

武汉大学国家网络安全学院

信息隐藏实验报告

学号	2021302141097
姓名	李宇
实验名称	基本图像信息隐藏方法实践
指导教师	王丽娜，任延珍

一、实验名称：基本图像信息隐藏方法实践

二、实验目的：

- 1) 熟悉并掌握 Matlab 的基本使用、函数编写，能够灵活使用 matlab 完成特定任务
- 2) 学习图像的基本格式，并明确不同类型格式之间的特点和差异
- 3) 掌握基本信息隐藏方法，并能够使用编程进行实现
- 4) 掌握 DCT 等基本图像转换方法，借助图像在频域上的特点进行信息隐藏
- 5) 掌握二值图像的隐写算法，并能对其中的细节进行处理
- 6) 了解算法鲁棒性的概念，并通过调整不同超参数体会算法的面对不同程度干扰时的鲁棒性
- 7) 观看视频，理解傅里叶变换的过程与效果

三、实验原理：

本次实验共分为 4 个小实验，分别对应的是 3.3，4.3，4.4，5.1（接下来统称它们为实验一、二……）。由于不同实验所涉及的原理不同，接下来对各实验原理进行一一分析。

1) 实验一：LSB 隐写：随机选取图像载体色素

实验一的要求主要是利用 **LSB(最不重要位)** 的思想，对读取的图片进行选取图像载体像素，修改其色素中的最低位数值，来实现信息隐藏；但如果仅仅是顺序选取像素点，可能会导致图像各部分统计特征不一致。为了解决这一问题，可以随机间隔选取像素序列。与顺序嵌入相比，需要设计**随机取点**的算法。

关于随机取点的算法原理较为简单，本质上即借助随机种子在二维平面上以不同的间距随机生成一些点。下面重点介绍 LSB 的原理。

最低有效位（LSB）隐写术是一种数字隐写技术，用于将秘密信息嵌入到图像、音频或视频等媒体文件中，以便隐藏信息并实现隐蔽传输。该技术的原理是将秘密信息的二进制位嵌入到媒体文件的最不显

眼的位置，通常是每个像素值的最低位或每个采样值的最低位。这种方法通常不引起人眼或人耳的察觉，因此被广泛用于隐蔽通信和数字水印等领域。

LSB 隐写技术的步骤包括：

1. **选择载体文件：**选择一个合适的图像、音频或视频文件作为载体，该文件将承载秘密信息。
2. **编码秘密信息：**将秘密信息转换为二进制形式，并将其嵌入到载体文件的 LSB 位置。
3. **嵌入策略：**确定嵌入秘密信息的策略，如 LSB 替代法、LSB 匹配法等。
4. **提取秘密信息：**接收方使用相同的 LSB 隐写技术从载体文件中提取出秘密信息。

LSB 隐写技术相对简单，但它有一些限制，包括容量限制（LSB 位数有限）、易受攻击（**直方图分析**、统计分析等）以及可能引起略微的质量损失。

2) 实验二：对图像 DCT 系数的高频系数做删除操作，逆 DCT 变换后，查看图像视觉质量

实验二需要具备图像 DCT 变换的相关知识，在频域上尝试对图像进行修改，删去高频系数再对比图像的变化。

DCT（离散余弦变换）是一种常用于图像压缩和隐写术的数学变换方法，通过将图像数据转换为频域来减小数据的维度。在图像中，DCT 通常会产生频域系数，其中包括低频系数和高频系数。低频系数通常包含图像中的主要结构和信息，而**高频系数则包含图像的细节和纹理。**

离散余弦变换（DCT）与离散傅里叶变换密切相关。它是可分离的线性变换；也就是说，该二维变换等效于先沿一个维度执行一维 DCT，然后在另一维度中执行一维 DCT。输入图像 A 和输出图像 B 的二维 DCT 的定义是：

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad 0 \leq p \leq M-1, 0 \leq q \leq N-1$$

其中：

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq q \leq N-1 \end{cases}$$

图像经过 DCT 变换之后得到 DCT 矩阵，而该矩阵中**高频部分集中在右下角，低频部分集中在左上角**，设计相应的系数矩阵保留 DCT 矩阵中左上角的值，清除右下角的值即可完成 DCT 高频系数的删除操作。

将处理后的图像经过逆 DCT 变换，从频域转化为时域，使用下式计算二维逆 DCT：

$$A_{mn} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \alpha_q B_{pq} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \quad 0 \leq m \leq M-1, 0 \leq n \leq N-1,$$

其中：

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq q \leq N-1 \end{cases}.$$

3) 实验三： DCT 域两点法信息隐藏方法信息的嵌入和提取

①信息嵌入

DCT（离散余弦变换）域的**两点法信息隐藏方法**是一种基于 DCT 变换的隐写技术，旨在将秘密信息嵌入到图像的高频 DCT 系数中，同时尽量减小对原始图像的视觉损失。该方法的主要原理是，通过微小的调整高频 DCT 系数，可以将秘密信息嵌入到图像中，而同时尽量减小对图像质量的影响，以防止检测。这种方法的成功与否取决于嵌入参数的选择，包括用于嵌入的 DCT 系数、调整的大小和秘密信息的编码。需要仔细平衡嵌入容量和视觉质量，以确保实验成功。

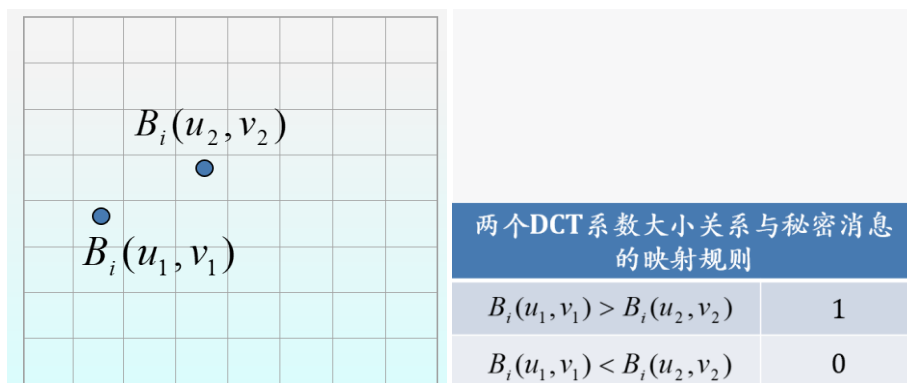


图 1: DCT 系数大小关系与隐藏信息映射说明

②信息提取

信息提取需要先将图像变换至 DCT 域，再按照每一块中约定位置的 DCT 系数值，根据其相对大小，得到隐藏信息的比特串，从而恢复出秘密信息。

③JPEG 压缩条件下的健壮参数 α 与算法鲁棒性的关系

由于实验过程中涉及到放大系数差值，在原本的系数差值的基础上进一步增大该插值，借助超参数 α 来实现此目的，增强信息隐藏的鲁棒性，使图像在保存、信道上传输不发生变化，具体的公式如下所示：

$$|B_i(u_1, v_1) - B_i(u_2, v_2)| > \alpha$$

α 越大鲁棒性越好，对图像的破坏越大。 α 越小鲁棒性越差，对图像的破坏越小。

JPEG 压缩的原理基于离散余弦变换（DCT）和量化。首先，DCT 将图像分解为不同频率成分，包括亮度和细节。然后，通过量化，DCT 系数被舍入为整数，引入信息损失，尤其是在高频部分。最后，经过熵编码，系数被编码以减小文件大小。JPEG 压缩有效地减小了图像文件的大小，但轻微损失了视觉质量。压缩率可调，以在图像质量和文件大小之间找到平衡。这使 JPEG 成为一种流行的图像压缩标准，用于图像传输和存储。

通过调控健壮系数 α 与压缩质量系数 q ，通过作图即可直观地感知算法鲁棒性与健壮系数 α 之间的关系。

4) 实验四原理分析：二值图像信息嵌入和提取

①信息嵌入

二值图像只有两个灰度级，每个像素点均为黑色或者白色。使用一个特定区域中黑像素或白色素的个数来编码秘密信息。

B_i 黑像素的个数与秘密消息的映射规则	
$P_1(B_i) > 50\%$	嵌入一个1
$P_0(B_i) > 50\%$	嵌入一个0

图 2：图像色素占比信息与秘密信息的映射规则

由于 $P_1(B_i)$ 在 50% 附近时，传输错误极有可能破坏嵌入的消息为了使整个系统对传输错误和图像修改具有鲁棒性，必须约束嵌入处理过程。

通过引入健壮系数 λ 、阈值 R_0 与阈值 R_1 ，来提升二值图像信息隐藏的鲁棒性。由于二值图像中像素块的颜色只有黑白两种，因此下面在讨论时均采用 $P_1(B_i)$ ，即指定分块 B_i 中白色像素块的个数进行讨论。

想要提升二值隐写的鲁棒性，则需要确保图像在发送时不同的分块中白色像素点的个数只有以下 3 种情况：

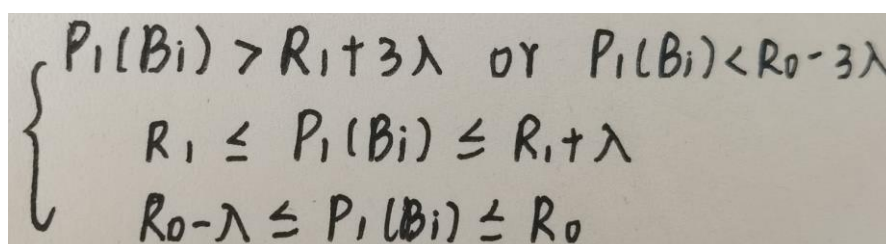

$$\left\{ \begin{array}{l} P_1(B_i) > R_1 + 3\lambda \text{ or } P_1(B_i) < R_0 - 3\lambda \\ R_1 \leq P_1(B_i) \leq R_1 + \lambda \\ R_0 - \lambda \leq P_1(B_i) \leq R_0 \end{array} \right.$$

