МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ

Основы стеганографии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Сокрытие информации в коэффициентах дискретного косинусного преобразования методом LSB»

Выполнил студент
группы N3349
До Бао Хоанг

Проверил: ассистент ФБИТ,
Университет ИТМО,
Давыдов Вадим Валерьевич

Санкт-Петербург
2020

Цель работы

Встраивание текстовой информации в картинки, используя метода LSB-DCT.

Извлечение тектовой информации из стегоконтейнера.

Построение графика значений PSNR исходного изображения и изображения с информацией, встроенные по наростающей одно слово, пять, десять.

Оценка целесообразности метода с реальными примерами.

Теория

Стеганография – способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи (хранения). Цифровая стеганография — направление классической стеганографии, основанное на сокрытии или внедрении дополнительной информации в цифровые объекты, вызывая при этом некоторые искажения этих объектов. В данной лабораторной работе был рассмотрен метод НЗБ с изображенным объектом.

НЗБ (наименьший значимый бит) - суть этого метода заключается в замене последних значащих битов в контейнере (изображения) на биты скрываемого сообщения. Разница между пустым и заполненным контейнерами должна быть не ощутима для органов восприятия человека.

Дискретное косинусное преобразование (англ. Discrete Cosine Transform, DCT) — одно из ортогональных преобразований. Вариант косинусного преобразования для вектора действительных чисел. Это преобразование тесно связано с дискретным преобразованием Фурье и является гомоморфизмом его векторного пространства. Математически преобразование можно осуществить умножением вектора на матрицу преобразования. Различные периодические продолжения сигнала ведут к различным типам ДКП. Ниже приводятся матрицы для первых четырёх типов ДКП:

$$\text{DCT-1}_n = \left[\cos(kl\tfrac{\pi}{n-1})\right]_{0\le k,l<n}$$

$$\text{DCT-2}_n = \left[\cos(k(l+\tfrac{1}{2})\tfrac{\pi}{n})\right]_{0\le k,l<n}$$

$$\text{DCT-3}_n = \left[\cos((k+\tfrac{1}{2})l\tfrac{\pi}{n})\right]_{0\le k,l<n}$$

$$\text{DCT-4}_n = \left[\cos((k+\tfrac{1}{2})(l+\tfrac{1}{2})\tfrac{\pi}{n})\right]_{0\le k,l<n}$$

В этом работе, метод замены наименьший значащего бита и 2D DCT-2_n выполнен с картинками в формате .bmp. BMP – это формат картинок, с помощью которого сохраняются только растровые изображения, а векторные нет. С английского языка слово переводится, как «Bitmap Picture» или BMP, что значит формат для хранения растровых изображений. Формула дискретного косинусного преобразования:

$$DCT[i,j] = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} IDCT[x,y] cos\left[\frac{(2x+1)i\pi}{2N}\right] cos\left[\frac{(2y+1)i\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, x = 0 \\ 1, x > 0 \end{cases}$$

Формула обратного дискретного косинусного преобразования

$$IDCT[i,j] = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)DCT[i,j] cos\left[\frac{(2x+1)i\pi}{2N}\right] cos\left[\frac{(2y+1)i\pi}{2N}\right]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, x = 0 \\ 1, x > 0 \end{cases}$$

(https://math.spbu.ru/SD_AIS/documents/2013-12-341/2013-12-tw-14.pdf).

Для того, чтобы встраивать исходное сообщение, оно преобразуется в двоичный код. Далее двоичная последовательность записывается в НЗБ нулевых коэффициентов DCT исходного изображения. С помощью функции IDCT вычислить значение писели стегоконтейнера.

Для оценки качества изображения RGB (три канала в одной пиксели) используют либо меру среднеквадратического искажения, определяемую как:

$$MSE = \frac{1}{mnt} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{k=0}^{t-1} [I(i,j,k) - K(i,j,k)]^2$$

Где: m,n – размер изображения.

k – количество каналов в одной писели.

I, K – значение каналов пикселей исходного и выходного изображения.

Пиковое отношение сигнала к шуму обозначается аббревиатурой PSNR и является инженерным термином, означающим соотношение между максимумом возможного значения сигнала и мощностью шума, искажающего значения сигнала. PSNR определяется так:

$$PSNR = 10log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Где: MAX_I — это максимальное значение, принимаемое пикселем изображения. Когда пиксели имеют разрядность 8 бит, MAXI = 255 (https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%B0_%D0%BA_%D1%88%D1%83%D0%BC%D1%83).

