浙江大学

课程名称：	多媒体安全
姓　　名：	庞懿非
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
学　　号：	3190104534
指导教师：	黄劲
完成时间：	2022 年 6 月 11 日

实验二：E_SIMPLE_8/D_SIMPLE_8 系统测试

一、实验目的

1. 理解 E_SIMPLE_8/D_SIMPLE_8 系统的基本原理，掌握简单的多位信息水印技术。

二、实验内容与要求

1. 实现 E_SIMPLE_8/D_SIMPLE_8 系统。

2. 设计一张水印，嵌入强度 $\alpha = \sqrt{8}$，使用该水印测试 E_SIMPLE_8/D_SIMPLE_8 系统应用于不同封面时的检测准确率，计算 False Positive/Negative Rate 和解码的准确率。要求封面数量不少于 40 张。False Positive/Negative Rate 的计算可以采取不同的原则。其中一种可以使用的原则是，预先设定一个固定的阈值，8 个检测值（detect value）中有 4 个超过了阈值，就认为存在水印，否则认为不存在水印。（也可以使用其他合理的原则，需要在报告中说明使用的是哪种原则）。准确率的计算，则是对确实添加了水印的图片，计算解码出来的信息的错误率。

3. 设计不少于 40 张不同的水印，使用固定的嵌入强度 $\alpha = \sqrt{8}$，测试 E_SIMPLE_8/D_SIMPLE_8 系统应用于同一封面时的检测准确率，计算 False Positive/Negative Rate。

4. 分析信息长度增加对检测准确率的影响。

三、实验环境

1.**IDE**: MATLAB, Eclipse（Java）

2.**OS**: Windows 10

四、实验过程

0.背景与思路介绍

这个实验的原理其实并不难，利用了码分复用编码的方式，将 8bit 信息用 8 个不同的水印传输，由于图片的向量空间维度非常高，通过随机生成的 8 个水印基本上是互相正交的，这样也满足了码分复用编码的条件。在实验中我直接用高斯随机数生成函数一次性生成了 8 个 512*512 的水印，每个图片也提前处理成了 512*512 大小。

在实验第一部分，我对每个水印的检测值都进行了统计并且设定了阈值。在实验第二部分，我使用了一个统一的阈值来对所有的水印检测值进行判断。很多情况下，我们会发现水印与图片的线性相关检测值并不以 0 位中心，有一些小的偏移，但这并不会影响我们的实验结果，具体的过程如下所示。

1. 设计水印

以 512×512 为图片分辨率，使用高斯分布函数一次性生成等大小的 8 个水印，并且将 8 个水印的数值按照字符串存储在 hex 文件中。由于高斯分布（正态分布）生成的数值为随机浮点数，小数点后位数可能过长，所以我统一使用 4 位小数存储在文件中作为水印，之后直接从文件中读取浮点数作为水印信息。

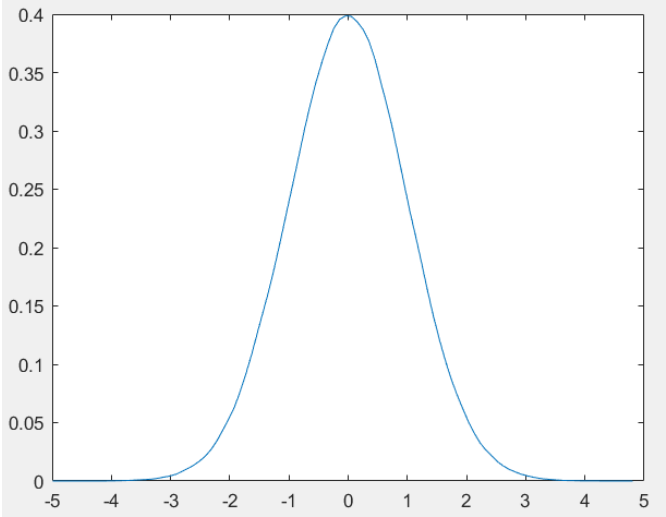
```
Random r = new Random();
(int) (r.nextGaussian() * 1);
```

在实际情况中，嵌入端和检测端不共享文件，通常是通过一个共享的种子（即密钥）作为随机数生成器然后生成出相同的水印编解码。在这里我们统一利用了一个共享的水印文件，可以方便查看水印的分布情况（将水印可视化），在利用水印信息的时候也更加便捷。

