目 录

目录

[题目名称： 图像信息伪装技术——图像隐写与置乱 1](#_Toc21127)

[1 系统概述 1](#_Toc32529)

[目标 1](#_Toc31238)

[功能概述 1](#_Toc286)

[背景知识： 1](#_Toc22238)

[1.图像隐写术 1](#_Toc23479)

[2.图像置乱 1](#_Toc9556)

[3.图片分割与融合 2](#_Toc16564)

[2 实验环境 3](#_Toc17375)

[3 系统设计与实现 3](#_Toc16555)

[置乱 3](#_Toc25807)

[分割 6](#_Toc24694)

[隐写 9](#_Toc20029)

[4 系统使用说明 12](#_Toc22502)

[5 项目总结 21](#_Toc8801)

[参考文献 22](#_Toc11065)

题目名称： 图像信息伪装技术——图像隐写与置乱

# 系统概述

目标**：**

为了实现图像信息伪装，本项目决定使用包括图像分割、隐写和图像置换的信息隐藏技术，首先对图片进行像素位置随机置乱方法进行图片置乱，将置乱后的图片分割打乱排序、然后使用4bit替代法，隐写到另外一张图片内，从而实现图像信息伪装。

功能概述**：**

对一张灰度图像进行了置乱处理并保存置乱图像与密钥，然后将置乱后的图像分割成四个部分，并且交换位置，随后作为要隐藏的内容，通过对载体图像（lenna图像）红色通道的像素位操作，把隐藏图像嵌入其中，最终生成了包含隐藏图像的新图像文件mix.bmp。

### **背景知识：**

### 1.图像隐写术

图像隐写术是一种将秘密信息隐藏在数字图像中的技术，目的是在不引起他人察觉的情况下传递隐秘信息。图像的低4bit，我们通常认为可以做隐藏信息的空间，也就是载体的冗余空间。所谓直接4bit替换法，就是直接用秘密图像像素值的高4bit去替换载体图像像素值的低4bit。

4bit 替代法属于空域图像隐写方法，其基本原理是利用图像像素值的低位比特（在这种情况下是最低的 4 位）来替换秘密信息对应的二进制位。例如，一幅灰度图像的像素值范围通常是 0 - 255（8 位表示），我们可以通过修改每个像素值的低 4 位来嵌入信息，这样对图像的视觉效果影响相对较小，人眼较难直接察觉出图像已经被嵌入了信息。

### 2.图像置乱

图像置乱主要是为了打乱图像的像素排列顺序或者改变图像的结构，使得图像在视觉上呈现出杂乱无章的效果，难以直接识别其内容。它常用于图像加密、隐私保护、数字水印等领域，作为一种预处理手段，增加图像信息的安全性，防止图像内容被未经授权的人轻易获取。代码中采用的是一种基于像素位置随机置换的置乱方法。首先将图像的像素按顺序（比如按列展开）整理成一个一维向量，然后通过 randperm 函数 （randperm 函数底层是依托于伪随机数生成器来实现，利用伪随机数生成器生成随机数序列，将随机数转换为索引值，去重与排序保障唯一性和随机性） 生成一个随机排列的整数序列作为置换的 “密钥”，按照这个密钥所指定的顺序对像素向量中的元素进行重新排列，之后再将重新排列后的像素向量转换回二维图像的形式，就实现了对图像像素位置的随机打乱，达到置乱效果。

这种方法简单直观，但置乱后的图像恢复需要准确保存和使用当时的置乱密钥（也就是那个随机排列的整数序列），否则很难还原出原始图像。

### 3.图片分割与融合

尺寸计算与均匀分割

在对图像进行分割时，首先需要获取图像的尺寸信息（通过 size 函数获取图像的行数、列数以及通道数），然后基于要分割的块数来计算每一块的大致宽度和高度。代码中是将图像均匀分割成 4 块，所以采用将图像的宽度和高度分别除以 2（向下取整，通过 floor 函数实现）的方式来确定每一块在水平和垂直方向上的尺寸，以此来保证分割的合理性和均匀性，便于后续进行有规律的操作和重组。

矩阵索引方式的分割操作

利用矩阵索引的方式来提取图像的不同区域，实现分割的目的。例如 topLeft = img(1:blockHeight, 1:blockWidth, :) 这样的语句，就是按照指定的行范围（从第 1 行到 blockHeight 行）、列范围（从第 1 列到 blockWidth 列）以及通道（用 : 表示包含所有通道，如果是彩色图像）来提取图像左上角的部分，其他如右上角、左下角、右下角区域的提取也是类似的原理，通过准确的矩阵索引定位到图像中的相应区域，把图像拆分成不同的块，这是基于矩阵操作在图像处理中常用的一种基本手段。

cat 函数的运用及图像拼接原理

代码中多次使用 cat 函数来实现图像块的拼接融合操作。cat 函数的基本原理是按照指定的维度将多个矩阵进行拼接。例如 cat(1, matrix1, matrix2) 表示在第一个维度（对于二维矩阵来说就是行维度）上把 matrix1 和 matrix2 拼接在一起，即把 matrix2 的行数据依次添加在 matrix1 的行数据后面形成一个新的矩阵；而 cat(2, matrix3, matrix4) 则是在第二个维度（列维度）上进行拼接，将 matrix4 的列数据依次添加在 matrix3 的列数据后面。在图像融合和拼接操作中，通过合理运用 cat 函数，按照特定的顺序和维度组合不同的图像块，就可以实现如对角交换位置融合或者恢复原始拼接顺序等不同的效果，改变或还原图像的整体视觉呈现。

