# 基于 LSB 算法的数字水印图片泄露检测系统实现

### 1.1 项目目标

1. 实现基于 LSB 算法的数字水印嵌入功能，确保水印嵌入后不影响图片的视觉效果。
2. 实现水印提取功能，能从经过处理的图片中准确提取出水印信息。
3. 对嵌入水印的图片进行多种常见处理（如翻转、对比度调整等），测试水印的鲁棒性。
4. 通过计算提取水印与原始水印的相似度，量化评估水印算法的性能。

## 二、相关算法原理

### 2.1 LSB 算法基本原理

LSB（Least Significant Bit，最低有效位）算法是一种典型的空间域水印算法，其核心思想是利用人类视觉系统（HVS）对图像像素最低位变化不敏感的特性，将水印信息嵌入到图像像素的最低有效位中。

对于数字图像，每个像素的颜色通常由红（R）、绿（G）、蓝（B）三个通道组成，每个通道的取值范围为 0-255，用 8 位二进制数表示。例如，一个蓝色通道值为 135，其 8 位二进制表示为 10000111。其中，最左边的位为最高有效位（MSB），对像素值的影响最大；最右边的位为最低有效位（LSB），对像素值的影响最小，仅为 1。

LSB 算法正是利用这一特性，将水印信息嵌入到像素的 LSB 中。由于单个像素的 LSB 变化对像素值的影响极小（最大变化为 1），人眼无法察觉这种细微变化，从而实现水印的隐蔽性。

### 2.2 水印嵌入的数学表示

设原始图像的某个像素的蓝通道值为，其 8 位二进制表示为 ，其中 为最高有效位，为最低有效位。

水印信息为二进制序列，其中

，表示水印在位置 处的比特值。

嵌入水印时，首先清除原始像素值的最低有效位，得到：

其中， 0xFE 是十六进制数，对应的二进制为 11111110，与原始像素值进行按位与运算后，可将最低有效位清零。

然后，将水印比特值嵌入到最低有效位：

其中， | 为按位或运算，通过该操作将水印比特值写入像素的最低有效位，得到嵌入水印后的像素值 P\_w 。

### 2.3 水印提取的数学表示

水印提取是嵌入过程的逆操作。对于嵌入水印后的像素值 P\_w ，其 8 位二进制表示为

提取水印时，通过与 1 进行按位与运算，获取最低有效位的值，即：

其中， 为提取到的水印比特值。

### 2.4 相似度计算原理

为评估水印提取的准确性，采用归一化相似度指标，其计算公式为：

其中， 表示提取水印与原始水印中相同比特的数量，表示水印的总比特数。相似度 S 的取值范围为 0-100%，值越高表明提取水印与原始水印的一致性越好，水印算法的鲁棒性越强。

## 三、系统设计与实现

### 3.1 系统总体结构

本系统主要由五个模块组成：水印生成模块、水印嵌入模块、水印提取模块、图片处理模块和相似度计算模块。系统总体结构如图 1 所示：

图 1 系统总体结构

* 水印生成模块：生成用于嵌入的二进制水印图案。
* 水印嵌入模块：将水印信息嵌入到原始图片中，生成带水印的图片。
* 图片处理模块：对带水印的图片进行各种常见处理，模拟图片泄露过程中可能受到的操作。
* 水印提取模块：从处理后的图片中提取水印信息。
* 相似度计算模块：计算提取水印与原始水印的相似度，评估水印算法的鲁棒性。

### 3.2 模块详细设计与实现

#### 3.2.1 水印生成模块

本模块生成 32×32 的二进制棋盘格水印图案，实现代码如下：

|  |
| --- |
| void generate\_watermark(unsigned char watermark[WATERMARK\_HEIGHT][WATERMARK\_WIDTH]) {  // 生成棋盘格图案作为示例水印  for (int i = 0; i < WATERMARK\_HEIGHT; i++) {  for (int j = 0; j < WATERMARK\_WIDTH; j++) {  watermark[i][j] = ((i/4 + j/4) % 2) ? 255 : 0;  }  }  } |

生成原理：通过双层循环遍历 32×32 的水印矩阵，对于每个位置 (i,j) ，根据 (i/4 + j/4) \% 2 的结果确定水印值。当结果为 1 时，水印值为 255（表示二进制 1）；当结果为 0 时，水印值为 0（表示二进制 0），从而形成棋盘格图案。

#### 3.2.2 水印嵌入模块

该模块将生成的水印嵌入到原始图片的蓝通道最低有效位，实现步骤如下：

1. 加载原始图片，获取图片的宽度、高度和通道数。
2. 检查图片尺寸是否不小于水印尺寸，确保水印能够完整嵌入。
3. 遍历水印的每个像素，计算其在图片中的对应位置。
4. 对图片对应位置的蓝通道值进行处理，嵌入水印比特。
5. 保存嵌入水印后的图片。