Практика

Для написания программы использовался язык Python v.3.7.6, так как мне нравится работа со строками в данном языке. Питон медленнее C/C++, но в данном случае мы не заинтересованы в космической скорости работы алгоритма, сотые доли секунд никак не влияют на эффективность программ.

Для того, чтобы реализовать вышеописанный метод, мною была написана 3 программа: встраивания-извлечения, анализа (посчитание и построение графика PSNR) и основа (взаимодействовать с пользователями).

Программная реализация включает в себя следующие модули:

- **lab3.py** содержатся основые функции: чтение информации из командной строки, чтение и запись с файлами.
- **LSB-DCT.py** включает две функции: встраивание инфромации в изображение и извлечения из него.
- Модуль **analysis.py** написан для построения график зависимости PSNR от размера встраивания.

Исходным контейнером является изображение “container.bmp” с размером 320x256 пикселей в cover.bmp” с размером 320x256 пикселей в формате BMP (https://drive.google.com/file/d/1WIn65biOYIn_QGgseWcKNSoM_GGiTuuq/view?usp=sharing).

Рисунок 1. Изображенный контейнер

Алгоритм

Встраивание выполнено по следующим прициpам:

- использовать синие каналы писели исходного изображения
- встраивать сверху вниз и слево направо.

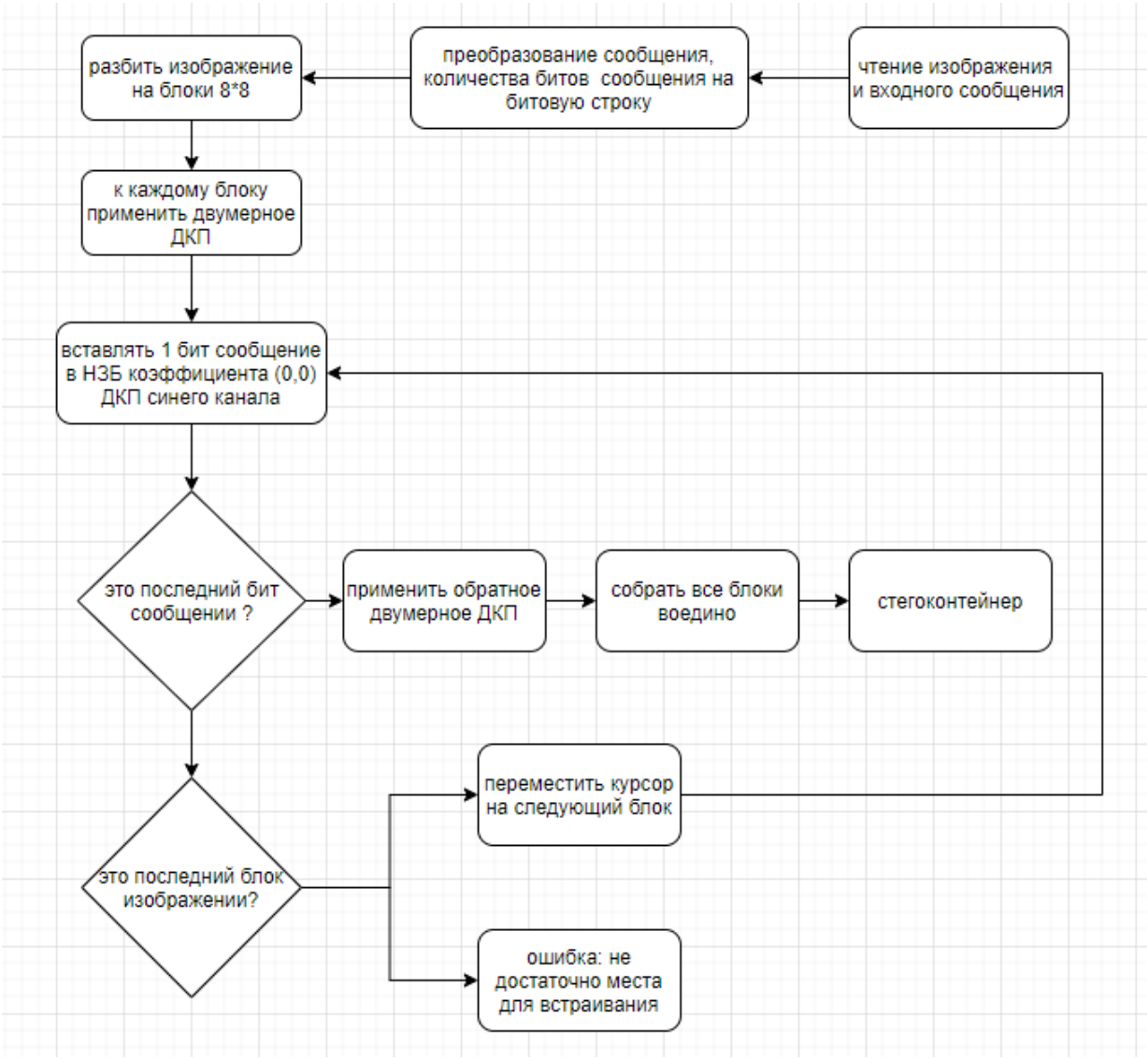

Рисунок 2. Блок-схема алгоритма встраивания

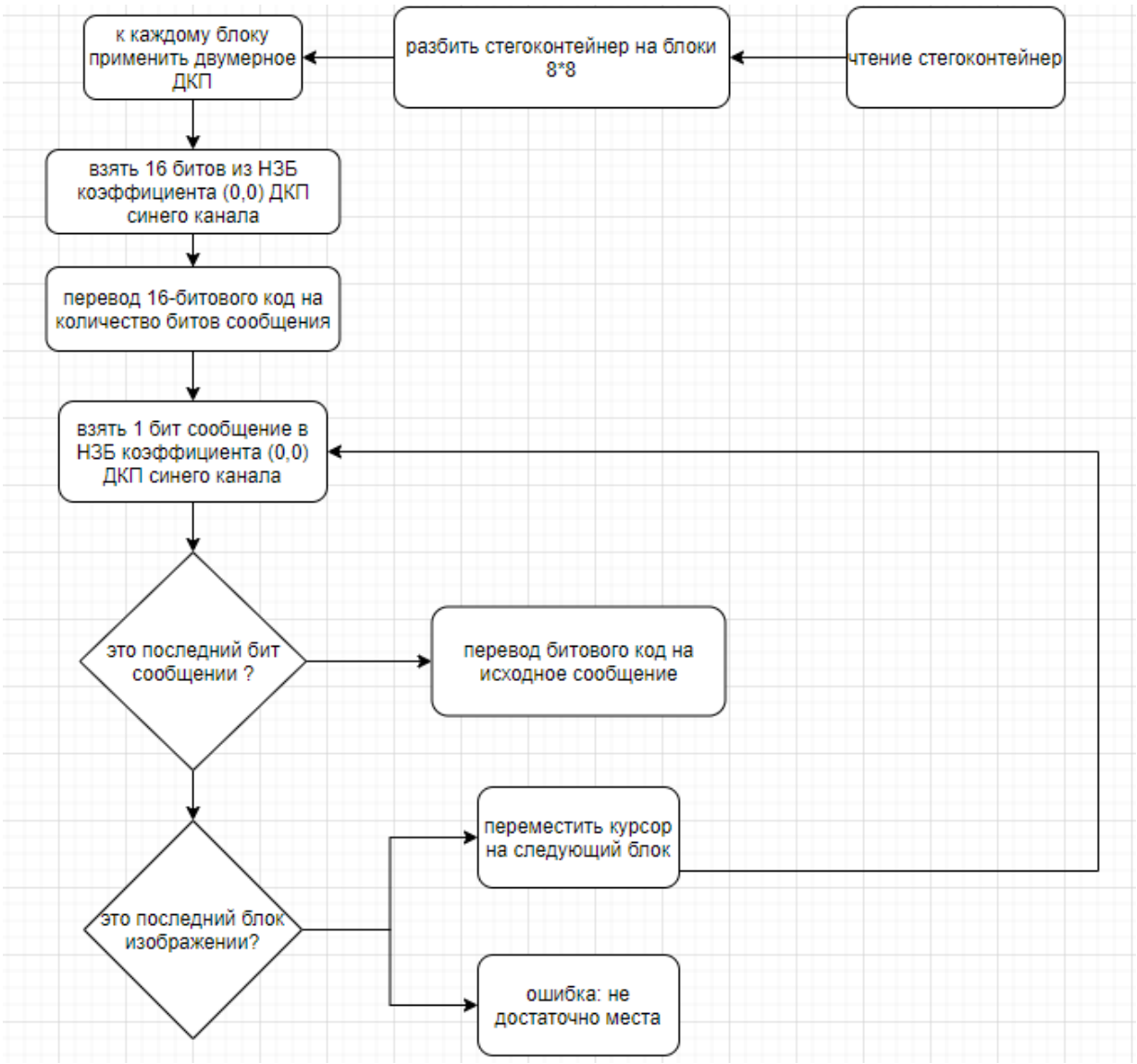

Рисунок 3. Блок-схема алгоритма извлечения

Текстовое исходное сообщение – файл in_text.txt.