图像变换与视觉效果改变

通过对图像进行分割后再以不同的顺序进行融合拼接，可以实现对图像视觉效果的改变。像代码中先进行对角交换位置融合的操作，就打乱了原始图像各个区域的布局，让图像呈现出一种不同的、经过变换后的样子，这种操作在图像特效制作、图像加密（简单的混淆效果层面）等领域有一定的应用场景；而后续的反向操作恢复原始拼接顺序，则是基于同样的分割和拼接原理，将经过变换的图像还原到最初的状态，展示了图像处理操作可逆性的一种体现，也说明了这些操作之间存在的内在逻辑关系以及在不同应用需求下的作用。

# 实验环境

MATLAB软件

# 系统设计与实现

在编写代码时，首先对代码进行了置乱操作，但是考虑到置乱后的结果容易引起注意，于是在置乱的基础上添加了隐写代码，以期实现隐蔽的效果。又考虑到所使用的置乱代码以及隐写代码都较为简单，为了在以上基础上实现对攻击者攻击难度的提升，于是又添加了对图像的分割和重组代码，这样当攻击者对载体图像进行攻击时，哪怕已经知道隐写和置乱算法，并且获取到置乱密钥时，仍然需要知道接收发送方事先约定的分割重排方式，才能实现对图片内容的获取。代码的伪装与提取的顺序为：置乱--分割重排列--隐写--提取隐写的图片--分割恢复成原排列--置乱的恢复。

置乱**：**

为了实现图像的置乱与恢复，本项目通过查阅资料，使用像素位置随机置乱法，核心思路是通过一个随机排列的整数矩阵（作为置乱密钥）对原图像像素的顺序进行打乱。恢复部分，我利用预先存储的密钥向量 rb，按照其规定的顺序对置乱后的图像像素数据进行重新排列，从而还原出原始图像的像素布局，最终实现图像的恢复并保存为可用的图像文件格式。其原理基于对图像矩阵的线性化处理（转换为单列矩阵）以及根据密钥进行的位置重映射操作。

#### 设计思路：

置乱：

首先，我确保通过 imread 函数读入的图像文件是200\*200图像文件olena.jpg ，并判断图像是否为彩色图像，是，则使用 rgb2gray 函数将其转换为灰度图像，这样后续处理就只针对图像的亮度信息，接着生成置乱密钥，先把读入并处理后的灰度图像 x 按列优先的顺序展开成一维向量 a（即 a = x(:)），获取图像所有像素值的一维排列形式。接着利用 randperm(numel(x)) 函数生成一个长度为 40000的随机排列整数向量 b，这个向量 b 就作为置乱的索引顺序，也就是置乱密钥的核心部分。通过循环 for i = 1:numel(x)，按照 b 所定义的随机顺序，将 a 中的像素值重新排列到新的向量 c 中（即 c(i) = a(b(i))），实现了对原图像像素顺序打乱的初步操作。

然后对置乱后的图像矩阵进行重塑，创建一个尺寸为 200×200 的全零矩阵 d，用于存放重塑后的置乱图像数据。利用两层嵌套循环（外层循环控制行数，内层循环控制列数），按照顺序从向量 c 中取出元素赋值给矩阵 d，具体通过语句 d(j, i) = c(k); k = k + 1; 实现，从而将一维的置乱像素向量 c 转换回与原图像相同尺寸的二维矩阵形式，完成图像置乱后的矩阵重塑。

置乱恢复：

首先通过 imread('getpart.bmp','bmp') 函数读入置乱后的图像文件，并将其存储为矩阵 rd。矩阵 rd 的元素代表了图像各个像素点的灰度值。然后使用 fopen('test.bin','rb') 打开存储密钥的二进制文件，以二进制读模式（'rb'）进行读取。通过 fread(fid, 40000,'double') 从打开的文件标识符 fid 对应的文件中读取 40000 个双精度类型的数据，存储为向量 rb，最后使用 fclose(fid) 关闭文件，释放相关资源。随后，生成单列矩阵：通过 rc = rd(:) 操作，将读入的二维图像矩阵 rd 转换为单列矩阵 rc，方便后续基于密钥进行重新排列。同时初始化一个同样大小（40000×1）的全零单列矩阵 ra，用于存储恢复后的图像数据。然后通过循环语句 for i = 1:40000 遍历所有的像素点，在循环中执行 ra(rb(i)) = rc(i)。通过这样的赋值操作，就按照密钥所规定的顺序，将置乱的像素值逐个还原到正确的位置上，实现了图像的恢复。最后初始化一个 200×200 的全零矩阵 x（通过两层嵌套的 for 循环（for i = 1:200 和 for j = 1:200），按照行列顺序将恢复后的单列矩阵 ra 中的像素值依次填充到矩阵 x 中，实现从单列矩阵到二维图像矩阵的转换，填充顺序保证了最终生成的 x 矩阵可以正确表示恢复后的图像。最后使用 uint8(x) 将矩阵 x 的数据类型转换为无符号 8 位整数类型，这符合常见的图像像素值的数据类型要求（例如灰度图像像素值范围通常是 0 - 255，用 8 位表示）。

