浙江大学

课程名称:	多媒体安全
姓　　名:	
学　　院:	计算机学院
专　　业:	计算机科学与技术
学　　号:	
指导老师:	黄劲
完成时间:	2023 年 5 月 13 日

实验三：E_BLK_8/D_BLK_8 系统测试

一、实验目的

1. 了解 E_BLK_8/D_BLK_8 系统的基本原理。
2. 了解 Hamming Code 和 Trellis Code 的工作原理。
3. 掌握 Correlation Coefficient 的计算。

二、实验内容与要求

1. 实现基于 E_SIMPLE_8/D_SIMPLE_8 系统的 E_BLK_8/D_BLK_8 系统。要求使用 Correlation Coefficient 作为检测值。
2. 设计一张水印，选择嵌入强度 $\alpha = \sqrt{8}$，使用该水印测试基于 E_SIMPLE_8/D_SIMPLE_8 系统的 E_BLK_8/D_BLK_8 系统应用于不同封面时的检测准确率。要求封面数量不少于 40 张。
3. 实现基于 Hamming Code 或 Trellis Code 的 E_BLK_8/D_BLK_8 系统。
4. 使用固定的水印和固定的嵌入强度，测试基于 Hamming Code 或 Trellis Code 的 E_BLK_8/D_BLK_8 系统应用于不同封面时的检测准确率。这里　取值根据所采用的 Hamming Code 或 Trellis Code 编码方式选定。比较在信息末尾添加两个 0 比特是否有助于提高检测的准确率，如果可以，请解释原因。
5. 比较基于不同系统，E_SIMPLE_8/D_SIMPLE_8 和（基于 Hamming Code 或 Trellis Code 的)E_BLK_8/D_BLK_8 系统的检测准确率，试分析原因。

三、实验环境

语言版本：MATLAB R2020b

四、实验过程

4.1 E_BLK 水印生成和 D_BLK 检测

此部分主要是实现 8*8 水印模式的生成和对应的 D_CC 检测。

由于多位 E_BLK 水印生成的原理基本上一致，我们让需要生成的水印张数 len 作为一个水印系统的参数。利用这个参数，我们采取均值为 0、方差为 1 的正态分布随机出 len 种 8*8 水印模式。为

保证这些水印合成之后仍然是一个均值为 0、方差为 1 的矩阵，我们把每一个存储的水印模式除以 len，在需要植入时，把 len 个水印模式简单相加即可得到方差为 1 的标准水印。

定义函数 E_BLK() 根据需要的水印张数 len 和随机数种子 seed 返回水印模式 watermark（len 页），实现如下:

```
function [watermark] = E_BLK(len, seed)

    rng(seed);

    siz = 8;
    watermark = zeros([siz, siz, len]);
    for i =1 : len
        watermark(:, :, i) = randn(siz, siz) / len;
    end
end
```

为了实现 E_BLK 系统，还需要有提取封面的 8*8 特征向量的过程。我们采取把原图分为 8*8 小块之后累加，最后除以小块数量的方式得到这个特征向量，可以认为是对原图求一个 8*8 的平均值。

定义函数 getMark() 根据图像矩阵 image 得到上述 8*8 特征向量，实现如下:

```
function [mark] = getMark(image)

    [sizeX, sizeY] = size(image);
    pageX = floor(sizeX / 8);
    pageY = floor(sizeY / 8);

    mark = zeros([8, 8]);
    for i = 1 : pageX
        for j = 1 : pageY
            mark = mark + image((i * 8 - 7) : (i * 8), (j * 8 - 7) : (j * 8));
        end
    end
    mark = mark / pageX / pageY;
end
```

