

实验一 实验环境及图像的基本操作

实验目的：熟悉实验环境，掌握 matlab 进行图像处理的基本操作。

实验原理：

(1)将一幅图像视为一个二维矩阵。

(2)利用 MATLAB 图像处理工具箱读、写和显示文件。

实验内容：

一、图像的基本操作

1、图像的读取和显示

(1)图像读取。

调用 `imread` 函数将图像文件读入图像数组（矩阵）。例如 `"f=imread('tire.tif')"`。其基本格式为：“`A=imread(filename fmt);`”，其中，`A` 为二维矩阵，`filename` 为文件名，`fmt` 为图像文件格式的扩展名。绝对路径和相对路径都可以，使用相对路径时要注意图片是放在当前工作目录下的。

(2)图像显示。

`imshow()`显示图像,注意在括号里面放的是图像对应的矩阵名。显示时，分多窗口和单窗口。`figure` 表示新建一个窗口的意思,若打开三个窗口，则需要使用三次 `figure`。很多时候,我们需要在一个窗口上显示多幅图像,以方便比较不同操作处理后的效果差异，因此需要用到 `subplot()`函数，这个函数用来控制窗口布局,以 `subplot(1,3,1)`为例，三个参数的含义依次是，布局为 1 行 3 列，当前图片所处的位置是第 1 个。接下来，**完成下列操作：**

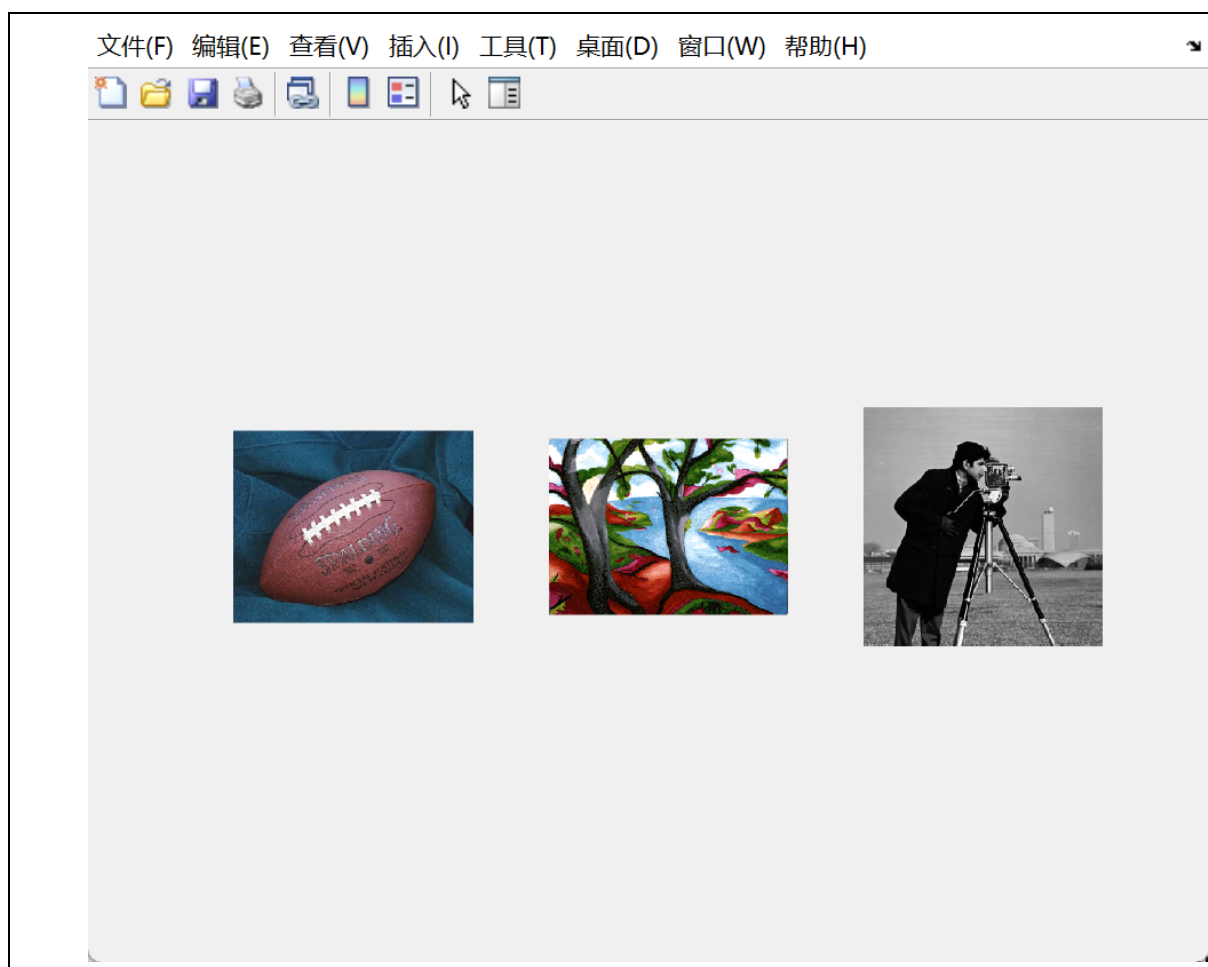
1. 使用函数 `imread` 将一幅 RGB 图像 'football.jpg' 读入 MATLAB 环境，使用 `imshow` 显示该图片；

2. 使用函数 `imread` 将一幅索引图' `trees.tif` ' 像读入 MATLAB 环境，使用 `imshow` 显示该图片；
3. 使用函数 `imread` 将一幅灰度图像'`cameraman.tif`'读入 MATLAB 环境，使用 `imshow` 显示该图片。
4. 使用 `subplot` 函数，将上面的三幅图像显示在一个图形窗口中。

请将实验代码贴在此处：

```
clear,clc,close all;  
% 1.图像的读取和显示  
f1 =imread('football.jpg');  
% imshow(f1);  
[f2,map2] = imread('trees.tif');  
% imshow(f2,map2);  
[f3,map3]=imread('cameraman.tif');  
% imshow(f3,map3);  
figure  
subplot(1,3,1);  
imshow(f1);  
subplot(1,3,2);  
imshow(f2,map2);  
subplot(1,3,3);  
imshow(f3,map3);
```

请将运行结果贴在此处：



二、不同类型的图像的转换（可以用前一题的图像）

1. 使用函数 `rgb2gray` 将一幅 `rgb` 图像转化为灰度图像，并且将原图和处理后的图显示在一个图像窗口中。
2. 使用函数 `im2bw` 将一幅灰度图像转化为二值图像，选择两个不同的阈值 `0.3`，`0.7`，并且将原图和处理后的图显示在一个图像窗口中。

