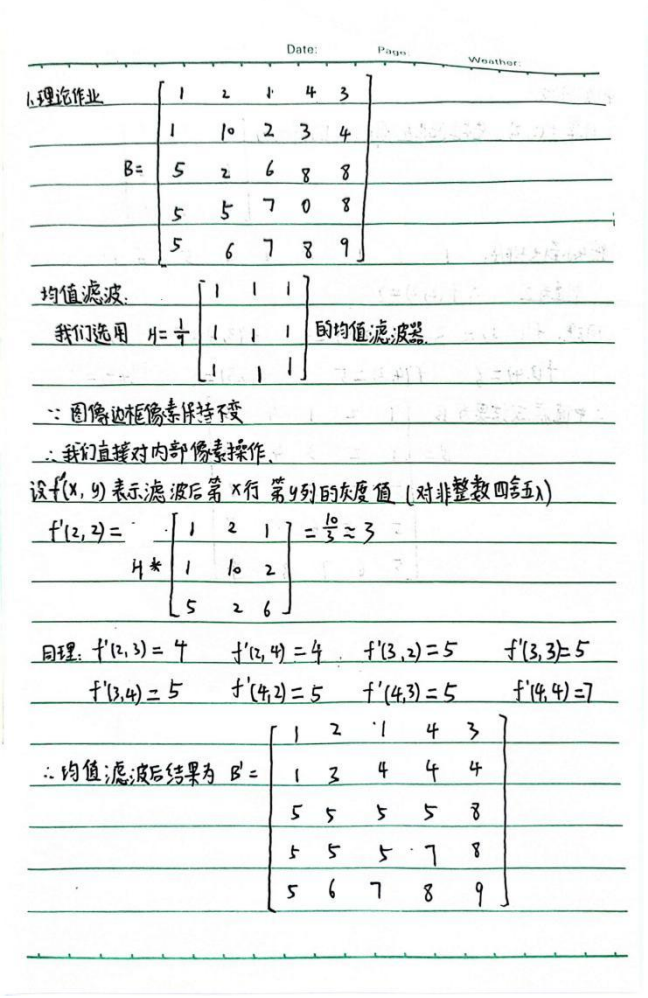
电子签名：

姚慧峰

一．理论作业



1.理论作业

$$B=\begin{bmatrix}1&2&1&4&3\\1&10&2&3&4\\5&2&6&8&8\\5&5&7&0&8\\5&6&7&8&9\end{bmatrix}$$

均值滤波：

我们选用 $H=\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ 的均值滤波器

∵图像边框像素保持不变

∴我们直接对内部像素操作。

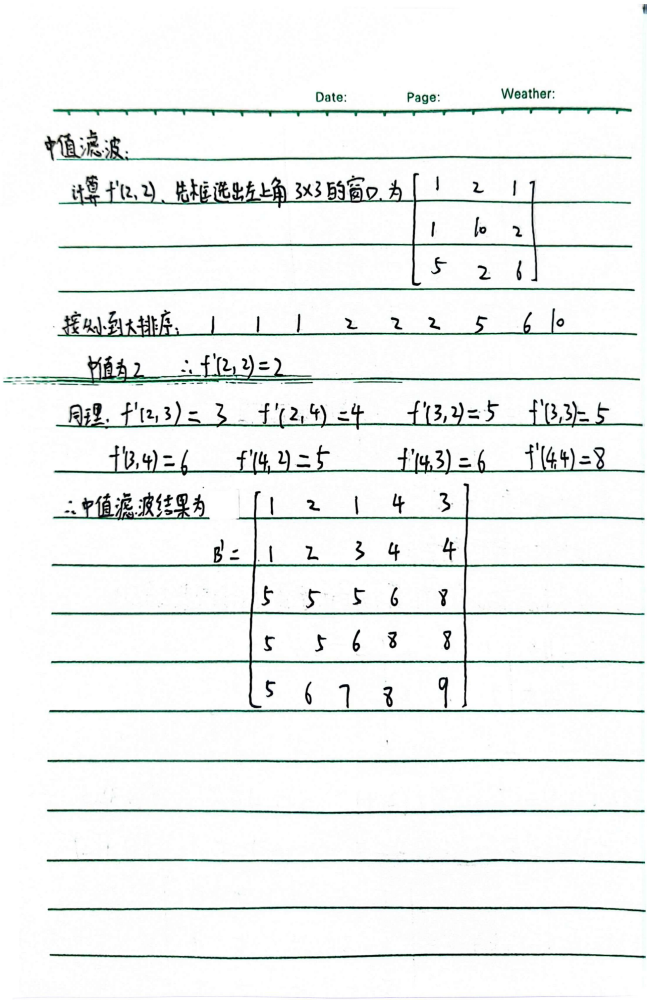
设 $f'(x,y)$ 表示滤波后第 x 行 第 y 列的灰度值（对非整数四舍五入）

