南　　开　　大　　学

网络空间安全学院

信息隐藏技术实验报告

实验 6：LSB 隐藏法

姓名：2113203 付政烨

年级：2021 级

专业：信息安全、法学

指导教师：李朝晖

2024 年 4 月 15 日

目录

一、 实验内容

(一) 实验目的

1. 实现将二值图像嵌入到位图中;
2. 实现将学号（一个整数）嵌入到位图中。

(二) 实验环境

- 运行系统：Windows11
- 实验工具：Matlab2022a
- 数据：PNG 格式图像

(三) 实验要求

- 在 MATLAB 中调试完成
- 编程实现，写出实验报告，含程序代码和截图，word/pdf 格式
- QQ 群提交作业

二、 实验原理

1. LSB 原理简介

最低有效位（LSB）方法是一种广泛使用的数字隐写技术，主要应用于数字图像和音频文件中。其核心思想是将待隐藏的秘密信息（通常为比特序列）替换进宿主文件中像素或采样值的最低有效位。这种方法的实现步骤简单，可以在不显著改变原始宿主文件视觉或听觉特性的前提下，嵌入大量的数据。

(1) 优点

- **简单性和易实现性**：LSB 方法由于其操作简单，易于在各种编程环境中实现。
- **高容量**：由于每个像素或样本至少包含一个可用于隐藏信息的位，因此这种方法能够嵌入较大量的数据，尤其适用于大尺寸的图像或长时间的音频文件。

(2) 缺点

- **安全性较低**：LSB 方法容易被简单的统计分析检测出来，尤其是在面对专门的隐写分析工具时。
- **抗噪声能力弱**：由于信息仅嵌入在最低位，任何形式的损害如加噪声、压缩等都可能导致信息的损坏或丢失。
- **脆弱性**：面对有损压缩（如 JPEG 压缩）等处理时，嵌入的信息容易被破坏。

(3) 安全性增强措施

- **信息加密**：在嵌入信息前对其进行加密处理，即使信息被提取，未经授权的用户也难以理解其真实含义。
- **多重嵌入**：通过在多个不同的位置重复嵌入相同信息，即使部分数据受损，整体信息仍然可以被完整恢复。
- **引入纠错编码技术**：在信息嵌入之前，先对其进行纠错编码，增加信息在面对干扰和损害时的抵抗力和恢复能力。

2. 流载体的 LSB 方法

- **嵌入**：选择一个载体元素的子集 $j_1, j_2 \ldots\ldots, j_{L(m)}$，其中共有 $L(m)$ 个元素，用以隐藏秘密信息的 $L(m)$ 个比特。然后在这个子集上执行替换操作，把 C_{j_i} 的最低比特用 m_i 来替换。
- **提取**：找到嵌入信息的伪装元素的子集 $j_1, j_2 \ldots\ldots, j_{L(m)}$，从这些伪装对象 S_{j_i} 中抽出它们的最低比特位，排列之后组成秘密信息。

3. LSB 方法的相关函数

1. 获取图像 x 的行数和列数 $[m, n] = size(x)$
2. 嵌入/提取图像 bit 位的值 $C = bitset(A, bit, v)/biget(y, 1)$;

三、 实验步骤

水印图像是二值图像，载体图像是与水印图像大小相同的 256 级灰度图像，进行 LSB 水印图像的嵌入和提取读取。述为该技术的基本操作步骤:

1. **信息编码**：将需要隐藏的秘密信息（水印图像）转换成二进制码序列。这一步骤是确保信息可以被逐位嵌入到载体图像中。
2. **图像预处理**：将载体图像的每一个像素点的灰度值转换为对应的二进制形式。这一转换是为了访问和修改这些像素点的最低有效位。
3. **数据嵌入**：将秘密信息的二进制码依次嵌入到载体图像像素点的最低有效位（LSB）中。此步骤需要精确操作，以确保每个像素的最低位都被正确地替换。
4. **图像重构**：在完成秘密信息的嵌入后，修改后的像素值将被重新组合，形成新的数字图像。此图像在视觉上应与原始图像相似，以确保水印的隐蔽性。
5. **信息提取**：在接收端，通过提取数字图像中每个像素的最低有效位，可以恢复出嵌入的秘密信息的二进制码。最后，这些二进制码将被转换回原始的水印信息。