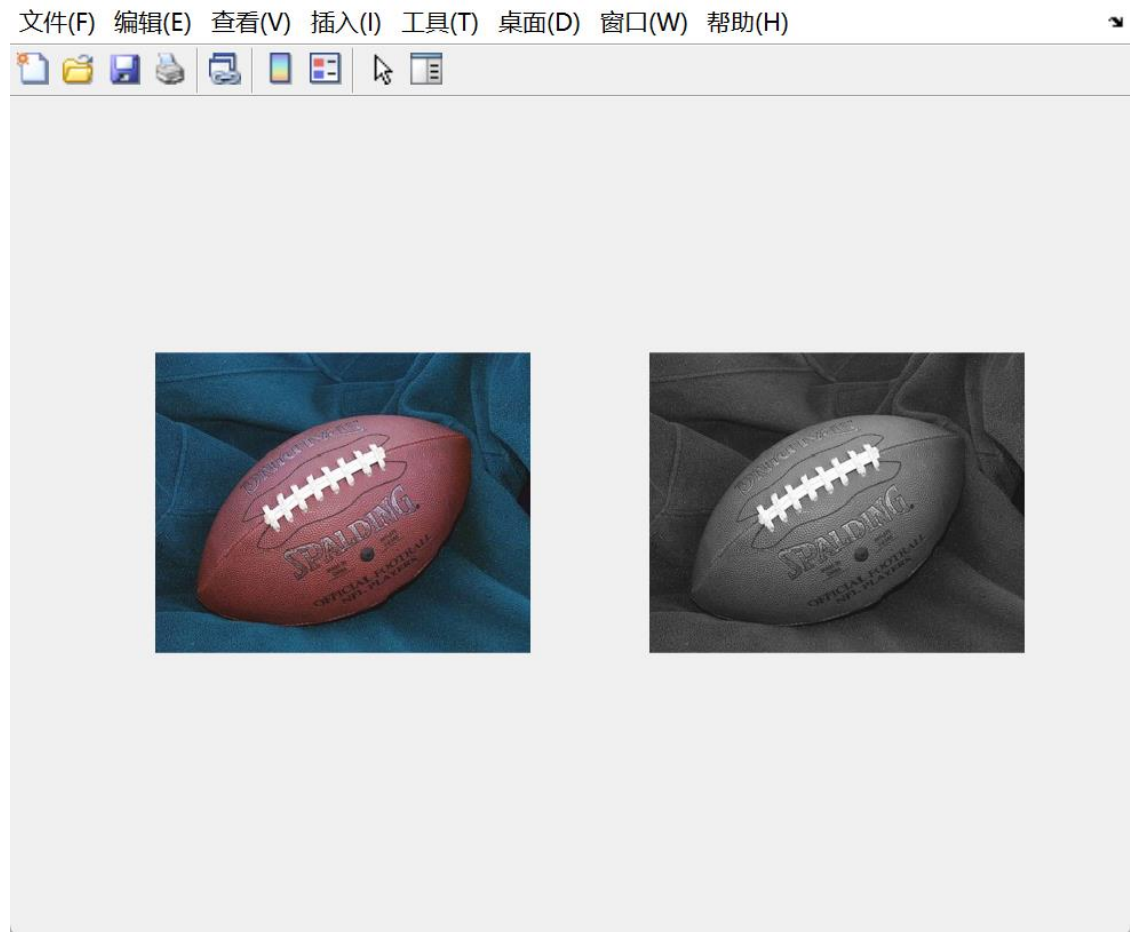
请将实验代码贴在此处：

```
%% 2.不同类型的图像的转换
% work1
g1 = rgb2gray(f1);
figure
subplot(1,2,1),imshow(f1);
subplot(1,2,2),imshow(g1);
% work2
bw1 = im2bw(f1,0.3);
bw2 = im2bw(f1,0.7);
%官方推荐使用 imbinarize
```

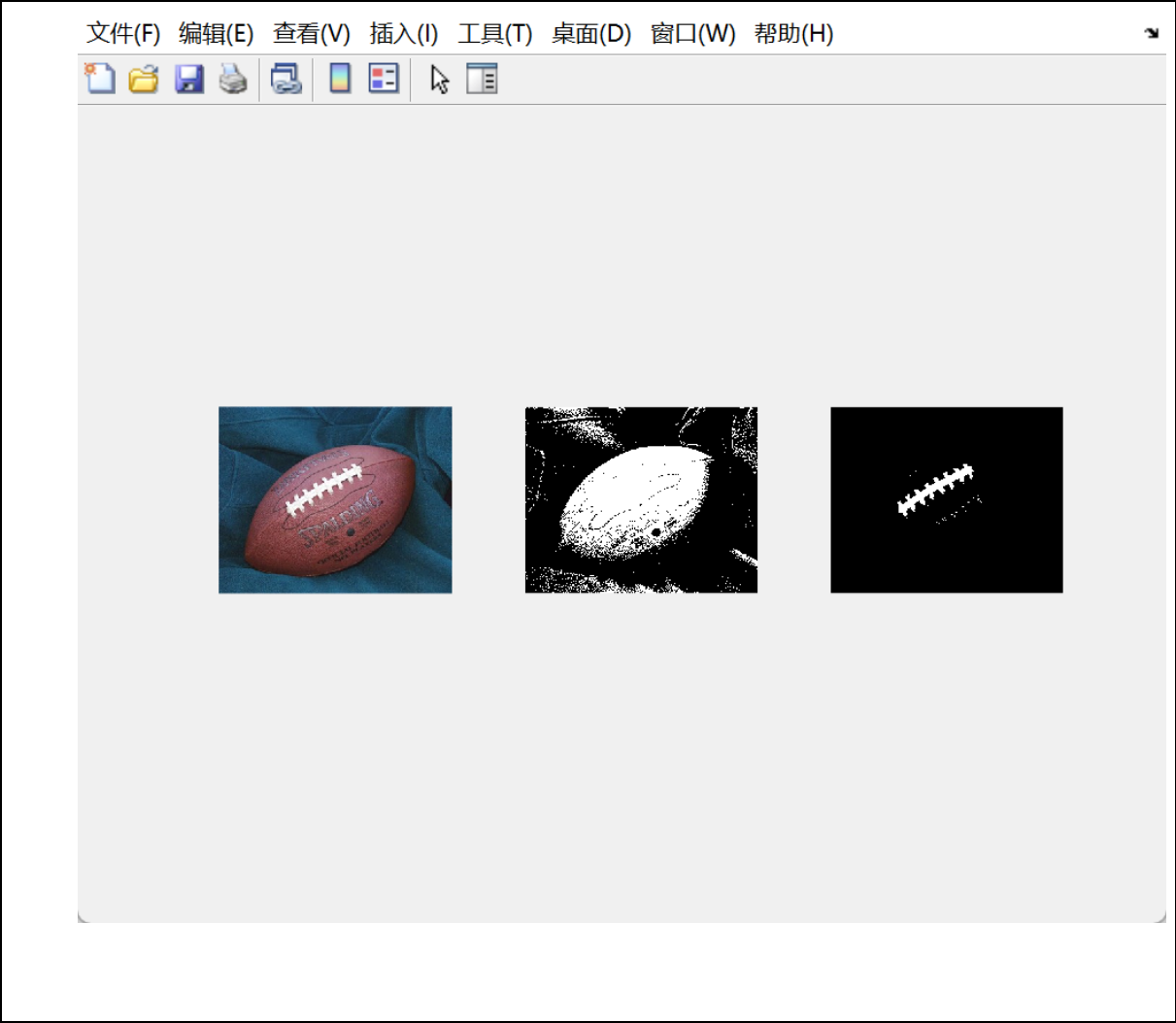
```
figure
subplot(1,3,1),imshow(f1);
subplot(1,3,2),imshow(bw1);
subplot(1,3,3),imshow(bw2);
```