图 3：图像发送时分块色素的统计状况

由图 3 可得，图像发送时每个分块内的色素数目情况只有 3 种情况：第一种情况说明该分块为不可用块，即它不携带任何信息；第二种情况说明该分块携带 bit 信息 ‘1’；第三种情况说明该分块携带 bit 信息 ‘0’。接收端对每个分块进行一一遍历，便可得到隐藏信息的二进制表示，从而捕获该信息。

但初始二值图像每个分块的色素统计情况基本上不会是像图 3 所示的理想情况，因此需要手动对部分色素进行调整，确保图片发送时能够符合图 3 的情况。

一般一个图像初始分块色素的统计情况如图 4 所示：

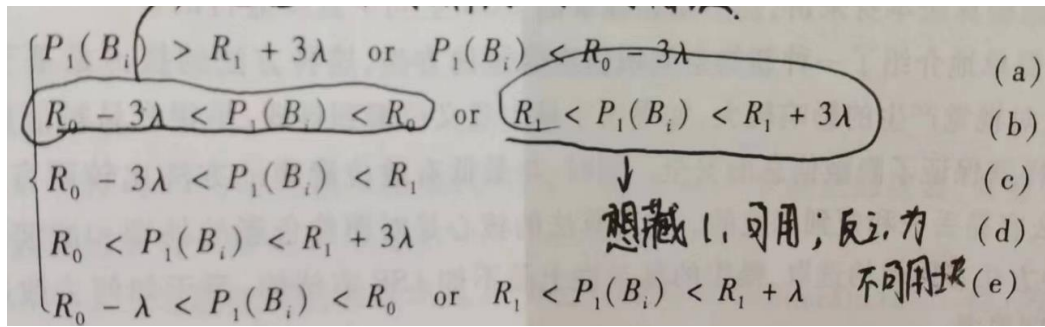


图 4：图像初始分块色素的统计情况

(a)对应的是不可用区域，当 $P_1(B_i)$ 属于这个范围时，表示对该块不加任何操作而跳过。在隐写算法中,图像载体某一块的 $P_1(B_i)$ 可能本来就是属于(a)，则不进行任何操作。 $P_1(B_i)$ 也可能被调整到这一区域，表示这一块与要嵌入的数据不匹配而舍去。

(b)是最复杂的一种情况，不能简单地认定 $P_1(B_i)$ 在这一区域中的图像块是否可用,而应该与秘密信息对应来分析。如果该块试图隐藏的信息是 0,而 $R_1 < P_1(B_i) < R_1 + 3\lambda$,则相应的 $P_1(B_i)$ 过小,与我们最初定义的 $P_1(B_i) > 50\%$,则嵌入一个 0 很不匹配,不能选用这一块用于表示信息 0 的隐藏;同理,如果该块试图隐藏的信息是 1,而 $R_0 - 3\lambda < P_1(B_i) < R_0$ 则相应的 $P_1(B_i)$ 过小,与我们最初定义的 $P_1(B_i) > 50\%$ 则嵌入一个 1 很不匹配,也不能选用这一块用于表示信息 1 的隐藏。在这种情况下,我们称 $P_1(B_i)$ 在 (b) 的块为难以调整块。但如果我们要嵌入的信息是 1 而 $R_1 < P_1(B_i) < R_1 + 3\lambda$ 或者要嵌入的是 0 而 $R_0 - 3\lambda < P_1(B_i) < R_0$,则是毫无影响的,这种情况就被包括在 (c) 和 (d) 中了。

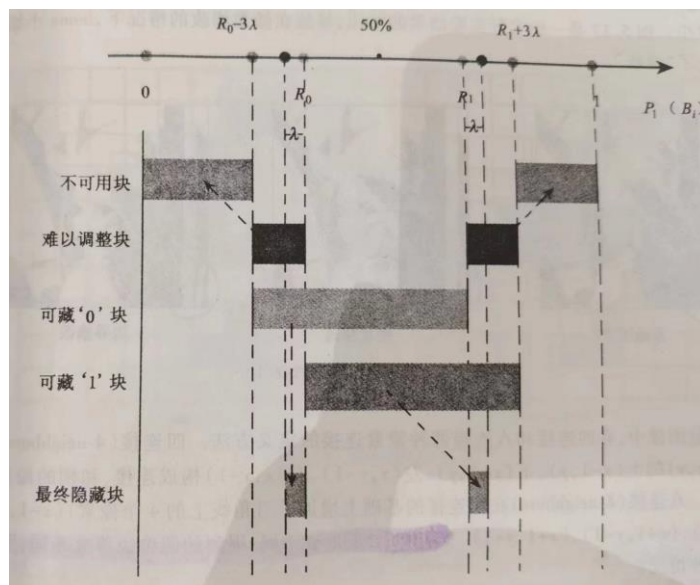


图 5: $P1(B_i)$ 与图像块

(c), (d) 分别表示可以嵌入 1 和可以嵌入 0 的情况, 具体要嵌入的是 0 还是 1 则由秘密信息决定。当 (c), (d) 要嵌入信息时, 需要将 B_i 中的色素块进行调整至 (e) 中的情况。整体调整流程如图 5 所示。

然而实际中对图像块的调整并不是一件很简单的事情, 其中涉及到**如何选择块、块内如何选择点、如何调整、调整数目、联通块的检查、边界扩散现象的预防**等问题, 这些实现中的具体的问题将在实验步骤中具体讨论。

② 信息提取

信息提取相比于信息嵌入过程就简单了很多, 只需要按照信息嵌入时约定好的分块位置, 对这些分块内的色素进行统计即可得到每个分块所隐藏的二进制信息, 从而得到最终的二进制信息。

③ 借助 JPEG 压缩对图像进行随机加噪, 分析阈值 $R0$ 、 $R1$ 、以及健壮系数 λ 对算法鲁棒性的影响

由于阈值 $R0$ 、 $R1$ 、以及健壮系数 λ 是在考虑了信道传输对图像像素点的破坏从而引入的超参数, 因此可以通过手动对图像进行加噪来呈现算法的鲁棒性。针对不同的阈值 $R0$ 、 $R1$ 、以及健壮系数 λ 只需遍历它们的不同取值与不同的压缩质量, 通过对比隐藏的二进制信息与提取出的二进制信息即可呈现算法对噪声干扰的鲁棒性。

四、实验内容:

四个实验所用的示例图像和隐藏文件均为同一份, 如下所示:

所用图像: LenaRGB.bmp

隐藏文件: information.txt



图 6: 实验所用示例图像与文本

（一）实验一（3.3）：LSB 隐写

实验一需要首先设计**随机取点算法**，随机选取像素点作为嵌入秘密信息的载体；将选中的像素点**对应像素值的最低位**改为想要隐藏的数值；重复此过程至所有的秘密信息均被隐藏完毕。

其次需要设计相应的信息提取算法，将隐藏的二进制信息提取出来，同时需要将 2 进制信息每 8bit 转化为 ASCII 码中对应的字符，从而获得隐藏信息的原始字符。

最后需要画图选中的随机位置。由于 LSB 替换只会导致 $0 \leftrightarrow 1, 2 \leftrightarrow 3, 2i \leftrightarrow 2i+1$ ，而不存在 $1 \leftrightarrow 2, 3 \leftrightarrow 4, 2i-1 \leftrightarrow 2i$ 的修改方式，使得灰度为 $2i$ 和 $2i+1$ 的像素数目趋于接近，因此有必要进行并对比隐写前后图像的直方图，分析 LSB 隐写导致的**值对效应**。

整体流程如图 7 所示：

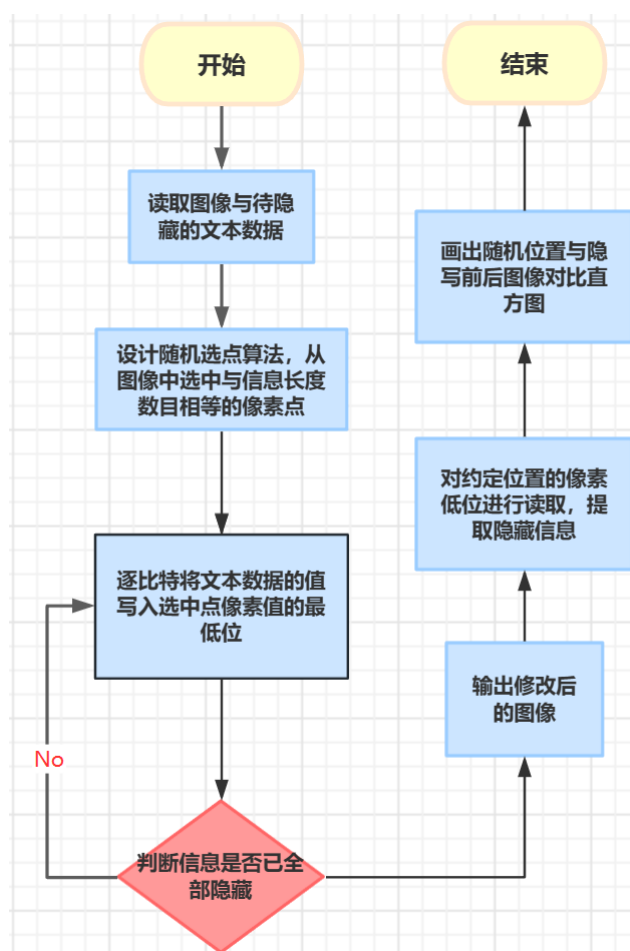


图 7：实验一整体流程

（二）实验二（4.3）：

实验二的整体流程较为简单，本质上是为了熟悉 DCT 域上的相关操作，为后续 DCT 域隐写打下基础。首先需要读取图像，对图像进行分块 DCT2 维变换，从而获得图像对应的 DCT 系数。由于 DCT 矩阵中，低频部分在矩阵的左上角，高频部分在图像的右下角，因此可以编写一个矩阵 x ，对系数矩阵中的高频部分进行滤波，最后再进行逆 DCT 变换，对比前后两张图片的视觉质量。