生成的水印需要植入封面。鉴于该水印系统的特点，将 len 张水印嵌入原图像的每个 8*8 小块即

可。在植入的信息以外，另需要一个水印强度参数 α 控制水印的鲁棒性。

定义函数 embed() 根据封面 cover，水印模式 watermark，植入信息 infm，水印强度 alpha 返回含水印的封面 markedWork，实现如下:

```
function [markedWork] = embed(cover, watermark, infm, alpha)

    [sizeX, sizeY] = size(cover);
    pageX = floor(sizeX / 8);
    pageY = floor(sizeY / 8);
    len = length(infm);

    markedWork = cover;
    for i = 1 : pageX
        for j = 1 : pageY
            cur = markedWork((i * 8 - 7) : (i * 8), (j * 8 - 7) : (j * 8));
            for k = 1 : len
                cur = cur + watermark(:, :, k) * (infm(k) * 2 - 1) * alpha;
            end
            markedWork((i * 8 - 7) : (i * 8), (j * 8 - 7) : (j * 8)) = cur;
        end
    end
end
```

对于 D_BLK 检测系统，我们采取 D_CC 检测方法实现。D_CC 的检测原理是计算图片向量与水印向量的相关系数，如有明显的绝对值，则认为水印存在。定量的检测函数是:

$$z_{cc}(c, w) = c' * w'$$

其中 c' 和 w' 分别是 c 和 w 归一化后的向量。

如果 z_{cc} 的绝对值大于人为设定的检测阈值 τ_{cc}，则认为图片中存在水印，并根据 z_{cc} 的符号判定是正或负水印。否则，认为未植入水印。

在此实验中，由于不需要判断有无水印，我们直接让 $\tau_{cc} = 0$，以达到最好的检测效果。

定义函数 D_CC() 根据待检测模式 img 和水印模式 watermark 返回水印判定值，实现如下:

```
function [result] = D_CC(img, watermark)

    [sizeX, sizeY] = size(img);
    watermark = watermark(1 : sizeX, 1 : sizeY);
```

```
    result = correlation(img, watermark);
end
```

D_BLK 解码器也需要水印张数信息 len 来进行解码，其实现如下:

```
function [infm] = D_BLK(len, cover, watermark)

    mark = getMark(cover);

    infm = zeros([1, len]);
    for i = 1 : len
        infm(i) = D_CC(mark, watermark(:, :, i));
        infm(i) = (sign(infm(i)) + 1) / 2;
    end
end
```

以上已经实现 E_BLK/D_BLK 系统的大部分工作，只需人为给出所需实验数据的判定逻辑即可。

4.2 E_BLK_8 检测

设定随机数种子 seed 为 19260817，取给定数据集中所有图片（共 241 张）作为测试封面，植入随机 8bit 信息 infm，并进行检测。

```
watermark = E_BLK(8, 19260817);
infm = randi(2, [1, 8]) - 1;
er = 0;
for i = 1 : fileCnt
    image = imread(fullfile('data', files(i).name));
    image = cast(image, "double");

    markedImage = embed(image, watermark, infm, sqrt(8));
    res = D_BLK(8, markedImage, watermark);

    flag = 0;
    for j = 1 : 8
        if res(j) ~= infm(j)
            flag = 1;
```

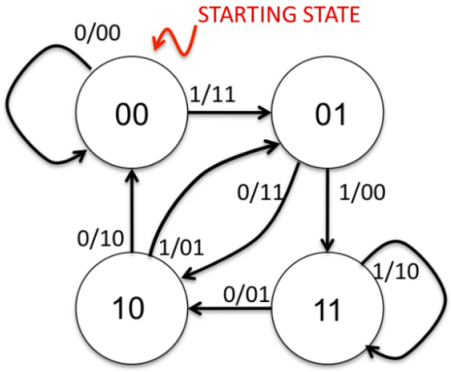
```
        end
    end
    er = er + flag;
end
```

所得检测结果为: 准确率 98.34%。

4.3 E_TRELLIS_8 检测

设定随机数种子 seed 为 19260817，取给定数据集中所有图片（共 241 张）作为测试封面，植入随机 8bit 信息 infm。该 8bit 信息需要经过 Trellis Encoder 转换成 2bit 卷积码，所以共需要植入 16bit 信息。

