

实验三： E_BLK_8/D_BLK_8系统测试

王睿 3180103650

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 Coefffficient 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 Coefffficient 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 α 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

openCV 4.4.0

四. 实验过程

4.1 实现基于E_BLK_8/D_BLK_8系统

4.1.1 E_BLK_8嵌入器实现

E_BLK_8嵌入器的实现过程简要如下：

首先，从未添加水印的图像 c_o 中提取出标记 v_o ；然后在标记空间中选择一个新向量 v_w ，使得 v_w 满足在尽可能接近 v_o 的同时处于检测区域内；最后，E_BLK_8将 v_w 映射到图像空间中，得到带有水印的作品 c_w 。

接下来逐步分析上述过程

- 从未添加水印的原始图像 c_o 中提取标记 v_o

为了从图像中提取标记，我们首先将图像分割成相等的 8×8 个block，然后将每个block叠加并取平均值得到mark标记 v_o ：

$$v[i, j] = \frac{1}{B} \sum_{x=0}^{\frac{width}{8}} \sum_{y=0}^{\frac{height}{8}} c[8x + i, 8y + j]$$

上式中， B 为分割出的block数量，width和height分别是未添加水印的图像 c_o 的宽和高。

- 在标记空间中选择一个新向量 v_w ，满足一定要求

v_w 的生成过程如下：

根据消息每一位的值，生成相应8x8的水印：

$$w_{mi} = \begin{cases} w_{ri} & \text{if } m[i] = 1 \\ -w_{ri} & \text{if } m[i] = 0 \end{cases}$$

然后将八个水印相加，合成成一张水印，并对其进行归一化处理，得到最后的水印 w_m 。

$$w_{tmp} = \sum_i w_{mi}$$
$$w_m = \frac{w_{tmp}}{s_{w_{tmp}}}$$

在嵌入水印中，message pattern w_m 由输入参数 α 进行缩放，生成添加的pattern α 。因此，得到的mark标记向量如下：

$$w_a = \alpha w_m$$
$$v_w = v_o + w_a$$

- 将 v_m 映射到图像空间中，得到带有水印的作品 c_w

将 v_w 映射到图像空间的过程中，需要生成一个基于原图的图像 c_w ，这里采用简单的把8x8矩阵映射到原图大小的空间中，方法如下：

$$c_w[x, y] = c_o[x, y] + (v_w \cdot [x \bmod 8, y \bmod 8] - v_o \cdot [x \bmod 8, y \bmod 8])$$

这一步保证了当检测器对 c_w 应用提取函数时，结果是 v_w ，水印将被检测到。

具体的实现代码如下：

```
1  Mat E_BLK_8(Mat Co, vector<int>& m, float alpha, int seed)
2  {
3      Mat Wm(8, 8, CV_32FC1);
4      Co.convertTo(Co, CV_32FC1);
5
6      for (int i = 0; i < 8; i++) {
7          Mat Wr = generate_watermark(8, 8, seed + i);
8
9          //write_watermark(Wr, "Wr_E_" + to_string(seed + i));
10
11         if (i == 0)
12             Wm = (m[i] == 1 ? Wr : -Wr);
13         else {
14             Wm += (m[i] == 1 ? Wr : -Wr);
15         }
16     }
17
18     // normalization
19     Scalar mean, stddev;
20     meanStdDev(Wm, mean, stddev);
21     double meanVal = mean.val[0];
22     double stddevVal = stddev.val[0];
23
24     Wm = Wm - meanVal;
25     Wm = Wm / stddevVal;
26
27     Mat Vo = Mat::zeros(8, 8, CV_32FC1);
28     Mat n = Mat::zeros(8, 8, CV_32FC1);
29     for(int i = 0; i < Co.rows; i++){
30         for(int j = 0; j < Co.cols; j++){
31             int id_x = i % 8;
32             int id_y = j % 8;
33             Vo.at<float>(id_x, id_y) += Co.at<float>(i, j);
```

```

34         n.at<float>(id_x, id_y)++;
35     }
36
37     Vo = Vo / n;
38
39     Mat Vw(8, 8, CV_32FC1);
40     Vw = Vo + alpha * Wm;
41
42     Mat vo_1 = Mat::zeros(8, 8, CV_32FC1);
43     Mat n_1 = Mat::zeros(8, 8, CV_32FC1);
44     for(int i = 0; i < Co.rows; i++){
45         for(int j = 0; j < Co.cols; j++){
46             int id_x = i % 8;
47             int id_y = j % 8;
48             vo_1.at<float>(id_x, id_y) += Co.at<float>(i, j);
49             n_1.at<float>(id_x, id_y)++;
50         }
51     }
52
53     Mat delta(8, 8, CV_32FC1);
54     delta = n.mul(Vw) - n_1;
55
56     Mat Cw = Mat::zeros(Co.rows, Co.cols, CV_32FC1);
57     for(int i = 0; i < Cw.rows; i++){
58         for(int j = 0; j < Cw.cols; j++){
59             int id_x = i % 8;
60             int id_y = j % 8;
61             float oldVal = Co.at<float>(i, j);
62             float newVal = Co.at<float>(i, j) + (delta.at<float>(id_x, id_y)
/ n.at<float>(id_x, id_y));
63
64             newVal = (newVal > 255 ? 255 : (newVal < 0 ? 0 : newVal));
65
66             Co.at<float>(i, j) = newVal;
67             n.at<float>(id_x, id_y)--;
68             delta.at<float>(id_x, id_y) -= (Co.at<float>(i, j) - oldVal);
69         }
70     }
71
72     Cw = Co;
73
74     return Cw;
75 }