(一) 将二值图像嵌入到位图中

1. 主函数：ImageHiding

主函数 ImageHiding 负责整个水印处理的流程控制。它首先读取两个图像文件：一个作为载体图像, 另一个作为水印图像。函数中使用 imread 函数分别读取”Lena.bmp” 和”lion.bmp”。接着, 使用 imshow 函数显示这两个图像。调用 Hide 函数将水印图像嵌入到载体图像中。这个函数的输

出是加水印后的图像 watermarkedImage。Hide 函数通过修改载体图像的像素的最低有效位来嵌入水印信息。接下来调用 Extract 函数从加水印的图像中提取水印，输出为 extractedWatermark。这个步骤验证了水印是否正确嵌入并能被成功提取。整个函数 ImageHiding 作为一个入口点，演示了从图像读取到水印嵌入，再到水印提取的完整流程，展示了图像处理中的水印技术的实际应用。

```
function ImageHiding()
    carrierImage = imread("Lena.bmp"); % 载体图像
    watermarkImage = imread("lion.bmp"); % 水印图像
    imshow(carrierImage,[]) % 显示载体图像
    imshow(watermarkImage,[]); % 显示水印图像
    watermarkedImage = Hide(carrierImage, watermarkImage); % 调用 Hide 函数进行水印嵌入
    extractedWatermark = Extract(watermarkedImage); % 调用 Extract 函数提取水印
end
```

2. 水印隐藏函数：Hide

函数 Hide 是一个核心的图像处理函数，其任务是将一个水印图像（通常是一个二值图像）嵌入到一个载体图像中，使用的是最低位（Least Significant Bit, LSB）替换技术。这种技术通过改变载体图像像素的最低有效位来隐藏水印，而不显著影响载体图像的视觉质量。

函数接收两个参数：originImage（原始载体图像）和 watermarkImage（水印图像）。使用 size 函数确定原始图像的尺寸（行和列），这是必需的，因为嵌入水印需要遍历每一个像素。创建一个与原始图像同样大小的 watermarkedImage，初始化为全零，之后将其转换为 uint8 类型，以便进行像素操作。使用双层 for 循环遍历原始图像的每个像素。对于每个像素，使用 bitset 函数将载体图像的最低位设置为水印图像的相应像素值。如果水印像素为 1，最低位设置为 1；如果为 0，则设置为 0。这种修改对于图像的视觉效果影响非常小，但可以有效地存储水印信息。使用 imwrite 函数将加水印后的图像保存到一个新的 BMP 文件'lsb_watermarked.bmp'。这一步确保了处理结果可以存档和进一步使用。使用 imshow 和 title 函数在 MATLAB 图形窗口中显示加水印后的图像。

```
function watermarkedImage = Hide(originImage, watermarkImage)
    [rows, cols] = size(originImage); % 获取图像的行和列数
    watermarkedImage = uint8(zeros(size(originImage))); % 初始化水印图像空间

    for i = 1:rows
        for j = 1:cols
            % 将水印图像的像素值嵌入载体图像的最低位
            watermarkedImage(i, j) = bitset(originImage(i, j), 1, watermarkImage(i, j));
        end
    end

    imwrite(watermarkedImage, 'lsb_watermarked.bmp', 'bmp'); % 保存加水印后的图像
    figure;
    imshow(watermarkedImage, []);
    title("Watermarked Image"); % 显示加水印后的图像
end
```

Hide 函数利用了数字图像处理中简单而有效的 LSB 技术，通过精巧地替换像素的最低位来隐藏

水印，而这种改变在视觉上几乎是不可察觉的。这种方法既保持了原图的完整性，又实现了信息的隐藏，是数字水印技术中的一种常见实践。

3. 水印提取函数：Extract

Extract 函数是专门用来从已嵌入水印的图像中提取水印信息的。此函数通过访问水印图像每个像素的最低位来恢复原始水印信息。这个过程是 Hide 函数的逆过程，验证了水印的隐藏是否成功，同时也展示了如何从修改过的图像中恢复隐藏的数据。