请将运行结果贴在此处：

任务 1



任务 2



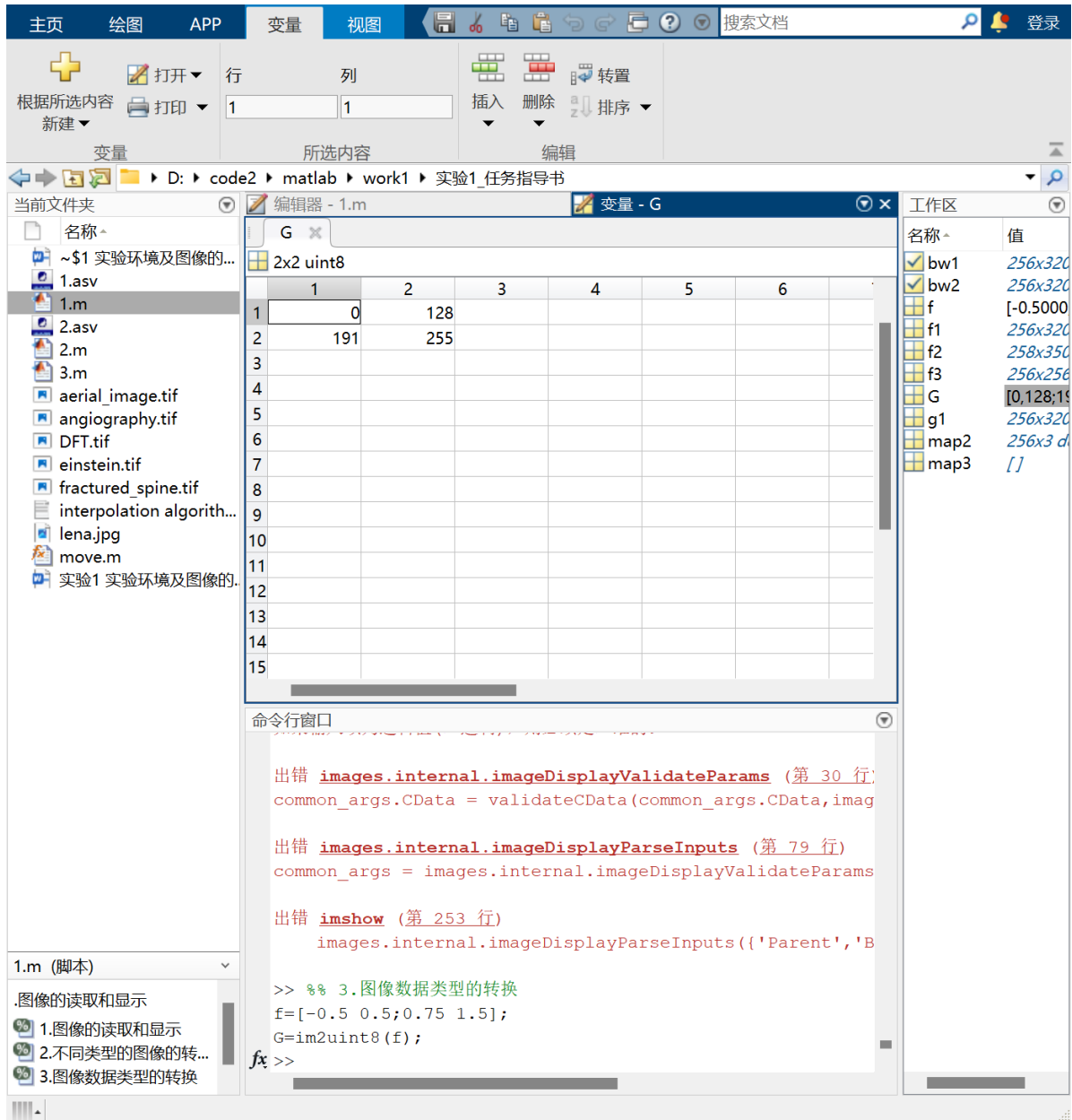
三、图像数据类型的转换

IPT 工具箱提供了特定的函数进行尺度放缩， 为图像数据类型转换做准备。函数 `im2uint8` 可以检测输入的数据类型， 并为工具箱执行所有必要的缩放以将数据识别为有效的图像数据。下图中是常用的图像数据类型的转换函数：

Name	Converts Input to:	Valid Input Image Data Classes
<code>im2uint8</code>	<code>uint8</code>	<code>logical</code> , <code>uint8</code> , <code>uint16</code> , and <code>double</code>
<code>im2uint16</code>	<code>uint16</code>	<code>logical</code> , <code>uint8</code> , <code>uint16</code> , and <code>double</code>
<code>mat2gray</code>	<code>double</code> (in range <code>[0, 1]</code>)	<code>double</code>
<code>im2double</code>	<code>double</code>	<code>logical</code> , <code>uint8</code> , <code>uint16</code> , and <code>double</code>
<code>im2bw</code>	<code>logical</code>	<code>uint8</code> , <code>uint16</code> , and <code>double</code>

1. 运行下面的程序段，把运行结果贴在下面：

```
f=[-0.5 0.5;0.75 1.5];  
G=im2uint8(f);
```



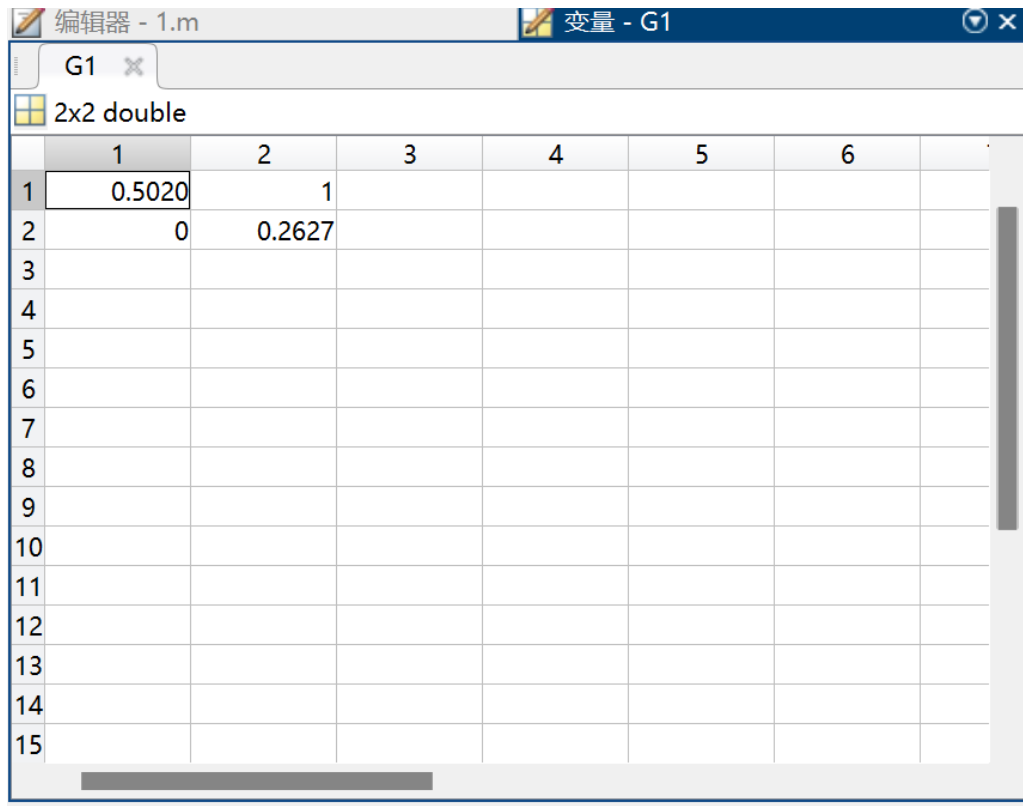
2. `mat2gray` 函数的用法。该函数可以将任意 `double` 类型的矩阵转换为放缩到 $[0, 1]$ 区间的 `double` 类型矩阵，其基本语法为：`G=mat2gray(A, [Amin, Amax])`。其中图像 `G` 中像素取值在区间 $[0, 1]$ 之间，0 代表纯黑，1 代表纯白。参数 `Amin` 和 `Amax` 的存在使得 `A` 中所有小于等于 `Amin` 值的像素值变成 0，所有大于等于 `Amax` 的像素值变成 1。如果不显式给出 `Amax` 和 `Amin`，如 `G=mat2gray(A)`，则 `Amax` 和 `Amin` 会被设置为 `A` 中实际最大和

最小的像素值。运行下面的程序段，把运行结果贴在下面：

```
A=[128,300;-12,66.98];
```

```
G1=mat2gray(A,[0,255]);
```

```
G2=mat2gray(A);
```



	1	2	3	4	5	6
1	0.5020	1				
2	0	0.2627				
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

G2							
2x2 double							
	1	2	3	4	5	6	
1	0.4487	1.0000					
2	0	0.2531					
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