$$f'(2,2)=\begin{bmatrix}1&2&1\\1&10&2\\5&2&6\end{bmatrix}*H=\frac{10}{3}\approx 3$$

同理：$f'(2,3)=4$　$f'(2,4)=4$　$f'(3,2)=5$　$f'(3,3)=5$

$f'(3,4)=5$　$f'(4,2)=5$　$f'(4,3)=5$　$f'(4,4)=7$

∴均值滤波后结果为 $B'=\begin{bmatrix}1&2&1&4&3\\1&3&4&4&4\\5&5&5&5&8\\5&5&5&7&8\\5&6&7&8&9\end{bmatrix}$



二. 编程作业

2.1 灰度直方图与离散傅里叶变换幅度图

2.1.1 原理

灰度直方图原理很简单，就是统计各个灰度值对应像素点的个数，然后将统计结果绘制为直方图展现出来。

离散傅里叶变换幅度图在上次作业中我们已经编程实现并给出了原理，本次作业我们只分析最终问题的结果。

2.1.2 实现代码

```
% 转换为灰度图
%编程转化为灰度图
% 初始化灰度图像矩阵
gray_img = zeros(size(img, 1), size(img, 2));
%遍历每个像素
for i = 1:size(img, 1)
    for j = 1:size(img, 2)
        % 获取 RGB 值并采用加权平均的方法求灰度值
        r = img(i, j, 1);
        g = img(i, j, 2);
        b = img(i, j, 3);
    gray_img(i, j) = 0.299 * r + 0.587 * g + 0.114 * b;
end
end
gray_img=uint8(gray_img);
% 显示灰度图
figure;
imshow(gray_img);title('Image 2');
% 输入图像
[m,n]=size(gray_img2); % 计算图像的长宽
p=zeros(1,256); %创建数组存储像素个数
% 统计每个像素值出现的概率， 得到概率直方图
% 用 length 函数计算相同像素的个数
for i=0:255
    p(i+1)=length(find(img==i));
end
figure;bar([0:255],p);
% 计算二维离散傅里叶变换（DFT）
perform_dft(gray_img2);
% 函数实现
function magnitude_spectrum = perform_dft(gray_img)
% 下面是傅里叶正变换必备的一些矩阵:
[M,N]=size(gray_img);
Wm = exp(-1i*2*pi/M);
Wn = exp(-1i*2*pi/N); % 不同 G 中用不同的 W
Em = zeros(M);
En = zeros(N); % E 是辅助计算矩阵
Gm = zeros(M)+Wm;
Gn = zeros(N)+Wn; % G 是计算时要用的矩阵
F = zeros(M,N); % F 是转换到频域的结果
E = zeros(M,N);
```

```
% 对 Gm 的计算：循环长度为 M
fprintf('二维离散傅里叶变换开始:\n');
for row = 0:M-1
    for col = 0:M-1
        Em(row+1,col+1) = row * col;
        Gm(row+1,col+1) = Gm(row+1,col+1)^Em(row+1,col+1);
    end
end
%变换到图像中点
% 对 Gn 的计算：循环长度为 N
for row = 0:N-1
    for col = 0:N-1
        En(row+1,col+1) = row * col;
        Gn(row+1,col+1) = Gn(row+1,col+1)^En(row+1,col+1);
    end
end
for row =1:M
    for col =1:N
        E(row,col)=double(gray_img(row,col))*((-1)^(row+col));
    end
end
F = real(Gm*E*Gn);
subplot(1, 2, 1), imshow(gray_img), title('Original Grayscale Image');
subplot(1, 2, 2), imshow(magnitude_spectrum, []), title('Magnitude Spectrum');
end
```

