南開大學

信息隐藏技术 课程实验报告

奇偶校验位隐藏法

学院　网络空间安全学院

专业　信息安全

姓名　齐明杰

学号　2113997

2024 年 4 月 18 日

目　　录

图表

1 实验目的

奇偶校验位隐藏法

隐藏：利用奇偶校验位隐藏法，实现将秘密图像嵌入到位图中；

提取：将秘密图像提取出来。

2 实验原理

奇偶校验位隐藏法是一种隐写技术，通过对载体图像的最低有效位（LSB）进行微小修改，以嵌入秘密信息。这种方法通过改变图像像素的 LSB，来控制每个像素或像素区域的奇偶校验位，从而隐藏信息。人类视觉对这种细微的像素级变化不敏感，使得这种方法在不显著改变图像外观的情况下，能有效隐藏信息。

奇偶校验位隐藏法的两种方法

方法一：全区域翻转

信息嵌入过程:

- 载体图像划分：将载体图像划分为 $L(m)$ 个不相重叠的区域 $R_1, R_2, ..., R_{L(m)}$，每个区域用于嵌入一位信息。
- 奇偶校验位计算：对于每个区域 R_i，计算区域内所有像素的 LSB 的奇偶校验位 $b_i = \left(\sum_{j \in R_i} \mathrm{LSB}(c_j)\right) \bmod 2$

其中，c_j 表示区域 R_i 中第 j 个像素的颜色值。

- 信息嵌入：对于每个区域 R_i，将秘密信息的比特 m_i 嵌入。如果 $b_i \neq m_i$ ，则翻转该区域中所有像素的 LSB，以使得新的奇偶校验位 b_i' 等于 m_i。

信息提取过程: 使用与信息嵌入时相同的区域划分策略，计算每个区域的奇偶校验位。 这些奇偶校验位即构成了隐藏的秘密信息。

方法二：单像素翻转

信息嵌入过程:

- 载体图像划分：与方法一相同，首先将载体图像划分为多个区域。
- 奇偶校验位计算：对每个区域 R_i，计算奇偶校验位 $b_i = \left(\sum_{j \in R_i} \mathrm{LSB}(c_j)\right) \bmod 2$
- 信息嵌入:如果区域的奇偶校验位 b_i 与要嵌入的信息比特 m_i 不匹配，选择该区域内的一个像素，翻转其 LSB。这通常足以改变整个区域的奇偶校验位，使 b_i' 与 m_i 匹配。

信息提取过程: 与信息嵌入时使用相同的区域划分策略，提取每个区域的奇偶校验位。 连接这些奇偶校验位以恢复嵌入的秘密信息。

以上两种方法均通过精确控制图像像素的 LSB 来隐藏信息，方法一通过翻转整个区域的 LSB 来确保信息的准确嵌入，而方法二则通过单个像素的 LSB 翻转来最小化对图像的影响。这些技术不仅提供了一种有效的信息隐藏手段，也保持了图像的视觉质量。

3 实验过程

3.1 实验图像

本次实验仍然使用 bmp 位图，无论是原图还是用于隐藏的图片。我沿用上次实验选用的图片如下：

图 3.1.1: Lena 图像

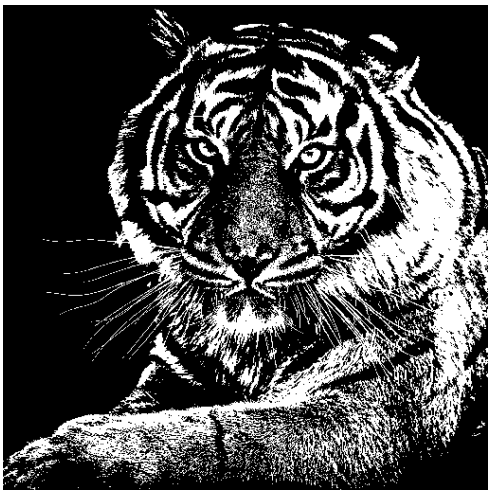

图 3.1.2: 一只凶猛的老虎

在事先，我已经将两个图片从 png 格式转成了 bmp 位图格式:

lady.bmp hide.bmp

另外要嵌入的图片需要转化为**二值图像**，如果事先不准备二值图片也可以，只需要在代码加入以下两行：

```matlab
y=rgb2gray(y);
y=imbinarize(y);
```

3.2 秘密图像嵌入到位图

由于方法二则通过单个像素的 LSB 翻转来最小化对图像的影响。这些技术不仅提供了一种有效的信息隐藏手段，也保持了图像的视觉质量。因此我本次选择使用单像素反转来进行。