将水印可视化的过程如下所示：

```
dat1=load('wr1.hex');
x=dat1(:);
x=reshape(x,8,512*512);
% tabulate(x(2,:));
x1 = sort(x(1,:));
[f,xi]=ksdensity(x1);
plot(xi,f);
[mu12,sigma12] = normfit(x1);
```

将某个水印文件加载并且转换为 8*(512*512)的二维矩阵，之后对其中某一个水印绘制曲线分析；该水印文件的第 1 个水印的概率密度曲线如下所示：



如图所示，该水印几乎良好的符合高斯分布曲线，使用正态分布的估计，均值约为-6e-04，标准差为 0.999，符合要求。

2. 嵌入水印，生成加水印的图片

在嵌入水印时我们统一使用 512×512 分辨率的图片，与水印信息相匹配。在提供的数据集中选择分辨率满足要求的图片。同时，可以使用 windows 的图片工具把分辨率不满足要求的图片调整至我们想要的分辨率，尽管这可能会造成一些失真。

不难发现，在 E_SIMPLE_8/D_SIMPLE_8 中，嵌入水印信息仍然是使用了 E_BLIND 的方法，唯一的区别就是，本系统中一次性传输 8bit 信息，要对 8 个水印进行混合。由于每个水印的嵌入都是使用公式 $\boldsymbol{c_w = c_o + \alpha c_m}$，即使用加法将水印加入到原图中，所以这里我们先根据要传输的信息，将所有的水印提前进行合成，之后生成一个新的水印嵌入到图片中。

水印合成的部分很简单。如果传输的 bit 为 1，则加该水印，为 0 则减该水印；最后将得到的水印结果除以一个标准差，使得总水印的方差为 1。

```
public static void getCompWr(int []message, int length, double []ori_wrs,double []result){
    //according to the message
    //calculate the compound watermarks to result
    //calculate the mean of the result and the variance of the result
    //devide the standard variance, make the new watermark have unit variance
}
```

嵌入水印的过程很简单，使用原图片像素 + $\sqrt{8}$ * 水印值（使用$\sqrt{8}$是因为在前面将合成水印方差变为 1 的过程中，合成水印基本由正交的 8 个单位方差的向量组成，总方差大约为 8，除以标准差之后每个水印的值都相对上只是原来的$1/\sqrt{8}$，这里将 α 的值取作$\sqrt{8}$，等价于将每个水印的强度恢复到 1，即和实验一中 E_BLIND 具有相同的水印嵌入强度）。

当然在这一过程中，如果新的像素值不在［0,255］的区间，我们要把值 snapping 到 0 或者 255 的有效范围内，最后的结果要四舍五入到整数生成图片。在实验中，由于提供的数据全部是灰度图片，即 RGB 三元素的值都相等，所以我们在获取灰度值时可以直接选取 RGB 中的任意一个，这点在对图像信息进行读取之后可以看出。

在使用生成的一个水印（使用文件“a.hex”中存储的水印）对多张图片嵌入的时候，我使用了遍历文件夹的方式，对“data”文件夹中的所有图片分别加入了 0，255 以及 101（01100101）的信息，生成了 120 张图片存储到了对应的文件夹中。

```
public static void addWrToPics(int[]message, int length) throws IOException {

    double []wr_data = new double[8*512*512];
    watermarking.readFromFile("a.hex",wr_data);
    double [] result = new double [512*512];
    getCompWr(message,length,wr_data,result);

    File file = new File("data");
    File flist[] = file.listFiles();
    if (flist == null || flist.length == 0) {
        return ;
    }

    for (File f : flist) {
        if (f.isDirectory()) {
            //这里将列出所有的文件夹
            System.out.println("Dir==>" + f.getPath());
        } else {
            //这里将列出所有的文件
            String imagePath = f.getPath();
            System.out.println("file==>" + imagePath);
            createNewPic(message, length, imagePath, result);
        }
    }
}
```

其余方法，包括读取图片信息，使用新的像素值生成文件，将水印嵌入图片等，和实验一完全一致，这里不予赘述。

3. 计算线性相关性z_{lc}

按照公式，线性相关性的计算本质就是点积和/像素数目，如下图所示。

$$z_{lc}(\mathbf{c}, \mathbf{w_r}) = \frac{1}{N}\mathbf{c} \cdot \mathbf{w_r} = \frac{1}{N}\sum_{x,y} \mathbf{c}[x,y]\,\mathbf{w_r}[x,y],$$