四、永远放在代码开头的 `clear,clc,close all;`

为了给即将运行的代码一个干净的环境，一般会在代码开头加上这段命令。

(1) **clear** 清除工作空间中的变量,有时候，我们会在 **matlab** 中运行多个代码文件,如果两个代码文件中有同名变量，那么会造成干扰但如果在运行时，清除掉之前运行的变量，就不会有这种干扰存在。

(2) **clc** 清除命令行窗口的输入。

(3) **close al** 关闭所有的窗口。

二、图像的基本运算

1. 用 **matlab** 编程实现灰度的反转，设图像的灰度级为 L ，则图像的灰度反转可表示为：

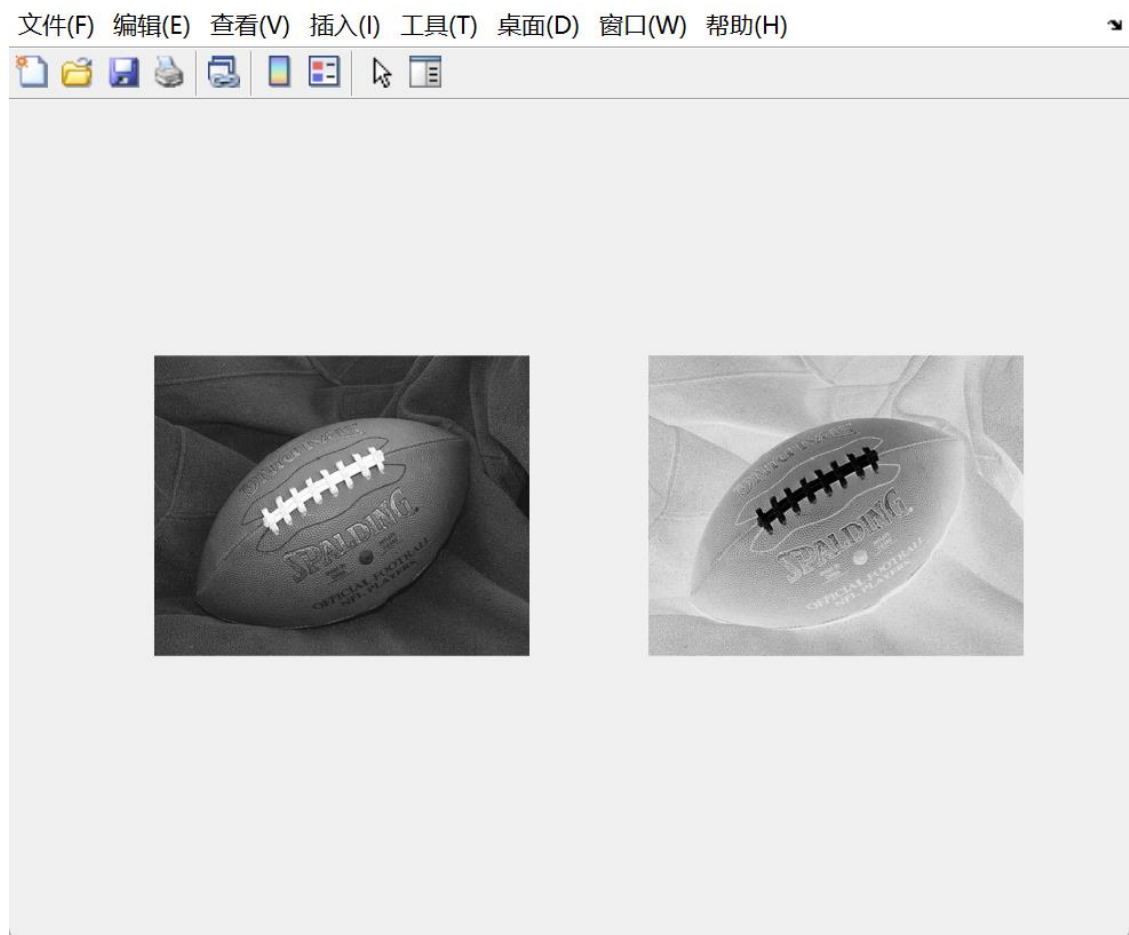
$$g(x,y)=L-1-f(x,y)$$

把反转前后的结果显示在一个窗口中。

请将实验代码贴在此处:

```
clear;clc;close all;  
f = imread("football.jpg");  
%% 1. 图像反转  
g = im2gray(f);  
g1 = 255-g;  
figure  
subplot(121),imshow(g);  
subplot(122),imshow(g1);
```

请将运行结果贴在此处:



2.用 matlab 编程实现对图像'DFT.tif'进行对数变换, 对原图像 $f(x,y)$ 进行对数变换的解析式可表示为:

$$g(x, y) = c \cdot \log(1 + f(x, y))$$

需要**注意**的是: 要先把图像的数据类型转换成 **double** 型, 再去做对数变换。

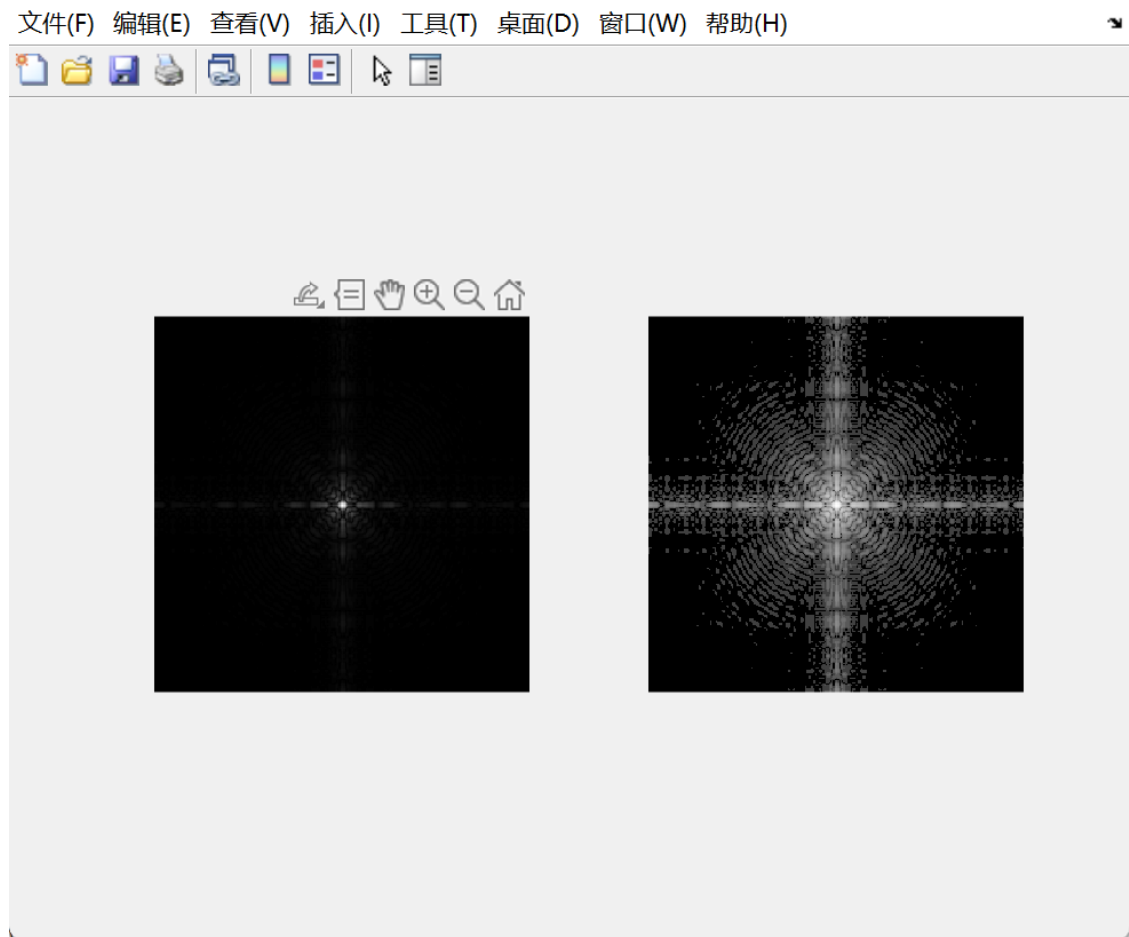
把对数变换前后的结果显示再一个窗口中。

请将实验代码贴在此处:

```
%% 2. 对数变换
```

```
f2 = imread('DFT.tif');
c = 1.0;
%mat2gray 将数据缩放到 0-1,直接使用 imshow 只会显示 0-1 之间的像素
g2 = mat2gray(c*log(1+double(f2)));
figure
subplot(121),imshow(f2);
subplot(122),imshow(g2);
```