#### 代码分析与公式推导：

置乱：

像素提取与随机排列映射：

首先检测原图像x的大小，将其修改为200\*200固定的px，并且确保是灰度图像，然后将图像x（已转换为灰度图像）按列优先的顺序展开成一维向量a，即a = x(:)。原图像尺寸假设为 200×200，展开后向量长度为 200×200 = 40000 个元素。

然后利用 randperm(40000) 函数生成一个从 1 到 40000 的随机排列整数向量 b，这个向量 b 就相当于置乱的索引顺序。

通过循环操作 for i = 1:40000，c(i) = a(b(i)); 实现了将原图像像素按照 b 所定义的随机顺序进行重新排列，c 即为置乱后的像素值一维向量。

重塑为二维矩阵：

后续将置乱后的一维像素向量 c 重新转换为与原图像相同尺寸（200×200）的二维矩阵 d。通过两层嵌套循环 for i = 1:200 和 for j = 1:200，按照顺序将 c 中的元素赋值给 d，实现了将单列矩阵 c 还原为二维矩阵的操作，即 d(j, i) = c(k); k = k + 1;，这里 k 作为索引用于遍历 c 向量。

保存密钥：

通过 fopen('test.bin', 'wb') 打开一个名为 test.bin 的二进制文件，以二进制写模式（'wb'）准备写入数据。然后使用 fwrite(fid, b, 'double') 将生成的作为密钥的随机排列向量 b 以双精度类型的数据格式写入文件中，最后通过 fclose(fid) 关闭文件，完成密钥的保存，以便后续利用该密钥恢复置乱的图像。

置乱恢复：

读入密钥和图像：

通过 imread('getpart.bmp','bmp') 函数读入置乱后的图像文件，并将其存储为矩阵 rd，使用 fopen('test.bin','rb') 打开存储密钥的二进制文件，以二进制读模式（'rb'）进行读取。

通过 fread(fid, 40000,'double') 从打开的文件标识符 fid 对应的文件中读取 40000 个双精度类型的数据，存储为向量 rb

生成单列矩阵：

通过 rc = rd(:) 操作，将读入的二维图像矩阵 rd 转换为单列矩阵 rc。这一操作实际上是将图像的像素值按照列优先的顺序依次排列成一个列向量，方便后续基于密钥进行重新排列。例如对于一个简单的 3×3 图像矩阵 [[a, b, c], [d, e, f], [g, h, i]]，经过 (:) 操作后会变成列向量 [a; d; g; b; e; h; c; f; i]。循环语句 for i = 1:40000 遍历所有的像素点，在循环中执行 ra(rb(i)) = rc(i)，其原理是根据密钥向量 rb 中的值来确定每个像素在恢复后的位置。rb(i) 的值表示第 i 个像素在恢复后的图像中应该处于的索引位置（索引从 1 开始计数），而 rc(i) 则是置乱图像中第 i 个像素的实际灰度值（或颜色值）。通过这样的赋值操作，就按照密钥所规定的顺序，将置乱的像素值逐个还原到正确的位置上，实现了图像的恢复。

将单列矩阵转换成与原图像同一尺寸的矩阵：

初始化一个 200×200 的全零矩阵 x。

通过两层嵌套的 for 循环（for i = 1:200 和 for j = 1:200），按照行列顺序将恢复后的单列矩阵 ra 中的像素值依次填充到矩阵 x 中，实现从单列矩阵到二维图像矩阵的转换，填充顺序保证了最终生成的 x 矩阵可以正确表示恢复后的图像。

**注：展示效果仅为测试图片，没有与其他代码的结果挂钩，完整结果在第四板块展出**

置乱效果：



提取效果：



分割**：**

接着，为了实现图片的分割打乱融合，本项目首先参考了文献，核心思路是通过特定分割与融合改变图像呈现分割与重组，并且依据同样的分割思路但反向拼接来还原图像到最初的状态，它们围绕图像的分割、拼接组合操作来实现相应的图像变换与恢复功能。

#### **设计思路：**

分割与重组：

在使用imread函数读入图片后，使用size**函数获取图像尺寸信息，为了将图像均匀分割成 4 块，代码通过 blockWidth = floor(width / 2); 和 blockHeight = floor(height / 2); 来计算每一块大致的宽度和高度。floor 函数的作用是向下取整，确保分割出来的块在尺寸上是整数，接着使用cat函数对图片进行重新拼接，**cat 函数在 MATLAB 中用于沿着指定维度拼接数组（对于图像来说就是拼接图像的不同部分），其基本语法是 cat(dim, A, B,...)，其中 dim 表示要拼接的维度（1 通常代表行维度，2 代表列维度等），A、B 等是要进行拼接的数组（在这里就是图像分割后的不同区域块）。

分割后的恢复：

首先通过 imread('gethide.bmp') 函数读取之前经过交换对角位置处理后的图像文件，并将其存储为 processedImg 矩阵形式，并且获取该图像的尺寸信息，随后根据获取到的图像整体宽度和高度，使用 blockWidth = floor(width / 2) 和 blockHeight = floor(height / 2) 分别计算出将图像均匀分割成 4 块时每一块在水平和垂直方向上的大致宽度和高度。运用矩阵索引的方式从 processedImg 中提取出 4 个不同的区域块，按照与之前交换对角位置操作相反的顺序来重新拼接这 4 个图像块，从而还原出原始图像的拼接状态。具体是通过 cat 函数来实现拼接。