```

4.1.2 D_BLK_8检测器实现

D_BLK_8检测器的实现过程简要如下：

首先从添加了水印的 c_w 中提取出mark标记 v ；然后通过D_SIMPLE_8算法对提取出的mark进行检测，判断水印是否存在。

检测水印的过程中，D_BLK_8中运用的是相关系数检测，在相关系数检测前需要对矩阵进行减去均值处理，这是为了防止对某一矩阵进行数据加减影响检测值，同时，根据两个矩阵的大小对线性相关进行归一化处理，这样能防止对某一矩阵进行数据乘除影响检测值。这样的操作有很好的鲁棒性，能够抵抗图像亮度和对比度的变化。

相关系数检测定义如下：

$$z_{cc}(v, w_r) = \frac{\tilde{v} \cdot \tilde{w}_r}{\sqrt{(\tilde{v} \cdot \tilde{v})(\tilde{w}_r \cdot \tilde{w}_r)}}$$

其中 \tilde{v} 和 \tilde{w}_r 经过了减去均值操作，相关系数检测可以看作 \tilde{v} 和 \tilde{w}_r 归一化处理后的内积，这个可以简单理解为两个向量之间的夹角，也就是 \tilde{v} 和 \tilde{w}_r 之间的夹角，所以：

$$-1 \leq z_{cc}(v, w_r) \leq 1$$

于是和D_LC类似，检测结果设置阈值如下：

$$m_n = \begin{cases} 1 & \text{if } z_{cc}(v, w_r) > \tau_{cc} \\ \text{no watermark} & \text{if } -\tau_{cc} < z_{cc}(v, w_r) < \tau_{cc} \\ 0 & \text{if } z_{cc}(v, w_r) < -\tau_{cc} \end{cases}$$

因此，D_BLK_8代码如下：

```

1  vector<int> D_BLK_8(Mat Cw, int seed, float Tcc)
2  {
3      Cw.convertTo(Cw, CV_32FC1);
4
5      // extract mark
6      Mat v = Mat::zeros(8, 8, CV_32FC1);
7      Mat n = Mat::zeros(8, 8, CV_32FC1);
8
9      for(int i = 0; i < Cw.rows; i++)
10         for(int j = 0; j < Cw.cols; j++){
11             int id_x = i % 8;
12             int id_y = j % 8;
13             v.at<float>(id_x, id_y) += Cw.at<float>(i, j);
14             n.at<float>(id_x, id_y)++;
15         }
16
17         v = v / n;
18
19         // demodulation
20         Mat Wr(8, 8, CV_32FC1);
21         vector<int> m;
22
23         for (int i = 0; i < 8; i++) {
24             Wr = generate_watermark(8, 8, seed + i);
25             //write_watermark(Wr, "Wr_D_" + to_string(seed + i));
26
27             double val = 0;
28             val = sum(v.mul(Wr))[0];
29
30             double temp = val / (double)(8 * 8);
31
32             //cout << "m[" << i << "]: " << temp << endl;
33             if (temp > thresh)
34                 m.push_back(1);
35             else if (temp < -thresh)
36                 m.push_back(0);
37             else
38                 m.push_back(-1);
39         }
40
41         // modulate