函数的主体代码：

```matlab
function SteganographyDemo()
    carrierImage = imread('lady.bmp');
    watermarkImage = imread('hide.bmp');
    disp(['Carrier image dimensions: ', mat2str(size(carrierImage))]);
```

```
    disp(['Carrier image type: ', class(carrierImage)]);
    disp(['Watermark image dimensions: ', mat2str(size(watermarkImage))]);
    disp(['Watermark image type: ', class(watermarkImage)]);
    subplot(2, 2, 1);
    imshow(carrierImage); title('Original Image');
    subplot(2, 2, 2);
    imshow(watermarkImage); title('Watermark Image');
    [height, width] = size(watermarkImage);
    carrierImage =
    EmbedWatermark(carrierImage,height,width,watermarkImage);
    subplot(2, 2, 3);
    imshow(carrierImage, []); title('Steganographed Image');
    extractedImage = ExtractWatermark();
    subplot(2, 2, 4);
    imshow(extractedImage, []); title('Extracted Watermark');
end
```

解析

在 SteganographyDemo 函数中，我们的主要目标是展示如何将一个秘密图像（水印）嵌入到一个载体图像中，并随后从嵌入后的图像中提取出这个水印。首先，该函数读取两个图像文件：载体图像 lady.bmp 和要嵌入的水印图像 hide.bmp。通过读取这些图像，我们能获取它们的属性，包括尺寸和类型，这些信息通过 disp 函数打印出来，帮助理解图像处理过程中的基础数据。

接下来，使用 subplot 和 imshow 函数在一个图形窗口中展示这两张图像。这不仅验证了图像是否正确加载，还提供了一个直观的视觉对比，展示了原始载体图像和秘密水印图像的外观。此外，通过这种方式，可以清楚地看到隐写术前后图像的变化。

函数接下来调用 EmbedWatermark 来将水印嵌入载体图像中，并将结果显示在另一个子图中。最后，调用 ExtractWatermark 函数尝试从修改后的图像中恢复水印，并展示恢复结果。这整个过程不仅展示了隐写技术的效果，也验证了嵌入和提取过程的有效性，是对隐写技术理解的直观展示。

其中，用到了函数 calculateParity:

```matlab
function parity = calculateParity(imageMatrix, row, col)
    pixelData = zeros(1, 4);
    pixelData(1) = bitget(imageMatrix(2*row-1, 2*col-1), 1);
    pixelData(2) = bitget(imageMatrix(2*row-1, 2*col), 1);
    pixelData(3) = bitget(imageMatrix(2*row, 2*col-1), 1);
    pixelData(4) = bitget(imageMatrix(2*row, 2*col), 1);
    parity = mod(sum(pixelData), 2);
```

```
8 end
```

解析

函数 calculateParity 是隐写术中核心的一个环节，它用于计算指定区域内像素的最低有效位（Least Significant Bit, LSB）的奇偶性。这个计算是嵌入和提取秘密信息至关重要的一步，因为它决定了是否需要修改像素以匹配所嵌入的数据。

在此函数中，首先初始化一个一维数组 pixelData，用于存储选定像素区域的 LSB 值。该区域由函数的参数 row 和 col 指定，通常对应于水印图像中的一个像素所对应的载体图像中的一个小区域。例如，在隐写实验中，一个水印图像的像素可能对应载体图像中的 2x2 像素块。

函数详细地读取这四个像素（左上、右上、左下、右下）的 LSB，并将它们存储到 pixelData 数组中。这一操作通过 bitget 函数完成，该函数能够提取图像像素值的指定位。在此例中，我们关注的是每个像素值的第 1 位，即 LSB。

计算完这四个像素的 LSB 后，通过求和并使用模 2 运算（即 mod(sum(pixelData), 2)）计算出整个区域的 LSB 的奇偶性。这个结果（奇偶性）被用作决定该区域是否需要修改以匹配水印图像的相应位。如果奇偶性与水印图像的当前像素不匹配，就意味着需要调整至少一个像素的 LSB，以确保整个区域的 LSB 奇偶性与水印图像相匹配。

另外，还用到了 EmbedWatermark 函数：

```matlab
function modifiedImage = EmbedWatermark(carrier, rows, cols, watermark)
    for row = 1:rows
        for col = 1:cols
            if calculateParity(carrier, row, col) ~= watermark(row, col)
                pixel = int8(rand() * 3);
                switch pixel
                    case 0
                        carrier(2*row-1, 2*col-1) = bitset(carrier(2*row-1, 2*col-1), 1, ~bitget(carrier(2*row-1, 2*col-1), 1));
                    case 1
                        carrier(2*row-1, 2*col) = bitset(carrier(2*row-1, 2*col), 1, ~bitget(carrier(2*row-1, 2*col), 1));
                    case 2
                        carrier(2*row, 2*col-1) = bitset(carrier(2*row, 2*col-1), 1, ~bitget(carrier(2*row, 2*col-1), 1));
                    case 3
```

```
                carrier(2*row, 2*col) = bitset(carrier(2*row, 2*col), 1, ~bitget(carrier(2*row, 2*col), 1));
                end
            end
        end
    end
    imwrite(carrier, 'watermarkedImage.bmp');
    modifiedImage = carrier;
end
```