Extract 函数接收 watermarkedImage（加水印后的图像）作为输入参数。使用 size 函数获取输入图像的行和列，这是必要的步骤，因为水印提取需要遍历整个图像。初始化 extractedWatermark 数组，该数组将用来存放提取出的水印。数组的大小与输入图像相同，并且初始化为全零的 uint8 类型图像。使用双层 for 循环遍历加水印图像的每个像素。对每个像素，使用 bitget 函数提取其最低有效位。由于 Hide 函数是通过修改这一位来嵌入水印的，提取这一位即可恢复水印图像的对应像素。bitget 函数返回指定位的值（0 或 1），这个值直接被存储到 extractedWatermark 的相应位置，构成了提取后的水印图像。最后，使用 imshow 函数和 title 函数将提取出的水印图像显示在 MATLAB 图形窗口中。通过 Extract 函数，可以验证水印的嵌入质量和提取技术的有效性。

```
function extractedWatermark = Extract(watermarkedImage)
    [rows, cols] = size(watermarkedImage); % 获取图像的行和列数
    extractedWatermark = uint8(zeros(size(watermarkedImage))); % 初始化提取水印的图像空间

    for i = 1:rows
        for j = 1:cols
            % 提取每个像素的最低位信息作为水印信息
            extractedWatermark(i, j) = bitget(watermarkedImage(i, j), 1);
        end
    end

    figure;
    imshow(extractedWatermark, []);
    title("Extracted Watermark"); % 显示提取出的水印图像
end
```

4. 实验结果

图 1: Carrier Image

图 2: Watermark Image

图 3: Image with Watermark

图 4: Extracted Watermark

(二) 将学号 (整数) 嵌入到位图中

在数字图像处理中，信息隐藏技术的一个典型应用是将整数数据嵌入到图像中。整个过程不仅需要精确地操作和修改数字图像的数据，还必须保持图像的视觉不被明显破坏，以达到隐蔽性的目的。以下是实现此技术的详细步骤：

1. 首先，将目标整数转换为其对应的二进制表示形式。这一步骤是数据嵌入过程的基础，确保数据能够以适当的格式进行处理。
2. 读取作为载体的图像文件，并将其像素值转换为二进制码。这一步骤涉及图像的预处理，以便于后续的数据嵌入操作。
3. 使用 bitset 函数逐位地将整数的二进制位嵌入到载体图像的像素点中的最低有效位（Least Significant Bit, LSB）。具体来说，这一过程涉及修改载体像素的 LSB，使其与整数的相应二进制位相匹配。
4. 在完成数据的嵌入后，将修改后的像素点重新组合，构建出新的图像。这一步骤是恢复图像显示功能的关键，确保图像在视觉上的连贯性和质量。
5. 将包含隐藏数据的新图像保存到存储设备上。这一步骤确保了数据的持久存储和传输的可行性。
6. 接收方可以通过分析图像的最低有效位来提取隐藏的整数数据。将这些位重新组合，即可恢复出原始的整数。

1. 初始化和图像加载

读取一个名为”Lena.bmp” 的 BMP 图像文件。此图像将作为水印嵌入的载体，意味着后续的操作都将在这个图像的基础上进行。接下来，使用 imshow 函数显示这个载体图像。imshow 是 MATLAB 中一个常用的图像显示函数，能够直接在 MATLAB 的图形窗口中展示图像。在这里，imshow(imgOriginal, []) 的调用自动调整图像的显示对比度，使其包含所有原始像素值。此外，定义了一个整数 intMessage，该整数值为 2003120。这个整数包含了将要被嵌入到图像中的信息。

```
function IntHiding()
    % 载入图像
    imgOriginal = imread("Lena.bmp"); % 载体图像
    imshow(imgOriginal, []) % 显示原始图像
```

```
    % 初始化要嵌入的信息
    intMessage = 2003120; % 要嵌入的信息
    % 进行水印嵌入和提取
    imgWatermarked = Hide(imgOriginal, intMessage);
    extractedWatermark = Extract(imgWatermarked);
    disp('Extracted watermark:')
    disp(extractedWatermark)
end
```

