

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №8  
“Стеганографические методы”

Выполнил:

студент группы 053504

Горожанкин В.О.

Проверил

ассистент кафедры информатики

Лещенко Евгений Александрович

Минск 2023

## СОДЕРЖАНИЕ

Введение.....	3
1 Демонстрация работы программы.....	4
2 Теоретические сведения.....	6
Заключение.....	7
Приложение А (обязательное) Листинг программного кода.....	8

## **ВВЕДЕНИЕ**

В современном информационном обществе защита конфиденциальной информации и обеспечение её безопасности становятся всё более актуальными задачами. Стеганография, наука о скрытом передаче данных, предоставляет один из способов сокрытия информации в других данных, таких как изображения. Одним из популярных методов стеганографии является сокрытие текстового сообщения в изображениях, сохраняя визуальную незаметность изменений и обеспечивая безопасность передачи.

Данная лабораторная работа нацелена на разработку программного средства для сокрытия и извлечения текстовых сообщений в (из) JPEG изображениях с использованием метода сокрытия в частотной области. Этот метод основан на использовании дискретного косинусного преобразования, которое позволяет представить изображение в частотной области и внести в него скрытую информацию без существенного воздействия на визуальное восприятие изображения.

# 1 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Исходное изображение представлено на рисунке 1.



Рисунок 1 – Исходное изображение

Исходный текст передаётся в файле input.txt.

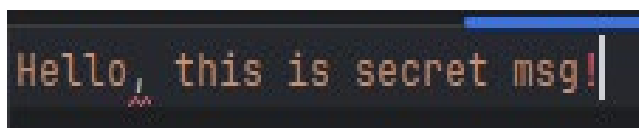


Рисунок 2 – Секретное сообщение

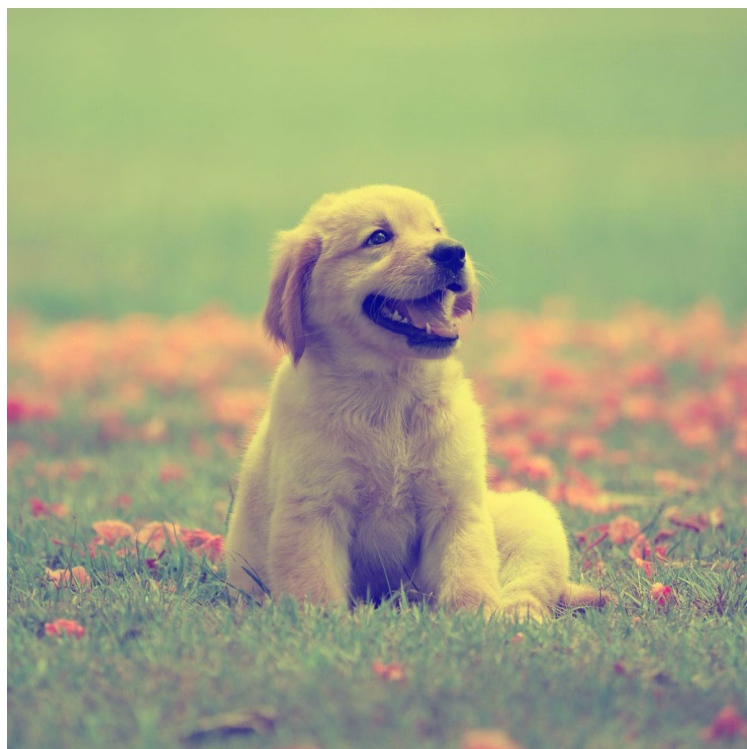


Рисунок 3 – Зашифрованное изображение

Результат дешифрования представлен на рисунке 4.

```
/home/katrina/cab3/m21/cab0/venv/bin/python /home,  
Hiding msg Hello, this is secret msg!  
Decoded Message: Hello, this is secret msg!
```

Рисунок 4 – Результат дешифрования

## 2 ОПИСАНИЕ БЛОК-СХЕМЫ АЛГОРИТМА

Блок-схема алгоритма представлена на рисунке 5. В данном способе изображение разделяется на блоки 8x8 пикселей, каждый из которых используется для шифрования одного бита сообщения. Шифрование начинается с произвольного подбора блока, используемого для шифрования  $i$ -го бита сообщения. Затем для подобранного блока применяют ДКП.

LSB (Least Significant Bit, наименьший значащий бит (НЗБ)) — суть этого метода заключается в замене последних значащих битов в контейнере (изображения, аудио или видеозаписи) на биты скрываемого сообщения. Разница между пустым и заполненным контейнерами должна быть не ощутима для органов восприятия человека.