请将运行结果贴在此处：



3. 使用 **imnoise** 函数给一幅图像添加高斯噪声，然后使用加法运算消除这幅图像的附加噪声。

请将实验代码贴在此处：

```
%% 3.消除加性噪声
clear;clc;close all;

rgb = imread('eight.tif');

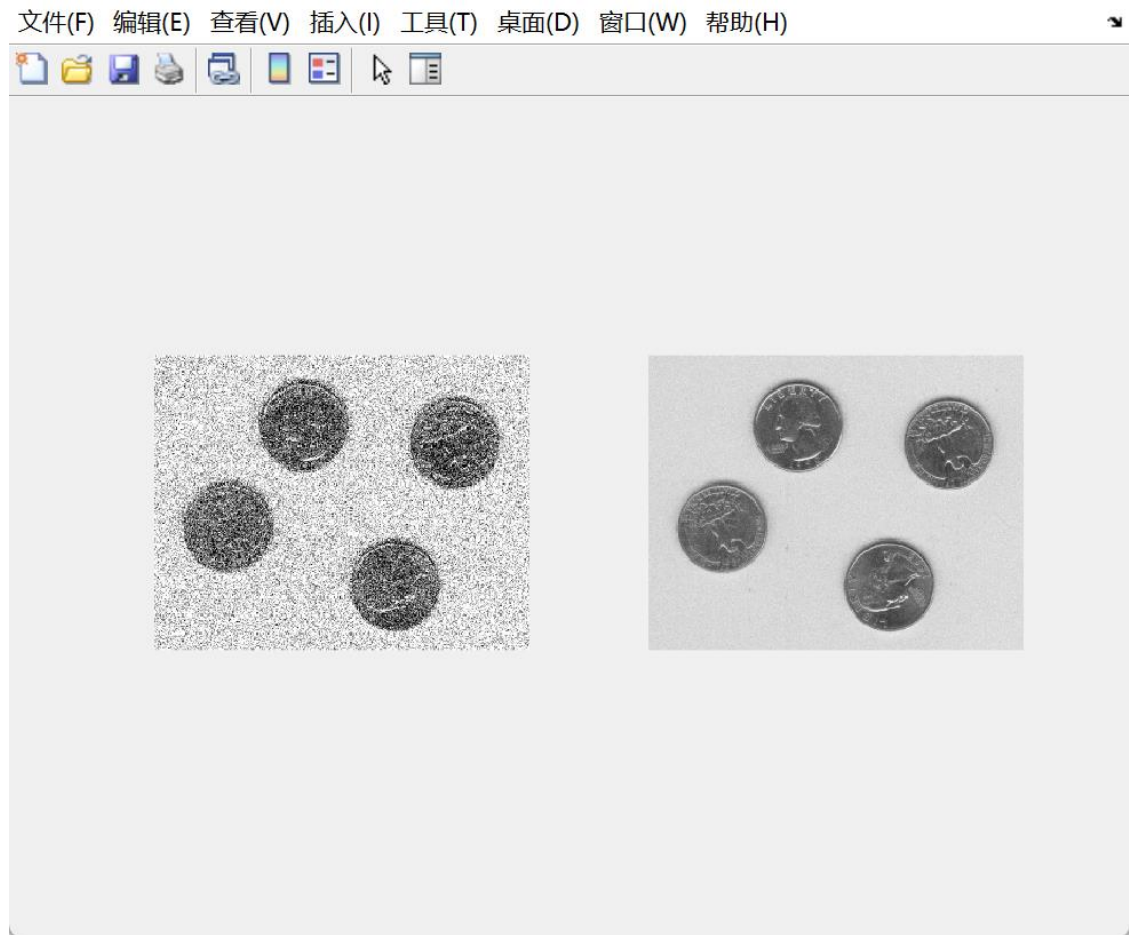
% A = imnoise(rgb,'gaussian',0,0.05);
I = im2double(zeros(size(rgb)));
M = 100;
rgb = im2double(rgb);
```

```

for i = 1:M
    A = imnoise(rgb,"gaussian",0,0.05);
    I = imadd(I,A);
end
avg_A = I/M;
figure
subplot(121),imshow(A);
subplot(122),imshow(avg_A);

```

请将运行结果贴在此处：



4 用 Matlab 语言编写统计一幅灰度图像的直方图，并在同一个图形窗口中显示该图像及其灰度直方图；

请将实验代码贴在此处：

```

%% 4. 灰度直方图
clear;clc;close all;

f4 = imread('football.jpg');
g = im2gray(f4);

```

```
% 计算灰度图像的直方图
```

```
[counts, ~] = imhist(g);
```

```
figure
```

```
subplot(131),imshow(f4),title('原始图像');
```

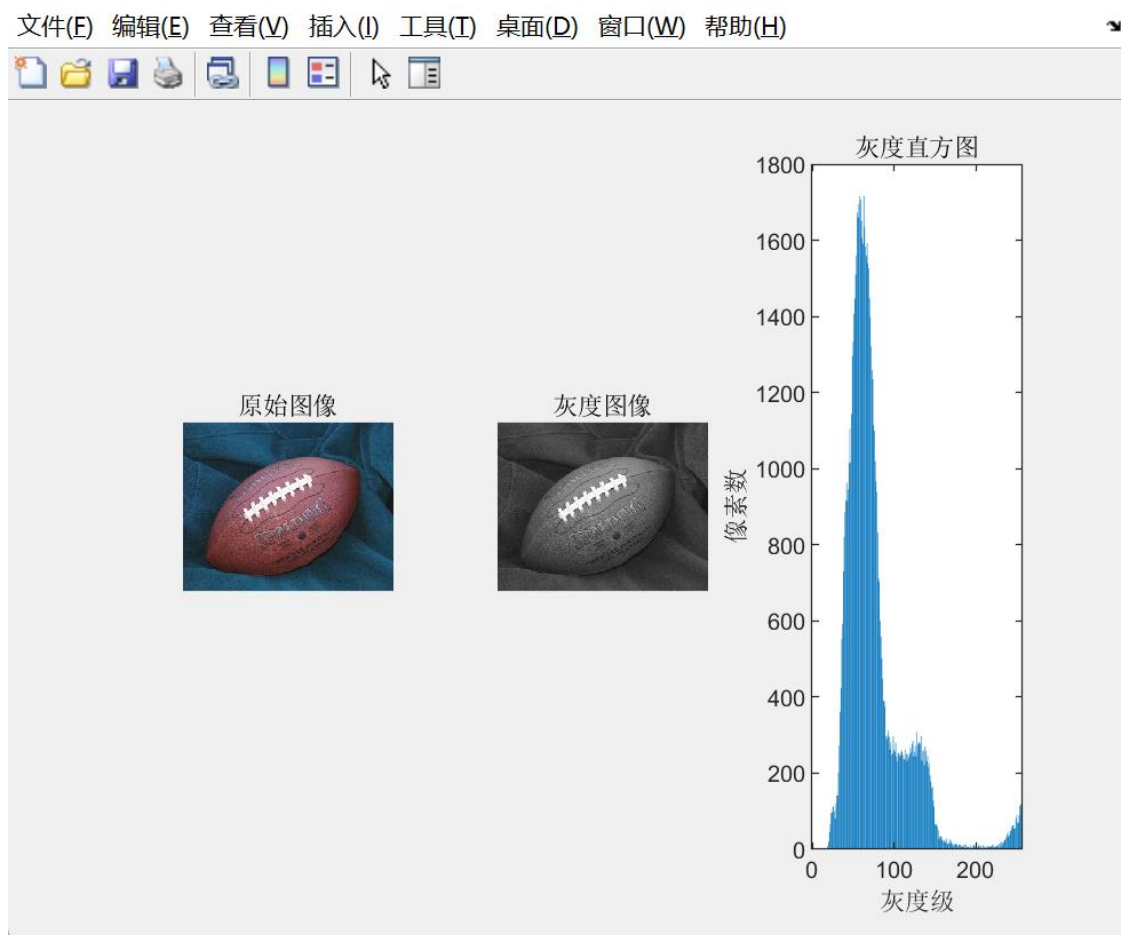
```
subplot(132),imshow(g),title('灰度图像');
```