#### 代码分析与公式推导：

分割：

代码通过 blockWidth = floor(width / 2); 和 blockHeight = floor(height / 2); 来计算每一块大致的宽度和高度

**topLeft = img(1:blockHeight, 1:blockWidth, :);：这是提取图像左上角部分，它从图像的第一行到 blockHeight 行、第一列到 blockWidth 列的区域进行提取，对于彩色图像，第三个 : 表示提取所有通道的数据，也就是完整地取出了图像左上角的那一块区域。**

topRight = img(1:blockHeight, blockWidth + 1 : end, :);：提取图像右上角部分，行范围同样是从第一行到 blockHeight 行，但列范围是从 blockWidth + 1 列到图像的最后一列，以此获取右上角的区域。

bottomLeft = img(blockHeight + 1 : end, 1:blockWidth, :);：提取图像左下角部分，行范围是从 blockHeight + 1 行到图像的最后一行，列范围是从第一列到 blockWidth 列，得到左下角的区域。

bottomRight = img(blockHeight + 1 : end, blockWidth + 1 : end, :);：提取图像右下角部分，行和列范围分别是从 blockHeight + 1 行到最后一行以及从 blockWidth + 1 列到最后一列，从而获取右下角区域。

通过这样的操作，就把一幅完整的图像分割成了四个大小相等（在图像尺寸能被均匀分割的情况下）的子区域，方便后续进行重新组合等变换操作。

代码中 fusedImg = cat(2, cat(1, topRight, topLeft), cat(1, bottomRight, bottomLeft)); 这一行实现了图像的融合与视觉呈现改变。

首先看内层的cat函数：

cat(1, topRight, topLeft)：这里 dim 为 1，表示沿着行维度（垂直方向）进行拼接，将右上角部分 topRight 和左上角部分 topLeft 按行拼接在一起，形成一个新的图像区域块，相当于把这两个部分在垂直方向上合并成了一个更大的区域，同理，cat(1, bottomRight, bottomLeft) 也是沿着行维度将右下角部分 bottomRight 和左下角部分 bottomLeft 在垂直方向上合并成了另一个大的图像区域块。

然后是外层的 cat 函数：

cat(2, cat(1, topRight, topLeft), cat(1, bottomRight, bottomLeft)) 中，外层的 dim 为 2，表示沿着列维度（水平方向）进行拼接。它把前面通过按行拼接得到的两个区域块（一个是 topRight 与 topLeft 拼接后的，另一个是 bottomRight 与 bottomLeft 拼接后的）在水平方向上拼接在一起，最终形成了完整的融合图像 fusedImg。

分割后的恢复：

把一幅完整的图像分割成了四个大小相等的部分在分割部分已经说明，不再赘述。

反向操作恢复：

首先，内部的 cat(1, bottomLeftRecover, bottomRightRecover) 操作，是依据矩阵拼接原理，在维度 1（对应图像的行方向）上把左下角的块 bottomLeftRecover 和右下角的块 bottomRightRecover 拼接在一起，形成一个新的下半部分的图像块。从直观理解上，就是将原本交换到对角位置的这两块恢复到它们在原始图像中应处的下半部分位置关系。

同样，cat(1, topLeftRecover, topRightRecover) 是在维度 1（行方向）上把左上角的块 topLeftRecover 和右上角的块 topRightRecover 拼接起来，构成一个新的上半部分的图像块，这也是将原本交换位置的两块还原到原始图像上半部分的正确位置关系。

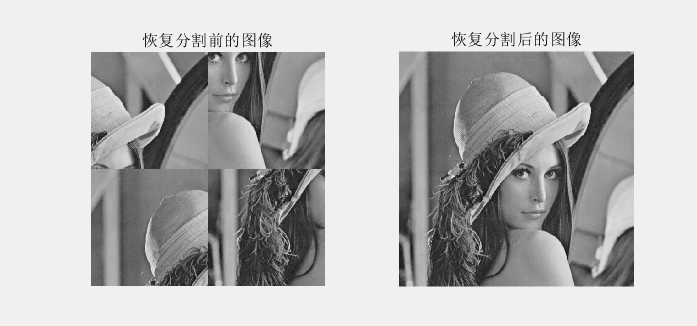
最后，外层的 cat(2, cat(1, bottomLeftRecover, bottomRightRecover), cat(1, topLeftRecover, topRightRecover)) 操作，是在维度 2（对应图像的列方向）上，将前面生成的新的下半部分图像块和上半部分图像块拼接在一起。通过这样先在行方向拼接好上下部分，再在列方向拼接上下部分的操作，就按照与之前对角交换位置相反的顺序，完成了对 4 个图像块的重新组合，得到了恢复原始拼接顺序后的图像 recoveredImg。

