进行傅里叶变换
% 傅里叶变换将图像从空间域转换到频率域
img_py_fft = fft2(img_py);

% 定义同态滤波函数（高斯滤波）
% 构建一个高斯滤波器，用于增强高频部分并抑制低频部分
H = zeros(M, N);
r = rH - rL;
D = D0^2; % 设置参数
m_mid = floor(M / 2); % 中心点坐标
n_mid = floor(N / 2);

for i = 1:M
    for j = 1:N
        dis = ((i - m_mid)^2 + (j - n_mid)^2); % 计算每个点到中心点的距离
        H(i, j) = r * (1 - exp((-c) * (dis / D))) + rL; % 高斯同态滤波函数
    end
end

% 应用滤波器
% 将滤波器应用于频率域中的图像
imgtemp = img_py_fft .* H;

% 开始反变换
% 将处理后的频率域图像转换回空间域
imgtemp = abs(real(ifft2(imgtemp)));

% 反对数变换
```

```
    % 将图像从对数域转换回原始域
    imgtemp = exp(imgtemp) - 1;

    % 归一化处理
    % 将图像数据归一化到 0-255 的范围内
    max_num = max(imgtemp(:));
    min_num = min(imgtemp(:));
    range = max_num - min_num;
    img_after = uint8(255 * (imgtemp - min_num) / range);

    % 计算灰度直方图
    % 使用 imhist 函数计算处理后的图像的灰度直方图
    hist_output = imhist(img_after);

    % 显示结果图像
    figure;

    imshow(img_after);
    imwrite(img_after,'img_after.jpg');

    % 显示灰度直方图
    stem(0:255, hist_output, 'filled');
    bar([0:255], hist_output);
    title('变换后灰度直方图');
    xlabel('灰度值');
    ylabel('像素数');
end
```

A.4 电子签名



陈博-人工智能 2204 班-U202214123，电子签名：