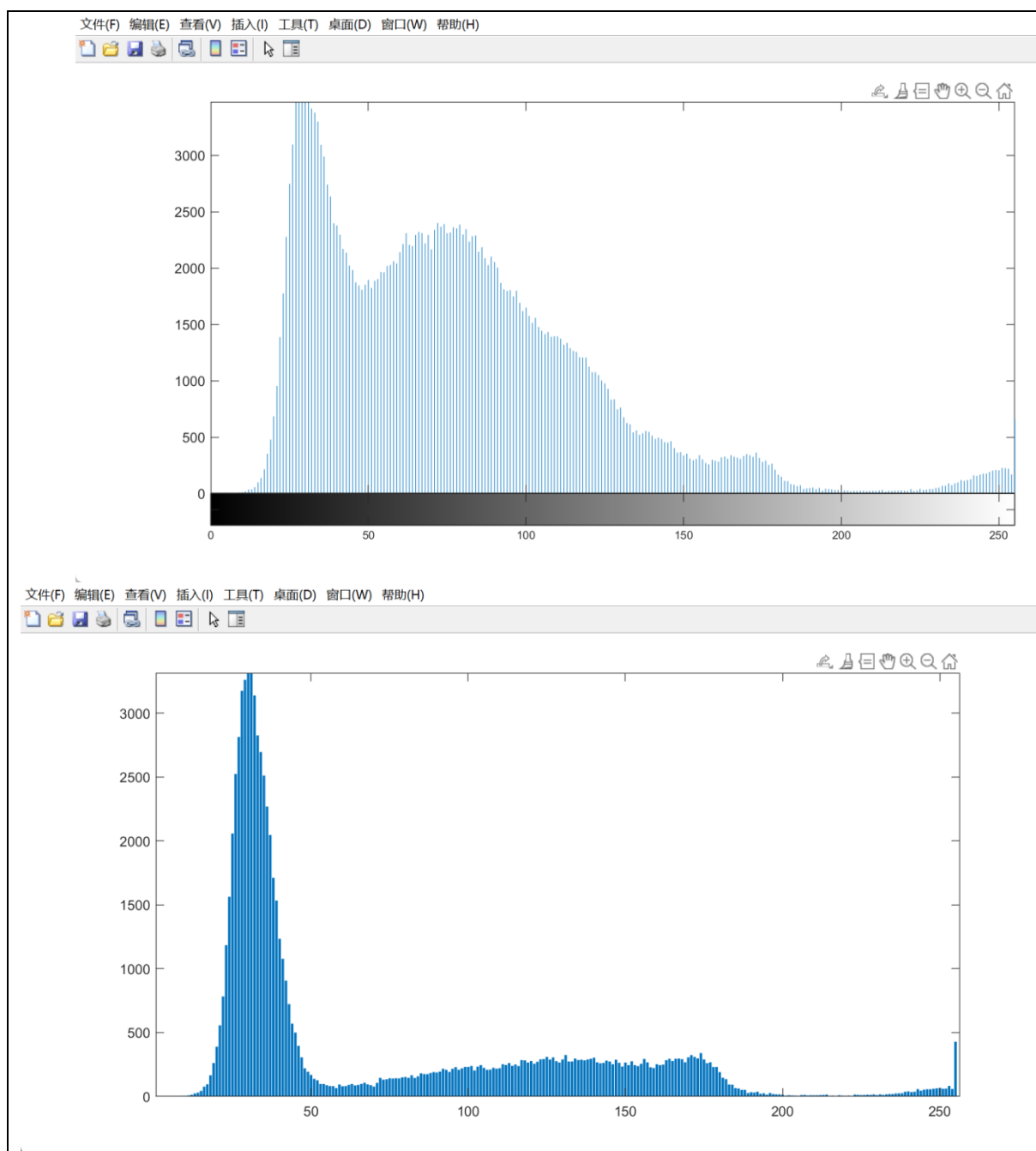
```
subplot(133),bar(counts),title('灰度直方图');
```

```
xlabel('灰度级');
```

```
ylabel('像素数');
```

请将运行结果贴在此处：





三、图像的几何运算

1. 编程实现图像的平移，旋转，缩放等几何变换，步骤如下：

(1) 用 `maketform` 函数定义变换参数，其调用格式如下：

`T= maketform('affine',A);`其中 `A` 为几何变换的矩阵。

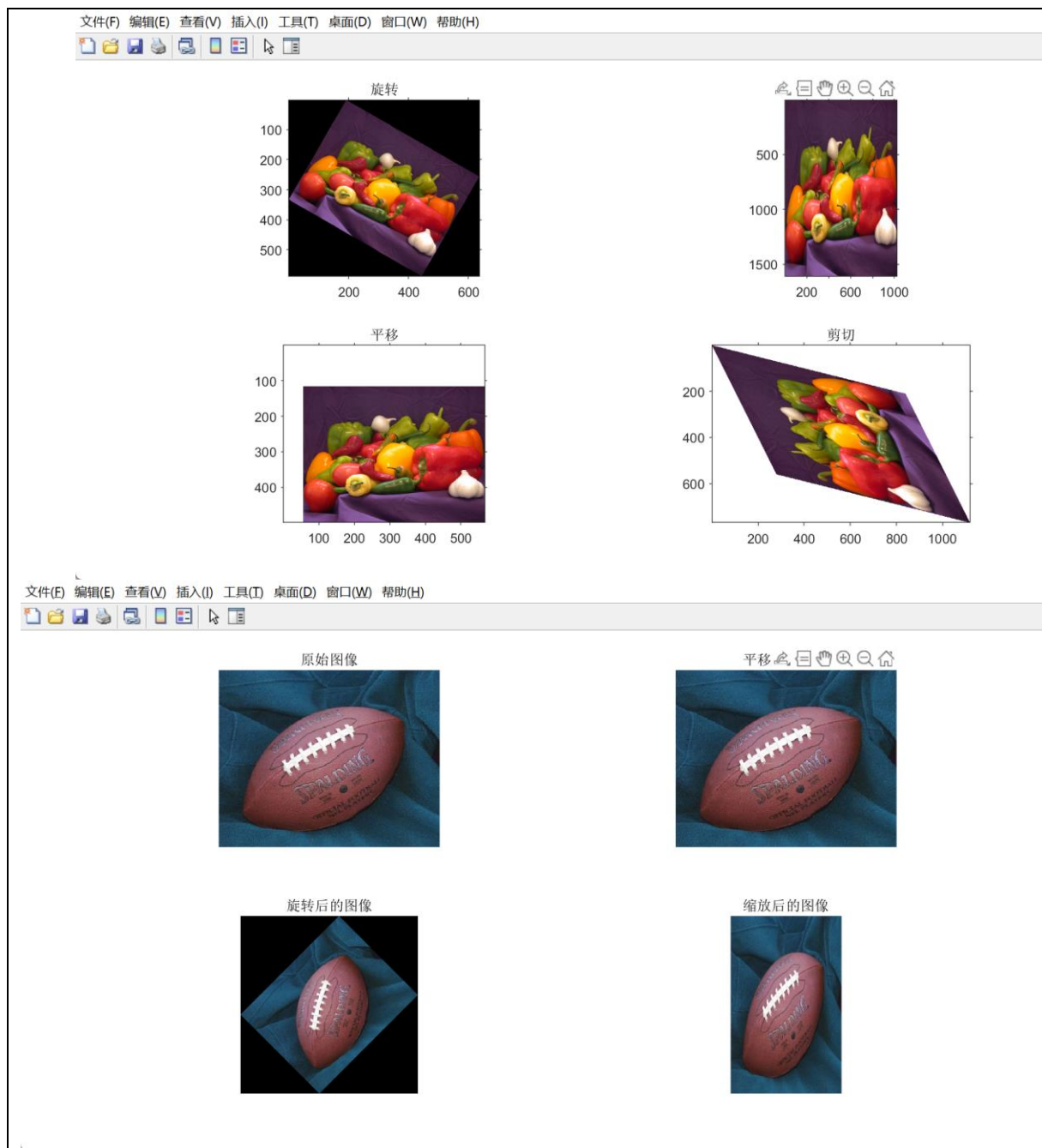
(2) 用函数 `imtransform` 进行图像的几何变换，其调用格式如下：