**注：展示效果仅为测试图片，没有与其他代码的结果挂钩，完整结果在第四板块展出**

分割效果：



还原效果：



隐写**：**

本项目针对图片隐写进行了代码的编写，首先参考了老师的ppt以及《信息隐藏技术实验教程》、

进行代码的编写测试，实现了对图片的隐写以及对隐写图片的提取。

为了找到隐秘的空间, 也就是载体的冗余空间，我通过对查找到的资料进行分析后发现，当图像矩阵的每个像素值都去掉了低4bit（如第（1,1)像素点），但改变后的图像和原图像在给人的视觉上并没有太大的变化，因此可以认为低4bit是冗余空间，可以改变，这是可以进行图像嵌入伪装的理论基础。核心思路是将载体图像r层的低4bit位置为0，将秘密图像高4bit右移后隐藏在载体图像r层低4bit，提取时，先提取出载体图像的r层，然后提取其低4bit位，将提取出的bit位左移4位，得到秘密图像。

#### 设计思路：

隐写：

首先选取了大小为200\*200px的两张图像作为载体图片和隐写图片，然后对其进行读取，提取原载体图片信息进行分层后，得到原来的载体图像的R层，并且置载体图像R层的低4bit和隐写图像的低4bit为0，后将隐写图像的高4bit右移4位，而后使用bitor进行图片隐藏，回写保存到mix.bmp内。

提取：

经过信息隐藏的图像文件中读取并获取其 R 层数据，因为秘密图像信息是隐藏在该层中。然后利用按位与运算提取出 R 层数据中隐藏图像信息所在的低 4 位数据，接着通过左移 4 位操作对提取的信息进行位置还原，尽量恢复隐藏图像的样子，尽管低 4 位信息由于之前隐藏操作有所缺失但仍能获取到关键的图像轮廓等内容，最后将还原后的隐藏图像信息保存为独立的图像文件

#### 代码分析与公式推导：

隐写：

代码直接将隐写图像的高 4bit 隐藏在载体图像的 R, G, B 层中所选的那一层的低 4bit, 此时，载体图像的大小大于等于秘密图像的大小, 且隐写图像是二值或灰度图像。

位运算基础

在图像信息隐藏过程中，主要运用了位与（bitand）、位或（bitor）和位右移（bitshift）等位运算操作，以下是相关的原理及推导说明。

位与运算（bitand） - 清除低 n 位：

在代码中，例如 cover1 = bitand(cover1, 240); 和 takemsg4 = bitand(msg, 240); 这两处使用了位与运算来清除图像像素值的低 4 位。

对于一个无符号 8 位整数（取值范围是 0 到 255），其二进制表示形式为 b7b6b5b4b3b2b1b0（b 表示二进制位）。想要清除低 4 位，即让 b3b2b1b0 都变为 0，可以将其与二进制数 11110000（十进制为 240）进行位与运算。这样就实现了将低 4 位清零的效果，只保留了高 4 位信息。

位右移运算（bitshift） - 提取并移位高 n 位：

在代码 shiftmsg4 = bitshift(takemsg4, -4); 中使用了位右移操作。

对于之前已经清除低 4 位后的像素值（假设其为 y = b\_7b\_6b\_5b\_40000），进行右移 4 位操作，其目的是将高 4 位移动到低 4 位的位置，方便后续嵌入到载体图像中，此时y=0000b\_7b\_6b\_5b\_4。

位或运算（bitor） - 嵌入信息：

在代码 cover1 = bitor(cover1, shiftmsg4); 中，使用位或运算将经过处理的秘密图像信息（已右移高 4 位后的结果）嵌入到载体图像（清除低 4 位后的 R 层图像）中。

设载体图像清除低 4 位后的像素值为 A = a\_7a\_6a\_5a\_40000，秘密图像右移 4 位后的像素值为 B = 0000b\_7b\_6b\_5b\_4，进行位或运算后结果如下：a\_7a\_6a\_5a\_4 b\_7b\_6b\_5b\_4

提取：

获取 R 层信息：

使用 [row, col] = size(data) 语句获取图像的行数和列数信息，然后通过 A = data(:,:,1) 提取图像的红色（R）通道层的数据，存储为 A 矩阵。这一步操作与之前在信息隐藏时聚焦于载体图像 R 层进行操作相呼应，因为秘密图像信息是隐藏在该层中，所以提取 R 层信息是后续准确提取隐藏图像的基础。

提取隐藏图像的低 4 位信息

要从隐写后的 R 层图像中提取出之前隐藏的秘密图像信息，首先需要获取其低 4 位数据。这里利用 bitand 函数执行 A = bitand(A, 15) 操作，其原理是基于按位与运算的特性。数字 15 的二进制表示为 00001111，当 A（隐写后 R 层图像数据每个像素对应的二进制表示）与 15 进行按位与操作时，根据按位与运算规则（对应位都为 1 时结果位才为 1，否则为 0），就能够只保留 A 中每个像素值的低 4 位信息，而将高 4 位清零，从而提取出之前嵌入在 R 层低 4 位中的秘密图像信息（经过之前隐藏操作时的处理，秘密图像的关键信息就在这里）。

还原隐藏图像信息（左移 4 位）

由于之前提取到的只是隐藏图像信息的右移 4 位后的低 4 位数据，为了尽量还原出原始隐藏图像的样子，需要将这低 4 位信息再左移 4 位。使用 bitshift 函数执行 A = bitshift(A, 4) 操作，其原理是将每个像素值对应的二进制表示向左移动 4 位，这样原本处于低 4 位的信息就会回到相对高位的位置，实现对隐藏图像信息在位置上的一种还原操作。不过需要注意的是，之前在隐藏图像时对秘密图像的低 4 位进行了清零处理，所以这部分信息已经丢失，此时还原出的图像只是基于之前保留的高 4 位信息进行的部分还原，低 4 位是受到影响的（通常为 0 或者不准确的值）。