解析

函数 EmbedWatermark 的核心目的是将一个水印（秘密信息）嵌入到一个载体图像中。这个过程通过逐像素地检查水印图像与载体图像中相应区域的奇偶性是否一致来实现，并在必要时调整载体图像的像素，以保证奇偶性与水印图像匹配。

在这个函数中，首先使用两层嵌套的循环遍历水印图像的每一个像素。对于每个水印图像中的像素位置 (row, col)，它对应到载体图像中的一个 2x2 像素块。函数使用 calculateParity 来计算这个区域的奇偶性，并将其与水印图像中的相应像素（即秘密位）进行比较。

如果载体图像中的像素块的奇偶性与水印图像的对应像素不一致，这意味着需要修改载体图像中的至少一个像素的 LSB，以确保整个区域的 LSB 奇偶性与水印图像相匹配。修改哪一个像素是通过随机选择实现的，以提高安全性。这一选择是通过生成一个 0 到 3 的随机整数（pixel），并根据这个数值选择具体修改哪一个像素的 LSB。

修改像素值的 LSB 是通过 bitset 和 bitget 函数实现的。bitset 函数用于设置指定的位（在这里是最低位，即 LSB）到一个新的值，这个新值是当前 LSB 的逻辑非（bitget(…)）。这种修改只影响像素的最低位，对图像的视觉质量影响极小，从而使得水印的嵌入对于观察者来说几乎是不可察觉的。

最后，修改后的载体图像被保存为一个新的文件 watermarkedImage.bmp，并将这个修改后的图像返回，用于后续的展示或进一步处理。

3.3 提取秘密图像

此过程旨在从经过修改的载体图像中提取先前嵌入的水印图像。这一过程核心依赖于重新计算在嵌入阶段定义的相应区域的像素奇偶性，利用这些奇偶性值恢复水印图像的每个像素。

编写如下代码：

```matlab
function output = ExtractWatermark()
    watermarkedImage = imread('watermarkedImage.bmp');
    [height, width] = size(watermarkedImage);
    watermark = zeros(height/2, width/2);
    for row = 1:height/2
        for col = 1: width/2
            watermark(row, col) = calculateParity(watermarkedImage, row, col);
        end
    end
    output = watermark;
end
```

解析

在 ExtractWatermark 函数中，首先通过 imread 函数读取之前保存的含有水印的图像 watermarkedImage.bmp。这步骤是必须的，因为我们需要处理这个已经被修改过的图像文件来恢复水印信息。

接下来，函数通过 size 函数获取这个图像的尺寸（height 和 width），这些尺寸信息用于确定遍历图像时的循环限制。由于每个水印图像的像素在载体图像中对应一个 2x2 的像素块，所以水印图像的高度和宽度分别是载体图像的一半。因此，创建一个新的矩阵 watermark，其尺寸为 height/2 和 width/2，用于存储恢复的水印图像数据。

在恢复水印的过程中，ExtractWatermark 函数通过两层嵌套的循环（从 1 到 height/2 和从 1 到 width/2）遍历水印图像中的每一个像素。对于每个像素位置 (row, col)，函数调用 calculateParity 来计算对应载体图像中的 2x2 像素块的 LSB 奇偶性。得到的奇偶性值（0 或 1）直接决定了水印图像中相应像素的值。

最后，这个恢复的水印图像存储在 watermark 矩阵中，并将其作为输出返回。这样，水印图像就被成功地从修改过的载体图像中提取出来了，完成了隐写术的整个过程。

3.4 实验结果

将上述过程代码合并到一个代码文件中，运行后得到的结果如下：

图 3.4.3: 实验结果

其中几张图分别是:

- Original Image 原始图像
- Watermark Image 嵌入图像
- Steganographed Image 伪装图像
- Extracted Image 提取出的嵌入图像

可见我们嵌入的图像和提取出的图像基本相同，据此实验完成。

4 实验心得

在完成这次关于图像隐写术的实验后，我深刻体会到了数字图像处理和信息安全领域的交叉重要性。通过实现和测试图像中的秘密信息嵌入与提取技术，我不仅提升了对MATLAB 编程和图像处理技术的掌握，还对如何通过修改像素的最低有效位来隐蔽地传递信息有了直观的理解。此外，这次实验也强化了我的问题解决能力，尤其是在调试和优化代码的过程中，我学会了如何有效地分析和解决程序中出现的错误和效率问题。整个过程不仅是对技术技能的锻炼，也是对隐写术潜在应用的探索，为我未来在数据安全领域的学习和研究奠定了坚实的基础。