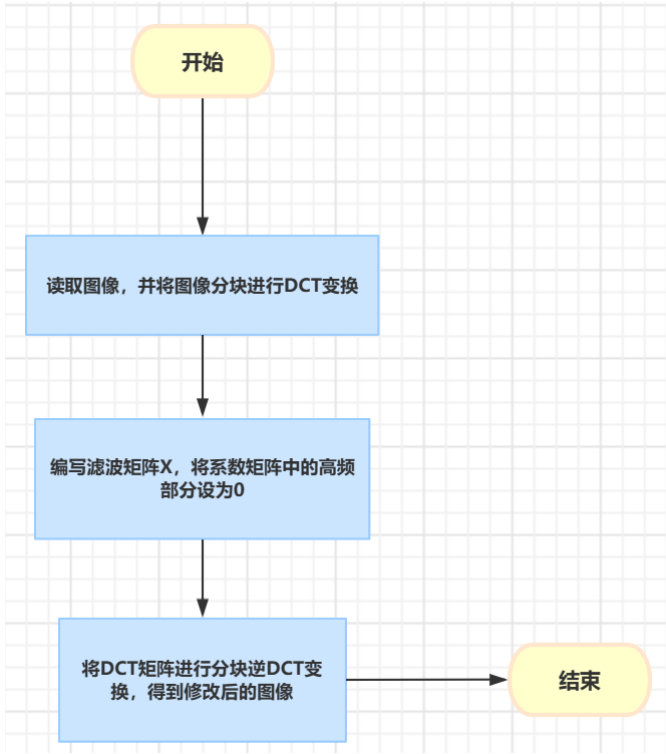


图 8：实验二整体流程

(三) 实验三 (4.4):

实验三此处采用两点法进行图像的信息隐藏。由此方法的原理可知，首先需要读取图像与隐藏文本，并将图像进行分块，对每个分块进行二维 DCT 变换。在分块的 DCT 域上选中指定位置的两点，计算其 DCT 系数的大小关系；根据要嵌入的信息是 0 还是 1 对这两点的系数大小进行调整，使其满足映射规则。最后再根据放大系数插值 α 对指定两点的 DCT 系数进行调整，使其差值的绝对值进一步增大。

完成信息嵌入后需要设计对应的信息提取方法，只需根据修改分块的位置，读取每个分块中指定两点的 DCT 系数大小关系，即可获得此分块所隐藏的比特信息，直到读取数目与嵌入信息长度相同时，即完成全部信息的读取。

最终还需要对隐写图像进行不同压缩质量下的 jpeg 压缩，画图分析健壮系数 α 与算法鲁棒性的关系。

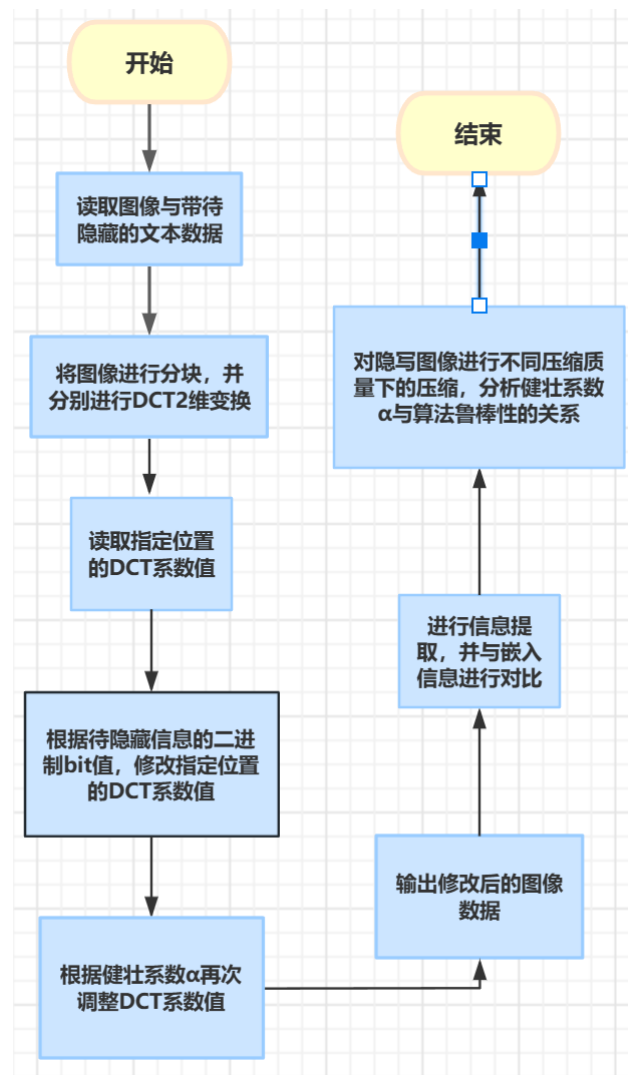


图 8：实验三整体流程

(四) 实验四 (5.1):

实验四的内容是四个实验中最复杂的一个，其牵扯到图像的二值格式转化、随机不重复分块选取算法的设计、md5 加密算法的使用、像素点的修改、分块像素点的统计、像素连接（四连通、八连通）的检查、边界扩散现象的预防、超参数 $R0, R1, \lambda$ 的选取。

其中图像的二值格式转换可以借助 matlab 中的现有函数 `im2bw` 完成，对于二值转换时的阈值选为 0.5；对于 md5 加密算法在此处不需要详细了解其内部原理（该算法的学习是密码学课程的内容），只需要能够使用 `md5` 函数完成不

重复选块/选点即可。每一部分的具体设计会在实验流程中具体讨论，此处只给出该实验的整体流程。

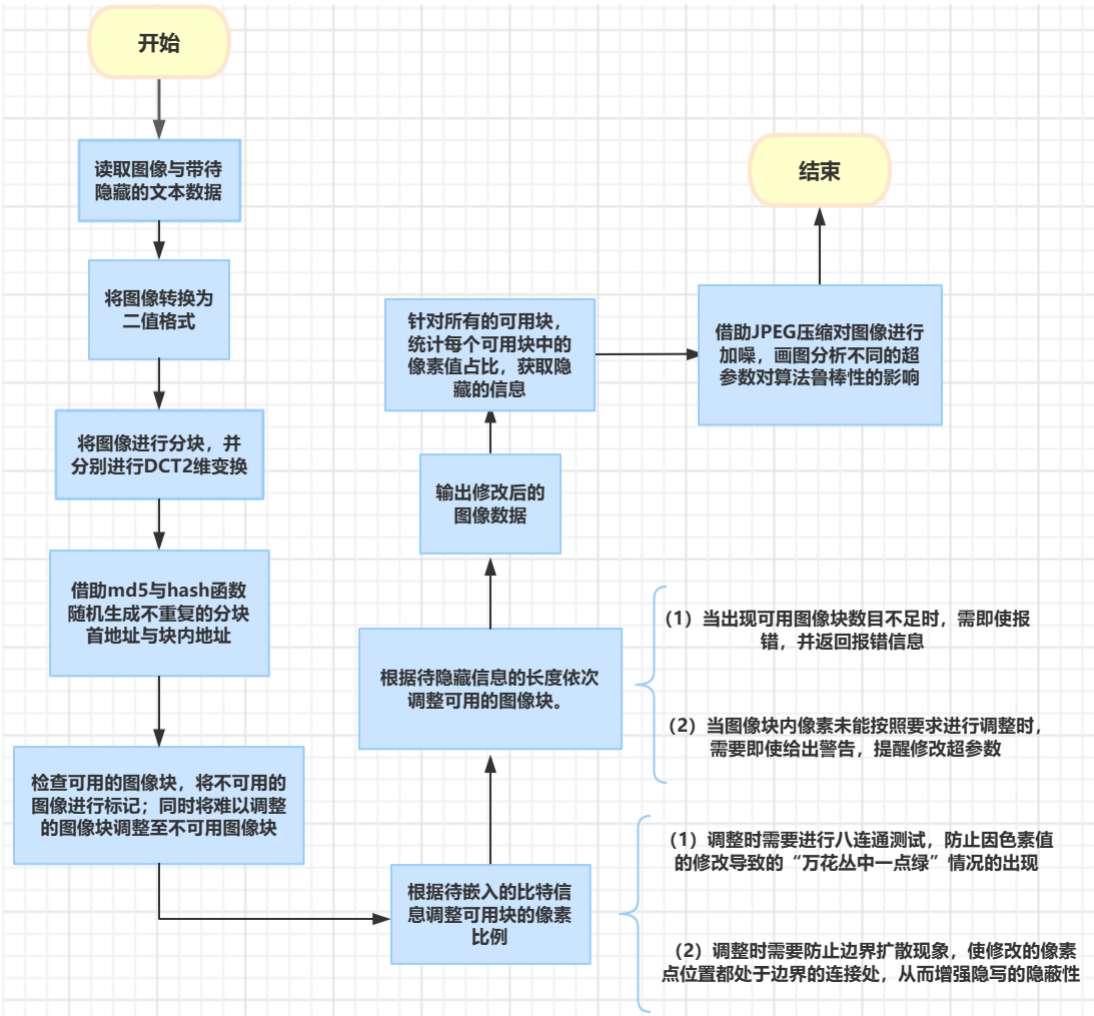


图 9：实验四整体流程

五、实验环境：

- ◆ 操作系统：Windows11 家庭版
- ◆ 基本硬件：Cpu：AMD Ryzen 5 5600H，16G 内存
- ◆ 所用软件：Matlab 2023a