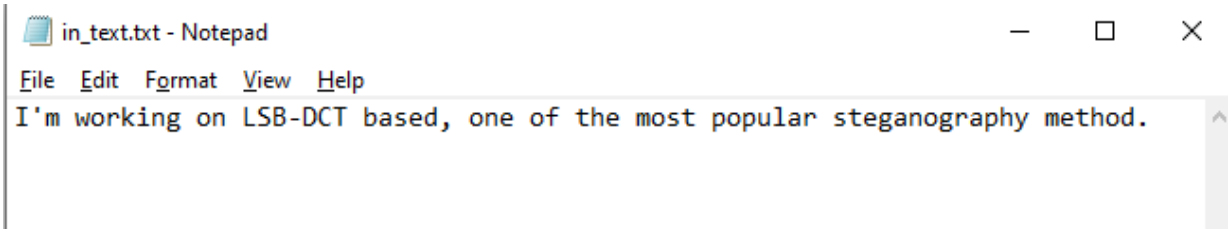

Рисунок 4. Текстовое исходное сообщение

```
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py embed -ii container.bmp -i in_text.txt -o stego.bmp
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py extract -ii stego.bmp -o out_text.txt
PS C:\Users\sau beo\Desktop\stenology\lab3>
```

Рисунок 5. Пример запуска встраивания-извлечения

Текстовое выходное сообщение - файл out_text.txt:

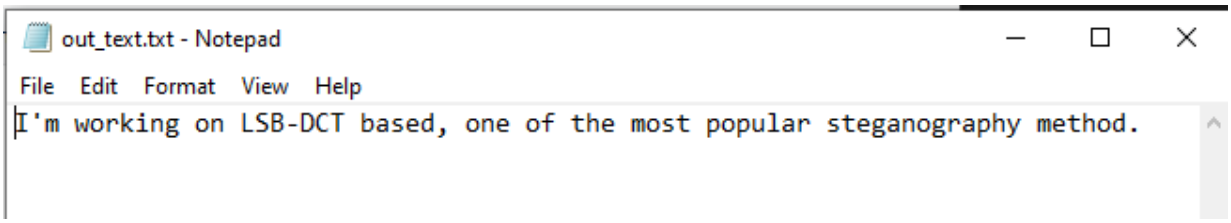

Рисунок 6. Текстовое выходное сообщение

Выходной стегоконтейнер – файл stegImage.bmp:

Рисунок 7. Выходной стегоконтейнер

Пиковое отношение сигнала к шуму:

```
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py PSNR -ii container.bmp -o stego.bmp
PSNR of your 2 images is: 43.2667144579972
```

Рисунок 8. Вычисление PSNR

Запускать программу для встраивания в другой коэффициент

Если выполнено встраивание на коэффициент (2,2) (у нас в работе встраивание на коэффициент 0,0), то получается результат на рисунке 9 и 10.

```
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py embed -ii container.bmp -i in_text.txt -o stego.bmp
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py extract -ii stego.bmp -o out_text.txt
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py PSNR -ii container.bmp -o stego.bmp
PSNR of your 2 images is: 43.340911295633724
```

Рисунок 9. Пример запуска встраивания на коэффициент (2,2)

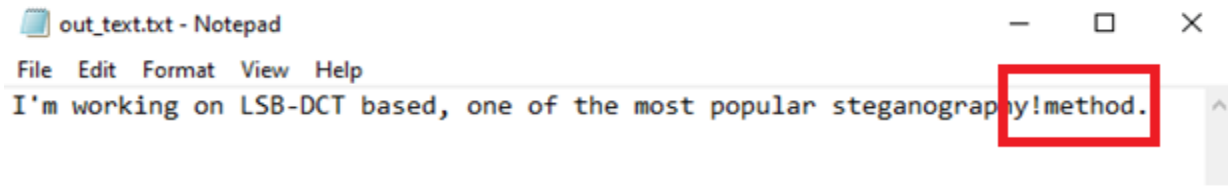

Рисунок 10. Текстовое выходное сообщение

Результаты: PSNR = 43,341 (вышее значение) – лучшее качество изображение. Потому что мы вставляем в коэффициент (2,2) а не (0,0) – коэффициент DCT, который

больше всего влияет на значение пикселей изображения. Так как после квантования значение коэффициента (2,2) некоторых блоков изменяется на 0, появились приближные проблемы когда извлечения. Следует в текстовом выходном есть ошибка (обведено красным).

Построение графика PSNR:

В этом работе, мы построим графику значений PSNR исходного изображения и изображения с информацией, встроенные по наростающей одно слово, пять, десять, .. до конца текстового файла.