具体实现上，只需要读取图片的像素数据与文件中 8 个水印的数据分别点积，然后除以像素数即可。

```java
public static double calculation(int [][]pixel_data, double []wr_data) {

    int width = pixel_data[0].length;
    int size = pixel_data.length * width;
    double correlation = 0;
    double sum = 0;
    for(int i = 0; i < size ; i++) {
        sum += pixel_data[i / width][i % width] * wr_data[i] ;
    }
    correlation = sum / size;
    return correlation;
}
```

有关线性相关的内容非常简单，直接进行计算即可。

4. 设计 40 张不同的水印

按照标准高斯分布曲线，生成 40 张水印，存入“wrs”文件夹中，分别命名为“wr0.hex”-“wr39.hex”；之后调用函数用这 40 张水印嵌入信息 101 到“lena512.bmp”中生成 40 张水印图。

```java
public static void addWrsToPic(int[]message, int length) throws IOException {

    double []wr_data = new double[8*512*512];
    double [] result = new double [512*512];
    int number = 0;

    File file = new File("wrs");
    File flist[] = file.listFiles();
    if (flist == null || flist.length == 0) {
        return ;
    }

    for (File f : flist) {
        if (f.isDirectory()) {
            //这里将列出所有的文件夹
            System.out.println("Dir==>" + f.getPath());
        } else {
            //这里将列出所有的文件

            String wrPath = f.getPath();
            System.out.println("file==>" + wrPath);
            watermarking.readFromFile(wrPath,wr_data);
            getCompWr(message,length,wr_data,result);
            createNewPic(message, length, "lena512.BMP", result,number);
            number ++;
        }
    }
}
```

注意：*文件的遍历顺序并不是从 0 到 39，是按照字符 ASCII 码的排序遍历的，对应图片的生成顺序与文件遍历顺序一致（图片生成顺序与图片标号对应），具体的遍历顺序可见文件“wrread sequence”。*

5.阈值设定和性能统计（数据分析）

通过对第一部分的 40 张原图以及 120 张生成图片的线性相关值合成曲线进行拟合，选取合适的阈值，计算 False Positive/Negative Rate 和解码的准确率。在第二部分中，由于没有每一个水印的大量信息，我们直接将所有水印的检测结果绘制曲线，并且设定一个通用的阈值来进行判断和检测，最后计算 False Positive/Negative Rate 和解码的准确率。

五、实验分析与结论

part1

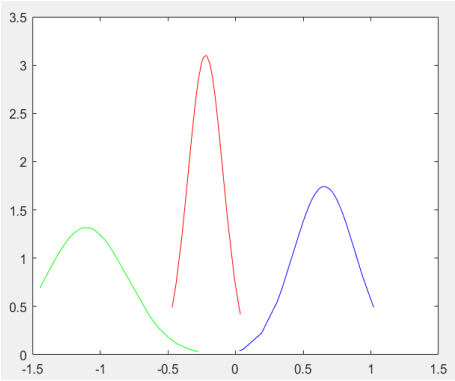
水印的生成与水印的性质检验在上一章节已经详细介绍过了，在这里我使用生成的水印按照传输信息 0，255，101 生成了 120 张图片，如下所示：

4 张图片几乎完全没有视觉上的差异，分别为加入水印信息 0 的 Lena 图（左上），加入水印信息 255 的 Lena 图（右上），加入水印信息 101 的 Lena 图（左下），未修改的 Lena 图（右下）。4 张图与水印的线性相关性检测值如下所示：

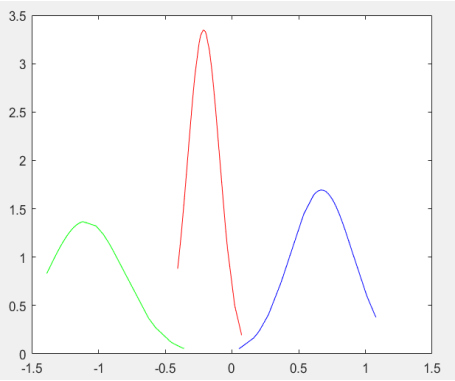
图片\水印\z_{lc}	wr0	wr1	wr2	wr3	wr4	wr5	wr6	wr7
lena512.bmp	-0.14185	-0.15393	-0.18743	-0.23749	-0.18193	0.31545	-0.26145	-0.27811
wrm0_lena512.bmp	-1.13433	-1.15222	-1.18386	-1.23268	-1.17501	-0.69501	-1.25921	-1.28736
wrm255_lena512.bmp	0.85062	0.84454	0.80865	0.7577	0.81102	1.32529	0.73635	0.73102
wrm101_lena512.bmp	-1.14397	0.85271	0.82915	-1.2251	-1.17966	1.30844	-1.2572	0.72368

