Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по лабораторной работе №8

**Стеганографические методы**

|  |  |
| --- | --- |
| Выполнил:  студент группы 653501  Ковалевская Алина | Проверил:  Артемьев В.С. |

Минск, 2019

**Задача**

1)Изучить теоретические сведения.

2)Создать программу, которая может реализовать сокрытия (извлечения) текстового сообщения в (из) JPEG изображение(я) на основе метода в частотной области изображения.

**Введение**

Стеганогра́фия (от [греч.](https://ru.wikipedia.org/wiki/Греческий_язык) στεγανός «скрытый»+ γράφω «пишу»; букв.«тайнопись») — 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.

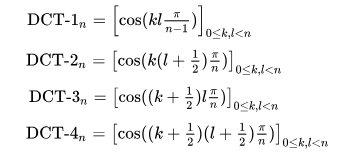
По способу встраивания информации стегоалгоритмы можно разделить на линейные (аддитивные), нелинейные и другие. Алгоритмы аддитивного внедрения информации заключаются в линейной модификации исходного изображения, а её извлечение в декодере производится корреляционными методами. При этом ЦВЗ обычно складывается 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 (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:

****

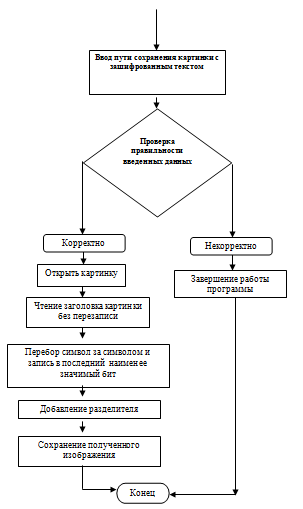
Именно DCT2 чаще всего встречается в практических приложениях благодаря свойству «уплотнения энергии».

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.

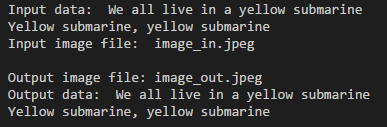
Существуют аналоги дискретного косинусного преобразования, приближающие косинус числами, легко получающимися путём небольшого количества операций сдвига и сложения, что позволяет избежать операций умножения и тем самым повысить скорость вычислений.

**Блок-схема алгоритма**



Блок-схема алгоритма зашифрования текстового сообщения в изображение

**Демонстрация работы программы**



**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.

**Программный код**

**lab8.py**

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

# Functions to change the DCT coefficients

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

# Inserting a message into an image

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)

# print(coefficients[u1, v1], coefficients[u2, v2])

patch = double\_to\_byte(idct(idct(coefficients, axis=0), axis=1) / (2 \* n) \*\* 2)

return patch

def embed\_message(original, msg):

changed = original.copy()

blue = changed[:, :, 2]

width, height = np.shape(blue)

width -= width % n

height -= height % n

blue = blue[:width, :height]

new\_width, new\_height = np.shape(blue)

blocks = view\_as\_blocks(blue, block\_shape=(n, n))

h = blocks.shape[1]

for index, bit in enumerate(msg):

# print('index=%d, bit=%d' % (index, bit))

i = index // h

j = index % h

block = blocks[i, j]

blue[i \* n: (i + 1) \* n, j \* n: (j + 1) \* n] = embed\_bit(block, bit)

changed[:width, :height, 2] = blue

return changed

# Extracting the hidden message

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)

# image\_in = io.imread('https://i.stack.imgur.com/TUV0V.png')

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():

# if len(sys.argv) < 5:

if len(sys.argv) < 4:

exit(1)

data\_in = sys.argv[1]

data\_out = sys.argv[2]

image\_in = sys.argv[3]

image\_out = sys.argv[4]

# keys\_file = sys.argv[5]

# part 1 - encoding

with open(data\_in, 'r+') as f:

input\_text = f.read()

length\_input\_text = len(input\_text)

print('Input data: ', input\_text)

print('Input image file: ', image\_in)

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)

with open(data\_out, 'w+') as f:

f.write(decoded\_data)

print('\nOutput image file:', image\_out)

print('Output data: ', decoded\_data)

if \_\_name\_\_ == '\_\_main\_\_':

main()