组　　号	
实验题目	图像处理的数学实验
队员姓名	1. 向飞宇
	2.
	3.
	4.

目　录

1 实验一：基于矩阵奇异值分解的数字水印嵌入算法

1.1 水印嵌入

以下是水印嵌入的算法描述.

- 导入一张待插入水印的图片，将其转化为灰度图片，选定插入水印的区域，记其构成的矩阵为 A_k；
- 对 A_k 进行奇异值分解，得 $A_k = USV^T$，S 为奇异值矩阵，U, V 为正交矩阵；
- 导入水印的图片，将其转化为灰度图片，记其构成的矩阵为 S；
- 选定强度因子 α，嵌入水印 $S' = S + \alpha W$；
- 对矩阵 S' 进行奇异值分解，得到 $S' = U_1 S_1 V_1^T$，用 $A'_K = US_1V^T$ 更新原始图像，完成水印嵌入.

根据以上的算法进行实验，设定待插入水印的图片与水印图片（图片均为彩色图片）如下：

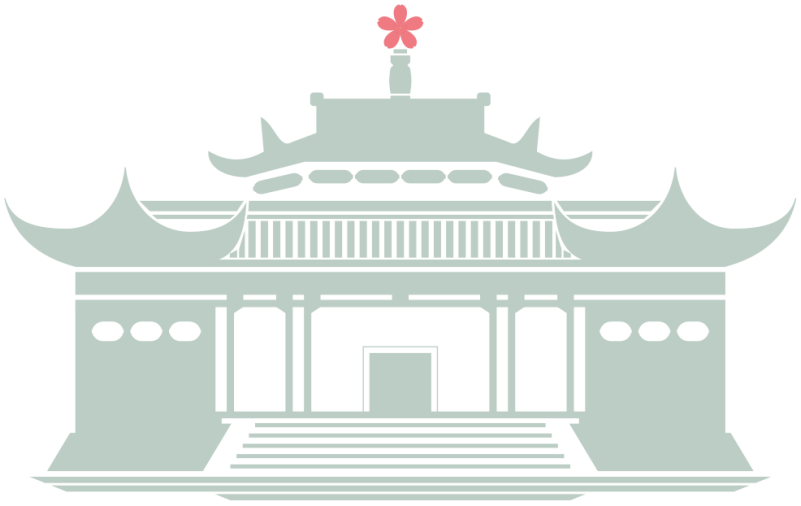

图 1 待插入水印图片（上）与水印图片（下）

在使用 MatLab 完成实验时，图片的导入使用 imread 实现，灰度图片转换使用 rgb2gray 实现，奇异值分解使用 svd 实现，选定待插入的部分为 $[341:348, 425:703]$ 像素区间.

除此之外，还需要对数据类型作一些处理. 对于 RGB 图像，每个像素点的范围为 $[0,1,\cdots,255]\times[0,1,\cdots,255]\times[0,1,\cdots,255]$，对于灰度图像，每个像素点的范围为 $[0,1,\cdots,255]$，rgb2gray 函数使用了灰度算法，数据类型仍然是 uint8，没有改变. 但是 svd 函数传入的参数必须是 double 类型，所以需要强制类型转换，但是在矩阵 A'_k 的计

算中很可能出现越界的情况，在利用 imshow 显示图片时，数组越界的图片将无法正常显示，所以需要将数据类型再次转换为 uint8 才可以正常显示图片.

最终得到的结果为:

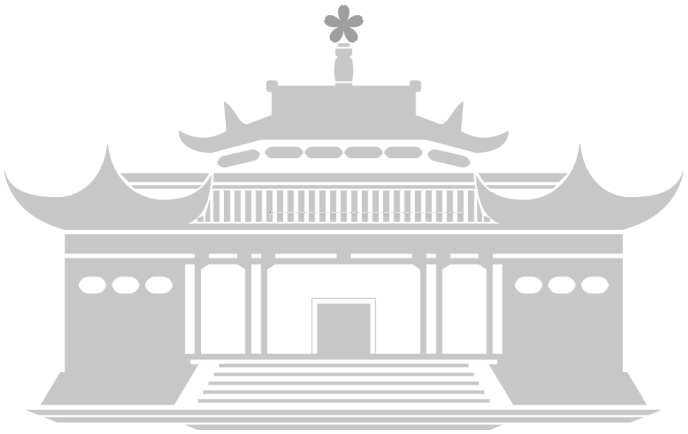

图 2 插入后水印图片

从得到的结果可以看出，水印图像很好地嵌入了原来的图像之中，对原图像的改变非常小.

1.2 水印提取

以下是水印提取的算法描述.