```

```

42     Mat Wm = Mat::zeros(8, 8, CV_32FC1);
43     for(int i = 0; i < 8; i++){
44         Mat temp_W = generate_watermark(8, 8, seed + i);
45
46         if(m[i] == 1)
47             Wm += temp_W;
48         else
49             Wm -= temp_W;
50     }
51
52     // compute correlation coefficient
53     Scalar mean, stddev;
54     meanStdDev(v, mean, stddev);
55     float meanVal = mean.val[0];
56     v = v - meanVal;
57
58     meanStdDev(Wm, mean, stddev);
59     meanVal = mean.val[0];
60     Wm = Wm - meanVal;
61
62     double vwm = sum(v.mul(Wm))[0];
63     double vv = sum(v.mul(v))[0];
64     double WmWm = sum(Wm.mul(Wm))[0];
65     double cc = 0;
66
67     if(abs(vv * WmWm) < 0.0000001)
68         cc = 0;
69     else
70         cc = vwm / sqrt(vv * WmWm);
71
72     if(cc < Tcc){
73         m.clear();
74         for(int i = 0; i < 8; i++)
75             m.push_back(-1);
76     }
77     return m;
78 }
79

```

4.2 E_BLK_8/D_BLK_8系统检测不同封面

实验统计了data文件夹内的41张图，分别添加了8位水印与未添加任何水印，共82个检测样本。

False Positive/Negative Rate 的计算采用的原则是，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，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正确率。

最终检测结果如下：

```

false positive rate = 45.12%
false negative rate = 1.22%
The decode accuracy of this system is 60.67%

```

4.3 实现基于Trellis Code的E_BLK_8/D_BLK_8系统

4.3.1 E_BLK_8_Trellis嵌入器

Trellis Code本质是一种编码后的状态机，每读入一位输入，Trellis Code就会输出4bit数据，并转换状态。当输入位是0时，它从当前状态穿过细线并输出线上标记的四位数字。如果输入位是1，它将遍历粗线并输出线上标记的4位数字。因此，4位的信息经过编码后将被转换为16位。