`B=imtransform(A,T);` `B` 为变换后的图像，`A` 为输入图像，`T` 为（1）中用

maketform 函数定义的变换参数结构体;

把程序和代码贴在下面:

请将实验代码贴在此处:
<pre>clear;clc;close all; %% 1.图像的平移, 旋转, 缩放等几何变换 clear;clc;close all; I = imread('football.jpg') %% 1.1 平移 % 定义平移变换矩阵 tform_translate = maketform('affine', [1 0 55; 0 1 115; 0 0 1]); I_translated = imtransform(I, tform_translate); %% 1.2 旋转 % 定义旋转变换矩阵, 顺时针旋转角度为 45 度 angle = 45; tform_rotate = maketform('affine', [cosd(angle) -sind(angle) 0; sind(angle) cosd(angle) 0; 0 0 1]); I_rotated = imtransform(I, tform_rotate); %% 1.3 缩放 % 定义缩放变换矩阵, 缩放比例 scale_x = 5; scale_y = 10; % [5 0 0;0 10 0;0 0 1] scale = 0.5; tform_scale = maketform('affine', [scale_x 0 0; 0 scale_y 0; 0 0 1]); I_scaled = imtransform(I, tform_scale); figure; subplot(2, 2, 1);imshow(I);title('原始图像'); subplot(2, 2, 2);imshow(I_translated);title('平移后的图像'); subplot(2, 2, 3);imshow(I_rotated);title('旋转后的图像'); subplot(2, 2, 4);imshow(I_scaled);title('缩放后的图像');</pre>
请将运行结果贴在此处:



2. 不用 IPT 提供的函数，自己编写函数实现图像的平移，转置，镜像等几何操作。

请将实验代码贴在此处：

```
%% 2.编写函数实现图像的平移，转置，镜像等几何操作
```

```
%% 2.1 平移
```

```
close all ;clear all ;clc ;
```

```
I = imread('football.jpg');
```

```
[H,W,Z] = size(I); % 获取图像大小，H 为垂直方向，W 为水平方向
```

```
I=im2double(I);%将图像类型转换成双精度
```

```

res = ones(H,W,Z); % 构造结果矩阵。每个像素点默认初始化为 1（白色）
delX = 50; % 平移量 X
delY = 100; % 平移量 Y

tras = [1 0 delX; 0 1 delY; 0 0 1]; % 平移的变换矩阵

for x0 = 1 : H %行
    for y0 = 1 : W %列
        temp = [x0; y0; 1];%将每一点的位置进行缓存，1 行 1 列，1 行 2 列...1 行 1024 列
        temp = tras * temp; % 根据算法进行，矩阵乘法：转换矩阵乘以原像素位置
        x1 = temp(1, 1);%新的像素 x1 位置，也就是新的行位置
        y1 = temp(2, 1);%新的像素 y1 位置,也就是新的列位置
        % 变换后的位置判断是否越界
        if (x1 <= H) & (y1 <= W) & (x1 >= 1) & (y1 >= 1)%新的行位置要小于新的列位置
            res1(x1,y1,:)= I(x0,y0,:);%进行图像平移，颜色赋值
        end
    end
end;

%% 2.2 转置
[H,W,Z] = size(I); % 获取图像大小，H 为垂直方向，W 为水平方向
I=im2double(I);%将图像类型转换成双精度
res = ones(H,W,Z); % 构造结果矩阵。每个像素点默认初始化为 1（白色）
tras = [0 1 0; 1 0 0; 0 0 1]; % 转置的变换矩阵
for x0 = 1 : H
    for y0 = 1 : W
        temp = [x0; y0; 1];%将每一点的位置进行缓存，1 行 1 列，1 行 2 列...1 行 1024 列
        temp = tras * temp; % 根据算法进行，矩阵乘法：转换矩阵乘以原像素位置
        x1 = temp(1, 1);%新的像素 x1 位置，也就是新的行位置（从 1~768）
        y1 = temp(2, 1);%新的像素 y1 位置,也就是新的列位置（从 1~1024）
        % 变换后的位置判断是否越界
        if (x1 <= H) & (y1 <= W) & (x1 >= 1) & (y1 >= 1)%新的行位置要小于新的列位置
            res2(x1,y1,:)= I(x0,y0,:);%进行图像颜色赋值
        end
    end
end;

%% 2.3 镜像
[H,W,Z] = size(I); % 获取图像大小，H 为垂直方向，W 为水平方向
I=im2double(I);%将图像类型转换成双精度
res = ones(H,W,Z); % 构造结果矩阵。每个像素点默认初始化为 1（白色）
tras = [1 0 0; 0 -1 W; 0 0 1]; % 水平镜像的变换矩阵
for x0 = 1 : H
    for y0 = 1 : W
        temp = [x0; y0; 1];%将每一点的位置进行缓存，1 行 1 列，1 行 2 列...1 行 1024 列

```

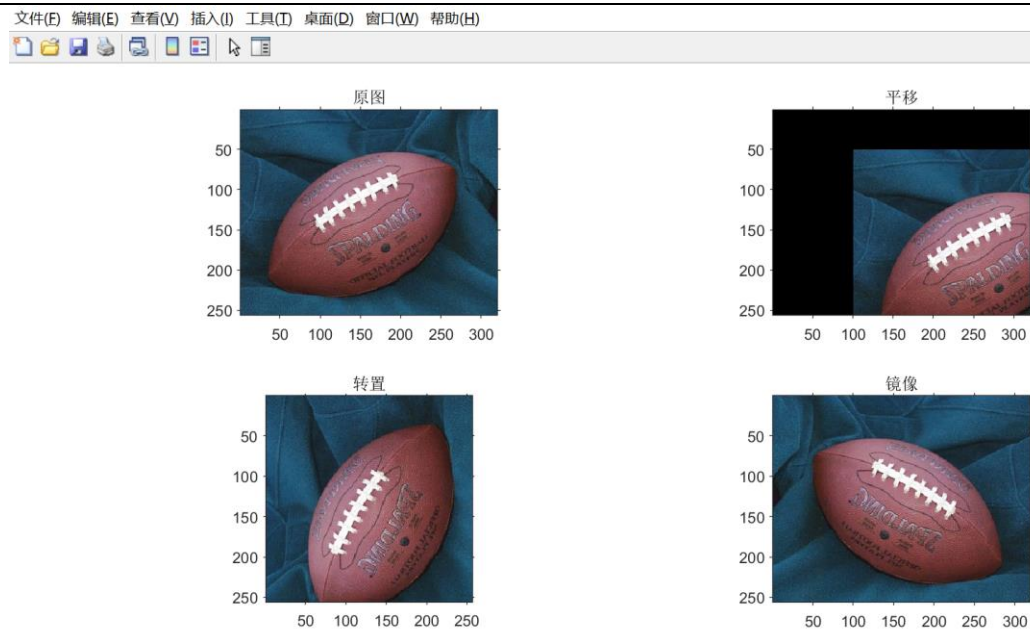


```

temp = tras * temp; % 根据算法进行，矩阵乘法：转换矩阵乘以原像素位置
x1 = temp(1, 1);%新的像素 x1 位置，也就是新的行位置
y1 = temp(2, 1);%新的像素 y1 位置，也就是新的列位置
% 变换后的位置判断是否越界
if (x1 <= H) & (y1 <= W) & (x1 >= 1) & (y1 >= 1)%新的行位置要小于新的列位置
    res3(x1,y1,:)= I(x0,y0,:);%进行图像颜色赋值
end
end
end;
set(0,'defaultFigurePosition',[100,100,1000,500]);%设置窗口大小
set(0,'defaultFigureColor',[1 1 1]);%设置窗口颜色
figure;%打开一个窗口，用来显示（多幅）图像
subplot(2,2,1), imshow(I),axis on ;title('原图');
subplot(2,2,2), imshow(res1),axis on;title('平移');
subplot(2,2,3), imshow(res2),axis on;title('转置');
subplot(2,2,4), imshow(res3),axis on;title('镜像');

