Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №8

По дисциплине «Методы защиты информации»

По теме «Стеганографические методы»

Выполнил:

Студент гр. 653501

Никитинская А. С.

Проверил:

Артемьев В. С.

Минск 2019

Содержание

[1. Постановка задачи 3](#_Toc26722193)

[2.Краткие теоретические сведения 4](#_Toc26722194)

[3. Блок-схемы алгоритма 6](#_Toc26722195)

[Вывод 7](#_Toc26722196)

[Приложение 1. Исходный код программы 8](#_Toc26722197)

[Приложение 2. Скриншот работы программы 12](#_Toc26722198)

## 1. Постановка задачи

1) Изучить теоретические сведения.

2) Реализовать программное средство, сокрытия (извлечения) текстового сообщения в (из) JPEG изображение(я) на основе метода сокрытия в частотной области изображения.

## Краткие теоретические сведения

Стеганография — способ передачи или хранения [информации](https://ru.wikipedia.org/wiki/Информация) с учётом сохранения в тайне самого факта такой передачи (хранения).

Все алгоритмы встраивания скрытой информации можно разделить на несколько подгрупп:

* Работающие с самим цифровым сигналом. Например, метод LSB.
* «Впаивание» скрытой информации. В данном случае происходит наложение скрываемого изображения (звука, иногда текста) поверх оригинала. Часто используется для встраивания цифровых водяных знаков (ЦВЗ).
* Использование особенностей форматов [файлов](https://ru.wikipedia.org/wiki/Файл). Сюда можно отнести запись информации в [метаданные](https://ru.wikipedia.org/wiki/Метаданные) или в различные другие не используемые зарезервированные поля файла.

По способу встраивания информации стегоалгоритмы можно разделить на линейные (аддитивные), нелинейные и другие. Алгоритмы аддитивного внедрения информации заключаются в линейной модификации исходного изображения, а её извлечение в декодере производится корреляционными методами. При этом ЦВЗ обычно складывается с изображением-контейнером либо «вплавляется» (fusion) в него. В нелинейных методах встраивания информации используется скалярное либо векторное квантование. Среди других методов определенный интерес представляют методы, использующие идеи фрактального кодирования изображений. К [аддитивным](https://ru.wikipedia.org/wiki/Аддитивность) алгоритмам можно отнести:

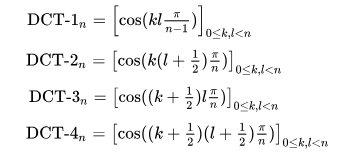
* [А17 (Cox)](https://ru.wikipedia.org/w/index.php?title=А17_(Cox)&action=edit&redlink=1);
* А18 (Barni);
* [L18D (Lange)](https://ru.wikipedia.org/w/index.php?title=L18D_(Lange)&action=edit&redlink=1);
* А21 (J. Kim);
* [А25 (С. Podilchuk)](https://ru.wikipedia.org/w/index.php?title=А25_(С._Podilchuk)&action=edit&redlink=1).

Дискретное косинусное преобразование

Дискретное косинусное преобразование (англ. Discrete Cosine Transform, DCT) — одно из ортогональных преобразований. Вариант косинусного преобразования для вектора действительных чисел. Применяется в алгоритмах сжатия информации с потерями, например, MPEG и JPEG. Это преобразование тесно связано с дискретным преобразованием Фурье и является гомоморфизмом его векторного пространства.

Математически преобразование можно осуществить умножением вектора на матрицу преобразования. При этом матрица обратного преобразования с точностью до множителя равна транспонированной матрице. В математике матрицы выбирают так, чтобы преобразование было ортонормированным, а постоянный множитель равен единице. В компьютерных приложениях это не всегда так.

Различные периодические продолжения сигнала ведут к различным типам дискретного косинусного преобразования. Ниже приводятся матрицы для первых четырёх типов дискретного косинусного преобразования:



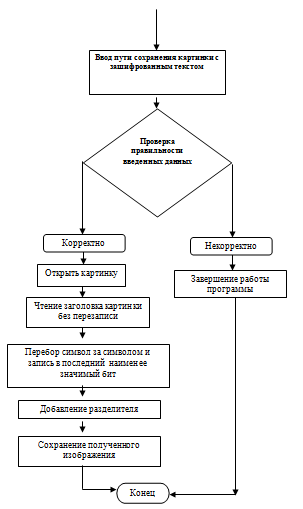
Именно DCT2 чаще всего встречается в практических приложениях благодаря свойству «уплотнения энергии».