六、实验步骤：

1. 实验一

1.1 随机取点算法的设计

关于随机取点算法的设计，本质上是在给定的图像大小上从左上角的点以一定的概率以不同间隔随机选点，而概率则由 **matlab** 自带的随机数产生函数进行生成。同时为了使实验结果的可重复性与信息提取的便捷性，在生成随机数时引入**随机种子 key**。函数最终返回一组 XY 值，代表随机选取点的位置。实现函数如下所示：

```
01. function [x,y]=randinterval(data,length,key)
02.
03.
04.     rand('seed',key);
05.     a=rand(length);
06.
07.     [m,n]=size(data);
08.     interval1=floor(m*n/length)+1;
09.     interval2=interval1-2;
10.     if interval2==0
11.         error("载体过小");
12.     end
13.     x=zeros(1,length);
14.     y=zeros(1,length);
15.
16.     tx=1;
17.     ty=1;
18.
19.     x(1)=tx;
20.     y(1)=ty;
21.
22.     for i=2:length
23.
24.         if a(i)>0.5
25.             ty=ty+interval1;
26.         else
27.             ty=ty+interval2;
28.         end
29.
30.         if ty>n
31.             tx=tx+1;
32.             if tx>m
33.                 error("载体过小");
34.             end
35.         end
36.         ty=mod(ty,n);
37.         if ty==0
38.             ty=1;
39.         end
40.         x(i)=tx;
41.         y(i)=ty;
42.     end
43. end
```

1.2 嵌入信息的函数

在嵌入信息时只需要先读取指定路径下的图像与文本，并将文本数据进行调整，文本信息的调整缘由是 **matlab** 读取数据的格式。当文本数据为 1 时，1 对应的 ASCII 码值为 00110001，而 **matlab** 采用小端存储，即读取 1 对应的二进

制数值为 10001100，为了后续提取信息时能够成功将二进制信息转化为文本信息，在信息嵌入时就进行调整，将读取的信息**转化为大端存储**。此过程会在后续每个需要嵌入信息的实验中重复进行，在此进行陈述，后续就不一一进行叙述。

同时该实验是**基于灰度图像**进行的，而读取的 lena 图像为 RGB 类型图像，因此还需将其进行**格式转化**。实现上述目的代码如下所示：

```
01. fid=fopen(filepath,'r');
02. [A,B]=fread(fid,'ubit1');
03. tA=A;
04. for i=1:8:B
05.     A(i)=tA(i+7);
06.     A(i+1)=tA(i+6);
07.     A(i+2)=tA(i+5);
08.     A(i+3)=tA(i+4);
09.     A(i+4)=tA(i+3);
10.     A(i+5)=tA(i+2);
11.     A(i+6)=tA(i+1);
12.     A(i+7)=tA(i);
13. end
14.
15. fclose(fid);
16.
17. data=rgb2gray(imread(imgpath));
```

在完成信息的读取之后，需要调用随机生成位置的函数，在生成的位置上修改对应像素的最低位的值，即首先将像素最低位的值置为 0（在此通过与原图像数据**模 2 结果相减**实现），再根据待嵌入信息的值修改最低位的值。

```
01. [x,y]=randinterval(data,B,key);
02. for i=1:B
03.
04.     data(x(i),y(i))=data(x(i),y(i))-mod(data(x(i),y(i)),2)+A(p);
05.     p=p+1;
06.
07. end
```

1.3 提取信息的函数

完成信息隐藏后，只需按照之前的修改像素点的位置，读取其最低位的数值，即可获得隐藏信息的二进制表示，具体代码实现如下所示：

```
01. for i=1:length
02.
03.     x=site(1,i);
04.     y=site(2,i);
05.     if bitand(data(x,y),1)==1
06.         fwrite(fid,1,'bit1');
07.         r(i)=1;
08.     else
09.         fwrite(fid,0,'bit1');
10.         r(i)=0;
11.     end
12.
13. end
```

完成信息的提取后，为了便于我们与原始嵌入的信息进行对比，因此将二进制信息转化为**字符信息**是很有必要的，转化过程如下所示：

```
01. resultString = '';
02.
03. for i = 1:8:length
04.     eightBits = r(i:i+7); % 获取下一个8位的二进制序列
05.     decimalValue = bin2dec(num2str(eightBits)); % 将二进制转换为十进制
06.     character = char(decimalValue); % 将十进制转换为ASCII字符
07.     resultString = [resultString, character]; % 将字符添加到结果字符串
08. end
```

最终函数返回提取出的二进制信息与对应的字符信息。

1.4 呈现选取的随机位置

在信息嵌入的过程中已经随机生成了像素点，只需要在图中在指定像素点的位置做出标记即可（此处采用的是画出**红色的圆圈**）。

```
01. function [r]=draw_site(m_data,site)
02.
03. figure;
04. imshow(m_data);
05. [a,length]=size(site);
06. for i=1:length
07.     x=site(1,i);
08.     y=site(2,i);
09.     hold on;
10.     viscircles([x, y], 2, 'EdgeColor', 'r', 'LineWidth', 2);
11. end
12. hold off; % 停止保持绘图
13. end
```

1.5 对比隐写前后的图像直方图

使用 matlab 自带的 **histogram** 函数，将原始图像矩阵与修改后的图像矩阵展平统计其不同像素值出现的次数即可，并将两个直方图绘制在一起。

将信息嵌入函数返回的原始图像与修改后的图像传入 **analyze_lsb** 函数中即可完成上述目的。

```

01. function [r]=analyze_lsb(origin_data,m_data)
02.
03. data1 = origin_data(:);
04. data2 = m_data(:);
05. figure;
06. % 第一个子图
07. subplot(1, 2, 1);
08. histogram(data1, 'BinMethod', 'integers');
09. title('原始图像');
10. xlabel('数值');
11. ylabel('出现次数');
12.
13. % 第二个子图
14. subplot(1, 2, 2);
15. histogram(data2, 'BinMethod', 'integers');
16. title('修改后的图像');
17. xlabel('数值');
18. ylabel('出现次数');
19. end

```

2. 实验二

2.1 图像的 DCT 变换与逆 DCT 变换

本实验比较简单，对于图像的分块 DCT 变化可直接通过 matlab 的内置函数完成。将读取的图像转化为灰度图像，并进行 DCT 转化，并根据设置的滤波矩阵 X 即可将 DCT 矩阵中的高频部分置为 0，并将处理后的图像进行逆 DCT 变换即可。

```

01. function [r]=dct_change(imgpath)
02. f=rgb2gray(imread(imgpath));
03. f=double(f)/255;
04. T=dctmtx(8);
05. B=blkproc(f,[8 8], 'P1*x*P2',T,T');
06. X=[1 1 1 1 0 0 0 0
07.    1 1 1 0 0 0 0 0
08.    1 1 0 0 0 0 0 0
09.    1 0 0 0 0 0 0 0
10.    0 0 0 0 0 0 0 0
11.    0 0 0 0 0 0 0 0
12.    0 0 0 0 0 0 0 0
13.    0 0 0 0 0 0 0 0];
14. B2=blkproc(B,[8 8], 'P1.*x',X);
15. % subplot(1,2,1),imshow(log(abs(B)),[]);title('原始图像');
16. % colormap parula
17. % colorbar
18. % subplot(1,2,2),imshow(log(abs(B2)),[]);title('修改后的图像');
19. % colormap parula
20. % colorbar
21. I2=blkproc(B2,[8 8], 'P1*x*P2',T',T);
22. subplot(1,3,1),imshow(f);title('原始图像');
23. subplot(1,3,2),imshow(I2);title('逆变换后的图像');
24. M=I2-f;
25. subplot(1,3,3),imshow(mat2gray(M)),title('图像细节');
26.
27. end

```


3. 实验三

3.1 图像的分块 DCT 变换并计算每个分块的首地址

首先对图像和文本信息进行读取，将图像进行分块，并分别进行 DCT 变换。再借助随机生成点函数，生成与分块数目相同的坐标点，再将其转化为每个分块的首地址。

```
01. T=dctmtx(8);
02. dctrgb=blkproc(data_R,[8 8], 'P1*x*P2',T,T');
03.
04. [row,col]=size(dctrgb);
05. row=floor(row/8);
06. col=floor(col/8);
07. a=zeros([row col]);
08.
09. [k1,k2]=randinterval(a,B,key);
10. for i=1:B
11.     k1(i)=(k1(i)-1)*8+1;
12.     k2(i)=(k2(i)-1)*8+1;
13. end
```

3.2 根据隐藏信息和映射规则调整 DCT 系数大小关系

获取到隐藏文本和每个分块指定两位置（两点法）的系数大小关系后，根据映射规则手动调整系数大小。具体的操作即欲使 $A > B$ ，而此时 $A < B$ ，只需交换两者的数值即可， $A < B$ 也同理。

调整分块结束之后，再根据健壮系数 α 进一步增大两点系数大小差异。

具体的实现代码如下所示：

```
01. temp=0;
02. for i=1:B
03.     if A(i)==0
04.         if dctrgb(k1(i)+4,k2(i)+1)>dctrgb(k1(i)+3,k2(i)+2)
05.             temp= dctrgb(k1(i)+4,k2(i)+1);
06.             dctrgb(k1(i)+4,k2(i)+1)=dctrgb(k1(i)+3,k2(i)+2);
07.             dctrgb(k1(i)+3,k2(i)+2)=temp;
08.         end
09.     else
10.         if dctrgb(k1(i)+4,k2(i)+1)<dctrgb(k1(i)+3,k2(i)+2)
11.             temp= dctrgb(k1(i)+4,k2(i)+1);
12.             dctrgb(k1(i)+4,k2(i)+1)=dctrgb(k1(i)+3,k2(i)+2);
13.             dctrgb(k1(i)+3,k2(i)+2)=temp;
14.         end
15.     end
16.     end
17.     %将原本小的系数放的更小，增大差异
18.     if dctrgb(k1(i)+4,k2(i)+1)>dctrgb(k1(i)+3,k2(i)+2)
19.         dctrgb(k1(i)+3,k2(i)+2)=dctrgb(k1(i)+3,k2(i)+2)-alpha;
20.     else
21.         dctrgb(k1(i)+4,k2(i)+1)=dctrgb(k1(i)+4,k2(i)+1)-alpha;
22.     end
23. end
```