2.1.3 结果分析

图 1 灰度图

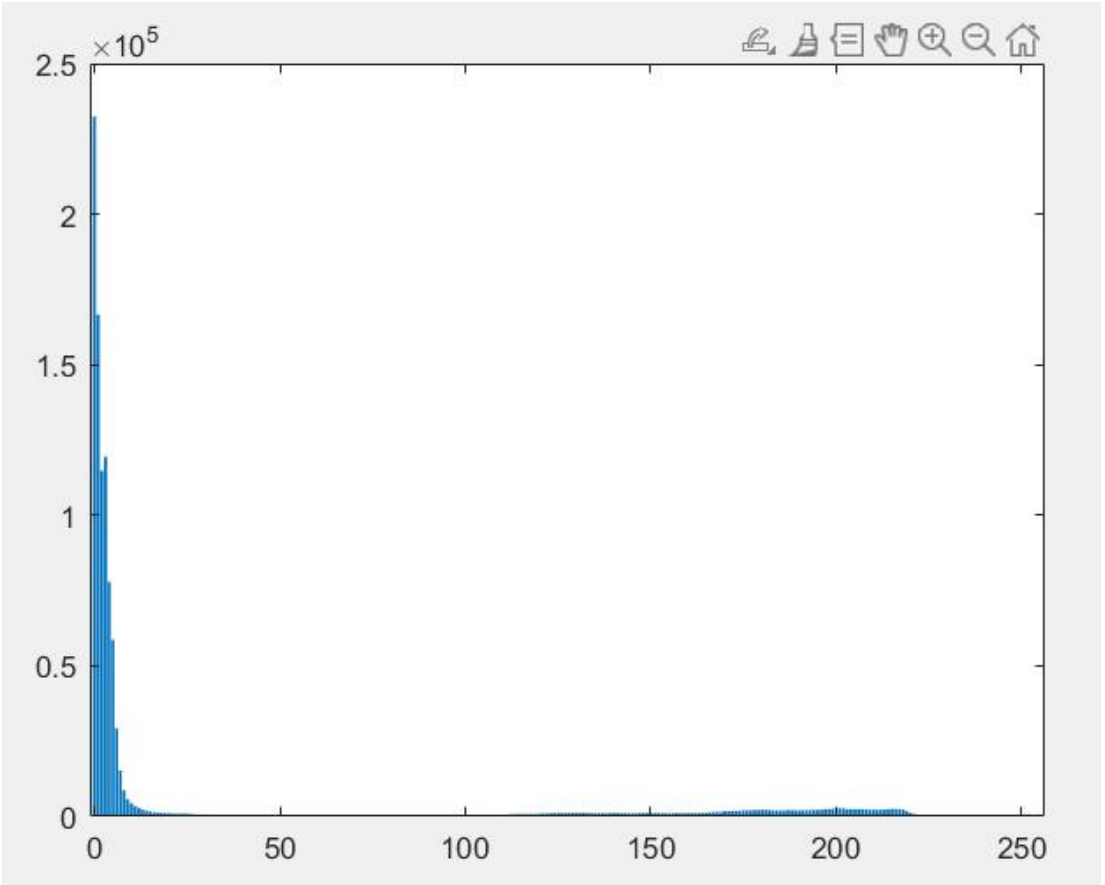


图 2 灰度直方图

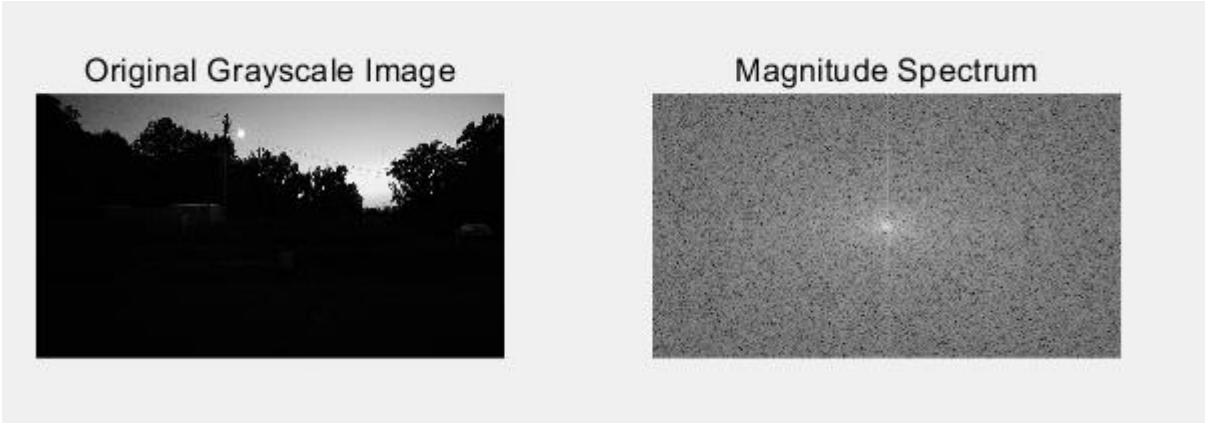


图 3　离散傅里叶变换幅度图

分析灰度图，我们可以发现这张照片整体水平偏暗，接着我们先看灰度直方图，如图 2，灰度级集中分布在 25 以下的部分，再看频谱图，如图 3，图像大部分频率分布在低频附近，也有高频成分，这可能是由于照片整体上灰度较暗的缘故，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。