```

请将运行结果贴在此处：



3. 实现最近邻插值和双线性插值算法。对 ‘lena.jpg’ 进行放大，采用两种插值方式，把结果显示在一个窗口中。

请将实验代码贴在此处：

```

%% 5 插值算法
% 双线性插值算法
% 载入图像
img = imread('football.jpg');

% 获取原图像大小

```

```

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

% 定义新的大小
newRows = 300; % 新的行数
newCols = 300; % 新的列数

% 初始化新图像
newImg = uint8(zeros(newRows, newCols, channels));

% 计算缩放比例
rowScale = rows / newRows;
colScale = cols / newCols;

% 双线性插值算法
for channel = 1:channels
    for i = 1:newRows
        for j = 1:newCols
            % 计算在原图中的坐标
            x = i * rowScale;
            y = j * colScale;

            % 计算周围的四个像素点
            x1 = floor(x);
            x2 = ceil(x);
            y1 = floor(y);
            y2 = ceil(y);

            % 确保坐标不超出图像边界
            x1 = max(x1, 1);
            x2 = min(x2, rows);
            y1 = max(y1, 1);
            y2 = min(y2, cols);

            % 双线性插值公式
            fa = double(img(x1, y1, channel)) * (x2 - x) + double(img(x2, y1,
channel)) * (x - x1);
            fb = double(img(x1, y2, channel)) * (x2 - x) + double(img(x2, y2,
channel)) * (x - x1);
            pixelValue = fa * (y2 - y) + fb * (y - y1);

            % 分配像素值到新图像
            if(pixelValue==0)
                newImg(i, j, channel) = img(x1,y1,channel);
            else
                newImg(i, j, channel) = pixelValue / ((x2 - x1) * (y2 - y1));

```

```

        end
    end
end
end

% 载入图像
img = imread('football.jpg'); % 替换 'path_to_your_image.jpg' 为你的图像文件路径

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

% 定义新图像的尺寸
newRows = 300; % 新的行数
newCols = 300; % 新的列数

% 初始化新图像
newImg1 = uint8(zeros(newRows, newCols, channels));

% 计算缩放比例
rowScale = rows / newRows;
colScale = cols / newCols;

% 最近邻插值算法
for channel = 1:channels
    for i = 1:newRows
        for j = 1:newCols
            % 计算在原图中的坐标
            x = round(i * rowScale);
            y = round(j * colScale);

            % 确保坐标不超出图像边界
            x = min(max(x, 1), rows);
            y = min(max(y, 1), cols);

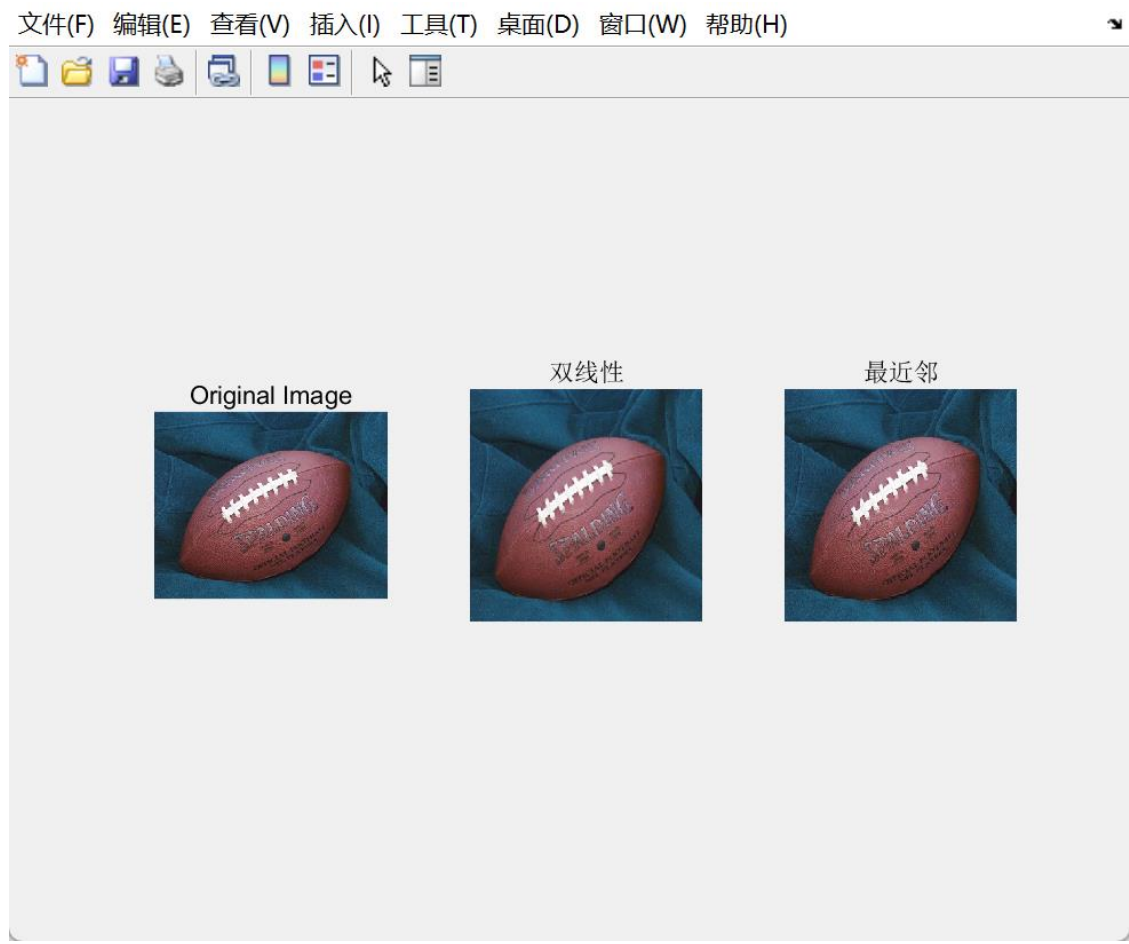
            % 直接使用最近的像素值
            newImg1(i, j, channel) = img(x, y, channel);
        end
    end
end

figure
subplot(131), imshow(img); title('Original Image');

```

```
subplot(132),imshow(newImg);title('双线性');  
subplot(133),imshow(newImg1);title('最近邻');
```

请将运行结果贴在此处：



注意：本实验报告要求直接将本 word 文档上传至课程平台