Скрытие: Исходное изображение-контейнер разделяется на блоки по 8x8 пикселей, к которым применяется DCT.  $F_{\{i\}}[a,b]=DCT(O_{\{i\}}[a,b])$ . Из каждого коэффициента матрицы  $F_{\{i\}}$  выделяются наименее значимые биты и заменяются на биты скрываемого сообщения[.

Извлечение: Изображение-контейнер разделяется на блоки  $O_{\{i\}}$  по 8x8 пикселей, к которым применяется DCT:  $F_{\{i\}}[a,b]=DCT(O_{\{i\}}[a,b])$ . Из каждого коэффициента матрицы  $F_{\{i\}}$  выделяются наименее значимые биты и объединяются, восстанавливая скрытое сообщение.

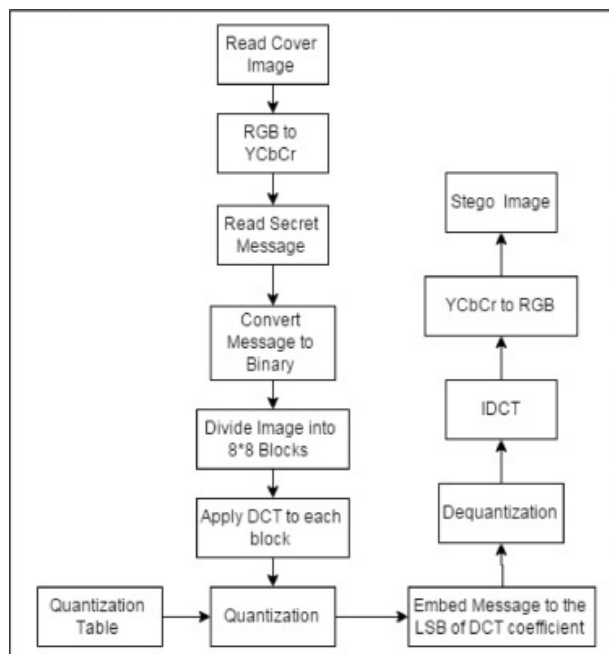


Рисунок 3 – Блок-схема работы алгоритма.

## **ЗАКЛЮЧЕНИЕ**

В рамках данной лабораторной работы было разработано программное средство для сокрытия и извлечения текстовых сообщений в (из) JPEG изображениях на основе метода сокрытия в частотной области с использованием дискретного косинусного преобразования.

Таким образом, разработанное программное средство предоставляет эффективный инструмент для скрытой передачи данных в изображениях с использованием метода ДКП, и может быть полезным в сферах, где обеспечение безопасности и конфиденциальности информации имеет высший приоритет.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Листинг программного кода

```
from __future__ import print_function
import warnings
import cv2
import itertools
import numpy as np

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]])

class DCT:
    def __init__(self, imPath):
        self.imPath = imPath
        self.message = None
        self.bitMess = None
        self.oriCol = 0
        self.oriRow = 0
        self.numBits = 0
        """Input: secret - secret message to be hidden
           outIm - name of the image you want to be output
           Function: takes message to be hidden and preforms dct stegonography to
           hide the image within the least
           significant bits of the DC coefficents.
           Output: writes out an image with the encoded message"""
    def DCTEn(self, secret, outIm):
        # load image for processing
        img = self.loadImage()
        if img is None:
            print("Error: File not found!")
            return

        self.message = str(len(secret)) + '*' + secret
        self.bitMess = self.toBits()

        # get size of image in pixels
        row, col = img.shape[:2]
        self.oriRow, self.oriCol = row, col

        if ((col / 8) * (row / 8) < len(secret)):
            print("Error: Message too large to encode in image")
            return

        # make divisible by 8x8
        if row % 8 != 0 or col % 8 != 0:
```



```

        img = self.addPadd(img, row, col)

    row, col = img.shape[:2]

    # split image into RGB channels
    bImg, gImg, rImg = cv2.split(img)

    # message to be hid in blue channel so converted to type float32 for
    dct function
    bImg = np.float32(bImg)
    # print(bImg[0:8,0:8])

    # break into 8x8 blocks
    imgBlocks = [np.round(bImg[j:j + 8, i:i + 8] - 128) for (j, i) in
                  itertools.product(range(0, row, 8), range(0, col, 8))]

    # Blocks are run through DCT function
    dctBlocks = [np.round(cv2.dct(img_Block)) for img_Block in imgBlocks]

    # blocks then run through quantization table
    quantizedDCT = [np.round(dct_Block / quant) for dct_Block in
                    dctBlocks]
    # set LSB in DC value corresponding bit of message
    messIndex = 0
    letterIndex = 0

    for quantizedBlock in quantizedDCT:
        # find LSB in DC coeff and replace with message bit
        DC = quantizedBlock[0][0]
        DC = np.uint8(DC)
        DC = np.unpackbits(DC)
        DC[7] = self.bitMess[messIndex][letterIndex]
        DC = np.packbits(DC)

        DC = np.float32(DC)
        DC = DC - 255
        with warnings.catch_warnings():
            warnings.filterwarnings("ignore",
category=DeprecationWarning)
            quantizedBlock[0][0] = DC
            letterIndex = letterIndex + 1
            if letterIndex == 8:
                letterIndex = 0
                messIndex = messIndex + 1
                if messIndex == len(self.message):
                    break

    # blocks run inversely through quantization table
    sImgBlocks = [quantizedBlock * quant + 128 for quantizedBlock in
                  quantizedDCT]

    # puts the new image back together
    sImg = []
    for chunkRowBlocks in self.chunks(sImgBlocks, col / 8):