2.2 直方图均衡化和同态滤波

2.2.1 原理

1.直方图均衡化原理：

直方图均衡化是将原始图像的直方图变为均衡分布的的形式，将一非均匀灰度概率密度分布图像，通过寻求某种灰度变换，变成一幅具有均匀概率密度分布的目的图像。

我们要找到一种变换 t=T(s)使直方图变均匀，为使变换后的灰度级仍保持从黑到白的次序不变，且变换后的像素灰度在允许的范围内，必须规定：

在 0≤s≤1 内，T(s)为单调递增函数；

在 0≤s≤1 内，有 0≤T(s)≤1。

因为灰度变换不影响像素的位置分布，而且也不会增减像素数目，所以有如下的推导公式：

$$\int_0^r p(r)dr = \int_0^s p(s)ds = \int_0^s 1 \cdot ds = s = T(r)$$

$$T(r) = \int_0^r p(r)dr$$

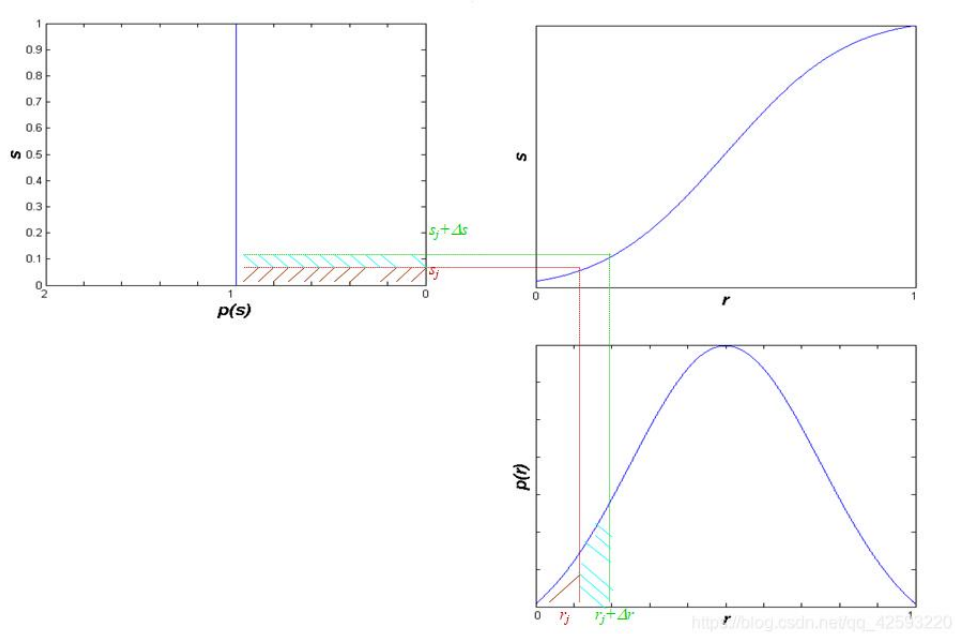


图 4　　直方图均衡化原理展示

2. 同态滤波原理

按照光图像的成像原理，可以对一幅图像进行如下的简单建模：

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

即把图像亮度 f(x,y)看成是由入射分量（入射到景物上的光强度）i(x,y)和反射分量（景物反射的光强度）r(x,y)组成。

同态滤波是一种在频域中同时进行图像对比度增强和压缩图像亮度范围的滤波方法，其基本思想是减少入射分量 i(x,y)，并同时增加反射分量 r(x,y)来改善图像 f(x,y)的显示效果，i(x,y)在空间上变化缓慢，其频谱集中在低频段，r(x,y)反映图像的细节和边缘，其频谱集中在高频段。

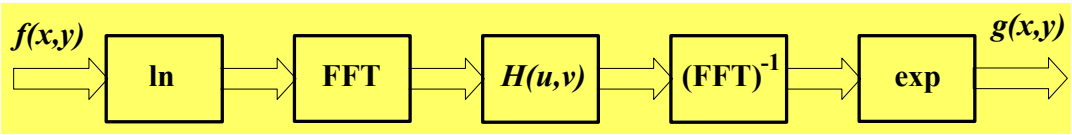


图 5 同态滤波过程演示

在本次作业中，我们选用高斯同态滤波对图片进行处理，下面是我们的代码展示。

2.1.2 实现代码

```
%直方图均衡化
hestepimg(gray_img)
%函数实现
function his=hestepimg(img)
    % 输入图像
```

```
    [m,n]=size(img); % 计算图像的长宽
    p=zeros(1,256); %创建数组存储像素概率
    % 统计每个像素值出现的概率， 得到概率直方图
    % 用 length 函数计算相同像素的个数
    for i=0:255
        p(i+1)=length(find(img==i))/(m*n);
    end
    % 求累计概率，得到累计直方图
    s=zeros(1,256);
    for i=1:256
        for j=1:i
            s(i)=p(j)+s(i);
        end
    end
    %完成每个像素点的映射
    a=round(s*255); %四舍五入