实现代码如下：

|  |
| --- |
| void embed\_watermark(const char \*input\_path, const char \*output\_path,  unsigned char watermark[WATERMARK\_HEIGHT][WATERMARK\_WIDTH]) {  int width, height, channels;  unsigned char \*image = stbi\_load(input\_path, &width, &height, &channels, 0);  if (!image) {  printf("无法加载图片: %s\n", input\_path);  return;  }  // 确保图片尺寸不小于水印  if (width < WATERMARK\_WIDTH || height < WATERMARK\_HEIGHT) {  printf("图片尺寸小于水印尺寸\n");  stbi\_image\_free(image);  return;  }  // 嵌入水印到蓝通道的最低位  for (int i = 0; i < WATERMARK\_HEIGHT; i++) {  for (int j = 0; j < WATERMARK\_WIDTH; j++) {  int pos = (i \* width + j) \* channels + 2; // 蓝通道位置  // 清除最低位，嵌入水印位  image[pos] = (image[pos] & 0xFE) | (watermark[i][j] ? 1 : 0);  }  }  // 保存带水印的图片  stbi\_write\_png(output\_path, width, height, channels, image, width \* channels);  stbi\_image\_free(image);  printf("水印嵌入完成: %s\n", output\_path);  } |

选择蓝通道进行水印嵌入的原因是，人眼对蓝色的敏感度相对较低，在蓝通道嵌入水印对图片视觉效果的影响更小。

#### 3.2.3 水印提取模块

该模块从处理后的图片中提取水印信息，实现步骤如下：

1. 加载待提取水印的图片，获取图片的宽度、高度和通道数。
2. 遍历水印矩阵的每个位置，计算其在图片中的对应位置。
3. 提取图片对应位置蓝通道值的最低有效位，作为水印比特。
4. 生成提取到的水印图案。

实现代码如下：

|  |
| --- |
| void extract\_watermark(const char \*input\_path, unsigned char watermark[WATERMARK\_HEIGHT][WATERMARK\_WIDTH]) {  int width, height, channels;  unsigned char \*image = stbi\_load(input\_path, &width, &height, &channels, 0);  if (!image) {  printf("无法加载图片: %s\n", input\_path);  return;  }  // 提取水印  for (int i = 0; i < WATERMARK\_HEIGHT; i++) {  for (int j = 0; j < WATERMARK\_WIDTH; j++) {  if (i >= height || j >= width) {  watermark[i][j] = 0;  continue;  }  int pos = (i \* width + j) \* channels + 2; // 蓝通道位置  watermark[i][j] = (image[pos] & 1) ? 255 : 0; // 提取最低位  }  }  stbi\_image\_free(image);  printf("水印提取完成\n");  } |

#### 3.2.4 图片处理模块

该模块模拟图片在传播过程中可能受到的处理操作，本系统实现了水平翻转和对比度调整两种处理，具体如下：

1. 水平翻转：将图片沿垂直中轴线进行翻转，实现代码如下：

|  |
| --- |
| void flip\_image(const char \*input\_path, const char \*output\_path, int horizontal) {  int width, height, channels;  unsigned char \*image = stbi\_load(input\_path, &width, &height, &channels, 0);  if (!image) return;  unsigned char \*flipped = malloc(width \* height \* channels);  for (int i = 0; i < height; i++) {  for (int j = 0; j < width; j++) {  int src\_x = horizontal ? (width - 1 - j) : j;  int src\_y = horizontal ? i : (height - 1 - i);  int src\_pos = (src\_y \* width + src\_x) \* channels;  int dst\_pos = (i \* width + j) \* channels;  memcpy(&flipped[dst\_pos], &image[src\_pos], channels);  }  }  stbi\_write\_png(output\_path, width, height, channels, flipped, width \* channels);  free(flipped);  stbi\_image\_free(image);  } |

水平翻转的原理是将图片中每个像素的水平坐标进行反转，即对于位置 (i,j) 的像素，翻转后其位置变为 (i, width-1-j) 。

1. 对比度调整：通过调整像素值的动态范围改变图片的对比度，实现代码如下：

|  |
| --- |
| void adjust\_contrast(const char \*input\_path, const char \*output\_path, float factor) {  int width, height, channels;  unsigned char \*image = stbi\_load(input\_path, &width, &height, &channels, 0);  if (!image) return;  for (int i = 0; i < width \* height \* channels; i++) {  float val = image[i] / 255.0f;  val = (val - 0.5f) \* factor + 0.5f; // 对比度调整公式  val = val < 0 ? 0 : (val > 1 ? 1 : val);  image[i] = (unsigned char)(val \* 255);  }  stbi\_write\_png(output\_path, width, height, channels, image, width \* channels);  stbi\_image\_free(image);  } |

对比度调整的数学公式为：

val' = (val - 0.5) \times factor + 0.5

其中， val 为归一化后的原始像素值（0-1）， factor 为对比度调整因子， val' 为调整后的像素值。当 factor > 1 时，对比度增强；当 0 < factor < 1 时，对比度减弱。

#### 3.2.5 相似度计算模块

该模块计算提取水印与原始水印的相似度，实现代码如下：

|  |
| --- |
| float calculate\_similarity(unsigned char orig[WATERMARK\_HEIGHT][WATERMARK\_WIDTH],  unsigned char extracted[WATERMARK\_HEIGHT][WATERMARK\_WIDTH]) {  int same = 0, total = 0;  for (int i = 0; i < WATERMARK\_HEIGHT; i++) {  for (int j = 0; j < WATERMARK\_WIDTH; j++) {  if ((orig[i][j] && extracted[i][j]) || (!orig[i][j] && !extracted[i][j])) {  same++;  }  total++;  }  }  return (float)same / total;  } |

计算原理：遍历原始水印和提取水印的每个比特，统计相同比特的数量，然后除以总比特数，得到相似度。