DCT для вектора из 8 чисел часто называют DCT2\_{8}. Наиболее распространён двумерный вариант преобразования для матриц 8x8, состоящий из последовательности DCT2\_8 сначала для каждой строки, а затем для каждого столбца матрицы.

Существуют алгоритмы быстрого DCT-преобразования, похожие на алгоритм быстрого преобразования Фурье. Для DCT2\_8 и других вариантов дискретного косинусного преобразования с фиксированной размерностью вектора существуют также алгоритмы, позволяющие свести количество операций умножения к минимуму.

Существуют аналоги дискретного косинусного преобразования, приближающие косинус числами, легко получающимися путём небольшого количества операций сдвига и сложения, что позволяет избежать операций умножения и тем самым повысить скорость вычислений.

## 3. Блок-схемы алгоритма



## Вывод

В ходе написания лабораторной работы были изучены алгоритмы сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения, а также написаны их программные реализации. Были получены навыки усложнения и увеличения криптостойкости алгоритма сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения, а также изучены модификации и режимы работы алгоритма сокрытия и извлечения текстового сообщения из JPEG изображений на основе метода в частотной области изображения.

## Приложение 1. Исходный код программы

|  |
| --- |
| import sys |
|  | import numpy as np |
|  | # from PIL import Image, ImageDraw |
|  | # from random import randint |
|  | # from re import findall |
|  | from skimage import io |
|  | from skimage.util import view\_as\_blocks |
|  | from scipy.fftpack import dct, idct |
|  |  |
|  | u1, v1 = 4, 5 |
|  | u2, v2 = 5, 4 |
|  | n = 8 |
|  | P = 25 |
|  |  |
|  |  |
|  | def nsplit(s, n): # Split a list into sublists of size "n" |
|  | return [s[k:k + n] for k in range(0, len(s), n)] |
|  |  |
|  |  |
|  | def binvalue(val, bitsize): # Return the binary value as a string of the given size |
|  | binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:] |
|  | if len(binval) > bitsize: |
|  | raise Exception("binary value larger than the expected size") |
|  | while len(binval) < bitsize: |
|  | binval = "0" + binval # Add as many 0 as needed to get the wanted size |
|  | return binval |
|  |  |
|  |  |
|  | def string\_to\_bit\_array(raw\_text): # Convert a string into a list of bits |
|  | array = list() |
|  | for char in raw\_text: |
|  | binval = binvalue(char, 8) # Get the char value on one byte |
|  | array.extend([int(x) for x in list(binval)]) # Add the bits to the final list |
|  | return array |
|  |  |
|  |  |
|  | def bit\_array\_to\_string(array): # Recreate the string from the bit array |
|  | res = ''.join([chr(int(y, 2)) for y in [''.join([str(x) for x in \_bytes]) for \_bytes in nsplit(array, 8)]]) |
|  | return res |
|  |  |
|  |  |
|  | def double\_to\_byte(arr): |
|  | return np.uint8(np.round(np.clip(arr, 0, 255), 0)) |
|  |  |
|  |  |
|  | def increment\_abs(x): |
|  | return x + 1 if x >= 0 else x - 1 |
|  |  |
|  |  |
|  | def decrement\_abs(x): |
|  | if np.abs(x) <= 1: |
|  | return 0 |
|  | else: |
|  | return x - 1 if x >= 0 else x + 1 |
|  |  |
|  |  |
|  | def abs\_diff\_coefs(transform): |
|  | return abs(transform[u1, v1]) - abs(transform[u2, v2]) |
|  |  |
|  |  |
|  | def valid\_coefficients(transform, bit, threshold): |
|  | difference = abs\_diff\_coefs(transform) |
|  | if (bit == 0) and (difference > threshold): |