    his=img;
    for i=0:255
        his(img==i)=a(i+1);
    end
    %均衡化后的直方图
    k=zeros(1,256); %创建数组存储像素概率
    % 用 length 函数计算相同像素的个数
    for i=0:255
        k(i+1)=length(find(his==i));
    end
    %展示均衡化前后图片
    figure;
    subplot(121),imshow(img);title('原图')
    subplot(122),imshow(his);title('均衡化后')
    figure;bar([0:255],k);%展示直方图均衡化后的灰度直方图
end
%同态滤波
homomorphic(gray_img2)
%函数实现
function imgtemp=homomorphic(image)
%参数声明
rH = 0.15;
rL = 0.1;
c = 0.15;%介于 rH 和 rL 之间
D0 = 0.2;
[M, N] = size(image);
%取对数
img_log = log(double(image) + 1);
```

```
%平移到中心，判断语句代替指数计算
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
img_py_fft = fft2(img_py);
%同态滤波函数
%滤波器选用高斯滤波
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);%到中心店的距离
        H(i, j) = r * (1-exp((-c)*(dis/D))) + rL;%高斯同态滤波函数
    end
end
imgtemp = img_py_fft.*H;%滤波
%开始反变换
imgtemp = abs(real(ifft2(imgtemp)));
imgtemp = exp(imgtemp) - 1;
%归一化处理
max_num = max(imgtemp(:));min_num = min(imgtemp(:));
range = max_num - min_num;
img_after = zeros(M,N,'uint8');
for i = 1 : M
    for j = 1 : N
        img_after(i,j) = uint8(255 * (imgtemp(i, j)-min_num) / range);
    end
end
figure;%展示同态滤波前后照片
subplot(1,2,1), imshow(image), title('原图像');
subplot(1,2,2), imshow(img_after), title('变换后');
end
```