3.3 提取信息

将修改后端图像转化到 DCT 域，并根据嵌入信息时所使用到的块，读取其

中指定两点的系数大小关系，即可获得隐藏的二进制信息，再通过 ASCII 码的转化即可获得最终的字符信息。

```
01. function [r,resultString]=dct_2_decode(data,key,B)
02.
03.     data=double(data)/255;
04.     data=data(:,:,1);
05.
06.     T=dctmtx(8);
07.     dctrgb=blkproc(data,[8 8], 'P1*x*P2',T,T');
08.     [row,col]=size(dctrgb);
09.     row=floor(row/8);
10.     col=floor(col/8);
11.     a=zeros([row col]);
12.     [k1,k2]=randinterval(a,B,key);
13.     r=zeros(1,B);
14.     for i=1:B
15.         k1(i)=(k1(i)-1)*8+1;
16.         k2(i)=(k2(i)-1)*8+1;
17.     end
18.     for i=1:B
19.         if dctrgb(k1(i)+4,k2(i)+1)<=dctrgb(k1(i)+3,k2(i)+2)
20.             r(i)=0;
21.         else
22.             r(i)=1;
23.         end
24.     end
25.
26.     resultString = '';
27.     for i = 1:8:B
28.         eightBits = r(i:i+7); % 获取下一个8位的二进制序列
29.         decimalValue = bin2dec(num2str(eightBits)); % 将二进制转换为十进制
30.         character = char(decimalValue); % 将十进制转换为ASCII字符
31.         resultString = [resultString, character]; % 将字符添加到结果字符串
32.     end
33.
34. end
```

3.4 分析 JPEG 压缩条件下健壮系数 α 和算法鲁棒性的关系

关于 JPEG 压缩可以通过 `imwrite` 写出图像时，指定图像为 `jpg` 格式，并控制压缩质量系数。再次读取经过 JPEG 压缩后的图像，对其中的内容进行提取，与原始信息进行对比即可获得差异的数目，通过画图获得健壮系数 α 和算法鲁棒性的关系。

```

01. for a=alpha
02.
03.     res_r=res_r+1;
04.     res_c=0;
05.     [origin,modify_data,count]=dct_2_encode(imgpath,filepath,key,a);
06.     for q=quality
07.         different=0;
08.         res_c=res_c+1;
09.         imwrite(modify_data,'temp.jpg','jpg','quality',q);
10.         tempdata=imread('temp.jpg');
11.         [r,resultString]=dct_2_decode(tempdata,key,count);
12.         for i=1:count
13.             if r(i)~=A(i)
14.                 different=different+1;
15.             end
16.         end
17.         res(res_r,res_c)=different/count;
18.     end
19.
20.     for i=1:10
21.         plot(quality,res(i,:));
22.         hold on;
23.     end
24. end

```

4. 实验四

4.1 哈希置换法实现随机选块

为确保在图像分块的选择与图像内部点的生成时的随机性与不重复性（重点是不重复性），此处借助 **md5 加密函数** 的特性（不重复性）实现上述目的，给定一个矩阵与想要生成的点的数目，在三个密钥（key1, key2, key3）的协助下即可生成目标点，具体的实现代码如下所示：

```

01. function [row,col]=hashreplacement(matrix,count,k1,k2,k3)
02.
03.     [X,Y]=size(matrix);
04.     row=zeros([1,count]);
05.     col=zeros([1,count]);
06.     j=zeros([1,count]);
07.     for i=1:count
08.
09.         v=round(i/X);
10.         u=mod(i,X);
11.         v=mod(v+md52num(GetMD5(num2str(u+k1))),Y); %注意m是个数值，需要改为str
12.         u=mod(u+md52num(GetMD5(num2str(v+k2))),X);
13.         v=mod(v+md52num(GetMD5(num2str(u+k3))),Y);
14.         j(i)=v*X+u+1;
15.         col(i)=mod(j(i),Y);
16.         row(i)=floor(j(i)/Y)+1;
17.
18.         %row(i)=double(uint8(row(i)))+1;
19.         if col(i)==0
20.             col(i)=Y;
21.             row(i)=row(i)-1;
22.         end
23.     end
24. end
25.

```

4.2 分块首地址的确定与内部点的生成

在经过上述哈希置换法后可得到每个分块的序列号（在此处选用 10×10 大小的分块，图片大小为 512×512 ，即整个图片最多可分为 51×51 个小块）， $row=x$ ， $col=y$ ，即代表选用第 x 行中的第 y 个分块，下述代码分块序号转化为它们对应的首地址。同时生成每个 10×10 分块的 8×8 个内部点。

```

01. [row,col]=hashreplacement(temp,m*n,m,key,n);
02.     for i=1:m*n
03.         if row(i)~=1
04.             row(i)=(row(i)-1)*10+1;
05.         end
06.
07.         if col(i)~=1
08.             col(i)=(col(i)-1)*10+1;
09.         end
10.     end
11.
12.
13.     temp=zeros(8);
14.     [randr,randc]=hashreplacement(temp,64,key,m,n);

```

之所以要生成 8×8 个内部块与后续的八连接检测紧密相关，由于处于边界位置的点定然不存在八连接，因此在选点的时候便忽略它们以提高算法的运行效率。

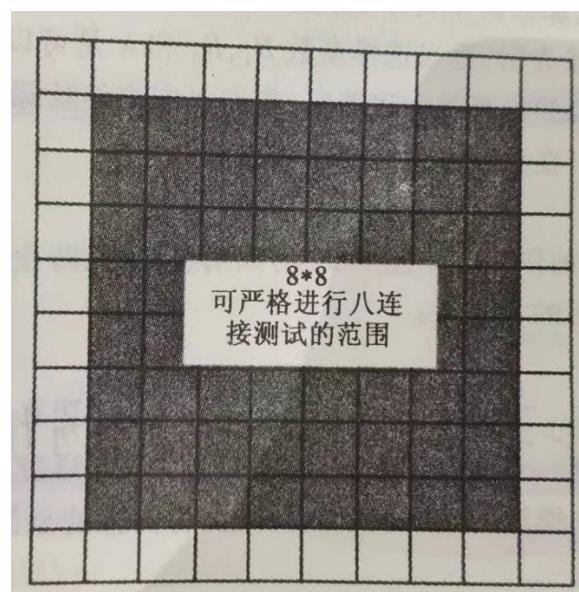


图 10：能够严格进行八连接检测的范围

4.3 分析所有分块中的可用块

在实验原理部分已经分析过初始块中哪些情况为不可用块，哪些情况为可用块；此部分的代码即要完成此目的。

当白色像素点的数目处于 $[0, R0-3\lambda]$ 或 $[R1+3\lambda, 100]$ 时可直接标记此块为不可用块；当要传送 bit ‘1’ 而白色像素点的数目为 $[R0-3\lambda, R0]$ 时可直接标记此块为

不可用块；当要传送 **bit ‘0’** 而白色像素点的数目为 $[R1, R1+3\lambda]$ 时可直接标记此块为不可用块。其余情况均可视为该块可用。

在上述三种不可用情况中的第二、三种情况中，不仅要标记该块为不可用块，同时还要使该块的白色像素点的数目处于 $[0, R0-3\lambda]$ 或 $[R1+3\lambda, 100]$ ，这样则便利后续信息提取过程。

最后将可用块的序列号进行保存，即可获得可满足待隐藏信息长度数目的可用块。

具体的实现代码如下所示：

```
01. for blockquan=1:m*n
02.
03.     p1bi=compute_p1bi(row(blockquan),col(blockquan),data); %计算一个10*10块中值为1的像素点
04.     if p1bi>=R1+3*lamda || p1bi<=R0-3*lamda
05.         row(blockquan)=-1;
06.         col(blockquan)=-1;
07.         unable=unable+1;
08.         msgquan=msgquan-1;
09.     elseif msg(msgquan)==1 && p1bi<=R0
10.         data=editp1bi(row(blockquan),col(blockquan),data,1,3*lamda,randr,randc);
11.         row(blockquan)=-1;
12.         col(blockquan)=-1;
13.         difficult=difficult+1;
14.         msgquan=msgquan-1;
15.     elseif msg(msgquan)==0 && p1bi>=R1
16.         data=editp1bi(row(blockquan),col(blockquan),data,0,3*lamda,randr,randc);
17.         row(blockquan)=-1;
18.         col(blockquan)=-1;
19.         difficult=difficult+1;
20.         msgquan=msgquan-1;
21.     else
22.         end
23.
24.     msgquan=msgquan+1;
25.     if msgquan==count+1
26.         for i=(blockquan+1):m*n
27.             row(i)=-1; %made 此处写成了row(blockquan)
28.             col(i)=-1;
29.         end
30.         break;
31.
32.     end
33. end
```

4.4 统计分块中白色像素点的数目

只需在 $10*10$ 大小的分块中遍历每个像素点的值即可。

```
01. function [r]=compute_p1bi(headr,headc,image)
02.
03.     r=0;
04.     for i=1:10
05.         for j=1:10
06.             if image(headr+i-1,headc+j-1)==1
07.                 r=r+1;
08.             end
09.         end
10.     end
11.
12. end
```

4.5 分块区域像素点的修改

对于分块像素点的修改，需要指定被修改的像素点的取值、需要修改的像素点的个数。在修改时并不是分块内的所有元素均能进行修改，只有通过了八连接检测的位置才能进行修改，确保修改的点周围存在与之相反的像素值，这样可用避免在修改的图像中出现“万花丛中一点绿”的显眼情况。