2. 水印嵌入过程

Hide 函数实现了水印嵌入过程。首先，获取原始图像的尺寸（行数和列数），这一步对于后续遍历图像的每一个像素至关重要。然后，初始化一个和原始图像同样大小的 imgWatermarked 矩阵，用于存储处理后的图像数据。接下来，通过双层循环（遍历行和列）对图像的每个像素进行操作。循环中的条件判断 if $i = 1$ && $j \leq 21$ 是核心，它确定了水印信息将被嵌入到图像的第一行的前 21 个像素中。这是因为整数 intMessage 被转换为二进制，最多可包含 21 位信息。在满足条件的像素上，执行位操作：使用 bitget 函数从 intMessage 中提取特定位置的位值（第 j 位）；使用 bitset 函数将此位值设置到当前像素的最低位上；对于不需要嵌入信息的像素，直接将原始图像的像素值赋给水印图像。循环结束后，使用 imwrite 函数将 imgWatermarked 矩阵保存为一个新的 BMP 文件”lsb_int_watermarked.bmp”。这个文件包含了嵌入了水印的图像。接着，使用 imshow 函数显示这个水印图像。

```
function imgWatermarked = Hide(origin, watermark)
    % 获取原始图像尺寸
    [rows, cols] = size(origin);
    imgWatermarked = uint8(zeros(size(origin)));

    for i = 1:rows
        for j = 1:cols
            if i == 1 && j <= 21
                bitValue = bitget(watermark, j);
                imgWatermarked(i, j) = bitset(origin(i, j), 1, bitValue);
            else
                imgWatermarked(i, j) = origin(i, j);
            end
        end
    end

    % 保存水印图像
    imwrite(imgWatermarked, 'lsb_int_watermarked.bmp', 'bmp');
    figure;
    imshow(imgWatermarked, []);
    title("水印图像");
end
```

3. 水印提取过程

水印提取函数 Extract 专门用于从修改后的图像中提取出原始嵌入的整数 intMessage。

函数开始时，首先初始化一个整数 extractedWatermark 来存放从图像中提取的水印数据。此变量最初被设置为 0，准备通过位操作逐步构建其值。接着，函数只遍历水印图像 imgWatermarked 的第一行中的前 21 个像素，这与嵌入过程中定义的嵌入区域相匹配。这是因为水印信息仅存储在这些特定的像素中。在遍历的过程中，每个像素的最低位（最不显著位，LSB）被检查：使用 bitget 函数，提取每个像素的最低位。此函数调用格式为 bitget(imgWatermarked(1, j), 1)，表示从第一个像素行的第 j 个像素的最低位中获取位值。将获取的位值通过 bitset 函数设置到 extractedWatermark 变量的对应位上。这个操作重建了原始整数 intMessage，确保了信息从图像中的恢复。这一过程持续进行，直到第一行的前 21 个像素被处理完毕。此时，extractedWatermark 中存储的值应与原始嵌入的整数 intMessage 完全一致，表示提取过程成功。

在此水印提取过程中，通过简单而有效的位操作技术，确保了能从图像中准确地恢复出原始的水印信息。这种提取技术的实现依赖于水印嵌入过程中选择的方法和参数，特别是关于嵌入位置和方法的选择。正确的对称操作（嵌入和提取）是保证信息完整性和可靠性的关键。

```
function extractedWatermark = Extract(imgWatermarked)
    extractedWatermark = 0;
    for j = 1:21
        bitValue = bitget(imgWatermarked(1, j), 1);
        extractedWatermark = bitset(extractedWatermark, j, bitValue);
    end
end
```

4. 实验结果

图 5: 原始图像

图 6: 水印图像

```
>> IntHiding
提取的水印：
     2003120
```

图 7: 提取水印（整数）结果

四、 扩展实验——彩色水印的嵌入和提取

有效的数字水印技术需要满足三个核心特性：不可觉察性、安全性和鲁棒性。首先，不可觉察性要求水印的嵌入不应明显地干扰原有数据的质量或内容。其次，安全性确保水印信息的安全和可靠性，意味着水印能够唯一地标识原始内容，且不可被第三方伪造。最后，鲁棒性指的是即使在非授权的干预下，水印也应当能够保持不被消除，从而确保文档的使用不受影响。