2bit Trellis Encoder 共有四个状态，其状态转移根据如下的状态机决定:



每读取一个字节信息，Encoder 输出箭头上的 2bit，同时状态转移到箭头指向的状态。起始状态为 00。

定义函数 Trellis_2() 根据待编码信息 msg 返回编码后比特序列 out，实现如下:

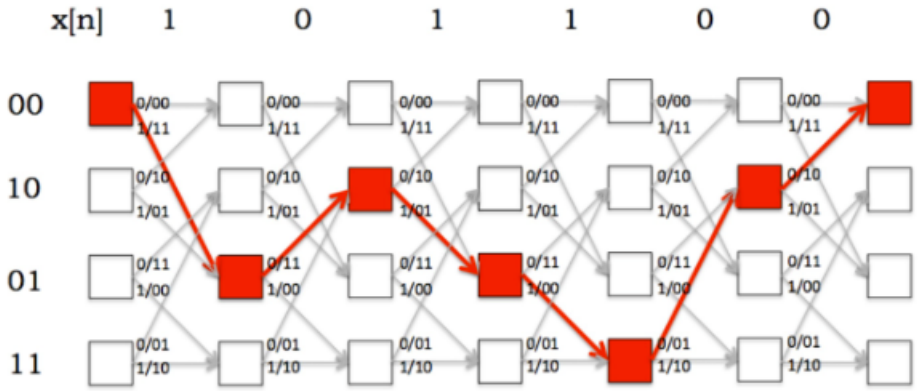
```
function [out] = Trellis_2(msg)

    state = 1;
    transfer = [1, 3, 1, 3, 2, 4, 2, 4];
    msgOut = [[0, 0], [1, 1], [1, 0], [0, 1], [1, 1], [0, 0], [0, 1], [1, 0]];

    len = length(msg);
    out = zeros([1, 2 * len]);
    for i = 1 : len
        out((i * 2 - 1) : (i * 2)) =
          msgOut((state * 4 + msg(i) * 2 - 3) : (state * 4 + msg(i) * 2 - 2));
```

```
        state = transfer(state * 2 + msg(i) - 1);
    end
end
```

为把 16bit 解码信息还原到 8bit 原信息，我们需要编写 Viterbi Decoder。Decoder 采用 DP 原理，让所有可能解码路径中错误率最小的一条路径代表的码字序列作为解码的信息。一个 Viterbi 解码的例子如下图:



定义函数 Viterbi_2() 根据编码信息 msg 返回解码信息 out，实现如下:

```
function [out] = Viterbi_2(msg)

    transfer = [1, 3, 1, 3, 2, 4, 2, 4];
    msgOut = [[0, 0], [1, 1], [1, 0], [0, 1], [1, 1], [0, 0], [0, 1], [1, 0]];

    len = length(msg);
    dp = repmat(len + 1, [len / 2 + 1, 4]);
    from = zeros([len / 2, 4]);
    dp(1, 1) = 0;

    for i = 1 : len / 2
        for state = 1 : 4
            res = dp(i, state) + sum(abs(
              msgOut((state * 4 - 3) : (state * 4 - 2))
              - msg((i * 2 - 1) : (i * 2))));
            if res < dp(i + 1, transfer(state * 2 - 1))
                dp(i + 1, transfer(state * 2 - 1)) = res;
                from(i, transfer(state * 2 - 1)) = state;
```

```
            end
            res = dp(i, state) + sum(abs(
              msgOut((state * 4 - 1) : (state * 4))
              - msg((i * 2 - 1) : (i * 2))));
            if res < dp(i + 1, transfer(state * 2))
                dp(i + 1, transfer(state * 2)) = res;
                from(i, transfer(state * 2)) = state;
            end
        end
    end

    loc = 1;
    for i = 1 : 4
        if dp(len / 2 + 1, i) < dp(len / 2 + 1, loc)
            loc = i;
        end
    end

    out = zeros([1, len / 2]);
    for i = len / 2 : -1 : 1
        out(i) = sum(loc > 2);
        loc = from(i, loc);
    end
end
```