其次在修改时，还要预防边界扩散现象，即刚刚修改的位置，成为临近区域通过八连接检测的“通行证”，这样可能会导致出现明显的图像修改。具体的操作即在修改的像素点处加一个较小的值，将其与原本的点区分开，这样便可以有效的预防边界扩散现象。



图 11：边界扩散现象说明

所用代码如下所示：

```
01. function [r]=editp1bi(headr,headc,data,pixel,count,randr,randc)
02. %pixel: 要求修改的色素块的颜色，即要把pixel改为其反
03. %count: 修改数量
04. c=0;
05. for i =1:64
06.     %disp([headr,"||||",headc]);
07.     row=headr+randr(i);
08.     col=headc+randc(i);
09.     if data(row,col)==pixel
10.         if (data(row-1,col)==~pixel || ...
11.             data(row+1,col)==~pixel || ...
12.             data(row,col-1)==~pixel || ...
13.             data(row,col+1)==~pixel || ...
14.             data(row-1,col-1)==~pixel || ...
15.             data(row-1,col+1)==~pixel || ...
16.             data(row+1,col-1)==~pixel || ...
17.             data(row+1,col+1)==~pixel)
18.
19.             data(row,col)=~pixel+0.01;
20.             c=c+1;
21.         end
22.     end
23.     if c==count
24.         r=data;
25.         disp("修改成功");
26.         return
27.     end
28.
29. end
30. if c~=count
31.     r=data;
32.     disp("warning, 未能按要求修改本块元素，可能出现信息无法提取的情况");
33. end
34. end
```

4.6 最终的信息隐藏过程

在取得了所有的可用块，并对不可用块进行调整之后，则可进行最终的信息嵌入，只需按照待嵌入的信息的取值，修改可用块中白色像素点的占比，确保图片发送时，代表 **bit1** 的分块中白色像素点的占比必在 $[R1, R1+\lambda]$ 之间；代表 **bit0** 的分块中白色像素点的占比必在 $[R0-\lambda, R0]$ 之间。

所用代码如下所示：

```
01. for i=1:B
02.     p1bi=compute_p1bi(availablel(i),availablec(i),image);
03.     if A(i)==1
04.         if p1bi<R1
05.             image=editp1bi(availablel(i),availablec(i),image,0,R1-p1bi+1,randr,randc);
06.         elseif p1bi>R1+lamda
07.             image=editp1bi(availablel(i),availablec(i),image,1,p1bi-R1-lamda+1,randr,randc);
08.         else
09.             end
10.     end
11.
12.     if A(i)==0
13.         if p1bi>R0
14.             image=editp1bi(availablel(i),availablec(i),image,1,p1bi+1-R0,randr,randc);
15.         elseif p1bi<R0-lamda
16.             image=editp1bi(availablel(i),availablec(i),image,0,R0-lamda-p1bi+1,randr,randc);
17.         else
18.             end
19.     end
20. end
```

4.7 信息提取过程

相比信息嵌入过程，信息提取过程则显得尤为简单，只需要在所有块中进行遍历，计算每一块的白色像素点的占比，并根据规定好的占比范围关系，获得隐藏的信息。

```
01. for i=1:m*n
02.     p1bi=compute_p1bi(row(i),col(i),data);
03.     %if p1bi<R1+3*lamda && p1bi>50
04.     if p1bi<=R1+lamda && p1bi>=R1
05.         r(quan)=1;
06.         quan=quan+1;
07.     elseif p1bi>R0-3*lamda && p1bi<50
08.     elseif p1bi>=R0-lamda && p1bi<=R0
09.         r(quan)=0;
10.         quan=quan+1;
11.     else
12.         quan=quan;
13.     end
14.     if quan==count+1
15.         break;
16.     end
17. end
```

4.8 在随机加噪的情况下分析阈值 $R0, R1$ 健壮系数 λ 对算法鲁棒性的影响

由 JPEG 格式压缩而产生的图像噪音也被称为**马赛克噪音 (Block Noise)**，压缩率越高图像噪音就越明显。虽然把图像缩小后这种噪音也会变得看不出来但放大打印后，进行色彩补偿就表现得非常明显。在此处借助 JPEG 的

特性对图像进行不同程度的加噪，来分析不同超参数的设置在面对随机噪声时的影响。

具体的代码实现与实验三中的 jpeg 压缩过程十分相似，只是将遍历的系数改为阈值 $R0, R1$ 健壮系数 λ 。

关于阈值 $R0$ $R1$ ($R0$ 和 $R1$ 的取值是对称的)的不同取值在随机加噪的情况下对算法鲁棒性的影响可由下述函数得出：

```
01. function [res]=jpg_R0(imgpath,filepath,key,R0,R1,lamda)
02.
03. fid=fopen(filepath,'r');
04. [A,B]=fread(fid,'ubit1');
05. fclose(fid);
06. tA=A;
07. for i=1:8:B
08.     A(i)=tA(i+7);
09.     A(i+1)=tA(i+6);
10.     A(i+2)=tA(i+5);
11.     A(i+3)=tA(i+4);
12.     A(i+4)=tA(i+3);
13.     A(i+5)=tA(i+2);
14.     A(i+6)=tA(i+1);
15.     A(i+7)=tA(i);
16. end
17.
18. quality=10:10:100;
19. R0=36:45;
20. result=zeros([max(size(R0)) max(size(quality))]);
21. res_r=0;
22. res_c=0;
23. j=1;
24. for r0=R0
25.
26.     res_r=res_r+1;
27.     res_c=0;
28.     [res,B]=binaryhide(imgpath,filepath,key,r0,R1,lamda);
29.     for q=quality
30.         different=0;
31.         res_c=res_c+1;
32.         imwrite(res,'temp2.jpg','jpg','quality',q);
33.         tempdata=imread('temp2.jpg');
34.         [r,r_string]=b_extract(tempdata,r0,R1,key,lamda,B);
35.         for i=1:B
36.             if r(i)~=A(i)
37.                 different=different+1;
38.             end
39.         end
40.         result(res_r,res_c)=different/B;
41.     end
42.
43.     for i=1:10
44.         plot(quality,result(i,:));
45.         hold on;
46.     end
47.     legend_str{j} = ['R0=',num2str(r0)];
48.     j=j+1;
49. end
50. legend(legend_str);
51. xlabel("jpeg压缩质量");
52. ylabel("误码率");
53. end
```

最终可得到误码率与 $R0$ $R1$,压缩质量的取值关系图，关于此图的分析在后续实验结果中给出；关于 λ 的探究与 $R0$ $R1$ 基本相同，此处就不再重复给出代码。

七、实验结果与分析

1. 实验一

1.1 图像隐写对比

通过运行信息嵌入代码，可得嵌入图像的数据，使用子图同时输出原始图像与修改图像如图 12 所示：



图 12：LSB 隐写对比

可以看到经过修改的图像几乎没有变化，即人眼看不出差异，说明 LSB 隐写具有较强的隐蔽性。

1.2 信息提取呈现

将上述被修改后的图像送入提取函数中，可提取出隐藏的信息，如图 13 所示。r 中存放了隐藏信息的二进制表示，r_ASCII 中存放了对应的字符数据。



图 13：LSB 隐写信息提取

1.3 随机位置的呈现

通过简单的画图函数即可呈现，如图 14 所示：

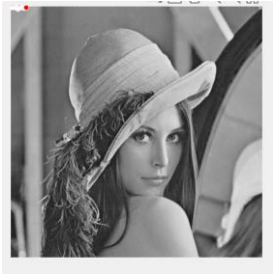


图 14：随机位置的呈现

可以看到生成的随机点基本上都集中在了左上角，这与随机算法的设计是有关的，随机算法的设计本身就是从左上角开始首先完成本行内的选点，当本行内的选点结束后，再向下前进一行。由于此处嵌入的信息长度较短，因此选

中的点都集中在了开始位置的附近。

1.4 隐写直方图对比

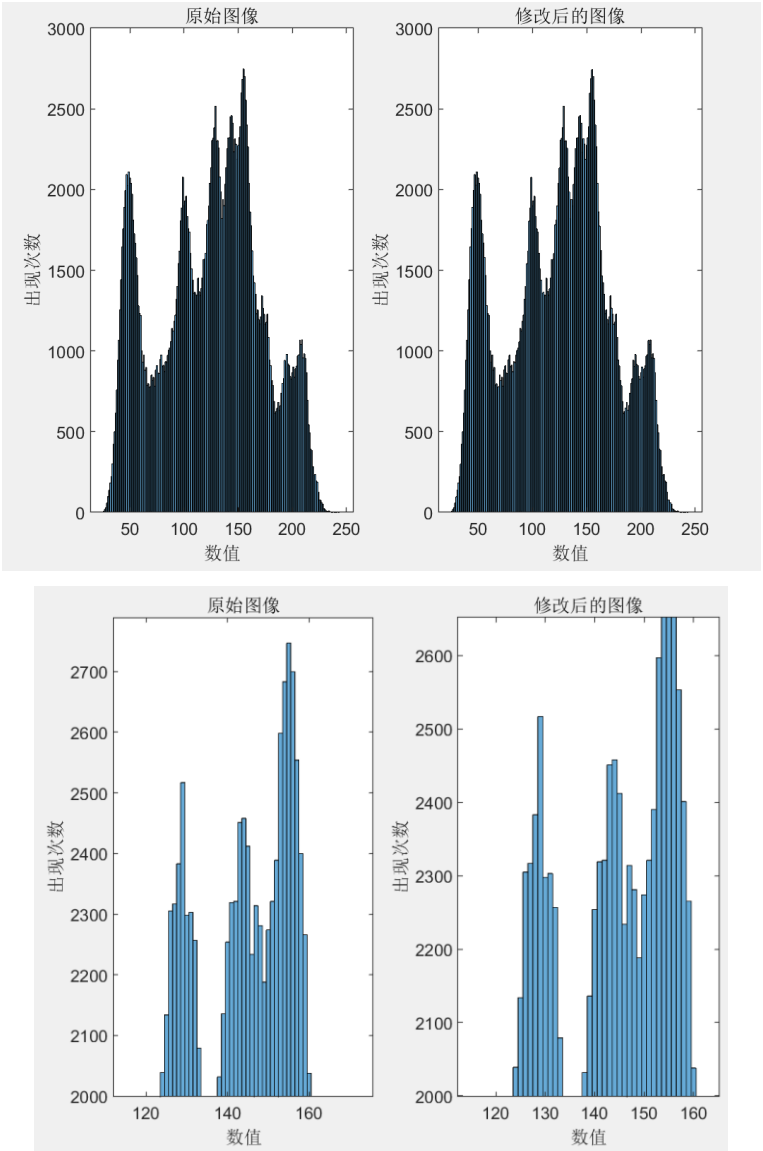


图 15：隐写直方图

从图 13 可以看到，经过 LSB 隐写后 $2i$ 与 $2i+1$ 的像素数目会稍趋于接近，但不是十分明显。主要原因是图像的像素个数较大（ 512×512 ）而嵌入的信息只为 32bit，故修改不明显；但也达到了查看 LSB 值对效应的效果。