- 得到受扰动的图像矩阵 A'_k，对其进行奇异值分解 $A'_k = U^* S_1^* V^T$;
- 利用水印嵌入时的矩阵 U_1, V_1，得到 $S' = U_1 S_1^* V_1^T$;
- 利用水印嵌入时的矩阵 S 与强度因子 α，得到 $W^* = \frac{S'-S}{\alpha}$.

通过上面的算法，恢复水印的图片为:

图 3 恢复后水印图片

从以上的水印回复图片中可以看出，原来的水印得到了很好的提取效果.

1.3 对水印嵌入的分析

从以上程序的结果可以看出，对于本实验“水印”的目的在于，最大程度地不破坏原图像的信息，同时能够保证水印信息的嵌入和提取也是完整的. 所以水印图片嵌入的位置可能对最终生成的质量有所影响. 所以以下将对插入水印的不同位置及恢复情况进行实验.

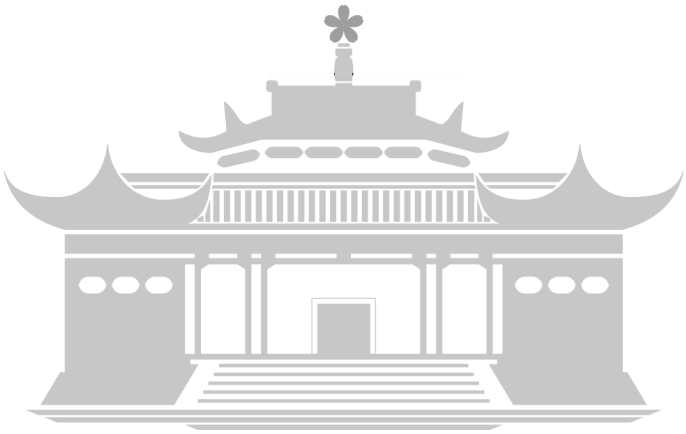

图 4 插入后水印图片，插入点为 $[141:148, 425:703]$

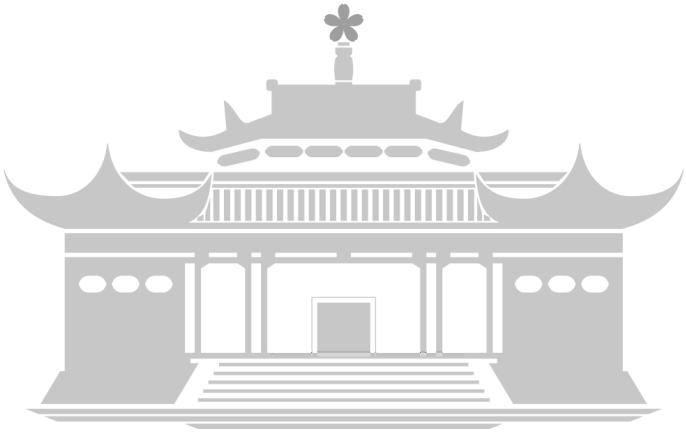

图 5 插入后水印图片，插入点为 $[541:548, 425:703]$

图 6　插入后水印图片，插入点为 $[741:748, 425:703]$

从以上的结果，结合嵌入部分的四张图像，可以发现四张图像的嵌入效果均比较好. 同时，由于插入了水印，原图像的灰度值在灰度突变的地方会有显著的变化（可能会出现纯黑或纯白的情况），但是从视觉效果上看，灰色加入黑色比加入白色**更为突出**. 所以需要尽量避免出现灰色加入黑色的水印图像. 其次，虽然一张水印图像不是满秩的，但是它理论上不会成为一张纯色图，所以对其进行奇异值分解后，对角元不会均为零，所以如果将一个水印置于一个纯色图上，得到的结果必然出现一条（或多条）纯黑（纯白）直线，这样对图片的整体效果也会有一定影响. 同时，水印的恢复效果均很好，此处不再赘述.

综上所述，水印图像插入的最佳情形应该在**灰度值较高的区域加入灰度值较低的水印图像，**这样对嵌入的效果最好.

现在再调整 α 的值，再观察实验的结果.