```

```

        for rowBlockNum in range(8):
            for block in chunkRowBlocks:
                sImg.extend(block[rowBlockNum])
sImg = np.array(sImg).reshape(row, col)

# converted from type float32
sImg = np.uint8(sImg)

sImg = cv2.merge((sImg, gImg, rImg))
cv2.imwrite(outIm, sImg)
return sImg
"""Input: no input needed for function
Function: takes an image with a hidden dct encoded message and extracts
the message into plaintext
Output: returns the plaintext string of the hidden message found"""
def DCTDe(self):
    img = cv2.imread(self.imPath, cv2.IMREAD_UNCHANGED)
    row, col = img.shape[:2]
    messSize = None
    messageBits = []
    buff = 0

    # split image into RGB channels
    bImg, gImg, rImg = cv2.split(img)
    # print(bImg[0:8,0:8])
    # message hid in blue channel so converted to type float32 for dct
function
    bImg = np.float32(bImg)
    # print(bImg[0:8,0:8])

    # break into 8x8 blocks
    imgBlocks = [bImg[j:j + 8, i:i + 8] - 128 for (j, i) in
itertools.product(range(0, row, 8),
range(0, col, 8))]
    # blocks run through quantization table
    quantizedDCT = [img_Block / quant for img_Block in imgBlocks]
    i = 0
    # message extracted from LSB of DC coeff
    for quantizedBlock in quantizedDCT:
        DC = quantizedBlock[0][0]
        DC = np.uint8(DC)
        DC = np.unpackbits(DC)
        if DC[7] == 1:
            buff += (0 & 1) << (7 - i)
        elif DC[7] == 0:
            buff += (1 & 1) << (7 - i)
        i = 1 + i
        if i == 8:
            messageBits.append(chr(buff))
            buff = 0
            i = 0

        if messageBits[-1] == '*' and messSize is None:
            try:

```

```

        messSize = int(''.join(messageBits[:-1]))
    except:
        pass
    if len(messageBits) - len(str(messSize)) - 1 == messSize:
        return ''.join(messageBits)[len(str(messSize)) + 1:]
    return ''
"""Helper function to 'stitch' new image back together"""

def chunks(self, l, n):
    m = int(n)
    for i in range(0, len(l), m):
        yield l[i:i + m]
"""Input: no input
Function: loads image into memory
Output: returns the memory where the image is"""

def loadImage(self):
    # load image
    img = cv2.imread(self.imPath, cv2.IMREAD_UNCHANGED)

    if img is None:
        return None

    return img

"""Input: img-the image to be padded
row-the number of rows of pixels in the image
col-the number of columns in the image
Function: add 'Padding' making image dividable by 8x8 blocks
Output: returns the new padded image"""

def addPadd(self, img, row, col):
    img = cv2.resize(img, (col + (8 - col % 8), row + (8 - row % 8)))
    return img
"""
Function: transforms the message that is wanted to be hidden
from plaintext to a list of bits
"""
def toBits(self):
    bits = []

    for char in self.message:
        binval = bin(ord(char))[2:].rjust(8, '0')

        bits.append(binval)
    self.numBits = bin(len(bits))[2:].rjust(8, '0')
    return bits

DCTEncode = DCT('dog.png')
with open("input.txt", "r", encoding="utf-8") as f:
    message_to_decode = f.read()
print(f"Hiding msg {message_to_decode}")
DCTEncode.DCTEn(message_to_decode, 'stegol.png')
DCTDecode = DCT('stegol.png')
message_to_decoded = DCTDecode.DCTDe()
print('Decoded Message: ', message_to_decode)

```