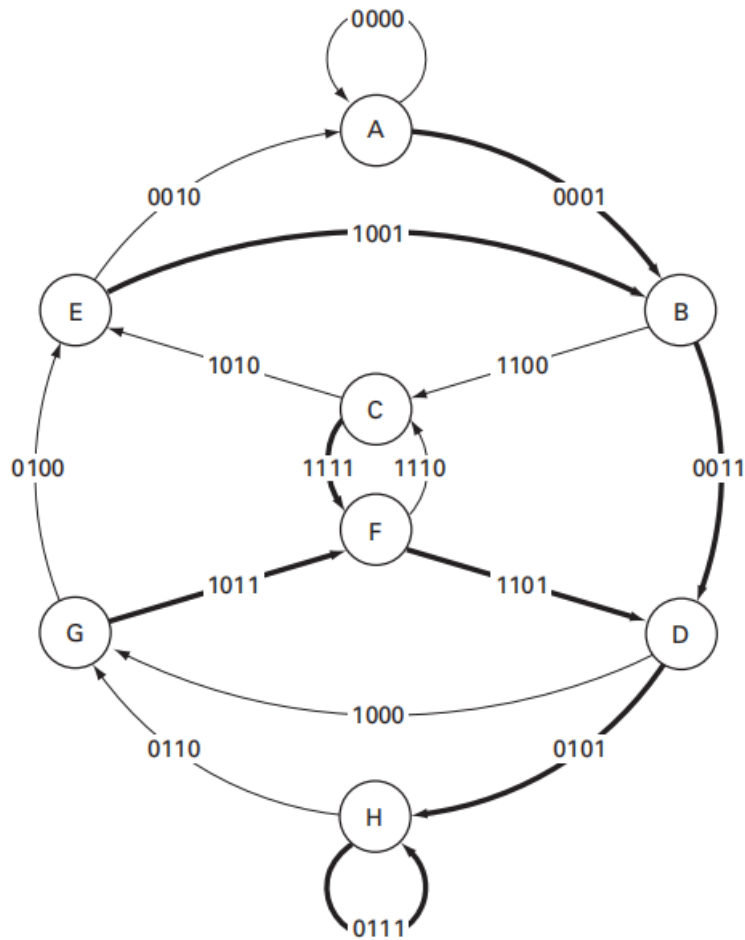


Fig. 4.8 An eight-state convolutional code.

具体代码如下，整体逻辑与上面无Trellis Code的相同：

```
1  Mat E_BLK_8_Trellis(Mat Co, vector<int>& m, float alpha, int seed)
2  {
3      Mat Wm(8, 8, CV_32FC1);
4      Co.convertTo(Co, CV_32FC1);
5      int current_state = 0;
6
7      for (int i = 0; i < 10; i++) {
8          Mat wr = generate_watermark(8, 8, seed + current_state * 10 + i -
9  1);
10         if (i < 8 && m[i] == 1) {
11             Wm += wr;
12             current_state = state[current_state][1];
13         }
14         else {
15             Wm -= wr;
16             current_state = state[current_state][0];
17         }
18     }
```

```

19 // normalization
20 scalar mean, stddev;
21 meanStdDev(Wm, mean, stddev);
22 double meanVal = mean.val[0];
23 double stddevVal = stddev.val[0];
24
25 Wm = Wm - meanVal;
26 Wm = Wm / stddevVal;
27
28 // extract mark
29 Mat Vo = Mat::zeros(8, 8, CV_32FC1);
30 Mat n = Mat::zeros(8, 8, CV_32FC1);
31 for (int i = 0; i < Co.rows; i++)
32     for (int j = 0; j < Co.cols; j++) {
33         int id_x = i % 8;
34         int id_y = j % 8;
35         Vo.at<float>(id_x, id_y) += Co.at<float>(i, j);
36         n.at<float>(id_x, id_y)++;
37     }
38
39 Vo = Vo / n;
40
41 // mix blind
42 Mat Vw(8, 8, CV_32FC1);
43 Vw = Vo + alpha * Wm;
44
45 // inv extract mark
46 Mat vo_1 = Mat::zeros(8, 8, CV_32FC1);
47 Mat n_1 = Mat::zeros(8, 8, CV_32FC1);
48 for (int i = 0; i < Co.rows; i++)
49     for (int j = 0; j < Co.cols; j++) {
50         int id_x = i % 8;
51         int id_y = j % 8;
52         vo_1.at<float>(id_x, id_y) += Co.at<float>(i, j);
53         n_1.at<float>(id_x, id_y)++;
54     }
55
56
57 Mat delta(8, 8, CV_32FC1);
58 delta = n_1.mul(Vw) - vo_1;
59
60 Mat Cw = Mat::zeros(Co.rows, Co.cols, CV_32FC1);
61 for (int i = 0; i < Cw.rows; i++) {
62     for (int j = 0; j < Cw.cols; j++) {
63         int id_x = i % 8;
64         int id_y = j % 8;
65         float oldVal = Co.at<float>(i, j);
66         float newVal = Co.at<float>(i, j) + (delta.at<float>(id_x, id_y)
/ n_1.at<float>(id_x, id_y));
67
68         newVal = (newVal > 255 ? 255 : (newVal < 0 ? 0 : newVal));
69
70         Co.at<float>(i, j) = newVal;
71         n_1.at<float>(id_x, id_y)--;
72         delta.at<float>(id_x, id_y) -= (Co.at<float>(i, j) - oldVal);
73     }
74 }
75