2.1.3 结果分析

图 6　直方图均衡化前后对比图

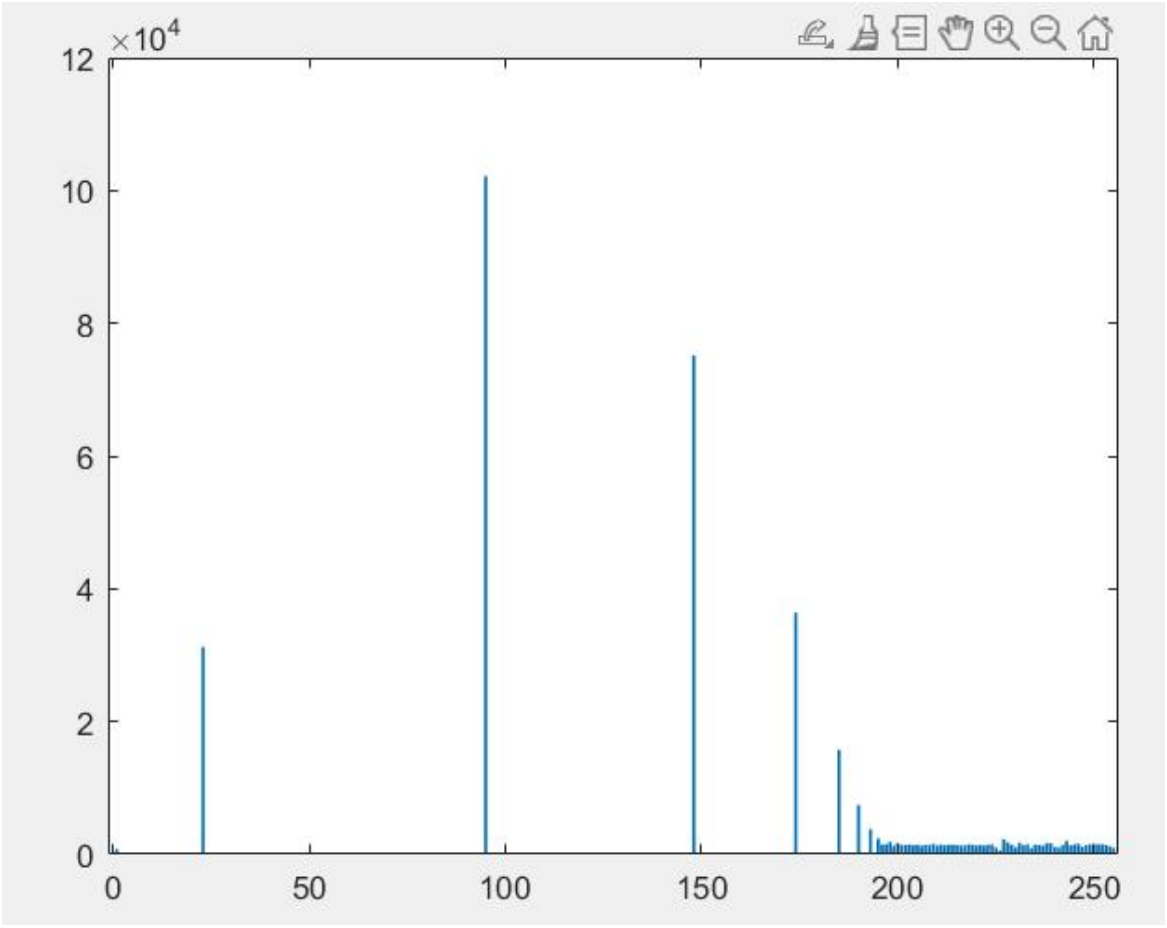


图 7 均衡化后的灰度直方图

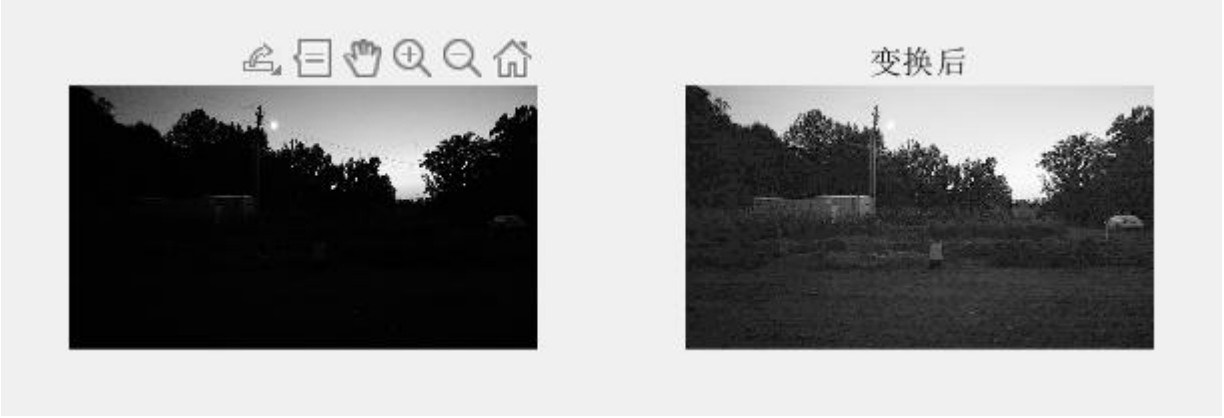


图 8　　同态滤波前后对比图

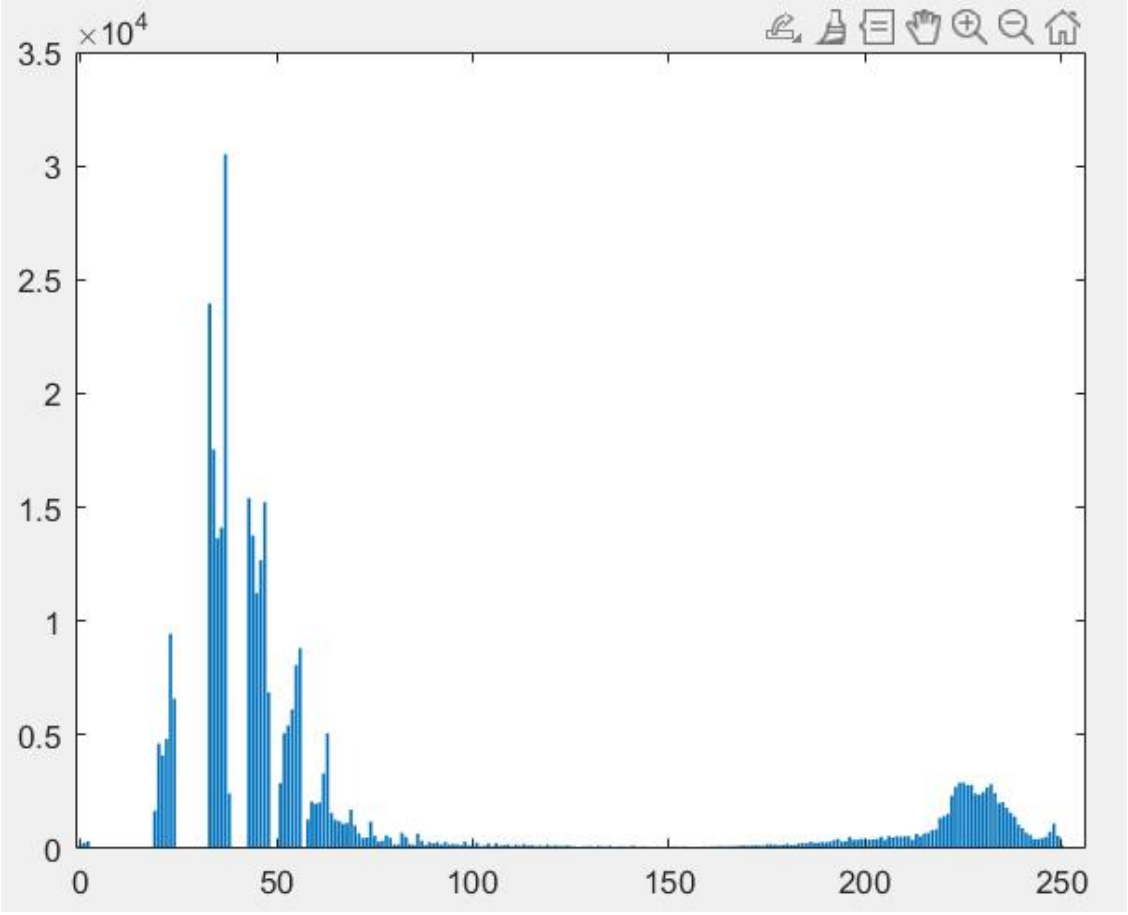


图 7　　同态滤波后的灰度直方图

我们首先来看直方图均衡化的结果，从结果的灰度直方图我们可以看出，虽然还是会有部分灰度数量很大，但是我们已经基本实现了讲将原本密集的灰度展开为比较均衡的结果。