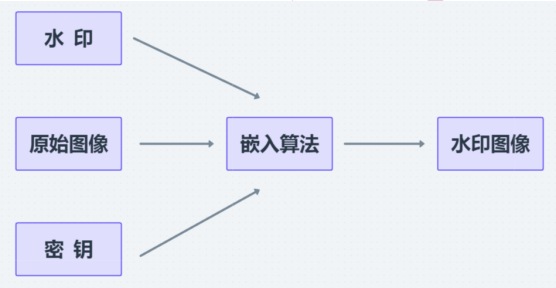
在实现数字水印的过程中，涉及到水印的嵌入和检测两个基本步骤。水印本身可以采用多种形式，例如随机数字序列、数字标识符、文本或图像等。考虑到鲁棒性和安全性，通常对水印进行随机化处理和加密，以增强其保护效能。然而，鲁棒性与不可觉察性之间存在潜在的矛盾，因此在设计水印算法时必须对这两者进行恰当的权衡。

(一) 原理简介

水印的嵌入过程通常涉及算法 E，原始图像 I，以及水印 W，那么水印图像可表示如下：

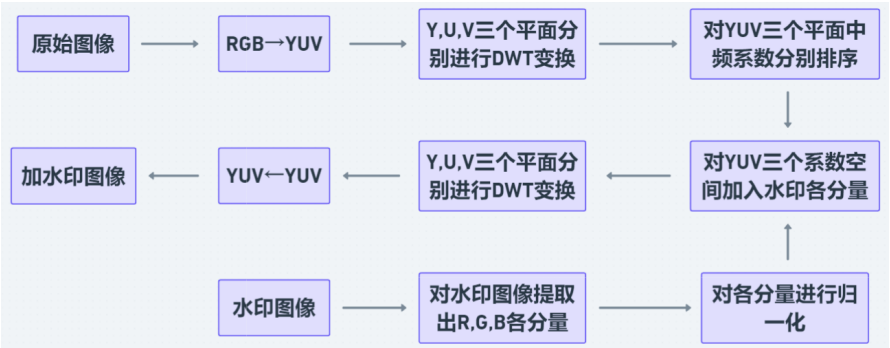
$$I_W = E(I, W)$$

水印嵌入过程如下图所示：



在彩色水印的应用方面，其嵌入与提取过程与一般的数字水印类似，但由于彩色图像包含更丰富的信息，彩色水印也呈现出其特有的复杂性。一些研究文献展示了使用基于离散余弦变换（DCT）的压缩编码技术对彩色水印进行处理。在这些方法中，彩色水印被编码为一系列二值 ID 数字序列，并直接作为掩码信息嵌入到基于 DCT 系数的原始图像中。这种方法的优点在于提取水印不需要原始图像，能够有效抵抗 JPEG 格式的有损压缩。然而，这种方法对其他类型的图像处理操作，如剪切、缩放或平滑等，的抵抗性较弱。

针对上述弱点，研究者们提出了一种基于离散小波变换（DWT）的双彩色图印方案。通过实验验证，该方案不仅能够抵抗 JPEG 的有损压缩，也能有效地对抗剪切、放缩和平滑等多种图像处理操作，显示出较高的鲁棒性和安全性。这种基于小波变换的方法因其在处理高频和低频信息方面的优势，能够更灵活地控制水印的不可觉察性与鲁棒性之间的权衡。

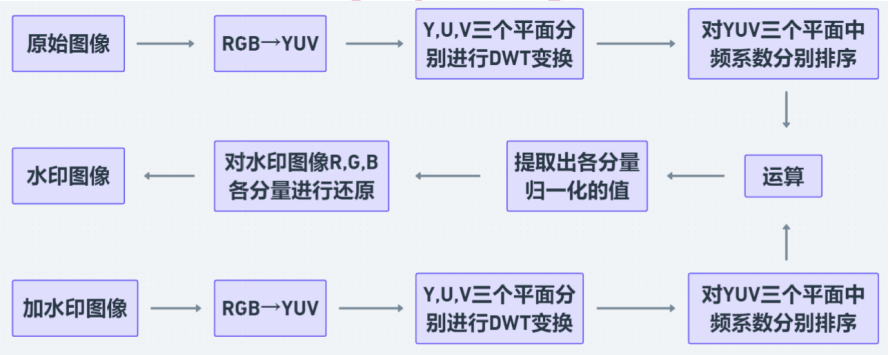


彩色水印的嵌入过程详细展示如上图所示，基本步骤包括选择合适的彩色通道，应用小波变换分解图像，将水印信息编码后嵌入到特定的小波系数中，最后通过逆小波变换重构图像以完成水印的嵌入。这一系列步骤确保了水印的有效隐藏同时维持了图像的视觉质量。