2. 实验二

2.1 高频系数删除后的对比图



图 16: DCT 滤波后的对比图

可以看到修改后的图像相比原图，在**缺失了一些细节**，尤其是人物的头发部分最为显著，一些颜色变化较为剧烈的部分变得平滑。

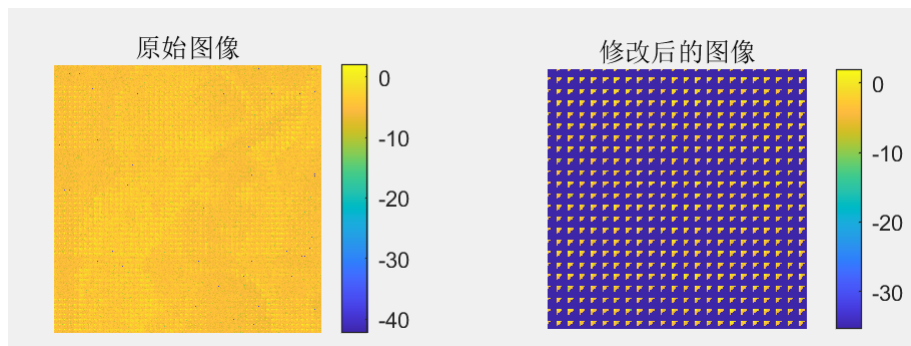


图 17: DCT 系数值对应的色阶图

可以看到，原始图像中的高频部分还是相对较多的，且由于修改时采用分块修改，即每个 8×8 的小块为一个单位进行高频滤波修改，因此在修改后的图像中可以明显看出每个正方形小块中的左上角为保留的高频部分，而右下角已经被清除。

而且高频信息往往携带了图像的细节部分，因此经过滤波后的图像会缺失一些细节信息。

3. 实验三

3.1 DCT 域隐写对比

通过两点法将信息隐藏于图像之中，并将原图像和修改后的图像进行对比，可以看到视觉上几乎无任何变化，说明 DCT 域的隐写具有**较强的隐蔽性**。

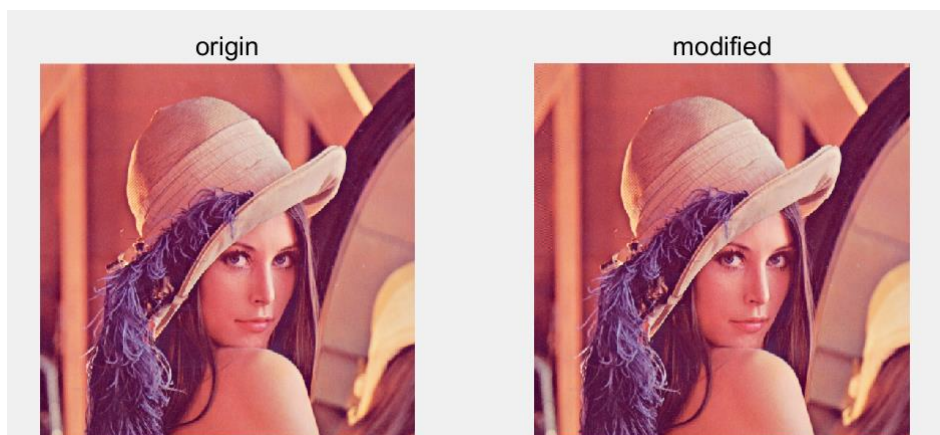


图 18: DCT 域隐写对比

3.2 信息提取结果

将上述被修改后的图像送入提取函数中，可提取出隐藏的信息，如图 19 所示。`r` 中存放了隐藏信息的二进制表示，`resultString` 中存放了对应的字符数据。



图 19: DCT 域隐写信息提取结果

3.3 JPEG 压缩条件下健壮系数 α 和算法鲁棒性的关系

通过遍历不同的压缩质量 q 与不同的健壮系数 α ，分别得到不同情况下信息提取的结果，将该结果与原始隐写信息进行一一比较，即可获得经过压缩后的信息提取准确率。实验结果如图 20 所示：

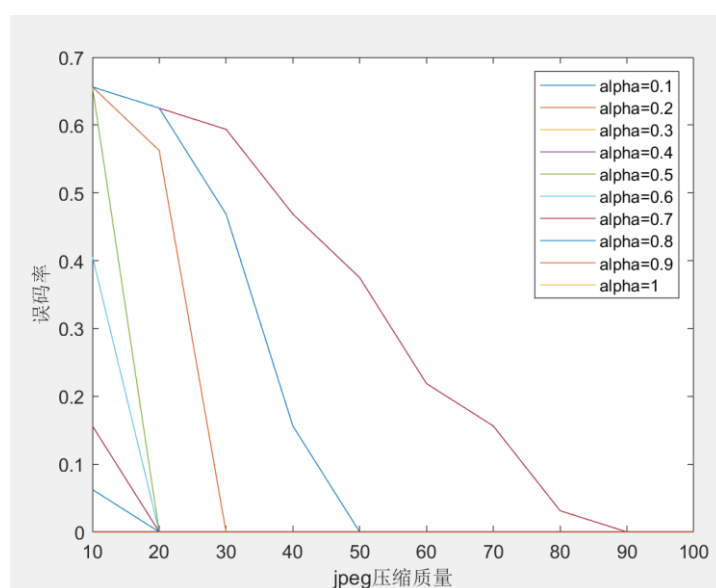


图 20: 不同情况下的信息提取准确率

图中不同颜色的曲线代表了不同的 α 取值，横坐标为 jpeg 压缩质量，纵坐标为提取的信息与原始信息不同的百分比例（即**误码率**）。在同一压缩质量下可以得到 α 越大，算法对于 jpeg 压缩的鲁棒性就越强；在同一 α 的取值下，压缩质量越高，误码率越小（纵坐标的值越小）。

当 α 的值为 0.1 时，完全不能保证信息提取的正确性；当 α 在[0.2,0.4]区间内，抗 JPEG 压缩的性能一般；当 $\alpha=1$ 时，基本可以认为是不受 JPEG 压缩的影响的。

4. 实验四

4.1 信息嵌入对比图



图 21：二值图像隐写对比图

此处使用的图像大小为 512*512，所用的超参数 $R0=47, R1=53$ ，健壮系数 $\lambda=3$ ，密钥 $key=12$ 。

可用看到经过二值图像隐写，图像也并不会产生明显的变化，将图像放大进行观看时只能在红框内看到极其细微的变化，说明该算法具有较好的隐蔽性。

4.2 信息提取

对于信息的提取只需将上述修改后的图像送入到提取函数中即可，与前面的信息提取相同，此处可以成功提取出隐藏的信息“1123”。

r	1x32 double
R0	47
R1	53
r_string	'1123'
res	512x512 double

图 22：二值图像隐写信息提取结果

4.3 随机加噪的情况下，R0、R1 的不同取值对算法鲁棒性的影响

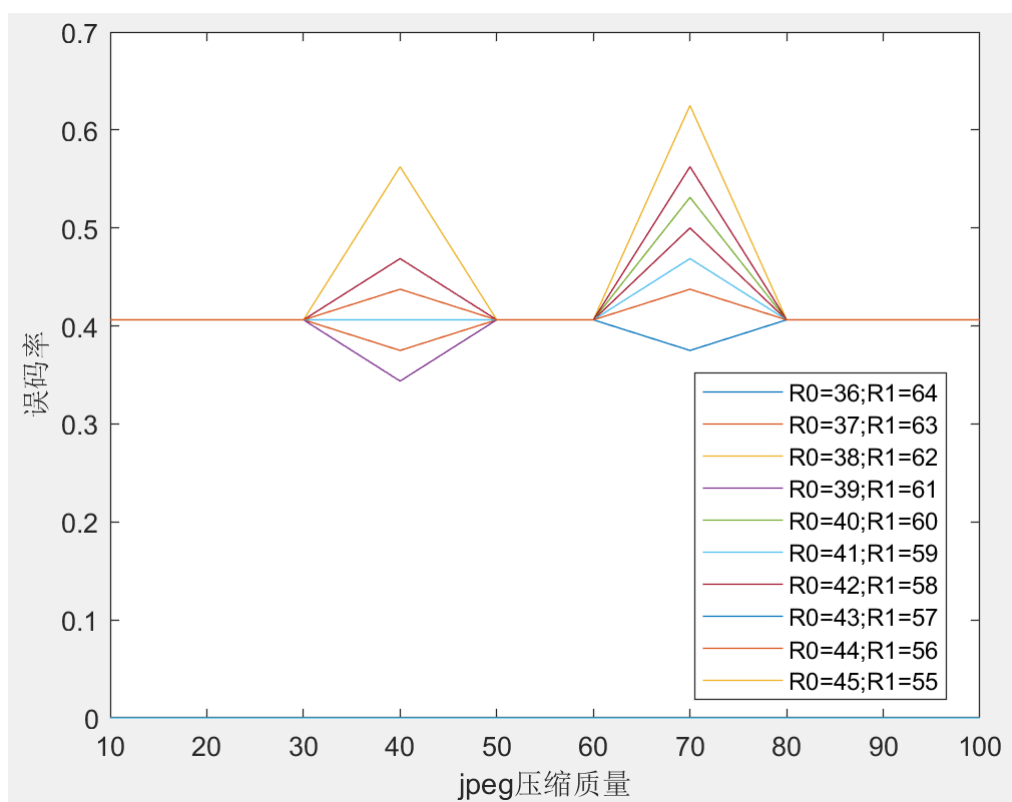


图 23：R0R1 对二值隐写鲁棒性的影响

此处遍历了 R1 从 36 逐步增加至 45，R2 由 64 逐步递减至 55。

从图中可以看到，在控制 λ 取值不变的情况下，在压缩质量为 40%与 70% 时，不同的 R0 R1 取值会出现不同的情况，虽然在此处并不一定 R0 值越小，误码率越大，但可以看到 R0 值小于等于 40 时往往会出现更高的误码率,其中当 R0=38,R1=62 时整体误码率最高；而当 R0=39,R1=61 或 R0=41,R1=59 时误码率相对较低，具有较强的鲁棒性。