从图像的效果来看，我们也可以看出改进后的图像明显在灰度对比上有了很大的提高，原本看不出来物体已经基本可以看出形状，而且明显灰度级更高。我们再来看同态滤波灰度直方图，我们可以看出在滤波后还是有相当一部分灰度集中在较低的灰度，但是跟原图相比还是

有所提高。再来看滤波后的图片，我们发现滤波后的照片整体还是偏暗，但是物体之间的对比度已经有了明显的提高，我们基本可以分辨出照片中的物体。最后，我们来对比直方图均衡化结果与同态滤波结果，从感官来上将，我个人觉得同态滤波的结果要优于直方图均衡化的结果，我们明显感觉到直方图均衡化后图片变得很不连续，让人感觉怪怪的，而同态滤波虽然整体比直方图均衡化结果要暗了许多，但是给人的感觉效果要远远优于直方图均衡化。

我们从原理上考虑二者的区别，直方图均衡化只是为了让灰度级的分布更加均衡，但实际上图片最好的效果灰度级往往不是这样分布的，我们想要达到更好的视觉效果，可以分析图中包含的要素，利用直方图规定化来操作，通过给定最终我们分析得出的灰度直方图，效果应该会有所提升。而同态滤波主要是利用入射与反射分量的关系来进行操作，通过在频域上降低低频部分，加强高频来达到减少入射分量，并同时增加反射分量的目的，进而增强对比度来达到让我们看清暗区中物体的目的。由于我们在滤波时保留了一部分低频分量，所以这样做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。