Вход: файл container.bmp, текстовой файл in_text.txt.

Выход: графика PSNR исходного изображения и изображения.

```
PS C:\Users\sau beo\Desktop\stenology\lab3> py lab3.py graph -ii container.bmp -i in_text.txt
```

Рисунок 11. Пример запуска модули графика

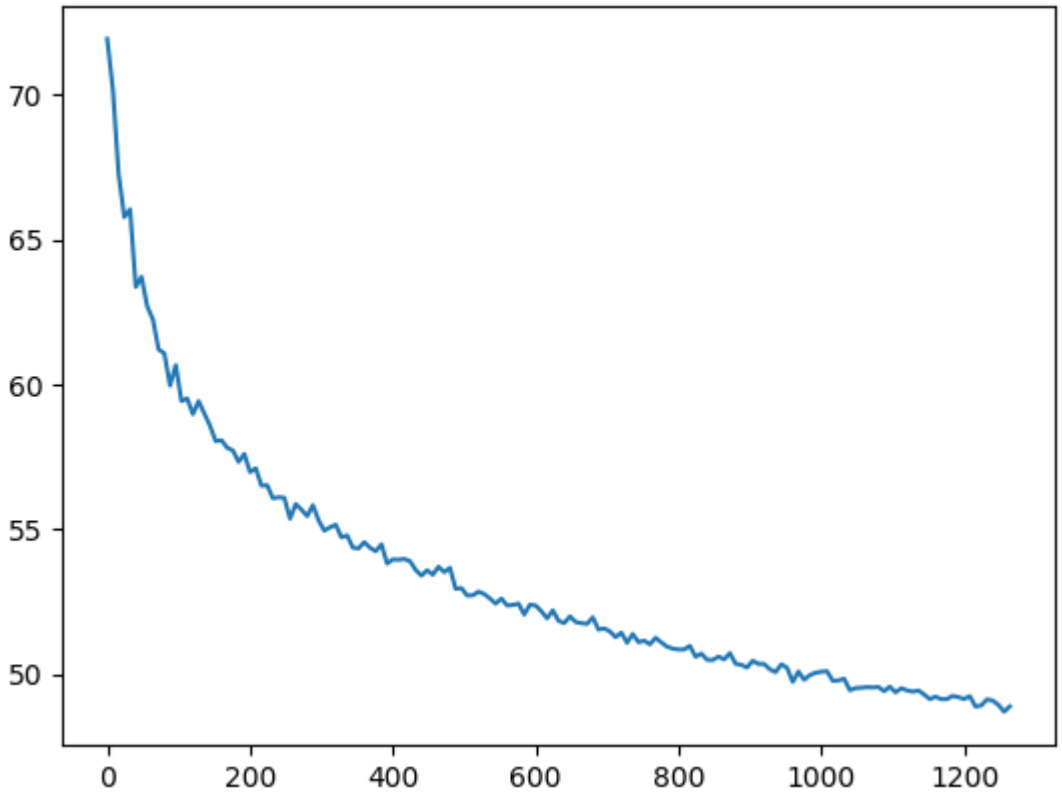

Рисунок 12. графика PSNR исходного изображения и изображения

Можно видеть, что это значение идет от бесконечного и продолжает падать, когда мы вставляем больше символа.

Оценка целесообразности

В первом случае в текст было встроено 40 битов информации, на первый взгляд размер не изменился.