```

```

76     Cw = Co;
77
78     return Cw;
79 }

```

4.3.2 D_BLK_8-Trellis检测器

解码的过程就是找到通过最有可能的路径。最有可能的路径的特征是，接收到的向量与该路径的信息向量之间具有最高线性相关性或内积。

找到最有可能的路径就是通过Viterbi解码，为了解释解码算法，定义如下：

- v 是需要解码的向量
- $w_{i,j}$ 是从状态图上状态 i 到状态 j 的路径上的reference mark
- $p[A \dots H]$ 是八条路径的数组， $p[i]$ 是最有可能到状态 i 的路径
- $z[A \dots H]$ 是八个内积， $z[i]$ 是 v 和 $p[i]$ 这条路径上reference mark的内积

最开始， $p[A \dots H]$ 初始化为无路径， $z[A \dots H]$ 初始化为0，在第一次迭代中，计算 v 与从列0到列1的状态的转换相关的16个reference mark之间的内积。为了计算 v 与从列0的状态到列1的状态的路径之间的内积之和，新定义一个 z 变量，将与之对应转换路径的计算结果加对应的 z 上。

比较两条路径上的内积，选择内积大的那条路径，再继续进行计算。直到到达最后，得到的内积结果值是最高的，这条路径就是解码结果。

具体代码如下：

```

1  vector<int> D_BLK_8-Trellis(Mat Cw, int seed, float Tcc)
2  {
3      Mat wr(8, 8, CV_32FC1);
4      Cw.convertTo(Cw, CV_32FC1);
5
6      // extract mark
7      Mat v = Mat::zeros(8, 8, CV_32FC1);
8      Mat n = Mat::zeros(8, 8, CV_32FC1);
9
10     for (int i = 0; i < Cw.rows; i++)
11         for (int j = 0; j < Cw.cols; j++) {
12             int id_x = i % 8;
13             int id_y = j % 8;
14             v.at<float>(id_x, id_y) += Cw.at<float>(i, j);
15             n.at<float>(id_x, id_y)++;
16         }
17
18     v = v / n;
19
20     // demodulation
21     int lc0[8] = { 0 };
22
23     for (int i = 0; i < 8; i++)
24         lc0[i]--;
25     lc0[1] = 0;
26
27     int m0[8][8] = { 0 };
28     int m1[8][8] = { 0 };
29
30
31     for (int i = 0; i < 10; i++) {
32         int lc1[8] = { 0 };

```



```

33     for (int j = 0; j < 8; j++)
34         lc1[j]--;
35
36     for (int current_state = 0; current_state < 8; current_state++) {
37         if (lc0[current_state] != -1) {
38             wr = generate_watermark(8, 8, seed + current_state * 10 + i
- 1);
39             float lc = sum(v.mul(wr))[0] / (float)(8 * 8);
40             int next_state = state[current_state][0];
41             if (lc1[next_state] == -1 || lc1[next_state] <
lc0[current_state] - lc) {
42                 lc1[next_state] = lc0[current_state] - lc;
43                 for (int j = 0; j < 8; j++) {
44                     m1[next_state][j] = m0[current_state][j];
45                 }
46             }
47             if (i < 8) {
48                 next_state = state[current_state][1];
49                 if (lc1[next_state] == -1 || lc1[next_state] <
lc0[current_state] + lc) {
50                     lc1[next_state] = lc0[current_state] + lc;
51                     for (int j = 0; j < 8; j++) {
52                         m1[next_state][j] = m0[current_state][j];
53                     }
54                     m1[next_state][i] = 1;
55                 }
56             }
57         }
58     }
59
60     for (int j = 0; j < 8; j++) {
61         lc0[j] = lc1[j];
62         for (int k = 0; k < 8; k++) {
63             m0[j][k] = m1[j][k];
64         }
65     }
66
67 }
68
69 int bestState = 0;
70 for (int i = 1; i < 8; i++) {
71     if (lc0[i] > lc0[bestState])
72         bestState = i;
73 }
74
75 vector<int> m_best(8, 0);
76 for (int i = 0; i < 8; i++) {
77     m_best[i] = m0[bestState][i];
78 }
79
80 // modulate
81 Mat Wm = Mat::zeros(8, 8, CV_32FC1);
82 int current_state = 0;
83
84 for (int i = 0; i < 10; i++) {
85     Wr = generate_watermark(8, 8, seed + current_state * 10 + i - 1);
86     if (i < 8 && m_best[i] == 1) {
87         Wm += Wr;