|  | return True |
|  | elif (bit == 1) and (difference < -threshold): |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | def change\_coefficients(transform, bit): |
|  | coefs = transform.copy() |
|  | if bit == 0: |
|  | coefs[u1, v1] = increment\_abs(coefs[u1, v1]) |
|  | coefs[u2, v2] = decrement\_abs(coefs[u2, v2]) |
|  | elif bit == 1: |
|  | coefs[u1, v1] = decrement\_abs(coefs[u1, v1]) |
|  | coefs[u2, v2] = increment\_abs(coefs[u2, v2]) |
|  | return coefs |
|  |  |
|  |  |
|  | def embed\_bit(block, bit): |
|  | patch = block.copy() |
|  | coefficients = dct(dct(patch, axis=0), axis=1) |
|  |  |
|  | while not valid\_coefficients(coefficients, bit, P) or (bit != retrieve\_bit(patch)): |
|  | coefficients = change\_coefficients(coefficients, bit) |
|  | patch = double\_to\_byte(idct(idct(coefficients, axis=0), axis=1) / (2 \* n) \*\* 2) |
|  | return patch |
|  |  |
|  |  |
|  | def embed\_message(original, msg): |
|  | changed = original.copy() |
|  | img = changed[:, :, 2] |
|  | width, height = np.shape(img) |
|  | width -= width % n |
|  | height -= height % n |
|  | img = img[:width, :height] |
|  |  |
|  | blocks = view\_as\_blocks(img, block\_shape=(n, n)) |
|  | h = blocks.shape[1] |
|  | for index, bit in enumerate(msg): |
|  | i = index // h |
|  | j = index % h |
|  | block = blocks[i, j] |
|  | img[i \* n: (i + 1) \* n, j \* n: (j + 1) \* n] = embed\_bit(block, bit) |
|  | changed[:width, :height, 2] = img |
|  | return changed |
|  |  |
|  |  |
|  | def retrieve\_bit(block): |
|  | transform = dct(dct(block, axis=0), axis=1) |
|  | return 0 if abs\_diff\_coefs(transform) > 0 else 1 |
|  |  |
|  |  |
|  | def retrieve\_message(img, length): |
|  | img = img[:, :, 2] |
|  | width, height = np.shape(img) |
|  | width -= width % n |
|  | height -= height % n |
|  | img = img[:width, :height] |
|  | blocks = view\_as\_blocks(img, block\_shape=(n, n)) |
|  | h = blocks.shape[1] |
|  | return [retrieve\_bit(blocks[index // h, index % h]) for index in range(length)] |
|  |  |
|  |  |
|  | def steganography\_encrypt(data\_in, image\_in, image\_out): |
|  | image\_in = io.imread(image\_in) |
|  | data\_in = string\_to\_bit\_array(data\_in) |
|  | data\_in\_len = len(data\_in) |
|  | image\_out = embed\_message(image\_in, data\_in) |
|  | return image\_out, data\_in\_len |
|  |  |
|  |  |
|  | def steganography\_decrypt(text\_length, image\_out): |
|  | image\_out = io.imread(image\_out) |
|  | out\_message = retrieve\_message(image\_out, text\_length) |
|  | out\_message = bit\_array\_to\_string(out\_message) |
|  | return out\_message |
|  |  |
|  |  |
|  | def main(): |
|  | data\_in = 'input.txt' |
|  | data\_out = 'output.txt' |
|  | image\_in = 'input.jpg' |
|  | image\_out = 'output.jpg' |
|  |  |
|  | with open(data\_in, 'r+') as f: |
|  | input\_text = f.read() |
|  | length\_input\_text = len(input\_text) |
|  |  |
|  | print('Input data: ', input\_text) |
|  | image\_out\_file, data\_in\_len = steganography\_encrypt(input\_text, image\_in, image\_out) |
|  | io.imsave(image\_out, image\_out\_file) |
|  |  |
|  | decoded\_data = steganography\_decrypt(data\_in\_len, image\_out) |
|  | print(decoded\_data) |
|  |  |
|  | with open(data\_out, 'w+') as f: |
|  | f.write(decoded\_data) |
|  |  |
|  | print('Output data: ', decoded\_data) |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |

## Приложение 2. Скриншот работы программы