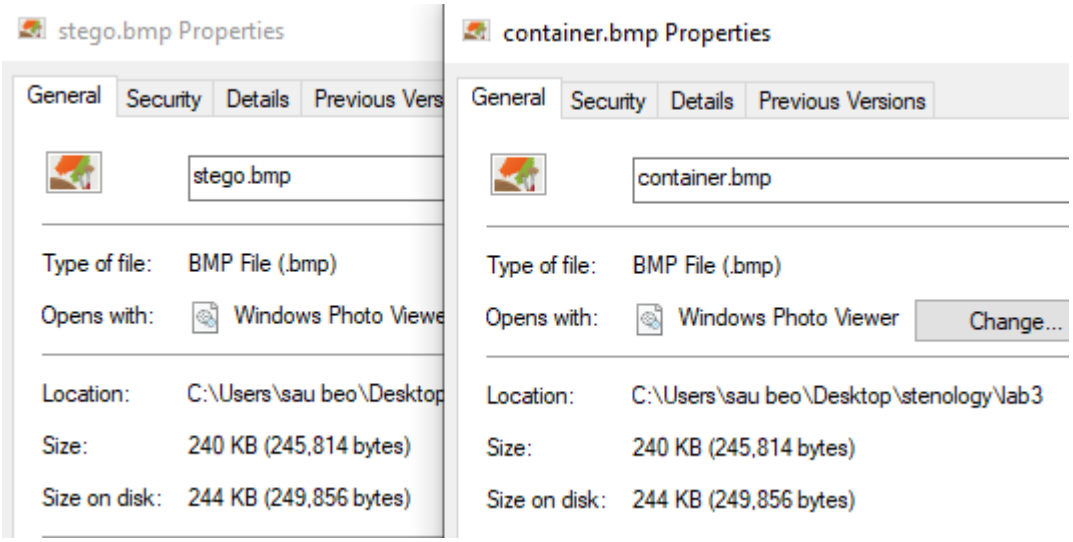

Рисунок 13. Сравнение размера файлов

Во втором случае в текст было встроено 1232 бита, размер также на первый взгляд не изменился.

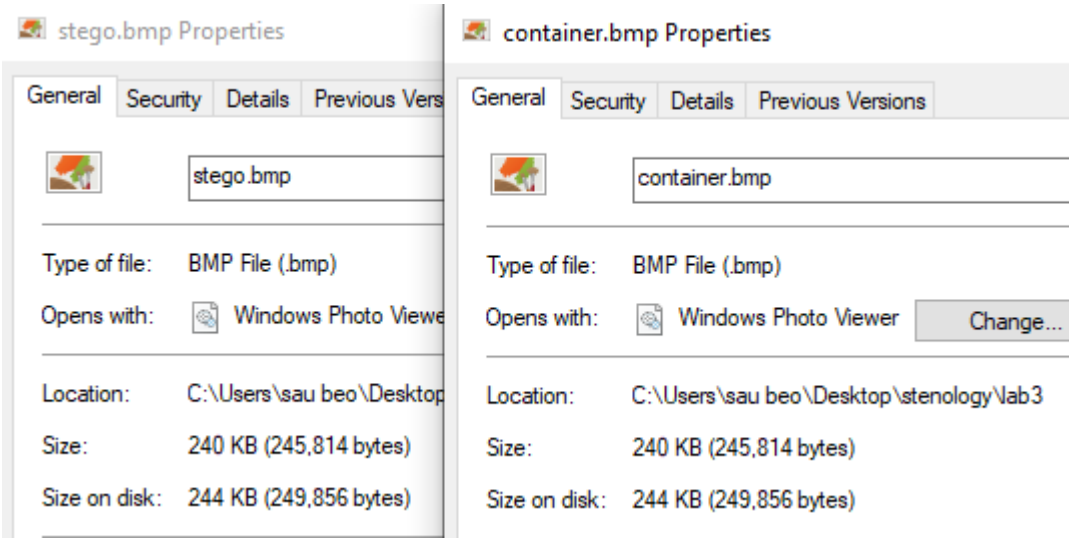

Рисунок 14. Сравнение размера файлов

Можно сделать вывод, что метод замены наименьший значащего бита целесообразен по следующим причинам:

- Можно встраивать небольшое сообщение (до 1/64 размера исходного файла).
- Размер стегоконтейнера остается таким же, как размер контейнера.

Вывод

При выполнении данной лабораторной работы мною был изучен метод LSB-DCT стеганографии. Я научился применять его и проводить последующую оценку их применению. Можно сделать вывод, что кочество изображения заданого метода только среднее и такой метод целесообразен для небольшего секретного сообщения.

Список использованной литературы

1. Стеганография [Электронный ресурс]: 2011 г. https://habr.com/ru/post/114597/ .
2. Метод сокрытия информации в квантованных коэффициентах дискретного косинус преобразования, http://ainjournal.ru/file/847577.html?__s=1 .
3. Стеганография и другие альтернативные методы сокрытия информации - 2013г https://www.bibliofond.ru/view.aspx?id=657009 .