在具体实践中，一些细节对于确保水印的不可觉察性和鲁棒性至关重要：

- **归一化过程**：将水印图像从原始的 [0, 255] 范围归一化到 [-1, 1] 是常见的处理方法。这种归一化有助于数据处理时达到零均值，使得算法的处理更加稳定。在实际叠加水印时，通过乘以一个系数 来调整水印的强度。选择 =4 被视为是一个平衡不可觉察性和鲁棒性的好方法。这种方法允许水印的强度足以抵抗某些干扰，同时又不至于对图像质量造成显著影响。
- **颜色空间选择**：选择在 YUV 色彩空间进行小波变换，因为 YUV 空间更符合人类视觉系统的感知特性。在 YUV 空间中，Y 通道负责亮度信息，而 U 和 V 通道则包含色彩信息。相较于 RGB 或其他色彩空间，YUV 空间在进行水印嵌入时能更好地隐藏水印信息，因为人眼对亮度的敏感度高于色彩。实验表明，在 YUV 空间中进行水印处理的效果确实优于其他色彩空间。
- **水印的嵌入位置**：在进行小波变换后，选择将水印添加至较大的中频系数上。这是因为较大的系数通常对应着图像中较强的信号部分，因此，在这些系数上加入较小的扰动（水印信息）不会导致明显的视觉变化。这种策略既保持了水印的隐蔽性，同时也利用了图像本身的特性来增强水印的鲁棒性。

提取水印图像则是加水印的逆过程，基本步骤如下图所示：



在 MATLAB 环境中对水印技术的实现进行了一些调整，以适应具体的应用需求和优化性能。具体变化如下：

- **小波变换的级别**：实际应用中使用了二级小波变换，而非原文中提到的四级。虽然这种变化可能导致鲁棒性有所降低（因为较高级别的小波变换能更细致地分离图像的细节和纹理，提高水印的隐藏性和抵抗剪切等攻击的能力），但它允许增大水印的尺寸，因此可以嵌入更多的信息。这种权衡主要是基于对水印可见性和容量需求的不同重视程度。
- **水印的添加位置**：虽然原始文章建议将水印添加到中频系数中，实践中则优先向低频分量添加水印。低频分量包含了图像的主要能量和结构信息，因此其能量较高且更能抵抗干扰。如果低频系数不足以容纳所需的水印信息，再考虑将水印添加至中频系数。这种方法提高了水印的抗干扰性，尽管可能略微影响到图像的整体视觉质量。
- **颜色空间的选择**：在这种实现中，采用了 YCbCr 颜色空间，而不是 YUV。虽然两者在概念上类似，都分离了亮度信息与色彩信息，使得水印可以更有效地嵌入而不影响图像的视

觉感知，但 YCbCr 是一种在数字视频和照片压缩中更为常用的标准，它适用于数字处理和传输。使用 YCbCr 空间有助于更好地适配现代数字图像处理和存储系统。

(二) 代码实现-隐藏处理后的图像

1. 输入参数与变量初始化

```
% 将OriImg从RGB色彩空间转换到YCbCr色彩空间
OriImg = rgb2ycbcr(OriImg);
% 初始化MarkedImg为零矩阵，大小与OriImg相同
MarkedImg = zeros(size(OriImg));
```

- **输入参数**:OriImg-一个大小为 512x512x3 的 RGB 彩色图像,用作隐藏信息的载体;Print-一个大小为 140x200x3 的 RGB 彩色图像，包含需要隐藏的信息。
- **输出参数**：Imgout-经过信息隐藏处理后的图像，格式和大小与 OriImg 相同。

代码开始时，首先将 OriImg 从 RGB 色彩空间转换到 YCbCr 色彩空间，这种转换常用于图像处理中，以便于处理色彩和亮度信息。

2. 循环处理每个颜色通道

MarkedImg 初始化为一个与 OriImg 同样大小的零矩阵。这个矩阵后面用来存放处理后的图像数据。对于图像的每个颜色通道（Y, Cb, Cr），执行以下步骤:

1. **提取颜色通道**：每个通道的数据被单独提取出来，以便进行处理。
2. **打印层归一化**：Print 图像的每个颜色通道都被归一化到-1 到 1 的范围内，以方便后续的计算。
3. **小波变换**：使用 wavedec2 函数对 Orilayer（载体图像的当前颜色通道）进行二维小波变换,得到小波系数 C 和相应的尺寸矩阵 S。
4. **隐写处理**：计算小波域中的系数位置：使用小波系数和尺寸矩阵来定位不同的小波子带（例如，近似子带 cA2 和水平细节子带 cH2)。
5. **信息嵌入**：通过修改小波系数来嵌入 Print 图像的信息。系数的修改基于一个称为 alpha 的缩放因子。
6. **逆小波变换**：使用修改后的小波系数和尺寸矩阵通过 waverec2 进行逆小波变换，得到处理后的颜色通道。

```
alpha = 4;
for i=1:3
    Orilayer = OriImg(:,:,i);
    Printlayer = Print(:,:,i);
    Printlayer = (double(Printlayer)-128)/128;  % 归一到-1~1
    ...
end
    [C,S] = wavedec2(Orilayer,2,'db2');
```

```
    [rows, cols] = size(Printlayer);
    totallength = rows*cols;
    cA2 = appcoef2(C,S,'db2',2);
    cH2 = detcoef2('h',C,S,2);
    [cA2_r, cA2_c] = size(cA2);
    [cH2_r, cH2_c] = size(cH2);
    part1length = cA2_r*cA2_c;
    part2length = cH2_r*cH2_c;
    flattenpri = reshape(Printlayer, 1, []);
    C(1:part1length) = C(1:part1length)+alpha*flattenpri(1:part1length);
    leftlength = totallength-part1length;

    cH2 = C(part1length+1:part1length+part2length);
    [~, IH] = sort(abs(cH2),'descend');
    IH = IH(1:leftlength);
    cH2(IH) = cH2(IH)+alpha*flattenpri(part1length+1:totallength);
    C(part1length+1:part1length+part2length) = cH2;
```

3. 输出图像的重组

```
Outlayer = waverec2(C,S,'db2');
MarkedImg(:,:,i) = uint8(Outlayer);
Imgout = ycbcr2rgb(uint8(MarkedImg));
```

所有处理后的颜色通道被重新组合并转换回 RGB 色彩空间，得到最终的输出图像 Imgout。整个过程主要涉及颜色空间转换、小波变换、信息嵌入和逆变换等图像处理技术，以实现在保持视觉质量的同时隐藏信息的目的。

(三) 代码实现-分离隐藏的图像

1. 输入参数与变量初始化

```
function Print = Recover_3(MarkedImg, OriImg)
rec_row = 140; % 此处为水印图片的高度和宽度
rec_col = 200;

% 将OriImg和MarkedImg从RGB色彩空间转换到YCbCr色彩空间
OriImg = rgb2ycbcr(OriImg);
MarkedImg = rgb2ycbcr(MarkedImg);

outImg = zeros(rec_row, rec_col, 3); % 初始化输出图像矩阵
alpha = 4; % 定义缩放因子，用于后续处理
```

- **输入参数**:MarkedImg：一个可能含有隐藏信息的图像，大小为 512x512x3，RGB 彩色图像，uint8 格式;OriImg：原始载体图像，同样是一个 512x512x3 大小的 RGB 彩色图像，uint8 格式，不含隐藏信息。
- 输出参数:

- **Print**: 输出图像, 大小为 140x200x3, RGB 彩色图像, uint8 格式, 其中包含从 MarkedImg 中提取的信息。

2. 提取隐藏信息的图像处理步骤

首先进行初始化和色彩空间转换。具体地，输入的图像 MarkedImg（可能含有隐藏信息的图像）和 OriImg（原始载体图像）都从 RGB 色彩空间转换到 YCbCr 色彩空间，以便分离亮度和色度信息。同时，定义输出图像 Print 的大小为 140x200x3，并初始化为零矩阵。使用的缩放因子 alpha 设为 4，用于后续计算。

接下来，代码通过循环处理 YCbCr 色彩空间中的每个通道（Y, Cb, Cr)。对于每个通道，执行以下步骤:

1. 使用 wavedec2 函数对 OriImg 和 MarkedImg 中的每个通道进行二维小波变换，提取相应的小波系数和尺寸矩阵。
2. 通过计算原始图像和标记图像小波系数的差异，提取出隐藏的信息。这一步计算两图像对应小波系数的差值，并根据隐藏时使用的缩放因子 alpha 进行调整，以恢复出原始的隐藏信息。
3. 重构隐藏信息，通过调整归一化和缩放处理后的数据范围，使其适合 uint8 格式，然后将其赋值给相应的通道。

最后，将处理后的三个通道重新组合，转换成 uint8 格式的 RGB 图像，得到 Print，这是从 MarkedImg 中提取的隐藏信息所在的输出图像。这个过程展示了如何使用小波变换和逆变换技术在图像中隐藏和提取信息的高级应用。

```
% 转换色彩空间为YCbCr
OriImg = rgb2ycbcr(OriImg);
MarkedImg = rgb2ycbcr(MarkedImg);

% 初始化输出图像和设置缩放因子
rec_row = 140; %水印图片的高度
rec_col = 200; %水印图片的宽度
outImg = zeros(rec_row, rec_col, 3);
alpha = 4;

% 处理每个颜色通道
for i=1:3
    % 提取原始和标记图像的单一通道
    Orilayer = OriImg(:,:,i);
    Marklayer = MarkedImg(:,:,i);

    % 对原始图像进行小波分解
    [Co,So] = wavedec2(Orilayer,2,'db2');
    coA2 = appcoef2(Co,So,'db2',2);
    [cA2_r, cA2_c] = size(coA2);
    part1length = cA2_r*cA2_c;
    coH2 = detcoef2('h',Co,So,2);
    [cH2_r, cH2_c] = size(coH2);
    part2length = cH2_r*cH2_c;
```

```
    coH2 = Co(part1length+1:part1length+part2length);
    [~, IHo] = sort(abs(coH2),'descend');
    leftlength = rec_row*rec_col - part1length;
    IHo = IHo(1:leftlength);

    % 对标记图像进行小波分解
    [Cm,Sm] = wavedec2(Marklayer,2,'db2');
    cmA2 = Cm(1:part1length);
    cmH2 = Cm(part1length+1:part1length+part2length);

    % 计算隐藏信息
    flat_print = zeros(1,rec_row*rec_col);
    flat_print(1:part1length) = cmA2 - coA2;
    flat_print(part1length+1:rec_row*rec_col) = cmH2(IHo) - coH2(IHo);

    % 重构隐藏的图像信息
    Print = reshape(flat_print, rec_row, rec_col) / alpha;
    Print = (Print * 128) + 128;
    outImg(:,:,i) = Print;
end

% 输出最终的提取图像
Print = uint8(outImg);
```

(四) 实验结果

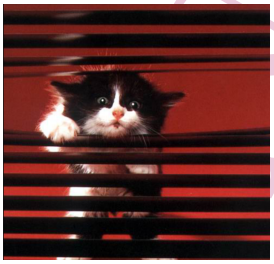


图 8: 原始彩色图像

图 9: 水印图像

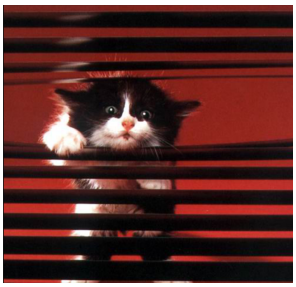


图 10: 加水印后图像

图 11: 恢复后图像

五、 实验心得体会

在本次实验中，我首先通过实施 LSB（最低有效位）隐写技术探索了基础的数字隐写方法，包括将二进制图像嵌入灰度图像和将整数信息隐藏于图像中。这不仅加深了我对数字隐写理论的理解，也让我亲手操作并观察了这些概念的实际应用效果。我意识到虽然 LSB 隐写技术简单且容量大，但它在抗干扰和安全性方面表现较弱。此外，扩展实验让我尝试了将彩色水印嵌入彩色图像，并采用基于离散小波变换（DWT）的方法，有效提升了水印的抗压缩和抗图像处理攻击的能力。通过在 YCbCr 颜色空间处理图像，我进一步了解了色彩空间选择对隐写效果的影响。这次实验不仅提升了我的编程能力和对相关算法的理解，也加深了我对多媒体安全技术应用的认识，为我的学术研究和未来职业生涯奠定了坚实基础。