明显可以看出，4 张图片对于 8 个水印的线性相关检测值不同，尽管这些图片看起来完全没有差别。类似的，其他图片也都按照加入的水印信息存储在相应的文件夹中，其余图片的线性相关数据也都存储在“**record.xlsx**”文件中（具体请看文档 Readme）。

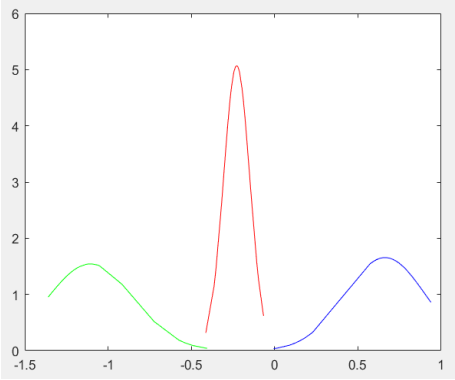
在第一部分，我对 160 张图片（120 张生成的加水印图与 40 张原图）关于 8 个水印的线性相关性检测值进行统计和正态分布模拟，得到如下的 8 个结果图：



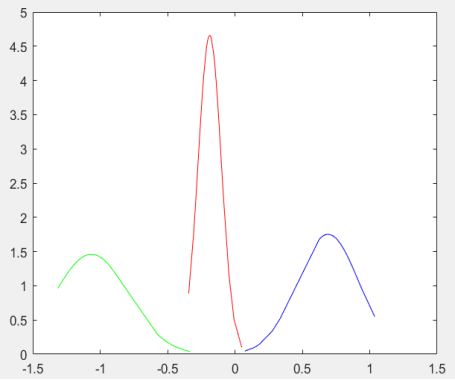
wr0



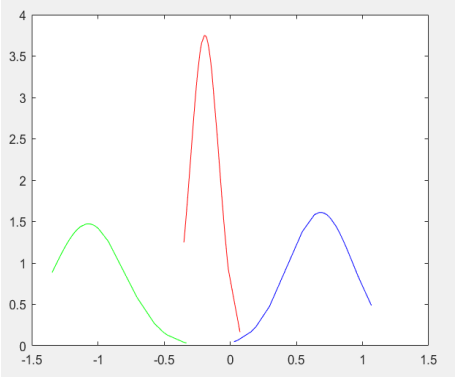
wr1



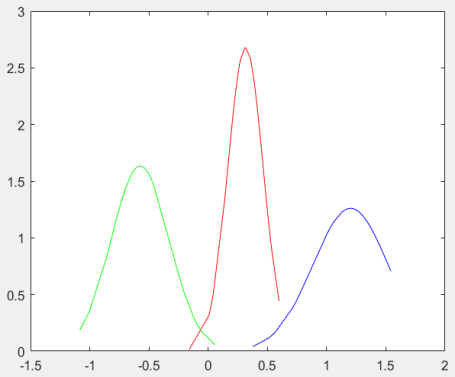
wr2



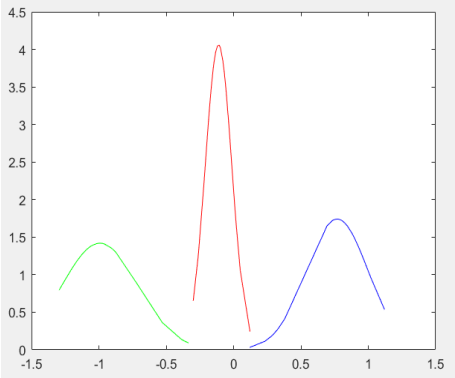
wr3



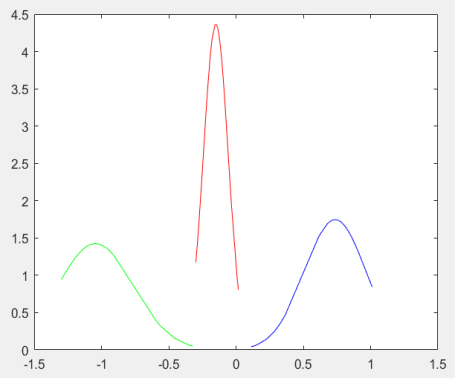
wr4



wr5



Wr6



wr7

一般情况下，我们认为需要取一个阈值 τ_{lc} 判断水印的存在性。

这里我们明显可以看到，无水印信息的正态模拟曲线均值不为 0，对于一张像素取值范围在[0,255]的图片，与一个正态分布、单位方差的水印的点积/像素个数很可能不为 0，略微有偏移也是正常现象，因为我们只使用了 40 张图片，所以拟合的结果可能与书上并不完全一致，但也是符合原理的。

经过对 8 张模拟曲线图进行综合分析，并且结合概率分布曲线，我选择 0.5 作为阈值 τ_{lc}。即，某个水印的线性相关性低于-0.5 或者高于 0.5 则认为是有水印信息，其他情况认为无水印。

水印图片的检测定义：图片的 8 个检测值（detect value）中有 4 个超过了阈值，就认为存在水印，否则认为不存在水印。

水印信息的定义：该水印与图片的线性相关性检测值为正，则认为该 bit 为含有信息 1，否则认为该 bit 位含有信息 0。