Приложение

lab3.py

```
from LSB_DCT import *
from analysis import *
import argparse


# config command line arguments
parse = argparse.ArgumentParser(description= "DBH lab3 staganography")
parse.add_argument('-i',metavar='ifile',type=str,dest= "in_text",help= "name of input text file")
parse.add_argument('-ii',metavar='inimg',type=str,dest= "in_img",help= "name of input image file",required= True)
parse.add_argument('-o',metavar='ofile',type=str,dest= "out_file",help= "name of output image/text file")
parse.add_argument('option',metavar='opts',type=str,help= "name of option (embed/extract/PSNR/detection)")
args = parse.parse_args()


# embed text data to image with LSB-DCT method
if args.option == "embed":
    # read text to embed
    text = open(args.in_text,"r",encoding="UTF-8").read()
    # get data and run LSB-DCT stegano
    data_in = cv2.imread(args.in_img)
    steg = LSB_DCT_steg(data_in)
    data_out = steg.embed(text)
    # write image's data to file
    cv2.imwrite(args.out_file,data_out)
# extract text data from image by LSB-DCT method
elif args.option == "extract":
    # get data and run inverse LSB-DCT stegano
```

```
    data_in = cv2.imread(args.in_img)
    steg = LSB_DCT_steg(data_in)
    text = steg.extract()
    # write extracted text to file
    open(args.out_file,"w",encoding="UTF-8").write(text)
# calculate PSNR of 2 image
elif args.option == "PSNR":
    # get 2 input images's data and run PSNR
    data1 = cv2.imread(args.in_img)
    data2 = cv2.imread(args.out_file)
    analize = Analysis(data1,data2)
    PSNR = analize.PSNR()
    # show PSNR
    print("PSNR of your 2 images is:",end = " ")
    print(PSNR)
# draw PSNR graphic with image embed by 1,2,.... symbols
elif args.option == "graph":
    # read input text
    text = open(args.in_text,"r",encoding="UTF-8").read()
    # get input images's data
    data_in = cv2.imread(args.in_img)
    # data_out = cv2.imread(args.out_file)
    in_text = ""
    PSNRs = []
    numWords = []
    for i in range(len(text)):
        in_text += text[i]
        print(in_text)
        steg = LSB_DCT_steg(data_in) # init LSB-DCT method
        data_out = steg.embed(in_text)
        data_in = cv2.imread(args.in_img)
```

```
            analize = Analysis(data_in,data_out)
            # save output data to array
            numWords.append(i)
            PSNR = analize.PSNR()
            PSNRs.append(PSNR)
        # draw PSNR graphic
        import matplotlib.pyplot as plt
        plt.plot(numWords,PSNRs)
        plt.xlabel("Number of words hided in image")
        plt.ylabel("PSNR value in Db")
        plt.title("Graphic PSNR with Number of words ratio")
        plt.show()
    # run a simple attack to LSB-DCT method
    elif args.option == "detection":
        # get input images's data
        data_in = cv2.imread(args.in_img)
        steg = LSB_DCT_steg(data_in)
        steg.statistic()
```