**注：展示效果仅为测试图片，没有与其他代码的结果挂钩，完整结果在第四板块展出**

隐藏效果：



提取结果：



# 系统使用说明

首先，将图像olena.jpg进行转换为灰度图片，然后实现像素位置随机置乱

clear

clc

% 读入原图像

x = imread('olena.jpg');

% 判断图像是否为彩色图像，如果是则转换为灰度图像，若已是灰度图像则保持不变

if size(x, 3) == 3

x = rgb2gray(x);

end

% 获取图像尺寸信息

[height, width] = size(x);

% 判断图像尺寸是否不是200\*200，若不是则进行尺寸调整

if height ~= 200 || width ~= 200

% 使用imresize函数将图像调整为200\*200大小，这里采用双线性插值方法 x = imresize(x, [200 200], 'bilinear');

end

subplot(1, 2, 1);

imshow(x);

title('原图像');

% 生成随机排列整数矩阵，作为置乱密钥

a = x(:);

b = randperm(numel(x)); % 根据调整后图像元素数量生成随机排列

c = zeros(numel(x), 1);

for i = 1:numel(x)

c(i) = a(b(i));

end

% 将单列矩阵转换成与原图像同一尺寸的矩阵

d = zeros(200);

k = 1;

for i = 1:200

for j = 1:200

d(j, i) = c(k);

k = k + 1;

end

end

subplot(1, 2, 2);

imshow(uint8(d));

title('置乱后的图像');

% 保存置乱后的图像

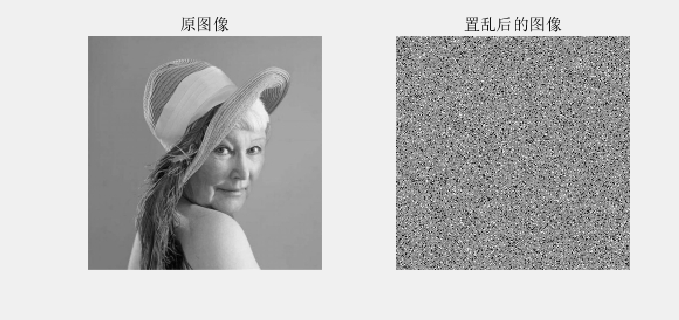
imwrite(uint8(d),'mess.bmp', 'bmp');

% 保存密钥

fid = fopen('test.bin', 'wb');

fwrite(fid, b, 'double');

fclose(fid);



然后对置乱后的图片mess.bmp进行分割，将分割后的四个部分进行重新排列，保存为part.bmp

% 读取图像

img = imread('mess.bmp');

subplot(1,2,1);

imshow('mess.bmp');

title('置乱后的图像');

% 获取图像的尺寸信息，[行 列 通道数]

[height, width, ~] = size(img);

% 计算每一块的大致宽度和高度（这里假设图像能均匀分割成4块）

blockWidth = floor(width / 2);

blockHeight = floor(height / 2);

% 分割图像为4块

topLeft = img(1:blockHeight, 1:blockWidth, :);

topRight = img(1:blockHeight, blockWidth + 1 : end, :);

bottomLeft = img(blockHeight + 1 : end, 1:blockWidth, :);

bottomRight = img(blockHeight + 1 : end, blockWidth + 1 : end, :);

% 进行对角交换位置融合，实现改变原始图像视觉呈现

fusedImg = cat(2, cat(1, topRight, topLeft), cat(1, bottomRight, bottomLeft));

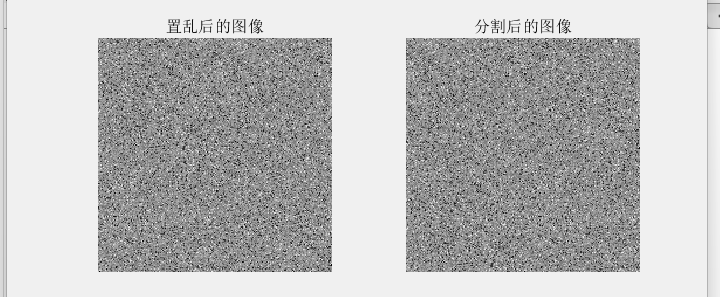
% 保存融合后的图像

imwrite(fusedImg, 'part.bmp');

subplot(1,2,2);

imshow('part.bmp');

title('分割后的图像');



然后再对分割后的图像进行图片隐写，读取part.bmp，将隐写后的图片保存为mix.bmp

%提取lena512和lenna图像，

cover = imread('lenna.jpg');

data=cover;

subplot(4,2,1);

imshow(cover);

title('lenna图像');

msg=imread('part.bmp' );

subplot(4,2,2);

imshow(msg);

title('隐藏图像');

[row,col]=size(cover);

cover1=cover(:,:,1);

subplot(4,2,3);

imshow(cover1);

title('lenna红（R）层图像');

%置载体图像R层的低4bit为0

cover1=bitand(cover1,240);

subplot(4,2,4);

imshow(cover1);

title('去掉低4位的lenna的R层图像');