False Positive Rate:

使用如下公式在 Excel 中对线性相关性检测值进行判断：

```
= OR(C3 <$M$1,C3 > $N$1) + OR(D3 <$M$1,D3 > $N$1) + OR(E3 <$M$1,E3 > $N$1) +
OR(F3 <$M$1,F3 > $N$1) + OR(G3 <$M$1,G3 > $N$1) + OR(H3 <$M$1,H3 > $N$1) + OR(I3
<$M$1,I3 > $N$1) + OR(J3 <$M$1,J3 > $N$1)
```

计算结果为每个图片超过阈值的检测值的个数。（统计结果在“record.xlsx”的 L 列）

之后通过上值与 4 比较，判断该图片是否有含有水印。（统计结果在“record.xlsx”的 M 列）

最终，40 张原始图片都被认为没有添加水印，即假阳性率为 **0%**。

假阳性的含义是：没有水印的图片被检测出有水印，如此可见这个概率非常小。

False Negative Rate:

利用“record.xlsx”的 M 列的统计数据，我们发现在 120 张嵌入水印的图片中，一共有 16 张图片被检测为没有水印，这是一个相当大的比例，假阴性率达到了 13.3%。出现错误的图片全是 jpg 图片。由于加入的高频水印信号在 jpg 存储的压缩中很容易损失掉，所以加水印的效果非常差。不过这样差的假阴性性质主要是由于有损压缩导致的，并不能反映算法的真正性质，这也从另一方面提示我们不要对 jpg 图片修改单个像素值存储信息，在后续课程中我们也讲到了有关的知识，对 jpg 图像的水印或者隐写通常是修改 DCT 系数来完成。

解码的准确率：

解码的信息通过如下公式在 Excel 中计算：

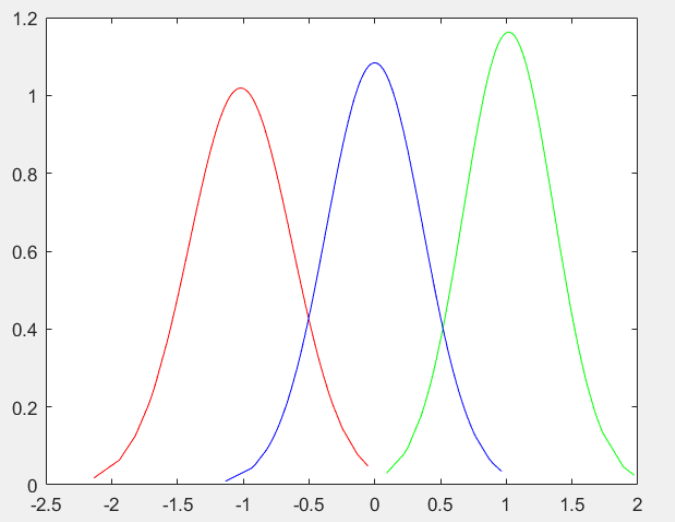
```
= (C3 > 0) * 128 + (D3 > 0) * 64 + (E3 > 0) * 32 + (F3 > 0) * 16 + (G3 > 0) * 8
+ (H3 > 0) * 4 + (I3 > 0) * 2 + (J3 > 0) * 1
```