这也证明了 R0 R1 的取值并不满足单调性，需要根据 λ 的取值、载体图像的情况动态的考虑其取值。

4.4 随机加噪的情况下， λ 的不同取值对算法鲁棒性的影响

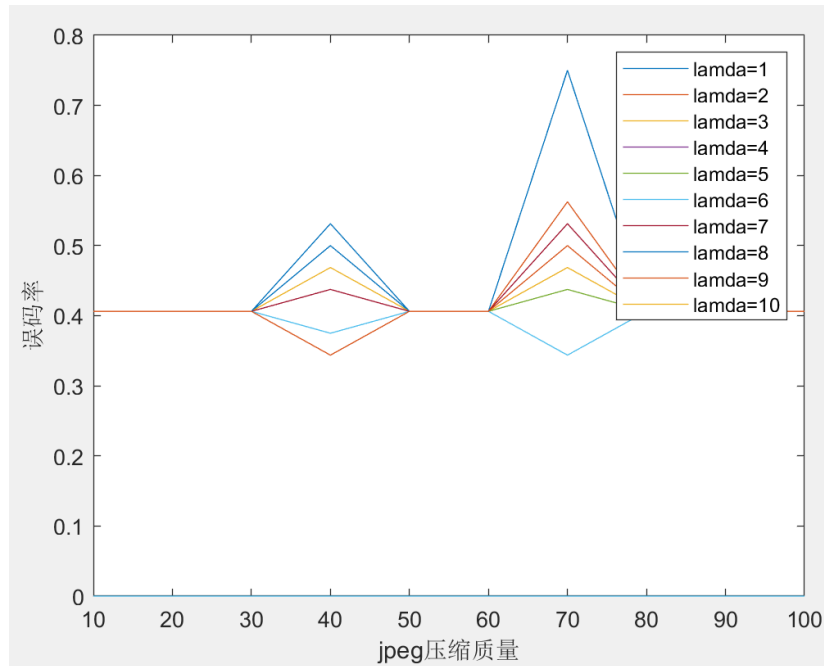


图 24: λ 对二值隐写鲁棒性的影响

此处遍历了 λ 从 1 逐步增加至 10。

从图中可以看到 λ 的不同取值仍在压缩质量为 40% 与 70% 时会呈现出更大的差异性。其中 λ 等于 6 的时候表现出较低的误码率，具有较强的鲁棒性；而在 λ 等于 1 时的误码率相对较高；而且即便时 $\lambda=10$ 时，其误码率依旧高于 $\lambda=6$ ，这可能是因为 λ 值较大时每个分块所需修改的像素点的个数较多，而分块中满足八连接的像素点不足，就导致大量的分块修改失败，从而使得最终的信息隐藏效果不好。

通过上面的讨论，可知 λ 的最优取值并不是可取范围的最大或最小值，往往时考虑了 R_0, R_1 取值后的一个中间值。

八、傅里叶变换

按照任老师的上课要求，自己在课下观看了李永乐老师讲述的傅里叶变换，在此作为记录。

首先是傅里叶级数，曾经一直把傅里叶级数和傅里叶变换混为一谈，现在明白了傅里叶级数是针对周期性信号而言的。一个周期性时域信号可以由若干个正（余）弦信号所组成，而对于这些正（余）弦信号，每个信号即三个部分：**频率、振幅、相位**；因此一个时域信号可以转化到时域上，便于我们进行分析。

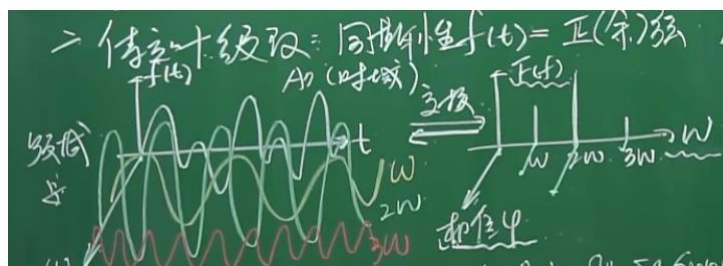


图 25：傅里叶级数说明图

然而大部分所需要处理的信号往往是非周期的，而这种非周期的表现其实也可以视为周期为无穷大；对于这种非周期信号的处理便引入了傅里叶变换。

傅里叶变换借助欧拉公式，“逆时针扫描信号”并进行积分，可以将一个时域信号转化为由无穷多个正（余）弦信号组成，如图 25 所示。

横坐标代表频率，纵坐标为此频率对应的正弦波的振幅， z 坐标为该正弦波的相位。即一个非周期性的时域信号被转化到了频域上。

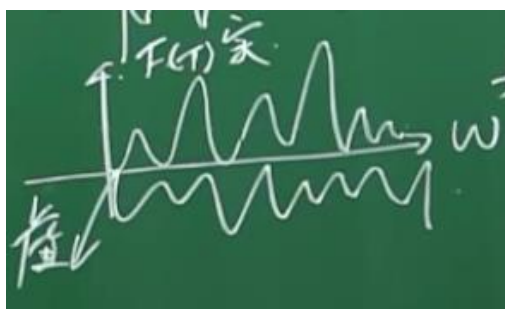


图 26：傅里叶变换说明图

关于傅里叶变换的应用首先是**声音的处理**。一个声波在时域上的表现并不能给出很直观的信息，而对该声波进行傅里叶变换将其转换到频域上信息就比较明显了；例如在超市的声波进行转换可以得到低频信息、中频信息、高频信息，而一般低频信息对应男生的声音、中频信息对应女生的声音、高频信息对应环境噪声。对应这样一段音频，如果我们想要突出男生的声音，就能够很方便的进行滤波，即将中高频的信息的振幅置为 0，就能完成上面的目的。

同样在图像领域依然能够利用傅里叶变化，一张图片在时域上的表现即不同的空间位置对应着不同的像素值大小，例如一张灰度图即表现为一个矩阵。经过傅里叶变换之后能够得到一个矩阵，该矩阵包含了原始图片的低频成分（代表轮廓），高频成分（代表细节）。对于一张人脸图片如果想要进行磨皮的操作，便可以将其进行傅里叶变换，去除其中的高频部分，再经过逆傅里叶变换即可得到上述目的。

九、总结及心得体会

(1) 通过此次实验我掌握了 LSB 隐写方法，能够将目标信息隐藏于图片之中并能够从中成功提取出来。

但在实验过程中觉得关于 LSB 的值对效应分析仍有不足，考虑到是载体图像较大 (512×512)，而待隐藏的信息 (32bit) 较少。可能当嵌入信息的比特较多时 (达到 2×10^5 以上数量级) 才能够显著的看出 LSB 隐写的值对效应。

这也从另一个方面反应了，当嵌入信息较少时，LSB 具有极高的隐蔽性。

(2) 此次实验中熟悉了 DCT 变换，能够在频域上对图像进行分析与处理。以往接触的是图像的时域表示，对于频域较为陌生，通过此次实验加强了频域的理解，频域也只不过是图像的另一种表现形式。

同时学习了图片进行频域表示时的优点，能够更方便的进行滤波、图像细节提取、信息隐藏等技术的实现。

(3) 在最后一个实验中了解了 md5 加密算法。通常只是把 md5 作为一种数据映射的工具，没有想到还可以利用它来进行非重复点的随机生成。并且在学习过程中学习了 md5 的具体过程。

(4) 掌握了图像的不同格式下的表现形式，如 RGB，二值，灰度等。也明确了不同表现形式下图像的特点，以及能够进行的信息隐藏技术。

(5) 实验 4 中学习了八连通与四连通，预防边界扩散的技术。各种细节处的处理让我对信息隐藏有了更全面的认识，即细节既是成功隐藏、提取信息的关键，同时也是防止各种检测方法探测的关键。

(6) 在实验 4 中体会到了二值隐藏的方法，但也发现了此算法的局限性，即对超参数选择较为敏感与可隐藏的信息长度有限。在实验结果分析中，可以看到此算法对于 R_0, R_1, λ 的取值不能一概而论，需要进行具体的实验分析；同时在实验过程中我也测试了隐藏其他更长的信息，发现随着信息量的增加，在寻找可用块时便更容易出错，这也增大了调整块内像素点的难度；并且在此实验中，采用 10×10 的小块，整个图像被分为 51×51 个部分，每部分只能隐藏 1bit 的信息，故此方法的隐藏能力受限。

(7) 此次实验的代码量较大，通过亲手实验也提升了自己的代码能力。

(8) 了解了傅里叶变换的基本过程，能够借助其进行图像和音频处理。