%置秘密图像的低4bit为0

takemsg4=bitand(msg,240);

subplot(4,2,5);

imshow(takemsg4);

title('去掉低4位的隐藏图像');

%将秘密图像的高4bit右移4位

shiftmsg4=bitshift(takemsg4,-4);

subplot(4,2,6);

imshow(shiftmsg4);

title('隐藏图像高4bit右移4位');

%图像隐藏

cover1=bitor(cover1,shiftmsg4);

subplot(4,2,7);

imshow(cover1);

title('隐藏图片隐藏在lenna的R层后的图像');

%写回并保存

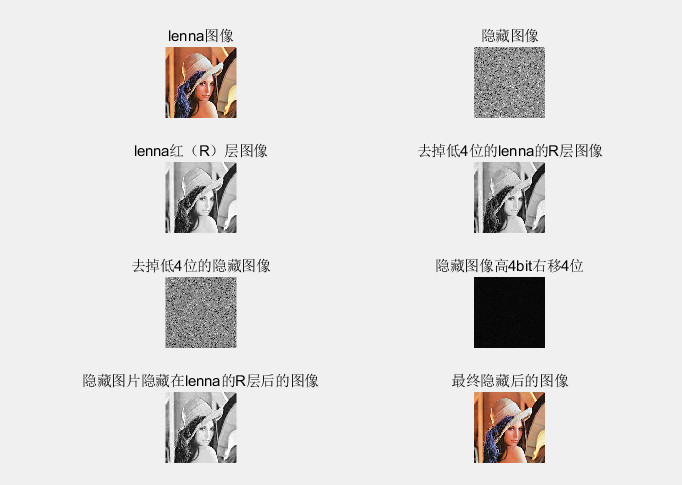
data(:,:,1) = cover1;

imwrite(data, 'mix.bmp');

subplot(4,2,8);

imshow(data);

title('最终隐藏后的图像');



接着我们使用gethide.m对隐写后的图片mix.bmp进行提取，并且保存为gethide.bmp

%提取秘密图像信息

data=imread('mix.bmp');

[row,col]=size(data);

A=data(:,:,1);

subplot(4,1,1);

imshow(A);

title('隐写图片后lenna的R层后的图像');

%提取隐蔽图像的低4bit

A=bitand(A,15);

subplot(4,1,2);

imshow(A);

title('隐藏图像高4bit右移4位后的图像');

%将提取的秘密图像的低4bit左移4位

A=bitshift(A,4);

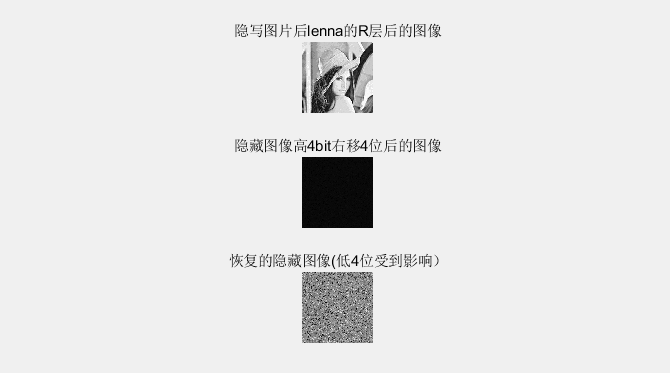
subplot(4,1,3);

imshow(A);

title('恢复的隐藏图像(低4位受到影响）');

% 保存提取出的隐藏图像，可以根据需要指定文件名，这里示例保存为 'extracted\_msg.bmp'

imwrite( A, 'gethide.bmp');



然后对提取出的隐藏图像进行重新分割，并且将分割后的四部分恢复成原有的排序

% 读取之前处理后（交换对角位置后）的图像

processedImg = imread('gethide.bmp');

subplot(1,2,1);

imshow('gethide.bmp');

title('恢复分割前的图像');

% 获取图像的尺寸信息，[行 列 通道数]

[height, width, ~] = size(processedImg);

% 计算每一块的大致宽度和高度（这里假设图像能均匀分割成4块）

blockWidth = floor(width / 2);

blockHeight = floor(height / 2);

% 分割处理后的图像为4块

topLeftRecover = processedImg(1:blockHeight, 1:blockWidth, :);

topRightRecover = processedImg(1:blockHeight, blockWidth + 1 : end, :);

bottomLeftRecover = processedImg(blockHeight + 1 : end, 1:blockWidth, :);

bottomRightRecover = processedImg(blockHeight + 1 : end, blockWidth + 1 : end, :);

% 进行反向操作，恢复原始图像的拼接顺序

recoveredImg = cat(2, cat(1, bottomLeftRecover, bottomRightRecover), cat(1, topLeftRecover, topRightRecover));

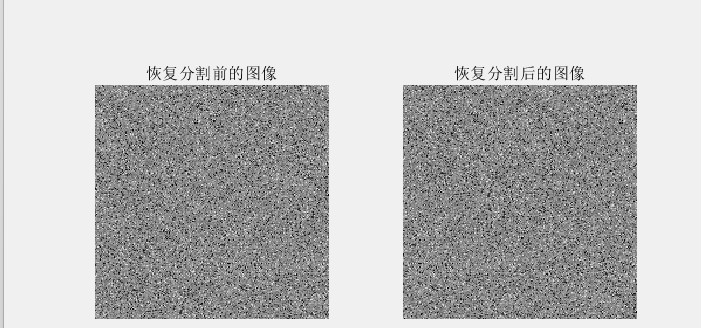
% 保存最终恢复后的图像，可以根据需要指定文件名

imwrite(recoveredImg, 'getpart.bmp');

subplot(1,2,2);

imshow('getpart.bmp');

title('恢复分割后的图像');