计算结果为根据检测信息的定义计算的信息值。（统计结果在“record.xlsx”的 N 列）

在所有被检测出有水印的图片中，解码的信息和编码的信息完全一致。最终解码的准确率（正确解码的概率）为 86.7%。未加水印的图片的信息值则千变万化，这也体现出设定阈值的重要性。

part2

使用“lena512.bmp”为封面，加入信息 101 后生成 40 张水印图片，每组水印和原图以及自己生成的水印图进行线性相关性检测，我们将所有 40*8 个水印的线性相关检测值进行统计和正态分布模拟，得到的结果如下所示：



由于不同的水印之间存在较大的差异，最后的曲线的重叠部分也很大。

经过对比分析，并且结合概率分布曲线进行分析，我选择 0.65 作为阈值。即，线性相关性低于-0.65 或者高于 0.65 则认为是有水印信息，其他情况认为无水印。

False Positive Rate:

与 part1 的实验类似，这里我们同样相同的公式，唯一的区别就是阈值不同。

最终，对于 40 组水印，原图“lena512.bmp”都被认为没有添加水印信息，假阳性率为 0%。

False Negative Rate:

利用“record.xlsx”的 M 列的统计数据，我们发现在 40 张嵌入水印信息 101 的图片中，所有的图片都被认为添加了水印，假阴性率也为 0%。这也从另一方面显示了使用 bmp 图片对像素值加水印信息的效果还是相当不错的。

解码的准确率:

解码的信息也使用 part1 中完全相同的公式计算，在所有被检测出有水印的图片中，解码的信息和编码的信息完全一致。最终解码的准确率也为 100%。未加水印的图片的检测值

则千变万化，这体现出设定阈值的重要性。

part3

信息长度增加对检测准确率的影响：

一般情况，使用图片的 media space 添加水印，水印的数量（维度）远远小于图片的维度，随机生成的水印在 N 维（N 为图片像素个数）是无关的，在一定范围里增加水印的数量并不会造成线性相关检测的干扰。

但是，水印的数量越多，每个像素点可能受到的扰动就越大，就有越多的点可能会超出边界从而强制被 snapping，这样也就会增大误差，更可能出现检测错误。

同时，水印的要求是 imperceptible，加入过多的水印信息会导致图片大量变形，对 fidelity 造成了很大的影响，也是不可接受的。

综上，在 E_SIMPLE_8 / D_SIMPLE_8 系统中，增加信息的长度会一定程度上降低检测的准确率。在后续的课程中我们会使用其他方法以一种较高的准确率传输更多更长的信息串。

总结

E_SIMPLE_8/D_SIMPLE_8 将原始传输 1bit 的 E_BLIND/D_LC 进行了一个升级，可以一次性传输 8 个 bit 信息，利用了 8 个相互正交的水印。但实际上，码分复用编码的功能远远不限于此，只需要将每组编码之间正交，就可以传递出很多组的信息。最简单的实现方法就是将一个图片分成很多块，每个块传递一定的信息，整张图片加起来就可以传输非常多的信息（最明显的如 LSB 算法）。

在实验中选取阈值其实是一件比较复杂的事情，我在处理上使用了简单的观测法，实际上可以通过绘制 ROC 曲线来选取一个更好的点来实现。另外，由于每组实验中也只是使用了 40 张图片或者 40 组水印进行嵌入，数量还远远不够，使用更多的数据量可能会得到更好的实验结果。

六、实验感想

这个实验基于实验一进一步设计，只需要做一些改动。在实验一中我将水印值都取为整数，这点我认为是不恰当的。取整的过程会使得我的水印分布曲线变形，而取整操作本身只是为了能将像素值存储到图片中，可以被认为是一种信道噪声噪声的损失。所以在此实验中我使用浮点数作为水印值，在生成图片之前再取整。但是在水印的存储上，我为了操作简单直接将浮点数按照字符串进行存储，这导致每次读取水印都要花费大量的时间，对水印的操作变得非常慢，这一点可能后续需要进行改进。

此外，在这次实验中，我使用对文件的操作使得程序可以批量化的处理文件，相比实验一来说，这样的处理可以节省大量的时间和精力。在数据分析方面，我仍然是使用了MATLAB 模拟正态分布曲线进行分析和结果可视化，取得了不错的结果。