**重庆大学课程报告**



|  |  |
| --- | --- |
| **题目** |  |
| **学院** |  |
| **专业班级** |  |
| **年级** |  |
| **姓名** |  |
| **学号** |  |
| **完成时间** |  |
| **成绩** |  |
| **指导教师** |  |

重庆大学教务处制

**隐藏式数字水印实现**

摘要：数字水印是指将特定的信息嵌入数字信号中，数字信号可能是音频、图片或是视频等，在著作权保护、信息影藏等方面有着重要作用。现有的数字水印技术可以分为浮现式和隐藏式两大类，本项目主要针对后者。在本项目中作者实现了LSB、DCT、DWT三种隐藏式图片水印算法，并在传统LSB算法的基础上使用Arnold图片置乱技术进行改进以弥补LSB不能抵御裁剪和遮挡攻击的不足。最终作者将这些算法编写为REST API，在此基础上实现了一个嵌入和提取数字水印的web应用程序。

关键字：数字水印；LSB；DCT；DWT

目录

[1 数字水印简介 5](#_Toc68780660)

[2 数字水印算法 5](#_Toc68780661)

[2.1 LSB 5](#_Toc68780662)

[2.1.1文本描述 5](#_Toc68780663)

[2.1.2 示例展示 6](#_Toc68780664)

[2.1.3 核心代码 6](#_Toc68780665)

[2.2 DCT 6](#_Toc68780666)

[2.2.1文本描述 6](#_Toc68780667)

[2.2.2 示例展示 7](#_Toc68780668)

[2.2.3核心代码 7](#_Toc68780669)

[2.3 DWT 8](#_Toc68780670)

[2.3.1文本描述 8](#_Toc68780671)

[2.3.2 示例展示 9](#_Toc68780672)

[2.3.3 核心代码 9](#_Toc68780673)

[2.4 LSB+Arnold 10](#_Toc68780674)

[2.4.1 文本描述 10](#_Toc68780675)

[2.4.2 示例展示 10](#_Toc68780676)

[2.4.3 核心代码 11](#_Toc68780677)

[3 API实现 12](#_Toc68780678)

[4 Web应用实现 12](#_Toc68780679)

[5 总结 12](#_Toc68780680)

# 数字水印简介

数字水印（Digital Watermarking）技术是将一些标识信息(即数字水印)直接嵌入数字载体当中(包括图片、文档等)或是间接表示(修改特定区域的结构)，且不影响原载体的使用价值，也不容易被探知和再次修改。但可以被生产方识别和辨认。通过这些隐藏在载体中的信息，可以达到确认内容创建者、购买者、传送隐秘信息或者判断载体是否被篡改等目的。数字水印是信息隐藏技术的一个重要研究方向。 数字水印是实现版权保护的有效办法，是信息隐藏技术研究领域的重要分支。

# 数字水印算法

## LSB

### 2.1.1文本描述

一种空域上的数字水印算法。具体的做法为将黑白水印信息存储在图片RGB通道的B通道的最后一位。用0和1分别表示黑白存放，因改变的位数处于最后一位，隐藏人眼并不能观测到水印。

### 2.1.2 示例展示

|  |  |  |
| --- | --- | --- |
| 原图 | 嵌入水印后图 | 提取的水印 |
|  |  |  |

### 2.1.3 核心代码

|  |
| --- |
| **嵌入代码：**  for i in range(mark.shape[0]):  for j in range(mark.shape[1]):  blue = pic[i, j, 2]  pic[i, j, 2] = blue - blue % 2 + mark[i][j]//将二值水印信息嵌入  **提取代码：**  for i in range(pic.shape[0]):          for j in range(pic.shape[1]):              mark[i, j] = pic[i, j, 2] % 2//提取出B通道的最后一位返回给水印 |

## DCT

### 2.2.1文本描述

离散余弦变换（DCT for Discrete Cosine Transform）是与傅里叶变换相关的一种变换，它与离散傅里叶变换类似，但是只使用实数。本项目采用DCT变换对图片进行处理，在这个过程中，每次变换图片里不重叠的8\*8矩阵块，这个矩阵块包含的信息为水印的一个像素，因此根据这个像素的黑白将对这个矩阵块所有的元素进行一个微小的变化，在提取水印的时候就是根据这个细微的变化进行提取。