通过以上定义的解编码器，我们可以进行卷积码水印的植入和检测（由于共有 16 张水印，水印强度为 $\sqrt{16}$）:

```
watermark = E_BLK(16, 19260817);
infmOrg = randi(2, [1, 8]) - 1;
infm = Trellis_2(infmOrg);
er = 0;
for i = 1 : fileCnt
    image = imread(fullfile('data', files(i).name));
    image = cast(image, "double");
```

```
8
9       markedImage = embed(image, watermark, infm, sqrt(16));
10      res = D_BLK(16, markedImage, watermark);
11      res = Viterbi_2(res);
12
13      flag = 0;
14      for j = 1 : 8
15          if res(j) ~= infmOrg(j)
16              flag = 1;
17          end
18      end
19      er = er + flag;
20  end
21
22  er = er / fileCnt
```

所得检测结果为: 准确率 98.34%。

4.4 添加 2 个 0 比特后的卷积码检测

我们在待植入信息末尾添加两个 0bit, 把原 8bit 信息扩增到 10bit, 然后进行 Trellis 编码和 Viterbi 解码。植入和检测过程如下（此时共有 20 张水印, 故水印强度为 $\sqrt{20}$）:

```
1   watermark = E_BLK(20, 19260817);
2   infmOrg = [randi(2, [1, 8]) - 1, 0, 0];
3   infm = Trellis_2(infmOrg);
4   er = 0;
5   for i = 1 : fileCnt
6       image = imread(fullfile('data', files(i).name));
7       image = cast(image, "double");
8
9       markedImage = embed(image, watermark, infm, sqrt(20));
10      res = D_BLK(20, markedImage, watermark);
11      res = Viterbi_2(res);
12
13      flag = 0;
```

```
    for j = 1 : 8
        if res(j) ~= infmOrg(j)
            flag = 1;
        end
    end
    er = er + flag;
end

er = er / fileCnt
```

所得检测结果为: 准确率 98.76%。

五、实验分析与结论

5.1 卷积码末尾补 0 检测准确率分析

单纯进行 Trellis 编码和 Viterbi 解码，比起在信息末尾添加两个 0 的准确率要略低。

通过在卷积码末位补 0，我们在解码时就有 2bit 的已知信息，能够对之前卷积累积的误差进行一定程度的修正，降低了错误路径保存到最后解码的概率。

同时，在末尾补 2 个已知的 0 相当于把 Viterbi 解码的所有可能结果全部收束到状态 1，在单个状态就能够简单且准确地判断出通过哪条路径传播的误差最小。

从而，信息末尾补 0 确实提高了水印的检测效率。

5.2 E_SIMPLE_8 和卷积码的检测准确率比较

在本次实验中，通过 E_SIMPLE_8 系统植入 8*8 水印的检测准确率和单纯卷积码的检测准确率差别不大，但低于末尾补 0 的卷积码检测准确率。

但我们知道，卷积码需要植入的水印张数比简单系统要多，从而单张水印的检测能力自然会下滑；但最终的检测准确率持平，可见，卷积码确实具有一定的纠错能力。而且，在本次实验中，卷积码系统通过一定的再纠错手段，可以做到比简单系统更高的准确率。

同时，我们可以想到，卷积码提高纠错能力，但降低单张水印的检测准确率。从而，选取适当的卷积码表示和位数，对优化卷积码水印系统的检测准确率尤为重要。

六、实验感想

我一开始是写的 1bit->4bit Encode（16 状态），从而不太理解为什么末位是加两个 0 而不是四个 0 让状态归零。这种情况下加两个零对检测准确率的影响极低，也不能得到较好的实验结果。之后仔细一想才想到是用 2bit Encode（4 状态）才能恰好归零，也让我对卷积码的灵活性有了更好的认识。

通过本实验，我更加深入地理解了多信息位水印和检测的方法及原理，也学习到了纠错码在数字水印中的应用，有助于后续对课程的学习。