LSB_DCT.py

```
from cv2 import cv2
import sys,numpy as np,itertools

class StegaException(Exception):
    pass

class LSB_DCT_steg():
    def __init__(self,img):
        self.image = img
        self.orirow,self.oricol,self.chan  = img.shape
        # Padding some pixel to make row and col divisible by 8
        if self.orirow % 8 != 0 or self.oricol %8 != 0:
```

```
        newRow = self.orirow+8-self.orirow%8
        newCol = self.oricol+8-self.oricol%8
        self.image = cv2.resize(self.image,(newCol,newRow))
    self.row,self.col,_ = self.image.shape
    # break image to 8*8 blocks
    self.blocks = self.break8()
    self.quant = np.array([[16,11,10,16,24,40,51,61],
            [12,12,14,19,26,58,60,55],
            [14,13,16,24,40,57,69,56],
            [14,17,22,29,51,87,80,62],
            [18,22,37,56,68,109,103,77],
            [24,35,55,64,81,104,113,92],
            [49,64,78,87,103,121,120,101],
            [72,92,95,98,112,100,103,99]])
# break image to 8*8 blocks
def break8(self):
    bImage,_,_ = cv2.split(self.image)
    bImage = np.float32(bImage)
    imgBlocks = [bImage[j:j+8, i:i+8]
            for (j,i) in itertools.product(range(0,self.row,8),range(0,self.col,8))]
    return imgBlocks

# return all groups, in which sum of all width = self.col
def chunkRows(self,blocks,n):
    m = int(n)
    for i in range(0, len(blocks), m):
        yield blocks[i:i + m]

def reshape(self,blocks):
    _,gImage,rImage = cv2.split(self.image)
    bImage = []
```

```
    for chunk in self.chunkRows(blocks,self.col/8):
        for numRow in range(8):
            for block in chunk:
                bImage.extend(block[numRow])
    bImage = np.array(bImage).reshape(self.row,self.col)
    bImage = np.uint8(bImage)
    img = cv2.merge((bImage,gImage,rImage))
    return img

# change value into binary form with fixed bitsize
def binary_value(self,value,bitsize):
    binVal = bin(value)[2:]
    if len(binVal) > bitsize:
        raise StegaException("Your message contain unexpected character!!!")
    while len(binVal) < bitsize:
        binVal = '0' + binVal
    return binVal

def binary_text(self,mess):
    res = ""
    l = len(mess)
    res += self.binary_value(l,16)
    for ch in mess:
        c = ord(ch)
        res += self.binary_value(c,8)
    return res

def embed(self,mess):
    # calculate DCT coefficients blocks from image
    DCT_blocks = [np.round(cv2.dct(block)) for block in self.blocks]
```

```
    # divide each coefficient by quantization value
    quantizedDCT = [np.round(DCT_block/self.quant) for DCT_block in DCT_blocks]
    # quantizedDCT = [np.round(DCT_block/1) for DCT_block in DCT_blocks]
    # translate message to binary form
    mess = self.binary_text(mess)
    messIndex = 0
    # embed each bit of message to the 1st coefficient of each block
    for block in quantizedDCT:
        DC = int(block[2,2])
        if DC % 2 == 0:
            DC ^= int(mess[messIndex])
        else:
            DC ^= int(mess[messIndex]) ^ 1
        block[2,2] = np.float32(DC)
        messIndex += 1
        if messIndex == len(mess): break
    if messIndex < len(mess)-1: raise StegaException("not enough spaces to embed")
    # multiply each coefficient with quantization value
    DCT_blocks = [block * self.quant for block in quantizedDCT]
    # DCT_blocks = [block * 1 for block in quantizedDCT]
    # calculate inverse DCT to get image's values
    Img_blocks = [np.round(cv2.idct(block)) for block in DCT_blocks]
    # reshape blocks to image form
    img = self.reshape(Img_blocks)
    # img = cv2.resize(img,(self.oricol,self.orirow))
    return img


# read all embed bits in blocks
def read_bits(self,blocks):
    res = ""
```

```
    for block in blocks:
        c = int(block[2,2])
        if c % 2 == 0: res += '0'
        else: res += '1'
    return res

def extract(self):
    # calculate DCT coefficients blocks from image
    DCT_blocks = [np.round(cv2.dct(block)) for block in self.blocks]
    # divide each coefficient by quantization value
    quantizedDCT = [np.round(DCT_block/self.quant) for DCT_block in DCT_blocks]
    # quantizedDCT = [np.round(DCT_block/1) for DCT_block in DCT_blocks]

    length = self.read_bits(quantizedDCT[:16])
    length = int(length,2)
    cursor = 16
    mess = ""
    for _ in range(length):
        c = self.read_bits(quantizedDCT[cursor:cursor+8])
        c = int(c,2)
        ch = chr(c)
        mess += ch
        cursor += 8
        if cursor >= len(quantizedDCT):
            raise StegaException("Your image hasn't been embed!!")
    return mess

def statistic(self):
    # calculate DCT coefficients blocks from image
    DCT_blocks = [np.round(cv2.dct(block)) for block in self.blocks]
    # divide each coefficient by quantization value
```

```
        quantizedDCT = [np.round(DCT_block/self.quant) for DCT_block in DCT_blocks]
        import matplotlib.pyplot as plt
        histogram = []
        for block in quantizedDCT:
            for i in range(8):
                for j in range(8):
                    val = float(block[i,j])
                    histogram.append(val)
        plt.hist(histogram,bins = 100)
        plt.show()
```

analysis.py

```
from cv2 import cv2
import numpy as np
class PSNRException(Exception):
    pass
class Analysis():
    def __init__(self,originImg,Img):
        self.origin = originImg
        self.changed = Img

        orirow,oricol = originImg.shape[:2]
        # Padding some pixel to make row and col divisible by 8
        if orirow % 8 != 0 or oricol %8 != 0:
            newRow = orirow+8-orirow%8
            newCol = oricol+8-oricol%8
            self.origin = cv2.resize(originImg,(newCol,newRow))

        self.oheight, self.owidth, self.ochannel = self.origin.shape
        self.osize = self.oheight * self.owidth * self.ochannel

        self.cheight, self.cwidth, self.cchannel = Img.shape
```

```
        self.csize = self.cheight * self.cwidth * self.ochannel

        self.MAXi = 255 # maximum posible channel value of the image = 2^8-1
        self.maskONE = 0b00000001 # use bitwise OR to make LSB 1

    def MSE(self): # calculate MSE = 1/size * sum((I(i,j)-K(i,j))^2) with i = 1..n;j = 1..m
        mse = np.mean((self.origin - self.changed)**2)
        return mse
    def PSNR(self): # calculate PSNR = 10*log10(MAXi^2/MSE)
        mse = np.mean((self.origin - self.changed)**2)
        res = self.MAXi**2/mse
        res = 10 * np.log(res) / np.log(10)
        return res

class AnalysisException(Exception):
    pass
```