最后将恢复顺序的置乱图片进行置乱的恢复，保存为图片getmess.bmp

clear

clc

%读入置乱后的图像

rd = imread('getpart.bmp','bmp');

subplot(1,2,1);

imshow(rd);

title('置乱后的图像');

%读入密钥

fid = fopen('test.bin','rb');

rb = fread(fid,40000,'double');

fclose(fid);

%生成单列矩阵

rc = rd(:);

ra = zeros(40000,1);

%恢复，用密钥重新排列置乱的矩阵

for i = 1:40000

ra(rb(i)) = rc(i);

end

%将单列矩阵转换成与原图像同一尺寸的矩阵

x = zeros(200);

k =1;

for i = 1:200

for j = 1:200

x(j,i) = ra(k);

k = k+1;

end

end

x = uint8(x);

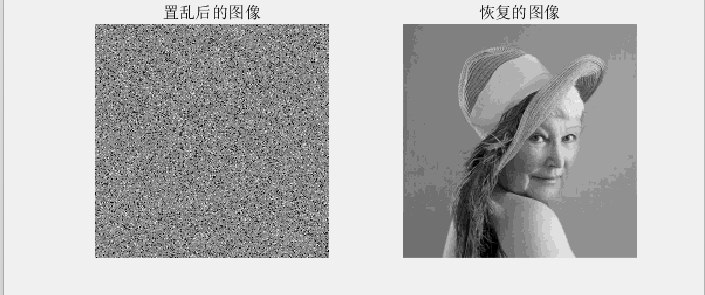
subplot(1,2,2);

imshow(x);

title('恢复的图像');

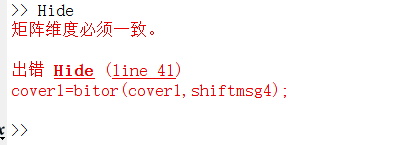
% 保存提取出的隐藏图像，可以根据需要指定文件名，这里示例保存为 'extracted\_msg.bmp'

imwrite( x, 'getmess.bmp');



# 项目总结

1.在进行图片隐写操作时，显示报错，检查后发现是进行隐写的图片和载体图像大小不一致，将隐写图片的px值修改后，解决报错。

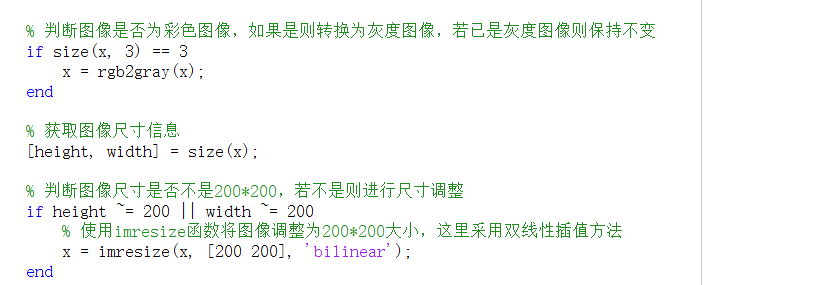


2.测试代码时（后面编写实际项目时，将测试图片换成了置乱图片，置乱图片本身已经是灰度图像），在进行图片隐写过程中，写回并保存一直出错，经排查发现，导入的被隐藏图片是彩色图像，属于三维图像，本身就是三维数组，在进行置秘密图像的低4bit为0和将秘密图像的高4bit右移4位操作时，并没有改变其本身的三维图像特性，导致在图像隐藏cover1=bitor(cover1,shiftmsg4);时，将cover1也变成了三维图像

解决方法：将彩色图像msg使用以下命令转为灰度图像：msg = rgb2gray(msg);



3.在进行图片置乱时，添加了以下代码，将置乱图片的大小剪切，并且固定为200\*200px，同时确保其始终为灰度图像，提高代码容错率，此时可以导入更多类型的图片。



由于在对图像使用了4bit替代法进行隐写，对于秘密图像msg，先将低 4 位清零（takemsg4 = bitand(msg,240)），然后将其高 4 位右移 4 位（shiftmsg4 = bitshift(takemsg4,-4)），最后将右移后的秘密图像信息嵌入到载体图像cover1的低 4 位（cover1 = bitor(cover1,shiftmsg4)）。

在恢复过程中，从隐写后的图像中提取R层（A = data(:,:,1)），然后提取其低 4 位（A = bitand(A,15)），这一步是为了获取之前嵌入的秘密图像信息。但是由于之前在隐藏图像时对秘密图像的低 4 位进行了清零操作，所以这部分信息已经丢失。当把提取的低 4 位左移 4 位（A = bitshift(A,4)）时，恢复出的图像缺少了原始秘密图像的低 4 位信息，因此完整图像不能完全恢复，会受到一定影响。

参考文献

《信息隐藏技术实验教程》

《数字图像置乱加密解密技术的研究与实现》

《MATLAB的cat()函数》CSDN

《真随机数和伪随机数以及伪随机数的MATLAB实现》 CSDN