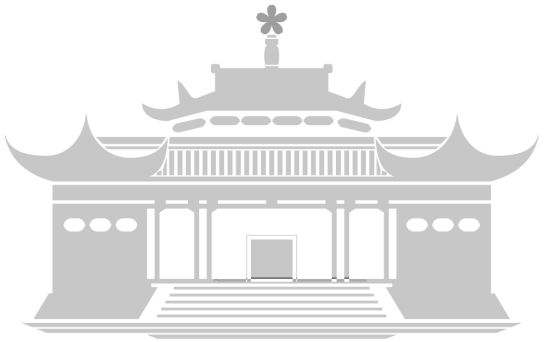

图 7　插入后水印图片， $\alpha = 0.3$

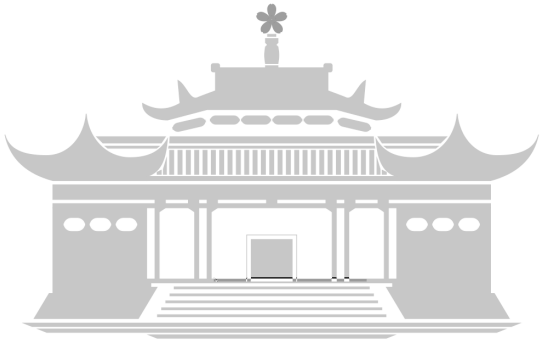

图 8 插入后水印图片，$\alpha = 0.5$

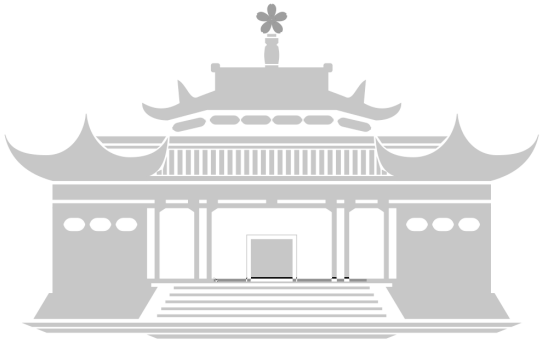

图 9 插入后水印图片，$\alpha = 0.7$

从以上的结果可以看出，α 越高，水印与外界环境对比度越高，所以使用 α 较低的值，可以得到更好的结果. 但是当 α 过小时，可能会出现水印图像无法恢复的情况，但是此题在 $\alpha = 10^{-9}$ 时仍然可以恢复水印图像，说明这个方法拥有很好的稳定性.

2 实验二：车牌方向校正与边框去除

问题描述：将下图中的车牌方位校正并去除边框.

首先，在完成这个实验的任务之前，我们需要对图片进行一些预处理. 首先，由于此图像非常像一张扫描的图像，所以直接对其进行灰度化，可以得到下图：

图 10　经过灰度处理的车牌

从灰度化之后的图片可以看出，此图像仍然具有一定的灰色阴影区域，这对接下来的步骤会造成一定影响，所以需要对图像进行二值化处理. 因为这个图像不具备明显的噪声，所以可以直接使用 Otsu 方法对其二值化，这个可以直接使用 MatLab 中的 graythresh 函数得到全局自适应阈值，然后使用 im2bw 函数对图像进行二值化，这样得到的结果为：

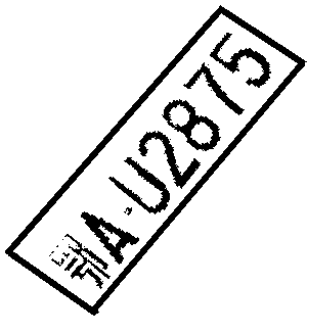

图 11　经过二值化处理的车牌

通过二值化处理，我们可以发现，图像中只剩下纯黑（0）和纯白（1）两种颜色. 然后处理图像的旋转校正部分，首先需要提取出图像的边界. 这个在 MatLab 中可以使用 edge 函数解决，由于图像已经为二值化图像，所以可以只用指定提取方法，而不需要加入附加参数，本文采用 Sobel 方法提取边界. 边界提取的结果为：

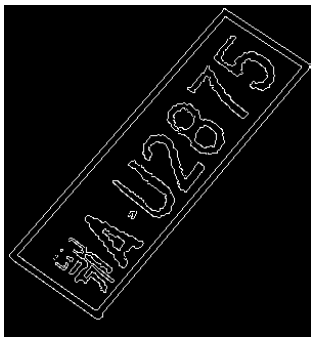

图 12　车牌的边界提取

当边界提取出来后，就可以作出一个外界矩形区域. 然后需要找出倾斜的角度，本文中采用了一种比较简单的方法. 直接寻找上边界上水平坐标最小的点与左边界上竖直坐标最小的点，由这两个点，给出一条直线，这条直线的斜率 k 作为图像顺时针旋转角度 α 的正切值，即 $k = \tan\alpha$. 然后根据旋转以后的图像再作一些微调，使得图像校正效果更好，最终得到的结果为：