```

```

88         current_state = state[current_state][1];
89     }
90     else {
91         wm -= wr;
92         current_state = state[current_state][0];
93     }
94 }
95
96 // compute correlation coefficient
97 scalar mean, stddev;
98 meanStdDev(v, mean, stddev);
99 float meanVal = mean.val[0];
100 v = v - meanVal;
101
102 meanStdDev(wm, mean, stddev);
103 meanVal = mean.val[0];
104 wm = wm - meanVal;
105
106 double vwm = sum(v.mul(wm))[0];
107 double vv = sum(v.mul(v))[0];
108 double wmw = sum(wm.mul(wm))[0];
109 double cc = 0;
110
111 if (abs(vv * wmw) < 0.0000001)
112     cc = 0;
113 else
114     cc = abs(vwm) / sqrt(abs(vv * wmw));
115
116 if (cc < Tcc) {
117     for (int i = 0; i < 8; i++)
118         m_best[i] = 2;
119 }
120 return m_best;
121 }

```

4.4 水印检测系统应用于不同封面以及PADDING对比

- m = [0, 0, 0, 0, 0, 0, 1, 0]

```

m = [0, 0, 0, 0, 0, 0, 1, 0]
Without Padding zeros:
    false positive rate: 0.00%
    false negative rate: 0.00%
    system decode accuracy: 50.00%
With Padding zeros:
    false positive rate: 0.00%
    false negative rate: 0.00%
    system decode accuracy: 100.00%

```

- m = [1, 0, 1, 0, 1, 0, 1, 0]

```

m = [1, 0, 1, 0, 1, 0, 1, 0]
Without Padding zeros:
    false positive rate: 1.26%
    false negative rate: 0.00%
    system decode accuracy: 48.75%
With Padding zeros:
    false positive rate: 0.00%
    false negative rate: 0.00%
    system decode accuracy: 100.00%

```

- $m = [1, 1, 1, 1, 1, 1, 1, 1]$

```
m = [1, 1, 1, 1, 1, 1, 1, 1]
Without Padding zeros:
    false positive rate: 0.00%
    false negative rate: 0.00%
    system decode accuracy: 50.00%
With Padding zeros:
    false positive rate: 0.00%
    false negative rate: 0.00%
    system decode accuracy: 100.00%
```

之所以在信息末尾添加两个0比特后，检测准确率能明显上升，是因为在信息末尾增加两个比特后，最后需要计算内积的路径扩展到了十位，其中最后两位是0，所以在构造 w_m 时，最后两个水印都是 $-w_r$ 。由于这是已知条件，在最后计算内积时，正确的路径上的最后两位能够确保是正确的，也就是正确的路径上，计算最后两位内积时得到的结果必然是最大的。

如果前面8位数据中有解码错误偏离了正确路径，比如路径A前八位的结果是 z_A ，路径B前八位的结果是 z_B ，正确结果是路径A，但是 $z_B > z_A$ ，这种情况下，计算最后两位的内积 z_{A2} 和 z_{B2} ，由于路径A的最后两位是正确的，所以 $z_{A2} > z_{B2}$ ，这样能够消除A和B路径之间的微小差距，不会让程序误以为路径B是正确的。

因此，添加两位信息后，检测结果正确率得到很大的提升。

4.5 比较E_SIMPLER_8/D_SIMPLE_8与E_BLK_8/D_BLK_8的检测准确率

```
E_SIMPLE_8/D_SIMPLE_8:
    false positive rate: 0.00%
    false negative rate: 7.56%
    system decode accuracy: 52.83%
E_BLK_8/D_BLK_8:
    false positive rate: 0.00%
    false negative rate: 3.79%
    system decode accuracy: 80.58%
```

可以看到，添加了Trellis Code的E_BLK_8/D_BLK_8的系统检测准确率远高于E_SIMPLE_8/D_SIMPLE_8的系统检测准确率。因为Trellis Code添加了两比特的信息，其作用相当于纠正码，能够在一定程度上校正解码错误。

五. 实验感想

本次实验难度比较大，水印算法的原理就比前两次实验复杂，且实现起来也有一定难度。通过本次实验，我对E_BLK_8, Trellis Code有了更加很深刻的认识，获益匪浅。