### 2.2.2 示例展示

|  |  |  |
| --- | --- | --- |
| 原图 | 嵌入水印后图 | 提取的水印 |
|  |  |  |

### 2.2.3核心代码

|  |
| --- |
| **嵌入代码：**  for i in range(row):          for j in range(col):              BLOCK=np.float32(im\_array[i\*block\_width:i\*block\_width+block\_width,j\*block\_width:j\*block\_width+block\_width])              BLOCK=scipy.fft.dct(BLOCK)//矩阵DCT              if mark\_array[i][j]==False://根据二值矩阵赋值给a                  a=-1              else:                  a=1              BLOCK[:,:,2]=BLOCK[:,:,2]\*(1+a\*0.03)//矩阵块的微小变化              BLOCK=scipy.fft.idct(BLOCK).astype(np.uint8)//将矩阵逆DCT              im\_array[i\*block\_width:i\*block\_width+block\_width,j\*block\_width:j\*block\_width+block\_width]=BLOCK  **提取代码：**  for i in range(row):          for j in range(col):              BLOCK\_ORIGIN=np.float32(im\_array[i\*block\_width:i\*block\_width+block\_width,j\*block\_width:j\*block\_width+block\_width])              BLOCK\_MARKED=np.float32(marked\_array[i\*block\_width:i\*block\_width+block\_width,j\*block\_width:j\*block\_width+block\_width])              BLOCK\_ORIGIN=scipy.fft.idct(BLOCK\_ORIGIN)              BLOCK\_MARKED=scipy.fft.idct(BLOCK\_MARKED)                bm=BLOCK\_MARKED[1,1,2]//获取水印拟DCT后指定位置元素，本质上只要是这个矩阵块的元素都是可以采用的              bo=BLOCK\_ORIGIN[1,1,2]//获取嵌入后水印逆DCT后元素              a=bm/bo-1//进行差距检测              if a<0:                  decode\_pic[i,j]=False              else:                  decode\_pic[i,j]=True |

## DWT

### 2.3.1文本描述

DWT即离散小波变换，是与傅里叶变换一样，小波变换的基本思想是将信号展开成簇基函数之加权和，即用一簇函数来表示信号和函数。本项目采用两次DWT处理图片的RGB通道的B通道，取返回的结果的近似矩阵进行处理。

### 2.3.2 示例展示

|  |  |  |
| --- | --- | --- |
| 原图 | 嵌入水印后图 | 提取的水印 |
|  |  |  |

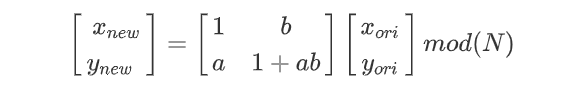
### 2.3.3 核心代码

|  |
| --- |
| **嵌入代码：**  idx = random.sample(range(ca2\_size), mark\_copy\_size)      ca2\_flatten = ca2.flatten()      mark\_copy\_flatten = mark\_copy.flatten()      for i in range(mark\_copy\_size):          c = ca2\_flatten[idx[i]]          z = c % mark\_w          if mark\_copy\_flatten[i]:  # 水印对应二进制为1  //将z替换成c-k\*mark\_w，就能发现水印为1时f模mark\_w为1/4mark\_w              if z < mark\_w/4:                  f = c-mark\_w/4-z              else:                  f = c+mark\_w\*3/4-z          else:  # 水印对应二进制为0  //将z替换成c-k\*mark\_w，就能发现水印为1时f模mark\_w为3/4mark\_w              if z < mark\_w\*3/4:                  f = c+mark\_w/4-z              else:                  f = c+mark\_w\*5/4-z          ca2\_flatten[idx[i]] = f  **提取代码：**  idx = random.sample(range(ca2w\_size), mark\_copy\_size)      res\_mark = mark\_copy.flatten()  # 就算是从新建立一个zeros矩阵初始化效果也一样      ca2w\_flatten = ca2w.flatten()      for i in range(mark\_copy\_size):          c = ca2w\_flatten[idx[i]]          z = c % mark\_w          if z < mark\_w/2:              res\_mark[i] = False          else:              res\_mark[i] = True |

## LSB+Arnold

### 2.4.1 文本描述

LSB已经介绍过了，这里介绍Arnold算法，它就是像素的位置转化，具体公式如下：



这里的N表示图片长或者宽，但是这里对图片统一认为长宽相等，且必须水印与和原图一样大，否则提取出的水印逆向置乱会产生错误。

### 2.4.2 示例展示

|  |  |  |
| --- | --- | --- |
| 原图 | 嵌入水印后进行遮盖的图 | 提取的水印 |
|  |  |  |

### 2.4.3 核心代码

|  |
| --- |
| **置乱代码**：  for \_ in range(shuffle\_times):          for ori\_x in range(h):              for ori\_y in range(w):                  new\_x = (1\*ori\_x + b\*ori\_y)% N                  new\_y = (a\*ori\_x + (a\*b+1)\*ori\_y) % N                  if mode == '1':                      arnold\_image[new\_x, new\_y] = image[ori\_x, ori\_y]                  else:                      arnold\_image[new\_x, new\_y, :] = image[ori\_x, ori\_y, :]  **恢复代码：**  for \_ in range(shuffle\_times):          for ori\_x in range(h):              for ori\_y in range(w):                  new\_x = ((a\*b+1)\*ori\_x + (-b)\* ori\_y)% N                  new\_y = ((-a)\*ori\_x + ori\_y) % N                  if mode == '1':                      decode\_image[new\_x, new\_y] = image[ori\_x, ori\_y]                  else:                      decode\_image[new\_x, new\_y, :] = image[ori\_x, ori\_y, :] |

# API实现

# Web应用实现

# 总结