图 13　车牌的旋转校正

由于旋转对像素点有改变，所以得到的图像像素点数目大于原来的图像. 然后我们需要去掉外围的黑色边框. 由于旋转时出现的多余像素点都会被填充为黑色，所以我们需要先处理这一部分的颜色问题. 由于黑色边框的连通性，所以可以使用 floodfill 算法来解决这个问题. 本实验中使用该算法的简单的一个实现方式——广度优先搜索. 这个算法的叙述如下：

- 首先需要寻找一个白色的点, 然后使用一个元胞数组实现一个队列. 使这个点入队，并将其颜色变为黑色；
- 如果队列非空，取队首元素，检查它的上下左右四个方向的四个点，如果某个点为黑色点，则不操作，如果为白色点，则将这个点放在队尾，同时将这个点置为黑色；
- 检查完队首元素后，将队首元素出队，转到第二步.

使用这个方式得到的图片没有了边框，但是出现了大量黑色部分，需要进行一次反色处理，使用 MatLab 中的 imcomplement 函数对其取反，得到的结果为：

图 14 floodfill 算法处理图片

此时再使用一次 floodfill 算法，将图像再次填充为黑色，再进行一次反色处理，这样就可以得到去除边框以后的图片了：

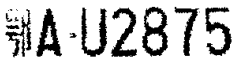

图 15 去除边框后的最终图片

至此，得到了最终的实验结果.

附录 A 实验一的代码

```
needed = imread('src.png');
needed = rgb2gray(needed);
water = imread('tgt.png');
water = rgb2gray(water);
alpha = 0.00000001;
[output, U1, V1, S] = waterprint(double(needed(541: 548, 425: 703)),
↪  double(water), alpha);
needed(541: 548, 425: 703) = uint8(output);

figure(1)
imshow(needed)
decode = de_waterprint(output, U1, V1, S, alpha);
figure(2)
imshow(uint8(decode))

function [output, U1, V1, S] = waterprint(src, tgt, alpha)
    [U, S, V] = svd(src);
    S0 = S + alpha * tgt;
    [U1, S1, V1] = svd(S0);
    output = U * S1 * V';
end

function output = de_waterprint(src, U1, V1, S, alpha)
    [~, S1, ~] = svd(src);
    S0 = U1 * S1 * V1';
    output = (S0 - S) ./ alpha;
end
```

附录 B 实验二的代码

```
car = imread('carpack.png');
gray = rgb2gray(car);
figure(1);
imshow(gray);
```

```
thres = imbinarize(gray, graythresh(gray));
figure(2);
imshow(thres);
edging = edge(thres, 'sobel');
figure(3);
imshow(edging);
% finding minimum bounding rectangle
box = find(edging == 1);
left = int16(box(1) / size(edging, 1));
udrange = mod(box - 1, size(edging, 1)) + 1;
up = min(udrange);
target1 = find(edging(:, left) == 1, 1);
target2 = find(edging(up, :) == 1, 1);
target_angle = double(target1 - left + 10) / double(target2 - up - 10);
angle = atan(target_angle);
rotation = imrotate(thres, -rad2deg(angle), 'bilinear');
figure(4);
imshow(rotation);
init = find(rotation(ceil(size(rotation, 1) / 2), :), 1);
flood = floodfill(rotation, [ceil(size(rotation, 1) / 2), init]);
negate = imcomplement(flood);
figure(5);
imshow(negate);
flood_2 = floodfill(negate, [1 1]);
final = imcomplement(flood_2);
figure(6)
imshow(final)

function output = floodfill(im, initial)
    if (im(initial(1), initial(2)) == 0)
        output = im;
    else
        list = {initial};
        im(initial(1), initial(2)) = 0;
        while ~isempty(list)
```

```
val = list{1};
if (val(1) > 1)
    left = [val(1) - 1, val(2)];
else
    left = [1, val(2)];
end
if (val(1) < size(im, 1))
    right = [val(1) + 1, val(2)];
else
    right = [size(im, 1), val(2)];
end
if (val(2) > 1)
    up = [val(1), val(2) - 1];
else
    up = [val(1), 1];
end
if (val(2) < size(im, 2))
    down = [val(1), val(2) + 1];
else
    down = [val(1), size(im, 2)];
end
if (im(left(1), left(2)) == 1)
    im(left(1), left(2)) = 0;
    list = [list, left];
end
if (im(right(1), right(2)) == 1)
    im(right(1), right(2)) = 0;
    list = [list, right];
end
if (im(up(1), up(2)) == 1)
    im(up(1), up(2)) = 0;
    list = [list, up];
end
if (im(down(1), down(2)) == 1)
    im(down(1), down(2)) = 0;
```

```
                list = [list, down];  %#ok<*AGROW>
            end
            list(1) = [];
        end
        output = im;
    end
end
```